Fundamenta Informaticae 75 (2007) 295–314
IOS Press

Towards a Pragmatic Mereology

Ryszard Janicki[*] **Dai Tri Man Lê**[†]
Department of Computing and Software
McMaster University
Hamilton, Ontario, L8S 4K1 Canada
{ *janicki, ledt* }*@mcmaster.ca*

Abstract. A version of mereology (i.e. theory of *parts* and *fusions*) is presented. Some applications to model software structures are discussed.

Preface. In summer 1999, late Zdzisław Pawlak suggested to the first author that Leśniewski's ideas [21, 31] might help in solving the problem of formally defining the concept of "part-of" for Parnas' Tabular Expressions [9, 14]. This is how the research presented in this paper began.

1. Introduction

A correct construction of complex entities from the more primitive ones is one of the basic problems software engineering is facing at this moment. The need for precise rules for both composition and decomposition had been recognized more than thirty years ago (see [22]), but widely accepted formal techniques have not yet been found. *Object-Oriented Programming* [1] and in particular *Software Components* [30] are two popular paradigms that emphasise "building a whole from parts", and enjoy some degree of successful applications.

What computer science and/or software engineering need are simple but powerful formal calculi for manipulating parts. Lack of such caluculi seems to be a main reason why theoretical bases of Software Components and even Object-Oriented Programming are still underdeveloped [1, 30].

Such calculi are also needed since the *common sense notion* of parthood in software and programming is fuzzy and not common at all [3]. Even in types of engineering where common sense notions of

[*]Partially supported by NSERC of Canada.
[†]Partially supported by McMaster University Senate Scholarship and J.L.W. Gill Prize.

parthood are well established, it was noticed that the complexity of objects and dissimilarities of experience require a more formal approach [26]. Mereology, especially when enriched with some Category Theory [6] constructions is a possible means to provide a basis for such calculi.

Attempts to formalize the concept of "part of" go back to S. Leśniewski (1916-37, [21, 31]), who invented the name "mereology", i.e. theory of parts, and H. Leonard, N. Goodman (1940-50, [7, 19]). Leśniewski's systems were invented as an alternative to what is now called "standard set theory" (i.e. based on Zermelo-Fraenkel axioms) [28], and translation of his ideas into the language of standard set theory is not obvious and often problematic [27, 28, 31], so more practical applications are rare and difficult. Leonard and Goodman Calculus, formulated within standard set theory, was invented to provide a formal model for a universal[1] concept of parthood [4, 7, 28]. Both models have been substantially extended, however none of them has been substantially applied outside philosophy, cognitive science and pure logic [4, 28, 31]. Only in the last decade have there been serious attempts to apply mereological ideas to industrial engineering [26], knowledge engineering [3], approximate reasoning [24], software engineering [11, 17, 18], databases [3], and other areas [3, 26].

Eventhough the ideas of Leśniewski and Leonard-Goodman appear to be similar (see [27, 28]), mathematical results about this relationship are hard to find, and a major one [8] has not been widely accepted among Leśniewski's disciples[2] [31].

Apart from different mathematics, the major difference seems to be that in Leśniewski's model it is assumed that all topological peoperties like connectedness are implicit. In other words, when parts are composed according to their characteristics, they "naturally" connect with each other to produce a whole. The Leonard-Goodman model also allows such situations but does not enforce them [28]. Most of the applications, including this paper, uses notation and terminology from Leonard-Goodman model, eventhough their spirit might be more of Leśniewski. The only recent practical application of pure Leśniewski's ideas known to the authors is *Rough Mereology* ([24] and some following papers) in the framework of Zdzisław Pawlak's *Rough Sets* [23].

The initial motivation for this work was provided by an attempt to define a formal semantics for *Parnas tabular expressions* [13]. Tabular expressions [9, 14, 15] are *relational* means to represent the complex relations that are used to specify and document software systems. The technique is quite popular in software industry [9]. When software engineers discuss a specification using *tabular expressions*, statements like "this is *a part of* a bigger relation", "this relation is composed of the following *parts*", etc., can be heard very often, but what it means formally is not clear. It appears the standard algebra of relations lacks the formal concept of being a *part of* (see [10, 11, 12] for more details).

A concept of "part of" for relations was first proposed in [13] and its properties were first analysed in [10]. In [11] a solution to the problems caused by the assumed uniqueness of mereological sum was proposed, namely, an operational algebraic version of mereology was created. In [11], parts are composed and decomposed using mereological constructors and destructors instead of mereological sum and mereological product. In [12] the concept of equivalent parts was added to the formal model of mereology. Surprisingly, this concept was neglected before in the context of mereological theories [7, 28, 31]. The core of the model of [12] was recently extended by some categorical constructions and used to model the concept of parts in the Component-Based Software Development [18].

[1] Universality causes problems for applications, as in reality there are many structurally different kinds of part-whole relations, and without making this distiction clear, we might get unreasonable conclusions (see [3]).

[2] The paper [8] is right in the case of elementary mereology of Leśniewski, however Leśniewski's non-elementary mereology is a much richer theory. Therefore, the paper [8] is often considered heresy for some of Leśniewski's students [31].

In this paper we will present the revised and corrected results of [10, 11, 12, 18] in the following order: Basic Mereology (Chapter 2), a generalisation of mereological sum (Chapter 3), an Algebraic Mereology (Chapter 4), a mereology for direct products, i.e. Tabular Expressions (Chapter 5) and a mereology for Software Components (Chapter 6). Due to lack of space, the concept of equivalent parts will not be discussed.

2. Basic Mereology

Basic Mereology was proposed in [12] and we believe it is well suited to modelling elementary parthood in various parts of Software Engineering. To make the paper self-sufficient, we start with a survey on the theory of *partial orders* (see for example [25]).

Let X is a set. A relation $\preceq \subseteq X \times X$ is called a *partial order* iff it is reflexive $(x \preceq x)$, anti-symmetric $(x \preceq y \wedge y \preceq x \Rightarrow x = y)$, and transitive $(x \preceq y \wedge y \preceq z \Rightarrow x \preceq z)$. If $\preceq$ is a partial order then the pair $(X, \preceq)$ is called a *partially ordered set* or *poset*. A relation $\prec$ defined as $x \prec y \Longleftrightarrow x \preceq y \wedge x \neq y$ is called a *strict partial order*. The element $\bot \in X$ satisfying $\forall x \in X.\ \bot \preceq x$ is called the *bottom* of X. The element $\top \in X$ satisfying $\forall x \in X.\ x \preceq \top$ is called the *top* of X. An element $a \in A$ is a *minimal (maximal)* element of A iff $\forall x \in A. \neg(x \prec a)$ $(\forall x \in A. \neg(a \prec x))$. The set of all *minimal (maximal)* elements of A will be denoted by $min(A)$ $(max(A))$. The minimal elements of the set $X \setminus \{\bot\}$ are called *atoms* of the poset $(X, \preceq)$, and $Atoms$ denotes the set of all atoms of X. Let $A \subseteq X$. An element $a \in X$ is called an *upper bound*) (a *lower bound*) of A iff $\forall x \in A.\ x \preceq a$ $(\forall x \in A.\ a \preceq x)$. The sets of all *upper bounds* and *lower bounds* of A are denoted by $ub(A)$ and $lb(A)$ respectively. An element $a \in X$ is called the *least upper bound* (*supremum*) of A, denoted $sup(A)$, iff $a \in ub(A)$ and $\forall x \in ub(A).\ a \preceq x$, and it is called the *greatest lower bound* (*infimum*) of A, denoted $inf(A)$, iff $a \in lb(A)$ and $\forall x \in lb(A).\ x \preceq a$.

The relation $\widehat{\prec}$ defined as: $x \widehat{\prec} y \Longleftrightarrow x \prec y \wedge \neg(\exists z. x \prec z \prec y)$ is called the *cover* relation for $\preceq$.

Now we will begin with mereological axioms, but to do so we need some definitions. Let $(X, \preceq)$ be a poset (with or without $\bot$). The relation $\preceq$ is now interpreted as *"part of"*; a is a *part of* b iff $a \preceq b$, and a is a *proper part of* b iff $a \prec b$. Notice that "a is a *part of* b" is equivalent to saying that "b is a *whole of* a". The element $\bot$ is interpreted as an empty part. The relations $\circ$, $\dagger$ and $\diamond$ on $X \setminus \{\bot\}$ defined as

$$x \circ y \Longleftrightarrow \exists z \in X \setminus \{\bot\}.\ z \preceq x \wedge z \preceq y \qquad \text{(overlap)}$$

$$x \dagger y \Longleftrightarrow \neg(x \circ y) \qquad \text{(disjoint)}$$

$$x \diamond y \Longleftrightarrow \exists z \in X \setminus \{\bot\}.\ x \preceq z \wedge y \preceq z \qquad \text{(underlap)}$$

are called *overlapping*, *disjointness* and *underlapping* respectively. Two element x and y *overlap* iff they have a common non-empty part, they are disjoint iff they do not have a common non-empty part, and they underlap if they are both parts of another element (see [4, 28] for more properties).

