Towards Class Diagram Algebra for Composing Data Models

Hidekazu ENJO \textsuperscript{a} and Junichi IIJIMA \textsuperscript{b}

\textsuperscript{a} NTT DATA CORPORATION, Japan
\textsuperscript{b} Tokyo Institute of Technology, Japan

Abstract. A large set of partial data models is used in designing a large information system. These partial data models provide several complementary views on the system to be developed. This however leads to a need for compositional models that are able to produce a single integrated model. These data models are often described by a class diagram of Unified Modeling Language because it is a very popular modeling language and describing a static view of a system. In this paper, we present syntax and semantics of a class diagram describing a data model. We propose a family of well-formed class diagrams as a domain of class diagram algebra and composition operations as merge and difference operations. We then show that algebraic properties as associativity, commutativity and involutivity are desired for model management to develop a large information system.

Keywords. Class diagram algebra, data modeling, model composition, Unified Modeling Language

1. Introduction

In order to design a large information system to handle huge complex information, it is important to analyze a lot of requirements and make a model of big complicated information. However, it is difficult to describe a big and complicated data model of an information system into one diagram. A large set of partial data models are used in designing an information system for a large information because each application consisting of an information system uses only a part of big and complicated data in most cases. These partial data models provide several complementary views on the system to be developed. This however leads to a need for compositional models that are able to produce a single integrated model.

There are several diagram methods to support data modeling. The Unified Modeling Language (UML)[14] is a popular standard modeling language, especially for object oriented design. A class diagram, which is a type of UML diagrams, is describing a static view of data model. Class diagrams are describing partial data models in designing an information service system.

In this paper, we describe a motivating scenario in which model compositions are used for developing systems and discuss model composition operations – merge and difference as examples and desired algebraic properties. Then we define syntax and semantics of class diagrams, a family of well-formed class diagrams and good property operations as merge, difference formally. We show that algebraic structure of a family of
well-formed class diagrams and composition operations has good algebraic properties as Boolean algebra. At the end, we add appendix of supplement definitions and lemmas and proofs of theorems and lemmas.

2. Model Composition in System Developments

Structured analysis is still popular for developing large information systems and views a system from the perspective of the data flowing through it. The function of the system is described by processes that transform the data flows. One of the result objects of structured analysis is data flow diagram. Data flow diagrams are directed graphs and arcs represent data and nodes represent processes that transform data. A process is able to be decomposed to a more detailed data flow diagram that explains subprocesses and data flows within it. The subprocesses are able to be decomposed with another set of data flow diagrams until subprocesses become primitives. A data flow diagram shows what kinds of data will be input to and output from the system, where the data will come from and go to, and where the data will be stored. Data modelers define models of all data in data flow diagrams for satisfying requirements and taking implementation of processes and subprocesses into consideration for analyzing relations between input data and output data of processes and subprocesses.

For example, one of the typical applications in information service systems is transaction processing. Figure 1 is an example of data flow diagram for transaction processing. A transaction processing application gathers and integrates business transaction data coming from customers into a business information base. And it also retrieves business data from a business information base and shows them to a customer. For designing transaction processing applications, modelers analyze requirements for the applications and create several data models for input transaction data, a business information base, output business data and so on. Then, for integrating business transactions, transaction processing applications are designed to convert instances of input transaction data model to an instance of a business information base and to insert the converted instance into the business information base. If it is possible by consideration of modelers to insert instances of input transaction data model into the business information base directly, it makes the applications simple by eliminating conversion. For generating output business
data, transaction processing applications are also designed to retrieve instances from a business information base and to convert the retrieved instances to an instance of output business data model. It also makes the applications simple by reducing conversion parts, if instances of output business data model are created by only deleting unnecessary parts from the retrieved instances. Unnecessary parts are instances of complemented model of output business data. It means that some general operations as insert and delete operations have a kind of relationships between input data models and an output data model. If modelers keep these relationships for creating data models, it is possible to make applications simple.
Figure 5. Example of an instance of Class Diagram Y

Figure 6. Class Diagram U — Result of Merging Class Diagram X and Class Diagram Y

Figure 7. Example of an instance of Class Diagram U

At the thought of data model, a class diagram is composed of a set of classes and associations. Meaning of a class diagram is a blueprint to create a set of objects and links as an instance of those classes and associations. For example, a set of objects and links in Fig. 3 is an instance of class diagram X in Fig. 2 and a set of objects and links in Fig. 5 is an instance of class diagram Y in Fig. 4. If an instance of class diagram X in Fig. 3 is inserted into an instance of class diagram Y in Fig. 5, the result of insertion is an instance of class diagram U in Fig. 7. This case needs no conversion because a set of objects and links in Fig. 3 is an instance of class diagram X and also a part of objects and links of
class diagram U in Fig. 7 and a set of objects and links in Fig. 5 is an instance of class diagram Y and also a part of objects and links of class diagram U in Fig. 7. At the thought of model composition, it is easy to merge two parts of a model as class diagram X in Fig. 2 and class diagram Y in Fig. 4 and the result is class diagram U in Fig. 6. Because same classes are merged into one class in a result diagram and same associations are merged into one association of that. However, if a target data model is different from the data model of insertion result, some kinds of conversion are needed for fitting a target data model.

Processes and subprocesses on data flow diagrams are related to a kind of model composition operations and data flow diagrams are related to model composition expression. There are many sequences of merging if many class diagrams are merged to generate a big class diagram. For example, one sequence is merging a diagram A and a diagram B then merging a diagram C such as \((A \cdot B) \cdot C\). Another sequence is merging a diagram B and a diagram C then merging a diagram A such as \((B \cdot C) \cdot A\). If a merge operation on a good domain is associative and commutative, results of all merging sequences are same. A merge operation is preferred commutative and associative because it prevents mistakes of modelers to do different sequences or combinations.

At the thought of deletion found in generating output business data, for example, a part of an instance of class diagram U in Fig. 7 is deleted by an instance of class diagram Y in Fig. 5. the result of deletion is an instance in Fig. 3 of class diagram X. There exist the instances “Oo1:Order,” “Oo2:Order,” “Ood1:Order Details,” and “Ood2:Order Details” of class “Order” and “Order Details” in Fig. 2 until after deletion. Because remained links “lp1:PO” and “lp2:PO” need these objects. At the thought of model composition, difference of class diagrams is analogized as difference of set but a little different from it. Class diagram U in Fig. 6 includes class “Order” and class “Order Details” and class diagram Y in Fig. 4 include same classes. Class diagram X in Fig. 2 is the result of difference of class diagram Y relative to class diagram U such as \(U - Y\). Class diagram X needs to include “Order” and “Order Details” because association “PO” consists of “Order,” “Order Details” and “Buyer” classes.

