ABSTRACT
In this paper a proof is presented that shows that the relation between technical functions and their subfunctions in functional descriptions of products can formally not be taken as a relation of parthood. Technical functions of two specific classes are modelled as well as their composition. In this modelling functions are taken as transformations of tokens of flows of energy, material and signals, which makes them proper instances of functions on many engineering accounts of functions. Then it is proved that the relations between the considered functions and their subfunctions do not in general meet the basic postulates of mereology, the theory of parthood relations. The ramification of this proof is that in engineering ontologies the relation between subfunctions and functions should not be described as a formal parthood relation.

INTRODUCTION
In functional descriptions of technical products, technical functions and their subfunctions are related to one another via functional compositions and decompositions. By this relation it is natural to take subfunctions as components of the functions they compose, as is typically done in the literature. Yet, subfunctions are now and then also presented as parts of these functions [1,2,3,4,5,6], and this additional characterisation is the topic of this paper. It will be shown that under a minimal formal interpretation of the relation of parthood as given by mereology [7,8,9], it is in general impossible to take subfunctions as parts of the functions they compose. That is, in the case of single functional compositions or decompositions, it is possible to take subfunctions as parts of functions. But in more complex cases, when compositions or decompositions are combined (e.g., [10], figure 2.3), the relation between subfunctions and functions cannot formally be taken as one of parthood.

The relevance of this result to mainstream engineering may be limited. Although there are ready arguments against taking subfunctions as parts of functions, it seems rather harmless to nevertheless do so. One of those ready arguments consists of pointing out that the concept of component is different to that of part, which implies that taking subfunctions as components of functions is something different from taking subfunctions as their parts. A vector (1,1,0), for instance, can have another vector (0,½√2,-½√2) as a component, say, when (1,1,0) is decomposed relative to a basis given by (-1,0,0), (0,½√2,½√2) and (0,½√2,-½√2). But (1,1,0) does not have (0,½√2,-½√2) as its part. Yet, although true, it seems harmless to occasionally take a component as a part, say the technical function of a train’s engine as a part of the technical function of the train. Sometimes components are even parts, as structural components of technical products are, like the train’s engine. It may even be helpful in designing to see subfunctions as parts of functions, say because it suggests considering the design solutions for subfunctions as parts of the solutions of the overall function. Moreover, even though mereology advances a formal relation of parthood meeting specific postulates, it is acknowledged by mereologists that the parthood relations that are used in natural language need not unconditionally comply with these postulates (e.g., [11]).

The result of this paper is instead relevant to research on engineering ontologies. Ontologies are aimed at giving the means to describe technical systems in general and unambiguous ways, and are increasingly accommodating functional descriptions (e.g., [3,6,12,13,14,15,16,17]). In engineering ontologies the relation of parthood now has a fundamental, precise and formal meaning as laid down by the postulates of mereology. And in engineering ontologies
subfunctions are also every now and then assumed to be parts of the functions they compose \[3,4,5,6\]. The result of this paper disproves this assumption, showing that in engineering ontologies the relation between subfunctions and functions should be modelled differently than as parthood.

Addressing an issue within engineering ontologies rather than within mainstream engineering, provides the possibility to derive the mentioned result with a modelling of technical functions as formal entities that are formal and somewhat detached from how engineers understand functions. A reason to opt for a formal modelling is to safeguard the generality of the result. In engineering there exist numerous accounts of technical functions \[18,19,20,21,22\], which implies that a general result about these functions should either be argued for by considering each account separately, or by considering specific cases of technical functions that reasonably should be accepted as proper instances of functions by all or most accounts. In this paper the second route is taken. The technical functions considered correspond to transformations of flows of energy, material and signals, yet the cases considered are rather specific and by far not representing all possible functions in engineering.

The paper starts by a brief discussion of mereology introducing the basic postulates of formal parthood relations. Then the considered classes of technical functions are introduced, with their modelling and the modelling of functional composition. In the second part of the paper a number of regular cases of functional composition are discussed in which the relation between subfunctions and functions meets the basic postulates of mereology. Yet cases beyond the regular ones lead to violations of these basic postulates.

