A Theory of Contexts in Information Bases

Manos Theodorakis[1,2], Anastasia Analyti[1], Panos Constantopoulos[1,2], Nikos Spyratos[3]

[1] *Institute of Computer Science, FORTH, P.O.Box 1385, GR 711 10 Heraklion, Crete, Greece*
[2] *Department of Computer Science, University of Crete, Heraklion, Greece*
[3] *Universite de Paris-Sud, LRI-Bat 490, 91405 Orsay Cedex, France*
E-mail: {*etheodor* | *analyti* | *panos*}*@ics.forth.gr*, *spyratos@lri.fr*

Abstract

Although semantic data models provide expressive conceptual modeling mechanisms, they do not support context, i.e. providing controlled partial information on conceptual entities by viewing them from different viewpoints or in different situations. In this paper, we present a model for representing contexts in information bases along with a set of operations for manipulating contexts. These operations support creating, updating, combining, and comparing contexts. Our model contributes to the efficient handling of information, especially in distributed, cooperative environments, as it enables (i) representing (possibly overlapping) partitions of an information base; (ii) partial representations of objects, (iii) flexible naming (e.g. relative names, synonyms and homonyms), (iv) focusing attention, and (v) combining and comparing different partial representations. This work advances towards the development of a formal framework intended to clarify several theoretical and practical issues related to the notion of context. The use of context in a cooperative environment is illustrated through a detailed example.

1 Introduction

The notion of context is a fundamental concern in cognitive psychology, linguistics, and computer science. Context has been considered in quite a few formalizations in several areas of computer science (see [17]), such as artificial intelligence [10, 15, 9], software development [22, 8, 24, 26, 27, 12, 13], (multiple) databases [1, 6, 11, 21], machine learning [16, 31, 14], and knowledge representation [19, 30, 29]. However, these formalizations are very diverse and serve different purposes.

In the area of knowledge representation, Mylopoulos and Motschnig-Pitrik [19] proposed a general mechanism for partitioning information bases using the concept of context. They introduced a generic framework for contexts and discussed naming conventions, operations on contexts, authorization, and transaction execution. However, they impose a strict constraint on naming, namely objects (called *information units*) are assigned unique names w.r.t. a context. Because of this constraint, several naming conflicts appear in operations among contexts, which the authors resolve in non-intuitive ways. In addition, basic operations among contexts, such as *union* (called *addition*) and *intersection* (called *product*), not only do not enjoy useful properties such as commutativity, associativity, and distributivity, but also give unexpected results. In [19], the major problem of the context union and context intersection operations is that it is possible for an object in the output context to have *no* name, even though it originally had one or more names. This can happen if an object of one input context has a name in common with an object of the other input context. For example, consider two contexts c and c' which correspond to two companies, the contents of c and c' being the employees of these two companies, respectively. Assume now that an employee in the first company has the same name with another employee in the second company. Then, the union of the contexts c and c' contains these two employees, but one of them will have no name. Such counter-intuitive results might seriously hinder the applicability of this otherwise appealing framework.

In [30], Theodorakis and Constantopoulos proposed a naming mechanism based on the concept of context, in order to resolve several naming problems that arise in information bases, i.e. object names being ambiguous, excessively long, or unable to follow the changes of the environment of the object. However, this approach imposes a hierarchical structure on contexts, i.e. a context may be contained in only one other context, which is rather restrictive.

In this paper, we try to combine these two approaches and alleviate their shortcomings by introducing a more general and more complete framework for context.

In particular, like in [19], a context is treated as a special object which is associated to a set of objects and a lexicon i.e. a binding of names to these objects. However, in our model, an object is allowed to have more than one names, even in the same context. This offers more flexibility and expressiveness and can handle the naming of real world entities in a more "natural" way, as it is possible for two objects to have the same name, even in the same frame of reference. This common name assignment may occur either accidentally, or by virtue of a common characteristic of the two objects (expressed through the common name). In our model, naming conflicts that may appear during operations on contexts are resolved through a sophisticated, yet intuitive naming mechanism. Specifically, the following situations can be handled: *synonyms* (different names that have been assigned to the same object w.r.t. the same or different contexts); *homonyms* (different objects that have the same name w.r.t. the same or different contexts); and *anonymous objects* (objects with no name w.r.t. a context). An object is externally identified using *references* w.r.t. a context. These references are either the object names w.r.t. that context, or composite names that are formed by taking into account the nesting of contexts. We distinguish an important class of contexts, called *well-defined*. Every object contained in a well-defined context has a unique reference w.r.t. that context.

Our model offers a set of operations for manipulating contexts. These operations provide support for creating, updating, combining, and comparing contexts. The most involved of the operations are those for combining and comparing contexts, namely context union, context intersection, and context difference. We prove that the class of well-defined contexts enjoys a closure property: the union, intersection, or difference of two well-defined contexts yields a well-defined context. Name ambiguities are resolved by adding to the resulting context views of the objects as seen from the input contexts. Besides being used for name disambiguation, these views carry useful information, as we demonstrate in the example of Section 5. Finally, it should be mentioned that our context union and context intersection operations are commutative, associative, and distributive, with the benefits that these properties usually carry.

The paper is organized as follows: In section 2, the context construct for information bases is introduced. In section 3, the basic operations of our model are presented. Section 5 discusses in detail an example of using context in a cooperative environment. In section 6, related work is reviewed and compared to ours, while section 7 concludes the paper.

2 The notion of context

A *context* is a higher order conceptual entity that describes a group of conceptual entities from a particular standpoint [17]. The conceptual entities described can be contexts themselves, thus allowing for nesting of contexts. Conceptual entities are named with respect to a context as part of their description.

Examples of contexts are:

- *Information bases*: An information base describes a set of conceptual entities from the point of view of its designer. Certainly, the designer's viewpoint is influenced by the particular needs of the targeted users.
- *View schemas*: A view schema in an object-oriented database [23, 1, 18], or in a relational database [7, 2] describes the conceptual entities in the view according to the person that defined that view.
- *Multiversion objects*: A multiversion object refers to a set of versions of a generic object [4, 12]. Therefore, a multiversion object can be seen as a context describing these versions.
- *Configurations*: A configuration is the binding between a version of a composite object and the particular versions of its components [12]. Therefore, a configuration of a composite object can be seen as a context describing the particular versions of its components in a joint interpretation.
- *Workspaces*: A workspace refers to a virtual space in which objects are created and manipulated under the responsibility of an individual person, or a group of persons [12]. Therefore, a workspace can be seen as a context that describes these objects according to the responsibilities of the persons involved.

An information base can be seen as containing objects that represent atomic or collective real world entities, attributes, (binary) relationships, or primitive values.

Contexts are a special kind of objects that represent real world divisions or environments. We shall call all objects which are not contexts, *simple objects*. Contexts allow us to focus on the objects of interest, as well as to name each of these objects using one or more convenient names. Informally, a context is an identifiable set of objects, each object being associated to a set of names. In order to define contexts more formally we need the concept of lexicon.

Definition 2.1 Lexicon. Let O be a set of objects and $\mathcal{N}$ the set of all atomic names. A *lexicon* is a mapping of the form:

$$l : O \longrightarrow \mathcal{P}(\mathcal{N})$$

that associates each object in O to a set of names. The objects in O are called *objects of* the lexicon l, and the set O is denoted by $objs(l)$. We denote by $\mathcal{L}$ the set of all lexicons. $\diamond$

Note that an object of a lexicon may be associated to an empty set of names.

We shall often think of a lexicon l as a set of pairs of the form $(o : l(o))$. In other words, if $objs(l) = \{o_1, \ldots, o_k\}$ then we shall write $l = \{(o_1 : l(o_1)), \ldots, (o_k : l(o_k))\}$.

The following is an example of lexicon l:

$$l = \begin{cases} o_1 : Panos \\ o_2 : head \\ o_3 : Manos \\ o_4 : Nikos, Nick \end{cases}$$

Each context c is associated to a lexicon, which we shall call the *lexicon of* c, and denote it by $lex(c)$. The context c can be used to focus on the objects of the lexicon, as well as to assign relative names to these objects.

Definition 2.2 Context lexicon. A *context lexicon* is a total function of the form:

$$lex : Cxt \longrightarrow \mathcal{L}$$

which associates a context with a lexicon. We denote by Cxt the set of all contexts. $\diamond$

Let c be the context with lexicon $\{(o_1 : N_1), \ldots, (o_k : N_k)\}$. We shall use the following notation and terminology:

- The objects $o_1, \ldots, o_k$ are called the *objects* of c and their set is denoted by $objs(c)$.
- We shall say that c *contains* $o_1, \ldots, o_k$.
- The names in N_i are called the names of o_i w.r.t. c, or the *c-names* of o_i. The set N_i is denoted by $names(o_i, c)$.

The same notation and terminology is used for a lexicon as well.

$$lex(c_1) = \begin{cases} o_1 : Dr\_Constantopoulos \\ o_4 : \\ o_5 : professor \\ c_2 : InfSys \\ c_3 : DSS \end{cases} \qquad lex(c_2) = \begin{cases} o_1 : Panos \\ o_2 : head \\ o_3 : Manos \\ o_4 : Nikos, Nick \end{cases} \qquad lex(c_3) = \begin{cases} o_6 : Panos \\ o_2 : head \\ o_1 : Constantopoulos \end{cases}$$

Figure 1: Example of context lexicons.

As an example, consider a context c_1 which represents an institute (see Figure 1). Context c_1 contains five objects in its lexicon, o_1, o_4, o_5, c_2, and c_3. Object o_1 is a simple object whose c_1-name is $Dr\_Constantopoulos$, and represents a specific person. Object o_4 is a simple object as well which represents an entity that we know it exists within the context c_1 but we don't know its name yet. Object o_5 represents the notion of professor (and not a particular person who happens to be a professor). Objects c_2 and c_3 are themselves contexts whose c_1-names are $InfSys$ and DSS, respectively. Context c_2 represents the environment of the Information Systems lab and describes the objects of that lab. Context c_3 represents the environment of the Decision Support Systems lab and describes the objects of that lab. The objects contained in contexts c_2 and c_3 are as shown in Figure 1. Note that object o_1 has only one c_2-name ($Panos$), whereas object o_4 has two c_2-names ($Nikos$ and $Nick$). Also note that the same object can be contained in more than one contexts under the same or different names. For instance, object o_1 is contained in three contexts c_1, c_2, and c_3. The c_1-name of object o_1 is $Dr\_Constantopoulos$, its c_2-name is $Panos$, whereas its c_3-name is $Constantopoulos$. Note also that two different objects, o_1 and o_6, have the same name w.r.t. two different contexts (c_2 and c_3).

As we saw in the previous example, a context may contain other contexts among its objects. We shall say that a context c *recursively contains* o if either c contains o, or there is a context contained in c that recursively contains o. This is denoted by $o \in^\star c$. For instance, in Figure 1, context c_1 recursively contains object o_2, as c_1 contains c_2 and c_2 contains o_2, i.e. $o_2 \in^\star c_1$. We shall call *nested subcontext* of a context c, any context that is recursively contained in c.

We can *refer* to every object of a context c either *directly*, using its c-names, or *indirectly*, using a sequence of dot-separated names, in the case that the object is contained in a nested subcontext of c.

Definition 2.3 References of an object w.r.t. a context. Let c be a context and let o be an object recursively contained in c. The set of all *references* of o w.r.t. c, denoted by $refs(o, c)$, is defined as follows:

$$\begin{aligned} refs(o,c) &= dirRefs(o,c) \cup indirRefs(o,c) \\ dirRefs(o,c) &= \{n \mid n \in names(o,c)\} \\ indirRefs(o,c) &= \{r.n \mid \exists c' \in^\star c \wedge n \in names(o,c') \wedge r \in refs(c',c)\} \end{aligned}$$

The set of all references is denoted by $\mathcal{R}$. $\diamond$

For example (see Figure 1), we can refer to object o_1 of context c_1 either directly, using the reference $Dr\_Constantopoulos$, or indirectly, using the references $InfSys.Panos$, or $DSS.Constantopoulos$.

Note that a reference r w.r.t. a context c may be *ambiguous*, in the sense that it refers to more than one objects. That is, a reference r is ambiguous if there are two objects o, o' such that $r \in refs(o, c) \cap refs(o', c)$. It is possible for all references to an object w.r.t. a context to be ambiguous. However, in practice, at least one unique reference to an object is required to be used for external identification. Therefore, we distinguish an important class of lexicons and contexts defined as follows:

Definition 2.4 Well-defined lexicon. A lexicon l is called *well-defined* iff it satisfies the following conditions:

1. **Unique reference.**
   For every object recursively contained in l, there is a unique reference w.r.t. l, i.e. for all objects o, o' of l:
   $o \neq o' \Rightarrow \exists r \in refs(o, l) : \ \forall r' \in refs(o', l), r \neq r'$.
2. **Acyclicity.**
   For every nested subcontext c' of l, it holds: $c' \notin^\star c'$. $\diamond$

Definition 2.5 Well-defined context. A context c is called *well-defined* iff its lexicon is well-defined and $c \notin^\star c$. $\diamond$

In the example of Figure 1, contexts c_1, c_2, and c_3 are well-defined.

Acyclicity is an important property of a context c, as it ensures that the set of references $refs(o, c)$ of any object o recursively contained in c, can be computed in finite time.

Proposition 2.1 Let c be a well-defined context, and let o be an object recursively contained in c. Then, the following hold:

1. Every reference of o w.r.t. c has finite length, and
2. The set $refs(o, c)$ is finite. $\diamond$

Proof: See Appendix B.

We can assume a special context that recursively contains *all* objects of interest in a given application. We refer to this context as the *Information Base* (IB). As mentioned, a user can refer to an object using references. A reference to an object can be either *absolute*, i.e. w.r.t. context IB, or *relative*. As a convention, if the reference is prefixed by @ then it is a *absolute reference*, otherwise it is a *relative reference*. Relative references are resolved with respect to a context specified by the user, which we call the *Current Context* (CC). The user sets the CC through the Set Current Context operation, introduced in the following section.

In order to guarantee that every object has a unique absolute reference, we assume that the IB is a well-defined context. Therefore, we introduce the following axiom:

Axiom 2.1 Well-defined Information Base. The context IB is a well-defined context. $\diamond$

Support for relative naming of objects is an important feature of our model. The following situations can be handled:

- *Synonyms*: Two different references w.r.t a contexts are called *synonymous*, if they refer to the same object. We view synonyms as alternative ways to externally identify the same object. This is an important feature of our model because people often refer to the same concept using different names. For example, in Figure 1, the references $Nick$ and $Nikos$ (which are the english and the greek names or a person) w.r.t. context c_2 are synonyms, as they refer to the same object o_4. Similarly, the references $Dr\_Constantopoulos$, $InfSys.Panos$, and $DSS.Constantopoulos$ w.r.t. context c_1 are synonyms, as they refer to the same object o_1.