Another way to divide class diagram U results in class diagram Z in Fig. 8 and class diagram Y. Merging class diagram Z and class diagram Y, the result is class diagram U. Difference of class diagram Z relative to class diagram U is class diagram Y such as \(U - Z\). The result of a combination of difference such as \((U - (U - Z))\) is class diagram X. If a complement operation on a good domain is involutive, results are same. A complement operation defined by a difference operation is preferred involutive.
3. Related Work

Model composition is one of the widespread research issues for UML in the area of model driven engineering (MDE)\cite{10}. Object Management Group\cite{14} defined Package Merge to merge UML diagrams including class diagrams. This operation is to merge classes and associations. However, they did not define it mathematically and it is unclear to algebraic structure.

There are several other studies\cite{2,3,7,15} discussed merge operations for model composition. Bézivin, et al.\cite{2} discussed three model composition tools such as Atlas Model Weaver, the Epsilon Merging Language and the Glue Generator Tool which were developed in Modelware project. It derives some common definitions from these discussions and clarifies some basic requirements for model composition tools and frameworks. These works address mainly syntactic properties and their implementation in tools and do not discuss algebraic structure. Boronat, et al.\cite{3} discussed a generic semantics of the merge operation developed in MOMENT project. It describes three steps of model merging: finding semantic equivalences, conflict resolution and copying non-duplicated elements. It concentrates on expressing semantic equalities by means of a metamodel. However, they focused manipulation of a merge operation Sabetzadeh and Easterbrook\cite{15} studied the merging of class diagrams to gain a unified perspective as an algebra of merging incomplete and inconsistent graph-based views. Category theory and colimits serve as theoretical bases to express the relationships between inconsistent and incomplete class diagrams. The basic intention consists in the identification of equal elements in different views. However, they have not been clear up detail of algebraic structure for a merge operation.

Some other studies\cite{4,8,9,12} discussed an operation set for model composition. Brunet, et al.\cite{4} presented a more theoretical view on different model management operations. It introduces algebraic properties of model merge such as commutativity, associativity and idempotency. The theoretical results are illustrated by two examples, merging entity relationship models and state machines, respectively. They concentrated a general overview of model management operations and their relationships and did not discuss the algebra of model composition in detail. Herrmann, et al.\cite{12} studied the semantic properties of model composition in detail. They discussed semantic properties formally and introduced semantic properties of model composition such as property preserving, fully property preserving, consistency preserving, neutral, absorbing, commutativity, associativity and idempotency. However, they discussed requirements for model management operations and did not mention concrete operations of model composition.

Enjo, Tanabu and Iijima\cite{7,8,9} studied model composition and discussed syntactical inconsistency issues. They proposed algebraic operations such as merge and difference in terms of first order logic for preventing inconsistency. They cleared up some algebraic properties of merge and difference operations such as associativity and commutativity of merge and involutivity of difference on the domain of class diagram syntax. However, they did not discuss semantics of model composition except a merge operation and did not clear up algebraic structure of semantics.

Following studies are related semantics of UML class diagrams. Berardi, Calvanese, and Giacomo\cite{1} presented the correspondence between class diagrams and Description Logics, which enable us to utilize Description Logics based systems for reasoning on class diagrams. They discussed inconsistency related disjoint generalizations with rea-
soning of Description Logics. Szlenk[16] presented the mathematical definition of a single class diagram including multiplicity issues. Kaneiwa and Satoh[13] proposed optimized algorithms that computed respective consistencies for class diagrams based on first order predicate logic. For discussing inconsistency of syntax errors and disjointness and/or completeness of generalizations, they proposed mathematical foundation of class diagram. They discussed on one class diagram only and did not mention algebraic structure that was relationship among class diagrams.

Following studies are related algebra for classes. Calvanese, Lenzerini, and Nardi[6] surveyed four type of knowledge representations – description logic, frame based system, semantic data model, object-oriented data model and propose unifying framework. Bueherr[5] formalized object-oriented data model based on Boolean algebra and query based on set theory. They discussed algebra for classes but did not discussed associations that were mandatory components of class diagrams.

4. Notation

We use basic mathematical notation but fix some notations as followed: Name denotes a set of names of all elements consist of a class diagram including a class name, an association name, an attribute name and a role name. DataType denotes a set of data types represented by {Int, Char, Bool, String, Date, . . . }, where types of all data such as integer, character and so on. A multiplicity (r, s) denotes a pair of lower bound r and upper bound s, where r and s are integers and 0 ≤ r ≤ s, 1 ≤ s. In open-ended cases, mark “*” is described at upper bound s. Multiplicity denotes the set of all multiplicities. ID denotes the set of all identifiers that identify objects and links. \( \mathcal{A} \) denotes the set of all subsets of \( \mathcal{A} \). \( 2^{\mathcal{X} \times \mathcal{Y}} \) denotes the set of all pairs as \( 2^{\mathcal{X}} \times 2^{\mathcal{Y}} \). \( \mathcal{X}^+ \) denotes all of any length of lists of \( \mathcal{X} \) as \( \bigcup_{n>0} \mathcal{X}^n = \mathcal{X}^1 \cup \mathcal{X}^2 \cup \ldots \). \( \mathcal{X}^0 \) denotes an empty list or all of any length of lists of \( \mathcal{X} \) as ( ) \( \mathcal{X}^+ \).

5. Syntax of a Class Diagram

A class diagram is a type of UML diagrams and employed to model concepts in static views for an information structure consisting of classes and their interrelationships. There are many definitions of class diagram[1,13,16] but concepts of class diagram are as same as class, attribute, association and class diagram. Following definition is same concept but simplified enough for following discussions. A class diagram has been represented by graphical and abstract syntax. We focus abstract syntax for simplifying discussions. We define the abstract syntax of the class diagram as followed:

**Definition 1 (Class)** A class \( c \) is a triple \((c, attr, op) \in Name \times (Name \times DataType \times Multiplicity)^+ \times (Name \times DataType^+ \times DataType)^+ \) where \( c \) is a class name, \( attr \) is a finite list of attributes and \( op \) is a finite list of operations. An attribute is a triple of an attribute name, a data type and a multiplicity. An operation is a triple of an operation name, data types for parameters and a data type of a result.
Class denotes the set of all classes. Type\((c, i)\) denotes a data type of \(i\)-th attribute of a class \(c\). \(C, C_X, C_* \ldots\) denote sets of classes \( \subseteq \text{Class} \). Figure 9 is an example of a class \(\text{Buyer}\).