**NOMENCLATURE**

- \(a, b, c, \ldots\) tokens of flows of energy, material or signals
- \(\phi\) a token function that transforms the token flows in a set \(I\) to the token flows in a set \(O\); represented by \(\langle I,O\rangle\)
- \(\Phi\) a type function that has as instances the token functions \(\phi = \langle I,O\rangle\), \(\phi' = \langle I',O'\rangle\), and so on; represented by \(\{\langle I,O\rangle, \langle I',O'\rangle, \ldots\}\)
- \(P(\phi,\phi')\) \(\phi\) is a part of \(\phi'\)
- \(PP(\phi,\phi')\) \(\phi\) is a proper part of \(\phi'\)

**MERELOGY**

Mereology \[7,8,9\], the theory of parthood, does not advance one relation of parthood. It rather defines and analyses different formalised parthood relations, which are compared with parthood relations as used in natural and specialised languages. Mereology thus provides a spectrum of possible parthood relations \[11\], for physical objects, for temporal periods, for social entities, and so on, similar to how logic provides a spectrum of possible logics for different cases. Yet, in mereology it is assumed that there is a common core to all parthood relations. This common core is called ground mereology, and captured by simple postulates as given below. This assumption is normative: well-defined formalised parthood relations, as used in ontologies, are by construction meeting the postulates, and when parthood relations as advanced in natural language are not meeting these postulates, this is considered to be problematic and in need of explanation and mending (as in, e.g., \[23,24,25\]).

There are in mereology two distinct concepts of part, and the postulates of ground mereology depend on what concept one adopts. The first is that of proper part by which an entity is by definition not a (proper) part of itself. The second concept is just called part and with this concept an entity is by definition always a part of itself. One can adopt either the concept of proper part or of part as primitive, and then define the other by means of the primitive one. Yet, the second concept of part is assumed to be more general, and typically taken as primitive.

Let \(P(\phi,\phi')\) represent the part-of relation, which is to be read as ‘\(\phi\) is a part of \(\phi'\)’, and which thus mean that \(\phi\) is identical to \(\phi'\) or a proper part of \(\phi'\). This relation counts as a ground mereology if it meets three postulates, called reflexivity, Eqn. (1), antisymmetry, Eqn. (2), and transitivity, Eqn. (3):

\[
P(\phi,\phi) \tag{1}
\]
\[
(P(\phi,\phi') \land P(\phi',\phi)) \Rightarrow \phi = \phi' \tag{2}
\]
\[
(P(\phi,\phi') \land P(\phi',\phi'')) \Rightarrow P(\phi,\phi'') \tag{3}
\]

These postulates are supposed to be met by any formal parthood relation. Mereology has more postulates, and parthood relations are categorised by the additional postulates they meet. Here I focus on the above postulates of ground mereology, but briefly mention the more special extensional parthood relations. These relations satisfy a fourth postulate called strong supplementation \[7,8,9\] and meet the extensionality condition:

\[
(\exists \psi(PP(\psi,\phi)) \lor \exists \psi(PP(\psi,\phi'))) \Rightarrow \\
(\forall \psi(PP(\psi,\phi) \iff PP(\psi,\phi'))) \Rightarrow \phi = \phi' \tag{4}
\]

where \(PP(\phi,\phi')\) is the proper part-of relation defined as:

\[
PP(\phi,\phi') = (P(\phi,\phi') \land \neg P(\phi',\phi)) \tag{5}
\]

Condition Eqn. (4) expresses that two entities \(\phi\) and \(\phi'\) are the same when they have exactly the same set of proper parts.

**FUNCTIONS AND MEREOLOGY**

Existing work in engineering ontologies on parthood in relation to technical functions has been dominated by analyses of how functional descriptions of products define parts of those products \[23,24,25,26,27,28,29,30\]. These parts (and the wholes they are part of) are then themselves not functions but structural parts of products, or temporal parts of processes associated with the products. For instance, a functional
description of a house may single out a door as a functionally defined structural part of the house, and a functional description of a door may single out the door’s handle as a structural part of the door (e.g., [23]). Central questions in this work are whether these functionally defined structural or temporal parthood relations meet the postulates of mereology, and how to explain possible violations. That such violations occur is generally accepted in mereology. It is, for instance, assumed that transitivity, Eqn. (3), is typically not met, as is illustrated in the literature with the house-door-handle example: although the door is a functionally defined structural part of the house and the handle is a functionally defined structural part of the door, it is argued that the handle need not be taken as a functionally defined structural part of the house (e.g., [23]).

In this paper a different mereological perspective is taken on functions. The parthood relations considered here apply directly on the level of functions, that is, the part is a function (typically referred to as a subfunction) and the whole this function is part of is a function. So, when sticking to the example, in this paper parthood relations are considered between the function of the house, the function of the door and the function of the handle. Such functional parthood relations are taken to be defined by compositions and decompositions of functions as used in engineering. Functional parthood relations are more rarely considered in the literature but are finding their way to engineering ontologies [6,12,13], with some indications that also they may not satisfy mereological postulates [31,32].

