A THEORY OF CONTEXTS IN INFORMATION BASES

Manos Theodorakis[1,2], Anastasia Analyti[1], Panos Constantopoulos[1,2] and Nicolas 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

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 context creation, update, copy, union, intersection, and difference. In particular, our operations of context union, intersection, and difference are different from these of set theory as they take into account the notion of context. However, they also satisfy the important properties of commutativity, associativity, and distributivity. 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.

Key words: Information Modeling, Contexts, Abstractions, Cooperative Environments, Views, Versions, Workspaces, Viewpoints

1. INTRODUCTION

The notion of context is a fundamental concern in cognitive psychology, linguistics, and computer science. Quite a number of formal or unformal expressions of a notion of context have appeared in several areas of computer science, such as artificial intelligence [11, 16, 10], software development [23, 9, 25, 27, 28, 13, 14], (multiple) databases [1, 7, 12, 22], machine learning [17, 34, 15], and knowledge representation [30, 20, 31, 32, 33, 6]. However, these are very diverse and serve different purposes. In artificial intelligence, the notion of context appears as means of partitioning knowledge into manageable sets [11], and as logical constructs that facilitate reasoning activities [16, 10]. In software development, views [24, 1], aspects [23], and roles [9, 25] appear for viewing data from different viewpoints, and workspaces are used for supporting cooperative work [13]. In machine learning, context is treated as environmental information for concept classification [17, 34, 15], and in multiple databases, as a collection of meta-attributes for capturing class semantics [12]. Finally, in knowledge representation, the notion of context has appeared as a viewpoint abstraction mechanism for partitioning an information base into possibly overlaping parts [30, 20, 31, 32, 33, 6]. Our objective is to establish a formal notion of context to support the development and effective use of large information bases in various application areas, especially in distributed, cooperative environments.

Our model has been mainly inspired by the work of Mylopoulos and Motschnig-Pitrik [20, 21], and incorporates previous work by Theodorakis and Constantopoulos [33].

In [20], Mylopoulos and Motschnig-Pitrik 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, whereby 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 rather arbitrary ways. In addition, operations among contexts, such as *union* (called *addition*) and *intersection* (called *product*),

are deprived of such useful properties as commutativity, associativity, and distributivity, and thus also can yield unexpected results.

In [33], 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, such as object names being ambiguous, excessively long, or unable to follow the changes of the environment of the object. However, that 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 the advantages of these previous two approaches and alleviate their shortcomings by introducing a more general and more complete framework for context.

A context is treated as a pair (cid, l) where cid is the context identifier and l is a lexicon i.e. a binding of names to objects. We note that an object is allowed to have more than one name, 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 *anonyms* (objects with no name w.r.t. a context). We also note that a context can belong to the objects of the lexicon of one or more other contexts. This allows for the nesting of contexts. 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 possesses a unique reference w.r.t. that context.

The present 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 all the benefits that these properties usually carry.

The paper is organized as follows: In section 2, the context construct for information bases is introduced. Sections 3 and 4 present the basic operations of our model and their properties, respectively. 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

In information modeling, a *context* is a higher-level conceptual entity that describes a group of conceptual entities from a particular standpoint [18]. 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 [24, 1, 19], or in a relational database [8, 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, 13]. Therefore, a multiversion object can be seen as a context in which the particular versions are contained.
- *Configurations*: A configuration is the binding between a version of a composite object and the particular versions of its components [13]. Therefore, a configuration of a composite object can be seen as a context containing a particular set of versions of its components.
- *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 [13]. Therefore, a workspace can be seen as a context in which the objects are viewed according to the responsibilities of the persons involved.

An information base can be considered as a repository of objects. Objects represent atomic or collective real world entities, attributes, (binary) relationships, or primitive values. We denote by Obj the set of all objects.

Contexts are taken as a special kind of objects that represent real world reference environments such as partitions, viewpoints, situations, or workspaces. We shall call all objects which are not contexts, *simple objects*. Contexts allow us to focus on a set of objects of interest, as well as to name each of these objects using one or more convenient names. Informally, we think of a context as containing objects, each object being associated with a set of names.

Definition 1 (Context) *Contexts* are a special kind of objects which can be thought of as containing objects, each object being associated with a set of names. Let Cxt be the set of all contexts. Then, $Cxt \subseteq Obj$. $\diamond$

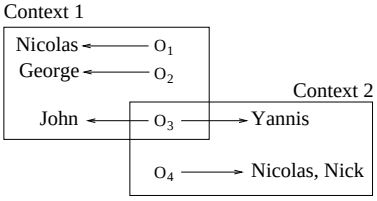


Fig. 1: The notion of context.

For example, Figure 1 illustrates two contexts, `Context_1` and `Context_2`, which represent the environment of two companies. The employees of those companies are represented by objects o_1 to o_4. `Context_1` contains the objects o_1, o_2, and o_3, and associates them with names `Nicolas`, `George`, and `John`, respectively. `Context_2` contains the objects o_3 and o_4, and associates them with names `Yannis`, and `Nicolas` or `Nick`, respectively. The employee represented by object o_3 works for both companies and is called *John* in the first company, whereas *Yannis* in the second.

In order to treat contexts more formally we need the concept of lexicon, i.e. a binding of names to objects in which an object may have zero, one or more names.

Definition 2 (Lexicon) Let $\mathcal{N}$ be the set of all atomic names and $\mathcal{P}(\mathcal{N})$ the power set of $\mathcal{N}$. A *lexicon* is a mapping l of the form:

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

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

Note that an object of a lexicon may be associated with 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 a lexicon: $l_1 = \{o_1 : \{\texttt{Panos}\}, o_2 : \{\texttt{head}\}, o_3 : \{\texttt{Manos}\}, o_4 : \{\texttt{Nicolas}, \texttt{Nick}\}\}$, where $objs(l_1) = \{o_1, o_2, o_3, o_4\}$. We depict this lexicon as follows:

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

As already mentioned, we think of a context as containing objects, each object being associated with a set of names. Formally, this is expressed by associating each context c with a lexicon 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 3 (Context lexicon) A *context lexicon* is a total function of the form:

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

which associates a context with a lexicon, which we shall call the *lexicon of* c. For each context c, objects of $lex(c)$ are also called *objects of* c, and denoted by $objs(c)$. That is, $objs(c) = objs(lex(c))$. $\diamond$

Let c be a 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 in c, or the *c-names* of o_i. The set N_i will also be denoted by $names(o_i, c)$.

A similar notation and terminology is used for a lexicon as well.

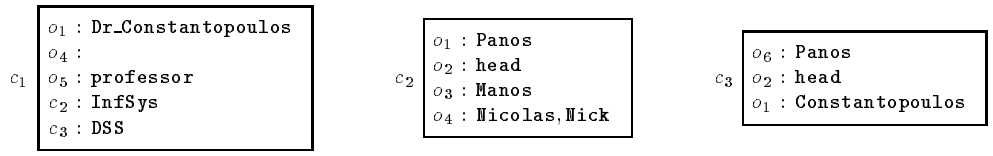


Fig. 2: Example of contexts.

As an example, consider a context c_1 which represents an institute (see Figure 2). 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 is known to exist within the context c_1 but we do not 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 2. Note that object o_1 has only one c_2-name (`Panos`), whereas object o_4 has two c_2-names (`Nicolas` and `Nick`). Also note that the same object can be contained in more than one context 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 in two different contexts (c_2 and c_3).

Recall that an object may represent real world attributes or binary relationships. We call these objects *link objects*. Link objects have a *source* and a *destination* object. This information is represented in our model by a triplet $< o_l, o_s, o_d >$, where o_l is a link object, and o_s and o_d are its source and destination, respectively. As any object, link objects are also defined w.r.t. a context. The link objects of a context c are determined by the function $links(c)$, which is defined as follows:

$$links(c) = \{< o_l, o_s, o_d > | o_l, o_s, o_d \in objs(c)\}.$$

Definition 4 (Recursive containment) We say that *a context* c *recursively contains object* o if either c contains o, or there is a context contained in c that recursively contains o. This is denoted by $o \in^\star c$. $\diamond$

For instance, in Figure 2, 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 by using one of its c-names, or by using a composite name, in case the object is contained in a nested subcontext of c. A *composite name* is a sequence of dot-separated names which are composed by taking into account the nesting of contexts, as shown in the following definition.

Definition 5 (Name paths of an object in a context) Let c be a context and let o be an object recursively contained in c. The set of all *name paths* of o in c, denoted by $npaths(o,c)$, is defined as follows:

$$npaths(o,c) = names(o,c) \ \cup \ compositeNames(o,c)$$
$$compositeNames(o,c) = \{r.n \mid \ \exists c' \in^\star c \ \wedge \ n \in names(o,c') \ \wedge \ r \in npaths(c',c)\}$$

The set of all name paths of all objects in all contexts is denoted by $\mathcal{NP}$. $\diamond$

For example (see Figure 2), we can refer to object o_1 of context c_1 either by using the name `Dr_Constantopoulos`, or by using the composite names `InfSys.Panos`, or `DSS.Constantopoulos`.