**Definition 2 (Association)** An association \(a\) is a pair \((aname, assocs) \in \text{Name} \times (\text{Name} \times \text{Multiplicity} \times \text{Class})^+\) where \(aname\) is an association name and \(assocs\) is a finite list of at least two association ends. An association end is a triple of a role name, a multiplicity and a related class participating in the association.

\(\text{Association}\) denotes the set of all associations. \(\text{Assocs}(a)\) denotes a finite list of association ends of an association \(a\). \(A, A_X, A_* \ldots\) denote sets of associations \( \subseteq \text{Association} \). \(\text{AC}(a, i)\) denotes a class of \(i\)-th association end of an association end list \(\text{Assocs}(a)\). \(J_a\) denotes number of an association end list \(\text{Assocs}(a)\). Aggregations and compositions are a particular kind of binary associations that multiplicity of one end is \((0,1)\) or \((1,1)\). Figure 10 is an example of an association \(\text{PO}\).

In this paper, we omit generalizations for simplifying following discussions.

**Definition 3 (Class Diagram)** A class diagram is a pair \((C, A)\), where \(C\) is a subset of \(\text{Class}\) and \(A\) is a subset of \(\text{Association}\). We assume that a class diagram \((C, A)\) satisfies following condition 1.

**Condition 1 (Closed Class Diagram)** All associated classes of an association are included in the set of classes of the class diagram.

\[
\forall a \in A, r \in \text{Name}, m \in \text{Multiplicity}, ac \in \text{Class},
\forall (r, m, ac) \in \text{Assocs}(a) \Rightarrow ac \in C
\]  

(1)
ClassDiagram denotes the set of all class diagrams. Figure 11 is an example of a class diagram.

6. Semantics of a Class Diagram

A class diagram is composed of a set of classes and associations. An instance of a class is called an object and an instance of an association is called a link. A link is a connection between two or more objects of the classes usually. Meaning of a class diagram is a blueprint to create a set of objects and links as an instance of those classes and associations.

6.1. A Domain for the Semantics of a Class Diagram

We introduce objects, links and complex data as a domain and interpretation mapping for the semantics of class diagram.

Definition 4 (Object) An object o is a pair \((x, ((v_{1,1}, \ldots, v_{1,j_1}), \ldots, (v_{i,1}, \ldots, v_{i,j_i})))\) ∈ \(1D \times Value^*\) where \(x \in 1D\) is an identifier such as \(O1\), \(v_{1,1}, \ldots, v_{1,j_1}, \ldots, v_{i,1}, \ldots, v_{i,j_i}\) ∈ Values are values, \(i, j_1, \ldots, j_i\) are integers and greater than and equal to 0. \((v_{1,1}, \ldots, v_{i,j_i})\) is a finite value list such as \((Yamada, Suzuki)\) and \(((v_{1,1}, \ldots, v_{1,j_1}), \ldots, (v_{i,1}, \ldots, v_{i,j_i}))\) is a finite list of value list such as \((Yamada, Suzuki), (030009999))\).

Objects denotes the set of all objects. Figure 12 is an example of an object.
Definition 5 (Link) A link \( l \) is a pair \((y, (o_1, \ldots, o_k)) \in ID \times Objects^+\), where \( k \) is integer and greater than 1, \( y \in ID \) is an identifier such as \( L1 \), \( o_1, \ldots, o_k \) are objects or \( 0 \)s, \((o_1, \ldots, o_k)\) is a finite list of objects that includes at least one object.

Lists denotes the set of all links. Figure 13 is an example of a link.

Definition 6 (Complex Data) A complex data is a pair \((O, L) \in 2^{(Objects, Links)}\), where a set \( O \subseteq Objects \) is a set of objects, a set \( L \subseteq Links \) is a set of links. We assume that a complex data satisfies following condition 2 that means all objects relating to all links \( L \) are included in objects \( O \).

Condition 2 (Closed Complex Data) Let \((O, L)\) be a complex data. Let \( O \) be a subset of all objects Objects and \( L \) be a subset of all links Links. All objects related to links \( L \) are included in objects \( O \).

\[
\forall(y, (o_1, \ldots, o_k)) \in L, \forall i, 1 \leq i \leq k, o_i \in O
\]

ComplexData denotes the set of all complex data as main part of domain of class diagram semantics. Figure 14 is an example of complex data.

6.2. Interpretation Mappings for Class Diagrams

In a straightforward way, we define interpretation mappings for data type, class and association as following.

Definition 7 (Interpretation Mapping of Data Type) An interpretation mapping \( I_M(t) \mapsto P_t(t) \) of a data type named \( t \) is a mapping to a predicate \((P_t(t))(v)\) as \( v \in V_t \), where \( V_t \) is all values of data type named \( t \).

\[
(P_t(t))(v) := (v \in V_t)
\]
\[(\{O1, ((Yamada, Suzuki), (030009999)), (O2, ((2008 – 08 – 26)), (O3, ((PC, Laptops), (1))), (O4, ((Linux), (1))), (L1, (O1, ((Yamada, Suzuki), (030009999))),(O2, ((2008 – 08 – 26)), (O3, ((PC, Laptops), (1))), (L2, (O1, ((Yamada, Suzuki), (030009999))), (O2, ((2008 – 08 – 26)), (O4, ((Linux), (1))))))\)}

**Figure 14.** Example of a complex data

\[(P(Buyer))(x, ((v_{1,1}, \ldots, v_{1,j1}), \ldots, (v_{i,1}, \ldots, v_{i,j_i}))) := (x \in ID) \land (1 \leq j_i) \land \bigwedge_{1 \leq k \leq j_i} (P(String)(v_{1,k})) \land (P(String)(v_{2,1}))\]

**Figure 15.** Example of a predicate for a class Buyer

For example, data type Date is mapped to a predicate \(P_{\text{Date}}\), where \((P_{\text{Date}})\) ("2009-12-20") is true.