FUNCTIONS

In this paper a proof is presented that the relation between subfunctions and the functions they compose cannot be taken as a formal parthood relation that meets the postulates Eqns. (1)-(3) of ground mereology. The functions that are considered in this proof are not the typical technical functions one encounters in the engineering literature; they are technical functions, but rather specific ones. This choice has the disadvantage that the functions considered in the proof are arguably not covering the whole engineering realm of technical functions. This choice has however two advantages which make them suitable for the job. The first is related to the fact that in engineering there is no consensus about what technical functions are. As said, there exist in engineering numerous accounts of technical functions [18,19,20,21,22]. The technical functions considered in this paper are now chosen such that they become instances of technical functions on a number of the available accounts. Hence, the results proved for the technical functions considered in this paper are holding for technical functions as defined by these accounts. I will later indicate which accounts are affected. The second advantage has to do with modelling functional composition: the technical functions considered allow modelling this composition in a straightforward manner.

The technical functions considered in this paper can be divided in two classes. The first class are token functions \( \phi \) modelled as transformations of tokens of flows, say flows of energy, material or signals. With a token flow is meant a specific flow that occurs only once. Say, a token flow of electrical energy \( a \) is a flow of energy that exists within one specific period in time and at one specific place in space. A second token flow of electrical energy \( a' \) may be a flow similar to \( a \) by having the same intensity and duration as \( a \). But when this second flow \( a' \) occurs in another period or at another place than \( a \), then \( a' \) is a token flow different to \( a \). A token function \( \phi \) is now represented by ordered sets \( (I,O) \) of input token flows in \( I \) and output token flows in \( O \) (this modelling is a variation of the one given in [17], section 5). For instance, the token function \( \phi = \langle a,b \rangle \) transforms the electrical energy token flow \( a \) to the rotational token flow \( b \). The token function \( \phi' = \langle a',b' \rangle \), which transforms the electrical energy token flow \( a' \) to the rotational energy token flow \( b' \), is then another token function than \( \phi = \langle a,b \rangle \) as soon as \( a \) and \( a' \) are different token flows, or \( b \) and \( b' \) are different token flows.

In engineering, functions are typically not token functions: the above functions \( \phi = \langle a,b \rangle \) and \( \phi' = \langle a',b' \rangle \) are typically taken together as instances of one and the same function. For being able to express this generalisation, a second class of type functions \( \Phi \) is introduced. Define types of flows \( X \) as equivalence classes \( \{x,x',x''\ldots\} \) of token flows that are similar in terms of having the same content, intensity and duration. Two token functions \( \phi = \langle I,O \rangle \) and \( \phi' = \langle I',O' \rangle \) can then be taken as similar if their input flows in \( I \) and \( I' \) are pair-wise the same types of flows and if the output flows in \( O \) and \( O' \) are pair-wise the same types of flows. Define then type functions \( \Phi \) as equivalence classes \( \{\phi,\phi',\phi''\ldots\} \) of similar token functions \( \phi = \langle I,O \rangle \), \( \phi' = \langle I',O' \rangle \), and so on. A type function can then be represented as \( \Phi = \{\langle I,O \rangle, \langle I',O' \rangle\ldots\} \). So, if the token flows of electrical energy \( a \) and \( a' \) are instances of the same type of flow, and if the token flows of rotational energy \( b \) and \( b' \) are instances of the same type of flow, one has that \( \phi = \langle a,b \rangle \) and \( \phi' = \langle a',b' \rangle \) are instances of the same type of function, namely the type function \( \Phi \) given by \( \{\langle a,b \rangle,\langle a',b' \rangle\ldots\} \).

Type functions as defined above may come closer to functions regular considered in engineering, yet type functions are still rather specific. Engineering functions are typically defined as operations with variable input, say electrical currents within a certain range of amperage, or, in the case of signals, “zero’s” and “one’s”. Type functions are however referring to only transformations that have input flows of one specific type, say on electrical currents of 2 ampere only, or on “zero’s” only.

Token and type functions can be represented by arrows for flows and boxes for transformations, see Fig. 1. For token functions the possibility of this representation is straightforward, for type functions it has to be kept in mind that type functions are represented by token functions that are instances of the type.

This arrow-box representation of functions is used in various engineering accounts of functions, e.g., [1,10,12,33,34,35,36,37,38], though not in all. 

---

2 Arrow-box representations of functions are rejected in, e.g., [2,39].
specifically, it can be argued that token and type functions as considered in this paper can be taken as functions on these accounts. Token and type functions should, as said before, not be taken as typical engineering functions, but they are transformations of flows, and thus functions on all mentioned accounts in which functions are modelled by, represented by, or simply taken as transformations of flows. Hence, the result proved for the technical functions considered in this paper, then holds for technical functions as acknowledged in these accounts.

