On the Complexities of Consistency Checking for Restricted UML Class Diagrams[1]

Ken Kaneiwa
National Institute of Information and Communications Technology
3-5 Hikaridai, Seika, Soraku, Kyoto 619-0289, Japan
kaneiwa@nict.go.jp

Ken Satoh
National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
ksatoh@nii.ac.jp

Abstract

Automatic debugging of UML class diagrams helps in the visual specification of software systems because users cannot detect errors in logical consistency easily. This study focuses on the *tractable* consistency checking of UML class diagrams. We accurately identify inconsistencies in these diagrams by translating them into first-order predicate logic that is generalized by counting quantifiers and classify their expressivities by eliminating certain components. We introduce optimized algorithms that compute the respective consistencies of class diagrams of different expressive powers in P, NP, PSPACE, or EXPTIME with respect to the size of the class diagrams. In particular, owing to the restrictions imposed on attribute value types, the complexities of consistency checking of class diagrams decrease from EXPTIME to P and PSPACE in two cases: (i) when the class diagrams contain disjointness constraints and overwriting/multiple inheritances and (ii) when the class diagrams contain both these components along with completeness constraints. Additionally, we confirm the existence of a restriction of class diagrams that prevents any logical inconsistency.

1 Introduction

The Unified Modeling Language (UML) [12, 7] is a standard modeling language; it is used as a visual tool for designing software systems. However, visualized descriptions make it difficult to determine consistency in formal semantics. In order to design UML diagrams, designers check not only for syntax errors but also for *logical inconsistency*, which may be present implicitly in the diagrams. Automatic detection of errors is very helpful for designers; for example, it enables them to revise erroneous portions of the UML diagrams by determining inconsistent classes or attributes. Moreover, in order to confirm the accuracy of debugging (soundness, completeness, and termination), it is necessary to computationally and theoretically develop a consistency checking algorithm.

[1]This paper is an extended version of [9], containing complete proofs as well as some additional definitions, theorems, lemmas, and examples.

Class diagrams, which are a type of UML diagrams, are employed to model concepts in static views. Many investigations on the consistency of class diagrams have been carried out. Evans [6] attempted a rigorous description of UML class diagrams by using OCL (Object Constraint Language) that enables reasoning on UML diagrams. Beckert, Keller, and Schmitt [2] defined a translation of UML class diagrams with OCL into FOPL (First-Order Predicate Logic). Further, Tsiolakis and Ehrig [14] analyzed the consistency of UML class and sequence diagrams by using attributed graph grammars. The use of OCL and other approaches provide rigorous semantics and logical reasoning on UML class diagrams; however, they do not theoretically analyze the worst-case complexity of consistency checking. A number of object-oriented models and their consistencies [11, 13] have been considered for developing software systems, but the models do not characterize the components of UML class diagrams; for example, the semantics of attribute multiplicities is not supported.

Berardi, Calvanese, and De Giacomo presented the correspondence between UML class diagrams and description logics (DLs), which enables us to utilize DL-based systems for reasoning on UML class diagrams [3]. Franconi and Ng implemented a concept modeling system called ICOM [8] using DLs. The cyclic expressions of class diagrams are represented by *general axioms* for DLs. For example, a class diagram is cyclic if a class C has an attribute and the attribute value type is defined by the same class. However, it is well known that reasoning on general axioms of the necessary DLs is exponential time hard [4]. Therefore, consistency checking of the class diagrams in DLs requires exponential time in the worst case.

In order to reduce the complexity, we consider restricted UML class diagrams obtained by deleting some components. A meaningful restriction of class diagrams is expected to avoid intractable reasoning, thus facilitating automatic debugging. This solution not only provides us with tractable consistency checking but also with a sound family of class diagrams (i.e., its consistency is theoretically guaranteed without checking).

The aim of this paper is to present optimized algorithms for testing the consistencies of restricted UML class diagrams, which are designed to be suitable for class diagrams of different expressive powers. The algorithms detect the logical inconsistency of class diagram formulation in FOPL that is generalized by counting quantifiers [10]. Although past approaches employ reasoning algorithms of DL and OCL, we develop consistency checking algorithms specifically for UML class diagrams. Our algorithms deal directly with the structure of UML class diagrams; hence, they have the following properties:

- Easy recognition of inconsistency triggers in the diagram structure, such as combinations of disjointness/completeness constraints, attribute multiplicities, and overwriting/multiple inheritances, and

- Refinement of the algorithms when the expressivity is changed due to the presence of the inconsistency triggers.

The inconsistency triggers captured by the diagram structure are used to restrict some relevant class diagram components in order to derive a classification of UML class diagrams. Since we can theoretically prove that no inconsistency arises for eliminated components, the algorithms are simplified and optimized for their respective expressivity.

The contributions of this paper are as follows:

1. *Inconsistency triggers:* We accurately identify inconsistency triggers that cause logical inconsistency among classes, attributes, and associations.

2. *Expressivity:* We classify the expressivity of UML class diagrams by deleting and adding certain inconsistency triggers.

3. *Algorithms and complexities:* We develop several consistency checking algorithms for class diagrams of different expressive powers and demonstrate that they compute the consistency of those class diagrams in P, NP, PSPACE, or EXPTIME with respect to the size of the class diagram.

4. *Tractable consistency checking in the optimized algorithms:* When the attribute value types are defined with restrictions in class diagrams, consistency checking is computable in P and PSPACE when the diagrams contain (i) disjointness constraints and overwriting/multiple inheritances and (ii) both these components along with completeness constraints, respectively.

5. *Consistent class diagrams:* We demonstrate that all class diagrams are consistent if their expressivities are restricted by deleting disjointness constraints and overwriting/multiple inheritances (but allowing attributes multiplicities and simple inheritances). Thus, we need not test the consistency of such less expressive class diagrams ($\mathcal{D}_0^-$ and $\mathcal{D}_{com}^-$).

The results of this study indicate two main advantages. First, the optimized algorithms support efficient reasoning for various expressive powers of class diagrams. In contrast, the DL formalisms do not provide optimized algorithms for the *restricted* UML class diagrams because general axioms of DLs require exponential time even if DLs are restricted [4]. Therefore, the classification of DLs does not fit into the classification of UML class diagrams[1]. Second, we analyze a meaningful restriction of UML class diagrams and confirm the existence of restricted class diagrams that permit attribute multiplicities, which cause no logical inconsistency.

This paper is arranged as follows. In Section 2 we discuss the translation of UML class diagrams into FOPL that is generalized by counting quantifiers. In Section 3, we clarify three inconsistency triggers in UML class diagrams. In Section 4, we develop an algorithm for testing the consistency. In Section 5, we modify the algorithm (proposed in Section 4) in order to provide optimized algorithms that are suitable for the expressive powers of class diagrams. In Section 6, we conclude our study and discuss our future work.

2 Class diagrams in FOPL with counting quantifiers

We define a translation of UML class diagrams into FOPL that is generalized by counting quantifiers. The reasons for encoding into FOPL with counting quantifiers are as follows. First, each UML class diagram should be defined by encoding it in a logical language because consistency checking is based on the syntax and semantics of encoded formulas. In other words, no consistency checking algorithm can operate on original diagrams without logical quantifiers and connectives, and the soundness and completeness of the algorithm cannot be guaranteed without formal semantics. Second, variables and quantifiers in FOPL lead to an explicit formulation that is useful in restricting/classifying the expressive powers. In

[1]Note that reasoning on general axioms becomes exponential time hard even if the small DL $\mathcal{AL}$ contains no disjunction, qualified existential restriction, and number restriction.

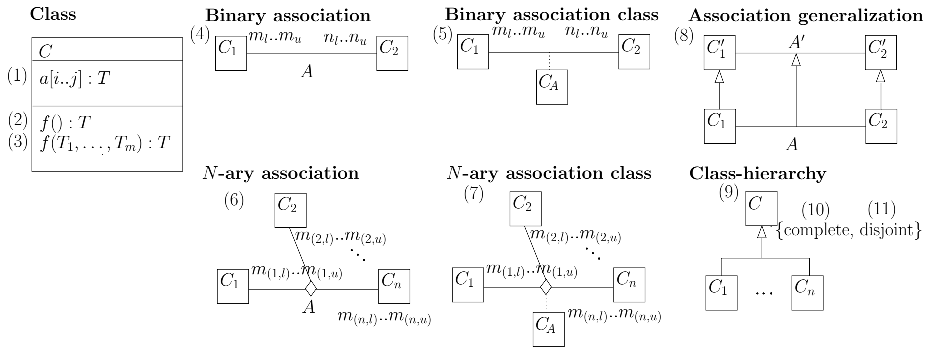

Figure 1: Components of UML class diagrams

contrast to encoding in FOPL, DL encoding [3] conceals the quantification of variables in expressions.

2.1 Classes

The alphabet of UML class diagrams consists of a set of class names, a set of attribute names, a set of operation names, a set of association names, and a set of datatype names. Let C, C', C_i be class names, a, a' attribute names, f, f' operation names, A, A' association names, and t, t', t_i datatype names. Let type T be either a class or a datatype. The leftmost figure in Figure 1 represents a class C with an attribute $a[i..j]\colon T$, a 0-ary operation $f()\colon T$, and an n-ary operation $f(T_1, \dots, T_n)\colon T$, where $[i..j]$ is the attribute multiplicity and T and $T_1, \dots, T_n$ are types. Any class C can be represented by the unary predicate C in FOPL. Let F_1 and F_2 be first-order formulas. We define the implication form $F_1 \to F_2$ as the universal closure $\forall x_1 \cdots \forall x_n.(F_1 \to F_2)$ where $x_1, \dots, x_n$ are all the free variables occurring in $F_1 \to F_2$. Let $F(x)$ denote a formula F in which the free variable x occurs. The counting quantifier formula $\exists_{\geq i} x.F(x)$ is true if at least i elements x satisfy $F(x)$, while the counting quantifier formula $\exists_{\leq i} x.F(x)$ is true if at most i elements x satisfy $F(x)$. The value type T and multiplicity $[i..j]$ of attribute a in class C are specified by the following implication forms:

(1) $C(x) \to (a(x,y) \to T(y))$ and $C(x) \to \exists_{\geq i} z.a(x,z) \land \exists_{\leq j} z.a(x,z)$

where a is a binary predicate and T is a unary predicate. The infinite multiplicity $[i..*]$ of attribute a in class C is translated into $C(x) \to \exists_{\geq i} z.a(x,z)$. That is, the unbounded upper limit '$*$' is not translated into any formula. The 0-ary operation $f()\colon T$ in class C is specified by the following implication forms:

(2) $C(x) \to (f(x,y) \to T(y))$ and $C(x) \to \exists_{\leq 1} z.f(x,z)$

where f is a binary predicate and T is a unary predicate. Moreover, the n-ary operation $f(T_1, \dots, T_n)\colon T$ in class C is specified by the following implication forms:

(3) $C(x) \to (f(x, y_1, \dots, y_n, z) \to T_1(y_1) \land \cdots \land T_n(y_n) \land T(z))$
$C(x) \to \exists_{\leq 1} z.f(x, y_1, \dots, y_n, z)$

where f is an $n+2$-ary predicate and $T_1, \dots, T_n$ and T are unary predicates.

2.2 Associations

We formalize the associations A that imply connections among classes $C_1, \ldots, C_n$ (as in (4) and (6) of Figure 1). The binary association A between two classes C_1 and C_2 and the multiplicities $m_l..m_u$ and $n_l..n_u$ are specified by the forms:

(4) $A(x_1, x_2) \rightarrow C_1(x_1) \wedge C_2(x_2)$
$C_1(x) \rightarrow \exists_{\geq n_l} x_2.A(x, x_2) \wedge \exists_{\leq n_u} x_2.A(x, x_2)$
$C_2(x) \rightarrow \exists_{\geq m_l} x_1.A(x_1, x) \wedge \exists_{\leq m_u} x_1.A(x_1, x)$

where A is a binary predicate and C_1, C_2 are unary predicates. In addition to the formulas, if an association is represented by a class, then the association class C_A is specified by supplementing the implication forms below:

(5) $A(x_1, x_2) \rightarrow (r_0(x_1, x_2, z) \rightarrow C_A(z))$
$A(x_1, x_2) \rightarrow \exists_{=1} z.r_0(x_1, x_2, z)$ and $\exists_{\leq 1} z.(r_0(x_1, x_2, z) \wedge C_A(z))$

where C_A is a unary predicate and r_0 is a ternary predicate. By extending the formulation of a binary association, the n-ary association A among classes $C_1, \ldots, C_n$ and their multiplicities "$m_{(1,l)}..m_{(1,u)}$", ..., "$m_{(n,l)}..m_{(n,u)}$" (as shown in (6) of Figure 1) are specified by the following implication forms:

(6) $A(x_1, \ldots, x_n) \rightarrow C_1(x_1) \wedge \cdots \wedge C_n(x_n)$
$C_k(x) \rightarrow \exists_{\geq m_{(1,l)}} x_1 \cdots \exists_{\geq m_{(k-1,l)}} x_{k-1} \exists_{\geq m_{(k+1,l)}} x_{k+1} \cdots \exists_{\geq m_{(n,l)}} x_n.A(x_1, \ldots, x_n)[x_k/x]$
$C_k(x) \rightarrow \exists_{\leq m_{(1,u)}} x_1 \cdots \exists_{\leq m_{(k-1,u)}} x_{k-1} \exists_{\leq m_{(k+1,u)}} x_{k+1} \cdots \exists_{\leq m_{(n,u)}} x_n.A(x_1, \ldots, x_n)[x_k/x]$

where A is an n-ary predicate and $[x_k/x]$ refers to the substitution of x_k with x. In addition, the association class C_A is specified by adding the implication forms below:

(7) $A(x_1, \ldots, x_n) \rightarrow (r_0(x_1, \ldots, x_n, z) \rightarrow C_A(z))$
$A(x_1, \ldots, x_n) \rightarrow \exists_{=1} z.r_0(x_1, \ldots, x_n, z)$ and $\exists_{\leq 1} z.(r_0(x_1, \ldots, x_n, z) \wedge C_A(z))$

where C_A is a unary predicate and r_0 is an $n+1$-ary predicate. Furthermore, we treat association generalization (not discussed in [3]) such that the binary association A' between classes C'_1 and C'_2 generalizes the binary association A between classes C_1 and C_2 (as in (8) of Figure 1). The generalization between binary associations A and A' is specified by the implication forms below:

(8) $A(x_1, x_2) \rightarrow A'(x_1, x_2)$, $C_1(x) \rightarrow C'_1(x)$, and $C_2(x) \rightarrow C'_2(x)$

where A, A' are binary predicates and C_1, C'_1, C_2, C'_2 are unary predicates. More universally, the generalization between n-ary associations A and A' is specified by the following implication forms:

(8)' $A(x_1, \ldots, x_n) \rightarrow A'(x_1, \ldots, x_n)$ and $C_1(x) \rightarrow C'_1(x)$, ..., $C_n(x) \rightarrow C'_n(x)$

where A, A' are n-ary predicates and C_i, C'_j are unary predicates.

2.3 Class hierarchies

We consider class hierarchies and disjointness/completeness constraints of the classes in hierarchies, as shown in (9), (10), and (11) of Figure 1. A class hierarchy (a class C generalizes classes $C_1, \ldots, C_n$) is specified by the implication forms below:

(9) $C_1(x) \rightarrow C(x), \ldots, C_n(x) \rightarrow C(x)$

where C and $C_1, \ldots, C_n$ are unary predicates. The completeness constraint $\{complete\}$ between a class C and classes $C_1, \ldots, C_n$ and the disjointness constraint $\{disjoint\}$ among classes $C_1, \ldots, C_n$ are specified by the implication forms:

(10) $C(x) \rightarrow C_1(x) \vee \cdots \vee C_n(x)$

(11) $C_i(x) \rightarrow \neg C_{i+1}(x) \wedge \cdots \wedge \neg C_n(x)$ for all $i \in \{1, \ldots, n-1\}$

respectively, where C and $C_1, \ldots, C_n$ are unary predicates.

Let D be a UML class diagram. $\mathcal{G}(D)$ is called the translation of D and denotes the set of implication forms obtained by the encoding of D in FOPL with counting quantifiers (using (1)–(11)). The translation into first-order logic is similar to and based on the study in references [2, 3].

In the encoding of UML class diagrams, no association roles and no aggregation between classes are considered. The reason behind this consideration is as follows: if association roles are encoded into first-order formulas, check the consistency where the equality of objects is interpreted for the multiplicities of association roles; this is more complicated than the equality of objects being interpreted for the multiplicities of attributes. For example, the encoded formulas of association roles with the multiplicities [3..∗] and [2..∗] impose the two conditions that there exist at least three objects and at least two objects, respectively. If the multiplicities are used for the identically named association roles in different places, it is necessary to check if there are common objects for the three and two objects, i.e., it is necessary to verify the formulas $\exists_{\geq 3} y.(r(y,x) \wedge C_1(y))$ and $\exists_{\geq 2} y.(r(y,x) \wedge C_2(y))$. The evaluation of any case of the equality of these objects essentially increases the complexity of consistency checking for role expressions. Moreover, we do not deal with aggregations and compositions; we consider them to be a particular type of association. Hence, there is no specific need to introduce the encoding of aggregations and compositions.

3 Inconsistencies in class diagrams

In this section, we analyze inconsistencies among classes, attributes, and associations in UML class diagrams. We first define the syntax errors of duplicate names and irrelevant attribute value types as described below.

Duplicate name errors/attribute value type errors. A UML class diagram D contains a duplicate name error if it contains the following:

(i) two different classes C_1 and C_2 of the same class name,

(ii) two different associations A_1 and A_2 of the same association name, or

(iii) two different attributes a_1 and a_2 of the same attribute name in a class C.

Moreover, if two classes have identically named attributes $a\colon T_1$ and $a\colon T_2$, such that T_1 is a class and T_2 is a datatype, then the class diagram contains an attribute value type error. Obviously, the checking of these syntax errors in a UML class diagram can be computed in linear time.

We elaborate three inconsistency triggers for the UML class diagrams. Let $\mathcal{G}(D)$ be the translation of a UML class diagram D into a set of implication forms; C, C' be classes; A, A' be associations; and $F(x)$ and $F(x_1, \ldots, x_n)$ be formulas with free variables. The reflexive and transitive closure $\mathcal{G}(D)^*$ of $\mathcal{G}(D)$ is defined by the following:

(i) $C(x) \to^* C(x) \in \mathcal{G}(D)^*$,

(ii) $A(x_1, \ldots, x_n) \to^* A(x_1, \ldots, x_n) \in \mathcal{G}(D)^*$,

(iii) if $C(x) \to F(x) \in \mathcal{G}(D)$, or $C(x) \to^* C'(x), C'(x) \to^* F(x) \in \mathcal{G}(D)^*$, then $C(x) \to^* F(x) \in \mathcal{G}(D)^*$, and

(iv) if $A(x_1, \ldots, x_n) \to F(x_1, \ldots, x_n) \in \mathcal{G}(D)$, or $A(x_1, \ldots, x_n) \to^* A'(x_1, \ldots, x_n), A'(x_1, \ldots, x_n) \to^* F(x_1, \ldots, x_n) \in \mathcal{G}(D)^*$, then $A(x_1, \ldots, x_n) \to^* F(x_1, \ldots, x_n) \in \mathcal{G}(D)^*$.