**Definition 8 (Interpretation Mapping of Class)** An interpretation mapping \(I_M(c) \mapsto P_c(c)\) of a class \(c\) is a mapping to a predicate \((P_c(c))(z)\), where numbers of attribute are within relevant lower and upper bounds \((r_k, s_k)\), and values \(v_{1,1}, \ldots, v_{1,j_1}, v_{i,1}, \ldots, v_{i,j_i}\) of attributes are members of data types related attributes.

\[
(P_c(c))(x, ((v_{1,1}, \ldots, v_{1,j_1}), \ldots, (v_{i,1}, \ldots, v_{i,j_i}))) := (x \in ID) \land \bigwedge_{1 \leq k \leq i} (r_k \leq j_k \leq s_k) \land \bigwedge_{1 \leq n \leq j_k} (P_c(\text{Type}(c,k))(v_{k,n})) \quad (4)
\]

\(P_c[C]\) denotes a set of predicates for objects mapped from a set \(C\) of classes as \(\{P_c(c) | c \in C\}\). \(P_c[\text{Class}]\) denotes all predicates for objects mapped from all classes Class. Figure 15 is an example of a predicate for a class Buyer.

**Definition 9 (Interpretation Mapping of Association)** An interpretation mapping \(I_M(a) \mapsto (P_a(a), P_m(a))\) of an association \(a\) is a mapping to a pair of predicates \((P_a(a))(l)\) and \((P_m(a))(o, L)\), where \((P_a(a))(l)\) predicates that a link \(l \in \text{Links}\) is an instance of an association \(a\) and \((P_m(a))(o, L)\) predicates that an object \(o \in \text{Objects}\) is an instance of \(k\)-th association end class and a number of objects related each object \(o\) among links \(L\) is within lower and upper bounds \((r_k, s_k)\).
\[(P_a(PO))([y, (x_1, x_2, x_3)]) := (y \in ID) \land (P_{\text{Buyer}})(x_1) \land (P_{\text{Order}})(x_2) \land (P_{\text{OrderDetails}})(x_3)\]

**Figure 16.** Example of a predicate for an association PO

\[(P_m(PO))(o, L) := (o \in \text{Objects}) \land (L \subseteq \text{Links}) \land ((\forall l \in L, (P_m(PO))(l)) \Rightarrow (((P_{\text{Buyer}}))(o) \Rightarrow \exists_{x_1, y_1}, ((y, (o, x_2, x_3)) \in L \land (P_m(PO))(y, (o, x_2, x_3))) \land ((P_{\text{Order}}))(o) \Rightarrow \exists_{x_1, y_1}, ((y, (x_1, o, x_3)) \in L \land (P_m(PO))(y, (x_1, o, x_3))) \land ((P_{\text{OrderDetails}}))(o) \Rightarrow \exists_{x_1, y_1}, ((y, (x_1, x_2, o)) \in L \land (P_m(PO))(y, (x_1, x_2, o)))) \land ((P_{\text{OrderDetails}}))(o) \Rightarrow \exists_{x_1, y_1}, ((y, (x_1, x_2, o)) \in L \land (P_m(PO))(y, (x_1, x_2, o))))\]

**Figure 17.** Example of a predicate for multiplicity of an association PO

\[(P_a(a))([y, (x_1, \ldots, x_i)]) := (y \in ID) \land \bigwedge_{1 \leq k \leq i} (P_m(AC(a, k)))(x_k) \quad (5)\]

\[(P_m(a))(o, L) := (o \in \text{Objects}) \land (L \subseteq \text{Links}) \land (\bigvee_{l \in L} (P_m(a))(l)) \Rightarrow \bigwedge_{1 \leq k \leq P_m(a)} (((P_m(AC(a, k))))(o) \Rightarrow \exists_{r_1, x_1 \ldots \exists_{r_k, x_k}, x_k-1 \exists_{r_k+1, x_k+1}, x_k \ldots \exists_{r_n, x_n}, y, ((y, (x_1, \ldots, x_k-1, o, x_{k+1}, \ldots, x_n)) \in L) \land (y \in ID) \land (P_a(a))(y, (x_1, \ldots, x_k-1, o, x_{k+1}, \ldots, x_n))) \land ((P_m(AC(a, k))))(o) \Rightarrow \exists_{s_1, x_1 \ldots \exists_{s_k, x_k}, x_k-1 \exists_{s_k+1, x_{k+1}}, x_k \ldots \exists_{s_n, x_n}, y, ((y, (x_1, \ldots, x_k-1, o, x_{k+1}, \ldots, x_n)) \in L) \land (y \in ID) \land (P_a(a))(y, (x_1, \ldots, x_k-1, o, x_{k+1}, \ldots, x_n)))\]

\(P_m[A]\) denotes a set of predicates for links mapped from a set \(A\) of associations as \(\{P_m(a) | a \in A\}\). \(P_m[A]\) denotes a set of predicates for multiplicities of associations mapped from a set \(A\) of classes as \(\{P_m(a) | a \in A\}\). \(P_m[\text{Association}]\) denotes all predicates for links mapped from all associations \(\text{Association}\). \(P_m[\text{Association}]\) denotes all predicates for multiplicities of associations mapped from all associations \(\text{Association}\). Figure 16 and Fig. 17 are examples of predicates for an association.

We define an interpretation mapping of class diagram as getting together interpretation mappings mapped from classes and associations.
\[(P_D((C, A)))((O, L)) := ((O, L) \in ComplexData) \land \\
(\bigwedge_{o \in O} (P_c(Buyer))(o) \lor (P_c(Order))(o) \lor (P_c(OrderDetails))(o)) \land \\
(\bigwedge_{l \in L} (P_m(PO))(l)) \land \\
(\bigwedge_{o \in O} (P_m(PO))(o, L))\]

Figure 18. Example of a predicate for a class diagram.

**Definition 10 (Interpretation Mapping of Class Diagram)** An interpretation mapping \(I_M((C, A)) \mapsto P_D((C, A))\) of a class diagram \((C, A) \in ClassDiagram\) is a mapping to a predicate \((P_D((C, A)))((O, L))\) that a complex data \((O, L) \in ComplexData\) is an instance of a class diagram \((C, A)\), as every predicate in \(P_c[C]\) mapped from classes is satisfied, every predicate in \(P_a[A]\) mapped from associations is satisfied and all predicates \(P_m[A]\) mapped from multiplicities is satisfied.

\[(C, A) \in ClassDiagram, \\
(P_D((C, A)))((O, L)) := ((O, L) \in ComplexData) \land \\
(\bigwedge_{o \in O} \bigvee_{x \in P_c[C]} (x)(o)) \land (\bigwedge_{l \in L} \bigvee_{y \in P_a[A]} (y)(l)) \land (\bigwedge_{o \in O} \bigwedge_{z \in P_m[A]} (z)(o, L)) \quad (7)\]