One issue that may jeopardise this last conclusion is that individual accounts may impose additional constraints on functions. Modarres and Cheon [35] explicitly require that the representations of technical functions satisfy conservation laws as given by natural science: the input flows listed in representations as given in Fig. 1, should together have equal energy as the listed output flows have, should together have equal charge, and so on. For instance, the function of transforming electrical energy to rotational energy with an efficiency of less than 1 should by Modarres and Cheon always be taken as a function that transforms electrical energy to rotational energy, and to, say, thermal energy. Keuneke [34] and Lind [1] however allow also for functions in which input and output flows do not match in terms of energy or any other conserved quantity (this tolerance does not imply that the technical phenomena that realise functions violate conservation laws, the tolerance is merely that these laws may be ignored in representations of functions). Especially for Lind creation of energy is an acceptable function (of, say, a battery). Secondly, in many accounts (e.g., Pahl and Beitz [10] and Stone and Wood [36]) technical functions are considered as being composed of basic functions consisting of basic transformations of basic flows. In order to let token and type functions as considered in this paper be instances of all the mentioned accounts, I will assume here too that the token functions \( \phi = (I,O) \) (and thus the type functions as well) meet conservation laws and consist of basic transformations of basic flows as defined in the libraries given in [40]. The cases used actually consist of transformations of one or two flows to one or two other flows, and I will identify for a number of cases what the flows can be taken to represent.

### COMPOSITION OF TOKEN FUNCTIONS

Because token functions are modelled as transformations of token flows, one has a straightforward way to also model the composition of those token functions: token flows that are shared by token functions represent immediately the connections between these functions.

Consider two token functions \( \phi_1 = (I_1,O_1) \) and \( \phi_2 = (I_2,O_2) \). These functions can compose to a third token function in three ways: (i) in parallel, which means that no token flows are shared by \( \phi_1 \) and \( \phi_2 \), (ii) in series, which means that some flows that are output of \( \phi_1 \) are input to \( \phi_2 \), or vice versa, or (iii) by a loop, meaning that some output flows of \( \phi_1 \) are input flows to \( \phi_2 \) and that some output flows of \( \phi_2 \) are input flows to \( \phi_1 \). Because the flows are tokens, the representations \( \phi_1 = (I_1,O_1) \) and \( \phi_2 = (I_2,O_2) \) of the individual functions already contain all information about how \( \phi_1 \) and \( \phi_2 \) are connected: (i) in parallel if no token flows are shared by \( \phi_1 \) and \( \phi_2 \), (ii) in series if there are token flows in \( O_1 \) that are also in \( I_2 \), or vice versa, and (iii) in a loop if there are flows in \( O_1 \) that are also in \( I_2 \) and if there are flows in \( O_2 \) that are also in \( I_1 \). For instance the two functions \( \phi_1 = (a,\{b,c\}) \) and \( \phi_2 = (b,\{d,e\}) \) are connected in series because the flow \( b \) is in the output of \( \phi_1 \) and in the input of \( \phi_2 \). In Fig. 2 this composition is depicted with an arrow-box representation. Moreover, since this composition lead to a net overall transformation of the token flow \( a \) to the token flows \( c, d \) and \( e \), the composed token function \( \phi \) is represented by \( (a,\{c,d,e\}) \).

Before generalising this approach to the composition of \( n \) functions, I return for a moment to the modelling of an individual token function \( \phi = (I,O) \). For an individual token flow \( a \) participating in this token function, there are three possibilities: (i) flow \( a \) is in \( I \), (ii) flow \( a \) is in \( O \), or (iii) flow \( a \) is both in \( I \) and \( O \). Possibility (iii) amounts to a feedback loop: flow \( a \) is output and input of \( \phi \), as in \( (a,\{b,c\}) \) (see Fig. 3). In engineering such feedback loops are typically accepted in the modelling of individual functions. Formally one can also accept the further possibility that token functions transform only feedback loops, as in \( (a,a) \) and in \( (k,l,\{k,l\}) \). Such token functions have no net input and no net output. The function of an ideal waste dumping site may be taken as such a token function: all output flows are reabsorbed by being input flows as well. Yet, from an engineering point of view one may take these cases as technically meaningless, and a condition on the modelling of token functions that rules them out is:

\[
O \cap I \neq \emptyset \tag{6}
\]
Token functions with no net input and no net output will resurface when functional composition is modelled and it is proved that such compositions do not define a formal parthood relation for functions. So, keep in mind that they are formally possible but problematic from an engineering point of view.