- *Homonyms*: Two different objects are called *homonymous* w.r.t. a given context if they have a common reference w.r.t. that context. If these two objects are recursively contained in a well-defined context c, then there exists a unique reference to each of these objects w.r.t. c. Note that there always exists such a context, because IB recursively contains every object and is assumed to be a well-defined context.

- *Anonyms*: An object o is called *anonymous* w.r.t. a context c, if o is associated with no name in c, i.e. $names(o, c) = \emptyset$. Intuitively, this is possible when an object is contained in a context, but we are not interested in naming it w.r.t. that context, or we don't know its name yet. However, there is no problem with the external identification of o, if there is a well-defined context c' such that $refs(o, c') \neq \emptyset$, and IB is such a context. For example, in Figure 1, objects o_4 w.r.t. context c_1 is anonymous.

3 Basic operations

In this section, we describe the basic operations of our model, and illustrate them through examples. Our model also includes auxiliary operations. Formal definitions and detailed algorithms of both basic and auxiliary operations are presented in the Appendix.

Our basic operations allow the user to do the following: to lookup for an object using references, create a new context, set the current context, insert/delete objects and object names into/from a context, and copy contexts. Additionally, our model provides basic operations for combining (Union operation) and comparing (Intersection and Difference operations) given lexicons and contexts, which are the most involved and will be discussed in detail.

During the computation of the comparing operations (Intersection and Difference) it is possible a new context c' to be created by copying the source or a copy of the source context c and then removing from the copy some of its objects. In this case, we need to know for each context c' its source context c in order to use it for further computation. Thus, in the definition of following operations, we will use the function src which associates a context c' with its source context c. Specifically, this function indicates that context c' has been derived from the source context c during the computation of the intersection and difference operations. Context c should have not been derived from another context and we denote this by $src(c) = c$.

In the following, assume an Information Base containing the context c_1 of Figure 1 under the name $Forth$.

3.1 Lookup operations

- **Lookup: lookup(r)**
  This operation takes a reference r as input and returns the set of objects o such that $r \in refs(o, c)$, where c is either the context IB if r is absolute, or the CC, otherwise. $\diamond$

- **Lookup one: lookupOne(r)**
  This operation takes a reference r as input and returns an object o such that if $R = refs(o, c)$ (where c is either the context IB if r is absolute, or the CC, otherwise) then $r \in R$ and cardinality of R is one. Otherwise, it returns ERROR. $\diamond$

- **Set current context: SCC(r)**
  This operation takes as input a reference r[1] to a context (call it c), and sets the CC to be the context c. $\diamond$

[1] In all operations, if a reference is ambiguous, an error message is returned.

Example: The operation $SCC(@.Forth)$ sets the CC to c_1, and the operation $SCC(@)$ sets the CC to IB.

3.2 Update operations

- **Create context: createCxt(l)**
  This operation takes a lexicon l as input, and returns a new context (call it c) such that $lex(c) = l$. Additionally, it sets $src(c) = c$. ⋄

  Example: The operation $createCxt(\{(o_1 : Panos), (c_1 : institute)\})$ results in the creation of a new context (call it c_{10}) with lexicon: $lex(c_{10}) = \begin{cases} o_1 : Panos \\ c_1 : institute. \end{cases}$

- **Insert an object into a context: insert(o, N, r)**
  This operation takes as input an object o, a set of names N, and a reference r to a context (call this context c). Then, it either inserts $(o : N))$ into the lexicon of c if object o is not contained in c, or adds the names in N to the c-names of o. Additionally, it sets $src(c) = c$. This is because, as a new object has been inserted into c, c is thought as a derivation of the original source of c. ⋄

- **Delete an object from a context: deleteObj(o, r)**
  This operation takes as input an object o and a reference r to a context, and deletes the pair $(o : N)$ from the lexicon of that context. ⋄

- **Delete an object name from a context: deleteName(o, n, r)**
  This operation takes as input an object o, a name n, and a reference r to a context (call this context c), and deletes the name n from the c-names of o. ⋄

3.3 Copy operations

- **Copy context: copyCxt(r)**
  This operation takes as input a reference r to a context (call this context c) and returns a new context (call it c') such that $lex(c') = lex(c)$. In other words: $copyCxt(r) = createCxt(lex(c))$. ⋄

  Example: The operation $copyCxt(Forth)$[2] returns a new context (call it c_{11}) with lexicon:

$$lex(c_{11}) = \begin{cases} o_1 : Dr\_Constantopoulos \\ o_4 : \\ o_5 : professor \\ c_2 : InfSys \\ c_3 : DSS \end{cases}$$

- **Deep copy context: deepCopyCxt(r)**
  This operation takes as input a reference r to a context (call this context c), and returns a new context (call it c'). Context c' contains the simple objects of c , and deep copies of the contexts contained in c[3]. ⋄

  Example: The operation $deepCopyCxt(Forth)$ returns a new context (call it c'_1) which contains copies of contexts c_2 and c_3 (call them c'_2 and c'_3). Contexts c'_1, c'_2, and c'_3 have the following lexicons:

$$lex(c'_1) = \begin{cases} o_1 : Dr\_Constantopoulos \\ o_4 : \\ o_5 : professor \\ c'_2 : InfSys \\ c'_3 : DSS \end{cases} \qquad lex(c'_2) = \begin{cases} o_1 : Panos \\ o_2 : head \\ o_3 : Manos \\ o_4 : Nikos, Nick \end{cases} \qquad lex(c'_3) = \begin{cases} o_6 : Panos \\ o_2 : head \\ o_1 : Constantopoulos \end{cases}$$

3.4 Combining and comparing operations

The Union operation combine two lexicons, one lexicon and a context, or two contexts.

- **Union:** $r_1 \uplus r_2$
  This operation takes as input two parameters r_1 and r_2 and returns a lexicon as a result. We distinguish three cases:

[2] Note that $Forth$ is a reference of c_1 w.r.t. the current context IB.

[3] In case that a context c'' is contained in two or more contexts that are recursively contained in c, then c'' is copied only once (i.e. c' does not recursively contain multiple copies of the same context).

1. If r_1 and r_2 are both lexicons, then the operation returns a lexicon l such that (let $O_1 = objs(r_1)$ and $O_2 = objs(r_2)$):
   (a) $objs(l) = O_1 \cup O_2$.
   (b) For each object $o \in objs(l)$: $\quad l(o) = \begin{cases} r_1(o) \cup r_2(o), & \text{if } o \in O_1 \cap O_2 \\ r_1(o), & \text{if } o \in O_1 \text{ and } o \notin O_2 \\ r_2(o), & \text{if } o \in O_2 \text{ and } o \notin O_1 \end{cases}$
   (c) Find all contexts of l with the same source (call this source c) and merge them into a new context with source c.
2. If r_1 is a lexicon and r_2 is a reference to a context (call this context c_2), then the operation returns a lexicon l such that:
   $l \quad = \quad r_1 \uplus (lex(c_2) \uplus \{(c_2 : \{str(r_2)\})\})$.
   In other words, to the lexicon of c_2, we add the context c_2, and use the name $str(r_2)$ as one of its names (the function $str(r)$ converts a reference r to a name by replacing dots by underscores).
3. If r_1 and r_2 are both references to contexts (call these contexts c_1 and c_2), then the operation returns a lexicon l such that:
   $l \quad = \quad (lex(c_1) \ \uplus \ \{(c_1 : \{str(r_1)\})\}) \ \uplus \ (lex(c_2) \ \uplus \ \{(c_2 : \{str(r_2)\})\})$. $\diamond$

Note that, in Case 1, if an object belongs to both lexicons then we can refer to it in the output lexicon, using any of its names in the two input lexicons. In Case 2 (where the second parameter is a context), context c_2 is added to the output lexicon under the name r_2. Intuitively, this adds a view over the objects of the combined lexicons as seen from c_2. We name this view r_2 to record the fact that this view has been referred to by the user as r_2[4]. Similarly, in Case 3 (where both inputs are contexts), contexts c_1 and c_2 are added to the output lexicon under the names r_1 and r_2, respectively.

Example: Assume that CC has been set to c_1. Then, the operations $lex(InfSys) \uplus lex(DSS)$ and $InfSys \uplus DSS$ return the lexicons l_1 and l_2, respectively, such that:

$$l_1 = \begin{cases} o_1 : Panos, \\ \quad Constantopoulos \\ o_2 : head \\ o_3 : Manos \\ o_4 : Nikos, Nick \\ o_6 : Panos \end{cases} \qquad l_2 = \begin{cases} o_1 : Panos, \\ \quad Constantopoulos \\ o_2 : head \\ o_3 : Manos \\ o_4 : Nikos, Nick \\ o_6 : Panos \\ c_2 : InfSys \\ c_3 : DSS \end{cases}$$

Note that object o_1 has two names: one originating from c_2 and the other from c_3. Note also that $InfSys$ and DSS are references (w.r.t. the CC) of contexts c_2 and c_3, respectively. Intuitively, the union of $InfSys$ and DSS contains the objects of l_1, as well as two views (that is contexts c_2 and c_3) over these objects, as seen from the Information System and DSS lab, respectively.

The Intersection operation computes the commonalities between two lexicons, one lexicon and a context, or two contexts. We first give the definition of the function $ComO$. Let l_1, l_2 be lexicons. We define $ComO(l_1, l_2) = objs(l_1) \cap objs(l_2) - \{c \in Cxt : src(c) \neq c\}$. $ComO(l_1, l_2)$ represents the common objects of the lexicons l_1 and l_2 that are not derived from any other context.

- **Intersection times:** $r_1 \ \owedge \ r_2$
  Intersection plus: $r_1 ; \owedge ; r_2$
  This operation takes as input two parameters r_1 and r_2, and returns a lexicon as a result. We distinguish three cases:

  1. If r_1 and r_2 are both lexicons, then the operation returns a lexicon l defined as follows (let $I = ComO(r_1, r_2)$):
     (a) If $o \in I$ then $o \in objs(l)$ and
        i. In case of $\owedge$: $l(o) = r_1(o) \cup r_2(o)$.
        ii. In case of ; $\owedge$;: $l(o) = r_1(o) \cap r_2(o)$.
     (b) If $o \notin I$ and o is a context recursively containing an object of I then:
        i. Make a deepcopy of o (call it c), and set the source of its copy context to be equal to the source of the original context.

[4] Obviously, the user can change this name using the operations: *deleteName* and *insert*.

ii. Remove from c and from every context recursively contained in c (i) any simple object that is not in I, and (ii) any context that is not in I and does not recursively contain objects in I.

iii. Add c to $objs(l)$ and define: $l(c) = \begin{cases} r_1(o), & \text{if } o \in objs(r_1) \\ r_2(o), & \text{if } o \in objs(r_2) \end{cases}$

(c) Find all contexts of l with the same source (call this source c) and merge them into a new context with source c.

2. If r_1 is a lexicon and r_2 is a reference to a context (call this context c_2), then the operation returns a lexicon l such that:
$l \;=\; r_1 \;\curlywedge\; (lex(c_2) \;\uplus\; \{(c_2' : \{str(r_2)\})\})$,
where c_2' is a new context such that $lex(c_2') = lex(c_2)$.

3. If r_1 and r_2 are both references to contexts (call these contexts c_1 and c_2), then the operation returns a lexicon l such that:
$l \;=\; (lex(r_1) \uplus \{(c_1' : \{str(r_1)\})\}) \;\curlywedge\; (lex(c_2) \uplus \{(c_2' : \{str(r_2)\})\})$,
where c_1' and c_2' are new contexts such that $lex(c_1') = lex(c_1)$ and $lex(c_2') = lex(c_2)$. $\diamond$

Note that, if an object belongs to both lexicons, then we can refer to it in the output lexicon using any of its names in the two input lexicons. In Case 2 (where the second parameter is a context), we add to the output lexicon a new context c_2' with name r_2. Intuitively, this adds a view over the objects of the output lexicon as seen from c_2. Context c_2' results from c_2 after removing from it and its nested subcontexts all simple objects that are not contained in the output lexicon. The same holds in Case 3.

Example: The operation $lex(InfSys) \;\curlywedge\; lex(DSS)$, returns the lexicon: $l_3' = \begin{cases} o_1 : \\ o_2 : head. \end{cases}$

The operation $lex(InfSys)$; $\curlywedge$;$lex(DSS)$, returns the lexicon: $l_3 = \begin{cases} o_1 : Panos, \\ \quad Constantopoulos \\ o_2 : head. \end{cases}$

Note that $I = \{o_1, o_2\}$. Therefore, objects o_1 and o_2 are added to the output lexicon in Step 1(a). Note that like in the Union operation, object o_1 has two names.

On the other hand, the operation $InfSys \;\curlywedge\; DSS$, returns the following lexicon:

$$l_4' = \begin{cases} o_1 : \\ o_2 : head \\ c_2'' : InfSys \\ c_3'' : DSS \end{cases} \qquad lex(c_2'') = \begin{cases} o_1 : Panos \\ o_2 : head \end{cases} \qquad lex(c_3'') = \begin{cases} o_2 : head \\ o_1 : Constantopoulos \end{cases}$$

The operation $InfSys$; $\curlywedge$;DSS, returns the following lexicon:

$$l_4 = \begin{cases} o_1 : Panos, Constantopoulos \\ o_2 : head \\ c_2'' : InfSys \\ c_3'' : DSS \end{cases} \qquad lex(c_2'') = \begin{cases} o_1 : Panos \\ o_2 : head \end{cases} \qquad lex(c_3'') = \begin{cases} o_2 : head \\ o_1 : Constantopoulos \end{cases}$$

Note that $I = \{o_1, o_2\}$. Contexts c_2'' and c_3'' are derived from contexts c_2 and c_3 after removing all simple objects not in I and thus, $src(c_2'') = src(c_2) = c_2$ and $src(c_3'') = src(c_3) = c_3$ (Step 1(b)ii). Contexts c_2'' and c_3'' are added to the output lexicon in Step 3.

Finally, the Difference operation computes the differences between two lexicons, one lexicon and a context, or two contexts.

- **Difference:** $r_1 \;\ominus\; r_2$
This operation takes as input two parameters r_1 and r_2, and returns a lexicon as a result. We distinguish three cases:

1. If r_1 and r_2 are both lexicons, then the operation returns a lexicon l such that (let $D = objs(r_1) - objs(r_2)$ and $I = objs(r_1) \cap objs(r_2)$):
   (a) If $o \in D$ then $o \in objs(l)$ and $l(o) = r_1(o)$.
   (b) If $o \in I$ and o is a context recursively containing an object of D then:
      i. Make a deepcopy of o (call it c), and set the source of its newly derived context (copy) to be equal to the source of the original context.
      ii. Remove from c and from every context recursively contained in c, any simple object that is not in D.

iii. Add c to $objs(l)$ and define: $l(c) = r_1(o)$.