Note that a name path r in a context c may be *ambiguous*, in the sense that it may refer to more than one objects. That is, a name path r is ambiguous if there are two objects o, o' such that $r \in npaths(o,c) \cap npaths(o',c)$. It is possible for all name paths of an object o recursively contained in a context c to be ambiguous, i.e.,

$$npaths(o,c) \subseteq \bigcup_{o' \in^\star c \ \wedge \ o' \neq o} npaths(o',c).$$

For example, in Figure 3, within context c_1, the name paths of object o_2, i.e. `A`, `C`, and `D.F`, are all ambiguous, as $\texttt{A} \in names(o_1,c_1)$, $\texttt{C} \in names(o_3,c_1)$, and $\texttt{D.F} \in npaths(o_4,c_1)$.

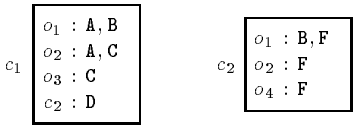


Fig. 3: Example of ambiguous name paths.

However, in practice, at least one unique name path of an object is required to be used for external identification. Thus, we distinguish an important class of contexts that possess at least one unique name path for every object and we call these contexts *well-defined*. An acyclicity contraint is also imposed.

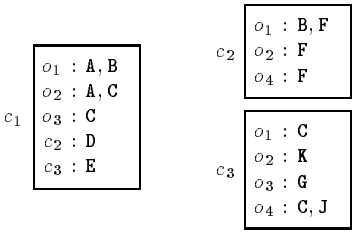
Definition 6 (Well-defined context) A lexicon l is called *well-defined* iff it satisfies the following conditions:

1. **Unique name path**: For every object recursively contained in l, there is a unique name path in l, i.e. for all objects o, o' of l:
   $o \neq o' \Rightarrow \exists r \in npaths(o,l) : \ \forall r' \in npaths(o',l), r \neq r'$.
2. **Acyclicity**: For every nested subcontext c' of l, it holds: $c' \notin^\star c'$.

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

In the example of Figure 2, contexts c_1, c_2, and c_3 are well-defined. Another example is shown in Figure 3, where context c_1 is not well-defined as there is at least an object recursively contained in c_1 with non unique name paths in c_1, (e.g. the object o_2 or the object o_3 or the object o_4). Context c_2 is not well-defined as well. On the other hand, if we add the context c_3 in the contents of c_1 (see Figure 4) then c_1 becomes well-defined. Note that, in Figure 4, c_1 is a well-defined context although its subcontexts c_2 and c_3 are not.

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



Context c_1 is well-defined whereas contexts c_2 and c_3 are not.

Fig. 4: Example of well-defined and non well-defined contexts.

Proposition 1 (Finite length and set of name paths) *Let c be a well-defined context, and let o be an object recursively contained in c. Then, the following hold:*

1. *Every name path of o in c has finite length, and*
2. *The set $npaths(o, c)$ is finite.*

Proof. It follows easily from Definition 5 and the fact that all contexts contained in c satisfy the acyclicity property. □

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 name paths. A name path to an object can be either *absolute*, i.e. in context IB, or *relative*. As a convention, if the name path is prefixed by @ then it is an *absolute name path*, otherwise it is a *relative name path*. Relative name paths 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 name path, we require 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.*

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

- *Synonyms*: Two different name paths w.r.t a context are called *synonymous*, if they refer to the same object. We view synonyms as alternative ways for externally identifying 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 2, the name paths `Nick` and `Nicolas` (which are the english and the french name of a person) in context c_2 are synonyms, as they refer to the same object o_4. Similarly, the name paths `Dr_Constantopoulos`, `InfSys.Panos`, and `DSS.Constantopoulos` in context c_1 are synonyms, as they refer to the same object o_1.
- *Homonyms*: Two different objects are called *homonymous* in a given context if they have a common name path in that context. If these two objects are recursively contained in a well-defined context c, then there exists a unique name path to each of these objects in c. Note that there always exists such a context, because IB recursively contains every object and it is a well-defined context, by assumption.
- *Anonyms*: An object o is called *anonymous* in 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 in that context, or we do not 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 $npaths(o, c') \neq \emptyset$, and IB is such a context. For example, in Figure 2, the object o_4 in context c_1 is anonymous.

3. OPERATIONS ON CONTEXTS

In this section we present six operations on contexts: lookup, browsing, update, copy, union, intersection and difference. The presentation is informal, and uses illustrative examples. Formal

definitions and computational algorithms are given in Appendix A.

Our definitions (both formal and informal) make use of two auxiliary concepts, namely, source context and derived context. Every context created by a single, explicit call of the operation *createCxt* is called a *source context*, otherwise it is called a *derived context*. Typically, a derived context is created from a single source context and possibly other derived contexts, using the operations that we define in this section.

In order to simplify the presentation, we introduce an auxiliary function $src(c)$ that returns the source of context c: if $src(c) = c'$ then c is a derived context and c' is its source, and if $src(c) = c$ then c is a source context.

With the above conventions in mind we now turn to the presentation of the operations.

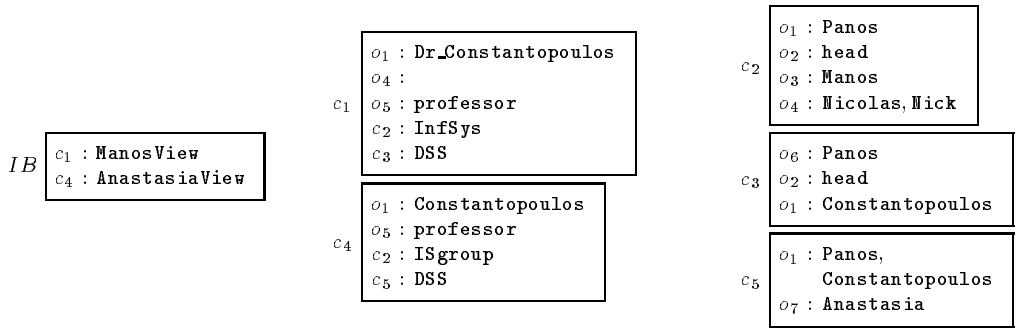


Fig. 5: An Information Base context.

Consider the Information Base illustrated in Figure 5. 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.

3.1. Lookup operations

- **lookup(r)**

  This operation takes as input a name path `r` and returns the set of objects o such that $\texttt{r} \in npaths(o, c)$, where: $c = IB$ if `r` is absolute, or $c =$`CC`, otherwise. $\diamond$

- **lookupOne(r)**

  This operation takes as input a name path `r` and returns an object o such that: $r \in npaths(o, c)$ and $|npaths(o, c)| = 1$, where: $c = IB$, if `r` is absolute, or $c =$`CC`, otherwise. $\diamond$

3.2. Browsing operations

- **Set current context: SCC(r)**

  This operation takes as input a name path `r`[†] to a context (call it c), and sets the current context to be the context c. $\diamond$

  Example: The operation $SCC(@.\texttt{ManosView.InfSys})$ sets the `CC` to c_2, and the operation $\overline{SCC(@)}$ sets the `CC` to IB.

[†] In all operations, if a name path is ambiguous, an error message is returned.

3.3. Update operations

- **Create context: createCxt(l)**

  This operation takes a lexicon l as input, and returns a context (call it c) such that $lex(c) = l$. Additionally, it sets $src(c) = c$. ◇

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

  $lex(c_{10}) = \begin{cases} o_1 : \texttt{Panos} \\ c_1 : \texttt{institute}. \end{cases}$

- **Insert an object into a context: insert($o, N, \texttt{r}$)**

  This operation takes as input an object o, a set of names N and a name path `r` to a context (call this context c), and 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. ◇

  Example: The operation $insert(o_{20}, \{\texttt{Nicolas}, \texttt{Nick}\}, @.\texttt{ManosView.DSS})$ results in the insertion of o_{20}:`Nicolas, Nick` into the context c_3:

  c_3

o_6 : Panos
o_2 : head
o_1 : Constantopoulos
o_{20} : Nicolas, Nick

  Note that synonyms or homonyms may occur as a result of an *insert* operation.

- **Delete an object from a context: deleteObj($o, \texttt{r}$)**

  This operation takes as input an object o and a name path `r` to a context, and deletes the pair $(o : N)$ from the lexicon of that context. ◇

- **Delete an object name from a context: deleteName($o, \texttt{n}, \texttt{r}$)**

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

Note that a *deleteName* operation may produce an anonym.