\(P_D[ClassDiagram]\) denotes the set of all predicates mapped from class diagrams as \(\{P_D((C, A))\}|(C, A) \in ClassDiagram\). \(\rho_E\) denotes an operator \(\text{ClassDiagram} \rightarrow (2^{P_c[Class]} \times 2^{P_a[Association]} \times 2^{P_m[Association]})\), for elements of predicate is a mapping from a class diagram \((C, A)\) to a triple \((P_c[C], P_a[A], P_m[A])\) of predicate sets of class, association and multiplicity as \((P_c[C], P_a[A], P_m[A])\). \(\rho_A\) denotes an operator \(\text{ClassDiagram} \rightarrow \text{ClassDiagram}\), for assembling predicates is a mapping from a triple \((P_c[C], P_a[A], P_m[A])\) to a predicate \(P_D((C, A))\) mapped from a class diagram as \((\bigwedge_{o \in O} \bigvee_{x \in X} (x)(o))\) \land \((\bigwedge_{l \in L} \bigvee_{y \in Y} (y)(l))\) \land \((\bigwedge_{o \in O} \bigwedge_{z \in Z} (z)(o, L))\).

\[(C, A) \in ClassDiagram, \\
(\rho_E((C, A)) := (P_c[C], P_a[A], P_m[A]) \quad (8)\]

\(X \subseteq P_c[Class], Y \subseteq P_a[Association], Z \subseteq P_m[Association], \\
(\rho_A((X, Y, Z)))((O, L)) := ((O, L) \in ComplexData) \land \\
(\bigwedge_{o \in O} \bigvee_{x \in X} (x)(o)) \land (\bigwedge_{l \in L} \bigvee_{y \in Y} (y)(l)) \land (\bigwedge_{o \in O} \bigwedge_{z \in Z} (z)(o, L)) \quad (9)\]

Figure 18 is an example of a predicate for a class diagram.

### 6.3. Semantical Structure of Class Diagrams

A structure of a class diagram is a pair \((D_M, I_M)\), where \(D_M\) is a domain which all elements consist of complex data including objects, links and values, where \(I_M\) is an interpretation mapping for data types, classes, associations and class diagrams defined above.
On this structure τ, we can evaluate whether a complex data \((O, L) \in \text{ComplexData}\) is a instance of a class diagram through the complex data is assigned into the class diagram as \(P_D(\{(C, A)\})((O, L))\). Figure 19 is an example of an assignment to a class diagram.

7. Boolean Algebra for Class Diagrams

7.1. Well-formed Class Diagrams

We define syntactical and semantical class closure operators as useful for following definitions and proofs.

Definition 11 (Class Closure) Let \(A\) be a subset of associations Association. A syntactical class closure operator \(\gamma : 2^{\text{Association}} \rightarrow 2^{\text{Class}}\) is a mapping from a set of associations to a set of related classes as \(\{a_c \mid a_c \in A, \{r, m, ac\} \in \text{Assocs}(a)\}\). A semantical class closure operator \(\gamma : 2^{P_c[A]} \rightarrow 2^{P_c[\text{Class}]}\) is a mapping from a set of predicates \(P_c[A]\) mapped from associations \(A\) to a set of predicates \(P_c[\gamma(A)]\) mapped from classes \(\gamma(A)\).
\( \gamma(A) := \{ ac \in \text{Class} | \forall a \in A \subseteq \text{Association} \} \)

\( r \in \text{Name, } m \in \text{Multiplicity, } (r, m, ac) \in \text{ASSOC}(a) \} \)

\( ^\ast (P) := \{ P(c) \} \)

\[ \bigvee_{p \in P \subseteq P_a[\text{Association}]} (p)((y, (x_1, \ldots, x_n))) \land \bigvee_{1 \leq k \leq n} (P(c))(x_k) \]  

\( \text{Lemma 1 (Class Closure Preserving)} \quad \gamma(P_a[A]) = P_\ast[\gamma(A)] \)

For a domain of desirable algebraic structure, we define a family of well-formed class diagrams as following.

\( \text{Definition 12 (Well-formed Class Diagrams) Given a set } C_\ast \text{ be a subset of all classes Class and a set } A_\ast \text{ be a subset of all associations Association, a family of well-formed class diagrams } \text{WFCD} \text{ is a set of all pairs } (C \cup \gamma(A), A) \text{, where } C \subseteq C_\ast \gamma(A) \text{ and } A \subseteq A_\ast. \)

\( \text{WFCD} := \{(C \cup \gamma(A), A) | C \subseteq C_\ast \gamma(A), A \subseteq A_\ast \} \)

\( \text{WFCD}(C_\ast, A_\ast) \text{ denotes a family of well-formed class diagrams generated from given classes } C_\ast \text{ and given associations } A_\ast. \)

7.2. Merge Operation

A merge operation is defined in analogy with union of set because same classes are merged into one class and same associations are merged into one association. A merge operation means to generate a new blueprint to create merged instances made of two blueprints of class diagrams.

Following definitions of syntactical and semantical merge operations are same concept of previous studies\[2,3,4,12,14,15\] but simplified enough for following discussions.

\( \text{Definition 13 (Syntactical Merge Operation) Given two class diagrams } (C_X, A_X) \text{ and } (C_Y, A_Y), \text{ a syntactical merge operation } \triangledown : \text{ClassDiagram} \times \text{ClassDiagram} \rightarrow \text{ClassDiagram} \text{ is defined as } (C_X \cup C_Y, A_X \cup A_Y). \)

\( (C_X, A_X) \triangledown (C_Y, A_Y) := (C_X \cup C_Y, A_X \cup A_Y) \)

\( \text{Definition 14 (Semantical Merge Operation) Let two predicates } P_D((C_X, A_X)) \text{ and } P_D((C_Y, A_Y)) \text{ be mapped from class diagrams } (C_X, A_X) \text{ and } (C_Y, A_Y) \text{ respectively. Where } (O, L) \text{ is a member of ComplexData, a semantical merge operation } \triangledown : P_D[\text{ClassDiagram}] \times P_D[\text{ClassDiagram}] \rightarrow P_D[\text{ClassDiagram}], \text{ is defined as followed:} \)

\( (P_D((C_X, A_X)) \triangledown P_D((C_Y, A_Y)))(O, L) := (\rho_A(\rho_E((C_X, A_X)) \cup \rho_E((C_Y, A_Y)))(O, L)) \)

\[
P_D((C_X, A_X) \uplus (C_Y, A_Y)) = P_D((C_X, A_X)) \uplus P_D((C_Y, A_Y))
\] (15)

Result of an insert operation is included in \(P_D((C_X, A_X)) \uplus P_D((C_Y, A_Y))\), if an insert operation is defined as \((O_X \cup O_Y, L_X \cup L_Y)\) and common parts of \((O_X, L_X)\) and \((O_Y, L_Y)\) are same, where \((O_X, L_X)\) and \((O_Y, L_Y)\) satisfy predicates \(P_D((C_X, A_X))((O_X, L_X))\) and \(P_D((C_Y, A_Y))((O_Y, L_Y))\) respectively [7].