(c) No other object is in $objs(l)$.

2. If r_1 is a lexicon and r_2 is a reference to a context (call this context c_2), then the operation returns a lexicon l such that:
$$l \quad = \quad r_1 \ominus lex(c_2).$$
3. If r_1 is a reference to a context (call this context c_1) and r_2 is a lexicon, then the operation returns a lexicon l such that:
$$l \quad = \quad (lex(c_1) \uplus \{(c_1' : \{str(r_1)\})\}) \ominus r_2,$$
where $lex(c_1') = lex(c_1)$.
4. If r_1 and r_2 are both references to contexts (call these contexts c_1 and c_2), then the operation returns a lexicon l such that:
$$l \quad = \quad (lex(c_1) \uplus \{(c_1' : \{str(r_1)\})\}) \ominus (lex(c_2) \uplus \{(c_2 : \{str(r_2)\})\}),$$
where $lex(c_1') = lex(c_1)$. $\diamond$

Note that, in cases 3 and 4, if the operands are references to contexts then the Difference operation operates on their respective lexicons.

Example: The operation $lex(InfSys) \ominus lex(DSS)$, returns the lexicon: $l_5 = \begin{cases} o_3 : Manos \\ o_4 : Nikos, Nick. \end{cases}$

Note that objects o_3 and o_4 are objects contained in c_2 but not in c_3. That is, $D = \{o_3, o_4\}$. These objects are added to the output lexicon in Step 1(a). Additionally, note that $I = \{o_1, o_2\}$. As I does not contain any context, Step 1(b) is not executed.

3.5 A more complex example

In this subsection, we illustrate the operations Union, Intersection, and Difference over contexts containing nested subcontexts.

Consider the Information Base illustrated in Figure 2. Context IB contains two contexts c_1 and c_4, namely $ManosView$ and $AnastasiaView$, respectively. These contexts represent the views of Manos and Anastasia regarding the institute. Context c_4 contains the already seen objects o_1, o_5 and c_2, as well as a new context c_5 that represents the view of Anastasia regarding the Decision Support Systems lab. The fact that both contexts c_1 and c_4 share context c_2 indicates that both Manos and Anastasia have the same view for the Information Systems lab.

$$lex(IB) = \begin{cases} c_1 : ManosView \\ c_4 : AnastasiaView \end{cases}$$

$$lex(c_1) = \begin{cases} o_1 : Dr\_Constantopoulos \\ o_4 : \\ o_5 : professor \\ c_2 : InfSys \\ c_3 : DSS \end{cases} \qquad lex(c_4) = \begin{cases} o_1 : Constantopoulos \\ o_5 : professor \\ c_2 : ISgroup \\ c_5 : DSS \end{cases}$$

$$lex(c_2) = \begin{cases} o_1 : Panos \\ o_2 : head \\ o_3 : Manos \\ o_4 : Nikos, Nick \end{cases} \qquad lex(c_3) = \begin{cases} o_6 : Panos \\ o_2 : head \\ o_1 : Constantopoulos \end{cases}$$

$$lex(c_5) = \begin{cases} o_1 : Panos, \\ \quad Constantopoulos \\ o_7 : Anastasia \end{cases}$$

Figure 2: An Information Base context.

Using our operations we can do the following: (i) combine the views of Manos and Anastasia to get a wider view of the institute, (ii) compare the views of Manos and Anastasia to get their commonalities, and their differences regarding the institute.

In the following, assume that the current context is the context IB, i.e. CC = IB.

The operation $ManosView \uplus AnastasiaView$ combines the views of Manos and Anastasia, and returns the following lexicon:

$$l_6 = \begin{cases} o_1 : Dr\_Constantopoulos, Constantopoulos \\ o_4 : \\ o_5 : professor \\ c_2 : InfSys, ISgroup \\ c_3 : DSS \\ c_5 : DSS \\ c_1 : ManosView \\ c_4 : AnastasiaView \end{cases}$$

Note that there are two different contexts c_3 and c_5 with the same name. However, no ambiguity is caused, as these contexts also belong to contexts c_1 and c_4, respectively. Therefore, we can refer to c_3 and c_5 uniquely through the references $ManosView.InfSys$ and $AnastasiaView.InfSys$, respectively.

The operation $ManosView \uparrow AnastasiaView$ computes the commonalities of the views of Manos and Anastasia, and returns the following lexicon:

$$l_7 = \begin{cases} o_1 : \\ o_5 : professor \\ c_2 : \\ c_3' : DSS \\ c_5' : DSS \\ c_1' : ManosView \\ c_4' : AnastasiaView \end{cases}$$

$$lex(c_3') = \{o_1 : Constantopoulos$$
$$lex(c_5') = \{o_1 : Panos$$
$$lex(c_1') = \begin{cases} o_1 : Dr\_Constantopoulos \\ o_5 : professor \\ c_2 : InfSys \\ c_3' : DSS \end{cases}$$
$$lex(c_4') = \begin{cases} o_1 : Constantopoulos \\ o_5 : professor \\ c_2 : ISgroup \\ c_5' : DSS \end{cases}$$

Note that the set I of the Intersection algorithm is $\{o_1, o_5, c_2\}$. That is, objects o_1, o_5, and c_2 are the common objects of c_1 and c_4. These objects are added to the lexicon of the intersection in Step 1(a) of the Intersection algorithm. Contexts c_3', and c_5' are copies of contexts c_3, and c_5 after removing all simple objects not in I. Contexts c_3', and c_5' are added to the lexicon of the intersection in Step 1(b) of the Intersection algorithm. These contexts represent views over the objects in I as seen from c_3, and c_5, respectively. Contexts c_1' and c_4' are copies of contexts c_1 and c_4 after removing all simple objects not in I, and all contexts not in I which do not recursively contain objects in I. Contexts c_1' and c_4' are added to the lexicon of the intersection in Step 3 of the Intersection algorithm. Contexts c_1' and c_4' represent views over the objects in I as seen from c_1 and c_4, respectively. On the other hand, the operation $ManosView ; \uparrow ; AnastasiaView$ also computes the commonalities of the views of Manos and Anastasia, and returns the following lexicon:

$$l_7 = \begin{cases} o_1 : Dr\_Constantopoulos, \\ \quad Constantopoulos \\ o_5 : professor \\ c_2 : InfSys, ISgroup \\ c_3'' : DSS \\ c_5'' : DSS \\ c_1'' : ManosView \\ c_4'' : AnastasiaView \end{cases}$$

$$lex(c_3'') = \{o_1 : Constantopoulos$$
$$lex(c_5'') = \{o_1 : Panos$$
$$lex(c_1'') = \begin{cases} o_1 : Dr\_Constantopoulos \\ o_5 : professor \\ c_2 : InfSys \\ c_3'' : DSS \end{cases}$$
$$lex(c_4'') = \begin{cases} o_1 : Constantopoulos \\ o_5 : professor \\ c_2 : ISgroup \\ c_5'' : DSS \end{cases}$$

The operation $ManosView \ominus AnastasiaView$ computes the differences between the views of Manos and Anastasia, and returns the lexicon:

$$l_8 = \begin{cases} o_4 : \\ c_2''' : InfSys \\ c_3 : DSS \\ c_1''' : ManosView \end{cases}$$

$$lex(c_2''') = \{o_4 : Nikos, Nick$$
$$lex(c_1''') = \begin{cases} o_4 : \\ c_2''' : InfSys \\ c_3 : DSS \end{cases}$$

Note that o_4 and c_3 are objects contained in c_1 but not in c_4. Note also that the Difference operation is not recursively applied to the nested subcontexts of $ManosView$ and $AnastasiaView$. Therefore, if the user wants to go into more depth, he has to call explicitly the operation $ManosView.InfSys \ominus AnastasiaView.InfSys$.

4 Properties of the operations

In the course of the Union, Intersection, and Difference operations, nested subcontexts are copied and merged into new contexts. This implies that even the same operation, if executed twice, will result into two different lexicons. However, these two lexicons will be related by the equivalence relation defined below.

Definition 4.1 Equivalence relation. We define the equivalence relation, denoted by $\sim$, as follows:

1. Let c and c' be contexts. Then, it holds:
   $c \sim c' \Leftrightarrow (c = c') \vee (lex(c) \sim lex(c') \wedge src(c) = src(c'))$
2. Let l and l' be lexicons. Then, it holds:

$$
\begin{aligned}
l \sim l' \Leftrightarrow \quad & (\forall o \in \mathcal{S} : \ (o{:}N) \in l \Leftrightarrow (o{:}N) \in l') \wedge \\
& (\forall c \in Cxt : \\
& ((c{:}N) \in l \Rightarrow \ \exists c' : \ (c'{:}N) \in l' \ \wedge \ c \sim c') \wedge \\
& ((c{:}N) \in l' \Rightarrow \ \exists c' : \ (c'{:}N) \in l \ \wedge \ c \sim c'))
\end{aligned}
$$

where $\mathcal{S}$ denotes the set of simple objects. $\diamond$

Proof: It can be easily seen that this relation is *reflexive*, *symmetric*, and *transitive*. Thus, it is an equivalence relation. □

It turns out that the operations of Union and Intersection have the properties of commutativity, associativity, and distributivity over lexicons and contexts, just like ordinary set union and intersection. These properties are important as they offer flexibility in the execution of operations. Specifically, commutativity allows us to ignore the order between two operands. Associativity allows us to omit an indication of precedence, in expressions with more than one instance of the operator. Finally, distributivity allows to factor out or to distribute on operand, so as to optimize further processing.

Proposition 4.1 Let A, B, and C be contexts or lexicons. The following properties hold:

- **Commutativity**:
  (1) $A \uplus B \sim B \uplus A$
  (2) $A \cap B \sim B \cap A$
  (3) $A; \Rightarrow ; B \sim B; \Rightarrow ; A$
- **Associativity**:
  (4) $(A \uplus B) \uplus C \sim A \uplus (B \uplus C)$
  (5) $(A \cap B) \cap C \sim A \cap (B \cap C)$
  (6) $(A; \Rightarrow ; B); \Rightarrow ; C \sim A; \Rightarrow ; (B; \Rightarrow ; C)$
- **Distributivity**:
  (7) $(A \cap B) \uplus C \sim (A \uplus C) \cap (B \uplus C)$
  (8) $(A \uplus B) \cap C \sim (A \cap C) \uplus (B \cap C)$ $\diamond$

Proof: See Appendix B.

For example (see Figure 1), assume the current context to be the context IB. The operation

$$(ManosView.InfSys \ \cap \ AnastasiaView.DSS) \ \uplus \ ManosView \tag{1}$$

computes the commonalities between the Information Systems lab as seen from Manos and the DSS lab as seen from Anastasia and then combines these commonalities with the view of Manos for the institute to get a wider view of that institute. Let l_1 be the intermediate lexicon returned by the operation $ManosView.InfSys \ \cap \ AnastasiaView.DSS$ and let l_2 be the lexicon returned by the operation 1. Then, we have:

$$l_1 = \begin{cases} o_1 : Panos \\ c_2' : ManosView\_InfSys \\ c_5' : AnastasiaView\_DSS \end{cases}$$

$$lex(c_2') = \{ o_1 : Panos$$

$$lex(c_5') = \{ o_1 : Panos, Constantopoulos$$

$$l_2 = \begin{cases} o_1 : Panos, Dr\_Constantopoulos \\ o_4 : \\ o_5 : professor \\ c_2 : ManosView\_InfSys, InfSys \\ c_3 : DSS \\ c_5' : AnastasiaView\_DSS \\ c_1 : ManosView \end{cases}$$

Note that during the computation of the operation $l_1 \ \uplus \ ManosView$ context c_2 is merged with context c_2' into context c_2 as $src(c_2') = src(c_2) = c_2$ (see Step 1(c) of the Union algorithm at page 6 and the detailed algorithms of the Operations A.14 and A.13 in Appendix A).

On the other hand, the operation

$$(ManosView.InfSys \ \uplus \ ManosView) \ \cap \ (AnastasiaView.DSS \ \uplus \ ManosView) \tag{2}$$

get two wider views of the institute as seen from Manos by (i) combining $ManosView$, context c_1, with the Information Systems lab as seen from Manos, context c_2, (call the returned lexicon l_3) and (ii) combining $ManosView$ with the DSS lab as seen from Anastasia, context c_5, (call the returned lexicon l_3), and then

computes the commonalities of these two wider views. Let l_4 be the lexicon returned by the operation 2. Then, we have:

$$l_3 = \begin{cases} o_1 : Panos, \\ \quad Dr\_Constantopoulos \\ o_2 : head \\ o_3 : Manos \\ o_4 : Nikos, Nick \\ o_5 : professor \\ c_2 : ManosView\_InfSys, \\ \quad InfSys \\ c_3 : DSS \\ c_1 : ManosView \end{cases} \quad l_4 = \begin{cases} o_1 : Panos, Constantopoulos, \\ \quad Dr\_Constantopoulos \\ o_4 : \\ o_5 : professor \\ o_7 : Anastasia \\ c_2 : InfSys \\ c_3 : DSS \\ c_5 : AnastasiaView\_DSS \\ c_1 : ManosView \end{cases}$$

$$l_5 = \begin{cases} o_1 : Panos, \\ \quad Dr\_Constantopoulos \\ o_4 : \\ o_5 : professor \\ c_2 : ManosView\_InfSys, \\ \quad InfSys \\ c_3 : DSS \\ c_5'' : AnastasiaView\_DSS \\ c_1 : ManosView \end{cases} \quad lex(c_5'') = \begin{cases} o_1 : Panos, \\ \quad Constantopoulos \end{cases}$$

According to property (6), lexicons l_2 and l_5 are equivelant.

We now define an important class of lexicons, called *operational lexicon*, which is closed over the operations Union, Intersection, and Difference. This closure property is expressed in Lemma 4.1 and Theorem 4.1.

In the following, we shall call *root context* any context of a lexicon l (resp. context c) which is not recursively contained in any other context of l (resp. c).

Definition 4.2 Operational lexicon. A lexicon l is called *operational* iff

1. it is a well-defined lexicon,
2. if c is a root context of l then $src(c)$ is well-defined, and
3. any object of l which is not a root context is recursively contained in a root context of l. $\diamond$

Lemma 4.1 Closure of the operationality property: two lexicons. Let l_1, l_2 be two operational lexicons. Assume that every root context c of l_1 (resp. l_2) has a name n w.r.t. l_1 (resp. l_2) such that there is no name n w.r.t. l_2 (resp. l_1). Then the operation $l_1 \odot l_2$, where $\odot \in \{ \uplus , \upharpoonright , \ominus \}$, results in an operational lexicon. $\diamond$

Proof: See Appendix B.