3.4. Copy operations

- **Copy context: copyCxt(r)**

  This operation takes as input a name path `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:
  $copyCxt(\texttt{r}) = createCxt(lex(c))$. ◇

  Example : The operation $copyCxt(@.\texttt{ManosView})$ returns a new context (call it c_{11}) shown as follows:

  c_{11}

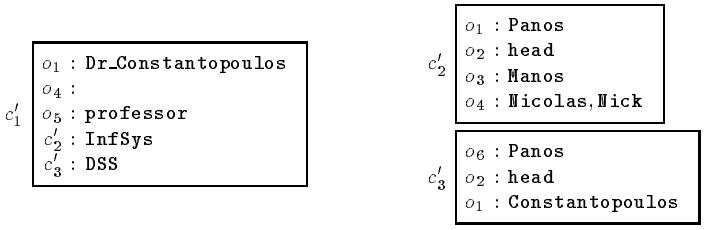
o_1 : Dr_Constantopoulos
o_4 :
o_5 : professor
c_2 : InfSys
c_3 : DSS

- **Deep copy context: deepCopyCxt(r)**

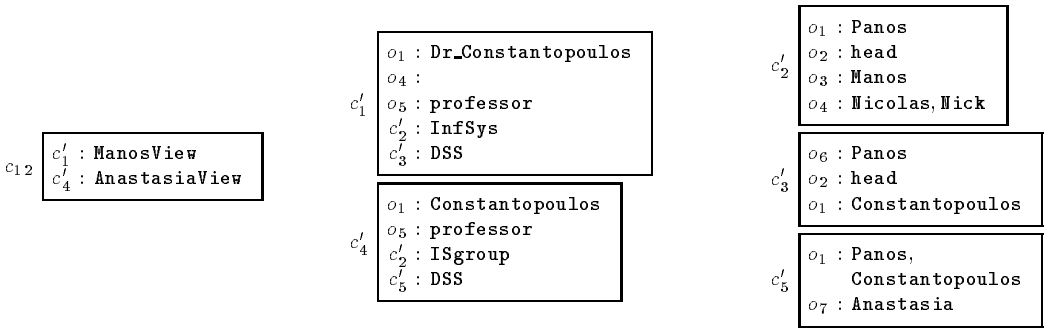
  This operation takes as input a name path `r` to a context (call this context c), and returns a new context (call it c') that contains the simple objects of c and deep copies of the contexts contained in c, (i.e. copies of those contexts together with their recursive expansions). In case a context c'' 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). $\diamond$

Example 1: The operation $deepCopyCxt(@.\texttt{ManosView})$ 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) as shown in the following picture:



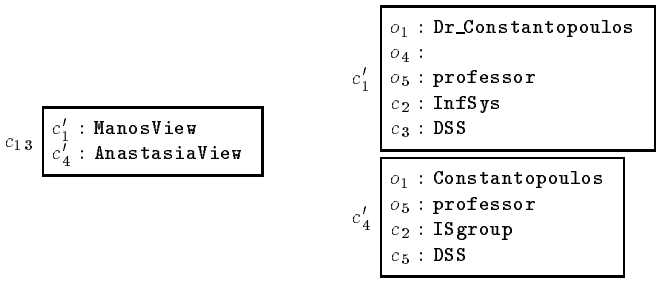
Example 2: The operation $deepCopyCxt(@)$ returns a new context (call it c_{12}) which contains deep copies of contexts c_1 and c_4 (call them c'_1 and c'_4). Context c'_1 contains copies of contexts c_2 and c_3 (call them c'_2 and , c'_3) whereas context c'_4 contains copies of contexts c_2 and c_5 (these are context c'_2 and c'_5). Note that although context c_2 is contained in both contexts c_1 and c_4 it is copied only once (context c'_2).



- **Deep copy context up to depth d: deepCopyCxt($\texttt{r}, d$)**

  This operation takes as input a name path $\texttt{r}$ to a context (call this context c) and an integer d, and returns a new context (call it c') such that: if $d = 1$ then $c' = copyCxt(r)$ otherwise if $d > 1$ then c' contains the simple objects of c and deep copies up to depth $d - 1$ of the contexts contained in c. In case a context c'' is contained in two or more contexts that are recursively contained in c, then c'' is copied only once at the shallowest level (i.e. c' does not recursively contain multiple copies of the same context). $\diamond$

  Example: The operation $deepCopyCxt(@, 2)$ returns a new context (call it c_{13}) which contains copies of contexts c_1 and c_4 (call them c'_1 and c'_4).



3.5. Union Operation

- **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:

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

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 $\mathtt{r_2}$. Intuitively, this adds a view over the objects of the combined lexicons as seen from c_2. We name this view $\mathtt{r_2}$ to record the fact that this view has been referred to by the user as $\mathtt{r_2}$[†]. Similarly, in Case 3 (where both inputs are contexts), contexts c_1 and c_2 are added to the output lexicon under the names $\mathtt{r_1}$ and $\mathtt{r_2}$, respectively.

It is important to note that of context union operation keeps track of the contexts the results come from, since the original contexts involved the operation are contained in the results as well (e.g., context c_2 in Case 2).

Example 1: Assume that `CC` has been set to c_1. Then, the operations $lex(\mathtt{InfSys}) \uplus lex(\mathtt{DSS})$ and $\mathtt{InfSys} \uplus \mathtt{DSS}$ return the lexicons l_1 and l_2, respectively, such that:

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

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 name paths (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 (contexts c_2 and c_3) over these objects, as seen from the Information Systems and DSS lab, respectively.

Example 2: Assume that the current context is the context IB, i.e. $\mathtt{CC} = IB$. The operation $\mathtt{ManosView} \uplus \mathtt{AnastasiaView}$ combines the views of Manos and Anastasia to get a wider view of the Institute, and returns the following lexicon:

$$l_3 = \begin{cases} o_1 : \mathtt{Dr\_Constantopoulos}, \mathtt{Constantopoulos} \\ o_4 : \\ o_5 : \mathtt{professor} \\ c_2 : \mathtt{InfSys}, \mathtt{ISgroup} \\ c_3 : \mathtt{DSS} \\ c_5 : \mathtt{DSS} \\ c_1 : \mathtt{ManosView} \\ c_4 : \mathtt{AnastasiaView} \end{cases}$$

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

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 name paths `ManosView.DSS` and `AnastasiaView.DSS`, respectively.

3.6. Intersection Operation

- **Intersection plus:** $r_1 \ \uplus_d \ r_2$

To define the intersection operation we need first to introduce the function $ComO$. Let l_1, l_2 be lexicons. We define $ComO(l_1, l_2) = objs(l_1) \cap objs(l_2)$. This operation takes as input two parameters r_1 and r_2, and returns a lexicon as a result. It also takes as input an integer d which is the depth of cleaning nested subcontexts from non-common objects. 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 $I = ComO(r_1, r_2)$):
   1. If $o \in I$ then $o \in objs(l)$ and $l(o) = r_1(o) \cup r_2(o)$.
   2. If $d > 1$ then
      If $o \notin I$ and o is a context recursively containing up to depth d an object of I then:
      (a) Make a deepcopy of o up to depth d (call it c), and set the source of its copy context to be equal to the source of the original context.
      (b) 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.
      (c) Add c to $objs(l)$ and define: $l(c) = \begin{cases} r_1(o), & \text{if } o \in objs(r_1) \\ r_2(o), & \text{if } o \in objs(r_2) \end{cases}$
   3. 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 $\mathtt{r_2}$ is a name path to a context (call this context c_2), then the operation returns a lexicon l such that:
   $$l = r_1 \ \uplus_d \ (lex(c_2) \ \uplus \ \{c_2' : \{str(\mathtt{r_2})\}\})$$
   where c_2' is a new context such that $lex(c_2') = lex(c_2)$.
3. If $\mathtt{r_1}$ and $\mathtt{r_2}$ are both name paths to contexts (call these contexts c_1 and c_2), then the operation returns a lexicon l such that:
   $$l = (lex(c_1) \uplus \{c_1' : \{str(\mathtt{r_1})\}\}) \ \uplus_d \ (lex(c_2) \uplus \{c_2' : \{str(\mathtt{r_2})\}\})$$
   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 $\mathtt{r_2}$. 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 $ComO(r_1, lex(c_2))$. The same holds in Case 3.

Parameter d determines how deep the nested subcontexts of the result will be cleaned from non common objects (i.e. objects not contained in $ComO(l_1, l_2)$). In fact, parameter d is used in practice to face up with the complexity of recursion (cleaning of nested subcontexts from non-common objects).