Inconsistency trigger 1 (generalization and disjointness) The first inconsistency trigger is caused by a combination of generalization and a disjointness constraint. A class diagram D has an inconsistency trigger if the translation $\mathcal{G}(D)^*$ contains the formulas $C(x) \to^* C_k(x)$ and $C(x) \to^* \neg C_1(x) \wedge \cdots \wedge \neg C_n(x)$ where $1 \leq k \leq n$. As shown in

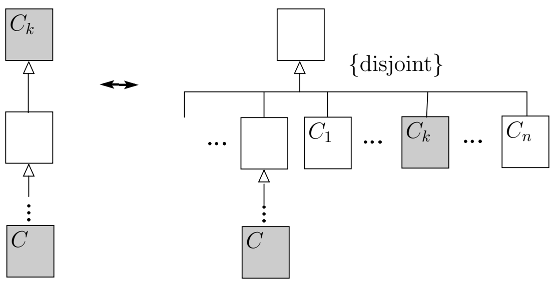

Figure 2: A combination of generalization and disjointness

Figure 2, this inconsistency arises when a class C has a superclass C_k and the classes C and C_k are defined as being disjoint to each other in the constraint of a class hierarchy.

Inconsistency trigger 2 (overwriting/multiple inheritance) The second inconsistency trigger is caused by one of the following situations:

1. (a) conflict between attributes value types T_1 and T_2 when they appear in attributes $a\colon T_1$ and $a\colon T_2$ with the same name, or (b) conflict between multiplicities $[i..j]$ and $[i'..j']$ when they appear in multiplicities $a\colon T_1$ and $a\colon T_2$ of attributes with the same name.

2. conflict between multiplicities when they appear in association and super-associations.

Types T_1 and T_2 are disjoint if they are classes C_1 and C_2 such that $C_1(x) \to^* \neg C_2 \in \mathcal{G}(D)$ or if they are datatypes t_1 and t_2 such that $t_1 \cap t_2 = \emptyset$. More formally, a class diagram D has an inconsistency trigger if the translation $\mathcal{G}(D)^*$ contains a group of the following formulas:

1. $C_2(x) \to^* C_1(x)$, or $C(x) \to^* C_1(x)$ and $C(x) \to^* C_2(x)$, together with
   (a) **Attribute value types:**
   $C_1(x) \to (a(x, y) \to T_1(y))$ and $C_2(x) \to (a(x, y) \to T_2(y))$ where T_1 and T_2 are disjoint, or
   (b) **Attribute multiplicities:**
   $C_1(x) \to \exists_{\geq i} z.a(x, z) \wedge \exists_{\leq j} z.a(x, z)$ and $C_2(x) \to \exists_{\geq i'} z.a(x, z) \wedge \exists_{\leq j'} z.a(x, z)$ where $i > j'$.

2. **Association multiplicities:** $A(x_1, \ldots, x_n) \to A'(x_1, \ldots, x_n)$ with
   $C_k(x) \to \exists_{\geq m_{(1,l)}} x_1 \cdots \exists_{\geq m_{(k-1,l)}} x_{k-1}\ \exists_{\geq m_{(k+1,l)}} x_{k+1} \cdots \exists_{\geq m_{(n,l)}} x_n.A(x_1, \ldots, x_n)[x_k/x]$
   and
   $C'_k(x') \to \exists_{\leq m'_{(1,u)}} x'_1 \cdots \exists_{\leq m'_{(k-1,u)}} x'_{k-1} \exists_{\leq m'_{(k+1,u)}} x'_{k+1} \cdots \exists_{\leq m'_{(n,u)}} x'_n.A'(x'_1, \ldots, x'_n)[x'_k/x']$
   where $m_{(i,l)} > m'_{(i,u)}$.

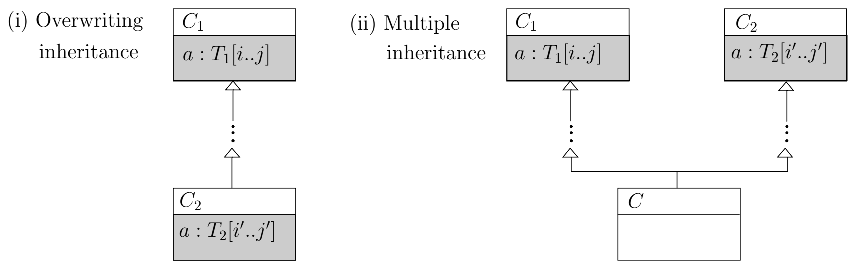

Figure 3: Inheritances in class hierarchy

From Figure 3, it can be seen that (i) a class C_2 with an attribute $a\colon T_2[i'..j']$ inherits the identically named attribute $a\colon T_1[i..j]$ from a superclass C_1 and (ii) a class C inherits the two attributes $a\colon T_1[i..j]$ and $a\colon T_2[i'..j']$ of the same name from superclasses C_1 and C_2. The former is called *overwriting inheritance* and the latter, *multiple inheritance*. In these cases, if the attribute value types T_1 and T_2 are disjoint or if the multiplicities $[i..j]$ and $[i'..j']$ conflict with each other, the attributes are determined to be inconsistent. For example, the multiplicities $[1..5]$ and $[10..*]$ cannot simultaneously hold for the identically named attributes.

Inconsistency trigger 3 (completeness and disjointness) By combining a disjointness constraint with a completeness constraint, we can yield the third inconsistency trigger. A class diagram D has an inconsistency trigger if the translation $\mathcal{G}(D)^*$ contains the following formulas:

$$C(x) \to^* C_1(x) \vee \cdots \vee C_n(x) \text{ and } C(x) \to^* \neg C'_1(x) \wedge \cdots \wedge \neg C'_m(x)$$

where $\{C_1, \ldots, C_n\} \subseteq \{C'_1, \ldots, C'_m\}$. This inconsistency arises when classes C and $C_1, \ldots, C_n$ satisfy the completeness constraint in a class hierarchy and classes C and $C'_1, \ldots, C'_m$ satisfy the disjointness constraint in another class hierarchy. Intuitively, any instance of class C must be an instance of one of the classes $C_1, \ldots, C_n$, but each instance of class C cannot be an instance of the classes $C'_1, \ldots, C'_m$. Hence, this situation is contradictory.

The third inconsistency trigger may be more complicated when the number of completeness and disjointness constraints that occur in a class diagram is increased. In other words, disjunctive expressions raised by many completeness constraints expand the search space of finding inconsistency. Let $\mathcal{G}(D)^*$ be the reflexive and transitive closure of $\mathcal{G}(D)$. We define the disjunctive closure $\mathcal{G}(D)^+$ of $\mathcal{G}(D)^*$ as follows:

(i) if $C(x) \to^* F(x) \in \mathcal{G}(D)^*$, then $C(x) \to^+ F(x) \in \mathcal{G}(D)^+$, and

(ii) if $C(x) \to^+ C_1(x) \vee \cdots \vee C_n(x), C_k(x) \to^+ DC(x) \in \mathcal{G}(D)^+$ where $1 \leq k \leq n$ and $DC(x) = C'_1(x) \vee \cdots \vee C'_m(x)$, then

$$C(x) \to^+ C_1(x) \vee \cdots \vee C_{k-1}(x) \vee DC(x) \vee C_{k+1}(x) \vee \cdots \vee C_n(x) \in \mathcal{G}(D)^+.$$

A class diagram D has an inconsistency trigger if the translation $\mathcal{G}(D)^+$ contains the formulas $C(x) \to^+ C_1(x) \vee \cdots \vee C_n(x)$ and $C(x) \to^+ \neg C_{(i,1)}(x) \wedge \cdots \wedge \neg C_{(i,m_i)}(x)$ for each $i \in \{1, \ldots, n\}$, where C_i is one of the classes $C_{(i,1)}, \ldots, C_{(i,m_i)}$. For example, Figure 4 illustrates that two completeness constraints are complicatedly inconsistent with respect to a disjointness constraint.

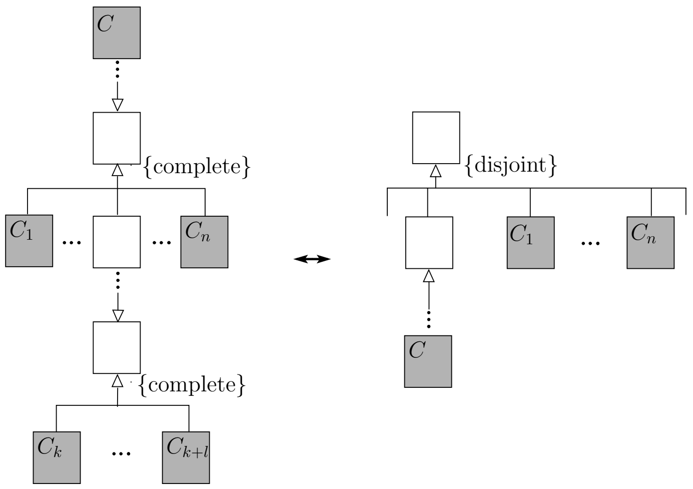

Figure 4: A combination of completeness and disjointness

We define a formal model of UML class diagrams using the semantics of FOPL with counting quantifiers. An interpretation $\mathcal{I}$ is an ordered pair (U, I) of the universe U and an interpretation function I for a first-order language.

Definition 1 (UML class diagram models) *Let $\mathcal{I} = (U, I)$ be an interpretation. The interpretation $\mathcal{I}$ is a model of a UML class diagram D (called a UML-model of D) if*

1. *$I(C) \neq \emptyset$ for every class C in D and*

2. *$\mathcal{I}$ satisfies $\mathcal{G}(D)$ where $\mathcal{G}(D)$ is the translation of D.*

The first condition indicates that all classes are non-empty (i.e., an instance of the class exists), and the second condition implies that $\mathcal{I}$ is a first-order model of the class diagram formulation $\mathcal{G}(D)$. A UML class diagram D is consistent if it has a UML-model.

Remark. The class diagram in Figure 5 is invalid because the association class C_A cannot be used for two different binary associations between classes C_1 and C_2 and between classes C_1 and C_3. In place of C_A, we describe a ternary association or two association classes. It

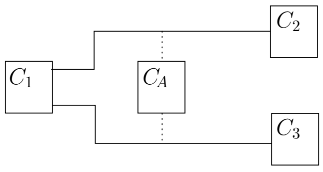

Figure 5: An invalid class diagram

appears that the EXPTIME-hardness in [3] relies on such expressions because they correspond to a knowledge base in the Description Logic $\mathcal{ALC}$ (denoted an $\mathcal{ALC}$ KB) where the satisfiability problem of an $\mathcal{ALC}$ KB is EXPTIME-hard. More precisely, when we reduce the (EXPTIME-hard) concept satisfiability in an $\mathcal{ALC}$ KB to class consistency in a UML class diagram, the $\mathcal{ALC}$ KB

$$\{C_1 \sqsubseteq \exists P_A.C_2,\ C_1 \sqsubseteq \exists P_A.C_3\}$$

is encoded into an invalid association class. This condition is important in order to avoid EXPTIME-hardness and therefore to derive the complexity results in Section 5. Because a UML class diagram with such an association class can encode any $\mathcal{ALC}$ KB, the consistency checking of a more expressive UML class diagram is also EXPTIME-hard. In a technical way, the association class can be expressed by using complex combinations of association generalization and multiplicities in a class diagram (as shown in [1]). However, such a complicated encoding is not important for the development of a software using UML class diagrams. Therefore, by simplifying association generalization in the following definition, we obtain UML class diagrams that have no expressive power to encode the $\mathcal{ALC}$ KB. This enables us to show that the consistency checking of some restricted UML class diagram groups is computable in P and PSPACE.

Definition 2 (Safe class diagrams) *Association generalization in a class diagram is simple if there is no class C that has associations $A_1, ..., A_n$ with classes $C_1, \dots, C_n$ ($n \geq 2$) such that $A_1, ..., A_n$ are subassociations of a common association A and the maximum multiplicity of A on the superclass of C is lower than the sum of the maximum multiplicities of $A_1, ..., A_n$ on $C_1, \dots, C_n$. A class diagram is safe if it is valid and its association generalization is simple.*

It can be checked in linear time if each class diagram is safe. In the rest of this paper, we assume that every class diagram is safe.

The following lemma shows that the three inconsistency triggers describe logical inconsistencies in UML class diagrams.

Lemma 3 *If a UML class diagram contains an inconsistency trigger, then it has no UML-model.*

Proof. Let D be a UML class diagram and let $\mathcal{G}(D)$ be the translation of D. If D contains an inconsistency trigger, then contradictory formulas are included in $\mathcal{G}(D)$. Therefore, there is no UML-model of D. ∎

The three inconsistency triggers can be found structurally in a UML class diagram; in particular, by tracing the UML components once, we can determine whether or not the conditions of the first and second inconsistency triggers hold. Intuitively, each inconsistency trigger indicates that a class is directly inconsistent with some components under the multiplicities and disjointness and completeness constraints.

Lemma 4 *Finding the first and second inconsistency triggers in a UML class diagram is computable in linear time. Moreover, finding the third inconsistency trigger in a UML class diagram is computable in NP (non-deterministic polynomial time).*

Proof. Let D be a UML class diagram and let $\mathcal{G}(D)$ be the translation of D. Suppose that $|\mathcal{G}(D)| = n$. Then, the first and second inconsistency triggers can be checked on class hierarchies in n steps.

In order to find the third inconsistency trigger, the reflexive and transitive closure $\mathcal{G}(D)^*$ of $\mathcal{G}(D)$ is computed in n^2 steps, i.e., for each class, all the reachable classes over implication forms are computed. The disjunctive closure $\mathcal{G}(D)^+$ of $\mathcal{G}(D)^*$ identifies this inconsistency trigger. The disjunctive closure $\mathcal{G}(D)^+$ is computed in exponential steps for each class C. However, each disjunctive form $C(x) \to^+ C_1(x) \vee \cdots \vee C_h(x) \in \mathcal{G}(D)^+$ is non-deterministically computed in at most n^3 steps because $h \leq n$ and $|\mathcal{G}(D)^*| \leq n^2$. If all formulas of the form $C(x) \to^+ \neg C'_1(x) \wedge \cdots \wedge \neg C'_m(x) \in \mathcal{G}(D)^*$ is consistent with the disjunctive form, then there is no third inconsistency trigger. Therefore, finding the third inconsistency trigger is non-deterministically computed in $n^3 + n^2$, i.e., $\mathcal{O}(n^3)$. ∎

However, it is difficult to structurally check *any* logical inconsistency in a UML class diagram if a class has or inherits the identically named attributes $a\colon C_1[i..j]$ and $a\colon C_2[i'..j']$ with $i, i' \geq 1$. This implies that C_1 and C_2 have a common object, and therefore the conjunction of C_1 and C_2 has to be checked. Many combinations of such conjunctions give rise to indirect checking in the UML class diagram. As a result, many subsets of the set of classes are checked in the worst case. In the next section, we will design a *complete* consistency checking algorithm for finding such complicated inconsistencies in a UML class diagram.

4 Consistency checking

This section presents a consistency checking algorithm for a set of implication forms Γ_0 (corresponding to the UML class diagram formulation $\mathcal{G}(D)$). It consists of two sub-algorithms *Cons* and *Assoc*; *Cons* checks the consistency of a class in Γ_0 and *Assoc* tests the consistency of association generalization in Γ_0.

4.1 Algorithm for testing consistency

We decompose an implication form set Γ_0 in order to apply our consistency checking algorithm to it. Let Γ_0 be a set of implication forms, C be a class, and $F_i(x)$ be any formula including a free variable x. Γ is a decomposed set of Γ_0 if the following conditions hold:

(i) $\Gamma_0 \subseteq \Gamma$,

(ii) if $C(x) \rightarrow F_1(x) \wedge \cdots \wedge F_n(x) \in \Gamma$, then $C(x) \rightarrow F_1(x) \in \Gamma, \ldots, C(x) \rightarrow F_n(x) \in \Gamma$, and

(iii) if $C(x) \rightarrow F_1(x) \vee \cdots \vee F_n(x) \in \Gamma$, then $C(x) \rightarrow F_i(x) \in \Gamma$ for some $i \in \{1, \ldots, n\}$.

We denote $\Sigma(\Gamma_0)$ as the family of decomposed sets of Γ_0.

We denote $cls(\Gamma_0)$ as the set of classes, $att(\Gamma_0)$ as the set of attributes, and $asc(\Gamma_0)$ as the set of associations that occur in the implication form set Γ_0.

Definition 5 *Let C be a class, Γ be a decomposed set of Γ_0, and δ be a set of classes. Then, the following operations will be embedded as subroutines in the consistency checking algorithm:*

1. $H(C, \Gamma) = \{C' \mid C(x) \rightarrow^* C'(x) \in \Gamma\} \cup \{\neg C' \mid C(x) \rightarrow^* \neg C'(x) \in \Gamma\}$.
2. $E(\delta, a, \Gamma) = \bigcup_{C \in \delta} E(C, a, \Gamma)$ *where*
   $E(C, a, \Gamma) = \{C' \mid C(x) \rightarrow^* (a(x,y) \rightarrow C'(y)) \in \Gamma$ *and* $C(x) \rightarrow^* \exists_{\geq i} z.a(x,z) \in \Gamma\}$ *with* $i \geq 1$.
3. $N(\delta, a, \Gamma) = \bigcup_{C \in \delta} N(C, a, \Gamma)$ *where*
   $N(C, a, \Gamma) = \{\geq i \mid C(x) \rightarrow^* \exists_{\geq i} z.a(x,\, z) \in \Gamma\} \cup \{\leq j \mid C(x) \rightarrow^* \exists_{\leq j} z.a(x, z) \in \Gamma\}$.
4. $\mu_0(\delta, \Gamma) = \{C\}$ *if for all* $C' \in \mu(\delta, \Gamma)$, $C \preceq C'$ *and* $C \in \mu(\delta, \Gamma)$ *where*
   $\mu(\delta, \Gamma) = \{C \in \delta \mid \delta \subseteq H(C, \Gamma)\}$ *and* $\preceq$ *is a linear order over* $cls(\Gamma_0)$.

The operation $H(C, \Gamma)$ denotes the set of superclasses C' of C and disjoint classes $\neg C'$ of C in Γ. The operation $E(\delta, a, \Gamma)$ gathers the set of value types T of attribute a in Γ such that each value type T is of classes in δ. Further, the operation $N(\delta, a, \Gamma)$ gathers the set of multiplicities $\geq i$ and $\leq j$ of attribute a in Γ such that each of these multiplicities is of classes in δ. The operation $\mu(\delta, \Gamma)$ returns a set $\{C_1, \ldots, C_n\}$ of classes in δ such that the superclasses of each C_i (in Γ) subsume all the classes in δ. The operation $\mu_0(\delta, \Gamma)$ returns the singleton set $\{C\}$ of a class in $\mu(\delta, \Gamma)$ such that C is the least class in $\mu(\delta, \Gamma)$ over $\preceq$.

The consistency checking algorithm $Cons$ is described in Figure 6. In order to decide the consistency of the input implication form set Γ_0, we execute the algorithm $Cons(\{C\}, \emptyset, \Gamma_0)$ for every class $C \in cls(\Gamma_0)$. If C is consistent in Γ_0, it returns 1, else 0. At the first step of the algorithm, a decomposed set Γ of Γ_0 (in $\Sigma(\Gamma_0)$) which is one of the disjunctive branches with respect to the completeness constraints in Γ_0, is selected. Subsequently, for each $\Gamma \in \Sigma(\Gamma_0)$, the following three steps are carried out.