Impossible cases are given by flows \( a \) that occur more than once in \( I_i \) or more than once in \( O_i \); a token flow \( a \) cannot enter or leave a transformation twice or more often. These cases are avoided in the modelling by taking \( I_i \) and \( O_i \) as sets; yet, a consequence is that the splitting and merging of flow has to be modelled as transformations of different token flow, as in \( \{a,a'a''\} \) and \( \{a'a'',a\} \), while expressing separately that \( a \), \( a' \) and \( a'' \) are, say, all token flows of electrical energy, all flows of water or all flows of signals.

Consider now, for modelling functional composition, \( n \) token function \( \phi_1 = (I_1,O_1) \), \( \phi_2 = (I_2,O_2) \), \ldots \( \phi_n = (I_n,O_n) \), and let \( \text{Comp}(\phi_1,\phi_2,\ldots,\phi_n) \) be the token function to which these \( n \) token functions compose. Because the sets \( \{I_1,O_1,I_2,O_2,\ldots,I_n,O_n\} \) contain token flows, there is again a limited number of possibilities for the occurrence of a specific token flow: a flow \( b \) can occur (i) in one input set \( I_i \), (ii) in one output set \( O_i \), (iii) in both one output set \( I_i \) and one input set \( O_i \), or (iv) in both one output set \( I_i \) and one input set \( O_i \), with \( i \neq j \). What is impossible is that a token flow \( b \) occurs in two or more different input sets \( I_i \), \( I_j \), \ldots, or in two or more different output sets \( O_i \), \( O_j \), \ldots: a token flow cannot simultaneously be input to two different functions, or simultaneously be output of two different functions. Conditions on sets of token functions that compose another function are therefore:

\[
I_i \cap I_j = \emptyset \quad \text{with: } i,j=1,\ldots,n; i \neq j \tag{7}
\]
\[
O_i \cap O_j = \emptyset \quad \text{with: } i,j=1,\ldots,n; i \neq j \tag{8}
\]

In case (i) the flow \( b \) represents a flow that is merely input to one of the functions \( \{ \phi_1,\phi_2,\ldots,\phi_n \} \) and thus also input to the composition \( \text{Comp}(\phi_1,\phi_2,\ldots,\phi_n) \). In case (ii) the flow \( b \) represents a flow that is merely output of one of the functions \( \{ \phi_1,\phi_2,\ldots,\phi_n \} \) and thus also output of the composition \( \text{Comp}(\phi_1,\phi_2,\ldots,\phi_n) \). In case (iii) the flow \( b \) represents a feedback loop as depicted in Fig. 3 for one of the functions in \( \{ \phi_1,\phi_2,\ldots,\phi_n \} \), and in case (iv) the flow \( b \) represents a connection, as depicted in Fig. 2, between two functions in \( \{ \phi_1,\phi_2,\ldots,\phi_n \} \). Flows that represent loops or connections, i.e., cases (iii) and (iv), are flows that are ‘locked in’ by the composition, and are therefore neither input to, nor output of \( \text{Comp}(\phi_1,\phi_2,\ldots,\phi_n) \). The input flows to \( \text{Comp}(\phi_1,\phi_2,\ldots,\phi_n) \) can now be determined as all input flows to the individual functions in \( \{ \phi_1,\phi_2,\ldots,\phi_n \} \) save those flows that are locked-in by \( \text{Comp}(\phi_1,\phi_2,\ldots,\phi_n) \). In a similar way the output flows of \( \text{Comp}(\phi_1,\phi_2,\ldots,\phi_n) \) can be fixed.

The set \( L \) of flows locked-in by \( \text{Comp}(\phi_1,\phi_2,\ldots,\phi_n) \) is given by the union:

\[
L = \bigcup_{i=1,\ldots,n} O_i \cap I_j \tag{9}
\]

Hence, the composite function \( \text{Comp}(\phi_1,\phi_2,\ldots,\phi_n) \) with \( \phi_i = (I_i,O_i) \) is represented by:

\[
\text{Comp}(\phi_1,\phi_2,\ldots,\phi_n) = (\bigcup_{i=1,\ldots,n} I_i \cup \bigcup_{i=1,\ldots,n} O_i) / L \tag{10}
\]

It may happen that all flows participating in a composite function \( \text{Comp}(\phi_1,\phi_2,\ldots,\phi_n) \) are locked-in, creating again the possibility to have a token function with no net input and no net output. Consider, for instance the composition of the token functions \( \phi_1 = (\{a,b\},\{c,d\}) \), \( \phi_2 = (\{c,d\},\{a,e\}) \) and \( \phi_3 = (\{e,b\}) \). The composite function \( \phi = \text{Comp}(\phi_1,\phi_2,\phi_3) \) is then given by \( \phi = (\emptyset,\emptyset) \) (see Fig. 4).