Parameter d increases the expressiveness of intersection in the following way: if d is equal to 1 the result contains all common objects. This is the most common type of intersection. However, if d is greater than 1, the result contains not only the common objects but also the subcontexts that contain these common objects in any depth less than or equal to d. For example, imagine two contexts: one containing the terminology used in Chemistry and the other the terminology used in Biology. Both contexts contain subcontexts representing departments of Chemistry and Biology, respectively, that contain the terminology used in

these departments. The intersection of these two contexts for $d = 1$ will result in the common terminology of Chemistry and Biology, as well as in their common departments. However, the same intersection for $d \geq 1$, say $d = 5$, will result not only in their common terminology and departments, but also in a mass of departments and subdepartments in depth 5 that use this common terminology. Note that departments and subdepartments contain only the common terminology, while the rest of the information has been removed from them.

In the rest of the thesis, whenever parameter d is not used it is assumed to be infinite.

Example 1: The operation $lex(\texttt{InfSys}) \uplus lex(\texttt{DSS})$, returns the lexicon:

$$l_4 = \begin{cases} o_1 : \texttt{Panos}, \\ \quad\ \texttt{Constantopoulos} \\ o_2 : \texttt{head}. \end{cases}$$

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.

Example 2: The operation $\texttt{InfSys} \uplus \texttt{DSS}$, returns the following lexicon:

$$l_5 = \begin{cases} o_1 : \texttt{Panos}, \texttt{Constantopoulos} \\ o_2 : \texttt{head} \\ c_2'' : \texttt{InfSys} \\ c_3'' : \texttt{DSS} \end{cases}$$

c_2'' { o_1 : Panos, o_2 : head }

c_3'' { o_2 : head, o_1 : Constantopoulos }

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 2b). Contexts c_2'' and c_3'' are added to the output lexicon in Step 3.

Example 3: The operation $\texttt{ManosView} \uplus \texttt{AnastasiaView}$ computes the commonalities of the views of Manos and Anastasia, and returns the following lexicon:

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

c_3' { o_1 : Constantopoulos }

c_5' { o_1 : Panos, Constantopoulos }

c_1' { o_1 : Dr_Constantopoulos, o_5 : professor, c_2 : InfSys, c_3' : DSS }

c_4' { o_1 : Constantopoulos, o_5 : professor, c_2 : ISgroup, c_5' : DSS }

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.

Example 4: The operation $lex(\texttt{ManosView}) \uplus_1 lex(\texttt{AnastasiaView})$ computes the commonalities of the the contents of views of Manos and Anastasia in depth 1, and returns the following lexicon:

$$l_7 = \begin{cases} o_1 : \texttt{Dr\_Constantopoulos}, \\ \quad\ \texttt{Constantopoulos} \\ o_5 : \texttt{professor} \\ c_2 : \texttt{InfSys}, \texttt{ISgroup} \end{cases}$$

Example 5: The operation `ManosView` $\uplus_1$ `AnastasiaView` computes the commonalities of the views of Manos and Anastasia in depth 1, and returns the following lexicon:

$$l_8 = \begin{cases} o_1 : \texttt{Dr\_Constantopoulos}, \\ \quad \texttt{Constantopoulos} \\ o_5 : \texttt{professor} \\ c_2 : \texttt{InfSys}, \texttt{ISgroup} \\ c_1' : \texttt{ManosView} \\ c_4' : \texttt{AnastasiaView} \end{cases}$$

c_1': o_1 : Dr_Constantopoulos; o_5 : professor; c_2 : InfSys

c_4': o_1 : Constantopoulos; o_5 : professor; c_2 : ISgroup

- **Intersection times:** $r_1 \uplus_d r_2$

It is defined as Intersection Plus except for the following Steps:

1(a). If $o \in I$ then $o \in objs(l)$ and $l(o) = r_1(o) \cap r_2(o)$.
2. $l = r_1 \uplus_d (lex(c_2) \ \uplus \ \{c_2' : \{str(\texttt{r}_2)\}\})$
3. $l = (lex(c_1) \uplus \{c_1' : \{str(\texttt{r}_1)\}\}) \uplus_d (lex(c_2) \uplus \{c_2' : \{str(\texttt{r}_2)\}\}) \diamond$

Motivation and use of parameter d are similar to these in the Intersection Plus operation.

Example 1: The operation $lex(\texttt{InfSys}) \ \uplus \ lex(\texttt{DSS})$, returns the lexicon:

$$l_4' = \begin{cases} o_1 : \\ o_2 : \texttt{head}. \end{cases}$$

Example 2: The operation `InfSys` $\uplus$ `DSS`, returns the following lexicon:

$$l_5' = \begin{cases} o_1 : \\ o_2 : \texttt{head} \\ c_2'' : \texttt{InfSys} \\ c_3'' : \texttt{DSS} \end{cases}$$

c_2'': o_1 : Panos; o_2 : head

c_3'': o_2 : head; o_1 : Constantopoulos

Example 3: The operation `ManosView` $\uplus$ `AnastasiaView` computes the commonalities of the views of Manos and Anastasia, and returns the following lexicon:

$$l_6' = \begin{cases} o_1 : \\ o_5 : \texttt{professor} \\ c_2 : \\ c_3' : \texttt{DSS} \\ c_5' : \texttt{DSS} \\ c_1' : \texttt{ManosView} \\ c_4' : \texttt{AnastasiaView} \end{cases}$$

c_3': o_1 : Constantopoulos

c_5': o_1 : Panos, Constantopoulos

c_1': o_1 : Dr_Constantopoulos; o_5 : professor; c_2 : InfSys; c_3' : DSS

c_4': o_1 : Constantopoulos; o_5 : professor; c_2 : ISgroup; c_5' : DSS

Note that the common objects o_1 and o_2 of lexicon l_6' are without any name. This means that although these two objects are known to both Manos and Anastasia, they use different set of names to describe them.

3.7. Difference Operation

- **Difference:** $r_1 \ \ominus_d \ r_2$

This operation takes as input two parameters r_1 and r_2, and returns a lexicon as a result. We distinguish four 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 $I = objs(r_1) \cap objs(r_2)$):

1. If $o \in D$ then $o \in objs(l)$ and $l(o) = r_1(o)$.
2. If $d > 1$ then
   If $o \in I$ and o is a context recursively containing up to depth d an object of D then:
   (a) Make a deepcopy of o up to depth d (call it c), and set the source of its newly derived context (copy) to be equal to the source of the original context.
   (b) Remove from c and from every context recursively contained in c, any simple object that is not in D.
   (c) Add c to $objs(l)$ and define: $l(c) = r_1(o)$.
3. No other object is in $objs(l)$.

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

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

Motivation and use of parameter d in the Difference operation are similar to these in the intersection operation.

Example 1: The operation $lex(\mathtt{InfSys}) \ominus lex(\mathtt{DSS})$, returns the lexicon:

$$l_9 = \begin{cases} o_3 : \mathtt{Manos} \\ o_4 : \mathtt{Nicolas}, \mathtt{Nick}. \end{cases}$$

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). Also, note that $I = \{o_1, o_2\}$. As I does not contain any context, Step 1(b) is not executed.

Example 2: The operation $\mathtt{ManosView} \ominus \mathtt{AnastasiaView}$ computes the differences between the views of Manos and Anastasia, and returns the lexicon:

$$l_{10} = \begin{cases} o_4 : \\ c_2''' : \mathtt{InfSys} \\ c_3 : \mathtt{DSS} \\ c_1''' : \mathtt{ManosView} \end{cases}$$

c_2''' | $o_4 : \mathtt{Nicolas}, \mathtt{Nick}$

c_1''' | $o_4 :$
 $c_2''' : \mathtt{InfSys}$
 $c_3 : \mathtt{DSS}$

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 execution 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 bear the equivalence relation defined below.

Definition 7 (Relation $\sim$) We define the relation $\sim$ between contexts or lexicons, as follows:

1. Let c and c' be contexts. Then
$$c \sim c' \Leftrightarrow (c = c') \ \vee \ (lex(c) \sim lex(c') \ \wedge \ src(c) = src(c'))$$

2. Let l and l' be lexicons. Then
$$\begin{aligned} l \sim l' \Leftrightarrow \quad & (\forall o \in \mathcal{S} : \ o{:}N \in l \Leftrightarrow o{:}N \in l') \wedge \\ & (\forall c \in Cxt : \\ & (c{:}N \in l \Rightarrow \ \exists c' : \ c'{:}N \in l' \ \wedge \ c \sim c') \wedge \\ & (c{:}N \in l' \Rightarrow \ \exists c' : \ c'{:}N \in l \ \wedge \ c \sim c')) \end{aligned}$$
where $\mathcal{S}$ denotes the set of simple objects. $\diamond$

It can be easily seen that the relation $\sim$ is *reflexive*, *symmetric*, and *transitive*, i.e. 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 one to ignore the order between two operands. Associativity allows one 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 an operand, so as to optimize further processing.