7.3. Difference Operation

Syntactical and semantical difference operations are defined in analogy with set difference but the results of difference operations should satisfy conditions of class diagram. Syntactical and semantical difference operations are defined as followed:

Definition 15 (Syntactical Difference Operation) Given two class diagrams \((C_X, A_X)\) and \((C_Y, A_Y)\), a syntactical difference operation \((-): ClassDiagram \times ClassDiagram \rightarrow ClassDiagram\), is defined as \(((C_X \setminus C_Y) \cup \gamma(A_X \setminus A_Y), A_X \setminus A_Y)\).

\[
(C_X, A_X) - (C_Y, A_Y) := ((C_X \setminus C_Y) \cup \gamma(A_X \setminus A_Y), A_X \setminus A_Y)
\] (16)

Definition 16 (Semantical Difference Operation) Let two predicates \(P_D((C_X, A_X))\) and \(P_D((C_Y, A_Y))\) be mapped from class diagrams \((C_X, A_X)\) and \((C_Y, A_Y)\). Where \((O, L)\) is a member of \(ComplexData\), a semantical difference operation \(-: P_D[ClassDiagram] \times P_D[ClassDiagram] \rightarrow P_D[ClassDiagram]\), is defined as followed:

\[
(P_D((C_X, A_X)) \uplus P_D((C_Y, A_Y)))((O, L)) := ((O, L) \in ComplexData) \land \\
(\rho_{\Lambda}((P_E((C_X, A_X))) \setminus \rho_E((C_Y, A_Y))) \cup (\gamma(P_a[A_X \setminus A_Y], \emptyset, \emptyset)))((O, L))
\] (17)

Lemma 3 (Difference Preserving) Let \((C_X, A_X)\) and \((C_Y, A_Y)\) be class diagrams. A predicate \(P_D((C, A))\) preserves difference operations.

\[
P_D((C_X, A_X) - (C_Y, A_Y)) = P_D((C_X, A_X)) \uplus P_D((C_Y, A_Y))
\] (18)

Result of a delete operation is included in \(P_D((C_X, A_X)) \uplus P_D((C_Y, A_Y))\), if a delete operation is defined as \((O_X \setminus O_Y, \mu(A_X \setminus A_Y, O_X), L_X \setminus L_Y)\), when common parts of \((O_X, L_X)\) and \((O_Y, L_Y)\) are same and \(\mu\) is an operator to recover objects included in remained classes as \(\{o \in O_X|\bigcup_{x \in \gamma(P_a[A_X \setminus A_Y])(x)}(o)\}\), where \((O_X, L_X)\) and \((O_Y, L_Y)\) satisfy predicates \(P_D((C_X, A_X))((O_X, L_X))\) and \(P_D((C_Y, A_Y))((O_Y, L_Y))\) respectively.
7.4. Complement Operation

Using merge and difference operations, syntactical and semantical complement operations on a family of well-formed class diagrams are defined as followed:

**Definition 17 (Syntactical Complement Operation)** Let a class diagram \((C, A) \in WFCD(C, A)\) be a member of a given family of well-formed class diagrams. A syntactical complement operation \(\mathbf{\bullet} : \text{ClassDiagram} \to \text{ClassDiagram}\), is defined as 
\[
(C, A) = (C \cup \gamma(A), A) - (C, A)
\]

\[
(C, A) := (C \cup \gamma(A), A) - (C, A)
\](19)

**Definition 18 (Semantical Complement Operation)** Let a predicate \(P_D((C, A))\) be mapped from a class diagram \((C, A)\). Where \((O, L)\) is a member of ComplexData, a semantical complement operation \((-) : P_D[\text{ClassDiagram}] \times P_D[\text{ClassDiagram}] \to P_D[\text{ClassDiagram}],\) is defined as 
\[
(P_D((C \cup \gamma(A), A)) - P_D((C, A)))(O, L).
\]

\[
(P_D((C \cup \gamma(A), A)) - P_D((C, A)))(O, L)\]
\]

\[
(\neg (P_D((C, A))) = (P_D((C, A)))(O, L)\)
\]

\[
(\neg ((P_D((C, A)))(O, L)))
\]

\[
(\neg ((P_D((C, A)))(O, L)))
\]

**Lemma 4 (Complement Preserving)** Let \((C, A)\) be a class diagram. A predicate \(P_D((C, A))\) preserves complement operations.

\[
(\neg (P_D((C, A))) = (P_D((C, A)))(O, L)
\]

Because a complement operation is a special case of a difference operation as difference of universe, a special case of a delete operation as deletion of an universal instance corresponds to a complement operation.

7.5. Intersection Operation

Syntactical and semantical intersection operations are defined with use of merge and complement operations as followed:

**Definition 19 (Syntactical Intersection Operation)** Given two class diagrams \((C_X, A_X)\) and \((C_Y, A_Y)\), a syntactical intersection operation \(\triangledown : \text{ClassDiagram} \times \text{ClassDiagram} \to \text{ClassDiagram}\), is defined as 
\[
(C_X, A_X) \triangledown (C_Y, A_Y) := (C_X \cup A_X) \triangledown (C_Y \cup A_Y)
\]

\[
(C_X, A_X) = (C_X \cup A_X) \triangledown (C_Y \cup A_Y)
\]

\[
(C_X, A_X) = (C_X \cup A_X) \triangledown (C_Y \cup A_Y)
\]

\[
(C_X, A_X) = (C_X \cup A_X) \triangledown (C_Y \cup A_Y)
\]

\[
(C_X, A_X) = (C_X \cup A_X) \triangledown (C_Y \cup A_Y)
\]
Definition 20 (Semantical Intersection Operation) Given two predicates $\mathcal{P}_D(\mathcal{C}_X;\mathcal{A}_X)$ and $\mathcal{P}_D(\mathcal{C}_Y;\mathcal{A}_Y)$ mapped from class diagrams $(\mathcal{C}_X;\mathcal{A}_X)$ and $(\mathcal{C}_Y;\mathcal{A}_Y)$, a semantical intersection operation $\triangleleft : \mathcal{P}_D[\text{ClassDiagram}] \times \mathcal{P}_D[\text{ClassDiagram}] \rightarrow \mathcal{P}_D[\text{ClassDiagram}]$, are defined as followed:

$$(\mathcal{P}_D(\mathcal{C}_X;\mathcal{A}_X) \triangleleft \mathcal{P}_D(\mathcal{C}_Y;\mathcal{A}_Y))(\mathcal{O};\mathcal{L}) := ((\mathcal{O},\mathcal{L}) \in \text{ComplexData}) \land \mathcal{P}_D((\mathcal{C}_X;\mathcal{A}_X)) \triangledown \mathcal{P}_D((\mathcal{C}_Y;\mathcal{A}_Y))$$ (23)