We will now introduce the set of axioms which helps us to define Basic Mereology:

$$x \prec y \Rightarrow (\exists z \in X.\, z \prec y \wedge x \dagger z) \vee x = \bot \quad \text{(WSP)}$$
$$\forall x \in X \setminus \{\bot\}.\, \exists y \in Atoms.\, y \preceq x \quad \text{(ATM)}$$
$$\bot \in X \quad \text{(BOT)}$$
$$x \preceq y \Longleftrightarrow x(\widehat{\prec})^* y \quad \text{(CCL)}$$
$$\forall x \in X.\, \exists y \in max(X).\ x \preceq y \quad \text{(WUB)}$$

where $(\widehat{\prec})^*$ is the reflexive and transitive closure of $\widehat{\prec}$, i.e. $(\widehat{\prec})^* = \bigcup_{i=0}^{\infty}(\widehat{\prec})^i$.

The axiom WSP, called *Weak Supplementation Principle*, is a part of all known mereologies. Among others, it guarantees that if an element has a proper non-empty part, it has more than one. It is widely believed that any reasonable mereology must conform this axiom [28]. The Axiom ATM says that all objects (except the empty part) are built from *elementary elements* called *atoms*. The axiom BOT simply says that the empty part does exist, while CCL states that the part-of relation is the reflexive and transitive closure of the cover relation for $\preceq$. If X is finite then CCL is trivially satisfied. The final axiom WUB (*Weakly Upper Bounded*) in principle means that the set $max(X)$ is a roof that cover the whole set. This axiom is crucial when the concept of equivalent parts is introduced. If $\top \in X$ then WUB is clearly satisfied.

Definition 2.1. ([12])
A poset $(X, \preceq)$ will be called a **Basic Mereology** if it satisfies BOT, WSP, CCL, WUB and ATM.

Basic Mereology will be the mereology that we are going to use in the rest of this paper.

Example 1. For every set A, let $\widehat{A} = \{\{a\} \mid a \in A\}$ be the set of all singletons generated by A, i.e. if $A = \{a, b\}$, then $\widehat{A} = \{\{a\}, \{b\}\}$.

Let $D_1 = \{a, b\}$, $D_2 = \{1, 2\}$ be sets and let $X = 2^{D_1} \cup 2^{D_2} \cup 2^{D_1 \times D_2}$.

Define the relation $\preceq$ in $X \times X$ as follows:

$$A \preceq B \iff A \subseteq B \vee A \subseteq \pi_i(B),\ i = 1, 2$$

where $\pi_i(B)$ is the projection of B on i-th coordinate, i.e. $\pi_1(B) = \{x_1 \mid (x_1, x_2) \in B\}$, $\pi_2(B) = \{x_2 \mid (x_1, x_2) \in B\}$.

One can show by inspection that the pair $(X, \preceq)$ is a *Basic Mereology* with $Atoms = \widehat{D_1} \cup \widehat{D_2}$, $\bot = \emptyset$ and $\top = \{(a,1), (a,2), (b,2), (b,2)\}$. This example is a special case of a more general model that will be discussed in Section 5. A Hasse diagram of the relation $\preceq$ is presented in Figure 1. □

Notation. The relation from Figure 1 will often be used to illustrate various concepts, so to make the formulas shorter the following notation will be used. We will omit all braces, parenthesis and commas, and will write it as a subscript. For instance the element $\{(a,1), (a,2), (b,1), (b,2)\}$ will be denoted by $x_{a1a2b1b2}$, $\{(a,1), (b,1), (b,2)\}$ by x_{a1b1b2}, $\{(b,1), (b,2)\}$ by x_{b1b2}, etc. We will also use $\top$ to denote $x_{a1a2b1b2}$. □

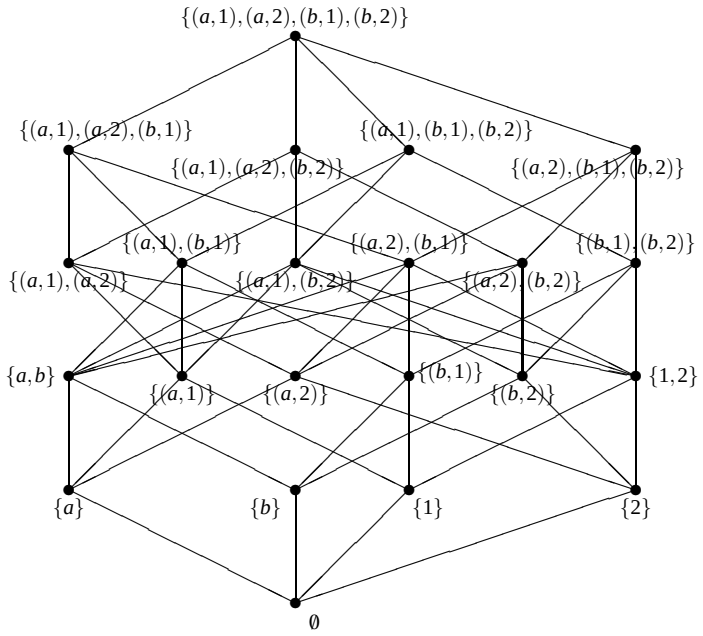


Figure 1. A Hasse diagram of the relation $\preceq$ from Example 1.

3. Mereological Sum

The operations $\oplus$ and $\odot$ defined by

$$z = x \oplus y \iff (\forall w \in X.\, w \circ z \Leftrightarrow w \circ x \vee w \circ y), \qquad \text{(sum)}$$

$$z = x \odot y \iff (\forall w \in X.\, w \preceq z \Leftrightarrow w \preceq x \wedge w \preceq y) \wedge z \neq \perp \qquad \text{(product)}$$

are called the *mereological sum* and *mereological product* respectively [4, 7, 28]. It is implicitly assumed for both definitions that in order to exist, z must be unique. Both concepts can easily be extended from two elements to any set in a standard way [4, 28]. The sum of elements of the set A (if exists) will be denoted by $\bigoplus A$, and the product of elements of the set A (if exists) will be denoted by $\bigodot A$. There is an obvious relationship between mereological sum and least upper bound and between mereological product and greatest lower bound, however those concepts are not identical. Let $X = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a,b,c\}\}$. The tuple $(X, \subseteq)$ is clearly a Basic Mereology, $sup(\{\{a\},\{b\}\}) = \{a,b,c\}$, $\{a\} \oplus \{b\}$ does not exist. On the other hand $sup(\{\{a\},\{b\},\{c\}\}) = \bigoplus\{\{a\},\{b\},\{c\}\} = \{a,b,c\}$. The idea is that if $\preceq$ represents "part-of" relation and the element $\{\{a\},\{b\},\{c\}\}$ is built from all three parts $\{a\}$, $\{b\}$ and $\{c\}$. Consider the relation $\preceq$ from Figure 1. The least upper bound $\sup(\{x_{b2}, x_{12}\})$ does not exist while $x_{b2} \oplus x_{12} = x_{b1b2}$ (see [12, 28] for more details).

Many mereologies assume that $x \odot y$ implies the existance of $x \oplus y$ [7, 28], which results in a very elegant model similar to semi-lattices or, when additional assumptions are made, to quasi boolean algebras [8, 28]. However, for our purposes such assumption is too strong, most of the models we are interested in do not have this property (including systems from [10, 11] and those discussed in Chapters 5 and 6). If different objects are allowed to have identical proper parts - and this is a usual case in Software Engineering applications, then the sum $x \oplus y$ often does not exist.

The mereological sum $\bigoplus A$, if it exists, can be interpreted as the most complex object built from the set of parts A only. What if $\bigoplus A$ does not exist? Intuitively, a most complex object (or a set of equivalent most complex objects) that can be built from the parts A might still exist. We will try to define such an object formally. Attempts to define such a construction, which could be seen as a generalisation of a mereological sum, were made in [10, 11, 12], however they are all, on one hand, too complex and on the other hand, too restrictive. The construction presented below is a refinement of the ideas presented in [12]. We intend to formally define an extension of mereological sum that will be called *mereological supremum* and denoted by $msup(A)$ for a given set A. If $\bigoplus A$ exists, we want $msup(A) = \bigoplus A$. To explain the other cases, let us consider some simple examples. Let $X = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a,b,c\}\}$. The tuple $(X, \subseteq)$ is clearly a Basic Mereology, $sup(\{\{a\},\{b\}\}) = \{a,b,c\}$, $\{a\} \oplus \{b\}$ does not exist, and we do not want $msup(\{\{a\},\{b\}\})$ to exist, as none of the elements of X can intuitively be seen as entirely built from $\{a\}$ and $\{b\}$.