Proposition 2 *Let A, B, and C be references to contexts or lexicons. The following properties hold:*

1. **Commutativity***:*
   (1) $A \uplus B \sim B \uplus A$
   (2) $A \curlywedge B \sim B \curlywedge A$
   (3) $A \barwedge B \sim B \barwedge A$
2. **Associativity***:*
   (4) $(A \uplus B) \uplus C \sim A \uplus (B \uplus C)$
   (5) $(A \curlywedge B) \curlywedge C \sim A \curlywedge (B \curlywedge C)$
   (6) $(A \barwedge B) \barwedge C \sim A \barwedge (B \barwedge C)$
3. **Distributivity***:*
   (7) $(A \curlywedge B) \uplus C \sim (A \uplus C) \curlywedge (B \uplus C)$
   (8) $(A \uplus B) \curlywedge C \sim (A \curlywedge C) \uplus (B \curlywedge C)$

Proof. See Appendix B. □

For example (see Figure 5), assume the current context to be the context IB. The operation

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

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 it. Let l_1 be the intermediate lexicon returned by the operation `ManosView.InfSys` $\curlywedge$ `AnastasiaView.DSS` and let l_2 be the lexicon returned by the Operation 1. Then we have:

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

c_2' $\boxed{o_1 : \texttt{Panos}}$

c_5' $\boxed{o_1 : \texttt{Panos, Constantopoulos}}$

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

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 in subsection 3.5 and the detailed algorithms of the Operations A.18 and A.14 in Appendix A).

On the other hand, the operation

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

yields 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_4), and then computes the commonalities of these two wider views. Let l_5 be the lexicon returned by the Operation 2. Then we have:

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

$$l_5 = \begin{cases} o_1 : \texttt{Panos}, \\ \quad \texttt{Dr\_Constantopoulos} \\ o_4 : \\ o_5 : \texttt{professor} \\ c_2 : \texttt{ManosView\_InfSys}, \\ \quad \texttt{InfSys} \\ c_3 : \texttt{DSS} \\ c_5'' : \texttt{AnastasiaView\_DSS} \\ c_1 : \texttt{ManosView} \end{cases} \qquad c_5'' \boxed{\begin{array}{l} o_1 : \texttt{Panos}, \\ \quad \texttt{Constantopoulos} \end{array}}$$

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

We now define an important class of lexicons, called *operational lexicons*, which is closed over the operations Union, Intersection, and Difference. This closure property is expressed in Lemma 1 and Theorem 3. In the following, we shall call *root context* any context contained in a lexicon l (resp. context c) which is not recursively contained in any other context contained in l (resp. c).

Definition 8 (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 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 in l_1 (resp. l_2) such that there is no name n in l_2 (resp. l_1). Then the operation $l_1 \odot l_2$, where $\odot \in \{ \uplus , \oplus , \ominus \}$, results in an operational lexicon.*

Proof. See Appendix B. □

Theorem 3 (Closure of the operationality property: arbitrary number of lexicons) *Let $l_1, \ldots l_k$ be operational lexicons. If every root context c of l_i has a name n in l_i such that there is no name n in any 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 $\odot_i \in \{ \uplus , \oplus , \oplus , \ominus \}$ are executed in any order, results in an operational lexicon.*

Proof. From Lemma 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 1. Thus, the sequence of operations $l_1 \odot_1 \ldots \odot_{k-1} l_k$ will result in an operational lexicon. □

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

Theorem 4 (Closure of the well-definedness of contexts.) *Let $\mathtt{r_1}, \ldots, \mathtt{r_k}$ be name paths of the well-defined contexts $c_1, \ldots, c_k$. If $str(\mathtt{r_i})$ is not a name of an object in any of $c_1, \ldots, c_k$, then the sequence of operations $\mathtt{r_1} \odot_1 \ldots \odot_{k-1} \mathtt{r_k}$, where the operations $\odot_i \in \{ \uplus , \oplus , \oplus , \ominus \}$ are executed in any order, results in a well-defined lexicon.*

Proof. 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 3 are met and hence $r_1 \odot_1 \ldots \odot_{k-1} r_k$ results in a well-defined lexicon. □

The closure of well-definedness of context under the operations of context union, intersection, and difference ensures that unique external identification of objects and acyclicity are preserved,

after applying the above operation on contexts. Thus, no naming conficts and no cycles will appear in the resulting contexts. Operations on contexts defined in other works [20, 21] lack this ability. Thus, in these works, information from the original contexts may get lost in the result of an operation, since conflicts appear and their conflict resolution strategy may cause units to be inaccessible in the resulting context. Operations in these works do not satisfy the properties of commutativity, associativity, and distributivity.

5. APPLYING CONTEXT IN A COOPERATION ENVIRONMENT

As pointed out in [20], contexts can serve as the basis for supporting certain constructs arising in cooperative work environments such as workspaces, versions, and configurations. 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 [13, 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 absolute stability and cannot be updated or deleted. However, any worker can read this workspace, and can add new object versions to it.

The group workspace contains object versions that have reached reasonable stability, 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 and can be accessed only by a specific user. User workspaces contain temporary object versions which are expected to undergo a significant amount of update before reaching a reasonably stable 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 into and out of the public workspace through the *check-in* and *check-out* operations, and into and out of the group workspace through the *import* and *export* operations. A user checks a version out of 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 workspace and is linked (through 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 ISA-related as shown in Figure 6, thus forming a hierarchy of contexts.

A cooperation environment can be thought of as an Information Base (IB), containing six contexts: ATOMIC, CONFIG, HISTORY, PUBLIC, PRIVATE, and GROUP (see Figure 7). 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,

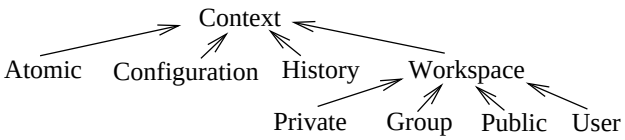


Fig. 6: Context types of the cooperation environment.

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.

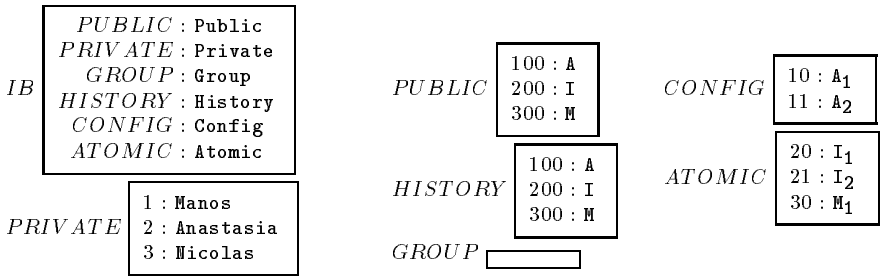


Fig. 7: Initial lexicons of IB and the six contexts of the cooperation scenario.

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 7 and 8. In Figure 8, we use the following conventions: A symbol of the form $o : \mathtt{n}_1, \mathtt{n}_2, \ldots$ denotes object o with names $\mathtt{n}_1, \mathtt{n}_2, \ldots$, e.g. $100 : \mathtt{A}$ 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.

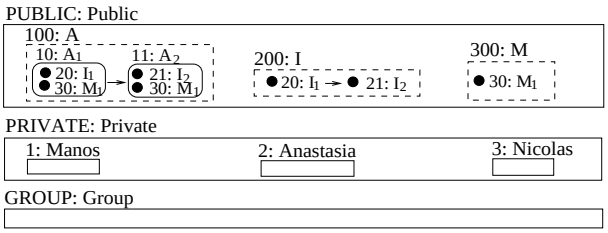


Fig. 8: Initial state of the cooperation scenario.

Specifically, the initial state of the Information Base is as follows (see Figures 7 and 8):

- 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 context 100 with name A, the history context for the introduction is context 200 with name I, and the history context for the main section is the context 300 with name M, as shown in Figures 7 and 8. The names A, I and M stand for "Article", "Introduction" and "Main section", respectively†. 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

† In practice one would use meaningful names instead of A, I and M, e.g. Article_on_Contexts instead of A.

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 `Nicolas`.
- 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 in the context PRIVATE, is his login name. Finally, the value of the variable `Home` is the global name path 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

For the revision of the article, each author has four commands at his disposal, as described below. These commands are high level operations, implemented using the context operations of the model. The full code of the operations is given in Appendix C. An example of their use is given in the following subsection.

- **check-out(`r`,`n`)**: This operation takes as input a name path `r` in 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').
  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'. ◇
- **check-in(`r`,`h`,`n`)**: This operation takes as input a name path `r` w.r.t. `CC`, a name path `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`. ◇
- **export(*exportedListOfContexts*, *exportedCxtName*)**: This operation takes as input a set of name paths *exportedListOfContexts* w.r.t. the `CC`, and a name *exportedCxtName*. Then, it does the following:
  1. Creates a context (call it c), which contains a copy of the context referenced by each name path contained in the input set.
  2. Inserts the context c into the group workspace, under the name *exportedCxtName*. ◇
- **import(`r`,`n`)**
  This operation takes as input a name path `r` w.r.t. the group workspace, and a name `n`. Then, it copies the context referenced by `r` from the group workspace into the `CC`, under the name `n`. ◇

5.3. A cooperation session

In this subsection, we present and discuss the commands issued by each author during a cooperation session. These commands are shown in Figure 9.