Theorem 4.1 Closure of the operationality property: three or more lexicons. Let $l_1 \odot_1 \ldots \odot_{k-1} l_k$ be operational lexicons. If every root context c of l_i has a name n w.r.t. l_i such that there is no name n w.r.t. each of $l_1, \ldots, l_{i-1}, l_{i+1}, \ldots, l_k$, then the sequence of operations $l_1 \odot_1 \ldots \odot_{k-1} l_k$, where the operations $\odot_i \in \{ \uplus , \upharpoonright , ; \Cap ;, \ominus \}$ are executed in any order, results in an operational lexicon. $\diamond$

Proof: See Appendix B.

The following theorem expresses that the Union, Intersection, and Difference operations preserve the well-definedness property of contexts.

Theorem 4.2 Closure of the well-definedness of contexts. Let $r_1, \ldots, r_k$ be references of the well-defined contexts $c_1, \ldots, c_k$. If $str(r_i)$ is not a name of an object w.r.t. each of $c_1, \ldots, c_k$, then the sequence of operations $r_1 \odot_1 \ldots \odot_{k-1} r_k$, where the operations $\odot_i \in \{ \uplus , \upharpoonright , ; \Cap ;, \ominus \}$ are executed in any order, results in a well-defined lexicon. $\diamond$

Proof: See Appendix B.

5 Cooperation using contexts

As pointed out in [19], contexts can serve as the basis for addressing certain issues related to cooperative work such as workspaces, versioning, and configuration. In this section, we present a comprehensive example that illustrates the use of contexts in a simple cooperation environment. A cooperation environment is usually organized into named repositories, called $workspaces$, to allow workers to share information concerning the work done on an object, in a secure and orderly manner [12, 5]. In a cooperation environment, there are three kinds of workspaces: *public*, *group*, and *private*.

The public workspace contains fully verified (i.e. released) and finished object versions, which have reached the final state of robustness and therefore cannot be updated or deleted. However, any worker can read this workspace, and can append new object versions to it.

The group workspace contains object versions that have reached a reasonable state of robustness, and therefore can be shared by two or more workers. Thus, the combination of work in-progress between different workers is achieved. This process is necessary before a version is finalized and migrates to the public workspace. Object versions of the group workspace cannot be updated but they can be deleted.

The private workspace consists of a number of user workspaces. Each user workspace is owned by a specific user and can be accessed only by him. User workspaces contain temporary object versions which are expected to undergo a significant amount of update before reaching a reasonably robust state (and moved to the group or to the public workspace). Therefore, object versions of a user workspace can be updated or deleted by its user.

Object versions can be moved in and out from the public workspace through the $check\text{-}in$ and $check\text{-}out$ operations, and in and out from the group workspace through the $import$ and $export$ operations. A user checks out a version from the public workspace into his private workspace, where he can make changes. The new version is possibly exported to the group workspace for integration testing with other objects. To correct errors, the version has to be imported to the private workspace. Finally, a new verified version is checked in the public space and is linked (with a version history link) to the original public version from which it was derived. At this point, the version history of the object has been updated.

An object is, in general, composed of other objects that are either atomic or composite. In our model, a version of an atomic object can be thought of as a simple object. Recall that a simple object is an object of the Information Base that is not a context. A *configuration* is a version of a composite object, composed of particular versions of its components. Therefore, a configuration can be thought of as a context that contains versions of its components. We refer to contexts that represent configurations as *configuration contexts.*

A version history of an object can be thought of as a context that contains (a) versions of the object, and (b) links from one version to another that indicate version derivation. We call such contexts, *history contexts.* The context types described above, are organized hierarchically (through the ISA relation) as shown in Figure 3.

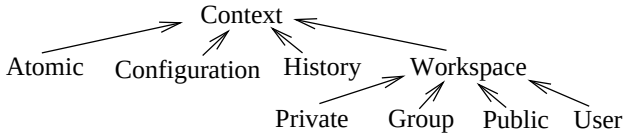


Figure 3: Context types of the cooperation environment.

A cooperation environment can be thought of as an Information Base (IB), containing six contexts: ATOMIC, CONFIG, HISTORY, PUBLIC, PRIVATE, and GROUP (see Figure 4). The context ATOMIC contains all versions of atomic objects. The context CONFIG contains all configuration contexts, and the context HISTORY contains all history contexts. The context PUBLIC contains all objects in the public workspace, which we assume to be history contexts, and the context PRIVATE contains all the user contexts. A user context may contain history contexts, configuration contexts, and atomic objects. A user context may also contain results of operations on contexts. The context GROUP essentially contains results of operations on contexts.

$$lex(IB) = \begin{cases} PUBLIC : Public \\ PRIVATE : Private \\ GROUP : Group \\ HISTORY : History \\ CONFIG : Config \\ ATOMIC : Atomic \end{cases}$$

$$lex(PRIVATE) = \begin{cases} 1 : Manos \\ 2 : Anastasia \\ 3 : Nikos \end{cases}$$

$$lex(PUBLIC) = \begin{cases} 100 : A \\ 200 : I \\ 300 : M \end{cases}$$

$$lex(HISTORY) = \begin{cases} 100 : A \\ 200 : I \\ 300 : M \end{cases}$$

$$lex(GROUP) = \emptyset$$

$$lex(CONFIG) = \begin{cases} 10 : A_1 \\ 11 : A_2 \end{cases}$$

$$lex(ATOMIC) = \begin{cases} 20 : I_1 \\ 21 : I_2 \\ 30 : M_1 \\ 31 : M_2 \\ 32 : M_3 \end{cases}$$

Figure 4: Initial lexicons of IB and the six contexts of the cooperation scenario.

5.1 Cooperation Scenario

We consider a cooperation scenario in which three authors cooperate on the revision of an article, composed of an introduction and a main section. The initial state of our cooperation scenario is shown in Figures 4 and 5. In Figure 5, we use the following conventions: A symbol of the form $o : n_1, n_2, \ldots$ denotes object o with names $n_1, n_2, \ldots$, e.g. $100 : A$ denotes object 100 with a single name A. Solid line rectangles represent workspaces, dashed line rectangles represent history contexts, rounded solid line boxes represent configuration contexts, and thick dots represent atomic objects.

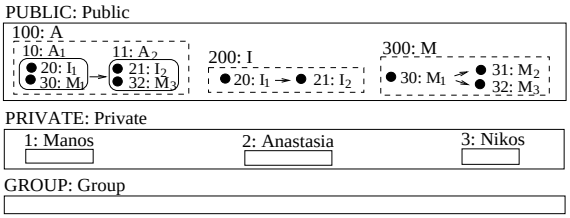


Figure 5: Initial state of the cooperation scenario.

Specifically, the initial state of the Information Base is as follows (see Figure 4 and 5):

- The context PUBLIC contains a history context for the article, and a history context for each component of the article. The history context for the article is the context 100 with name A, the history context for the introduction is the context 200 with name I, and the history context for the main section is the context 300 with name M, as shown in Figures 4 and 5. The names A, I and M stand for "Article", "Introduction" and "Main section", respectively[5]. Here, versions of the introduction and the main section are simple objects, as any piece of (unstructured) text is considered to be an atomic object. Context 100 contains two contexts (these are 10 and 11) representing two different versions of the article, as well as a link object from context 10 to context 11. Similarly, contexts 200 and 300 contain versions of the introduction and the main section, respectively, as well as link objects.

- The context PRIVATE contains three user contexts, one for each author. The first author is assigned the user context 1 with name $Manos$, the second author is assigned the user context 2 with name $Anastasia$, and the third author is assigned the user context 3 with name $Nikos$.

- The context GROUP is initially empty.

We refer to a user workspace as the *home workspace* of the corresponding user. We assume that each user has his own variable *current context* (`CC`) whose initial value is his home workspace. For each user, the value of the variable `Username` is his login name. Also, the name of his home workspace w.r.t. the context PRIVATE, is his login name. Finally, the value of the variable `Home` is the global reference of the home workspace of the user. For example, for user $Manos$, `CC` $= 1$, `Username` $= Manos$, and `Home` $= @.Private.Manos$. In the following, whenever we refer to the variables `CC`, `Home`, and `Username` we use their values. Variables are written in a special character font to be distinguished from strings.

5.2 Cooperation commands

During revision of the article, each author has four commands at his disposal, as described below. These commands are high level operations, implemented using the basic operations of the model. An example of their use is given in the following subsection.

- **check-out**(r, n)
  This operation takes as input a reference r w.r.t. the public workspace, and a name n, and does the following:

  1. Copies the history context of the version referred to by r, from the public workspace into the home workspace of the user, under the same name.
  2. Copies the version referred to by r (call this version v), from the public workspace into the `CC` (call this copy v').

[5] In practice one would use meaningful names instead of A, I and M, e.g. $Article\_on\_Contexts$ instead of A.

1. Commands by user Manos.
$/* CC = 1,\ Home = @.Private.Manos,$
$Username = Manos\ */$

(a) $check\text{-}out(A.A_2, A_3)$.
(b) $SCC(A_3)$.
(c) $check\text{-}out(I.I_2, I_3)$.
(d) $\cdots$.
(e) $SCC(Home)$.
(f) $a_2 = lookup(A.A_2)$.
(g) $insert(\ createCxt(\{(a_2 : Public, Current)\}), \{TMP\}, Home)$.
(h) $export(TMP, A_3, A)$.

2. Commands by user Anastasia.
$/* CC = 2,\ Home = @.Private.Anastasia,$
$Username = Anastasia.\ */$

(a) $import(A, TMP)$.
(b) $check\text{-}out(A.A_2, A_3)$.
(c) $SCC(A_3)$.
(d) $check\text{-}out(I.I_2, I_3)$.
(e) $check\text{-}out(M.M_3, M_4)$.
$\cdots$
(f) $SCC(Home)$.
(g) $export(TMP, A_4, A)$.

3. Commands by user Nikos.
$/* CC = 3,\ Home = @.Private.Nikos,$
$Username = Nikos.\ */$

(a) $import(A, TMP)$.
(b) $copy(A.Current, A_5)$.
$\cdots$
(c) $check\text{-}in(A_5, TMP, A_3)$.

Figure 6: User commands during a cooperation session.

3. Adds v' into the copy of the history context, under the name n.
4. Updates the copy of the history context by adding a link from v to v'. $\diamond$

- **check-in**(r, h, n)
  This operation takes as input a reference r w.r.t. the CC, a reference h w.r.t. the public workspace, and a name n. Then, it copies the version referred to by r from the CC into the history context of the public workspace referred to by h, under the name n. $\diamond$

- **export**(r_1, r_2, n)
  This operation takes as input two references r_1 and r_2, w.r.t. the CC, and a name n. Then, it does the following:

  1. Creates a context (call it c), whose lexicon is the union of the lexicon of the context referenced by r_1, and the context referenced by r_2 (call the last context c_2).
  2. Creates a link from the last edited version (that is, the one named $Current$) to the context c_2.
  3. Context c_2 is assigned two names w.r.t. c: (a) The value of Username, to indicate the author of the version, and (b) $Current$, to indicate that c_2 is the last edited version (the name $Current$ is then deleted from the names of the previously edited version).
  4. Copies the context c into the group workspace, under the name n. $\diamond$

- **import**(r, n)
  This operation takes as input a reference r, w.r.t. the group workspace, and a name n. Then, it does the following:

  1. Copies the context referenced by r from the group workspace into the CC, under the name n.
  2. Deletes the original context from the group workspace. $\diamond$

5.3 An example of cooperation

In this subsection, we present and discuss the commands issued by each author during a cooperation session. These commands are shown in Figure 6.

Commands by Manos

User Manos checks-out version A_2 of the article, and copies it as version A_3 to his home workspace (see Figure 7.(a)). This is done through the command $check\text{-}out(A.A_2, A_3)$. As the user wants to revise version A_3, he focuses on context A_3. This is done through the command $SCC(A_3)$. As he wants to revise the introduction, he checks-out object I_2 to his home workspace (replacing the object I_2 contained in context A_3 by a new version of the introduction, named I_3, as shown in Figure 7.(b)). This is done through the operation $check\text{-}out(I.I_2, I_3)$. The local editing of I_3 is indicated by three dots in Figure 6.

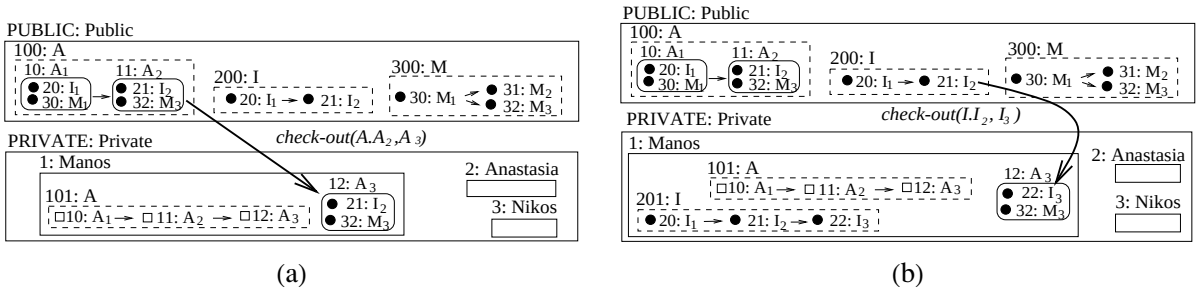


Figure 7: Manos' interaction with the public and private workspaces.

Manos is the first to edit the article and therefore, he needs to create the necessary initial environment to exchange information with the other authors. Intuitively, this environment works as a coordinating unit for comparing the versions prepared by the different authors, before the final version is checked in the public workspace. This comparison requires knowledge about which authors have edited a particular version, and what changes have been made to it. Specifically, he creates a context named TMP in his home workspace

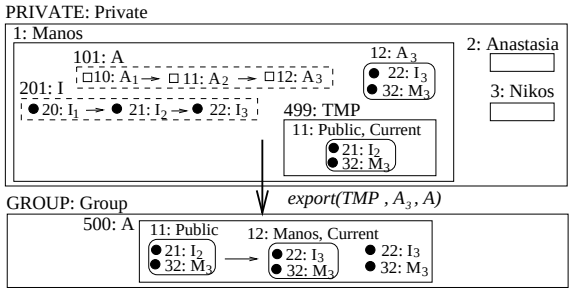


Figure 8: Manos' export to the group workspace.