Consider a Basic Mereology from Figure 1 and two elements x_{ab} and x_{12}. Neither $x_{ab} \oplus x_{12}$ nor $sup(\{x_{ab}, x_{12}\})$ exist but we want $msup(\{x_{ab}, x_{12}\})$ to exist and we want $msup(\{x_{ab}, x_{12}\}) = x_{a1a2b1b2}$, since $x_{a1a2b1b2}$ is intuitively the biggest element that can be built from x_{ab} and x_{12} (for example by decomposing x_{ab} and x_{12} into atoms x_a, x_b, x_1, x_2 and then composing $x_{a1a2b1b2}$ again). We will now give a formal definition of mereological supremum. Until the end of this chapter, we will assume that $(X, \preceq)$ is a basic mereology.

Let $\alpha : X \to 2^{Atoms}$ be a mapping defined as $\alpha(x) = \{a \mid a \in Atoms \wedge a \preceq x\}$.

The set $\alpha(x)$ is interpreted as the set of all atoms from which the element x is built. For each $A \subseteq X$, we define standardly $\alpha(A) = \bigcup_{x \in A} \alpha(x)$.

Definition 3.1. Let $A, C \subseteq X$. A set C is a **mereological cone** (or just **cone**) over A if and only if the following conditions are satisfied:

1. $A \subseteq C \subseteq ub(A)$,
2. $sup(C) \in C$,
3. $x \in ub(A) \wedge x \preceq sup(C) \;\Rightarrow\; x \in C$,
4. $\alpha(A) = \alpha(C)$. □

If $\bigoplus A$ exists, then $A \cup \{\bigoplus A\}$ is a mereological cone over A. Consider a Basic Mereology from Figure 1, and let $A_1 = \{x_b, x_1, x_2\}$, $A_2 = \{x_{b2}, x_{12}\}$, and $A_3 = \{x_a, x_b, x_1, x_2\}$. There is only one cone over A_1, namely $C_1 = A_1 \cup \{x_{b1b2}\}$ and only one cone over A_2, namely $C_2 = A_2 \cup \{x_{b1b2}\}$. However there are seven cones over A_3, $C_3^1 = A_3 \cup \{x_{a1b2}\}$, $C_3^2 = A_3 \cup \{x_{a2b1}\}$, $C_3^3 = C_3^1 \cup \{x_{a1a2b2}\}$, $C_3^4 = C_3^1 \cup \{x_{a1b1b2}\}$, $C_3^5 = C_3^2 \cup \{x_{a1a2b1}\}$, $C_3^6 = C_3^2 \cup \{x_{a2b1b2}\}$ and $C_3^7 = C_3^3 \cup C_3^4 \cup C_3^5 \cup C_3^6 \cup \{x_{a1a2b1b2}\} = ub(A_3)$. A mereological cone may not exist. Consider a Basic Mereology $(X, \subseteq)$, where $X = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a,b,c\}\}$. For the set $A = \{\{a\},\{b\}\}$ no mereological cone exists. The only set that satisfies (1), (2) and (3) of Definition 3.1 is $C = \{\{a\},\{b\},\{a,b,c\}\}$, but $\alpha(C) = \{\{a\},\{b\},\{c\}\}$ while $\alpha(A) = \{\{a\},\{b\}\}$, so the condition (4) of Definition 3.1 is not satisfied.

Let $MCones(A)$ denote the set of all merological cones over A. For instance for the sets A_1, A_2, A_3 discussed above, we have $MCones(A_1) = \{C_1\}$, $MCones(A_2) = \{C_2\}$ and $MCones(A_3) = \{C_3^1, ..., C_3^7\}$. Let $\mathbf{MC}_A$ be a partially ordered set defined as $\mathbf{MC}_A = (MCones(A), \subseteq)$. Each cone C can be interpreted as a partially ordered set $(C, \preceq_C)$, where $\preceq_C$ is the mereological order $\preceq$ restricted to the subset C of X.

Definition 3.2. An element $z \in X$ is called the **mereological supremum** of a set $A \subseteq X$, denoted $msup(A)$, if and only if

$$z = sup_{\preceq_C}(D_A),$$

where $D_A = sup_{\subseteq}(min_{\subseteq}(MCones(A)))$. □

If $MCones(A) = \emptyset$ then $msup(A)$ does not exist. For the sets A_1 and A_2 discussed previously, we have trivially $D_{A_1} = C_1$ and $D_{A_2} = C_2$, and consequently $msup(A_1) = msup(A_2) = x_{b1b2}$. The partially ordered set $\mathbf{MC}_{A_3}$ is fully described by the formulas: $C_3^1 \subseteq C_3^k \subseteq C_3^7$, $k = 3, 4$ and $C_3^2 \subseteq C_3^j \subseteq C_3^7$, $j = 5, 6$. Hence $min_{\subseteq}(MCones(A_3)) = \{C_3^1, C_3^2\}$ and $D_{A_3} = sup_{\subseteq}(\{C_3^1, C_3^2\}) = C_3^7 = ub(A_3)$. Therefore $msup(A_3) = sup_{\preceq_C}(D_{A_3}) = sup_{\preceq_C}(ub(A_3) = x_{a1a2b1b2} = \top$.

Proposition 3.1. (1) $\mathbf{MC}_A$ has a bottom $\bot_{\mathbf{MC}_A}$ if and only if $\bigoplus A$ exists.
(2) If $\bot_{\mathbf{MC}_A}$ exists then $msup(A) = sup_{\preceq}(A) = sup_{\preceq}(\bot_{\mathbf{MC}_A}) = \bigoplus A$. □

To illustrate the above Proposition, let us take again a Basic Mereology from Figure 1 and the set $A_4 = \{x_{b1}, x_{b2}, x_{12}\}$. There are four cones over A_4, $C_4^1 = A_4 \cup \{x_{b1b2}\}$, $C_4^2 = C_4^1 \cup \{x_{a2b1b2}\}$, $C_4^3 = C_4^1 \cup \{x_{a1b1b2}\}$, $C_4^4 = C_4^2 \cup C_4^3 \cup \{x_{a1a2b1b2}\}$. The partially ordered set $\mathbf{MC}_{A_4}$ is fully described by the formula: $C_4^1 \subseteq C_4^k \subseteq C_4^4$, $k = 2, 3$, i.e. $\bot_{\mathbf{MC}_{A_4}} = C_4^1$. In this case we also have: $msup(A_4) = sup_{\preceq}(A_4) = sup_{\preceq}(\bot_{\mathbf{MC}_{A_4}}) = \bigoplus A_4 = x_{b1b2}$.

Definition 3.3. A basic mereology $(X, \preceq)$ is **mereologically complete** (or just **complete**) if for each $A \in X$, $msup(A)$ does exist. □

If a mereology is (mereologically) complete, then for each set of parts A a single most complex object can be built from these parts. One can verify by inspection that the mereological system from Example 1 is complete. A Basic Mereology $(X, \subseteq)$, where $X = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b, c\}\}$ is not complete as for instance $msup(\{\{a\}, \{b\}\})$ does not exist.

4. Algebraic Mereological Systems

How can we build complex object from the more primitive ones? Usually we assume to have a set of *elementary* objects, or *atoms* and two sets of (partial) operations, *constructors*, that transform less complex objects into more complex, and *destructors*, that transform more complex object into less complex. There should be some relationship between the operators and both the *part of* and *equivalence* relations. Formally, we are looking for some sort of an abstract algebra. This kind of mereology was first proposed in [11], below we will present its recently revised and improved version. The following notation will be used in this and following sections.

Let I be a set of *indexes* and let $\{X_i \mid i \in I\}$ be a family of sets. By $\prod_{i \in I} X_i$ we denote a *Direct Product* over I, i.e. the set of all total functions $f : I \to \bigcup_{i \in I} X_i$ such that $\forall i \in I.\ f(i) \in X_i$. For finite

sets of indexes, Cartesian product notation is frequently used, and the functions belonging to $\prod_{i\in I} X_i$ are represented as vectors. For example if $I=\{2,5,9\}$, then $\prod_{i\in I} X_i = X_2 \times X_5 \times X_9$ and $f : \{2,5,9\} \to X_2 \cup X_5 \cup X_9$ with $f(2)=a_2$, $f(5)=a_5$, $f(9)=a_9$ is represented as a vector $(a_2,a_5,a_9) \in X_2 \times X_5 \times X_9$, i.e. $f=(a_2,a_5,a_9)$. We will adopt this convention for an infinite set of indexes. If for each $i \in I$, $f(i)=x_i$, we will write $f=\Pi_{i\in I}(x_i)$. We will use parenthesis "$(,)$", to make a distiction between $\Pi_{i\in I}(x_i)$ and $\prod_{i\in I} X_i$, as $\Pi_{i\in I}(x_i) \in \prod_{i\in I} X_i$. When $X_i = X$ for all $i \in I$ we will write X^k if I is finite and $|X|=k$, and X^I if I is not finite.