Commands by Manos

User Manos checks-out version $\mathtt{A_2}$ of the article, and copies it as version $\mathtt{A_3}$ to his home workspace (see Figure 10.(a)). This is done through the command *check-out*($\mathtt{A.A_2}$, $\mathtt{A_3}$). As the user wants to revise version $\mathtt{A_3}$, he focuses on context $\mathtt{A_3}$. This is done through the command $SCC(\mathtt{A_3})$. As he wants to revise the main section, he checks-out object $\mathtt{M_1}$ to his home workspace

1. Commands by user Manos.
   /* CC = 1, Home = @.Private.Manos, Username = Manos */
   (a) *check-out*(A.A_2, A_3)
   (b) *SCC*(A_3)
   (c) *check-out*(M.M_2, M_3)
   (d) ...
   (e) *SCC*(Home)
   (f) *export*({A,M}, Manos_changes)

2. Commands by user Anastasia.
   /* CC = 2, Home = @.Private.Anastasia, Username = Anastasia. */
   (a) *check-out*(A.A_2, A_3)
   (b) *SCC*(A_3)
   (c) *check-out*(I.I_2, I_3)
   (d) *check-out*(M.M_1, M_2)
   ...
   (e) *SCC*(Home)
   (f) *export*({A, I, M}, Anastasia_changes)

3. Commands by user Nicolas.
   /* CC = 3, Home = @.Private.Nicolas, Username = Nicolas */
   (a) *import*(Manos_changes, Manos)
   (b) *import*(Anastasia_changes, Anastasia)
   (c) history_A = *createCxt*(Manos.A ⊎ Anastasia.A))
   (d) history_I = *createCxt*({ } ⊎ Anastasia.I))
   (e) history_M = *createCxt*(Manos.M ⊎ Anastasia.M))
   (f) histCxt = *createCxt*({})
   (g) *insert*(history_A, {A}, histCxt)
   (h) *insert*(history_M, {M}, histCxt)
   (i) *insert*(histCxt, {Histories}, Home)
   (j) changesLex = *lex*(Manos.A) ⊎ *lex*(Anastasia.A) ⊎
   lex(Manos.M) ⊎ *lex*(Anastasia.M) ⊎ *lex*(Anastasia.I) ⊎
   {*lookupOne*(Manos):Manos, *lookupOne*(Anastasia):Anastasia} ⊖
   lex(@.Public.A) ⊖ *lex*(@.Public.I) ⊖ *lex*(@.Public.M)
   (k) *insert*(*createCxt*(changesLex), Changes, Home)
   (l) final_A = *createCxt*({*lookupOne*(Changes.I_3):I_3,
   lookupOne(Changes.Manos.M.M_2):M_2})
   (m) *insert*(final_A, A_3, Home)
   (n) *check-in*(A_3.I_3, I, I_3)
   (o) *check-in*(A_3.M_2, M, M_2)
   (p) *check-in*(A_3, A, A_3)

Fig. 9: User commands during a cooperation session.

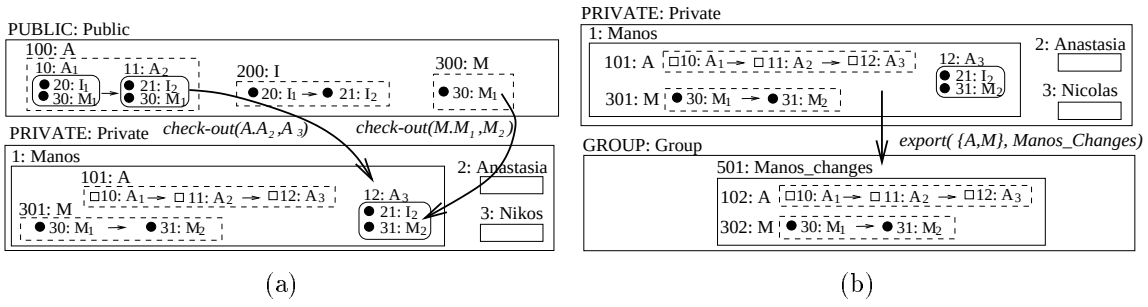


Fig. 10: Manos' interaction with the public, private and group workspaces.

(replacing the object $\mathtt{M_1}$ contained in context $\mathtt{A_3}$ by a new version of the main section, named $\mathtt{M_2}$, as shown in Figure 10.(a)). This is done through the operation *check-out*($\mathtt{M.M_1, M_2}$). The local editing of $\mathtt{M_2}$ is indicated by three dots in Figure 9.

After revision is completed, Manos needs to exchange information with the other authors for further revision. To this end, he needs to create the necessary environment which 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 uses the command *export*($\{\mathtt{A}, \mathtt{M}\}$, `Manos_changes`) to create a context named `Manos_changes` that contains copies of the history contexts of the edited objects, i.e. copies of the contexts 101 and 301 that represent the history of the article and its main section, respectively (see Figure 10.(b)). These contexts contain the original versions of the article and its main section, as well as their new versions created by user Manos.

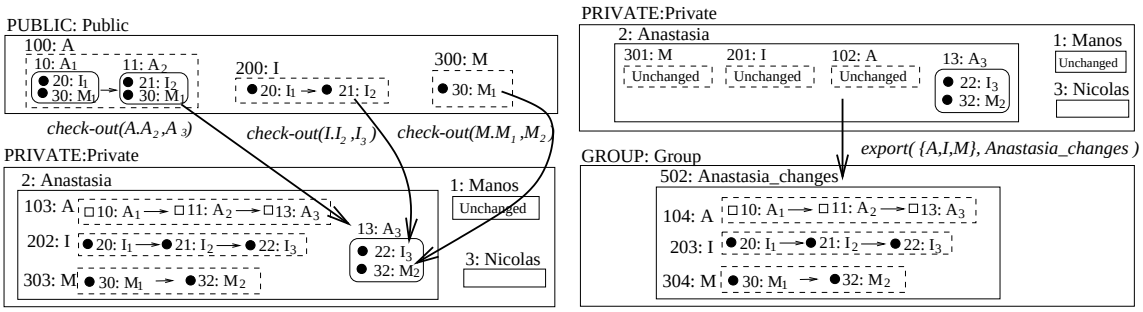


Fig. 11: Anastasia's interaction with the public, private, and group workspaces.

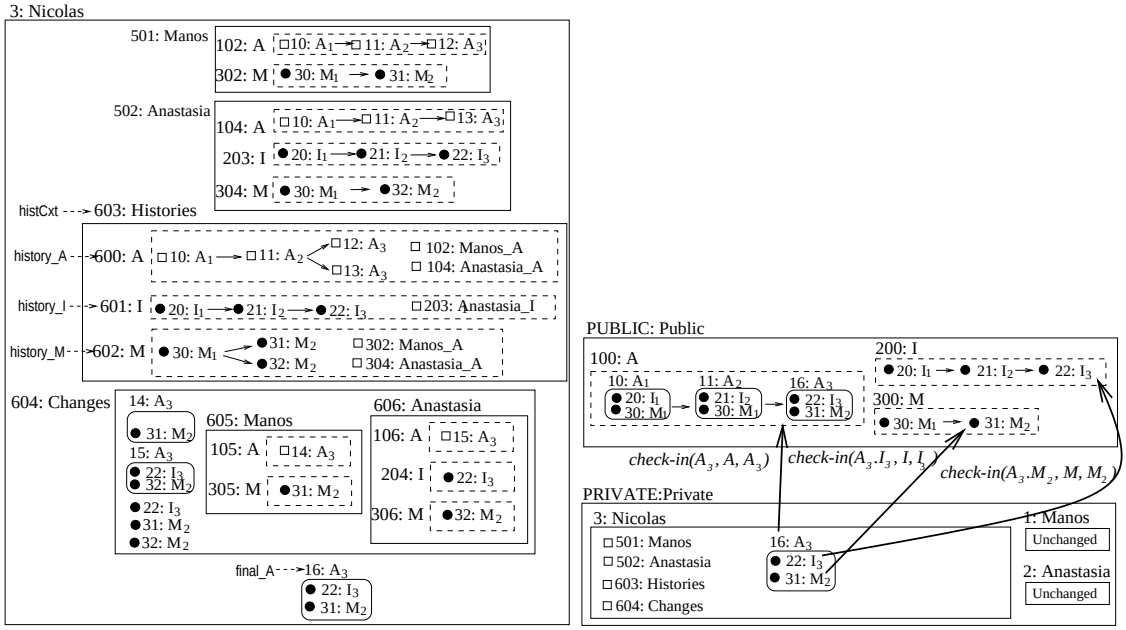
Commands by Anastasia

Concurrently, user Anastasia also checks-out version $\mathtt{A_2}$ of the article, and copies it as version $\mathtt{A_3}$ in her home workspace[†] (see Figure 11.(a)). As she wants to revise version $\mathtt{A_3}$, she focuses on context $\mathtt{A_3}$. She then checks-out $\mathtt{I_2}$ and $\mathtt{M_1}$, and copies them as $\mathtt{I_3}$ and $\mathtt{M_2}$ in her home workspace (see Figure 11.(a)). Anastasia can now start editing $\mathtt{I_3}$ and $\mathtt{M_1}$. Once editing is finished, she exports her modifications to the group workspace for further revision (see Figure 11.(b)). This is done through the command *export*($\{\mathtt{A}, \mathtt{I}, \mathtt{M}\}$, `Anastasia_changes`)