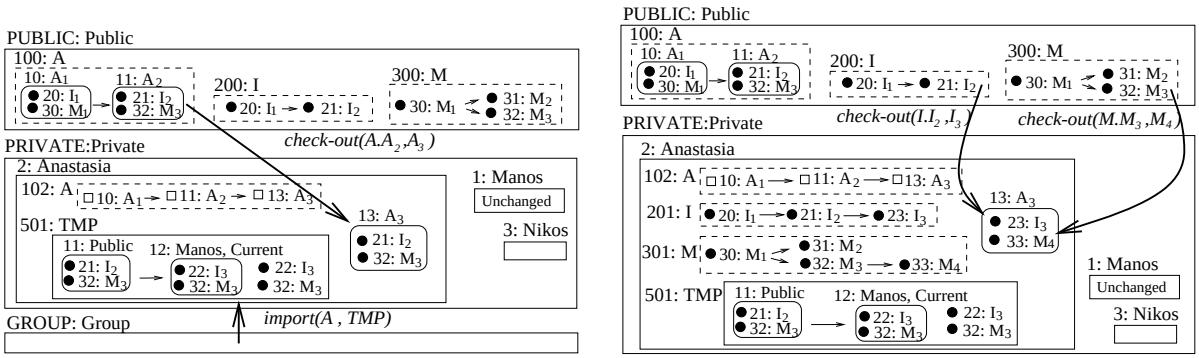
(see Figure 8). This context contains the original version of article A_2 with names: (a) $Public$, to indicate that this is the original version, contained in the public workspace, and (b) $Current$, to indicate that it is the last edited version. This is done through the operation 1(f) of Figure 6. Then, the user exports the necessary information into the group workspace for further revision. Specifically, he creates a context named A w.r.t. the group workspace, that contains the original context A_2, the revised context A_3, and a link from A_2 to A_3 that denotes the direction of the revision (see Figure 8). The object A_3 contained in A is assigned two names w.r.t. A: (a) $Manos$, to indicate the author of the version, and (b) $Current$, to indicate that it is the last edited version. This is done through the command, $export(TMP, A_3, A)$.

Commands by Anastasia

Subsequently, user Anastasia imports context A into her home workspace, under the name TMP (see Figure 9.(a)). She also checks-out version A_2 of the article, and copies it as version A_3 in her home workspace[6] (see Figure 9.(a)). As she wants to revise version A_3, she focuses on context A_3. She then checks-out I_2 and M_3, and copies them as I_3 and M_4 in her home workspace (see Figure 9.(b)). Anastasia can now start editing I_3 and M_3 taking into account the modifications that Manos has done on A_3. This information can be obtained by performing context difference and context intersection between $TMP.Manos$ and $TMP.Public$. Commands corresponding to local processing, such as context comparisons or editing, are indicated by three dots in Figure 6.

Once editing is finished, she exports A_3 and the information contained in TMP to the group workspace for further revision (see Figure 10.(a)). Specifically, she creates context 502, named A that contains the lexicon of TMP, the revised context A_3 (object 13) and a link from object 12 to 13 that shows the direction of the revision. The object 13 contained in A, is assigned two names w.r.t. A: (a) $Anastasia$, to indicate

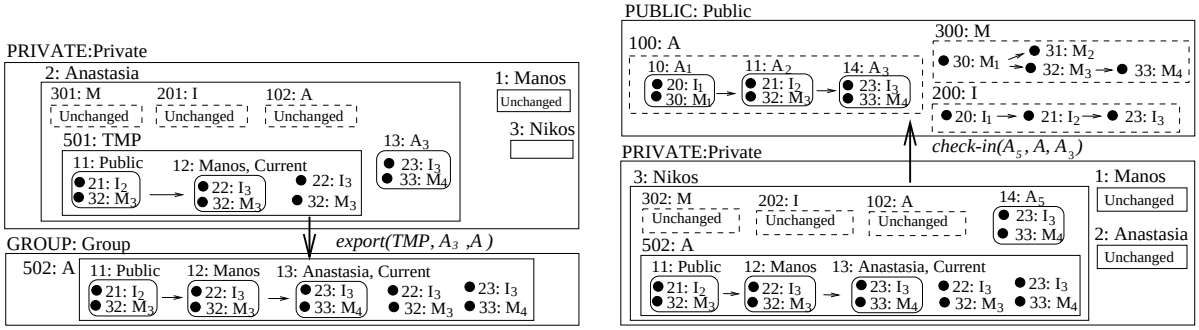
[6]Note that she uses the same name A_3 as Manos did, for naming a different version of the article. However, there is no ambiguity as the two A_3's are contained in different contexts.



(a) (b)

Figure 9: Anastasia's interaction with the public and private workspaces.

the author of the version, and (b) $Current$ to indicate the last edited version. Further, the name $Current$ is deleted from the names of object 12. Note that context 502 contains two objects with the same name (these are 22 and 23). However, these objects can be referenced uniquely through contexts 12 and 13, respectively.



(a): Anastasia's export to the group workspace. (b): Nikos' check-in to the public workspace.

Figure 10:

Commands by Nikos

Finally, user Nikos imports context A to his home workspace under the name TMP. He then copies the current version of the article (i.e. the version named $Current$ w.r.t. TMP) into A_5. After editing the copy, he checks it in the history context referred to by A w.r.t. the public workspace, under the name A_3 (see Figure 10.(b)).

We would like to stress that the purpose of the example presented here, is to illustrate the use of context in a simple cooperation environment. The commands *check-in*, *check-out*, *import* and *export*, are examples of simple communication commands that can be implemented using the basic operations of our model.

In a more complex environment, however, the users most likely will need information on various aspects of the cooperation. For example, in a software engineering project, where several groups are developing software in parallel, a coordinating unit may need to compare modules coming from various groups, before merging them into a single module. Such information can be obtained through more sophisticated higher level commands that can also be implemented using the basic operations of the model.

The Information Base can be organized in a number of different ways. Choosing the appropriate organization is a design problem that depends on the application. However, this problem lies outside the scope of this paper.

6 Related work

As mentioned in the introduction, the notion of context has appeared in several areas, and has been treated in various ways depending on the purposes of the particular application. However, the semantics given to the notion of context in these areas are not always the same and the various semantics are not always comparable. In this section, we compare our approach with other approaches that treat the notion of context in a comparable way.

As already mentioned, our model has been inspired by the work of Mylopoulos and Motschnig-Pitrik [19, 20], and incorporates previous work by Theodorakis and Constantopoulos [30]. In the introduction, we provide a comparison of our model with these two approaches.

HAM [3] is a general purpose abstract machine that supports contexts. In HAM, a graph usually contains all the information regarding a general topic and contexts are used to partition the data within a graph. Therefore, a context may contain nodes, links, or other contexts. Contexts are organized hierarchically, i.e. a context is contained in only one other context. By contrast, in our model, a context may be contained in more than one contexts. Contexts in HAM have been used to support configurations, private workspaces, and version history trees [6]. HAM provides a set of context editing, context inquiry, and context attribute operations. From these, we will discuss only the context editing operations, as inquiries on contexts and attributes of contexts are not considered in our paper. All the context editing operations of HAM, namely *createContext*, *destroyContext*, *compactContext*, and *mergeContext*, can be simulated in our model using its operations. On the other hand, HAM does not support name relativism.

In [28], the notion of context is used to support collaborative work in hypermedia design. A context node contains links, terminal nodes, and other context nodes. Furthermore, context nodes are specialized into annotations, public bases, hyperbases, private bases, and user contexts. Using this notion of context, the authors define operations *check-in* and *check-out* for hypermedia objects. However, there is no support for name relativism, and neither are generic operations on contexts provided.

The notion of context has also appeared in the area of heterogeneous databases [25, 21, 11]. There, the word ”context” refers to the implicit assumptions underlying the manner in which an agent represents or interprets data. To allow exchange between heterogeneous information systems, information specific to them can be captured in specific contexts. Therefore, contexts are used for interpreting data. At present our model cannot be compared with these works, because it does not address heterogeneous databases, as we assume a unique Information Base (which guarantees that real world objects are represented by unique objects in the Information Base).

7 Conclusions

In this paper, we developed a model for representing contexts in information bases along with a set of operations for creating, updating, combining, and comparing contexts. A context is treated as a special object which is associated to a set of objects and a lexicon, i.e. a binding of names to these objects. Contexts may overlap, in the sense that an object may be contained in more than one contexts simultaneously. Contexts may also be nested, in the sense that a context may contain other contexts. Also, a context may be contained in more than one contexts.

The main contributions of this work are:

- It allows an object to have zero, one, or more names, not necessarily unique, w.r.t. a context. Therefore, we can handle synonymous, homonymous, and anonymous objects. Possible name ambiguities are resolved by assuming that objects contained in well-defined contexts have *at least one* unique external identification (i.e. reference).
- The operations context union, intersection, and difference preserve the well-definedness of contexts. This ensures that unique external identification of objects is preserved, after applying the above operations on contexts.

Currently, we investigate additional properties of our operations. Further research includes extending our set of basic operations with searching operations, and developing set of generic commands based on our basic operations.

References

[1] Serge Abiteboul and Anthony Bonner. Objects and Views. In *Proceedings of ACM-SIGMOD Conference*, pages 238--247, February 1991.

[2] F. Bancilhon and N. Spyratos. Update Semantics of Relational Views. *ACM Transactions on Database Systems*, December 1981.

[3] Brad Campbell and Joseph M. Goodman. HAM: A General Purpose Hypertext Abstruct Machine. *Communications of the ACM*, 31(7):856--861, July 1988.

[4] Wojciech Cellary and Genevieve Jomier. Consistency of Versions in Object-Oriented Databases. In *Proceedings of the 16th International Conference on Very Large Data Bases - VLDB'90*, pages 432--441, Brisbane, Australia, 1990.

[5] Hong-Tai Chou and Won Kim. A Unified Framework for Version Control in a CAD Environment. In *Proceedings of the 12th International Conference on Very Large Data Bases - VLDB'86*, pages 275--281, Kyoto, August 1986.

[6] N. Delisle and M. Schwartz. Contexts: A Partitioning Concept for Hypertext. *ACM Transactions on Office Information Systems*, 5(2):168--186, April 1987.

[7] Georg Gottlob, Paolo Paolini, and Roberto Zicari. Properties and Update Semantics of Consistent Views. *ACM Transactions on Database Systems*, 13(4):486--524, December 1988.

[8] Georg Gottlob, Michael Schrefl, and Brigitte Röck. Extending Object - Oriented Systems with Roles. *ACM Transactions on Information Systems*, 14(3):268--296, July 1996.

[9] Ramanathan V. Guha. *Contexts: A Formalization and Some Applications.* PhD thesis, Stanford Iniversity, 1991. Also published as Technical Report STAN-CS-91-1399-Thesis, and MCC Technical Report Number ACT-CYC-423-91.

[10] Garry Hendrix. Encoding Knowledge in Partitioned Networks. In Nicolas Findler, editor, *Associative Networks.* New York: Academic Press, 1979.

[11] Vipul Kashyap and Amit Sheth. Semantic and Schematic Similarities between Database Objects: A Context-Based Approach. *VLDB Journal*, 5(4):276--304, December 1996.

[12] Randy Katz. Towards a Unified Framework for Version Modeling in Engineering Databases. *ACM Computing Surveys*, 22(4):375--408, December 1990.

[13] Gerald Kotonya and Ian Sommerville. Requirements Engineering with Viewpoints. *Software Engineering Journal*, pages 5--19, January 1996.

[14] Stan Matwin and Miroslav Kubat. The role of Context in Concept Learning. In *Proceedings of the ICML-96, Workshop on Learning in Context-Sensitive Domains*, pages 1--5, Bari, Italy, July 1996.

[15] John McCarthy. Notes on Formalizing Context. In *Proc. IJCAI-93*, pages 555--560, Chambery, France, 1993.

[16] Ryszard Michalski. How to Learn Impressive Concepts: A Method Employing a Two-Tiered Knowledge Representation for Learning. In *Proceedings of the 4th International Workshope in Machine Learning*, pages 50--58, Irvine, CA, 1987.

[17] Renate Motschnig-Pirtik. An Integrated View on the Viewing Abstraction: Contexts and Perspectives in Software Develepment, AI, and Databases. *Journal of Systems Integration*, 5(1):23--60, April 1995.

[18] Renate Motschnig-Pirtik. Requirements and Comparison of View Mechanisms for Object-Oriented Databases. *Information Systems*, 21(3):229--252, 1996.

[19] John Mylopoulos and Renate Motschnig-Pirtik. Patritioning Information Bases with Contexts. In *Proc. of CoopIS'95*, pages 44--55, Vienna, Austria, 1995.

[20] John Mylopoulos and Renate Motschnig-Pirtik. Semantics, Features, and Applications of the Viewpoint Abstraction. In *Proc. of CAiSE'96*, pages 514--539, Heraklion, Greece, May 1996.

[21] Aris Ouksel and Channah Naiman. Coordinating Context Building in Heterogeneous Information Systems. *J. of Intelligent Inf. Systems*, 3(2):151--183, 1994.

[22] Joel Richardson and Peter Schwarz. Aspects: Extending Objects to Support Multiple, Independent Roles. In *Proceedings of ACM-SIGMOD Conference*, pages 298--307, Denver, Colorado, May 1991.

[23] Marc H. Scholl, Cristian Laasch, and Markus Tresch. Updatable Views in Object-Oriented Databases. In *Proc. of the 2nd Int. Conf. on Deductive and Object-Oriented Database Systems*, pages 189--207, Munich, December 1991.

[24] Edward Sciore. Object Specialization. *ACM Transactions on Information Systems*, 7(2):103--122, April 1989.

[25] Edward Sciore, Michael Siegel, and Arnon Rosenthal. Using Semantic Values to Facilitate Interoperability Among Heterogeneous Information Systems. *ACM Transactions on Database Systems*, 19(2):254--290, 1994.

[26] John J. Shilling and Peter F. Sweeney. Three Steps to Views: Extending the Object-Oriented Paradigm. In *Proceedings of of Object-Oriented Programming, Systems, Languages and Applications - OOPSLA*, pages 353--361, October 1989.

[27] Yuh-Ming Shyy and Stanley Y.W. Su. K: A High-level Knowledge Base Programming Language for Advanced Dadabase Applications. In *Proceedings of ACM-SIGMOD Conference*, pages 338--347, Denver, Colorado, May 1991.

[28] Luiz Fernando G. Soares, Noemi L. R. Rodriguez, and Marco A. Casanova. Nested Composite Nodes and Version Control in an Open Hypermedia System. *Information Systems*, 20(6):501--519, 1995.