(1) For the selected Γ, the algorithm checks whether all the superclasses of classes in $\delta = \{C\}$ (obtained from $S = \bigcup_{C \in \delta} H(C, \Gamma)$) are disjoint to each other. Intuitively, it sets a dummy instance of class C; the dummy instance is regarded as an instance of the superclasses C' of C and of the disjoint classes $\neg C'$ of C along the implication forms $C(x) \rightarrow^* C'(x)$ and

```
Algorithm $Cons$
input set of classes $\delta$, family of sets of classes $\Delta$, set of implication forms $\Gamma_0$
output 1 (consistent) or 0 (inconsistent)
begin
  for $\Gamma \in \Sigma(\Gamma_0)$ do
    $S = \bigcup_{C \in \delta} H(C, \Gamma)$; $f_\Gamma = 0$;
    if $\{C, \neg C\} \not\subseteq S$ and $\{t_1, \ldots, t_n\} \not\subseteq S$ s.t. $t_1 \cap \cdots \cap t_n = \emptyset$ then $f_\Gamma = 1$;
      for $a \in att(\Gamma_0)$ do
        if $i > j$ s.t. $\{\geq i, \leq j\} \subseteq N(\delta, a, \Gamma)$ then $f_\Gamma = 0$;
        else $\delta_a = E(\delta, a, \Gamma)$;
            if $\delta_a \neq \emptyset$ and $\delta_a, \mu_0(\delta_a, \Gamma) \not\subseteq \Delta$ then $f_\Gamma = Cons(\delta_a, \Delta \cup \{\delta\}, \Gamma_0)$;
        esle;
      rof;
    fi;
    if $f_\Gamma = 1$ then return 1;
  rof;
  return 0;
end;
```

Figure 6: The consistency checking algorithm *Cons*

$C(x) \to^* \neg C'(x)$ in Γ. If an inconsistent pair C_i and $\neg C_i$ possesses the dummy instance, then δ is determined to be inconsistent in Γ. For example, $\{C\}$ is inconsistent in $\Gamma_1 = \{C(x) \to C_1(x), C_1(x) \to C_2(x), C(x) \to \neg C_2(x)\}$ since it is necessary for the inconsistent pair C_2 and $\neg C_2$ to have the dummy instance of class C, i.e., $H(C, \Gamma_1) = \{C, C_1, C_2, \neg C_2\}$.

(2) If step (1) finds no inconsistency in Γ, the algorithm then checks the multiplicities of all the attributes $a \in att(\Gamma_0)$. The multiplicities of the same attribute name a are obtained by $N(\delta, a, \Gamma)$; therefore, when $N(\delta, a, \Gamma)$ contains $\{\geq i, \leq j\}$ with $i > j$, these multiplicities are inconsistent. Intuitively, similar to phase (1), the algorithm checks whether superclasses involve conflicting multiplicities along the implication form $C(x) \to^* C'(x)$ in Γ. For example, $\{C\}$ is inconsistent in

$$\Gamma_2 = \{C(x) \to \exists_{\geq 10} z.a(x, z), C(x) \to C_1(x), C_1(x) \to \exists_{\leq 5} z.a(x, z)\}$$

since the counting quantifiers $\exists_{\geq 10}$ and $\exists_{\leq 5}$ cannot simultaneously hold when $N(\{C\}, a, \Gamma_2) = \{\geq 10, \leq 5\}$.

(3) Next, the disjointness of attribute value types is checked. Along the implication form $C(x) \to^* C'(x)$ in Γ, the algorithm gathers all the value types of the identically named attributes, obtained by $\delta_a = E(\delta, a, \Gamma)$ for each $a \in att(\Gamma_0)$. For example,

$$\Gamma_3 = \{C(x) \to C_1(x), C(x) \to C_2(x), C_1(x) \to (a(x, y) \to C_3(y)), C_2(x) \to (a(x, y) \to C_4(y))\}$$

derives $\delta_a = \{C_3, C_4\}$ by $E(\{C\}, a, \Gamma_3)$ since superclasses C_1 and C_2 of C have the attributes $a\colon C_3$ and $a\colon C_4$. In other words, each value of attribute a is typed by C_3 and C_4. Hence, the algorithm needs to check the consistency of $\delta_a = \{C_3, C_4\}$. In order to accomplish this, it recursively calls $Cons(\delta_a, \Delta \cup \{\{C\}\}, \Gamma_0)$, where δ_a is consistent if 1 is returned. The second argument $\Delta \cup \{\{C\}\}$ prevents infinite looping by storing sets of classes wherein each set is already checked in the caller processes.

In order to find a consistent decomposed set Γ in the disjunctive branches of $\Sigma(\Gamma_0)$, if the three phases (1), (2), and (3) do not detect any inconsistency in Γ, the algorithm sets

```
Algorithm Assoc
input set of implication forms $\Gamma_0$
output 1 (consistent) or 0 (inconsistent)
begin
  for $A \in asc(\Gamma_0)$ and $k \in \{1, \ldots, n\}$ s.t. $arity(A) = n$ do
    if $i_v > j_v$ s.t. $\{(\geq i_1, \ldots, \geq i_{k-1}, \geq i_{k+1}, \ldots, \geq i_n),$
      $(\leq j_1, \ldots, \leq j_{k-1}, \leq j_{k+1}, \ldots, \leq j_n)\} \subseteq N_k(H(A, \Gamma_0), \Gamma_0)$ then return 0;
  rof;
  return 1;
end;
```

Figure 7: The association checking algorithm *Assoc*

the flag $f_\Gamma = 1$, else it sets $f_\Gamma = 0$. Thus, the flag $f_\Gamma = 1$ indicates that $\{C\}$ is consistent in the input Γ_0, i.e., $Cons(\{C\}, \emptyset, \Gamma_0) = 1$.

As defined below, the operations $H(A, \Gamma_0)$ and $N_k(\alpha, \Gamma_0)$ return the set of super-associations A' of A and the set of $n-1$-tuples of multiplicities of n-ary associations A in α along the implication forms

$$C_k(x) \to \exists_{\geq i_1} x_1 \cdots \exists_{\geq i_{k-1}} x_{k-1} \exists_{\geq i_{k+1}} x_{k+1} \cdots \exists_{\geq i_n} x_n . A(x_1, \ldots, x_n)[x_k/x]$$

and

$$C_k(x) \to \exists_{\leq j_1} x_1 \cdots \exists_{\leq j_{k-1}} x_{k-1} \exists_{\leq j_{k+1}} x_{k+1} \cdots \exists_{\leq j_n} x_n . A(x_1, \ldots, x_n)[x_k/x],$$

respectively.

Definition 6 *The operations $H(A, \Gamma_0)$ and $N_k(\alpha, \Gamma_0)$ are defined as follows:*

1. $H(A, \Gamma_0) = \{A' \mid A(x_1, \ldots, x_n) \to^* A'(x_1, \ldots, x_n) \in \Gamma_0\}$.

2. $N_k(\alpha, \Gamma_0) = \bigcup_{A \in \alpha} N_k(A, \Gamma_0)$ *where*
   $N_k(A, \Gamma_0) = \{(\geq i_1, \ldots, \geq i_{k-1}, \geq i_{k+1}, \ldots, \geq i_n) \mid$
   $C_k(x) \to \exists_{\geq i_1} x_1 \cdots \exists_{\geq i_{k-1}} x_{k-1} \exists_{\geq i_{k+1}} x_{k+1} \cdots \exists_{\geq i_n} x_n . A(x_1, \ldots, x_n)[x_k/x] \in \Gamma_0\} \cup$
   $\{(\leq j_1, \ldots, \leq j_{k-1}, \leq j_{k+1}, \ldots, \leq j_n) \mid$
   $C_k(x) \to \exists_{\leq j_1} x_1 \cdots \exists_{\leq j_{k-1}} x_{k-1} \exists_{\leq j_{k+1}} x_{k+1} \cdots \exists_{\leq j_n} x_n . A(x_1, \ldots, x_n)[x_k/x] \in \Gamma_0\}$.

In addition to the algorithm *Cons*, the consistency checking of multiplicities over association generalization is processed by the algorithm *Assoc* in Figure 7. If Γ_0 does not cause any inconsistency with respect to associations, $Assoc(\Gamma_0)$ returns 1, which is computable in polynomial time.

Lemma 7 *The algorithm Assoc computes the consistency of association generalization in polynomial time.*

Proof. Suppose that $|\Gamma_0| = m$. Then, $|cls(\Gamma_0)| \leq m$ and $|asc(\Gamma_0)| \leq m$. When $Assoc(\Gamma_0)$ is called, the number of loops is bounded by at most

$$m^2 = |asc(\Gamma_0)| \times Max(\{arity(A) \mid A \in asc(\Gamma_0)\}).$$

The subroutines $H(A, \Gamma_0)$ and $N_k(H(A, \Gamma_0), \Gamma_0)$ are computable in at most m and $3m$ steps, respectively. Hence, this algorithm is implemented in at most $m^2 \times (m + 3m)$ steps, i.e., $\mathcal{O}(m^4)$. ∎

4.2 Soundness, completeness, and termination

We sketch a proof of the completeness for the algorithms *Cons* and *Assoc*. Assume that $Cons(\{C\}, \emptyset, \mathcal{G}(D))$ for all $C \in cls(\mathcal{G}(D))$ and $Assoc(\mathcal{G}(D))$ are called. We construct an implication tree of $(C, \mathcal{G}(D))$ that expresses the consistency checking proof of C in $\mathcal{G}(D)$. If $Cons(\{C\}, \emptyset, \mathcal{G}(D)) = 1$, there exists a non-closed implication tree of $(C, \mathcal{G}(D))$. In order to prove the existence of a UML-model of D, a canonical interpretation is constructed by consistent subtrees of the non-closed implication trees of $(C_1, \mathcal{G}(D)), \ldots, (C_n, \mathcal{G}(D))$ (with $cls(\mathcal{G}(D)) = \{C_1, \ldots, C_n\}$) and by $Assoc(\mathcal{G}(D)) = 1$. This proves that D is consistent.

Corresponding to calling $Cons(\delta_0, \emptyset, \Gamma_0)$, we define an implication tree of a class set δ_0 that expresses the consistency checking proof of δ_0.

Definition 8 *Let Γ_0 be a set of implication forms and let $\delta_0 \subseteq cls(\Gamma_0)$. An implication tree of (δ_0, Γ_0) is a finite and minimal tree such that (i) the root is a node labeled with δ_0, (ii) each non-leaf node is labeled with a non-empty set of classes, (iii) each leaf is labeled with 0, 1, or w, (iv) each edge is labeled with Γ or (Γ, a) where $\Gamma \in \Sigma(\Gamma_0)$ and $a \in att(\Gamma_0)$, and (v) for each node labeled with δ and each $\Gamma \in \Sigma(\Gamma_0)$, if $\bigcup_{C \in \delta} H(C, \Gamma)$ contains $\{C, \neg C\}$ or $\{t_1, \ldots, t_n\}$ with $t_1 \cap \cdots \cap t_n = \emptyset$, then there is a child of δ labeled with 0 and the edge of the nodes δ and 0 is labeled with Γ, and otherwise:*

- *if $att(\Gamma_0) = \emptyset$, then there is a child of δ labeled with 1 and the edge of the nodes δ and 1 is labeled with Γ, and*
- *for all $a \in att(\Gamma_0)$, the following conditions hold:*
  1. *if $i > j$ such that $\{\geq i, \leq j\} \in N(\delta, a, \Gamma)$, then there is a child of δ labeled with 0 and the edge of the nodes δ and 0 is labeled with (Γ, a),*
  2. *if $E(\delta, a, \Gamma) = \emptyset$, then there is a child of δ labeled with 1 and the edge of the nodes δ and 1 is labeled with (Γ, a),*
  3. *if there is an ancestor labeled with $E(\delta, a, \Gamma)$ or $\mu_0(E(\delta, a, \Gamma), \Gamma)$ (called a witness of the node labeled with δ), then there is a child of δ labeled with w and the edge of the nodes δ and w is labeled with (Γ, a), and*
  4. *otherwise, there is a child of δ labeled with $E(\delta, a, \Gamma)$ and the edge of the nodes δ and $E(\delta, a, \Gamma)$ is labeled with (Γ, a).*

Let $\mathcal{T}$ be an implication tree of (δ_0, Γ_0). A node d in $\mathcal{T}$ is closed if (i) d is labeled with 0 or if (ii) d is labeled with δ and for every $\Gamma \in \Sigma(\Gamma_0)$, there is an edge (d, d') labeled with Γ or (Γ, a) such that d' is closed. An implication tree of (δ_0, Γ_0) is closed if the root is closed. The implication tree of $(\{C\}, \mathcal{G}(D))$ is a finite tree that determines whether or not there is a UML-model for the class C in D. That is, a non-closed implication tree of $(\{C\}, \mathcal{G}(D))$ indicates that the class C is consistent in D. The following is an example of an implication tree.

Example 1 *Let $\mathcal{G}(D)$ be the translation of a UML class diagram D that contains the following formulas:*

$$C_1(x) \to C(x), C_2(x) \to C(x), C_1(x) \to \neg C_2(x),$$

$$C(x) \to (a_1(x, y) \to t(y)), C(x) \to (a_2(x, y) \to C_2(y)).$$

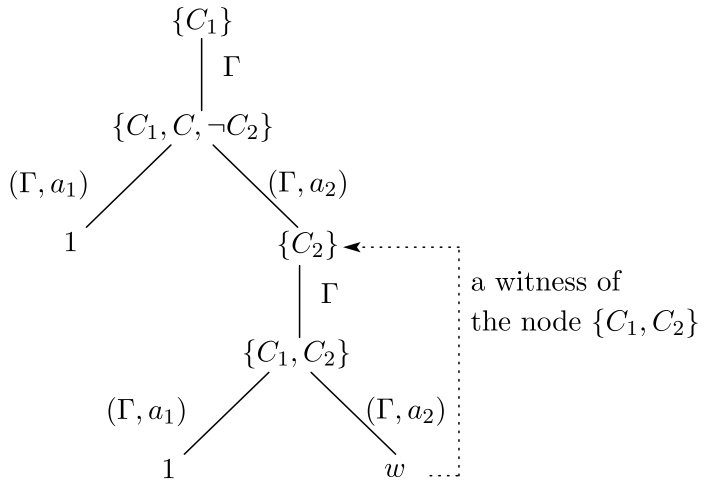

Figure 8: An implication tree of $(\{C_1\}, \mathcal{G}(D))$

As shown in Figure 8, we can construct the implication tree of $(\{C_1\}, \mathcal{G}(D))$ *that is not closed since it does not contain any node labeled with* 0*. In the implication tree, the root is labeled with* $\{C_1\}$ *and every leaf is labeled with* 1 *or* w *where the leaf labeled with* w *has a witness of the parent node labeled with* $\{C_1, C_2\}$*.*

A forest of Γ_0 is a set of implication trees of $(\{C_1\}, \Gamma_0), \ldots, (\{C_n\}, \Gamma_0)$ such that $cls(\Gamma_0) = \{C_1, \ldots, C_n\}$. A forest $\mathcal{S}$ of Γ_0 is closed if there exists a closed implication tree $\mathcal{T}$ in $\mathcal{S}$. The following lemma states the correspondence between the consistency checking for every $C \in cls(\Gamma_0)$ and the existence of a non-closed forest of Γ_0.

Lemma 9 *Let* Γ_0 *be a set of implication forms. For every class* $C \in cls(\Gamma_0)$*,* $Cons(\{C\}, \emptyset, \Gamma_0) = 1$ *if and only if there is a non-closed forest of* Γ_0*.*

Proof. ($\Rightarrow$) Let us assume that for every class $C_0 \in cls(\Gamma_0)$, $Cons(\{C_0\}, \emptyset, \Gamma_0) = 1$. For each $C_0 \in cls(\Gamma_0)$, we construct a tree of $(\{C_0\}, \Gamma_0)$ as follows.

1. Create the root d_0 labeled with $\{C_0\}$.
2. Perform the following operations if a node d labeled with δ is created:
   (a) Create a new node d' labeled with 0 and add the edge (d, d') labeled with Γ if $f_\Gamma = 0$ is kept by satisfying the condition $\{C, \neg C\} \subseteq S$ or $\{t_1, \ldots, t_n\} \subseteq S$ such that $t_1 \cap \cdots \cap t_n = \emptyset$.
   (b) Create a new node d' labeled with 1 and add the edge (d, d') labeled with Γ if $f_\Gamma = 1$ is set by satisfying the conditions $att(\Gamma_0) = \emptyset$, $\{C, \neg C\} \not\subseteq S$, and $\{t_1, \ldots, t_n\} \not\subseteq S$ such that $t_1 \cap \cdots \cap t_n = \emptyset$.
   (c) Perform the following operations for each $a \in att(\Gamma_0)$ if $att(\Gamma_0) \neq \emptyset$, $\{C, \neg C\} \not\subseteq S$, and $\{t_1, \ldots, t_n\} \not\subseteq S$ such that $t_1 \cap \cdots \cap t_n = \emptyset$:
      i. Create a new node d' labeled with 0 and add the edge (d, d') labeled with (Γ, a) if $f_\Gamma = 0$ is set by satisfying the condition $i > j$ such that $\{\geq i, \leq j\} \subseteq N(\delta, a, \Gamma)$.

ii. Create a new node d' labeled with 1 and add the edge (d, d') labeled with (Γ, a) if $f_\Gamma = 1$ is kept by satisfying the conditions $E(\delta, a, \Gamma) = \emptyset$ and $i < j$ for any $\geq i, \leq j \in N(\delta, a, \Gamma)$.

iii. Create a new node d' labeled with w and add the edge (d, d') labeled with (Γ, a) if $f_\Gamma = 1$ is kept by satisfying the conditions that there exists an ancestor labeled with $E(\delta, a, \Gamma)$ or $\mu_0(E(\delta, a, \Gamma), \Gamma)$ (i.e., it belongs to Δ) and $i < j$ for any $\geq i, \leq j \in N(\delta, a, \Gamma)$.

iv. Create a new node d' labeled with δ' and add the edge (d, d') labeled with (Γ, a) if $Cons(E(\delta, a, \Gamma), \Delta \cup \{\delta\}, \Gamma_0)$ is called by satisfying the conditions that $E(\delta, a, \Gamma) \neq \emptyset$, there exists no ancestor labeled with $E(\delta, a, \Gamma)$ or $\mu_0(E(\delta, a, \Gamma), \Gamma)$, and $i < j$ for any $\geq i, \leq j \in N(\delta, a, \Gamma)$.

We will show that this tree satisfies the conditions in Definition 8. By Operation 1, it satisfies Condition (i). By Operation 2 (a), (b), and (c)-i,ii, and iii, every node labeled with 0, 1 or w has no child, and by Operation (c)-iv, if a node has a child, then it is labeled with a set of classes (Conditions (ii) and (iii)). By Operation 2 (a)-(c), Condition (iv) holds. Let d be a node labeled with δ and let $\Gamma \in \Sigma(\Gamma_0)$. If the node d is labeled with 0, 1 or w, then it satisfies Condition (v) by Operation 2 (a), (b), and (c)-i,ii and iii. If the node d is labeled with δ, then it satisfies Condition (v) and by the induction hypothesis, all the children nodes d' satisfy Condition (v).