Commands by Nicolas

Finally, user Nicolas imports contexts `Manos_changes` and `Anastasia_changes`, which contain modifications made by Manos and Anastasia, to his home workspace under the names `Manos` and `Anastasia`, respectively (commands 3.(a) and 3.(b) in Figure 9). As Nicolas wants to unify these modifications, he issues the commands 3.(c) to 3.(i), shown in Figure 9, and he creates the context 603 (assigned the variable `histCxt`) with name `Histories` in his home workspace (see Figure 12.(a)). Context 603 contains three history contexts: (i) context 600 (assigned the variable `history_A`) with name `A`, which contains the whole history of the article after the modifications made on it by Manos and Anastasia; it also contains information about who made each modification (contexts 102 and 104); (ii) context 601 (assigned the variable `history_I`) with name `I`, which contains the history of the introduction after the modifications made on it by Anastasia (Manos did not modify the introduction); and (iii) context 602 (assigned the variable `history_M`) with name `M`, which contains the history of the main section after the modifications made on it by Manos and Anastasia; it also contains information about who made each modification (contexts 302 and 304). Then, he can see that versions 12 and 13 (with name $\mathtt{A_3}$) of the article are two

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



(a): The context of user Nicolas. (b): Nicolas' check-in to the public workspace.

Fig. 12: Nicolas' interaction with the public and private workspaces.

parallel versions of version 11 and that version 12 was created by Manos (`Manos_A`.A_3) and version 13 by Anastasia (`Anastasia_A`.A_3).

Nicolas then wants to isolate the changes made by Manos and Anastasia and get rid of the whole history of the article and its parts contained in the public workspace. Thus, he creates the context 604 with name `Changes`, by issuing the commands 3.(j) and 3.(k) shown in Figure 9. This context contains the modifications made by Manos and Anastasia (objects $14, 15, 22, 31$ and 32), as well as where these modifications appear within the structure of imported contexts 501 and 502; thus two new views of the structures of contexts 501 and 502 are created, which are contexts 605 and 606, respectively (see Figure 12.(a)).

Then, Nicolas studies modifications and creates the final version of article (command 3.(l) in Figure 9) composed by the version 22 (`Changes`.I_3) of the introduction made by Anastasia and the version 31 (`Changes.Manos.M`.M_2) of the main section made by Manos. This final version is checked in the public workspace through the commands 3.(n) to 3.(p) (see Figure 12.(b)).

We would like to stress that the purpose of the example presented here was 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 context operations of our model.

In a more complex environment, however, the users will most likely 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 context 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 those 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.

In [20], Mylopoulos and Motschnig-Pitrik 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, whereby 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 rather arbitrary ways. In addition, operations among contexts, such as *union* (called *addition*) and *intersection* (called *product*), are deprived of such useful properties as commutativity, associativity, and distributivity, and thus also can yield unexpected results. In [20], 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 results might seriously hinder the applicability of this otherwise appealing framework.

In [33], 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, such as object names being ambiguous, excessively long, or unable to follow the changes of the environment of the object. However, that approach imposes a hierarchical structure on contexts, i.e. a context may be contained in only one other context, which is rather restrictive.

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 [7]. HAM provides a set of context editing, context inquiry, and context attribute operations. 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. Inquiries on and attributes of contexts can be supported by our model, however they are outside of the scope of this paper.

In [29], 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, neither are generic operations on contexts provided.

The notion of context has also appeared in the area of heterogeneous databases [26, 22, 12]. 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 single 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. Contexts are treated as a special kind of objects which are 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 context operations. Further work includes extending the set of context operations with searching operations, and developing a set of generic high level commands based on the context operations. Another line of work addresses the incorporation of context mechanism, as prescribed by our model, in specific data models.

REFERENCES

[1] Serge Abiteboul and Anthony Bonner. Objects and Views. In *Proceedings of ACM-SIGMOD Conference*, pp. 238–247 (1991).

[2] F. Bancilhon and N. Spyratos. Update Semantics of Relational Views. *ACM Transactions on Database Systems*, **6**(4):557–575 (1981).

[3] Brad Campbell and Joseph M. Goodman. HAM: A General Purpose Hypertext Abstruct Machine. *Communications of the ACM*, **31**(7):856–861 (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*, pp. 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*, pp. 275–281, Kyoto (1986).

[6] Panos Constantopoulos and Yannis Tzitzikas. Context-Driven Information Base Update. In *Proc. of CAiSE'96*, Lecture Notes in CS 1080, pp. 319–344, Heraklion, Crete, Greece. Springer (1996).

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

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

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

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

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

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

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

[14] Gerald Kotonya and Ian Sommerville. Requirements Engineering with Viewpoints. *Software Engineering Journal*, **11**(1):5–18 (1996).

[15] 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*, pp. 1–5, Bari, Italy (1996).

[16] John McCarthy. Notes on Formalizing Context. In *Proc. IJCAI-93*, pp. 555–560, Chambery, France (1993).

[17] 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*, pp. 50–58, Irvine, CA (1987).

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

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

[20] John Mylopoulos and Renate Motschnig-Pitrik. Patritioning Information Bases with Contexts. In *Proc. of CoopIS'95*, pp. 44–55, Vienna, Austria (1995).

[21] John Mylopoulos and Renate Motschnig-Pitrik. Semantics, Features, and Applications of the Viewpoint Abstraction. In *Proc. of CAiSE'96*, pp. 514–539, Heraklion, Greece (1996).

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

[23] Joel Richardson and Peter Schwarz. Aspects: Extending Objects to Support Multiple, Independent Roles. In *Proceedings of ACM-SIGMOD Conference*, pp. 298–307, Denver, Colorado (1991).

[24] 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*, pp. 189–207, Munich (1991).

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

[26] 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).

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

[28] 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*, pp. 338–347, Denver, Colorado (1991).

[29] 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).

[30] Manos Theodorakis. *Contextualization: An Abstraction Mechanism for Information Modeling.* PhD thesis, Department of Computer Science, University of Crete (2001).

[31] Manos Theodorakis, Anastasia Analyti, Panos Constantopoulos, and Nicolas Spyratos. Contextualization as an Abstraction Mechanism for Conceptual Modeling. In *Proceedings of the 18th International Conference on Conceptual Modeling (ER'99)*, pp. 475–489, Paris, France (1999).

[32] Manos Theodorakis, Anastasia Analyti, Panos Constantopoulos, and Nikos Spyratos. Context in Information Bases. In *Proceedings of the 3rd International Conference on Cooperative Information Systems (CoopIS'98)*, pp. 260–270, New York City, NY, USA. IEEE Computer Society (1998).

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

[34] Peter Turney. Robust Classification with Context-Sensitive Features. In *Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE-93*, pp. 268–276, Edinburgh. Scotland: Gordon and Breach (1993).

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 $r : \mathcal{NP}$; **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 $r \in npaths(o, IB)$
   else O is the set of all objects o such that $r \in npaths(o, \texttt{CC})$.
2. End.

Operation A.2 Lookup one.
lookupOne(Input $r : \mathcal{NP}$; **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.

A.2. Browsing operations

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

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

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

A.3. 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 $o : Obj, N : \mathcal{P}(\mathcal{N}), r : \mathcal{NP}$).

/* 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 $o : Obj, r : \mathcal{NP}$).

/* 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 $o : Obj, n : \mathcal{N}, r : \mathcal{NP}$).

/* 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.4. Copy operations

Operation A.8 Copy context.
copyCxt(**Input** $r : \mathcal{NP}$; **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** $r : \mathcal{NP}$; **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.

Operation A.10 Deep copy context up to depth d.
deepCopyCxt(**Input** $r : \mathcal{NP}, d : Integer$; **Output** $out\_c : Cxt$).

/* This operation takes as input a reference r to a context (call this context c) and an integer d, 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 up to depth d. */

The same as Operation A.9 except for Step 2:

2 Let $RecCxt$ be the contexts recursively contained in c up to depth d.

A.5. Auxiliary operations

Operation A.11 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.12 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.13 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.14 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.