Lemma 5 (Intersection Preserving) Let $(\mathcal{C}_X;\mathcal{A}_X)$ and $(\mathcal{C}_Y;\mathcal{A}_Y)$ be class diagrams. A predicate $\mathcal{P}_D((\mathcal{C},\mathcal{A}))$ preserves intersection operations.

$$\mathcal{P}_D((\mathcal{C}_X;\mathcal{A}_X) \triangle (\mathcal{C}_Y;\mathcal{A}_Y)) = \mathcal{P}_D((\mathcal{C}_X;\mathcal{A}_X)) \triangleleft \mathcal{P}_D((\mathcal{C}_Y;\mathcal{A}_Y))$$ (24)

An intersection operation is a dual of a merge operation under a complement operation. We expect that a kind of a projection operation on instances corresponds to an intersection operation of data models. However, we have not cleared up what kind of operations corresponds to an intersection operation.

7.6. Boolean Algebra for Well-formed Class Diagrams

Since following theorem 1 and 2, structure of any family of well-formed class diagrams and model composition operations is Boolean algebra syntactically and semantically.

Theorem 1 (Syntactical Algebra) For any family of well-formed class diagram WFCD $(\mathcal{C}_*,\mathcal{A}_*)$, structure $(\text{WFCD}(\mathcal{C}_*,\mathcal{A}_*); \triangledown, \triangle, \bar{\circ}, (\emptyset,\emptyset), (\mathcal{C}_* \cup \gamma(\mathcal{A}_*),\mathcal{A}_*))$ is Boolean algebra.

Theorem 2 (Semantical Algebra) For any family of well-formed class diagram WFCD $(\mathcal{C}_*,\mathcal{A}_*)$, structure $(\{\mathcal{P}_D((\mathcal{C},\mathcal{A}))|\mathcal{C},\mathcal{A} \in \text{WFCD}(\mathcal{C}_*,\mathcal{A}_*)\}; \triangledown, \triangle, \bar{\circ}, \mathcal{P}_D((\emptyset,\emptyset)), \mathcal{P}_D((\mathcal{C}_* \cup \gamma(\mathcal{A}_*),\mathcal{A}_*)))$ is Boolean algebra.

Since any family of well-formed class diagrams with merge, difference, complement and intersection operations as class diagram algebra is Boolean syntactically and semantically, class diagram algebra has good properties such as idempotency, commutativity, associativity, involutivity, monotonicity and totality. Especially, merge and intersection operations are associative and commutative and the complement operation is involutive. Since operations of merge, intersection, difference and complement are very powerful such as logical connectives, a lot of various class diagrams are constructed with combinational uses of these operations. And it is clear to define model composition on data flow diagrams on class diagram algebra.
8. Conclusion

We present syntax and semantics of a class diagram describing a partial data model and introduce the foundation of class diagram algebra consist of merge, difference, complement and intersection operations on well-formed class diagrams. We prove that class diagram algebra is Boolean syntactically and semantically that has many good algebraic properties. It means that models described in class diagrams are able to be manipulated logically. Logical manipulation of class diagrams prevents mistakes of modelers that occur frequently in development projects of large information systems.

It is possible to implement composing operations of Boolean algebra on well-formed class diagrams into UML modeling tools. However, we omit to discuss some capabilities of UML – generalization and OCL, we will study the algebraic structure of generalization and OCL. And we should study operations for instances and clear up relationship between instance operations and model composition operations.

References

A. Appendix: Supplement of Definitions and Lemmas for Proving

Definition 21 (Atomic Well-formed Class Diagrams) Let a set $\text{WFCD}(C_*, A_*)$ be a family of well-formed class diagrams generated from a given set $C_*$ of classes and a given set $A_*$ of associations. A family of atomic well-formed class diagrams $\text{AWFCD}$ is $\{(c, \emptyset) | c \in C_* \setminus (\gamma(A_*)) \} \cup \{((\{a\}, \{a\}) | a \in A_* \}$.

\[
\text{AWFCD} := \{(c, \emptyset) | c \in C_* \setminus (\gamma(A_*)) \} \cup \{((\{a\}, \{a\}) | a \in A_* \} \tag{25}
\]

$\text{AWFCD}(C_*, A_*)$ denotes a family of atomic well-formed class diagrams generated from given classes $C_*$ and given associations $A_*$. For proving algebraic structure, we introduce an operator $\rho_U$ as follows:

Definition 22 (Union of $\text{AWFCD}$) Let an operator $\rho_U : 2^\text{AWFCD}(C_*, A_*) \rightarrow \text{WFCD}(C_*, A_*)$ be a mapping from a subset $X \subseteq \text{AWFCD}(C_*, A_*)$ of a family of atomic well-formed class diagrams to a well-formed class diagram in $\text{WFCD}(C_*, A_*)$ as $\bigcup_{(C, A) \in X} (C, A)$.

\[
\rho_U(X) := \bigcup_{(C, A) \in X} (C, A) \tag{26}
\]

Lemma 6 (Onto Union of $\text{AWFCD}$) An operator $\rho_U$ maps $2^\text{AWFCD}(C_*, A_*)$ onto $\text{WFCD}(C_*, A_*)$.