($\Leftarrow$) Let $\mathcal{S}$ be a non-closed forest of Γ_0 and $\mathcal{T}$ be a non-closed implication tree of $(\{C_0\}, \Gamma_0)$ in $\mathcal{S}$. By induction on the depth of $\mathcal{T}$, we will show that if a node d (in $\mathcal{T}$) labeled with δ is not closed, then $Cons(\delta, \Delta_d, \Gamma_0) = 1$ where Δ_d is the set of ancestor nodes of d. Since d is not closed, a non-closed child d' of d exists. By definition, there exists some $\Gamma \in \Sigma(\Gamma_0)$, and if $att(\Gamma_0) = \emptyset$, then d' is labeled with 1 and the edge (d, d') is labeled with Γ, otherwise, the node d has a non-closed child d_a and the edge (d, d_a) is labeled with (Γ, a) for all $a \in att(\Gamma_0)$. If the child d_a is labeled with 1, then by Definition 8, $E(\delta, a, \Gamma) = \emptyset$. Hence, $\bigcup_{C \in \delta} H(C, \Gamma)$ does not contain $\{C, \neg C\}$ or $\{t_1, \ldots, t_n\}$ with $t_1 \cap \cdots \cap t_n = \emptyset$. If the child d_a is labeled with w, then by Definition 8, $E(\delta, a, \Gamma) \in \Delta_d$. If the child d_a is labeled with δ', then by the induction hypothesis, $Cons(\delta', \Delta_{d_a}, \Gamma_0) = 1$. Thus, there exists no $\{\geq i, \leq j\} \subseteq N(\delta, a, \Gamma)$ such that $i > j$. Therefore, for the non-leaf and non-closed node d labeled with δ, $Cons(\delta, \Delta_d, \Gamma_0) = 1$. It follows that $Cons(\{C_0\}, \emptyset, \Gamma_0) = 1$ for the root labeled with $\{C_0\}$. ∎

We define a consistent subtree $\mathcal{T}'$ of a non-closed implication tree $\mathcal{T}$ such that $\mathcal{T}'$ is constructed by non-closed nodes in $\mathcal{T}$.

Definition 10 (Consistent subtree) *Let $\mathcal{T}$ be a non-closed implication tree of $(\{C_0\}, \Gamma_0)$ and d_0 be the root where Γ_0 is a set of implication forms and $C_0 \in cls(\Gamma_0)$. A tree $\mathcal{T}'$ is a consistent subtree of $\mathcal{T}$ if (i) $\mathcal{T}'$ is a subtree of $\mathcal{T}$, (ii) every node in $\mathcal{T}'$ is not closed, and (iii) every non-leaf node has m children of all the attributes $a_1, \ldots, a_m \in att(\Gamma_0)$ where each child is labeled with 1, w, or a set of classes and each edge of the non-leaf node and its child is labeled with (Γ, a_i).*

We show the correspondence between the consistency of an implication form set Γ_0 and the existence of a non-closed forest of Γ_0. We extend the first-order language by adding the new constants $\bar{d}$ for all the elements $d \in U$ such that each new constant is interpreted by itself, i.e., $I(\bar{d}) = d$. In addition, we define the following operations:

1. $proj_k^n(x_1,\dots,x_n) = x_k$ where $1 \le k \le n$.

2. $Max_{\geq}(X) = (Max(X_1),\dots,Max(X_n))$ where X is a set of n-tuples and for each $v \in \{1,\dots,n\}$, $X_v = \{\ proj_v^n\ (i_1,\ \dots,\ i_n) \mid (\geq i_1,\dots,\geq i_n) \in X\}$.

3. $AC(A,\Gamma) = (C_1,\dots,C_n)$ if $A(x_1,\dots,x_n) \to C_1(x_1) \wedge \cdots \wedge C_n(x_n) \in \Gamma$.

A canonical interpretation of an implication form set Γ_0 is constructed by consistent subtrees of the non-closed implication trees in a forest of Γ_0, that is used to prove the completeness of the algorithm *Cons*. A class C is consistent in Γ if there exists a non-closed implication tree of $(\{C\},\Gamma_0)$ such that the root labeled with $\{C\}$ has a non-closed child node labeled with Γ or (Γ,a).

Definition 11 (Canonical interpretation) *Let Γ_0 be a set of implication forms such that $Assoc(\Gamma_0) = 1$ and let $\mathcal{S} = \{\mathcal{T}_1,\dots,\mathcal{T}_n\}$ be a non-closed forest of Γ_0. For every $\mathcal{T}_i \in \mathcal{S}$, there is a consistent subtree $\mathcal{T}_i'$ of $\mathcal{T}_i$, and we set $\mathcal{S}' = \{\mathcal{T}_1',\dots,\mathcal{T}_n'\}$ as the set of consistent subtrees of $\mathcal{T}_1,\dots,\mathcal{T}_n$ in $\mathcal{S}$. An canonical interpretation of Γ_0 is a pair $\mathcal{I} = (U,I)$ such that $U_0 = \{d \mid d$ is a non-leaf node in $\mathcal{T}_1' \cup \cdots \cup \mathcal{T}_n'\}$, each $e_0, e_j, e_{(v,w)}$ are new individuals, and the following conditions hold:*

$$U = U_0 \cup \bigcup_{\substack{d\in\mathcal{T}_1'\cup\cdots\cup\mathcal{T}_n' \\ a\in att(\Gamma_0)}} U_{d,a} \cup \bigcup_{\substack{d\in\mathcal{T}_1'\cup\cdots\cup\mathcal{T}_n' \\ A\in asc(\Gamma_0)}} U_{d,A} \ \text{ and } I(x) = I_0(x) \cup \bigcup_{\substack{d\in\mathcal{T}_1'\cup\cdots\cup\mathcal{T}_n' \\ a\in att(\Gamma_0)}} I_{d,a}(x) \cup \bigcup_{\substack{d\in\mathcal{T}_1'\cup\cdots\cup\mathcal{T}_n' \\ A\in asc(\Gamma_0)}} I_{d,A}(x)$$

where I_0, $I_{d,a}$, and $I_{d,A}$ are the minimal functions satisfying the following statements in $\mathcal{S}'$:

1. For each $\Gamma \in \Sigma(\Gamma_0)$,

- *$d \in I_0(C)$ if a non-leaf node d is labeled with δ where $C \in \bigcup_{C'\in\delta} H(C',\Gamma)$, and*
- *$(d,d') \in I_0(a)$ if (i) d' is a non-leaf node and (d,d') is an edge labeled with (Γ,a), or (ii) a node d has a child labeled with w, the edge (d,w) is labeled with (Γ,a), and there is a witness d' of d*[2].

2. For each edge (d,d') labeled with (Γ,a) such that the node d is labeled with δ and $Max_{\geq}(N(\delta,a,\Gamma)) = k$,

- *$U_{d,a} = \{e_1,\dots,e_{k-1}\}$,*
- *$(d,e_1),\dots,(d,e_{k-1}) \in I_{d,a}(a)$ if $(d,d') \in I_0(a)$,*
- *$e_1,\dots,e_{k-1} \in I_{d,a}(C)$ if $d' \in I_0(C)$, and*
- *$(e_1,d''),\dots,(e_{k-1},d'') \in I_{d,a}(a')$ if $(d',d'') \in I_0(a')$.*

3. For all nodes $d \in I_0(C_k)$ such that $AC(A,\Gamma_0) = (C_1,\dots,C_k,\dots,C_n)$ and

$$Max_{\geq}(N_k(H(A,\Gamma_0),\Gamma_0)) = (i_1,\dots,i_{k-1},i_{k+1},\dots,i_n),$$

- *$U_{d,A} = \{e_0\} \cup \bigcup_{v\in\{1,\dots,n\}\setminus\{k\}} \{e_{(v,1)},\dots,e_{(v,i_v)}\}$,*

[2]In Definition 8, for each node labeled with w, there is a witness of the parent node d.

- *for all* $(w_1,\dots,w_{k-1},w_{k+1},\dots,w_n) \in \mathbf{N}^{n-1}$ *with* $1 \le w_v \le i_v$,
$$(e_{(1,w_1)},\dots,e_{(k-1,w_{k-1})},d,e_{(k+1,w_{k+1})},\dots,e_{(n,w_n)}) \in I_{d,A}(A) \textit{ and}$$
$$e_{(1,w_1)} \in I_{d,A}(C_1),\dots,e_{(k-1,w_{k-1})} \in I_{d,A}(C_{k-1}),$$
$$e_{(k+1,w_{k+1})} \in I_{d,A}(C_{k+1}),\dots,e_{(n,w_n)} \in I_{d,A}(C_n),$$
- $e_{(v,w)} \in I_{d,A}(C')$ *for all* $C' \in H(C_v,\Gamma')$ *if* $e_{(v,w)} \in I_{d,A}(C_v)$ *and* C_v *is consistent in* Γ',
- $(u_1,\dots,u_n) \in I_{d,A}(A')$ *for all* $A' \in H(A,\Gamma_0)$ *if* $(u_1,\dots,u_n) \in I_{d,A}(A)$[3],
- $(e_{(v,w)},d'') \in I_{d,A}(a)$ *and* $e_{(v,w)} \in I_{d,A}(C_v)$ *if* $(d',d'') \in I_0(a)$ *and* $d' \in I_0(C_v)$, *and*
- *for all* $(w_1,\dots,w_{k-1},w_{k+1},\dots,w_n) \in \mathbf{N}^{n-1}$ *with* $1 \le w_v \le i_v$,
$$(e_{(1,w_1)},\dots,e_{(k-1,w_{k-1})},e,e_{(k+1,w_{k+1})},\dots,e_{(n,w_n)}) \in I_{d,A}(A)$$
if $e \in I(C_k)$ *where* e *is* e_0, e_j, *or* $e_{(x,y)}$.

4. *For all* $A \in asc(\Gamma_0)$,
   - $(u_1,\dots,u_n,e_0) \in I_{d,A}(r_0)$ *and* $e_0 \in I_{d,A}(C_A)$ *if* $(u_1,\dots,u_n) \in I_{d,A}(A)$,
   - $e_0 \in I_{d,A}(C)$ *for all* $C \in H(C_A,\Gamma')$ *if* $e_0 \in I_{d,A}(C_A)$ *and* C_v *is consistent in* Γ', *and*
   - $(e_0,d'') \in I_{d,A}(a)$ *and* $e_0 \in I_{d,A}(C_A)$ *if* $(d',d'') \in I_0(a)$ *and* $d' \in I_0(C_A)$.

This canonical interpretation is generated from a set of non-closed implication trees in order to define a UML-model of D. If an implication tree contains a leaf labeled with w to avoid a cyclic structure, the cyclic structure is constructed in $I_0(a)$ according to Statement 1 of Definition 11. Moreover, the multiplicities of attributes a and associations A are actually modeled in $I_{d,a}$ and $I_{d,A}$, respectively, according to Statements 2–4 of Definition 11 where $U_{d,a}$ and $U_{d,A}$ are introduced as the sets of individuals in the interpretation of the multiplicities.

Lemma 12 *Let* Γ_0 *be a set of implication forms. There exists an interpretation* $\mathcal{I}$ *such that for every* $C_0 \in cls(\Gamma_0)$, $\mathcal{I} \models \exists x.C_0(x)$ *if and only if (i) there exists a non-closed forest of* Γ_0 *and (ii)* $Assoc(\Gamma_0) = 1$.

Proof. ($\Rightarrow$) Let $\mathcal{I} = (U,I)$ be a FOPL-model $\mathcal{I}$ of Γ_0 such that for every $C_0 \in cls(\Gamma_0)$, $\mathcal{I} \models \exists x.C_0(x)$. Using the tree construction in the proof of Lemma 9, we can construct an implication tree $\mathcal{T}$ of $(\{C_0\}\Gamma_0)$.

(i) We show that if $\mathcal{I} \models \exists x.C_1(x) \wedge\cdots\wedge C_n(x)$ and there exists a node d_0 in $\mathcal{T}$ labeled with $\delta = \{C_1,\dots,C_n\}$, then the node d_0 is not closed. Let $u_0 \in U$ such that $\mathcal{I} \models C_1(\bar{u}_0) \wedge \cdots \wedge C_n(\bar{u}_0)$ and let a node d_0 be labeled with $\delta = \{C_1,\dots,C_n\}$. Due to $\mathcal{I} \models \Gamma_0$, a decomposed set Γ of Γ_0 is satisfied by $\mathcal{I}$, and therefore, for every $L \in \bigcup_{C\in\delta} H(C,\Gamma)$, $\mathcal{I} \models L(\bar{u}_0)$ where L is a class C, a disjoint class $\neg C$, or a datatype t. Hence, $\bigcup_{C\in\delta} H(C,\Gamma)$ does not contain $\{C,\neg C\}$ or $\{t_1,\dots,t_n\}$ with $t_1 \cap \cdots \cap t_n = \emptyset$.

[3]Note that d, d', d'', d_0 are nodes, $e_0, e_j, e_{(v,w)}$ are new constants, and u, u_j are nodes or new constants.

If $att(\Gamma_0) = \emptyset$, then a non-closed child node d' of d_0 is labeled with 1 and the edge (d_0, d') is labeled with Γ. Otherwise, for all $\geq j \in N(\delta, a, \Gamma)$, $\mathcal{I} \models \exists_{\geq i} z.a(\bar{u}_0, z)$ where $a \in att(\Gamma_0)$. For all $\leq j \in N(\delta, a, \Gamma)$, $\mathcal{I} \models \exists_{\leq j} z.a(\bar{u}_0, z)$. Hence, there exists no $\{\geq i, \leq j\} \subseteq N(\delta, a, \Gamma)$ such that $i > j$.

For each $a \in att(\Gamma_0)$, if $E(\delta, a, \Gamma) = \emptyset$, then a non-closed child node d' of d_0 is labeled with 1 and the edge (d_0, d') is labeled with (Γ, a). If there is an ancestor labeled with $E(\delta, a, \Gamma)$ or $\mu_0(E(\delta, a, \Gamma), \Gamma)$, then a non-closed child node d' of d_0 is labeled with w and the edge (d_0, d') is labeled with (Γ, a). Otherwise, there exists $u \in U$ such that for all $v \in \{1, \ldots, n\}$, $\mathcal{I} \models C_v(\bar{u})$ where $E(\delta, a, \Gamma) = \{C_1, \ldots, C_n\}$. So, since for all $v \in \{1, \ldots, n\}$, $\mathcal{I} \models \exists_{\geq i} z.a(\bar{u}_0, z)$ $(i \geq 1)$ and $\mathcal{I} \models a(\bar{u}_0, y) \to C_v(y)$, we have $\mathcal{I} \models \exists x.C_1(x) \wedge \cdots \wedge C_n(x)$. By the induction hypothesis, there exists a non-closed implication tree of $(E(\delta, a, \Gamma), \Gamma_0)$.

By the assumption, for every $C_0 \in cls(\Gamma_0)$, $\mathcal{I} \models \exists x.C_0(x)$ and the root of an implication tree of $(\{C_0\}, \Gamma_0)$ is labeled with $\{C_0\}$. Therefore, the tree is not closed. It follows that a non-closed forest of Γ_0 exists.

(ii) Due to $\mathcal{I} \models \Gamma_0$, a decomposed set Γ of Γ_0 is satisfied by $\mathcal{I}$. Let $A \in asc(\Gamma_0)$ and $k \in \{1, \ldots, n\}$ such that $arity(A) = n$. If $A(x_1, \ldots, x_n) \to A'(x_1, \ldots, x_n) \in \Gamma_0$, then for all $\bar{u}_1, \ldots, \bar{u}_n \in U, \mathcal{I} \models A(\bar{u}_1, \ldots, \bar{u}_n) \to A'(\bar{u}_1, \ldots, \bar{u}_n)$. If $C_1(x) \to C'_1(x), \ldots, C_n(x) \to C'_n(x)$ in Γ, then for all $\bar{u}_0 \in U, \mathcal{I} \models C_1(\bar{u}_0) \to C'_1(\bar{u}_0), \ldots, \mathcal{I} \models C_n(\bar{u}_0) \to C'_n(\bar{u}_0)$. Let $\bar{u}_0 \in I(C_k)$. We have

$$\mathcal{I} \models \exists_{\geq i_1} x_1 \cdots \exists_{\geq i_{k-1}} x_{k-1} \exists_{\geq i_{k+1}} x_{k+1} \cdots \exists_{\geq i_n} x_n.A'(x_1, \ldots, x_n)[x_k/\bar{u}_0]$$

and

$$\mathcal{I} \models \exists_{\leq j_1} x_1 \cdots \exists_{\leq j_{k-1}} x_{k-1} \exists_{\leq j_{k+1}} x_{k+1} \cdots \exists_{\leq j_n} x_n.A'(x_1, \ldots, x_n)[x_k/\bar{u}_0].$$

Hence, there exists no $i_v > j_v$ such that

$$\{(\geq i_1, \ldots, \geq i_{k-1}, \geq i_{k+1}, \ldots, \geq i_n), (\leq j_1, \ldots, \leq j_{k-1}, \leq j_{k+1}, \ldots, \leq j_n)\} \subseteq N_k(H(A, \Gamma_0), \Gamma_0).$$

Therefore, $Assoc(\Gamma_0) = 1$.

$(\Leftarrow)$ Let $\mathcal{S} = \{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$ be a non-closed forest of Γ_0 and let $Assoc(\Gamma_0) = 1$. Then, there exists the set $\mathcal{S}' = \{\mathcal{T}'_1, \ldots, \mathcal{T}'_n\}$ of consistent subtrees of $\mathcal{T}_1, \ldots, \mathcal{T}_n$ in $\mathcal{S}$, that is used to construct a canonical interpretation $\mathcal{I} = (U, I)$ of Γ_0. We want to show that it satisfies Γ_0 and $\exists x.C_0(x)$ for every $C_0 \in cls(\Gamma_0)$. By definition, if $\mathcal{T}'_i$ is a consistent subtree of an implication tree $\mathcal{T}$ of (C_0, Γ_0), the root d is an element of $I_0(C_0)$. Hence, $\mathcal{I} \models \exists x.C_0(x)$.