Such composite token functions with no net input and no net output are again from a formal point of view possible outcomes of composition. And such composite functions still arise when one adopts condition Eqn. (6) for avoiding that the token functions \( \phi_1 \), \( \phi_2 \), \ldots, \( \phi_n \) in the composition are functions with no net input and no net output. So, if one also wants to rule out these composite token functions with no net input and no net output, a generalisation of the condition Eqn. (6) should be accepted:

\[
\bigcup_{i=1,\ldots,n} O_i \cap \bigcup_{i=1,\ldots,n} I_i \neq \emptyset \tag{11}
\]

Composition of type functions is not defined in this paper.\(^3\)

---

\(^3\) Attempts to define composition of type functions via composition of token functions will in general not fix one particular composite type function. For instance, a composition \( \text{Comp}(\Phi_1,\Phi_2) \) with \( \Phi_1 = \{(a,b),(a,b'),\ldots\} \) and \( \Phi_2 = \{(b,c),(b,c'),\ldots\} \) may via the token compositions \( \text{Comp}(a,b),(b,c) = (a,c) \) and \( \text{Comp}(a,b),(b,c') = (a,c') \) lead to the identification of at least two composite type functions \( \text{Comp}(\Phi_1,\Phi_2) = \{(a,c),(a,c'),\ldots\} \) and \( \text{Comp}(\Phi_1,\Phi_2) = \{(a,b'),(b,c'),\ldots\} \).
A FUNCTIONAL PARTHOOD RELATION

Assume now that the relations between token subfunctions and the token functions they compose, define a parthood relation for token functions with part-of relations \( P(\phi',\phi) \) generated by the following sufficient condition:

\[
\phi = \text{Comp}(\phi_1,\ldots,\phi_n) \Rightarrow \forall \phi'((\phi' \in [\phi_1,\ldots,\phi_n] \Rightarrow P(\phi',\phi))
\]

(12)

The parthood relation obtained by this condition describes actual states of affairs. That is, the part-of relation \( P(\phi',\phi) \) holds only if the token function \( \phi' \) is actually a subfunction of the token function \( \phi \); it does not imply that \( \phi' \) is a part of \( \phi \) if \( \phi' \) is possibly though not actually, a subfunction of \( \phi \). So, a specific token function of transforming electrical energy to rotational energy may be a part of another specific function of transporting people. But this part-of relation still does not hold for other pairs of similar token functions. This generalisation is also in principle not needed: there are clearly cases where transforming electrical energy to rotational energy is not a subfunction of the function of transporting people. When a parthood relation is introduced on the level of type functions, the parthood relation for token functions is however generalised. This generalisation may be plausible for some functions: the function of transforming electrical energy to rotational energy always have conducting electrical energy as a subfunction. Moreover, though this generalisation may not be necessary for all cases, it may nevertheless be possible without contradiction. Hence, it makes sense to just try it. An advantage of having this generalisation would be, for instance, that one can express all potential (though not actual) compositions and decomposition of functions by means of a parthood relation between functions. In the early conceptual phase of designing engineers typically use such generalisations about how functions compose and decompose (e.g., [10]).

Assume therefore that functional composition for token functions amounts also to a parthood relation for type functions, and assume that this second parthood relation is defined by the sufficient condition Eqn. (12) and by the following sufficient condition for part-of relations \( P(\Phi',\Phi) \) for type functions:

\[
(P(\phi',\phi) \wedge \phi' \text{ is of type } \Phi' \wedge \phi \text{ is of type } \Phi \Rightarrow P(\Phi',\Phi)
\]

(13)

When the conditions Eqns. (12) and (13) are also necessary conditions, problems most probably are avoided. Yet when it is assumed that the parthood relations they introduce meet the three minimal postulates Eqns. (1)-(3) of ground mereology, contradictions will occur. Two of these postulates, reflexivity, Eqn. (1), and transitivity, Eqn. (3), are generating additional part-of relations, which implies that Eqns. (12) and (13) are merely sufficient conditions. And the remaining postulate, antisymmetry, Eqn. (2), puts constraints on the set of part-of relations thus generated, constraints that are not always met.