For every set of functions F, and every set A, let A^F denote the smallest set containing A and closed under F. Formally the set A^F can be defined as follows. For every $A \subseteq X$ and every $\phi : X^I \to X$, let

$$A^\phi = \{\phi(\Pi_{i\in I}(x_i)) \mid \forall i \in I . x_i \in A\}.$$

The set A^F is then the smallest set satisfying:

$$\forall \phi \in F \,.\, A^\phi \subseteq A^F,$$
$$\forall B \subseteq A^F \,.\, \forall \phi \in F \,.\, B^\phi \subseteq A^F.$$

We may now provide a formal definition of an algebraic mereological system.

Definition 4.1. By an **algebraic mereological system** we mean a tuple:

$$AMS = (X, Atoms, \bot, \Theta, \Delta, \alpha, \preceq)$$

where:

- X is a set of *elements*,
- $Atoms \subseteq X$ is a set of *elementary* elements (or *atoms*),
- $\bot \in X \setminus Atoms$ is an *empty* element (empty part),
- Θ is the set of *constructors*, each $\theta \in \Theta$ is a partial function $\theta : X^I \to X$, for some I,
- Δ is the set of *destructors*, each $\delta \in \Delta$ is a partial function $\delta : X \times X^k \to X$, for some $k \geq 1$, or $\delta : X \to X$,
- $\alpha : X \to 2^{Atoms}$ is a total function interpreted as the *elementary elements assignment* function, or as the *universal destructor* (it decomposes objects into elementary elements).
- $\preceq \subseteq X \times X$ is a "part of" relation,

that satisfies the following four properties AMS1, AMS2, AMS3 and AMS4. □

The first property we require is:

(AMS1) $B=(X,\preceq)$ is a *Basis Mereology*, $Atoms$ are atoms of B, $\bot$ is the bottom of B, and $\alpha(x) = \{a \mid a \in Atoms \wedge a \preceq x\}$

Before adding additional properties we need to discuss the role of constructors and destructors. We need constructors with an infinite number of parameters to build infinite sets from their elements; and set generalized union is an obvious example of such constructor, $\theta^\cup(\Pi_{i\in I}(A_i)) = \bigcup_{i\in I} A_i$. Now we can for example define $\{0,1,2,...\} = \bigcup_{i=0}^{\infty}\{i\} = \theta^\cup(0,1,2,...)$. We do not see any need for destructors with an infinite number of parameters, however the model can be easily extended to have them.

We will assume that each element is either elementary, or empty, or it can be constructed from the elementary elements by using the constructors and destructors. It can be formally expressed as:

(AMS2) $\quad Atoms^{\Theta\cup\Delta} = X$

and this is the second condition *AMS* must satisfy.

The relation $\preceq$ is interpreted as a "part of". What properties should it satisfy? Since operations from Θ are used to construct more complex elements, one property $\preceq$ should satisfy is:

$$\forall\theta\in\Theta.\ \theta(\Pi_{i\in I}(x_i)) = y \ \Rightarrow\ \forall i\in I.\ x_i \preceq y,$$
$$\text{or if } |I| = k,\ \forall\theta\in\Theta.\ \theta(x_1,\ldots,x_k) = y \ \Rightarrow\ (\forall i = 1,\ldots,k.\ x_i \preceq y).$$

Similar reasoning can be applied to "destructors", with one exception. If $\delta\in\Delta$ and the arity of δ is bigger than 1, i.e. $\delta : X^k \to X$, and $k > 1$, then the first argument of δ is treated differently then the remaining $k-1$. If $\delta(x,x_1\ldots,x_k) = y$, then we say that y was obtained from x with "help" of $x_1,\ldots,x_k$. In other words we assume that:

$$\forall\delta\in\Delta.\ \delta(x,x_1\ldots,x_k) = y \ \Rightarrow\ y\preceq x.$$

The definition of destructors requires some additional explanation, as many kinds of them are possible to define. Intuitively the destructors are inverses of constructors, so one may think they should be defined as partial functions $\delta : X \to X^k$ or $\delta : X \to 2^X$. This is indeed a valid approach and such destructors could be called "uncontrolled" destructors. As an example consider the following case. Let Y be a non-empty set, $X = 2^Y \cup 2^{Y\times Y}$, $\delta : X \to X^2$, $dom(\delta) = 2^{Y\times Y} \subseteq X$, and for each $A\subseteq Y\times Y$, $\delta(A) = (\pi_1(A),\pi_2(A))$, where π_i, $i = 1,2$, is a projection on the i-th coordinate. The function α is interpreted as a this kind of destructor. However uncontrolled destructors cannot model "controlled" destructions, like for instance set substruction operation. The destructors like $\delta(A,B) = A\setminus B$ are "controlled" destructors, A is the set that is being "destructed", and B is a "control" that governs this destruction. Another example of a controlled destructor is $\delta(A,B) = A\cap B$. In this case $\delta(A,B) = \delta(B,A)$ but it still fits in the definition. The control may be empty, and then we have a desctructor type $\delta : X \to X$. For $X = 2^Y \cup 2^{Y\times Y}$, projections π_1 and π_2 are examples of such destructors. Each uncontrolled destructor $\delta : X \to X^k$ can be simulated by k controlled destructors $\delta_i : X \to X$ and $\delta_i(x) = \pi_i(\delta(x))$, so the definition of *AMS* contains controlled destructors only.

We believe there is no need for destructors with infinite number of control arguments as their use is different than the use of constructors. For constructors, everything begins with just a set $Atoms$, so if X contains infinite elements we need constructors with infinite number of arguments to build them. For destructors, the starting point is the set $X\setminus Atoms$ (as for each $\delta\in\Delta$, $a\in Atoms$, either $\delta(a,C) = a$ or $\delta(a,C) = \perp$). Since there is not much restrictions on controllers, we believe there is always a way to define destructors in such a manner that the number of control arguments is finite, and for each element of X, its elementary components can be obtained by using destructors (as defined above) finite number of times. For example if $Atoms = \{\{i\} \mid i = 0,1,2,...\}$, then we need at least one constructor with infinite number of arguments to contract for instance $Y = \{0,1,2,...\}$. But if we have Y and $X = 2^Y$, we can retrieve $\{i\}\in Atoms$ from Y using a destructor $\delta(A,B) = A\setminus B$, by $\{i\} = \delta(Y,Z)$, where $Z = \{0,1,2,...,i-1,i+1,...\}\in X$.

We will now show how constructors and destructors relate to the relation "part of". Let $\dot{\preceq}\subseteq X\times X$ be the following relation:

$$x\dot{\preceq}y \iff (\exists\theta\in\Theta.\ \exists I.\ \exists\Pi_{i\in I}(x_i)\in X^I.\ \exists i\in I.\ x = x_i\wedge y = \theta(\Pi_{i\in I}(x_i)))$$
$$\vee\ (\exists\delta\in\Delta.\exists y_1,\ldots,y_k\in X.\ x = \delta(y,y_1,...,y_k))$$

If all constructors have finite number of arguments, the relation $\dot{\preceq}$ can be defined as follows:

$$x \dot{\preceq} y \iff (\exists \theta \in \Theta. \exists x_1,...,x_k \in X. \exists i \in \{1,...,k\}. x = x_i \wedge y = \theta(x_1,...,x_k)) \\ \vee (\exists \delta \in \Delta. \exists y_1,\ldots,y_k \in X.\ x = \delta(y,y_1,...,y_k)).$$

We can now formulate the third property *AMS* is required to satisfy, namely the relationship between $\dot{\preceq}$ and $\preceq$:

(AMS3) $\qquad \preceq = \dot{\preceq}^* = \bigcup_{i=0}^{\infty} \dot{\preceq}^i$

The fourth and the last property of *AMS* describes the relationship between constructors, destructors and atoms. In principle it says that no constructor can create nor destroy any atom, and no destructor can create an atom. Formally it can be formulated as follows

(AMS4) $\qquad \forall \theta \in \Theta.\ \alpha(\theta(\Pi_{i\in I}(x_i))) = \bigcup_{i\in I} \alpha(x_i)$, and
$\forall \delta \in \Delta.\ \alpha(\delta(x,x_1,\ldots,x_k)) \subseteq \alpha(x)$.

Definition 4.2. An algebraic mereological system $AMS = (X, Atoms, \bot, \Theta, \Delta, \alpha, \preceq)$ is **mereologically complete** if the basic mereology $(X, \preceq)$ is complete. □

Example 2. Let, as in Example 1, $D_1 = \{a,b\}$, $D_2 = \{1,2\}$, $X = 2^{D_1} \cup 2^{D_2} \cup 2^{D_1 \times D_2}$, $\bot = \emptyset$ and $Atoms = \widehat{D_1} \cup \widehat{D_2}$.