We now show that each formula in Γ_0 is satisfied by $\mathcal{I}$. Let $C(x) \to F(x) \in \Gamma_0$. If $d \in I_0(C)$, then by the definition of canonical interpretation, for some $\Gamma \in \Sigma(\Gamma_0)$, d is labeled with δ where $C \in \bigcup_{C' \in \delta} H(C', \Gamma)$. If $F(x) = C_1(x) \vee \cdots \vee C_m(x)$ $(m \geq 1)$, then for some $i \in \{1, \ldots, m\}$, $C(x) \to C_i(x) \in \Gamma$. Then, $d \in I_0(C_i)$ since $C_i \in \bigcup_{C' \in \delta} H(C', \Gamma)$. If $F(x) = \neg C_1(x) \wedge \cdots \wedge \neg C_m(x)$ $(m \geq 1)$, then $C(x) \to \neg C_1(x), \ldots, C(x) \to \neg C_m(x) \in \Gamma$. So, $\{C_1, \ldots, C_m\} \cap \bigcup_{C' \in \delta} H(C', \Gamma) = \emptyset$ because $\{\neg C_1, \ldots, \neg C_m\} \subseteq \bigcup_{C' \in \delta} H(C', \Gamma)$ and d is not closed. By definition, $d \notin I_0(C_1) \cup \cdots \cup I_0(C_m)$. Hence, $\mathcal{I} \models C(\bar{d}) \to \neg C_1(\bar{d}) \wedge \cdots \wedge \neg C_m(\bar{d})$. If $F(x) = (a(x, y) \to C'(y))$, then consider the two cases for $(d, d') \in I_0(a)$ where (i) the edge (d, d') is labeled with (Γ, a) and (ii) there is a witness d_0 of d, the edge (d_0, d') is labeled with (Γ, a), and d has a child node labeled with w. Let $\delta_a = E(\delta, a, \Gamma)$. For (i), due to $\delta_a \neq \emptyset$, d' is labeled with δ_a and $C' \in \delta_a$. Thus, $d' \in I_0(C')$ since $C' \in \bigcup_{C'' \in \delta_a} H(C'', \Gamma)$. For (ii), by definition, $(d_0, d') \in I_0(a)$, the child node d' of d_0 is labeled with δ_a, and $C' \in \delta_a$. So, $C' \in \bigcup_{C'' \in \delta_a} H(C'', \Gamma)$ implies $d' \in I_0(C')$. Moreover, we have to consider the case where $(d, e_1), \ldots, (d, e_{k-1}) \in I_{d,a}(a)$. By definition, there exists $(d, d') \in I_0(a)$. Since d' is a non-leaf node labeled with δ_a and $C' \in \bigcup_{C'' \in \delta_a} H(C'', \Gamma)$, we have $d' \in I_0(C')$ and it implies $e_1, \ldots, e_{k-1} \in I_{d,a}(C')$. Hence, $\mathcal{I} \models C(\bar{d}) \rightarrow (a(\bar{d}, y) \rightarrow C'(y))$. If $F(x) = \exists_{\geq i} z.a(x, z)$, then since d is not closed, the following two cases are considered. If the node d has a child node d' such that d' is a non-leaf node and (d, d') is an edge labeled with (Γ, a). Thus, $(d, d') \in I_0(a)$. By definition, $(d, e_1), \ldots, (d, e_{k-1}) \in I_{d,a}(a)$ where $k = Max_{\geq}(N(\delta, a, \Gamma)) \geq i$. If the node d has a child labeled with w, then there is a witness d_0 of d and (d_0, d') is labeled with (Γ, a). By definition, $(d, d') \in I_0(a)$, and hence $(d, e_1), \ldots, (d, e_{k-1}) \in I_{d,a}(a)$ where $k = Max_{\geq}(N(\delta, a, \Gamma)) \geq i$. It follows $\mathcal{I} \models \exists_{\geq i} z.a(\bar{d}, z)$. If $F(x) = \exists_{\leq j} z.a(x, z)$, then since d is not closed, there is no implication form $C(x) \rightarrow^{*} \exists_{\geq i} z.a(x, z) \in \Gamma$ such that $i > j$. It derives $Max_{\geq}(N(\delta, a, \Gamma)) \leq j$. By definition, $|\{(d, d')\} \cup \{(d, e_1), \ldots, (d, e_{k-1})\}| \leq j$. Hence, $\mathcal{I} \models \exists_{\leq j} z.a(\bar{d}, z)$.

Let $e_j \in I_{d,a}(C)$ where $1 \leq j \leq k-1$. By definition, there exists a node d labeled with δ such that $C \in \bigcup_{C' \in \delta} H(C', \Gamma)$. So, $d \in I_0(C)$. If $F(x) = C_1(x) \vee \cdots \vee C_m(x)$ $(m \geq 1)$, then for some $C_i \in \{C_1, \ldots, C_m\}$, $C(x) \rightarrow C_i(x) \in \Gamma$. By $C_i \in \bigcup_{C' \in \delta} H(C', \Gamma)$, $d \in I_0(C_i)$, and hence $e_1, \ldots, e_{k-1} \in I_{d,a}(C_i)$. Also, if $F(x) = \neg C_1(x) \wedge \cdots \wedge \neg C_m(x)$ $(m \geq 1)$, then $C(x) \rightarrow \neg C_1(x), \ldots, C(x) \rightarrow \neg C_m(x) \in \Gamma$. Due to $d \notin I_0(C_1) \cup \cdots \cup I_0(C_m)$, $\{e_1, \ldots, e_{k-1}\} \cap I_{d,a}(C_1) \cap \cdots \cap I_{d,a}(C_m) = \emptyset$. Hence, $\mathcal{I} \models C(\bar{e_j}) \rightarrow \neg C_1(\bar{e_j}) \wedge \cdots \wedge \neg C_m(\bar{e_j})$. Similarly, we can prove it for the cases where $F(x) = (a(x, y) \rightarrow C'(y))$, $\exists_{\geq i} z.a(x, z)$, and $\exists_{\leq j} z.a(x, z)$.

Let $e_{(v,w)} \in I_{d,A}(C)$. Then, there exists $d \in I_{d',A'}(C_k)$ such that $AC(A, \Gamma_0) = (C_1, \ldots, C_k, \ldots, C_n)$ and $Max_{\geq}(N_k(H(A, \Gamma_0), \Gamma_0)) = (i_1, \ldots, i_{k-1}, i_{k+1}, \ldots, i_n)$. By definition, $e_{(v,w)} \in I_{d,A}(C_v)$ with $C \in H(C_v, \Gamma')$ where $v \in \{1, \ldots, n\} \backslash \{k\}$ and C_v is consistent in Γ'. So, $e_{(v,w)} \in I_{d,A}(C')$ for all $C' \in H(C, \Gamma')$. If $F(x) = C_1(x) \vee \cdots \vee C_m(x)$ $(m \geq 1)$, then for some $C_i \in \{C_1, \ldots, C_m\}$, $C(x) \rightarrow C_i(x) \in \Gamma$. Then, $e_{(v,w)} \in I_0(C_i)$ by $C_i \in H(C, \Gamma')$. If $F(x) = \neg C_1(x) \wedge \cdots \wedge \neg C_m(x)$ $(m \geq 1)$, then for any $\Gamma' \in \Sigma(\Gamma_0)$, $C(x) \rightarrow \neg C_1(x), \ldots, C(x) \rightarrow \neg C_m(x) \in \Gamma'$. Since C_v is consistent in Γ', $H(C, \Gamma')$ does not contain any inconsistent pair C_i and $\neg C_i$. So, $\{C_1, \ldots, C_m\} \cap H(C, \Gamma') = \emptyset$. This derives $e_{(v,w)} \notin I_{d,A}(C_i)$. So, $\mathcal{I} \models \neg C_1(\bar{e}_{(v,w)}) \wedge \cdots \wedge \neg C_m(\bar{e}_{(v,w)})$. If $F(x) = (a(x, y) \rightarrow C'(y))$, then for every $(e_{(v,w)}, d'') \in I_{d,A}(a)$, there exists d'' such that $(d', d'') \in I_0(a)$ and $d' \in I_0(C_v)$. By the above proof, $d'' \in I_0(C')$. Hence, $\mathcal{I} \models a(\bar{e}_{(v,w)}, \bar{d''}) \rightarrow C'(\bar{d''}))$. If $F(x) = \exists_{\geq i} z.a(x, z)$, then for every $(e_{(v,w)}, d'') \in I_{d,A}(a)$, there exists d' such that $(d', d'') \in I_0(a)$ and $d' \in I_0(C_v)$. By the above proof, we have $\mathcal{I} \models \exists_{\geq i} z.a(\bar{d'}, z)$. By the definition of canonical interpretation, $\mathcal{I} \models \exists_{\geq i} z.a(\bar{e}_{(v,w)}, z)$. Similarly, if $F(x) = \exists_{\leq j} z.a(x, z)$, then $\mathcal{I} \models \exists_{\leq j} z.a(\bar{e}_{(v,w)}, z)$.

Let $e_0 \in I_{d,A}(C)$. Similar to the case $e_{(v,w)} \in I_{d,A}(C)$.

Let $C_k(x) \rightarrow \exists_{\geq i_1} x_1 \cdots \exists_{\geq i_{k-1}} x_{k-1}\ \exists_{\geq i_{k+1}} x_{k+1} \cdots \exists_{\geq i_n} x_n.A(x_1, \ldots, x_n)\ [x_k/x]$ in Γ_0. Due to $Assoc(\Gamma_0) = 1$, there exists no $i_v > j_v$ such that

$$\{(\geq i_1, \ldots, \geq i_{k-1}, \geq i_{k+1}, \ldots, \geq i_n), (\leq j_1, \ldots, \leq j_{k-1}, \leq j_{k+1}, \ldots, \leq j_n)\} \subseteq N_k(H(A, \Gamma_0), \Gamma_0).$$

For each $v \in \{1, \ldots, n\} \backslash \{k\}$, $\{e_{(v,1)}, \ldots, e_{(v,i_v)}\} \subseteq U_{d,A}$ and $i_v \geq i'_v$ where $AC(A, \Gamma_0) = (C_1, \ldots, C_k, \ldots, C_n)$ and $Max_{\geq}(N_k(H(A, \Gamma_0), \Gamma_0)) = (i'_1, \ldots, i'_{k-1}, i'_{k+1}, \ldots, i'_n)$. If $d \in I(C_k)$, then by definition, for all $(w_1, \ldots, w_{k-1}, w_{k+1}, \ldots, w_n)$ with $1 \leq w_v \leq i'_v$, $(e_{(1,w_1)}, \ldots, e_{(k-1,w_{k-1})}, d, e_{(k+1,w_{k+1})}, \ldots, e_{(n,w_n)}) \in I_{d,A}(A)$. Therefore,

$$\mathcal{I} \models C_k(\bar{d}) \rightarrow \exists_{\geq i_1} x_1 \cdots \exists_{\geq i_{k-1}} x_{k-1} \exists_{\geq i_{k+1}} x_{k+1} \cdots \exists_{\geq i_n} x_n.A(x_1, \ldots, x_n)[x_k/\bar{d}].$$

Similarly, if $e \in I(C_k)$ where e is e_0, e_j, or $e_{(x,y)}$, then

$$\mathcal{I} \models C_k(\bar{e}) \rightarrow \exists_{\geq i_1} x_1 \cdots \exists_{\geq i_{k-1}} x_{k-1} \exists_{\geq i_{k+1}} x_{k+1} \cdots \exists_{\geq i_n} x_n.A(x_1, \ldots, x_n)[x_k/\bar{e}].$$

Let $C_k(x) \to \exists_{\leq j_1} x_1 \cdots \exists_{\leq j_{k-1}} x_{k-1}\, \exists_{\leq j_{k+1}} x_{k+1} \cdots \exists_{\leq j_n} x_n.\ A(x_1, \ldots, x_n)[x_k/x]$ in Γ_0. For each $v \in \{1, \ldots, n\} \backslash \{k\}$, $\{e_{(v,1)}, \ldots, e_{(v,i_v)}\} \subseteq U_{d,A}$ and $i_v < j_v$ (due to $Assoc(\Gamma_0) = 1$) where $AC(A, \Gamma_0) = (C_1, \ldots, C_k, \ldots, C_n)$ and $Max_{\geq}(N_k(H(A,\Gamma_0), \Gamma_0)) = (i_1, \ldots, i_{k-1}, i_{k+1}, \ldots, i_n)$. If $d \in I(C_k)$, then by definition, for all $(w_1, \ldots, w_{k-1}, w_{k+1}, \ldots, w_n)$ with $1 \leq w_v \leq i_v$, $(e_{(1,w_1)}, \ldots, e_{(k-1,w_{k-1})}, d, e_{(k+1,w_{k+1})}, \ldots, e_{(n,w_n)}) \in I_{d,A}(A)$. Hence, $\mathcal{I} \models C_k(\bar{d}) \to \exists_{\leq j_1} x_1 \cdots \exists_{\leq j_{k-1}} x_{k-1}\, \exists_{\leq j_{k+1}} x_{k+1} \cdots \exists_{\leq j_n} x_n.\ A(x_1, \ldots, x_n)[x_k/\bar{d}]$. Similarly, if $e \in I(C_k)$ where e is e_0, e_j, or $e_{(x,y)}$, then

$$\mathcal{I} \models C_k(\bar{e}) \to \exists_{\leq j_1} x_1 \cdots \exists_{\leq j_{k-1}} x_{k-1} \exists_{\leq j_{k+1}} x_{k+1} \cdots \exists_{\leq j_n} x_n . A(x_1, \ldots, x_n)[x_k/\bar{e}].$$

Let $A(x_1, \ldots, x_n) \to A'(x_1, \ldots, x_n) \in \Gamma_0$. If $(u_1, \ldots, u_n) \in I_{d,A}(A)$, then by definition, $(u_1, \ldots, u_n) \in I_{d,A}(A')$. If $(u_1, \ldots, u_n) \in I_{d,A''}(A)$ with $A \neq A''$, then $A''(x_1, \ldots, x_n) \to^* A(x_1, \ldots, x_n)$ in Γ_0. By definition, $(u_1, \ldots, u_n) \in I_{d,A''}(A')$.

Let $A(x_1, \ldots, x_n) \to C_1(x_1) \wedge \cdots \wedge C_n(x_n) \in \Gamma_0$. If $(u_1, \ldots, u_n) \in I_{d,A}(A)$, then by definition $u_1 \in I_{d,A}(C_1), \ldots, u_n \in I_{d,A}(C_n)$. If $(u_1, \ldots, u_n) \in I_{d,A'}(A)$ with $A \neq A'$ with $A \neq A''$, then $A'(x_1, \ldots, x_n) \to^* A(x_1, \ldots, x_n) \in \Gamma_0$, $AC(A', \Gamma_0) = (C'_1, \ldots, C'_n)$, $u_1 \in I_{d,A'}(C'_1), \ldots, u_n \in I_{d,A'}(C'_n)$, and $\{C'_1(x) \to^* C_1(x), \ldots, C'_n(x) \to^* C_n(x)\} \subseteq \Gamma_0$. By definition $u_1 \in I_{d,A'}(C_1), \ldots, u_n \in I_{d,A'}(C_n)$.

Let $A(x_1, \ldots, x_n) \to (r_0(x_1, \ldots, x_n, z) \to C_A(z)) \in \Gamma_0$. If $(u_1, \ldots, u_n) \in I_{d,A}(A)$, then for every $(u_1, \ldots, u_n, e_0) \in I_{d,A}(r_0)$, $e_0 \in I_{d,A}(C_A)$.

Let $A(x_1, \ldots, x_n) \to \exists_{=1} z . r_0(x_1, \ldots, x_n, z) \in \Gamma_0$. If $(u_1, \ldots, u_n) \in I_{d,A}(A)$, then $(u_1, \ldots, u_n, e_0) \in I_{d,A}(r_0)$. Since the element e_0 is introduced for $(u_1, \ldots, u_n) \in I_{d,A}(A)$, $|\{e_0 \mid (u_1, \ldots, u_n, e_0) \in I_{d,A}(r_0)\}| = 1$. Hence, $\mathcal{I} \models \exists_{=1} z . r_0(\bar{d}_1, \ldots, \bar{d}_n, z)$.

Let $\exists_{\leq 1} z.(C_A(z) \wedge r_0(x_1, \ldots, x_n, z)) \in \Gamma_0$. Then, there must exist $A(x_1, \ldots, x_n) \to \exists_{=1} z . r_0(x_1, \ldots, x_n, z) \in \Gamma_0$. Hence, for any $(u_1, \ldots, u_n) \in U^n$, if $(u_1, \ldots, u_n) \in I_{d,A}(A)$, then $(u_1, \ldots, u_n, e_0) \in I_{d,A}(r_0)$ and $e_0 \in I_{d,A}(C_A)$, otherwise, there exits no e_0 such that $(u_1, \ldots, u_n, e_0) \in I_{d,A}(r_0)$ and $e_0 \in I_{d,A}(C_A)$. Therefore, $\mathcal{I} \models \exists_{\leq 1} z.(C_A(z) \wedge r_0(x_1, \ldots, x_n, z))$. ∎

The correctness for the algorithms *Cons* and *Assoc* is obtained as follows:

Theorem 13 (Completeness) *Let D be a UML class diagram with association generalization and without roles, and let $\mathcal{G}(D)$ be the translation of D into a set of implication forms. D is consistent if and only if $Cons(\{C\}, \emptyset, \mathcal{G}(D)) = 1$ for all $C \in cls(\mathcal{G}(D))$ and $Assoc(\mathcal{G}(D)) = 1$.*

Proof. ($\Rightarrow$) Suppose $\mathcal{G}(D)$ has a UML-model. Then, by the definition of UML-models, for all $C \in cls(\mathcal{G}(D))$, $\mathcal{G}(D) \models \exists x.C(x)$. By Lemma 9 and Lemma 12, for all $C \in cls(\mathcal{G}(D))$, $Cons(\{C\}, \emptyset, \mathcal{G}(D)) = 1$ and $Assoc(\Gamma_0) = 1$.

($\Leftarrow$) By Lemma 9 and Lemma 12, there is an interpretation $\mathcal{I}$ such that for every $C \in cls(\mathcal{G}(D))$, $\mathcal{I} \models \exists x.C(x)$. It follows that a UML-model of D exists. ∎

Theorem 14 (Termination) *The consistency checking algorithm Cons terminates.*

Proof. The conditions $\delta_a \neq \emptyset$ and $\delta_a \notin \Delta$ in the algorithm lead to the termination. In the worst case, Δ contains all the classes in $cls(\Gamma_0)$ but it must be a finite set. ∎

Theorem 15 (Complexity) *The algorithm Cons computes the consistency of $\mathcal{D}^-_{ful}$ in 2EXPTIME.*

Proof. Suppose that $|\Gamma_0| = m$. Then, $|cls(\Gamma_0)| \leq m$ and $|att(\Gamma_0)| \leq m$. Let D be a class diagram in $\mathcal{D}^-_{ful}$ and let $\mathcal{G}(D)$ be the translation of D. The algorithm *Cons* contains the loops for all $\Gamma \in \Sigma(\mathcal{G}(D))$ and $a \in att(\Gamma_0)$. Moreover, the number of recursive calls is the number of subsets of the set $cls(\mathcal{G}(D))$ of classes in Δ that is exponential. So, the total number of recursive calls is $|2^{cls(\mathcal{G}(D))}|^{|\Sigma(\mathcal{G}(D))| \times |att(\Gamma_0)|} = 2^{2m \cdot 2^m}$ where each call is computed in at most $m^2 + m$ steps due to $|\bigcup_{C \in \delta} H(C, \Gamma)| \leq m^2$ and $|N(\delta, a, \Gamma)| \leq m$. Therefore, the consistency checking for every class is computable in $m \times (m^2 + m) \times 2^{2m \cdot 2^m}$ steps in the worst case. ∎

In Theorems 14 and 15, the proposed consistency checking algorithm *Cons* terminates; however, it still exhibits a double-exponential complexity in the worst case (and *Assoc* exhibits polynomial time complexity).

5 Algorithms and complexities for various expressivities

In this section, we will present optimized consistency checking algorithms for class diagrams of different expressive powers.