Lemma 7 (Preserving on Union of $\text{AWFCD}$) Let $X$ and $Y$ be subsets of $\text{AWFCD}(C_*, A_*)$. An operator $\rho_U(X)$ preserves merge operations.

\[
\rho_U(X \cup Y) = \rho_U(X) \cup \rho_U(Y) \tag{27}
\]
\[
\rho_U(X \setminus Y) = \rho_U(X) \setminus \rho_U(Y) \tag{28}
\]
\[
\rho_U((X \cup \gamma(A_*)) \setminus Y) = \rho_U(Y) \tag{29}
\]
\[
\rho_U(X \cap Y) = \rho_U(X) \cap \rho_U(Y) \tag{30}
\]

B. Appendix: Proofs of Theorems and Lemmas

Proof 1 (Class Closure Preserving)
\[
\gamma(P_a[A]) = \{ P_c(e) | \bigvee_{P \in P_a[A]}(p)((y, (x_1, ..., x_n))) \land \bigwedge_{1 \leq k \leq n}(P(c)(x_k)) = P_c[\{ c | \bigvee_{p \in P_a[A]}(p)((y, (x_1, ..., x_n))) \land \bigwedge_{1 \leq k \leq n}(P(c)(x_k)) \} = P_c[\{ c | \bigvee_{a \in A}(P_a(a))((y, (x_1, ..., x_n))) \land \bigwedge_{1 \leq k \leq n}(P(c)(x_k)) \} = P_c[\{ \gamma(A) | a \in A, \forall k, 1 \leq k \leq n \} = P_c[\{ ac | a \in A, r, m, ac \in \text{Assocs}(a) \} = P_c[\gamma(A)] \}
\]
Proof 2 (Onto Union of AWFCD)
∀(Cₓ ∪ γ(Aₓ), Aₓ) ∈ WFCD(Cₓ), (Cₓ ∪ γ(Aₓ), Aₓ) = (∪c∈Cₓ{c}∪∪a∈Aₓ(γ(a)), a) = ρᵥ({{c, ø} | c ∈ Cₓ} ∪ {γ(a)}, a) | a ∈ Aₓ), Then, {{c, ø} | c ∈ Cₓ} ∪ {{γ(a)}, a) | a ∈ Aₓ) ∈ AWFCD(Cₓ, Aₓ) □

Proof 3 (Merge Preserving)
Let (Cₓ, Aₓ) and (Cᵧ, Aᵧ) be class diagrams. Eq.(15) P_D((Cₓ, Aₓ) ∨ (Cᵧ, Aᵧ)) = ρ_A(ρ_E(Cₓ, Aₓ) ∨ (Cᵧ, Aᵧ)) = ρ_A(ρ_E(Cₓ ∪ Cᵧ, Aₓ ∪ Aᵧ)) = ρ_A(P_E[Cₓ] ∪ P_E[Cᵧ], P_m[Aₓ ∪ P_m[Aᵧ], P_m[Aₓ ∪ P_m[Aᵧ]]) = P_D((Cₓ, Aₓ) ∨ P_D((Cᵧ, Aᵧ)) □

Proof 4 (Difference Preserving)
Let (Cₓ, Aₓ) and (Cᵧ, Aᵧ) be class diagrams. Eq.(18) P_D((Cₓ, Aₓ) \ (Cᵧ, Aᵧ)) = ρ_A(ρ_E(((Cₓ \ Cᵧ) ∪ γ(Aₓ) \ Aᵧ), Aₓ \ Aᵧ)) = ρ_A((P_E[Cₓ] \ Cᵧ, γ(Aₓ) \ Aᵧ], P_m[Aₓ \ Aᵧ], P_m[Aₓ \ Aᵧ]) = ρ_A((P_E[Cₓ] \ Cᵧ, P_m[Aₓ \ Aᵧ], P_m[Aₓ \ Aᵧ]) = ρ_A(P_E[Cₓ] \ Cᵧ, P_m[Aₓ \ Aᵧ], P_m[Aₓ \ Aᵧ] ∪ γ(P_m[Aₓ \ Aᵧ], ø, ø)) = P_D((Cₓ, Aₓ)) □

Proof 5 (Complement Preserving)
Since syntactical and semantical operations of merge and difference are preserved by lemma 2 and 3, syntactical and semantical complement operations are preserved.

Proof 6 (Intersection Preserving)
Since syntactical and semantical operations of merge and difference are preserved by lemma 2 and 3, syntactical and semantical intersection operations are preserved.

Proof 7 (Preserving on Union of AWFCD)
Let X and Y be subsets of AWFCD(Cₓ, Aₓ). Eq.(27) ρᵥ(X) ∪ ρᵥ(Y) = ∪(C,A)∈X∪(C,A)∈Y(C,A) = ρᵥ(X ∪ Y). Eq.(28) ρᵥ(X ∪ Y) = ∪(C,A)∈X∪(C,A)∈Y(C,A) = ∪(C,A)∈X∪Y(C, ø) ∪ ∪(C,A)∈X∪(C,A)∈Y(γ(A), A) = ρᵥ(X) ∪ ρᵥ(Y). Eq.(29) Since syntactical and semantical operations of merge and difference are preserved on Union of AWFCD by Eq.(27) and Eq.(28) of lemma 7, syntactical and semantical complement and intersection operations are preserved.

Proof 8 (Syntactical Algebra)
Since 2_AWFCD(Cₓ, Aₓ) is a powerset, structure (2_AWFCD(Cₓ, Aₓ); ∪, ∩, c, ø, ø, Ø, AWFCD(Cₓ, Aₓ)) is a Boolean algebra[11]. Since an operator ρᵥ maps 2_AWFCD(Cₓ, Aₓ) onto WFCD(Cₓ, Aₓ) by lemma 6 and an operator ρᵥ is boolean homomorphism by lemma 7, structure (WFCD(Cₓ, Aₓ); ∨, ∩, c, ø, ø, (Cₓ ∪ γ(Aₓ), Aₓ)) is a Boolean algebra[11]. □

Proof 9 (Semantical Algebra)
Since structure (WFCD(Cₓ, Aₓ); ∨, ∩, c, ø, ø, (Cₓ ∪ γ(Aₓ), Aₓ)) is Boolean algebra by theorem 1, operators ρ_E and ρ_A are boolean homomorphism by lemma 2, 4
and 5 and \( \{ P_D((C, A)) | (C, A) \in WFCD(C_+, A_+) \} \) is image of the operator \( P_D \) of the entire domain \( WFCD(C_+, A_+) \). \( \{ P_D((C, A)) | (C, A) \in WFCD(C_+, A_+) \} \); \( \diamond \), \( \bigtriangleup, \bigtriangledown, P_D((\emptyset, \emptyset)), P_D((C_+ \cup \gamma(A_+), A_+)) \) is Boolean algebra[11]. \( \square \)