MEETING GROUND MEREOLOGY

Before introducing cases in which the defined parthood relations for token functions and type functions run into trouble, it can be shown that for a large class of regular cases the postulates of ground mereology are satisfied. These cases include all that involve only a single functional composition or decomposition, as, for instance, the decomposition of the function of ‘packing carpet tiles’ as described by Pahl and Beitz ([10], p. 33) (see Fig. 5).

Consider any set of token functions \( \{\phi_1,\ldots,\phi_n\} \) and the token function \( \phi = \text{Comp}(\phi_1,\ldots,\phi_n) \) they compose. The condition Eqn. (12) leads then to the following part-of relations:

\[
P(\phi_i,\phi) \text{ for all } i = 1,\ldots,n
\]

(14)

The postulates of ground mereology add via reflexivity, Eqn. (1), the part-of relations:

\[
P(\phi,\phi) \text{ and } P(\phi,\phi) \text{ for all } i = 1,\ldots,n
\]

(15)

Transitivity, Eqn. (3), does not add additional part-of relations: there are no non-trivial sequences \( P(\phi,\phi') \) and \( P(\phi',\phi'') \), and for the cases that \( \phi=\phi' \) or \( \phi=\phi'' \), the part-of relations \( P(\phi,\phi'') \) that the transitivity postulate adds are already given by Eqns. (14) and (15). Antisymmetry, Eqn. (2), is met because for each non-trivial part-of relation \( P(\phi,\phi) \) there is no reverse relation \( P(\phi,\phi) \). So, say, the token function of counting carpet tiles in Fig. 5 can formally be taken as a part of the composite token function of packing those tiles.

For type functions these cases of single token compositions \( \phi = \text{Comp}(\phi_1,\ldots,\phi_n) \), are also unproblematic. Let \( \{\Phi_1,\Phi_2,\ldots,\Phi_m\} \) be the set of different type functions that occur in the set \( \{\phi_1,\phi_2,\ldots,\phi_n\} \). One then has two options. First \( \phi = \text{Comp}(\phi_1,\ldots,\phi_n) \) is of a separate type \( \Phi \in \{\Phi_1,\Phi_2,\ldots,\Phi_m\} \). Application of condition Eqn. (13) then leads to the following part-of relations:

\[
P(\Phi_i,\Phi) \text{ for all } j = 1,\ldots,m
\]

(16)

\[
P(\Phi,\Phi) \text{ and } P(\Phi,\Phi) \text{ for all } j = 1,\ldots,m
\]

(17)
The second option is that \( \phi = \text{Comp}(\phi_i, \ldots, \phi_n) \) is not of a separate type, i.e., \( \phi \) is of a type \( \Phi \) element of \( \{\Phi_1, \Phi_2, \ldots, \Phi_m\} \). The part-of relations then become:

\[
P(\Phi_j, \Phi_k) \text{ for all } j = 1, \ldots, m, \text{ with } j \neq k
\]

\[
P(\Phi_j, \Phi_j) \text{ for all } j = 1, \ldots, m
\]

For both options all postulates of ground mereology are met. So, also the type function of counting carpet tiles in Fig. 5 can formally be taken as a part of the composite type function of packing tiles.

**SOME INITIAL NEGATIVE RESULTS**

The above results about the parthood relations between functions are positive but unfortunately also exhaustive. Checking these parthood relations against additional postulates of mereology or for cases in which one has more than one composition or decomposition quickly yields negative results.

A parthood relation that is stronger than a ground mereology is the already mentioned extensional mereology, that satisfy a fourth postulate called strong supplementation [7,8,9]. Extensional part-of relations are meeting the condition:

\[
(\exists \psi(PP(\psi, \phi)) \lor \exists \psi(PP(\psi, \phi'))) \Rightarrow \nabla \psi(PP(\psi, \phi)) \iff PP(\psi, \phi') \Rightarrow \phi = \phi' 
\]

which expresses that two functions that have the same proper parts, are the same function. Token functions may satisfy this condition, yet type functions do not.