Define Θ as $\Theta = \{\dot\cup, \dot\times\}$, where $\dot\cup$ (a restricted union) and $\dot\times$ (a restricted Cartesian Product) are partial binary operations defined as follows

$$A \dot\cup B = \begin{cases} A \cup B & \text{if } A \cup B \in X \\ \text{undefined} & \text{otherwise} \end{cases} \qquad A \dot\times B = \begin{cases} A \times B & \text{if } A \times B \in X \\ \text{undefined} & \text{otherwise} \end{cases}$$

Let $\Delta = \{\pi_1, \pi_2\}$, where π_i, $i = 1,2$, is a projection of sets on the ith coordinate, formally defined as

$$\pi_1(A) = \begin{cases} A & \text{if } A \subseteq D_1 \\ \emptyset & \text{if } A \subseteq D_2 \\ \{x \mid (x,y) \in A\} & \text{if } A \subseteq D_1 \times D_2 \end{cases}$$

$$\pi_2(A) = \begin{cases} A & \text{if } A \subseteq D_2 \\ \emptyset & \text{if } A \subseteq D_1 \\ \{y \mid (x,y) \in A\} & \text{if } A \subseteq D_1 \times D_2 \end{cases}$$

and let $\alpha : X \to 2^{Atoms}$ be given by

$$\alpha(A) = \begin{cases} \hat{A} & \text{if } A \subseteq D_1 \cup D_2 \\ \{\{x\} \mid (x,y) \in A\} \cup \{\{y\} \mid (x,y) \in A\} & \text{if } A \subseteq D_1 \times D_2. \end{cases}$$

Note that in this case we have $X = Atoms^{\{\dot\cup, \dot\times\}}$. For instance: $\{(a,1),(b,2)\} = \{a\} \dot\times \{1\} \dot\cup \{b\} \dot\times \{2\}$.
Define the relation $\preceq$ in $X \times X$ in the same way as in Example 1, i.e.:

$$A \preceq B \iff A \subseteq B \vee A \subseteq \pi_i(B),\ i = 1,2.$$

One can show by inspection that the tuple $AMS = (X, Atoms, \bot, \Theta, \Delta, \alpha, \preceq)$ is a complete algebraic mereological system. □

	$x_2 \leq 0$	$x_2 > 0$
$y_1 =$	$x_1 + x_2$	$x_1 - x_2$
$y_2 \mid$	$y_2 x_1 - x_2 = y_2^2$	$x_1 + x_2 y_2 = \lvert y_2 \rvert$
$y_3 \mid$	$y_3 + x_1 x_2 = \lvert y_3 \rvert^3$	$y_3 = x_1$

Figure 2. The relation G defined by a *vector table*. The symbol "=" after y_1 indicates that the value of y_1 is a function of other variables, the symbol "|" after y_2 and y_3 indicates that the relationship between y_i, $i = 2,3$, is relational and not functional.

5. Mereology of Direct Products

As we mentioned before an initial motivation for this work was provided by an attempt to define formal semantics for *tabular expressions* [13, 15]. Tabular expressions (Parnas et al. [14, 9]) are means to represent the complex relations that are used to specify and document software systems. The technique is quite popular in the software industry (see [9]). To illustrate what *tabular expressions* are, we will analyze one *very* simple example.

Consider the following relation $G \subseteq IN \times OUT$, where $IN = Reals \times Reals$, $OUT = Reals \times Reals \times Reals$, x_1, x_2 are the variables over IN, y_1, y_2, y_3 are variables over OUT, and

$$(x_1,x_2)G(y_1,y_2,y_3) \iff \left\{ \begin{array}{l} y_1 = x_1 + x_2 \ \wedge\ y_2 x_1 - x_2 = y_2^2 \\ \wedge\, y_3 + x_1 x_2 = |y_3|^3 \end{array} \right\} \text{if } x_2 \leq 0$$
$$\left\{ \begin{array}{l} y_1 = x_1 - x_2 \ \wedge\ x_1 + x_2 + x_2 y_2 = |y_2| \\ \wedge\, y_3 = x_1 \end{array} \right\} \text{if } x_2 > 0$$

The relation G is more readable when defined by a *tabular expression* in Figure 2. This kind of tabular expression is called a *vector table* and its intuitive meaning is practically self-explanatory. It reads that if $x_2 \leq 0$ then $y_1 = x_1 + x_2$, y_2 must satisfy $y_2 x_1 - x_2 = y_2^2$, y_3 must satisfy $y_3 + x_1 x_2 = |y_3|^3$, and similarly for $x_2 > 0$.

The relation G is a composition of its "atomic" parts, i.e. $G = \bigcirc G_{i,j}$, where $G_{i,j}$, $i = 1,2$, $j = 1,2,3$, are relations defined by expressions in single cells. For instance $G_{1,3} \subseteq IN_{1,3} \times OUT_{1,3}$, where $IN_{1,3} = Reals \times (-\infty, 0\rangle$, $OUT_{1,3} = Reals$, and $(x_1,x_2)G_{1,3}y_3 \iff y_3 + x_1 x_2 = |y_3|^3$. The relation $G_{1,1}$ is a function $G_{1,1} : IN_{1,1} \to OUT_{1,1}$, with $IN_{1,1} = IN_{1,3}$ and $OUT_{1,1} = Reals$, and $(x_1,x_2)G_{1,1}y_1 \iff y_1 = G_{1,1}(x_1,x_2) = x_1 + x_2$. The relations $G_{i,1}$ are functions, which is indicated by the symbol "=" after variable y_1 in the left header. The symbol "|" after y_2 and y_3 indicates that $G_{i,2}$ and $G_{i,3}$ are relations with y_2 and y_3 as respective range variables.

Of course every $G_{i,j}$ is a part of G, every subset of G is a part of G, every relation defined by a tabular expression derived from the tabular expression that defines G by removing any number of rows and columns, is also a part of G. There are many types of tabular expressions, we gave a simple example of only one of them. For all tabular expressions, the global relation/function is defined as a composition of its parts, for each type of tabular expressions, an appropriate composition operation is different.

One of the biggest advantages of tabular expressions is the ability to define a relation R that describes the properties of the system specified, as an easy to understand composition[3] of the relations R_α, $\alpha \in I$, where R_α is a *part of* R.

The problem is that the standard algebra of relations lacks the formal concept of being a *part of*. The concept of subset is not enough, for instance if $A \subseteq B$ and $D = B \times C$, then A is not a subset of D, but according to standard intuition it is a *part of* D.

A semantics for Tabular Expressions was defined in terms of Heterogenous Relations [13, 15]. In this paper, for simplicity we will only discuss Direct Products, as Heterogenous Relations can be seen as a subclass of Direct Products.

Let T be a universal set of indexes and let $D = \{D_t \mid t \in T\}$ be an appropriate set of domains. We assume also that the set T is *finite*. From the viewpoint of applications in Software Engineering this is not a restriction at all ([13]).

For every $I \subseteq T$, let $D_I = \prod_{i \in I} D_i$. Even though each I in this section is finite, using the functional representation of elements of $\prod_{i \in I} D_i$ is more convenient for our reasoning.

For every function $f : X \to Y$, and every $Z \subseteq X$, the symbol $f|_Z$ will denote the restriction of f to Z, and for every function f, $dom(f)$ denotes the domain of f.

We will also assume that $D_i \cap D_j = \emptyset$ *if* $i \neq j$. This assumption allows us to identify every element of $a \in D_i$ with the function $f_a : \{i\} \to D_i$ where $f_a(i) = a$, which makes the notation more consistent and less ambiguous. This is a powerful assumption since most of the results of this section either false or undefined if $D_i \cap D_j \neq \emptyset$ for some $i \neq j$, which means we lose some theoretical generality.

We do not lose however much of practical generality here, for in practical applications each D_i has a different interpretation anyway (for instance: input current, output current; Amperes in both cases but different meaning).

For every D_I, $I \subseteq T$, let $X_I = \bigcup_{J \subseteq I} 2^{D_J}$. We shall write X instead of X_T (i.e. we put $X = X_T$). Also let $Atoms = \bigcup_{i \in T} \widehat{D}_i$. Clearly $Atoms \subseteq X$. Note also that for every $A \in X$ and for every $f, g \in A$ we have $dom(f) = dom(g)$.

For every $A \in X \setminus \{\emptyset\}$, let $\tau(A) \subseteq T$, the *index set* of A, be defined as follows: $I = \tau(A) \iff A \subseteq D_I$. In other words, $f \in A \Rightarrow dom(f) = \tau(A)$. We assume that $\tau(\emptyset) = \emptyset$. For instance if $A \subseteq D_2 \times D_5 \times D_7$ then $\tau(A) = \{2,5,7\}$. Note that $\tau(A)$ is only correctly defined if $D_i \cap D_j = \emptyset$ for $i \neq j$.

For every $A \in X$ and every $K \subseteq T$, let $A|_K = \{f|_{K \cap \tau(A)} \mid f \in A\}$ if $K \cap \tau(A) \neq \emptyset$, and $A|_K = \emptyset$ if $K \cap \tau(A) = \emptyset$. Clearly $A|_K \subseteq D_{K \cap \tau(A)}$. We will write $A|_i$ instead of $A|_{\{i\}}$ for all $i \in T$.