5.1 Restriction of inconsistency triggers

We denote the set of UML class diagrams with association generalization and without roles as $\mathcal{D}^-_{ful}$. By deleting certain inconsistency triggers, we classify UML class diagrams that are less expressive than $\mathcal{D}^-_{ful}$. The least set $\mathcal{D}^-_0$ of class diagrams is obtained by deleting disjointness/completeness constraints and overwriting/multiple inheritances. We define $\mathcal{D}^-_{dis}$, $\mathcal{D}^-_{com}$, and $\mathcal{D}^-_{inh}$ as extensions of $\mathcal{D}^-_0$ by adding disjointness constraints, completeness constraints, and overwriting/multiple inheritances, respectively. We denote $\mathcal{D}^-_{dis+com}$, $\mathcal{D}^-_{dis+inh}$, and $\mathcal{D}^-_{inh+com}$ as the unions of $\mathcal{D}^-_{dis}$ and $\mathcal{D}^-_{com}$, $\mathcal{D}^-_{dis}$ and $\mathcal{D}^-_{inh}$, and $\mathcal{D}^-_{inh}$ and $\mathcal{D}^-_{com}$, respectively.

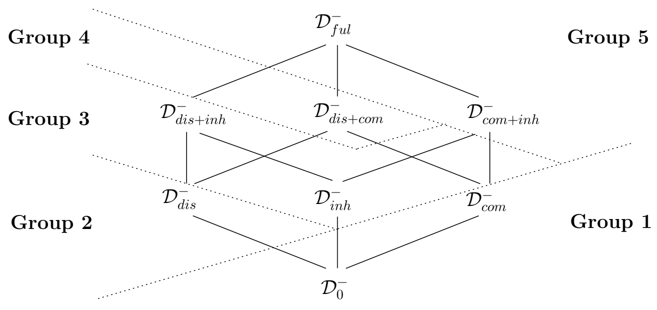

Figure 9: Classification of UML class diagrams

In order to design algorithms that are suitable for these expressivities, we divide the class diagrams into five groups, as shown in Figure 9. Group 1 comprises the least expressive class diagrams obtained by deleting disjointness constraints and overwriting/multiple inheritances (but allowing attribute multiplicities). Groups 2 and 3 prohibit the form $C_1(x) \vee \cdots \vee C_m(x)$ as disjunctive classes by deleting completeness constraints. Furthermore, Group 2 contains no overwriting/multiple inheritances. Group 4 is restricted by eliminating overwriting/multiple inheritances (but allowing disjointness constraints, completeness constraints, and attribute multiplicities).

Given a real UML class diagram, we need to select a group to which the diagram belongs in order to apply an optimized algorithm to it. One of the five groups is determined by means of which combinations of overwriting/multiple inheritances, disjointness constraints, and completeness constraints are included in the UML class diagram.

Definition 16 (Classification rules of UML class diagrams) *For any UML class diagram D, its belonged group is uniquely selected by the following rules:*

(i) if D contains the disjointness constraint $\{disjoint\}$, then it belongs to Group 2, 3, or 4,

(ii) if D contains the completeness constraint $\{complete\}$, then it belongs to Group 1, 4, or 5,

(iii) if D contains identically named attributes in two classes C_1 and C_2 such that C_1 is a subclass of C_2 or C_1 and C_2 have a common subclass, then it belongs to Group 3 or 5,

(iv) if (i), (ii), or (iii) does not hold, then it belongs to Group 1, and

(v) if (i) - (iv) imply more than one group, then the least group number is selected.

These rules classify any real UML class diagram into a group even if the diagram includes additional expressions beyond the class diagrams defined in the groups.

5.2 Restriction of attribute value types

Apart from the restriction of inconsistency triggers, we naturally restrict attribute value types in the overwriting/multiple inheritances. Consider the class hierarchy in Figure 10. A

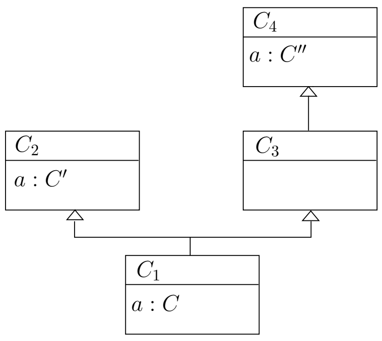

Figure 10: Attribute value types in overwriting/multiple inheritances

class C_1 with attribute $a\colon C$ inherits attributes $a\colon C'$ and $a\colon C''$ from superclasses C_2 and C_4. In this case, if the value type C is a subclass of all the other value types C' and C'' of the identically named attributes in the class hierarchy, the consistency checking of the value types C, C', and C'' can be guaranteed by the consistency checking of only the value type C.

Let $C \in cls(\Gamma_0)$ and let $\Gamma \in \Sigma(\Gamma_0)$. The attribute value types used in class C are said to be *restrictedly defined in* Γ when if the superclasses $C_1, \ldots, C_n$ of C (i.e., $H(C, \Gamma) = \{C_1, \ldots, C_n\}$) have identically named attributes and the attribute value types are classes $C'_1, \ldots, C'_m$, a attribute value type C'_i is a subclass of the other attribute value types $\{C'_1, \ldots, C'_m\} \backslash \{C'_i\}$, i.e., $\{C'_1, \ldots, C'_m\} \subseteq H(C'_i, \Gamma)$. All the attribute value types are restrictedly defined if the attribute value types in any class $C \in cls(\Gamma_0)$ are restrictedly defined in any $\Gamma \in \Sigma(\Gamma_0)$.

Example 2 *As shown in Figure 10, the value types C, C', and C'' of attribute a in class C_1 are restrictedly defined in*

$$\Gamma_1 = \{C_1(x) \to C_2(x), C_1(x) \to C_3(x), C_3(x) \to C_4(x), C_1(x) \to (a(x, y) \to C(y)),$$
$$C_2(x) \to (a(x, y) \to C'(y)), C_4(x) \to (a(x, y) \to C''(y)), \ldots\}$$

if $\{C, C', C''\} \subseteq H(C_0, \Gamma_1)$, where C_0 is C, C', or C''.

The restriction of inconsistency triggers and the restriction of attribute value types are relevant for users to obtain a simple syntax and effective consistency checking in class diagrams. In practice, the users can make the specification of a software system more abstract by excluding attributes and operations or disjointness and completeness constraints. In the restriction of inconsistency triggers, the simplest diagrams become class hierarchies and the other simplified diagrams correspond to one among Groups 1–5 (according to the rules mentioned in Definition 16). Moreover, the restriction of attribute value types in multiple inheritances is realized by two safety design patterns of class diagrams. The first is to prohibit the use of two classes that have identically named attributes and a common subclass in order to avoid any conflict of attribute value types. The second is that users should decide a unique value type for each attribute name, i.e., they must set a general value type for each attribute name. This is a simple way to restrict the attribute value types.

5.3 Optimized algorithms

We show that Group 1 does not cause any inconsistency and devise four consistency checking algorithms *Cons*1–*Cons*4 that are suitably optimized for Groups 2–5 (because *Cons* is not effectively designed for each of the groups). For Groups 2 and 3, we develop the optimized algorithm *Cons*1 for the class diagrams with no completeness constraints. This algorithm does not process any recursive calls but performs looping of consistency checking for unchecked sets of classes. Hence, the computation is limited to polynomial time (when Group 2 or attribute value types are restricted in Group 3). For Group 4, we design the optimized algorithm *Cons*2 for the class diagrams with no overwriting or multiple inheritances. The diagrams in Group 4 do not create complex sets of target classes during the evaluation of attributes because of the absence of overwriting, or multiple inheritances. Even if the completeness constraints expand the searching space exponentially, the depth of a recursive call tree is limited to polynomial size. For Group 5, we develop the two optimized algorithms *Cons*3 and *Cons*4 for the class diagrams with completeness constraints and overwriting and multiple inheritances. The algorithm *Cons*4 is a single exponential time algorithm as an optimization of *Cons* that eliminates redundant steps. The algorithm *Cons*3 can be used to reduce space complexity if attribute value types are restricted in the class diagrams of Group 5.

The optimized algorithm $Cons1$ (in Figure 11) computes the consistency of class diagrams in $\mathcal{D}^{-}_{dis+inh}$, $\mathcal{D}^{-}_{inh}$, and $\mathcal{D}^{-}_{dis}$ (in Groups 2 and 3) by calling $Cons1(\{C_0\}, \emptyset, \Gamma_0)$ for every class $C_0 \in cls(\Gamma_0)$. Let X be a set and Y be a family of sets. Then, we define $ADD(X, Y) = \{X_i \in Y \mid X_i \not\subset X\} \cup \{X\}$ such that X is added to Y and all $X_i \subset X$ are removed from Y. Since $\mathcal{D}^{-}_{dis+inh}$, $\mathcal{D}^{-}_{inh}$, and $\mathcal{D}^{-}_{dis}$ do not contain any completeness constraints, there is a unique decomposed set of Γ_0, namely, $\Sigma(\Gamma_0) = \{\Gamma\}$. Instead of recursive calls, $Cons1$ performs looping of consistency checking for each element of variable P that stores unchecked sets of classes. Moreover, $Cons1$ is optimized by skipping over the sets of classes that have already been checked to be consistent in any former routine. The sets are stored in a good variable set $G = \{\delta_1, \dots, \delta_n\}$ that is a family of sets of classes such that each set δ_i is consistent in a decomposed set of Γ_0 (in $\Sigma(\Gamma_0)$). The condition "$\delta_a, \mu_0(\delta_a, \Gamma) \not\subseteq \delta'$ for all $\delta' \in G$" makes $Cons1$ skip the consistency checking of the target set δ_a if a superset δ' of either δ_a or $\mu_0(\delta_a, a, \Gamma)$ is already checked in former processes (i.e., $\delta' \in G$). The optimization method of using good and no good variable sets G and NG is based on the EXPTIME tableau algorithm presented in reference [5].

```
**Algorithm** $Cons1$ for $\mathcal{D}^{-}_{dis+inh}$, $\mathcal{D}^{-}_{inh}$, and $\mathcal{D}^{-}_{dis}$
**input** set of classes $\delta$, family of sets of classes $\Delta$, set of implication forms $\Gamma_0$
**output** 1 (consistent) or 0 (inconsistent)
**begin**
  $P = \{\delta\}$; $G = \Delta$;
  **while** $P \neq \emptyset$ **do**
    $\delta \in P$; $P = P - \{\delta\}$; $\Gamma \in \Sigma(\Gamma_0)$; $S = \bigcup_{C \in \delta} H(C, \Gamma)$;
    **if** $\{C, \neg C\} \subseteq S$ or $\{t_1, \dots, t_n\} \subseteq S$ s.t. $t_1 \cap \cdots \cap t_n = \emptyset$ **then return** 0;
    **else** $G = ADD(\delta, G)$;
         **for** $a \in att(\Gamma_0)$ **do**
            **if** $i > j$ s.t. $\{\geq i, \leq j\} \subseteq N(\delta, a, \Gamma)$ **then return** 0;
            **else** $\delta_a = E(\delta, a, \Gamma)$;
                 **if** $\delta_a \neq \emptyset$ and $\delta_a, \mu_0(\delta_a, \Gamma) \not\subseteq \delta'$ for all $\delta' \in G$ **then**
                    **if** $\mu(\delta_a, \Gamma) \neq \emptyset$ **then** $\delta_a = \mu_0(\delta_a, \Gamma)$;
                    $P = ADD(\delta_a, P)$;
                 **fi**;
            **esle**;
         **rof**;
    **esle**;
  **elihw**;
  **return** 1;
**end**;
```

Figure 11: The optimized consistency checking algorithm $Cons1$

We need Lemmas 17, 18, and 19 in order to guarantee that the optimized algorithm $Cons1$ preserves the completeness (Theorem 20).

Lemma 17 *Let Γ_0 be a set of implication forms and let $C_0 \in cls(\Gamma_0)$. There is a non-closed implication tree $\mathcal{T}$ of $(\{C_0\}, \Gamma_0)$ if and only if there is a consistent subtree of $\mathcal{T}$.*

Proof. ($\Rightarrow$) Trivial. ($\Leftarrow$) Suppose that there exists no a non-closed implication tree $\mathcal{T}$ of $(\{C_0\}, \Gamma_0)$. Let $\mathcal{T}'$ be a subtree of $\mathcal{T}$ such that each node satisfies Condition (iii) in Definition 10. Then, $\mathcal{T}'$ is closed. Hence, there is no consistent subtree of $\mathcal{T}$. ∎

Lemma 18 *Let Γ_0 be a set of implication forms and let $\delta_0 \subseteq cls(\Gamma_0)$. If there is a non-closed implication tree of (δ_0, Γ_0), then for every $\delta'_0 \subseteq \delta_0$ with $\delta_0 \neq \emptyset$, there is a non-closed implication tree of (δ'_0, Γ_0).*

Proof. Let $\mathcal{T}$ be a non-closed implication tree of (δ_0, Γ_0) and let $\delta'_0 \subseteq \delta_0$ with $\delta_0 \neq \emptyset$. In order to construct a non-closed implication tree $\mathcal{T}'$ of (δ'_0, Γ_0), we use the tree construction in the proof of Lemma 9. Since it terminates, there must exist an implication tree $\mathcal{T}'$ of (δ'_0, Γ_0). For each node d labeled with δ in $\mathcal{T}$, the tree $\mathcal{T}'$ has the corresponding node d' labeled with δ' such that d' has the same path to the root of $\mathcal{T}$. So, $\delta' \subseteq \delta$ because $\delta'_0 \subseteq \delta_0$ and if $\delta'_i \subseteq \delta_i$ then $E(\delta'_i, a, \Gamma) \subseteq E(\delta_i, a, \Gamma)$ for any $a \in att(\Gamma_0)$ and $\Gamma \in \Sigma(\Gamma_0)$. Therefore, there exists a consistent subtree of $\mathcal{T}'$. By Lemma 17, $\mathcal{T}'$ is not closed. ∎

Lemma 19 *Let Γ_0 be a set of implication forms in $\mathcal{D}^-_{dis+inh}$, $\mathcal{D}^-_{inh}$, or $\mathcal{D}^-_{dis}$. For every class $C_0 \in cls(\Gamma_0)$, $Cons1(\{C_0\}, \emptyset, \Gamma_0) = 1$ if and only if there is a non-closed forest of Γ_0.*

Proof. ($\Rightarrow$) Let us assume that for every class $C_0 \in cls(\Gamma_0)$, $Cons1(\{C_0\}, \emptyset, \Gamma_0) = 1$. For each $C_0 \in cls(\Gamma_0)$, we construct a tree $\mathcal{T}$ of $(\{C_0\}, \Gamma_0)$ as follows.

1. Create the root d_0 labeled with $\{C_0\}$.
2. Perform the following operations if a node d labeled with δ is created:
   (a) Create a new node d' labeled with 0 and add the edge (d, d') labeled with Γ if 0 is returned by satisfying the condition $\{C, \neg C\} \subseteq S$ or $\{t_1, \dots, t_n\} \subseteq S$ such that $t_1 \cap \dots \cap t_n = \emptyset$.
   (b) Create a new node d' labeled with 1 and add the edge (d, d') labeled with Γ if 1 is returned by satisfying the conditions $att(\Gamma_0) = \emptyset$, $\{C, \neg C\} \not\subseteq S$, and $\{t_1, \dots, t_n\} \not\subseteq S$ such that $t_1 \cap \dots \cap t_n = \emptyset$,
   (c) Perform the following operations for each $a \in att(\Gamma_0)$ if $att(\Gamma_0) \neq \emptyset$, $\{C, \neg C\} \not\subseteq S$, and $\{t_1, \dots, t_n\} \not\subseteq S$ such that $t_1 \cap \dots \cap t_n = \emptyset$:
      i. Create a new node d' labeled with 0 and add the edge (d, d') labeled with (Γ, a) if 0 is returned by satisfying the condition $i > j$ such that $\{\geq i, \leq j\} \subseteq N(\delta, a, \Gamma)$,
      ii. Create a new node d' labeled with 1 and add the edge (d, d') labeled with (Γ, a) if δ_a is not added to P by satisfying the conditions $E(\delta, a, \Gamma) = \emptyset$ and $i < j$ for any $\geq i, \leq j \in N(\delta, a, \Gamma)$,
      iii. Create a new node d' labeled with w and add the edge (d, d') labeled with (Γ, a) if δ_a is not added to P by satisfying the condition that there exists an ancestor node labeled with $E(\delta, a, \Gamma)$ or $\mu_0(E(\delta, a, \Gamma), \Gamma)$ and $i < j$ for any $\geq i, \leq j \in N(\delta, a, \Gamma)$,
      iv. Add the non-closed implication tree of (δ', Γ_0) and the edge (d, d') labeled with (Γ, a) to the node d if δ_a is not added to P by satisfying the condition that $E(\delta, a, \Gamma) \subseteq \delta'$ or $\mu_0(E(\delta, a, \Gamma), \Gamma) \subseteq \delta'$ with $\delta' \in G$ and there exists no ancestor labeled with $E(\delta, a, \Gamma)$ or $\mu_0(E(\delta, a, \Gamma), \Gamma)$. The non-closed implication tree can be obtained by Lemma 18 because for every $\delta' \in G$, there exists a non-closed implication tree of (δ', Γ_0). Moreover, for every descendant node d'' of d in the added tree, apply the following operations:

```
Algorithm $Cons2$ for $\mathcal{D}^-_{dis+com}$
input class $C_0$, set of implication forms $\Gamma_0$
output 1 (consistent) or 0 (inconsistent)
begin
   for $\Gamma \in \Sigma(\Gamma_0)$ do
      $S = H(C_0, \Gamma)$;
      if $\{C, \neg C\} \not\subseteq S$ and $\{t_1, \ldots, t_n\} \not\subseteq S$ s.t. $t_1 \cap \cdots \cap t_n = \emptyset$ then
         for $a \in att(\Gamma_0)$ do
            if $i > j$ s.t. $\{\geq i, \leq j\} \subseteq N(C_0, a, \Gamma)$ then return 0;
         return 1;
      fi;
   rof;
   return 0;
end;
```

Figure 12: The optimized consistency checking algorithm $Cons2$

A. Replace δ'' with w and delete all the descendant nodes of d'' if there exists an ancestor node labeled with δ'' or $\mu_0(\delta'', \Gamma)$.

B. Recursively apply operation (iv) to the node d'' if d'' is labeled with w such that there exists no ancestor node labeled with $E(\delta'', a, \Gamma)$ or $\mu_0(E(\delta'', a, \Gamma), \Gamma)$.

v. Create a new node d' labeled with δ' and add the edge (d, d') labeled with (Γ, a) if δ_a is added to P by satisfying the conditions that $E(\delta, a, \Gamma) \neq \emptyset$, $\mu_0(E(\delta, a, \Gamma), \Gamma) \not\subseteq \delta'$ and $E(\delta, a, \Gamma) \not\subseteq \delta'$ for any $\delta' \in G$, and $i < j$ for any $\geq i, \leq j \in N(\delta, a, \Gamma)$.

Similar to the proof of Lemma 9, this tree $\mathcal{T}$ satisfies the conditions in Definition 8.

($\Leftarrow$) By the expressivity of $\mathcal{D}^-_{dis+inh}$, $\mathcal{D}^-_{inh}$, or $\mathcal{D}^-_{dis}$, $\Sigma(\Gamma_0) = \{\Gamma\}$ for an implication form set Γ_0. This can be proved similar to the case of the algorithm $Cons$. ∎

The following theorem guarantees that the optimized algorithm $Cons1$ preserves the completeness.