[29] Manos Theodorakis, Anastasia Analyti, Panos Constantopoulos, and Nikos Spyratos. A Theory of Contexts in Information Bases. Technical Report 216, Institute of Computer Science Foundation for Research and Technology - Hellas, March 1998.

[30] Manos Theodorakis and Panos Constantopoulos. Context-Based Naming in Information Bases. *International Journal of Cooperative Information Systems*, 6(3 & 4):269--292, 1997.

[31] Peter Turney. Robust classification with context-sensitive features. In *Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE-93*, pages 268--276, Edinburgh, 1993. Scotland: Gordon and Breach.

A Operation algorithms

In this appendix, we give the detailed algorithms of the basic operations presented in section 3. These operation are distinguished into lookup operations, update operations, copy operations, combining and comparing operations. We also give the detailed algorithms of two auxiliary operations, which are not basic operations, but they are used in the algorithms of the basic operations.

To simplify notation, assume that for each operation p, which takes as input a reference r, there is another one with the same name p, which takes as input an object referenced by r. In the following, we denote by Obj the set of all objects, i.e. simple objects and contexts.

A.1 Lookup operations

Operation A.1 Lookup.
lookup(**Input** $r : \mathcal{R}$; **Output** $O : \mathcal{P}(Obj)$).

/* This operation takes as input a reference r and returns all objects with reference. */

1. If r starts with @ then
   O is the set of all objects o such that $r \in refs(o, IB)$
   else O is the set of all objects o such that $r \in refs(o, \mathtt{CC})$.
2. End.

Operation A.2 Lookup one.
lookupOne(**Input** $r : \mathcal{R}$; **Output** $o : Obj$).

/* This operation takes as input a reference r and returns the object referenced by r, if it is just one. Otherwise, it returns ERROR. */

1. $O = lookup(r)$.
2. If the cardinality of the set O is one then
   return the element o of O
   else ERROR.
3. End.

Operation A.3 Set current context.
SCC(**Input** $r : \mathcal{R}$).

/* This operation takes as input a reference r to a context and sets CC to be this context. */

1. $c = lookup(r)$.
2. $\mathtt{CC} = c$; /* CC holds the current context of the user issuing the command */
3. End.

A.2 Update operations

Operation A.4 Create context.
createCxt(**Input** $l : \mathcal{L}$; **Output** $c : Cxt$).

/* This operation takes a lexicon l as input, and returns a new context c with lexicon l. */

1. Create a new context c such that $lex(c) = l$.
2. Set $src(c) = c$.
3. End.

Operation A.5 Insert an object into a context.
insert(**Input** $o : Obj, N : \mathcal{P}(\mathcal{N}), r : \mathcal{R}$).

/* This operation takes as input an object o, a set of names N, and a reference r to a context (call this context c). Then, it either inserts $(o{:}N)$ into the lexicon of c if object o is not contained in c, or adds the names contained in N to the c-names of o. */

1. $c = lookupOne(r)$.
2. If $(o{:}N') \in lex(c)$ then
   replace $(o{:}N')$ by $(o{:}N' \cup N)$ in $lex(c)$
   else add $(o{:}N)$ into $lex(c)$.
3. Set $src(c) = c$.
4. End.

Operation A.6 Delete an object from a context.
deleteObj(**Input** $o : Obj, r : \mathcal{R}$).

/* This operation takes as input an object o and a reference r to a context, and deletes the pair $(o{:}N)$ from the lexicon of that context. */

1. $c = lookupOne(r)$.
2. Delete the pair $(o{:}N)$ from $lex(c)$.
3. End.

Operation A.7 Delete an object name from a context.
deleteName(**Input** $o : Obj, n : \mathcal{N}, r : \mathcal{R}$).

/* This operation takes as input an object o, a name n, and a reference r to a context (call this context c), and deletes the name n from the c-names of o. */

1. $c = lookupOne(r)$.
2. If $(o{:}N) \in lex(c)$ then
   replace $(o{:}N)$ by $(o{:}N - \{n\})$ in $lex(c)$.
3. End.

A.3 Copy operations

Operation A.8 Copy context.
copyCxt(**Input** $r : \mathcal{R}$; **Output** $c' : Cxt$).

/* This operation takes as input a reference r to a context (call this context c) and returns a new context c' such that $lex(c') = lex(c)$. */

1. $c = lookupOne(r)$.
2. $c' = createCxt(lex(c))$.
3. End.

Operation A.9 Deep copy context.
deepCopyCxt(**Input** $r : \mathcal{R}$; **Output** $out\_c : Cxt$).

/* This operation takes as input a reference r to a context (call this context c) and returns a new context out_c. Context out_c contains the original simple objects of c, and deep copies of the contexts contained in c. */

1. $c = lookupOne(r)$.
2. Let $RecCxt$ be the contexts recursively contained in c.
3. $OrigCxt = RecCxt \cup \{c\}$.
4. $CopiedCxt = \emptyset$.
5. While $OrigCxt \neq \emptyset$ do
   (a) Find context $c' \in OrigCxt$ which is not contained in any other context in $OrigCxt$.
   (b) $c'' = copyCxt(c')$.
   (c) If $c = c'$ then $out\_c = c''$.
   (d) If c' is contained in some contexts in $CopiedCxt$ then
   replace c' with c'' in the lexicon of these contexts.
   (e) $OrigCxt = OrigCxt - \{c'\}$.
   (f) $CopiedCxt = CopiedCxt \cup \{c''\}$.
6. End.

A.4 Auxiliary operations

Operation A.10 lexUnion.
lexUnion(**Input** $O : \mathcal{P}(Obj)$; $l_1, l_2 : \mathcal{L}$; **Output** $l : \mathcal{L}$).

/* This operation takes as input a set of object O and two lexicons l_1 and l_2, and returns a lexicon l. Lexicon l contains objects of O that are also contained in l_1 or l_2. The names of each object o of l is the union of the names of o w.r.t. l_1 with the names of o w.r.t. l_2. */

1. Let $l = \emptyset$.
2. For each $o \in O$ do
   If $o \in objs(l_1) \cap objs(l_2)$ then $l = l \cup \{(o : names(o, l_1) \cup names(o, l_2))\}$
   else if $o \in objs(l_1)$ then $l = l \cup \{(o : names(o, l_1))\}$
   else $l = l \cup \{(o : names(o, l_2))\}$
3. End.

Operation A.11 lexIntersection.
lexIntersection(**Input** $O : \mathcal{P}(Obj)$; $l_1, l_2 : \mathcal{L}$; **Output** $l : \mathcal{L}$).

/* This operation takes as input a set of object O and two lexicons l_1 and l_2, and returns a lexicon l. Lexicon l contains objects of O that are also contained in l_1 and l_2. The names of each object o of l is the intersection the l_1-names of o with the l_2-names of o. */

1. Let $l = \emptyset$.
2. For each $o \in O$ do
   If $o \in objs(l_1) \cap objs(l_2)$ then $l = l \cup \{(o : names(o, l_1) \cap names(o, l_2))\}$
3. End.

Operation A.12 Elimination.
elimObj(**Input** $O : \mathcal{P}(Obj); C : \mathcal{P}(Cxt)$).

/* This operation takes as input a set of objects O and set of contexts C, and works as follows: The objects in O are eliminated from each context in C. If a context $c \in C$ is shared by another context $c' \in C$, then c is not eliminated from the objects of c'. */

1. While $C \neq \emptyset$ do
   (a) Find context $c \in C$ which does not contain any other context in C.

(b) For each $o \in objs(c)$ do
If $o \notin O$ then $deleteObj(o, c)$.

(c) If $c \neq \emptyset$ then $O = O \cup \{c\}$.

(d) $C = C - \{c'\}$.

2. End.

Operation A.13 Merge cleaned subcontexts.
merge(**Input** $l : \mathcal{L}$; **Output** $out\_l : \mathcal{L}$).

/* This operation takes as input a lexicon l and merges its subcontexts $c_1, \ldots, c_k$ with the same source context, i.e. $src(c_1) = \ldots = src(c_k)$. */

1. $c = createCxt(l)$.
2. Let $RecCxt$ be the contexts recursively contained in c.
3. $OrigCxt = RecCxt \cup \{c\}$.
4. While $OrigCxt \neq \emptyset$ do
   (a) Find context $c' \in OrigCxt$ which is not contained in any other context in $OrigCxt$.
   (b) Let $M = \{c_1, \ldots, c_k\} \subseteq Cxt$, where $\forall i \in \{1, \ldots, k\} : c_i \in objs(c') \wedge src(c_i) = src(c_1)$.
   (c) If $M \neq \emptyset$ then
      i. $N_m = names(c_1, c') \cup \ldots \cup names(c_k, c')$.
      ii. If $\exists c_i : c_i = src(c_i)$ then $c_m = c_i$
      else $c_m = createCxt(lex(c_1) \uplus \ldots \uplus lex(c_k))$. /* Merges the lexicon of contexts $c_1, \ldots, c_k$ */
      iii. Set $src(c_m) = src(c_1)$.
      iv. For $i \in \{1, \ldots, k\}$ do
      If $c_i \neq src(c_i)$ then $deleteObj(c_i, c')$.
      v. $insert(c_m, N_m, c')$.
      vi. If $c_m \neq src(c_m)$ then
      $OrigCxt = OrigCxt \cup \{c_m\}$. /* $merge$ will be called for c_m as well */
   (d) $OrigCxt = OrigCxt - \{c'\}$.
5. $out\_l = lex(c)$.
6. End.

A.5 Combining and comparing operations

Operation A.14 Union ($\uplus$).
1. Lexicon Union
$\uplus$ (**Input** $l_1, l_2 : \mathcal{L}$; **Output** $out\_l : \mathcal{L}$)

/* This operation takes as input two lexicons and returns their union. */

1. $out\_l = lexUnion(objs(l_1) \cup objs(l_2), l_1, l_2)$.
2. $out\_l = merge(out\_l)$.
3. End.

2. Context-Lexicon Union
$\uplus$ (**Input** $r_1 : \mathcal{R}$, $l_2 : \mathcal{L}$; **Output** $out\_l : \mathcal{L}$)

/* This operation takes as input a reference r_1 to a context and a lexicon and returns the union between this context and this lexicon. */

1. $c_1 = lookupOne(r_1)$.
2. $l_1 = lex(c_1) \uplus \{ (c_1 : \{str(r_1)\}) \}$.
3. $out\_l = l_1 \uplus l_2$.
4. End.

3. Context Union
$\uplus$ (**Input** $r_1, r_2 : \mathcal{R}$; **Output** $out\_l : \mathcal{L}$)

/* This operation takes as input two references r_1 and r_2 to two contexts and returns the union of these contexts. */

1. $c_1 = lookupOne(r_1)$.
2. $c_2 = lookupOne(r_2)$.
3. $l_1 = lex(c_1) \uplus \{ (c_1 : \{str(r_1)\}) \}$.
4. $l_2 = lex(c_2) \uplus \{ (c_2 : \{str(r_2)\}) \}$.
5. $out\_l = l_1 \uplus l_2$.
6. End.

Operation A.15 Intersection ($\cap$).

1. Lexicon Intersection Times

$\cap$ (**Input** $l_1, l_2 : \mathcal{L}$; **Output** $out\_l : \mathcal{L}$)

/* This operation takes as input two lexicons and returns their intersection. */

1. $I = ComO(l_1, l_2)$.
2. $out\_l = lexIntersection(I, lex(l_1), lex(l_2))$.
3. $ComC = I \cap Cxt$. /* $ComC$ stands for Common Contexts */
4. Let $RecCxt$ be the contexts recursively contained in l_1 or l_2.
5. $OrigCxt = RecCxt - ComC$.
6. $CopiedCxt = \emptyset$.
7. While $OrigCxt \neq \emptyset$ do
   (a) Find context $c \in OrigCxt$ which is not contained in any other context in $OrigCxt$.
   /* c is contained either in l_1 or in l_2 */
   (b) $c' = copyCxt(c)$.
   (c) Set $src(c') = src(c)$.
   (d) If $c \in objs(l_1)$ then $out\_l = out\_l \uplus \{(c' : names(c, l_1))\}$
   else If $c \in objs(l_2)$ then $out\_l = out\_l \uplus \{(c' : names(c, l_2))\}$
   (e) If c is contained in some contexts in $CopiedCxt$
   then replace c with c' in the lexicon of these contexts.
   (f) $OrigCxt = OrigCxt - \{c\}$.
   (g) $CopiedCxt = CopiedCxt \cup \{c'\}$.
8. $elimObj(I, CopiedCxt)$.
9. $out\_l = merge(out\_l)$.
10. End.

2. Context-Lexicon Intersection Times

$\cap$ (**Input** $r_1 : \mathcal{R}$, $l_2 : \mathcal{L}$; **Output** $out\_l : \mathcal{L}$)

/* This operation takes a reference r_1 to a context and a lexicon and returns the intersection between this context and this lexicon. */

1. $c_1 = lookupOne(r_1)$.
2. $c_1' = createCxt(lex(c_1))$.
3. $l_1 = lex(c_1) \uplus \{(c_1' : \{str(r_1)\})\}$.
4. $out\_l = l_1 \cap l_2$.
5. End.

3. Context Intersection Times

$\cap$ (**Input** $r_1, r_2 : \mathcal{R}$; **Output** $out\_l : \mathcal{L}$)

/* This operation takes as input two references r_1 and r_2 to two contexts and returns the intersection of these contexts. */

1. $c_1 = lookupOne(r_1)$.
2. $c_2 = lookupOne(r_2)$.
3. $c_1' = createCxt(lex(c_1))$.

4. $c_2' = createCxt(lex(c_2))$.
5. $l_1 = lex(c_1) \uplus \{ (c_1' : \{str(r_1)\}) \}$.
6. $l_1 = lex(c_2) \uplus \{ (c_2' : \{str(r_2)\}) \}$.
7. $out\_l = l_1 \barwedge l_2$.
8. End.

4. Lexicon Intersection Plus
; $\barwedge$;(**Input** $l_1, l_2 : \mathcal{L}$; **Output** $out\_l : \mathcal{L}$)
The same as Lexicon Intersection Times except for Step 2:

2. $out\_l = lexUnion(I, lex(l_1), lex(l_2))$.

5. Context-Lexicon Intersection Plus
; $\barwedge$;(**Input** $r_1 : \mathcal{R},\ l_2 : \mathcal{L}$; **Output** $out\_l : \mathcal{L}$)
The same as Context-Lexicon Intersection Times except for Step 4:

4. $out\_l = l_1$; $\barwedge$;l_2.