Define the relation $\preceq$ in $X \times X$ as follows:

$$\forall A, B \in X.\, A \preceq B \iff \tau(A) \subseteq \tau(B) \wedge A \subseteq B|_{\tau(A)}.$$

Theorem 5.1. The pair $(X, \preceq)$ is a complete basic mereology with $\top = \prod_{i \in T} D_i$. □

For $I = \{1,2\}$, $D_1 = \{a,b\}$, $D_2 = \{1,2\}$ we have a case analysed in Example 1.

We will now start to define an algebraic mereological system for direct products.

[3]The word "composition" here means "the act of putting together" (*Oxford English Dictionary*, 1990), not "the" mathematical composition of relations. In this sense "∪" is a composition.

For every $I \subseteq T$, let $\pi_I : X \to X_I$, be mapping defined by: $\pi_I(A) = A|_I$. Every such mapping is called a *projection*. We shall write $\pi_i(A)$ rather than $\pi_{\{i\}}(A)$ if $i \in T$. Since T is finite, every $I \subseteq T$ can be presented as $I = \{i_1, ..., i_k\}$, and we also have $\pi_I(A) = \pi_{i_1}(\pi_{i_2}(...(\pi_{i_k}(A))...))$.

Let $\dot{\bigcup}$ and $\dot{\times}$ be the following operations. For every family of sets $\{A_\alpha \mid \alpha \in I\}$, we define

$$\dot{\bigcup}_{\alpha \in I} A_\alpha = \begin{cases} \bigcup_{\alpha \in I} A_\alpha & \forall \alpha, \beta \in I \,.\, \tau(A_\alpha) = \tau(A_\beta) \\ \text{undefined} & \text{otherwise.} \end{cases}$$

For every $A, B \in X$, we define

$$A \,\dot{\times}\, B = \begin{cases} A \times B & \tau(A) \cap \tau(B) = \emptyset \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Define the set of constructors Θ as, $\Theta = \{\dot{\bigcup}, \dot{\times}\}$, and the set of destructors Δ as, $\Delta = \{\backslash\} \cup \{\pi_i \mid i \in T\}$. Also define $\alpha : X \to 2^{Atoms}$ as : for every $A \in X$ $\alpha(A) = \bigcup_{i \in \tau(A)} \widehat{A|_i}$. For example, $\alpha(\{(a,1),(b,2)\}) = \{\{a\},\{b\}\} \cup \{\{1\},\{2\}\} = \{\{a\},\{b\},\{1\},\{2\}\}$.

Theorem 5.2. The tuple $AMS = (X, Atoms, \bot, \Theta, \Delta, \alpha, \preceq)$, where X, $Atoms$, $\bot$, Θ, Δ, α and $\preceq$ are defined as above, is a complete algebraic mereological system. □

For $I = \{1,2\}$, $D_1 = \{a,b\}$, $D_2 = \{1,2\}$ we have a case analysed in Example 2.

The fundamental principle behind a success of a specification technique based on Tabular Expressions ([13, 14]) specification technique is that most of relations may be described as $R = \bigcirc_{i \in I} R_i$, where $\bigcirc$ is an operation, or composition of operations, each R_i is easy to specify. A variety of operations was introduced and discussed (see [13]). In this paper we will discuss only two operations denoted by $\uplus$ and $\otimes$. We shall show that the operator $\uplus$ corresponds to *mereological supremum* from Section 3, and that $\uplus$ could be defined in terms of more intuitive $\otimes$, where $\otimes$ corresponds to the well known join operator of Codd's relational data-base model [5].

Let $A, B \in X_T$ and let $K = \tau(A) \cup \tau(B)$, $J = \tau(A) \cap \tau(B)$. We define the operations "$\uplus$", and "$\otimes$" as follows.

$A \uplus B = \{ f \mid dom(f) = K \wedge ((f|_{\tau(A)} \in A \wedge f|_{\tau(B)\setminus\tau(A)} \in B|_{\tau(B)\setminus\tau(A)}) \vee$
$((f|_{\tau(B)} \in B \wedge f|_{\tau(A)\setminus\tau(B)} \in A|_{\tau(A)\setminus\tau(B)}))\}$,

$A \otimes B = \{ f \mid dom(f) = K \wedge (f|_{\tau(A)} \in A \wedge f|_{\tau(B)} \in B)\}$.

Let $A \subseteq D|_{\{1,3,5\}}$, $B \subseteq D|_{\{1,2,4\}}$. Then

$A \uplus B = \{(x_1,x_2,x_3,x_4,x_5) \mid ((x_1,x_3,x_5) \in A \wedge (x_2,x_4) \in B|_{\{2,4\}}) \vee ((x_1,x_2,x_4) \in B \wedge (x_3,x_5) \in A|_{\{3,5\}})\}$,

$A \otimes B = \{(x_1,x_2,x_3,x_4,x_5) \mid (x_1,x_3,x_5) \in A \wedge (x_1,x_2,x_4) \in B\}$.

Let I be some index set, and let $A = \{A_i \mid A_i \in X \wedge i \in I\}$,

$K = \bigcup_{i \in I} \tau(A_i)$, $J = \bigcap_{i \in I} \tau(A_i)$.

Let $\mathrm{Comp}_i A$ be the set of all the (set theory) *components*[4] of K that are NOT contained in $\tau(A_i)$.

For example if $I = \{1,2\}$, $\tau(A_1) \setminus \tau(A_2) \neq \emptyset$ and $\tau(A_2) \setminus \tau(A_1) \neq \emptyset$ then there are three components of K generated by $\tau(A_1)$ and $\tau(A_2)$, namely $\tau(A_1) \cap \tau(A_2)$, $\tau(A_1) \setminus \tau(A_2)$, and $\tau(A_2) \setminus \tau(A_1)$, so $\mathrm{Comp}_1\{A_1,A_2\} = \{\tau(A_2) \setminus \tau(A_1)\}$, and $\mathrm{Comp}_2\{A_1,A_2\} = \{\tau(A_1) \setminus \tau(A_2)\}$.

[4]Let X be a set, $X_i \subseteq X$ for all $i \in I$. Define $X_i^0 = X_i$ and $X_i^1 = X \setminus X_i$. A nonempty set $A = \bigcap_{i \in I} X_i^{k_i}$, where $k_i = 0, 1$, is called a (set theory) *component* of X generated by the sets X_i, $i \in I$. The components are disjoint and cover the entire set X (see [25]).

We define the operations "$\biguplus_{i\in I}$" as:

$$\biguplus_{i\in I} A_i = \{ f \mid dom(f) = K \wedge \exists i \in I.\ (f|_{\tau(A)} \in A \wedge \forall C \in \mathrm{Comp}_i A.\ f|_C \in \bigcup_{i\neq i} A_j|_C)\}.$$

It turns out the operation $\uplus$ can be defined in terms of the operation $\otimes$.

Lemma 5.1. ([10])
$A \uplus B = (A \otimes B|_{\tau(B)\setminus\tau(A)}) \cup (B \otimes A|_{\tau(A)\setminus\tau(B)})$.

It also turns out that $\uplus$ provides an operational definition of mereological supremum for the mereology of direct products.

Theorem 5.3. $\biguplus_{i\in I} A_i = msup(\{A_i \mid i \in I\})$ □

It may happen that $A \uplus (B \uplus C) \neq (A \uplus B) \uplus C$. Consider the mereological system from Examples 1 and 2, and $A = x_a$, $B = x_1$ and $C = x_{b2}$. We have $A \uplus (B \uplus C) = x_a \uplus (x_1 \uplus x_{b2}) = x_a \uplus x_{b1b2} = x_{a1a2b1b2}$, while $(A \uplus B) \uplus C = (x_a \uplus (x_1) \uplus x_{b2} = x_{a1} \uplus x_{b2} = x_{a1b2}$. This should not be surprising as we do not assume that the same parts always create the same wholes.

6. *MereoCat* and Software Components

This section provides a more refined version of Basic Mereology by using a combination of both Mereology and Category Theory to create a more expressive framework for component software. We assume that the reader is familiar with the basic concepts of elementary Category Theory [6]. Instead of axiomatizing the connectedness properties like in [4, 29], we use *morphisms* to describe the connection and *colimit* in Category Theory to describe the mereological construction and the part-of relations are built on top of these. The resulting framework is called ***MereoCat***, which stands for Mereo-Category.

6.1. Categorical Connector Framework and Architectural Views

We start with a description of categorical connector framework. To do this we adopt CommUnity, the architectural design framework invented by Fiadeiro et al. in [6, 20] because of its flexibility and generality, which does not restrict us to any specific architecture description language, its ability to model different aspects of parallel design, and especially its categorical power. To keep this paper as self-sufficient as possible we will give a brief overview of CommUnity's three architectural elements: components, configurations and connectors.