Theorem 20 (Completeness) *Let D be a UML class diagram in $\mathcal{D}^-_{dis+inh}$, $\mathcal{D}^-_{inh}$, or $\mathcal{D}^-_{dis}$, and let $\mathcal{G}(D)$ be the translation of D into a set of implication forms. D is consistent if and only if $Cons1(\{C\}, \emptyset, \mathcal{G}(D)) = 1$ for all $C \in cls(\mathcal{G}(D))$ and $Assoc(\mathcal{G}(D)) = 1$.*

Proof. By Lemmas 12 and 19. ∎

The optimized algorithm $Cons2$ (in Figure 12) computes the consistency of $\mathcal{D}^-_{dis+com}$ (in Group 4) if $Cons2(C_0, \Gamma_0)$ is called for every class $C_0 \in cls(\Gamma_0)$. This algorithm is simply designed for testing the consistency of an input class C_0 in every $\Gamma \in \Sigma(\Gamma_0)$, where the multiplicities of attributes in C_0 are checked but the disjointness of the attribute value types are not. This is because $\mathcal{D}^-_{dis+com}$ involves no overwriting/multiple inheritances, i.e., each attribute value is uniquely typed and if type T is a class (in $cls(\Gamma_0)$), the consistency of T can be checked in another call $Cons2(T, \Gamma_0)$.

We need Lemma 21 in order to guarantee that the optimized algorithm $Cons2$ preserves the completeness (Theorem 22).

Lemma 21 *Let Γ_0 be a set of implication forms in $\mathcal{D}^{-}_{dis+com}$. For every class $C_0 \in cls(\Gamma_0)$, $Cons2(C_0, \Gamma_0) = 1$ if and only if there is a non-closed forest of Γ_0.*

Proof. ($\Rightarrow$) Let us assume that for every class $C_0 \in cls(\Gamma_0)$, $Cons2(C_0, \Gamma_0) = 1$. If we apply $Cons(\{C_0\}, \emptyset, \Gamma_0)$ to each C_0, then since Γ_0 does not contain overwriting/multiple inheritances, every recursively call in the algorithm $Cons(\{C_0\}, \emptyset, \Gamma_0)$ is limited to the calls $Cons(\{C_i\}, \Delta, \Gamma_0)$ where $C_i \in cls(\Gamma_0)$ and $\Delta \subseteq cls(\Gamma_0)$. Therefore, by the assumption, for every class $C_0 \in cls(\Gamma_0)$, $Cons(\{C_0\}, \emptyset, \Gamma_0) = 1$. By Lemma 9, a non-closed forest of Γ_0 exists.

($\Leftarrow$) Let $\mathcal{S}$ be a non-closed forest of Γ_0 and $\mathcal{T}$ be a non-closed implication tree of $(\{C_0\}, \Gamma_0)$ in $\mathcal{S}$. So, the root d (in $\mathcal{T}$) labeled with $\{C_0\}$ is not closed. For each $\Gamma \in \Sigma(\Gamma_0)$, a child d' of d is labeled with 1 and the edge (d, d') is labeled with Γ if $att(\Gamma_0) = \emptyset$, otherwise, for all $a \in att(\Gamma_0)$, a child d_a of d is labeled with a set of classes. Therefore, $Cons2(C_0, \Gamma_0) = 1$. ∎

The following theorem guarantees that the optimized algorithm $Cons2$ preserves the completeness.

Theorem 22 (Completeness) *Let D be a UML class diagram in $\mathcal{D}^{-}_{dis+com}$, and let $\mathcal{G}(D)$ be the translation of D into a set of implication forms. D is consistent if and only if $Cons2(\{C\}, \emptyset, \mathcal{G}(D)) = 1$ for all $C \in cls(\mathcal{G}(D))$ and $Assoc(\mathcal{G}(D)) = 1$.*

Proof. By Lemmas 12 and 21. ∎

The optimized algorithm $Cons3$ (in Figure 13) computes the consistency of $\mathcal{D}^{-}_{com+inh}$ and $\mathcal{D}^{-}_{ful}$ (in Group 5) if we call $Cons3(\{C_0\}, \emptyset, \Gamma_0)$ for every class $C_0 \in cls(\Gamma_0)$. It should be noted that the algorithm $Cons$ requires double exponential time in the worst case. This algorithm is optimized as a single exponential version by skipping the sets of classes that are already checked as consistent or inconsistent in any former routine (but $Cons$ limits the skipping to the set Δ stored in the caller processes). The no good variable set NG is a family of pairs of a set δ of classes and a decomposed set Γ of Γ_0 such that δ is inconsistent in Γ. Each element in NG exactly indicates the inconsistency of δ in the set Γ by storing the pair (δ, Γ), so that it is never checked again.

In addition to this method, we consider that further elements can be skipped by the condition "$\delta_a, \mu_0(\delta_a, \Gamma) \not\subseteq \delta'$ for all $\delta' \in \Delta \cup G$." This implies that $Cons3$ skips the consistency checking of the target set δ_a if a superset δ' of either δ_a or $\mu_0(\delta_a, a, \Gamma)$ is already checked in former processes (i.e., $\delta' \in \Delta \cup G$). With regard to the skipping condition, the following lemma guarantees that if $\mu(\delta, \Gamma) \neq \emptyset$, then all the classes $C_1, \ldots, C_n$ in δ and the sole class C in $\mu_0(\delta, \Gamma)$ ($= \{C\}$) have the same superclasses. In other words, the consistency checking of δ can be replaced with the consistency checking of $\mu_0(\delta, \Gamma)$. Therefore, the computational steps can be decreased by skipping the target set δ_a since this set can be replaced by an already checked superset of the singleton $\mu_0(\delta_a, a, \Gamma)$.

Lemma 23 *Let Γ_0 be a set of implication forms and let $\Gamma \in \Sigma(\Gamma_0)$. For all $\delta \subseteq cls(\Gamma_0)$ and $a \in att(\Gamma_0)$, if $\mu(\delta, \Gamma) \neq \emptyset$, then*

1. $\bigcup_{C \in \delta} H(C, \Gamma) = \bigcup_{C \in \mu_0(\delta, \Gamma)} H(C, \Gamma)$,

```
Algorithm $Cons3$ for $\mathcal{D}^-_{com+inh}$ and $\mathcal{D}^-_{ful}$
input set of classes $\delta$, family of sets of classes $\Delta$, set of implication forms $\Gamma_0$
output 1 (consistent) or 0 (inconsistent)
global variables $G = \emptyset$,  $NG = \emptyset$
begin
  for $\Gamma \in \Sigma(\Gamma_0)$ s.t. $(\delta, \Gamma) \notin NG$ do
    $S = \bigcup_{C\in\delta} H(C, \Gamma)$; $f_\Gamma = 0$;
    if $\{C, \neg C\} \not\subseteq S$ and $\{t_1, \ldots, t_n\} \not\subseteq S$ s.t. $t_1 \cap \cdots \cap t_n = \emptyset$ then $f_\Gamma = 1$;
      for $a \in att(\Gamma_0)$ do
        if $i > j$ s.t. $\{\geq i, \leq j\} \subseteq N(\delta, a, \Gamma)$ then $f_\Gamma = 0$;
        else $\delta_a = E(\delta, a, \Gamma)$;
            if $\delta_a \neq \emptyset$ and $\delta_a, \mu_0(\delta_a, \Gamma) \not\subseteq \delta'$ for all $\delta' \in \Delta \cup G$ then
              if $\mu(\delta_a, \Gamma) \neq \emptyset$ then $\delta_a = \mu_0(\delta_a, \Gamma)$;
              $f_\Gamma = Cons3(\delta_a, \Delta, \Gamma_0)$;
            fi;
        esle;
      rof;
    fi;
    if $f_\Gamma = 1$ then $G = ADD(\delta, G)$; return 1;
    else $NG = ADD((\delta, \Gamma), NG)$;
  rof;
  return 0;
end;
```

Figure 13: The optimized consistency checking algorithm $Cons3$

2. $N(\delta, a, \Gamma) = N(\mu_0(\delta, \Gamma), a, \Gamma)$*, and*

3. $E(\delta, a, \Gamma) = E(\mu_0(\delta, \Gamma), a, \Gamma)$*.*

Proof. Let δ be a set of classes in $cls(\Gamma_0)$.

$C_0 \in \bigcup_{C\in\delta} H(C, \Gamma)$ if and only if there exists $C \in \delta$ such that $C(x) \rightarrow^* C_0(x) \in \Gamma$. By the definition of $\mu(\delta, \Gamma)$, for all $C' \in \mu(\delta, \Gamma)$, $\delta \subseteq H(C', \Gamma)$. Then, for every $C \in \delta$, $C'(x) \rightarrow^* C(x) \in \Gamma$. Hence, $C'(x) \rightarrow^* C_0(x) \in \Gamma$ if and only if $C_0 \in H(C', \Gamma)$. Since $\mu_0(\delta, \Gamma) \subseteq \mu(\delta, \Gamma)$, it implies $C_0 \in \bigcup_{C'_0\in\mu_0(\delta,\Gamma)} H(C', \Gamma)$. Inversely, if $C_0 \in \bigcup_{C'_0\in\mu_0(\delta,\Gamma)} H(C'_0, \Gamma)$, then $C'_0(x) \rightarrow C_0(x) \in \Gamma$ where $\mu_0(\delta, \Gamma) = \{C'_0\}$. By $\mu_0(\delta, \Gamma) \subseteq \delta \subseteq H(C', \Gamma)$ where $C' \in \mu(\delta, \Gamma)$, for all $C' \in \mu(\delta, \Gamma)$, $C'(x) \rightarrow^* C'_0(x) \in \Gamma$. Thus, $C'(x) \rightarrow^* C_0(x) \in \Gamma$. So, $C_0 \in H(C', \Gamma)$ for all $C' \in \mu(\delta, \Gamma)$.

We have that $\leq j \in N(\delta, a, \Gamma)$ (or $\geq i \in N(\delta, a, \Gamma)$) if and only if there exists $C \in \delta$ such that $C(x) \rightarrow^* \exists_{\leq j} z.a(x, z) \in \Gamma$ (or $C(x) \rightarrow^* \exists_{\geq i} z.a(x, z) \in \Gamma$). By the definition of $\mu(\delta, \Gamma)$, for all $C' \in \mu(\delta, \Gamma)$, $\delta \subseteq H(C', \Gamma)$. Then, for every $C \in \delta$, $C'(x) \rightarrow^* C(x) \in \Gamma$. Hence, $C'(x) \rightarrow^* \exists_{\leq j} z.a(x, z) \in \Gamma$ (or $C'(x) \rightarrow^* \exists_{\geq i} z.a(x, z) \in \Gamma$) if and only if $\leq j \in N(C', a, \Gamma)$ (or $\geq i \in N(\delta, a, \Gamma)$). Since $\mu_0(\delta, \Gamma) \subseteq \mu(\delta, \Gamma)$, it implies $\leq j \in N(\mu_0(\delta, \Gamma), a, \Gamma)$ (or $\geq i \in N(\mu_0(\delta, \Gamma), a, \Gamma)$). Inversely, if $\leq j \in N(\mu_0(\delta, \Gamma), a, \Gamma)$ (or $\geq i \in N(\mu_0(\delta, \Gamma), a, \Gamma)$), then $C'_0(x) \rightarrow^* \exists_{\leq j} z.a(x, z) \in \Gamma$ (or $C'_0(x) \rightarrow^* \exists_{\geq i} z.a(x, z) \in \Gamma$) where $\mu_0(\delta, \Gamma) = \{C'_0\}$. By $\mu_0(\delta, \Gamma) \subseteq \delta \subseteq H(C', \Gamma)$ where $C' \in \mu(\delta, \Gamma)$, for all $C' \in \mu(\delta, \Gamma)$, $C' \rightarrow^* C'_0 \in \Gamma$. Thus, $C'(x) \rightarrow^* \exists_{\leq j} z.a(x, z) \in \Gamma$ (or $C'(x) \rightarrow^* \exists_{\geq i} z.a(x, z) \in \Gamma$). So, $\leq j \in N(C', a, \Gamma)$ (or $\geq i \in N(\delta, a, \Gamma)$) for all $C' \in \mu(\delta, \Gamma)$.

$C_0 \in E(\delta, a, \Gamma)$ if and only if there exists $C \in \delta$ such that $C(x) \rightarrow^* (a(x, y) \rightarrow C_0(y)) \in \Gamma$ and $C(x) \rightarrow^* \exists_{\geq i} z.a(x, z) \in \Gamma$ $(i \geq 1)$. By the definition of $\mu(\delta, \Gamma)$, for all $C' \in \mu(\delta, \Gamma)$,

```
Algorithm $Cons4$ for $\mathcal{D}^{-}_{com+inh}$ and $\mathcal{D}^{-}_{ful}$
input set of classes $\delta$, family of sets of classes $\Delta$, set of implication forms $\Gamma_0$
output 1 (consistent) or 0 (inconsistent)
global variables $G = \emptyset$, $NG = \emptyset$
begin
  for $\Gamma \in \Sigma(\Gamma_0)$ do
    $S = \bigcup_{C \in \delta} H(C, \Gamma)$; $f_\Gamma = 0$;
    if $\{C, \neg C\} \not\subseteq S$ and $\{t_1, \dots, t_n\} \not\subseteq S$ s.t. $t_1 \cap \dots \cap t_n = \emptyset$ then $f_\Gamma = 1$;
      for $a \in att(\Gamma_0)$ do
        if $i > j$ s.t. $\{\geq i, \leq j\} \subseteq N(\delta, a, \Gamma)$ then $f_\Gamma = 0$;
        else $\delta_a = E(\delta, a, \Gamma)$;
             if $\delta_a \neq \emptyset$ and $\delta_a, \mu_0(\delta_a, \Gamma) \not\subseteq \delta'$ for all $\delta' \in \Delta \cup G$ then
               if $\mu(\delta_a, \Gamma) \neq \emptyset$ then $\delta_a = \mu_0(\delta_a, \Gamma)$;
               $\underline{\textbf{if } \delta_a \in NG \textbf{ then } f_\Gamma = 0}$;
               else $f_\Gamma = Cons4(\delta_a, \Delta, \Gamma_0)$;
             fi;
        esle;
      rof;
    fi;
    if $f_\Gamma = 1$ then $G = ADD(\delta, G)$; return 1;
  rof;
  $\underline{NG = ADD(\delta, NG)}$; return 0;
end;
```

Figure 14: The optimized consistency checking algorithm $Cons4$

$\delta \subseteq H(C', \Gamma)$. Then, for every $C \in \delta$, $C'(x) \to^* C(x) \in \Gamma$. Hence,

$$C'(x) \to^* (a(x,y) \to C(x)) \in \Gamma \text{ and } C'(x) \to^* \exists_{\geq i} z.a(x,z) \in \Gamma$$

if and only if $C_0 \in E(C', a, \Gamma)$. Since $\mu_0(\delta, \Gamma) \subseteq \mu(\delta, \Gamma)$, it implies $C \in E(\mu_0(\delta, \Gamma), a, \Gamma)$. Inversely, if $C \in E(\mu_0(\delta, \Gamma), a, \Gamma)$, then

$$C'_0(x) \to^* (a(x,y) \to C_0(y)) \in \Gamma \text{ and } C'_0(x) \to^* \exists_{\geq i} z.a(x,z) \in \Gamma$$

where $\mu_0(\delta, \Gamma) = \{C'_0\}$. By $\mu_0(\delta, \Gamma) \subseteq \delta \subseteq H(C', \Gamma)$ where $C' \in \mu(\delta, \Gamma)$, for all $C' \in \mu(\delta, \Gamma)$, $C'(x) \to^* C'_0(x) \in \Gamma$. Thus, $C'(x) \to (a(x,y) \to^* C_0(y)) \in \Gamma$ and $C'(x) \to^* \exists_{\geq i} z.a(x,z) \in \Gamma$. So, $C_0 \in E(C', a, \Gamma)$ for all $C' \in \mu(\delta, \Gamma)$. ∎

We adjust the algorithm *Cons*3 to class diagrams in which all the attribute value types are restrictedly defined. The optimized algorithm *Cons*4 is shown in Figure 14; as indicated by the underlined text, this algorithm is improved by only storing the sets of classes in *NG* (similar to G). The restriction of value types leads to $\mu(\delta_a, \Gamma) \neq \emptyset$; therefore, the size of *NG* is limited to a set of singletons of classes. In other words, *Cons*4 can be adjusted to decrease the space complexity (i.e., *NG*) to polynomial space by using the property of Lemma 23. Unfortunately, this adjustment does not yield a single exponential algorithm if the attribute value types are *unrestrictedly* defined. Hence, we need both *Cons*3 and *Cons*4 for the cases where the attribute value types are restrictedly and unrestrictedly defined.

We need Lemma 24 in order to guarantee that the optimized algorithms *Cons*3 and *Cons*4 preserve the completeness (Theorem 25).

Lemma 24 *Let Γ_0 be a set of implication forms in $\mathcal{D}^-_{com+inh}$ or $\mathcal{D}^-_{ful}$. For every class $C_0 \in cls(\Gamma_0)$, $Cons3(\{C_0\}, \emptyset, \Gamma_0) = 1$ (or $Cons4(\{C_0\}, \emptyset, \Gamma_0) = 1$) if and only if there is a non-closed forest of Γ_0.*

Proof. ($\Rightarrow$) Let us assume that for every class $C_0 \in cls(\Gamma_0)$, $Cons3(\{C_0\}, \emptyset, \Gamma_0) = 1$ (or $Cons4(\{C_0\}, \emptyset, \Gamma_0) = 1$). For each $C_0 \in cls(\Gamma_0)$, we construct a tree $\mathcal{T}$ of $(\{C_0\}, \Gamma_0)$ as follows.

1. Create the root d_0 labeled with $\{C_0\}$.
2. Perform the following operations if a node d labeled with δ is created:
   (a) Create a new node d' labeled with 0 and add the edge (d, d') labeled with Γ if $f_\Gamma = 0$ is kept by satisfying the conditions $\{C, \neg C\} \subseteq S$ or $\{t_1, \ldots, t_n\} \subseteq S$ with $t_1 \cap \cdots \cap t_n = \emptyset$, and $(\delta, \Gamma) \notin NG$ (or $\Gamma \notin NG$).
   (b) Create a new node d' labeled with 1 and add the edge (d, d') labeled with Γ if 1 is returned by satisfying the conditions $att(\Gamma_0) = \emptyset$, $\{C, \neg C\} \not\subseteq S$, $\{t_1, \ldots, t_n\} \not\subseteq S$ with $t_1 \cap \cdots \cap t_n = \emptyset$, and $(\delta, \Gamma) \notin NG$ (or $\Gamma \notin NG$).
   (c) Perform the following operations for each $a \in att(\Gamma_0)$ if $att(\Gamma_0) \neq \emptyset$, $\{C, \neg C\} \not\subseteq S$, $\{t_1, \ldots, t_n\} \not\subseteq S$ with $t_1 \cap \cdots \cap t_n = \emptyset$, and $(\delta, \Gamma) \notin NG$ (or $\Gamma \notin NG$):
      i. - iii. Perform the same operations as the tree construction in the proof of Lemma 9.
      iv. Perform the same operations in the proof of Lemma 19.
      v. Create a new node d' labeled with δ' and add the edge (d, d') labeled with (Γ, a) if δ_a is added to P by satisfying the conditions that $E(\delta, a, \Gamma) \neq \emptyset$, $E(\delta, a, \Gamma) \not\subseteq \delta'$ (or $\mu_0(E(\delta, a, \Gamma), \Gamma) \not\subseteq \delta'$ by Lemma 23) for any $\delta \in G$, there exists no ancestor labeled with $E(\delta, a, \Gamma)$ (or $\mu_0(E(\delta, a, \Gamma), \Gamma)$ by Lemma 23), and $i < j$ for any $\geq i, \leq j \in N(\delta, a, \Gamma)$.
   (d) Add all the children d' of d such that the edge (d, d') is labeled with Γ or (Γ, a) and their descendants to the node d if $(\delta, \Gamma) \in NG$ (or $\Gamma \in NG$). Moreover, for every descendant node d'' of d that is labeled with δ_i, apply the following operations:
      i. Replace δ_i with w and delete all the descendant nodes of d'' if d'' is labeled with δ_i such that there exists an ancestor labeled with δ_i (or $\mu_0(\delta_i, \Gamma)$ by Lemma 23).
      ii. Recursively apply operation (v) to the node d'' if d'' is labeled with w such that there exists no ancestor labeled with $E(\delta_i, a, \Gamma)$.