6. Context Intersection Plus
; $\barwedge$;(**Input** $r_1, r_2 : \mathcal{R}$; **Output** $out\_l : \mathcal{L}$)
The same as Context Intersection Times except for Step 7:

7. $out\_l = l_1$; $\barwedge$;l_2.

Operation A.16 Difference ($\ominus$).
1. Lexicon Difference
$\ominus$ (**Input** $l_1, l_2 : \mathcal{L}$; **Output** $out\_l : \mathcal{L}$)

/* This operation takes as input two lexicons and returns their difference. */

1. Let $DifO = objs(c_1) - objs(c_2)$.
2. $out\_l = lexUnion(DifO, l_1, \emptyset)$.
3. Let $DifC = DifO \cap Cxt$.
4. Let $RecCxt$ be the contexts recursively contained in l_1.
5. $OrigCxt = RecCxt - DifC$.
6. $CopiedCxt = \emptyset$.
7. While $OrigCxt \neq \emptyset$ do
   (a) Find context $c \in OrigCxt$ which is not contained in any other context in $OrigCxt$.
   /* c is contained in both l_1 and l_2 */
   (b) $c' = copyCxt(c)$.
   (c) Set $src(c') = src(c)$.
   (d) $out\_l = out\_l \uplus \{(c' : names(c, l_1))\}$.
   (e) If c is contained in some contexts in $CopiedCxt$ then
   replace c with c' in the lexicon of these contexts.
   (f) $OrigCxt = OrigCxt - \{c\}$.
   (g) $CopiedCxt = CopiedCxt \cup \{c'\}$.
8. $elimObj(DifO, CopiedCxt)$.
9. $out\_l = merge(out\_l)$.
10. End.

2. Context-Lexicon Difference
$\ominus$ (**Input** $r_1 : \mathcal{R},\ r_2 : \mathcal{L}$; **Output** $out\_l : \mathcal{L}$)

/* This operation takes as input a reference to a context r_1 and a lexicon r_2 and returns the differene if this context and this lexicon. */

1. $c_1 = lookupOne(r_1)$.

2. $c_1' = createCxt(lex(c_1))$.
3. $l_1 = lex(c_1) \uplus \{ (c_1' : \{str(r_1)\}) \}$.
4. $out\_1 = l_1 \ominus l_2$.
5. End.

3. Context Difference

$\ominus$ (**Input** $r_1, r_2 : \mathcal{R}$; **Output** $out_l : \mathcal{L}$)

/* This operation takes as input two references r_1 and r_2 to two contexts and returns the difference of these two contexts. */

1. $c_1 = lookupOne(r_1)$.
2. $c_2 = lookupOne(r_2)$.
3. $c_1' = createCxt(lex(c_1))$.
4. $l_1 = lex(c_1) \uplus \{ (c_1' : \{str(r_1)\}) \}$.
5. $l_2 = lex(c_2) \uplus \{ (c_2 : \{str(r_2)\}) \}$.
6. $out\_1 = l_1 \ominus l_2$.
7. End.

B Proofs of propositions, lemmas, and theorems

In this appendix, we give the proofs of the propositions, lemmas, and theorems given in the paper.

We shall say that a context c is *cleaned* if $src(c) \neq c$.

Proof of Proposition 2.1

It follows easily from Definition 2.3 and the fact that all contexts contained in c satisfy the acyclicity property. □

Proof of Proposition 4.1

(1), (2), (3) **Commutativity**.

The reader can easily verify this property by looking at the code of the Union and Intersection operations.

(4) **Union Associativity**.

Let $l_1 = (A \uplus B) \uplus C$, and $l_2 = A \uplus (B \uplus C)$.
We shall prove that: $l_1 \sim l_2$, that is: $\forall o \in Obj, N \in \mathcal{P}(\mathcal{N})$:

$$(o:N) \in l_1 \Leftrightarrow (o:N) \in l_2 \ \vee \ (\ \exists o' : \ (o':N) \in l_2 \ \wedge \ o \sim o').$$

We will first prove the forward derivation. The backwards derivation is proved similarly.
Let A, B, C be lexicons. We distinguish the following cases:

1. o is a simple object.
2. o is a context.
   (a) o is a context such that $src(o) = o$.
      i. No merging takes place between o and other cleaned contexts during the computation of l_1.
      ii. o is produced by merging o with one or more cleaned contexts.
   (b) o is a cleaned context (i.e. $src(o) \neq o$).
      i. There is only one context o' with $src(o') = src(o)$ contained in the lexicons A, B, or C.
      ii. There exist more than one objects o_i with $src(o_i) = src(o)$ in the lexicons A, B, or C.

Cases 1, 2(a)i

1. If $o \in objs(C)$ and $o \notin objs(A \uplus B)$ then we have $o \notin objs(A)$ and $o \notin objs(B)$, and $N = C(o)$. Hence, $(o:N) \in B \uplus C$ and $(o:N) \in l_2$.
2. If $o \in objs(A \uplus B)$ and $o \notin objs(C)$ then we have that either $o \in objs(A)$, or $o \in objs(B)$, or both.
   (a) If $o \in objs(A)$ and $o \notin objs(B)$ then $N = A(o)$. Hence, $o \notin objs(B \uplus C)$ and because $(o:N) \in A$ we have $(o:N) \in l_2$.
   (b) If $o \in objs(B)$ and $o \notin objs(A)$ then similarly to the previous case we can prove that $N = B(o)$ and $(o:N) \in l_2$.

(c) If $o \in objs(A)$ and $o \in objs(B)$ then $N = A(o) \cup B(o)$. On the other hand, $(o\!:\!B(o)) \in B \uplus C$ and $(o\!:\!A(o) \cup B(o)) \in A \uplus (B \uplus C)$. Hence, $(o\!:\!N) \in l_2$.

3. If $o \in objs(A \uplus B)$ and $o \in objs(C)$ then similarly to the previous case we can prove that $N = A(o) \cup B(o) \cup C(o)$ and $(o\!:\!N) \in l_2$.

Case 2(a)ii

Without loss of generality, assume that there is context $o_1 \in objs(A)$ with $src(o_1) = src(o)$, $o \in objs(B)$, and there is no context $o_3 \in objs(C)$ with $src(o_3) = src(o)$ (the rest of the cases are proved similarly). Then, during the computation $A \uplus B$, the $merge$ operation (called at Step 2 of the Lexicon Union algorithm given in Appendix A) merges o_1 with o. The result of this merging is again the context o, but now o has names $A(o_1) \cup B(o)$. Note that as there is no context $o_3 \in objs(C)$ with $src(o_3) = src(o)$ no other merging will take place and thus, l_1 will contain o with names $N = A(o_1) \cup B(o)$.

On the other hand, $B \uplus C$ contains o with names $B(o)$. Then, the $merge$ operation (called at Step 2 of the Lexicon Union algorithm computing l_2) merges o_1 with o resulting again in the context o, but now with names $A(o_1) \cup B(o)$. Hence, $(o\!:\!N) \in l_2$.

Case 2(b)i

Without loss of generality, assume that o' is contained in only one of A, B, or C (call this lexicon D) and $o = o'$. Similarly to the previous cases we can prove that $N = D(o)$ and $(o\!:\!N) \in l_2$.

Case 2(b)ii

Without loss of generality, assume that there are contexts $o_1 \in objs(A)$ and $o_2 \in objs(B)$ such that $src(o_1) = src(o_2) = src(o)$, and there is no context $o_3 \in objs(C)$ with $src(o_3) = src(o)$. Then, $o \in objs(A \uplus B)$ and o is produced by merging o_1, o_2 through the $merge$ operation (called at Step 2 of the Lexicon Union algorithm computing $A \uplus B$). Hence, $N = A(o_1) \cup B(o_2)$.

On the other hand, note that $(o_2\!:\!B(o_2)) \in B \uplus C$. Therefore, there is a context o' such that $(o'\!:\!N) \in l_2$, which is produced by merging o_1, o_2 through the $merge$ operation (called during the computation of l_2). Obviously, $o \sim o'$.

Let A, B be lexicons, and C be a reference to a context (call this context c). Then, $l_1 = (A \uplus B) \uplus C = (A \uplus B) \uplus (lex(c) \uplus \{(c\!:\!str(C))\})$, and $l_2 = A \uplus (B \uplus C) = A \uplus (B \uplus (lex(c) \uplus \{(c\!:\!str(C))\}))$. As $lex(c) \uplus \{(c\!:\!str(C))\}$ is a lexicon and associativity holds among lexicons, it follows that associativity holds among A, B, C as well. Similarly, we can prove the associativity property in the case that any of A, B, or C is a context.

(5), (6) **Intersection Associativity**.

In the following, we will prove the property (6). We can prove the property (5) similarly.
Let $l_1 = (A \Cap B) \Cap C$, and $l_2 = A \Cap (B \Cap C)$.
We shall prove that: $l_1 \sim l_2$, that is: $\forall o \in Obj, N \in \mathcal{P}(\mathcal{N})$:

$$(o\!:\!N) \in l_1 \Leftrightarrow (o\!:\!N) \in l_2 \ \vee \ (\ \exists o' : \ (o'\!:\!N) \in l_2 \ \wedge \ o \sim o').$$

We will first prove the forward derivation. The backwards derivation is proved similarly.
Let A, B, C be lexicons. We distinguish the following cases:

1. o is a simple object.
2. o is a context.
   (a) o is a context such that $src(o) = o$.
      i. No merging takes place between o and other cleaned contexts during the computation of l_1.
   (b) o is a cleaned context (i.e. $src(o) \neq o$).
      i. There is only one context c contained in A, B, or C that recursively contains objects in $ComO(A \Cap B, C)$ and $src(c) = src(o)$.
      ii. There exist more than one contexts c_i that recursively contain objects in $ComO(A \Cap B, C)$ and $src(c) = src(o)$.

Cases 1, 2(a)i

As o is not a cleaned subcontext, $o \in ComO(A \Cap B, C)$ and $N = (A \Cap B)(o) \cup C(o)$. Therefore, $o \in objs(A \Cap B)$ and $o \in objs(C)$. From this it follows that $o \in ComO(A, B)$ and $(A \Cap B)(o) = A(o) \cup B(o)$. Thus, $o \in objs(A)$ and $o \in objs(B)$. It now easily follows that $(o\!:\!B(o) \cup C(o)) \in B \Cap C$ and thus $(o\!:\!A(o) \cup (B(o) \cup C(o))) \in l_2$. Hence, $(o\!:\!N) \in l_2$.

Case 2(b)i

Without loss of generality, assume that there is context $c \in objs(A)$ and $c \notin objs(B) \cup objs(C)$. Then, during the operation $A \Cap B$, a new cleaned context c' is produced in the Step 7b of Lexicon Intersection Algorithm by

copying context c. Then, the objects of c' which are not in I or which do not recursively contain objects in I are eliminated from c' through the operation $elimObj(ComO(A,B),\{c',\ldots\})$. Thus, $(c':A(c)) \in objs(A \uplus B)$. Similarly, during the operation $(A \uplus B) \uplus C$, the cleaned context o is produced by copying c'. Context o is cleaned through the operation $elimObj(ComO(A \uplus B, C),\{o,\ldots\})$. Note also that $N = A(c)$.

Similarly, on the other hand, during the operation $A \uplus (B \uplus C)$ a new cleaned context c'' is produced by copying c such that $(c'':A(c)) \in l_2$. Context c'' is cleaned through the operation $elimObj(ComO(A, B \uplus C),\{o,\ldots\})$. It can be easily proved $ComO(A \uplus B, C) = ComO(A, B \uplus C)$. Hence, $(c'':N) \in l_2$ and $o \sim c''$.

Case 2(b)ii

Without loss of generality, assume that there exist contexts $c_1 \in objs(A)$ and $c_2 \in objs(B)$ such that $src(c_1) = src(c_2) = src(o)$, and there is no context $c_3 \in objs(C)$ which recursively contain objects in $ComO(A \uplus B, C)$ and $src(c_3) = src(o)$. Then, during the operation $A \uplus B$, two new cleaned contexts, c_1' and c_2', are produced by copying contexts c_1 and c_2, respectively. Then, contexts c_1' and c_2' are cleaned through the operation $elimObj(ComO(A,B),\{c_1', c_2', \ldots\})$. It also holds that $src(c_1') = src(c_1)$ and $src(c_2') = src(c_2)$. As $src(c_1) = src(c_2)$ then $src(c_1') = src(c_2')$ and contexts c_1' and c_2' are merged through the $merge$ operation, and a new context c' is produced with names $A(c_1) \cup B(c_2)$. Then, during the operation $(A \uplus B) \uplus C$, the cleaned context o is produced by copying c'. Context c' is cleaned through the operation $elimObj(ComO(A \uplus B, C),\{o,\ldots\})$. Also, $N = A(c_1) \cup B(c_2)$.

On the other hand, during the operation $B \uplus C$, a new cleaned context c_2'' is produced by copying c_2 and having name $B(c_2)$. Context c_2'' is cleaned through the operation $elimObj(ComO(B,C),\{c_2'',\ldots\})$. Then, during the operation $A \uplus (B \uplus C)$, two new cleaned contexts, c_1'' and c_2''', are produced by copying contexts c_1 and c_2'', and $A(c_1)$ and $B(c_2)$, respectively. Then, contexts c_1'' and c_2''' are cleaned through the operation $elimObj(ComO(A, B \uplus C),\{c_1'', c_2''', \ldots\})$. As $src(c_1'') = src(c_2''')$, contexts c_1' and c_2''' are merged through the $merge$ operation and a new context c'' is produced with names $A(c_1) \cup B(c_2)$. As $ComO(A \uplus B, C) = ComO(A, B \uplus C)$, it follows that $(c'':N) \in l_2$ and $o \sim c''$.

Let A, B be lexicons, and C be a reference to a context (call this context c). Then, there are contexts c', c'' such that $l_1 = (A \uplus B) \uplus C = (A \uplus B) \uplus (lex(c) \uplus \{(c':str(C))\})$, and $l_2 = A \uplus (B \uplus C) = A \uplus (B \uplus (lex(c) \uplus \{(c'':str(C))\}))$. As $lex(c) \uplus \{(c':str(C))\}$ and $lex(c) \uplus \{(c'':str(C))\}$ are lexicons and associativity holds among lexicons, it follows that associativity holds among A, B, C as well.
Similarly, we can prove the associativity property in the case that any of A, B, or C is a context.

(5) **Distributivity**.