Components [30], are the model entities that perform computation and are able to synchronize with their environments and exchange information through channels. Hence, components are given in terms of their channels and actions in a form of "designs". For example, the component design ***print*** below consists of input channel *i*, output channel *po* and private channel *rd*. Actions of ***print*** are given in CommUnity as a special form of "guarded commands", except satisfying the guards only means the actions can be executed but does not force the action to be executed right away. In ***print***, if *rd=false* then action *print* is allowed to be executed and change *rd* to *true*. Action *prod* of ***print*** does the "opposite" of action *print* and also assign input *i* to output *po*. The ***convert*** component does the task of of a conversion module which converts an MSWord document to a PS document.

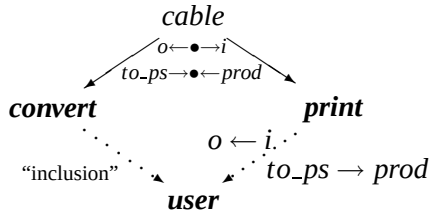
design	***print***
in	*i:ps*
out	*po:ps*
prv	*rd:bool*
do	*print[rd]:* $\neg rd \rightarrow po{:=}i \parallel rd{:=}\ true$
[]	*prod[rd]:* $rd \rightarrow rd{:=}\ false$

design	***convert***
out	*o:ps*
prv	*w:MSWord*
do	*to_ps[o]: true, false* $\rightarrow$ *o:=ps(w)*

Configurations are diagrams in a category of designs where objects are designs and morphisms are *superposition*, also called design morphisms. A design morphism $\sigma : P_1 \rightarrow P_2$ identifies that P_2 can be obtained from P_1 by "augmenting" additional behaviors to P_1 while still preserving properties of P_1 in P_2.

¿From a *meaningful* configuration (e.g. an output channel is not connected to other output channels) [20], a new design can be constructed using the colimit construction. For example, we want to build a new useful design from the previous designs ***print*** and ***convert***, using the configuration in the diagram below where *cable*, ***convert***, ***print*** are objects and each arrow represents a morphism between them.

The explicit names are not given to the action and channel of *cable* used for interconnection, but • symbols are used instead [6, 20]. The reason is that the interconnection does not reply to the global naming but precisely to associations (name binding). For example, we need to explicitly specify that *o*, *to_ps* are bound to *i* and *prod* respectively.



design	***user***
out	*o,po:ps*
prv	*rd:bool, w: MSWord*
do	*print[rd]:* $\neg rd \rightarrow po{:=}o \parallel rd{:=}\ true$
[]	*to_ps[o,rd]:* $rd \rightarrow o{:=}ps(w) \parallel rd{:=}\ false$

Using the colimit construction, the new resulting object ***user*** and two arrows from ***convert*** and ***print*** to ***user*** is introduced into the diagram. Here the colimit, as the *amalgamated sum* (*"module sum"*), will return the minimal single design representing the whole configuration. *The design objects and design morphisms constitute the category* ***C-DSGN***.

Connectors are model entities independent from components whose purpose is to coordinate interactions between components as in the spirit of [2]. Connectors are given in CommUnity in terms of a "glue" design and collection of "role" designs. Since the formal concept of connectors is quite lengthy, readers are referred to [20].

Now we will discuss architectural views. In the real world, Component Based Software Development for complex systems is more than just composing a system from pre-existing components together using connectors. When a system becomes larger and larger, it helps to understand the architectural structure of the system better by analyzing different architectural views, which are the different abstractions of the same software system. The first kind of view is by partitioning a system vertically into subsystems, which *aggregate* modules implementing related function functionalities. The second kind of view is by looking at the horizontal sections that may have different scope within the system. Layers may belongs to a single subsystem, a part of subsystem or across different subsystems [16].

The categorical framework discussed previously is designed to support composition of subsystems

from component designs. However, it is not quite obvious how the framework supports the layer and subsystem views of software architecture. Their approach mathematically treats all the designs as categorical objects and strongly emphasizes the properties preserved by morphisms, but also "flattens" down the whole architectural structure. Our goal is to complement the framework by bringing depth to the architectural structure using the part-of relation. We will start by constructing a suitable part-of relation for software components.

6.2. Construction of Part-of Relation and Naïve *MereoCat*

One of the controversies which is usually discussed within mereology is the transitivity of "part-of" relation. This can be expressed in terms of the software component concept as follows. Suppose a component x was used to build a subsystem y and y is again used to build a software system s. Is x is a part of s? Since by the encapsulation rule x is hidden from s by y. This confusion comes from the ambiguity in the meaning of "part", because natural language uses the same word "part" for different kinds of part/whole relationships. However, from a more abstract point of view, transitivity does hold [4]. If x is defective, s will not work anymore, since x does contribute the service to s *indirectly*.

Hence, due to encapsulation, there should exist (at least) two different kinds of parts. The first kind is when a whole can directly access the service provided by a part, and the second kind is when a part *indirectly* contributes the service to the whole by being hidden in another part. We will now characterize the *direct part-of* relation denoted by $\prec_d$, which describe the part-of relation between a whole and its direct parts, as follows:

Definition 6.1. Assume that a design S, which can be software system or subsystem, is constructed using colimit construction from a pair $(C, \textbf{C-DSGN})$ where: (1) C is a set of designs which includes the glue, component and subsystem designs; and (2) **C-DSGN** is the design category with respect to C. Then we define $\prec_d$ on C as:

$$\forall P \in C.\ P \prec_d S \qquad \square$$

Definition 6.1 describes a view of parts and wholes at a single *level* of composition where encapsulation is preserved in a sense such that the designs in C will appear to be *"atomic"* with respect to the system (or subsystem). In the previous example, we can now say ***convert*** $\prec_d$ ***user*** and ***print*** $\prec_d$ ***user***, but not *cable* $\prec_d$ ***user***, since *cable* is precisely used for "name-bindings" but is not really a part which constitutes subsystem ***user***.

To have a "multi-level" part-of relation we get the reflexive transitive closure of $\prec_d$:

Definition 6.2. $\preceq \stackrel{\text{df}}{=} (\prec_d)^* = \bigcup_{i=0}^{\infty} (\prec_d)^i \qquad \square$

Next, let us consider the poset $(X, \preceq)$, where X is the component domain which contains all the designs (components, subsystems, glue and software systems) and the *empty design* $\perp$. Let $Atoms$ be the set of all component designs. We have to show that $(X, \preceq)$ is actually a *Basic Mereology*. Hence, $\preceq$ is a correct "part of" relation for CBS.

Proposition 6.1. If $(X, \preceq)$ satisfies WUB then $(X, \preceq)$ is a *Basic Mereology*. $\square$

Definition 6.2 presents the proper definition of the parthood relation in Component-based Software. Besides, the preceding part-of relation construction also shows how closely part-whole relations and encapsulation concept are related. To say some part x is encapsulated, we need the information of what whole hides it, which is equivalent to say what whole x is a part of. Therefore, without being able to formalize the part-of relations, it is difficult to formalize the visibility in Component-based Software.

We are ready to give a formal definition the ***MereoCat*** System

Definition 6.3. A ***MereoCat*** System is a tuple $MC = (X, Atoms, \perp, \Theta, \preceq, \textbf{C-DSGN})$ where X, $Atoms$, $\perp$, $\preceq$, **C-DSGN** as defined previously and

- $(X, \preceq)$ is a Basic Mereology,
- Θ is the set of constructors, each $\theta \in \Theta$ is a partial function $\theta : X^I \to X$, for some I,
- The semantic of each θ is the colimit construction which constitutes designs from meaningful categorical diagrams of elements in X^I with respect to **C-DSGN**. □

The ***MereoCat*** System is just the Basic Mereology $(X, \preceq)$ conjuncted with the design morphisms in **C-DSGN**. Connectedness properties are the results of the graph-based semantics of Category Theory, but more than that the connections here embed the abstraction of complex interaction or communication protocols between designs. It can be proven that the ***MereoCat*** system is an Algebraic Mereological System with the empty set of destructors and with the set of constructors defined in categorical style. It can also be proven that ***MereoCat*** has all the expressive power of Ground Mereotopology [4, 29].

We took advantage of the semantic of design morphisms to describe the part-of relationship in Definitions 6.1, 6.2 and 6.3, since according to [6, 20], a design morphism $\sigma : P_1 \to P_2$ identifies a way in which P_1 is "augmented" to become P_2 through the interconnection of *one or more* components (the superposition of additional behavior). Hence, the design of the categorical approach implicitly assumes some of the part-whole relationship in mind, except they have not made it as formal and clear as we do in this paper. However, we still call this ***MereoCat*** System "Naïve ***MereoCat***" since it still requires axioms for connectedness and some formal concept of consistency.