Similar to the proof of Lemma 9, this tree $\mathcal{T}$ satisfies the conditions in Definition 8.
($\Leftarrow$) Similar to the case of the algorithm *Cons*. ∎

The following theorem guarantees that the optimized algorithms *Cons*3 and *Cons*4 preserve the completeness.

Theorem 25 (Completeness) *Let D be a UML class diagram in $\mathcal{D}^-_{com+inh}$ or $\mathcal{D}^-_{ful}$, and let $\mathcal{G}(D)$ be the translation of D into a set of implication forms. D is consistent if and only if $Cons3(\{C\}, \emptyset, \mathcal{G}(D)) = 1$ (or $Cons4(\{C\}, \emptyset, \mathcal{G}(D)) = 1$) for all $C \in cls(\mathcal{G}(D))$ and $Assoc(\mathcal{G}(D)) = 1$.*

Proof. By Lemmas 12 and 24. ∎

5.4 Upper-bound complexities

Without losing the completeness of consistency checking, the optimized algorithms *Cons*1 – *Cons*4 have the following computational properties for all the class diagram groups (as shown in Table 1).

Theorem 26 (Complexities)

1. *Every class diagram in $\mathcal{D}_0^-$ and $\mathcal{D}_{com}^-$ is consistent.*
2. *The algorithm Cons*1 *computes the consistency of $\mathcal{D}_{dis}^-$ in polynomial time and computes the consistency of $\mathcal{D}_{inh}^-$ and $\mathcal{D}_{dis+inh}^-$ in EXPTIME. If every attribute value type is restrictedly defined, then it computes the consistency of $\mathcal{D}_{inh}^-$ and $\mathcal{D}_{dis+inh}^-$ in polynomial time.*
3. *The algorithm Cons*2 *computes the consistency of $\mathcal{D}_{dis+com}^-$ in NP.*
4. *The algorithm Cons*3 *computes the consistency of $\mathcal{D}_{com+inh}^-$ and $\mathcal{D}_{ful}^-$ in EXPTIME. If every attribute value type is restrictedly defined, then the algorithm Cons*4 *computes the consistency of $\mathcal{D}_{com+inh}^-$ and $\mathcal{D}_{ful}^-$ in PSPACE.*

Proof. Suppose that $|\Gamma_0| = m$. Then, $|cls(\Gamma_0)| \leq m$ and $|att(\Gamma_0)| \leq m$.

(1) Let D be a class diagram in $\mathcal{D}_0^-$ or $\mathcal{D}_{com}^-$ and let $\mathcal{G}(D)$ be the translation of D. The class diagram does not contain disjointness constraints nor overwriting/multiple inheritances. By the expressivity, there exist no disjoint classes in $\mathcal{G}(D)$, every class inherits no more than one attribute of the same name (i.e., for each $\Gamma \in \Sigma(\mathcal{G}(D))$, $N(H(C,\Gamma), a, \Gamma)$ has the two elements denoting the multiplicity of one attribute such as $\{\geq i, \leq j\}$ with $i > j$), and every class in associations has no multiplicities if multiplicities are already defined in classes of the super-associations. Therefore, if $Cons(\{C_0\}, \emptyset, \Gamma_0)$ for all $C_0 \in cls(\mathcal{G}(D))$ and $Assoc(\mathcal{G}(D))$ are called, then they cannot find any inconsistency. That is, $Cons(\{C_0\}, \emptyset, \Gamma_0) = 1$ for all $C_0 \in cls(\mathcal{G}(D))$ and $Assoc(\Gamma_0) = 1$, and by Theorem 13, D is consistent.

(2) Let D be a class diagram in $\mathcal{D}_{dis}^-$ and let $\mathcal{G}(D)$ be the translation of D. Let us assume that the algorithm $Cons1(C_0, \Gamma_0)$ for all $C_0 \in cls(\mathcal{G}(D))$ is called. Then, the number of loops is decided by the variable P where P is a subset of the power set of $cls(\mathcal{G}(D))$. Each loop for elements in P performs to check disjointness in class-hierarchies (whether the set of superclasses and disjoint classes $\bigcup_{C\in\delta} H(C,\Gamma)$ contains an inconsistent pair C' and $\neg C'$) and to check the conflicted multiplicities of the identically named attributes for every $a \in att(\Gamma_0)$. They are computable in at most $2^m \times (m + m^2)$ steps. Moreover, any class diagram in $\mathcal{D}_{dis}^-$ does not contain overwriting/multiple inheritances, so that the variable P is limited to a set of singletons of classes, precisely, $\mu_0(\delta, \Gamma)$ is added to P by applying $P = ADD(\mu_0(\delta,\Gamma), P)$ where $\mu_0(\delta,\Gamma)$ is the singleton of a class. The number of loops is at most the number of classes in $cls(\mathcal{G}(D))$, and hence the algorithm computes the consistency in at most $m \times (m + m^2)$ steps. We have to consider that the algorithm $Cons1(\{C_0\}, \emptyset, \mathcal{G}(D))$ for all $C_0 \in cls(\mathcal{G}(D))$ is called. Therefore, the complexity totally becomes $\mathcal{O}(m^4)$.

Let D be a class diagram in $\mathcal{D}_{inh}^-$ or $\mathcal{D}_{dis+inh}^-$ and let $\mathcal{G}(D)$ be the translation of D. The class diagrams in $\mathcal{D}_{inh}^-$ and $\mathcal{D}_{dis+inh}^-$ contain overwriting/multiple inheritances, so that the variable P is a subset of the power set of $cls(\mathcal{G}(D))$ by applying $P = ADD(\delta_a, P)$ where $\delta_a = E(\delta, a, \Gamma)$ is a set of classes. Therefore, the number of loops is exponential in the worst case. Moreover, it is clear that $|\bigcup_{C \in \delta} H(C, \Gamma)| \leq m$, $|N(\delta, a, \Gamma)| \leq m$, and $|E(\delta, a, \Gamma)| \leq m$. Each loop is bounded by at most $m + m^2$ steps. Hence, this algorithm is implemented by using at most $\mathcal{O}(2^m)$ steps. When every attribute value type is restrictedly defined in D, if a class $C \in cls(\mathcal{G}(D))$ has attributes, then the type C_0 of an attribute in the class C is a subclass of the types $C_1, \ldots, C_n$ of other attributes in the class C such that $E(H(C, \Gamma), a, \Gamma) = \{C_0, C_1, \ldots, C_n\}$ and $\{C_1, \ldots, C_n\} \subseteq H(C_0, \Gamma)$. Due to $\mu(\delta, \Gamma) \neq \emptyset$, P contains only the singleton of a class by applying $P = ADD(\delta_a, P)$ where $\delta_a = \mu_0(\delta, \Gamma)$. Similar to the proof of $\mathcal{D}^-_{dis}$, the number of loops is bounded by m steps. Hence, the consistency is computable in polynomial time.

(3) Let D be a class diagram in $\mathcal{D}^-_{dis+com}$ and let $\mathcal{G}(D)$ be the translation of D. Let us assume that $Cons2(C_0, \mathcal{G}(D))$ for all $C_0 \in cls(\mathcal{G}(D))$ is called. First, a decomposed set $\Gamma \in \Sigma(\mathcal{G}(D))$ is non-deterministically chosen. Next, it checks disjointness in class-hierarchies (for $H(C_0, \Gamma)$) and checks the multiplicities of the identically named attributes for every $a \in att(\Gamma_0)$. For each Γ, they are computable in at most $m \times (m + m^2)$ steps. Since we need to call the algorithm $Cons2(C_0, \mathcal{G}(D))$ for all $C_0 \in cls(\mathcal{G}(D))$, the consistency of D is decided non-deterministically in $\mathcal{O}(m^4)$ steps.

(4) Let D be a class diagram in $\mathcal{D}^-_{com+inh}$ or $\mathcal{D}^-_{ful}$ and let $\mathcal{G}(D)$ be the translation of D. The algorithm $Cons3$ recursively calls itself in the loops for all $\Gamma \in \Sigma(\mathcal{G}(D))$ and $a \in att(\Gamma_0)$. The number of recursive calls is decreased by the two conditions in the algorithm $Cons3$ that are (i) $\Gamma \in \Sigma(\Gamma_0)$ such that $(\delta, \Gamma) \notin NG$ and (ii) $\delta_a \not\subseteq \delta'$ for all $\delta' \in \Delta \cup G$. With respect to (i), each $(\delta, \Gamma) \in 2^{cls(\mathcal{G}(D))} \times \Sigma(\mathcal{G}(D))$ is added to NG if it causes inconsistency, otherwise δ is added to G. So, the total number of recursive calls is bounded by at most $|2^{cls(\mathcal{G}(D))}| \times |\Sigma(\mathcal{G}(D))| = 2^{m^2}$ where each call is computed in at most $m^2 + m$ steps due to $|\bigcup_{C \in \delta} H(C, \Gamma)| \leq m^2$ and $|N(\delta, a, \Gamma)| \leq m$. Therefore, the consistency checking for every class is computable in at most $m \times (m^2 + m) \times 2^{m^2}$ steps, i.e., $\mathcal{O}(2^{m^2})$.

Next we show that if every attribute value type is restrictedly defined in class diagrams of $\mathcal{D}^-_{ful}$, then the consistency checking of the algorithm $Cons4$ is computable by using at most polynomial size memory (i.e., it belongs to PSPACE). The total number of recursive calls is bounded by single exponential time, precisely, at most $|att(\Gamma_0)| \times |cls(\mathcal{G}(D))| \times |\Sigma(\mathcal{G}(D))| = 2m \times 2^m$. The restricted attribute value types imply $\mu(\delta_a, \Gamma) \neq \emptyset$ for any $a \in att(\Gamma_0)$ and $\Gamma \in \Sigma(\Gamma_0)$. So, the depth of recursive calls is bounded by at most m. In the recursive calls, the trace and the variables G, NG, and Δ have to be stored. When $Cons4(\delta_a, \Delta, \Gamma_0)$ is recursively called, δ_a is a set of singletons of classes. So, G, NG, and Δ can be stored by using at most $3m^2$ bits because they are sets of singletons of classes (i.e., $|G| \leq m$, $|NG| \leq m$, and $|\Delta| \leq m$). Moreover, we can reuse space to store each decomposed set $\Gamma \in \Sigma(\Gamma_0)$, that is, it is sufficient that each loop stores one element of $\Sigma(\Gamma_0)$. Hence, this algorithm is implemented by using $\mathcal{O}(m^2)$ bits. ∎

We believe that the complexity classes 0, P, NP, and PSPACE, whose complexities are lower than that of EXPTIME, are suitable for implementing the algorithms for different expressive powers of class diagram groups. For all the class diagram groups, the column "complexity1" in Table 1 shows the complexities of the algorithms $Cons1$, $Cons2$, and $Cons3$ with respect to the size of the class diagram. Every class diagram in $\mathcal{D}^-_0$ and $\mathcal{D}^-_{com}$ is consistent; therefore, the complexity is zero (i.e., we do not need to check consistency).

$Cons1$ computes the consistency of $\mathcal{D}^{-}_{dis}$ in P (polynomial time) and that of $\mathcal{D}^{-}_{inh}$ and $\mathcal{D}^{-}_{dis+inh}$ in EXPTIME (exponential time). $Cons2$ computes the consistency of $\mathcal{D}^{-}_{dis+com}$ in NP (non-deterministic polynomial time), and $Cons3$ computes the consistency of $\mathcal{D}^{-}_{com+inh}$ and $\mathcal{D}^{-}_{ful}$ in EXPTIME.

Moreover, the column "complexity2" in Table 1 shows the complexities of the algorithms $Cons1$, $Cons2$, and $Cons4$ for the cases in which all the attribute value types are restrictedly defined. In particular, $Cons1$ computes the consistency of $\mathcal{D}^{-}_{inh}$ and $\mathcal{D}^{-}_{dis+inh}$ in P, and $Cons4$ computes the consistency of $\mathcal{D}^{-}_{com+inh}$ and $\mathcal{D}^{-}_{ful}$ in PSPACE (polynomial space). Therefore, according to Lemma 23 and by the skipping of consistency checking, the complexities of $Cons1$ and $Cons4$ are reduced from EXPTIME to P and PSPACE, respectively.

Remark. We discuss the complexities of our algorithms with respect to the depth of the class hierarchies. The complexity results in Theorem 26 depend on the number of classes in the diagram because consistent and inconsistent sets of classes in the good and no good variables are restored in the loops or recursive calls in the algorithms. In the proof of Theorem 26, it is shown that the number of classes in $cls(\Gamma_0)$ determines the complexity of all algorithms. In particular, the size of P in $Cons1$ leads to polynomial time complexity, and the sizes of G, NG, and Δ in $Cons4$ lead to polynomial space complexity. Therefore, the complexities are not changed even when they are measured on the basis of the number of classes in the diagram (instead of the size of the diagram). Furthermore, if the complexities are analyzed with respect to the depth of the class hierarchies, then the algorithms result in the same complexities. This is because in the worst case, the depth of the class hierarchies corresponds to the number of classes.

6 Conclusion and future work

We introduced the restriction of UML class diagrams based on

(i) inconsistency triggers (disjointness constraints, completeness constraints, and overwriting/multiple inheritances) and

(ii) attribute value types defined with restrictions in overwriting/multiple inheritances.

Inconsistency triggers are employed to classify the expressivity of class diagrams, and their combination with the attribute value types results in tractable consistency checking of the restricted class diagrams. First, we presented a complete algorithm for testing the consistency of class diagrams that includes any inconsistency triggers. Second, the algorithm was suitably refined in order to develop optimized algorithms for different expressive powers of class diagrams that are obtained by deleting some of the inconsistency triggers. Our algorithms were easily modified depending on the presence of diagram components. From the algorithms, we clarified that it is necessary for all the class diagrams in $\mathcal{D}^{-}_{0}$ and $\mathcal{D}^{-}_{com}$ to have a UML model (i.e., consistency is guaranteed); we further clarified that when every attribute value type is restrictedly defined, the complexities of class diagrams in $\mathcal{D}^{-}_{inh}$ and $\mathcal{D}^{-}_{dis+inh}$ and in $\mathcal{D}^{-}_{com+inh}$ and $\mathcal{D}^{-}_{ful}$ essentially decrease from EXPTIME to P and PSPACE, respectively. In this study, we classified UML class diagrams and developed optimized algorithms for testing the consistencies of restricted UML class diagrams

Table 1: Upper-bound complexities of algorithms for testing consistency

UML group	complexity1	algorithm	complexity2	algorithm
$\mathcal{D}_0^-$	**0**		**0**	
$\mathcal{D}_{com}^-$	**0**		**0**	
$\mathcal{D}_{dis}^-$	**P**	$Cons1$	**P**	$Cons1$
$\mathcal{D}_{inh}^-$	EXPTIME		**P**	
$\mathcal{D}_{dis+inh}^-$	EXPTIME		**P**	
$\mathcal{D}_{dis+com}^-$	**NP**	$Cons2$	**NP**	$Cons2$
$\mathcal{D}_{com+inh}^-$	EXPTIME	$Cons3$	**PSPACE**	$Cons4$
$\mathcal{D}_{ful}^-$	EXPTIME		**PSPACE**	

Our future research is concerned with the average-case complexity for consistency checking. Furthermore, we intend to perform an experimental evaluation to ascertain the applicability of optimized consistency algorithms.

References

[1] A. Artale, D. Calvanese, R. Kontchakov, V. Ryzhikov, and M. Zakharyaschev. Reasoning over extended er models. In *Proc. of the 26th Int. Conf. on Conceptual Modeling (ER 2007)*, volume 4801 of *LNCS*, pages 277–292. Springer, 2007.

[2] B. Beckert, U. Keller, and P. H. Schmitt. Translating the object constraint language into first-order predicate logic. In *Proceedings of VERIFY, Workshop at Federated Logic Conferences (FLoC)*, pages 113–123, 2002.

[3] D. Berardi, D. Calvanese, and G. De Giacomo. Reasoning on UML class diagrams. *Artificial Intelligence*, 168(1-2):70–118, 2005.

[4] F. M. Donini. Complexity of reasoning. In *Description Logic Handbook*, pages 96–136, 2003.

[5] F. M. Donini and F. Massacci. EXPTIME tableaux for ALC. *Artificial Intelligence*, 124(1):87–138, 2000.

[6] A. S. Evans. Reasoning with UML class diagrams. In *Proceedings of the Second IEEE Workshop on Industrial Strength Formal Specification Techniques, WIFT'98, USA*, pages 102–113, 1998.

[7] M. Fowler. *UML Distilled: A Brief Guide to the Standard Modeling Object Language.* Object Technology Series. Addison-Wesley, third edition, September 2003.

[8] E. Franconi and G. Ng. The i.com tool for intelligent conceptual modeling. In *Proceedings of the 7th International Workshop on Knowledge Representation meets Databases (KRDB 2000)*, pages 45–53, 2000.

[9] K. Kaneiwa and K. Satoh. Consistency checking algorithms for restricted UML class diagrams. In *Proceedings of the Fourth International Symposium on Foundations of Information and Knowledge Systems (FoIKS2006)*, pages 219–239. LNCS 3861, Springer–Verlag, 2006.

[10] P. G. Kolaitis and J. A. Väänänen. Generalized quantifiers and pebble games on finite structures. *Annals of Pure and Applied Logic*, 74(1):23–75, 1995.

[11] H. Mannila and K.-J. Räihä. On the complexity of inferring functional dependencies. *Discrete Applied Mathematics*, 40(2):237–243, 1992.

[12] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual.* Addison-Wesley, Reading, Massachusetts, USA, 1st edition, 1999.

[13] K.-D. Schewe and B. Thalheim. Fundamental concepts of object oriented databases. *Acta Cybern*, 11(1-2):49–84, 1993.

[14] A. Tsiolakis and H. Ehrig. Consistency analysis between UML class and sequence diagrams using attributed graph gammars. In *Proceedings of joint APPLIGRAPH/ GETGRATS Workshop on Graph Transformation Systems*, pages 77–86, 2000.