Operation A.15 Body of Intersection Plus.
BodyInterPlus(**Input** $l_1, l_2 : \mathcal{L},\ c_1, c_2 : Cxt,\ d : Integer$; **Output** $out\_l : \mathcal{L}$).

/* This operation takes as input two lexicons and an integer d and returns their intersection up to depth d. It also takes as input two contexts which are two views over the objects of the result (if some of these contexts is NIL it is not taken into account). */

1. $I = ComO(l_1, l_2)$.
2. $out\_l = lexUnion(I, lex(l_1), lex(l_2))$.
3. $ComC = I \cap Cxt$. /* $ComC$ stands for Common Contexts */
4. Let $RecCxt$ be the contexts recursively contained in l_1 or l_2 up to depth d, i.e., in depth $d_i < d$.
   /* contexts contained in l_1 or l_2 are in depth 1. */
5. If $c_1 \neq NIL$ then $RecCxt = RecCxt \cup \{c_1\}$.
6. If $c_2 \neq NIL$ then $RecCxt = RecCxt \cup \{c_2\}$.
7. $OrigCxt = RecCxt - ComC$.
8. $CopiedCxt = \emptyset$.
9. 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) $CopiedCxt = CopiedCxt \cup \{c'\}$.
10. $elimObj(I, CopiedCxt)$.
11. $out\_l = merge(out\_l)$.
12. End.

Operation A.16 Body of Intersection Times.
BodyInterTimes(**Input** $l_1, l_2 : \mathcal{L},\ c_1, c_2 : Cxt,\ d : Integer$; **Output** $out\_l : \mathcal{L}$).

/* The same as Lexicon Intersection Plus except for Step 2: */
2. $out\_l = lexIntersection(I, lex(l_1), lex(l_2))$.

Operation A.17 Body of Difference.
BodyDiff(**Input** $l_1, l_2 : \mathcal{L},\ c_1 : Cxt,\ d : Integer$; **Output** $out\_l : \mathcal{L}$).

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 up to depth d, i.e., in depth $d_i < d$.
   /* contexts contained in l_1 are in depth 1. */
5. If $c_1 \neq NIL$ then $RecCxt = RecCxt \cup \{c_1\}$.
6. $OrigCxt = RecCxt - DifC$.
7. $CopiedCxt = \emptyset$.
8. 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) $CopiedCxt = CopiedCxt \cup \{c'\}$.
9. $elimObj(DifO, CopiedCxt)$.
10. $out\_l = merge(out\_l)$.
11. End.

A.6. Union operation

Operation A.18 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** $r_1 : \mathcal{NP},\ l_2 : \mathcal{L}$; **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** $r_1, r_2 : \mathcal{NP}$; **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.

A.7. Intersection operation

Operation A.19 Intersection Plus($\uplus\!\!\!+$).

1. Lexicon Intersection Plus

$\uplus\!\!\!+$ (**Input** $l_1, l_2 : \mathcal{L}, d : Integer$; **Output** $out\_l : \mathcal{L}$)

/* This operation takes as input two lexicons and an integer d, and returns the intersection up to depth d of these two lexicon. */

1. $out\_l = BodyInterPlus(l_1, l_2, NIL, NIL, d)$.
2. End.

2. Context-Lexicon Intersection Plus

$\uplus\!\!\!+$ (**Input** $r_1 : \mathcal{NP},\ l_2 : \mathcal{L}, d : Integer$; **Output** $out\_l : \mathcal{L}$)

/* This operation takes a reference r_1 to a context and a lexicon, and an integer d, and returns the intersection up to depth d between this context and this lexicon. */

1. $c_1 = lookupOne(r_1)$.
2. $c_1' = copyCxt(c_1)$.
3. $l_1 = lex(c_1) \ \uplus \ \{c_1' : \{str(r_1)\}\}$.
4. $out\_l = BodyInterPlus(l_1, l_2, c_1', NIL, d)$.
5. End.

3. Context Intersection Plus

$\uplus\!\!\!+$ (**Input** $r_1, r_2 : \mathcal{NP}, d : Integer$; **Output** $out\_l : \mathcal{L}$)

/* This operation takes as input two references r_1 and r_2 to two contexts and an integer d, and returns the intersection up to depth d 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. $l_2 = lex(c_2) \uplus \{ c_2' : \{str(r_2)\} \}$.
7. $out\_l = BodyInterPlus(l_1, l_2, c_1', c_2', d)$.
8. End.

Operation A.20 Intersection Times($\barwedge$).

1. Lexicon Intersection Times

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

1. $out\_l = BodyInterTimes(l_1, l_2, NIL, NIL, d)$.
2. End.

2. Context-Lexicon Intersection Times

$\barwedge$ (**Input** $r_1 : \mathcal{NP},\ l_2 : \mathcal{L}, d : Integer$; **Output** $out\_l : \mathcal{L}$)

1. $c_1 = lookupOne(r_1)$.
2. $c_1' = copyCxt(c_1)$.
3. $l_1 = lex(c_1) \uplus \{c_1' : \{str(r_1)\}\}$.
4. $out\_l = BodyInterTimes(l_1, l_2, c_1', NIL, d)$.
5. End.

3. Context Intersection Times

$\barwedge$ (**Input** $r_1, r_2 : \mathcal{NP}, d : Integer$; **Output** $out\_l : \mathcal{L}$)

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. $l_2 = lex(c_2) \uplus \{ c_2' : \{str(r_2)\} \}$.
7. $out\_l = BodyInterTimes(l_1, l_2, c_1', c_2', d)$.
8. End.

A.8. Difference operation

Operation A.21 Difference ($\ominus$).

1. Lexicon Difference

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

/* This operation takes as input two lexicons and an integer d, and returns the difference up to depth d of these two lexicon. */

1. $out\_l = BodyDiff(l_1, l_2, NIL, d)$.
2. End.

2. Context-Lexicon Difference

$\ominus$ (**Input** $r_1 : \mathcal{NP},\ l_2 : \mathcal{L}, d : Integer$; **Output** $out\_l : \mathcal{L}$)

/* This operation takes a reference r_1 to a context and a lexicon, and an integer d, and returns the difference up to depth d between this context and this lexicon. */

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

3. Context Difference

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

/* This operation takes as input two references r_1 and r_2 to two contexts and an integer d, and returns the difference up to depth d of these contexts. */

1. $c_1 = lookupOne(r_1)$.
2. $c_2 = lookupOne(r_2)$.
3. $c_1' = copyCxt(c_1)$.
4. $c_2' = copyCxt(c_2)$.
5. $l_1 = lex(c_1) \uplus \{ c_1' : \{str(r_1)\} \}$.
6. $l_2 = lex(c_2) \uplus \{ c_2 : \{str(r_2)\} \}$.
7. $out\_l = BodyDiff(l_1, l_2, c_1', d)$.
8. 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 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 9b of the Body of Intersection Plus algorithm (Operation A.15) 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 9b 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

$$names(c, l_1) \subseteq names(o, l) \tag{3}$$

   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 $n \in npaths(o', l)$. 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.

□

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 $r : \mathcal{NP},\ n : \mathcal{N}$**).**

/* 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. `CurrentWorkingContext` = `CC`.
2. SCC(`@.Public`).
3. $o = lookupOne(r)$.
4. $o\_copy = copy(o)$.
5. $hc = whereContainedIn(o, HISTORY)$.
6. $hc\_copy = copy(hc)$.
7. $insert(hc\_copy, names(hc, HISTORY), Home)$.
8. $insert(o\_copy, \{n\}, hc\_copy)$.
9. $updateHistory(o\_copy, hc\_copy)$.
10. If $o \in objs$(`CurrentWorkingContext`) then $deleteObj(o,$ `CurrentWorkingContext`$)$.
11. $insert(o\_copy, \{n\},$ `CurrentWorkingContext`$)$.
12. `CC` = `CurrentWorkingContext`.
13. End.

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

/* 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 $exportedListOfContexts : \mathcal{P}(\mathcal{NP}),\ exportedCxtName : \mathcal{N}$**).**

/* With this operation, the designer creates a context c which contains a copy of the context referenced by each name path $\mathtt{r_i}$ contained in the input set $exportedListOfContexts$. Then, it inserts the context c into the group workspace, under the name $exportedCxtName$. */

1. $toBeExportedCxt = createCxt(\{\})$.
2. For each $r_i \in exportedListOfContexts$ do
3. $\quad insert(lookupOne(r_i), \{str(r_i)\}, toBeExportedCxt)$.
4. $insert(toBeExportedCxt, \{exportedCxtName\}, GROUP)$.
5. End.

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

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

1. $SCC(Group)$.
2. $c = lookupOne(r)$.
3. $insert(c, \{n\}, \mathtt{Home})$.
4. 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. $ccxt\_old = \mathtt{CC}$.
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'.