6.3. An Example

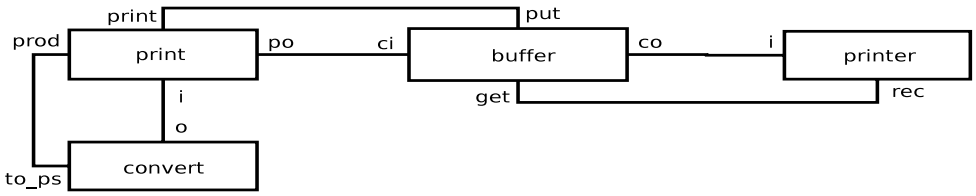
Using the previous components ***convert*** and ***print***, we can design a ***User-Printer*** application where, a user application send a PS document to a "printing server" component ***printer*** to print the document. All the communication is done through a bounded buffer ***buffer***, which prevents ***user*** from sending a new document when there is no space and prevents ***printer*** from reading a new message when no new message has been sent. The designs of ***buffer*** and ***printer*** are given as follows:

design	***buffer***
in	*ci:ps*
out	*co:ps*
prv	*rd: bool; q:queue*
do	*put:* $\neg$*full(q)* $\to$ *q:= enqueue (i,q)*
[]*prv*	*next:* $\neg$*empty(q)* $\wedge \neg$ *rd* $\to$ *o:=head(q)* $\|\|$ *q:= tail(q)* $\|\|$ *rd:=true*
[]	*get: rd* $\to$ *rd:= false*

design	***printer***
in	*i:ps*
prv	*busy:bool*
do	*rec:* $\neg$*busy* $\to$ *busy:=true*
[]	*end_print:busy* $\to$ *busy:=false*

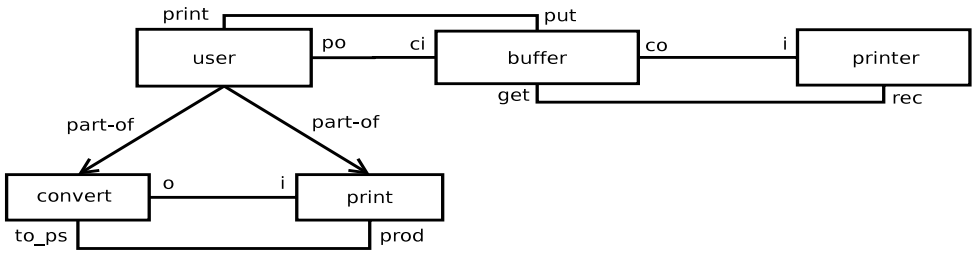
We can specify the configuration of the situation in two different ways.

The first method is to specify the architectural configuration as in the diagram in the following figure.



Notice that instead of drawing the categorical diagram as in previous section, we use the "syntactic sugar" of name-binding to associate the correspondent methods and channels of designs. According to the name-binding method of [20], we bind ***print*** component with the input of ***buffer***.

The second method, using ***MereoCat***, is to specify the part-of relation as a way to "modularize" the configuration as in the following figure.



We connect the resulting *subsystem* ***user*** $= \theta($***convert***$,$***print***$)$ to ***buffer*** where θ denotes the constructor that constructs ***user*** from two components ***convert*** and***print***. The semantics of θ is the colimit construction (amalgamated sum) as previously discussed in Section 6.1. As a result, we have a more hierarchical view of the whole ***User-Printer*** system.

Obviously, we can recursively apply this method to different parts of the system when the system grows larger and larger. Notice that ***buffer*** (the "glue" design) and its associations (name-bindings) constitute a *connector* according to the connector definition in [20].

For other examples of ***MereoCat*** applications, readers are referred to [18].

7. Final Comments

A version of mereology tuned towards applications to Software Engineering has been presented. The concept of mereological sum has been generalised, and an algebraic operational mereological system has also been analysed. Two applications have been discussed. The first one is to analyse parthood for direct products, that can be used to model parthood for Tabular Expressions. The second one involves elements of category theory and shows how parthood can be modelled for Software Components. There are many problems that we have not yet dealt with. How can equivalent parts of [12] be incorporated into the algebraic model presented in Chapter 4? How to model abstraction/refinement relationship between various mereological models? It appears that adding some categorical constructions to our mereology makes it more useful, but the research in this direction has just begun. The task of specifying part-whole

relation is more natural than one might think. According to [32], 58% of artificial objects and 42.7% of biological objects are parts. This shows how important the part-whole relation is as an abstraction underlying the organisation of human knowledge.

References

[1] A. Abadi, L. Cardelli, *Theory of Objects*, Springer 1996.

[2] R. Allen, D. Garlan. A Formal Basis for Architectural Connection, *ACM Transactions on Software Engineering and Methodology*, 6,3 (1997) 213-249.

[3] A. Artale, G. Franconi, E. Guarina, L. Pazzi, Part-whole relations in object-centered systems, *Data and Knowledge Engineering*, 20 (1996), 347-383.

[4] R. Casati, A. C. Varzi, *Parts and Places*, MIT Press, 1999.

[5] E. F. Codd, A relational model of data for large shared data banks, *Comm. of the ACM*, 13 (1970) 377-388.

[6] J. L. Fiadeiro, *Categories for Software Engineering*, Springer 2004

[7] N. Goodman, *The Structure of Appearance*, 3rd edition, D. Reidel, 1977.

[8] A. Grzegorczyk, The System of Leśniewski in Relation to Contemporary Logical Research, *Studia Logica* 3 (1955), 77-95.

[9] D. M. Hoffman, D. M. Weiss (eds.), *Collected Papers by David L. Parnas*, Addison-Wesley, 2001.

[10] R. Janicki, Remarks on Mereology of Direct Products and Relations, in J. Desharnais, M. Frappier, W. MacCaull (eds.), *Relational Methods in Computer Science*, Methodos Publ. 2002, pp. 65-84.

[11] R. Janicki, On a Mereological System for Relational Software Specifications, Proc. of MFCS'2002, *Lecture Notes in Comp. Science* 2420, Springer 2002, pp. 375-386.

[12] R. Janicki, Basic Mereology with Equivalence Relations, Proc. of MFCS'2005, *Lecture Notes in Comp. Science* 3618, Springer 2005, pp. 507-519.

[13] R. Janicki, R. Khédri, On a Formal Semantics of Tabular Expressions, *Science of Computer Programming*, 39 (2001) 189-214.

[14] R. Janicki, D. L. Parnas, J. Zucker, Tabular Representations in Relational Documents, in C. Brink, W. Kahl, G. Schmidt (eds.), *Relational Methods in Computer Science*, Springer 1997, pp. 184-196.

[15] R. Janicki, A. Wassyng, Tabular Expressions and Their Relational Semantics, *Fundamenta Informaticae* 67,4 (2005), 343-370

[16] M. Jazayeri, A. Ran, F. van der Linden, *Software architecture for product families: principles and practice*. Boston, MA, USA, 2000.

[17] R. Khédri, L. Wang, L. Situ, Requirements Specification Decomposition: A System Testing Driven Approach, *Proc. 7th Int'l Seminar On Relational Methods in Computer Science*, Bad Melente, Germany, 2003,pp. 97-104.

[18] D. T. M. Le, R. Janicki, On a Parthood Specification for Component Software, Proc. of RSCTC'06, *Lecture Notes in Artificial Intelligence* 4259, Springer 2006, pp. 537-546.

[19] H. Leonard, N. Goodman, The calculus of individuals and its uses, *Journal of Symbolic Logic*, 5 (1940), 45-55.

[20] A. Lopes, M. Wermelinger, J. L. Fiadeiro, High-order architectural connectors. *ACM Transactions on Software Engineering and Methodology*, 12,1 (2003) 64-104.

[21] S. Leśniewski, Grundzüge eines neuen Systems der Grundlagen der Mathematik, *Fundamenta Mathematicae* 24 (1929), 1-81.

[22] D. L. Parnas, On the criteria to be used in decomposing systems into modules, *Comm. of ACM*, 15, 12 (1972), 1053-1058.

[23] Z. Pawlak, Rough Sets, Kluwer, 1991.

[24] L. Polkowski, A. Skowron, Rough Mereology, A New Paradigm for Approximate Reasoning, *Journal of Approximate Reasoning*, 15, 4 (1997), 316-333.

[25] K. H. Rosen, *Discrete Mathematics and its Applications*, McGraw-Hill 1991.

[26] F. A. Salustri, J. C. Lockledge, Towards a formal theory of products including mereology, *Proc. 12th Int'l Conf. on Engn. Design*, Munich, 1999, pp. 1125-30.

[27] P. Simons, On Understanding Leśniewski, *Hist. and Phil. of Logic* 3 (1982),165-191.

[28] P. Simons, *Parts. A Study in Ontology*, Claredon Press, 1987.

[29] B. Smith, Mereotopology: a Theory of Parts and Boundaries. *Data and Knowledge Engineering*, 20,3 (1996) 287-303.

[30] C. Szyperski, *Component Software*, Addison-Wesley, 1997.

[31] J. T. J. Srzednicki, V. F. Rickey (eds.), *Leśniewski's Systems*, Kluwer, 1984.

[32] B. Tversky, K. Hemenway, Objects, Parts and Categories, *Journal of Experimental Psychology*, 113, 2 (1984) 169-193.