Let $l_1 = (A \uplus B) \uplus C$, and $l_2 = (A \uplus C) \uplus (B \uplus C)$.
We shall prove that: $l_1 \sim l_2$, that is: $\forall o \in Obj, N \in \mathcal{P}(\mathcal{N})$:

$$(o:N) \in l_1 \Leftrightarrow (o:N) \in l_2 \ \vee \ (\ \exists o' : \ (o':N) \in l_2 \ \wedge \ o \sim o').$$

We will first prove the forward derivation. The backwards derivation is proved similarly.
Let A, B, C be lexicons. We distinguish the following cases:

1. o is a simple object.
2. o is a context.
   (a) o is a context such that $src(o) = o$.
      i. No merging takes place between o and other cleaned contexts during the computation of l_1.
      ii. o is produced by merging o with one or more cleaned contexts.
   (b) o is a cleaned context (i.e. $src(o) \neq o$).

Cases 1, 2(a)i

Then, o is contained in either $A \uplus B$, or C, or both. Without loss of generality, assume that o is contained in $A \uplus B$, but not in C. Then, $o \in ComO(A,B)$ and $N = A(o) \cap B(o)$. Therefore, $o \in objs(A)$ and $o \in objs(B)$. Thus, $(o:A(o)) \in A \uplus C$ and $(o:B(o)) \in B \uplus C$. From this it follows that $o \in ComO(A \uplus C, B \uplus C)$ and $(o:(A \uplus C)(o) \cap (B \uplus C)(o)) \in l_2$. Hence $(o:A(o) \cup B(o)) \in l_2$.

Case 2(a)ii

Without loss of generality, assume that o is contained in $A \uplus B$, and there is a cleaned context c contained in C such that $src(c) = o$.

Then, on one hand, $o \in ComO(A,B)$ and $(A \uplus B)(o) = A(o) \cap B(o)$. Therefore, $o \in objs(A)$ and $o \in objs(B)$. The $merge$ operation (called during the computation of l_1) merges o with c resulting again in the context o, but now with names $(A \uplus B)(o) \cup C(c)$. Hence, $N = (A(o) \cap B(o)) \cup C(c)$.

On the other hand, the $merge$ operation (called during the computation of $A \uplus C$) merges o with c, resulting again in the context o, but now with names $A(o) \cup C(c)$. Similarly, the $merge$ operation (called during the computation of $B \uplus C$) merges o with c, resulting again in the context o, but now with names $B(o) \cup C(c)$. Therefore, $o \in ComO(A \uplus C, B \uplus C)$ and $(o{:}(A \uplus C)(o) \cap (B \uplus C)(o)) \in l_2$. That is $(o{:}(A(o) \cup C(c)) \cap (B(o) \cup C(c))) \in l_2$. Hence, since distributivity of set union and set intersection is hold, $(o{:}N) \in l_2$.

Case 2b

Without loss of generality, assume that there is a context c contained in $A \curlywedge B$ such that $src(c) = src(o)$, and there is a context c' contained in C such that $src(c') = src(o)$.

As o is contained in $A \curlywedge B$, assume that there are contexts c_1, c_2 contained in A, B, respectively, which both recursively contain objects in $I = ComO(A, B)$ and $src(c_1) = src(c_2) = src(o)$. Then, during the operation $A \curlywedge B$, two new cleaned contexts, c_1' and c_2' are produced in the Step 7b of Lexicon Intersection Algorithm by copying contexts c_1 and c_2, respectively. Then, the objects of c_1' and c_2' that are not in I or do not recursively contain objects in I are eliminated from these contexts through the operation $elimObj(I, \{c_1', c_2', \ldots\})$. As $src(c_1') = src(c_2') = src(o)$, contexts c_1' and c_2' are merged through the $merge$ operation, and the new context c is produced with names $A(c_1) \cap B(c_2)$. Then, during the operation $(A \curlywedge B) \uplus C$, contexts c, c' are merged through the $merge$ operation and the context o is produced with names $N = (A(c_1) \cap B(c_2)) \cup C(c')$.

On the other hand, during the operation $A \uplus C$, contexts c_1 and c' are merged through the $merge$ operation, and a new context c_1'' is produced with names $A(c_1) \cup C(c')$. Similarly, during the operation $B \uplus C$, contexts c_2 and c' are merged through the $merge$ operation, and a new context c_2'' is produced with names $B(c_2) \cup C(c')$. Note that contexts c_1'' and c_2'' recursively contain objects in I and thus they also recursively contain objects in $I' = ComO(A \uplus C, B \uplus C)$. Then, during the operation $(A \uplus C) \curlywedge (B \uplus C)$, contexts c_1'' and c_2'' are first cleaned through the operations $elimObj(I', \{c_1'', c_2'', \ldots\})$, and then merged through the $merge$ operation. This will produce a new context c'' with names $(A(c_1) \cup C(c')) \cap (B(c_2) \cup C(c'))$. Hence, $(c'{:}N) \in l_2$ and, since I' contain contexts equivalent to the contexts contained in $I \cup objs(C)$, $o \sim c''$.

Let A, B be lexicons, and C be a reference to a context (call this context c). Then, $l_1 = (A \curlywedge B) \uplus C = (A \curlywedge B) \uplus (lex(c) \uplus \{(c{:}str(C))\})$, and $l_2 = A \curlywedge (B \curlywedge C) = (A \curlywedge (lex(c) \uplus \{(c{:}str(C))\}) \curlywedge (B \curlywedge (lex(c) \uplus \{(c{:}str(C))\})$. As $lex(c) \uplus \{(c{:}str(C))\}$ is a lexicon, and associativity holds among lexicons, it follows that associativity holds among A, B, C as well.

Similarly, we can prove the associativity property in the case that any of A, B, or C is a context. □

Proof of Lemma 4.1

Let $l = l_1 \odot l_2$. We shall prove that l is an operational lexicon, that is:

1. We will first prove that l is a well-defined context.
   We will prove that for each object o of l there is a unique reference of o w.r.t. l. Let first o be a context, which comes from a root context c of l_1 or l_2 ("comes from" means that either (i) o is the context c, or (ii) o is the result of cleaning the context c, or (iii) o is the result of merging a context of l_1 with a context of l_2, one of which is c). Assume that c is a context of l_1 (proceed similarly if c is a context of l_2). From the definition of the Union, Intersection, and Difference operations, we have
   $$names(c, l_1) \quad \subseteq \quad names(o, l) \tag{3}$$
   As c is a root context of l_1, there is a name $n \in names(c, l_1)$ such that there is no name n w.r.t. l_2. Equation (3) implies that $n \in names(o, l)$. We will prove that n is a unique reference of o w.r.t. l. Assume that there is another object o' such that $n \in refs(o', l)$. Then, there is an object o'' contained in l_1 or l_2 such that o' comes from $o'' \neq o$. Then, $n \in names(o'', l_1)$ or $n \in names(o'', l_2)$. However, this is impossible because n is a unique name w.r.t. l_1 and there is no name n w.r.t. l_2.
   Any other object o of l comes from objects that are not root contexts, but they are recursively contained in a root context (call this context c). Since o is contained in l, there must be a context c' of l coming from c (this is because of the definition of the Union, Intersection and Difference operations). Since c is well-defined, c' is well-defined as well. Thus, there is a unique reference r of o w.r.t c'. As c is a root context, we proved above that there is n such that n is a unique name of c' w.r.t. l. Thus, $n.r$ is a unique reference of o w.r.t. l.
   We shall now prove that every nested subcontext of l satisfies the acyclicity property. As every nested subcontext of l_1, l_2 satisfies the acyclicity property, it can be easily seen that every nested subcontext of l satisfies the acyclicity property as well.
2. We will now prove that if c is a root contexts of l then $src(c)$ is well-defined.
   Let c be a root context of l. Then, c either (i) is a root context of l_1 or l_2, or (ii) is the result of cleaning a root context c' of l_1 or l_2, or (iii) is the result of merging a context c' of l_1 with a context c'' of l_2. For case (i), as l_1 and l_2 are operational contexts, $src(c)$ is a well-defined context. Similarly for case (ii), $src(c) = src(c')$, and

hence $src(c)$ is a well-defined context. For case (iii), c' or c'' should be root context. Thus, $src(c')$ or $src(c'')$ is a well-defined context. Therefore, $src(c) = src(c') = src(c'')$ is a well-defined context as well.

3. We will now prove that any object of l which is not a root context is recursively contained in a root context of l. Let o be an object of l which is not a root context. Then, o comes from an object o' of l_1 or l_2, which either (i) is a root context of l_1 or l_2, or (ii) is recursively contained in a root context of l_1 or l_2. Consider first case (i), and without loss of generality, assume that o' is a root context of l_1. As o is not a root context of l, o' is recursively contained in a root context c' of l_2. Let c be the context of l which comes from c'. Obviously, c is a root context of l and o is recursively contained in c. Consider now case (ii), and without loss of generality, assume that o is recursively contained in a root context c' of l_1. Let c be the context of l which comes from c'. Obviously, c is a root context of l and o is recursively contained in c. □

Proof of Theorem 4.1

From Lemma 4.1, the operation $l_i \odot_i l_{i+1}$ results in an operational context. Therefore, we can compute the sequence of operations $l_1 \odot_1 \ldots \odot_{k-1} l_k$ through a sequence of computations of the form $l \odot l'$, where l and l' are operational lexicons and satisfy the condition of Lemma 4.1. Thus, the sequence of operations $l_1 \odot_1 \ldots \odot_{k-1} l_k$ will result in an operational lexicon. □

Proof of Theorem 4.2

Note that for any $i \leq k$, $c_i \odot_i c_{i+1} = (lex(c_i) \uplus \{(c_i' : str(r_i))\}) \odot_i (lex(c_{i+1}) \uplus \{(c_{i+1}' : str(r_{i+1}))\})$, where c_i' and c_{i+1}' are determined according to the particular operation $\odot_i$ (see the definitions of the Union, Intersection, and Difference operations). Note also that as c_i is a well-defined lexicon, $l_i = lex(c_i) \uplus \{(c_i' : str(r_i))\}$ results in an operational lexicon with root context c_i'. As $str(r_i)$ is not a name of an object w.r.t. each lexicon $l_1, \ldots, l_{i-1}, l_{i+1}, \ldots, l_k$, all conditions of Theorem 4.1 are met and hence $r_1 \odot_1 \ldots \odot_{k-1} r_k$ results in a well-defined lexicon. □

C Operations on versioning

In this Appendix, we give the detailed algorithms of the operations presented in section 5.

Operation C.1 Check-out.
check-out(**Input** $r : \mathcal{R},\ n : \mathcal{N}$).

/* With this operation, a designer checks-out a new version of the version o (referred to as r) from the public workspace into his/her private workspace. The new version is a copy of o and is given the name n w.r.t. the private workspace. This operation also copies the history context that contains o from the public to the private workspace. */

1. $o = lookupOne(r)$.
2. $o\_copy = copy(o)$.
3. $hc = whereContainedIn(o, HISTORY)$.
4. $hc\_copy = copy(hc)$.
5. $insert(o\_copy, \{n\}, hc\_copy)$.
6. $updateHistory(o\_copy, hc\_copy)$.
7. If $o \in objs(\mathtt{CC})$ then $deleteObj(o, \mathtt{CC})$.
8. $insert(o\_copy, \{n\}, \mathtt{CC})$.
9. End.

Operation C.2 Check-in.
check-in(**Input** $r, h : \mathcal{R},\ n : \mathcal{N}$).

/* With this operation, a designer checks-in his own new version (referred to as r) into the public workspace with a name n. This operation also inserts the new version into the history context referred to as h and will update the version history hierarchy. */

1. $o = lookup(r)$.
2. $hc = lookup(h)$.
3. $v = copy(o)$.
4. $insert(v, \{n\}, hc)$.

5. $updateHistory(ver\_o, hc)$.
6. End.

Operation C.3 Export to group.
export(**Input** $r_1, r_2 : \mathcal{R},\ n : \mathcal{N}$).

/* With this operation, the designer creates a context c whose lexicon is the union of the lexicons of the context referenced by r_1, and the context referenced by r_2 (this is the context c_2). Then, it creates a link from the last edited version $curr$ (that is, the one named $Current$) to the context c_2. Context c_2 is assigned two names w.r.t. c: $Current$ and `Username`. Finally, it copies the context c into the group workspace, under the name n. */

1. $c_1 = lookupOne(r_1)$.
2. $c_2 = lookupOne(r_2)$.
3. $c = createCxt(lex(c_1) \uplus r_2)$.
4. $SCC(c)$.
5. $curr = lookupOne(Current)$.
6. $deleteName(curr, Current, c)$.
7. $insert(c_2, \{Current, \texttt{Username}\}, c)$.
8. $updateHistory(c_2, curr)$.
9. $insert(c, \{n\}, GROUP)$.
10. End.

Operation C.4 Import from group.
import(**Input** $r : \mathcal{R},\ n : \mathcal{N}$).

/* With this operation, a designer imports the context referred to by r from the group workspace into his/her private workspace. This context is finally deleted from the group workspace. */

1. $SCC(Group)$.
2. $c = lookupOne(r)$.
3. $insert(c, \{n\}, \texttt{Home})$.
4. $deleteObj(c, Group)$.
5. End.

Operation C.5 Update history.
updateHistory(**Input** $v : Obj,\ c : Cxt$).

/* This operation creates a link object (named "derived-from") from the version named $Current$ w.r.t. context c to the version v. Then moves the name $Current$ from the version currently named $Current$ to version v. */

1. $ccxt\_old = \texttt{CC}$.
2. $SCC(c)$.
3. $curr = lookupOne(Current)$.
4. Create a link object l from object $curr$ to object v.
5. Insert the pair $(l : \{derived\text{-}from\})$ in $lex(c)$.
6. $deleteName(curr, Current, c)$.
7. $addName(v, Current, c)$.
8. $SCC(ccxt\_old)$.
9. End.

The operation $copy(Input\ o : Obj;\ Output\ o' : Obj)$ calls $copyCxt(o, o')$ in the case that o is a context, or copies the simple object o to a new one o'.

D Notations and symbols

Tables 1, 2 collects all notations and symbols of this paper.

Set	Description
Cxt	set of contexts
$\mathcal{S}$	set of simple objects
Obj	set of objects ($Obj = \mathcal{S} \cup Cxt$)
$\mathcal{N}$	set of atomic names
$\mathcal{L}$	set of lexicons
$\mathcal{R}$	set of references

Table 1: Table of sets.

Function	Description
$objs(c)$	returns the objects of lexicon or context c
$names(o, c)$ $c\text{-}names(o)$	returns the set of names that the object o has w.r.t. the lexicon or context c
$refs(o, c)$	returns the set of references that the object o has w.r.t. the lexicon or context c (Definition 2.3)
$str(r)$	converts a reference r to a name by replacing dots by underscores
$src(c)$	returns the source context of the context c
$ComO(l_1, l_2)$	returns the set of common objects of lexicons l_1 and l_2 which are not cleaned

Table 2: Table of functions.