Consider the following case of three token functions \( \phi_1 = \langle a, b \rangle, \phi_2 = \langle b, a' \rangle \) and \( \phi_3 = \langle a', b' \rangle \), where \( a \) and \( a' \) are instances of the same type flow, and where \( b \) and \( b' \) are instances of the same type flow. The first two functions compose to the token function \( \phi = \text{Comp}(\phi_1, \phi_2) = \langle a, a' \rangle \), the last two compose to \( \phi' = \text{Comp}(\phi_2, \phi_3) = \langle b, b' \rangle \). With condition Eqn. (12) one obtains for token functions the part-of relations:

\[
P(\phi_1, \phi), P(\phi_2, \phi), P(\phi_2, \phi') \text{ and } P(\phi_3, \phi')
\]

Since the token functions \( \phi_1 \) and \( \phi_3 \) are of the same type \( \Phi_1 = \{(a, b), \langle a', b' \rangle, \ldots\} \), one obtains for type functions the relations:

\[
P(\Phi_1, \Phi), P(\Phi_2, \Phi), P(\Phi_1, \Phi') \text{ and } P(\Phi_2, \Phi')
\]

with \( \Phi_1 = \{(a, b), \ldots\} \), \( \Phi = \{(a, a'), \ldots\} \) and \( \Phi' = \{(b, b'), \ldots\} \). In terms of proper-part-of relations one obtains:

\[
PP(\Phi_1, \Phi), PP(\Phi_2, \Phi), PP(\Phi_1, \Phi') \text{ and } PP(\Phi_2, \Phi')
\]

The first proper-part-of relation \( PP(\Phi_1, \Phi) \), for instance, is obtained with Eqn. (5) because one can derive for this particular case that \( P(\Phi_1, \Phi) \) does not hold: one does not have that \( \phi = \langle a, a' \rangle \) is a part of \( \phi_1 = \langle a, b \rangle \) or a part of \( \phi_3 = \langle a', b' \rangle \); so, one does not have \( P(\phi_1, \phi) \) or \( P(\phi_3, \phi) \) for token functions in this case; hence, and one does not have \( P(\Phi_1, \Phi) \) for type functions. The other proper-part-of relations in Eqn. (22) can be derived by similar reasoning, and together they prove that type functions may violate extensionality. Eqn. (4): the type functions \( \Phi = \{(a, a'), \ldots\} \) and \( \Phi' = \{(b, b'), \ldots\} \) have the same proper parts though are still different type functions.

This result is not surprising since it shows that the ordering of functions in a functional composition matters. Consider two type functions, the first \( \Phi_1 \) being the increasing of the temperature of a flow of material with 150 degrees centigrade, and the second \( \Phi_2 \) being the decreasing of the temperature of a flow of material with 150 degrees centigrade. These type functions can be composed by, intuitively, connecting them in series.\(^4\) One has however two options: first \( \Phi_1 \) and then \( \Phi_2 \) yields the composite type function \( \Phi \) of baking the material flow; first \( \Phi_2 \) and then \( \Phi_1 \) yields the composite type function \( \Phi' \) of refrigerating that flow [32]. Yet, although not surprising, it indicates that importing mereological parthood relations for functions has limited value in the modelling of functions in engineering ontologies and engineering in general.

A second result for the parthood relation for type functions generated by single compositions \( \phi = \text{Comp}(\phi_1, \ldots, \phi_n) \) is more counterintuitive. This result is that a simple and basic type function can have parts that are in general not in a sensible way related to the original type function.

Consider a case with the token flows \( a, b, c \) and \( d \), and the token functions \( \phi_1 = \langle a, b \rangle, \phi_2 = \langle b, c \rangle \) and \( \phi_3 = \langle c, d \rangle \). Let \( a \) be a flow of electrical energy, \( b \) a flow of thermal energy, \( c \) a flow of chemical energy, and \( d \) a flow of rotational energy. The token function \( \phi_1 = \langle a, b \rangle \) then transforms the token flow \( a \) of electrical energy to the token flow \( b \) of thermal energy, which is a basic function in [40]. The functions \( \phi_1, \phi_2 \) and \( \phi_3 \) compose to the token function \( \phi = \text{Comp}(\phi_1, \phi_2, \phi_3) = \langle a, d \rangle \), which transforms the token flow \( a \) of electrical energy to the token flow \( d \) of rotational energy. Condition Eqn. (12) gives the following part-of relation for token functions:

\[
P(\phi_1, \phi) = P(\langle b, c \rangle, \langle a, d \rangle)
\]

In this case the token flow \( a \) is indeed actually transformed to the token flow \( d \) via the intermediate transformations of flow \( a \) to flow \( b \), of \( b \) to \( c \), and of \( c \) to \( d \). Hence, the relation \( P(\phi, \phi) \) makes sense: the token function \( \phi \) of transforming the electrical energy flow \( a \) to the rotational energy flow \( d \) is in this case realised via the token function \( \phi_2 \) of transforming a thermal energy flow \( b \) to a chemical energy flow \( c \). But when generalising this part-of relation \( P(\phi_1, \phi) \) to type functions, the result makes less sense. With condition Eqn. (13) one obtains:

\[
P(\Phi_1, \Phi) = P(\langle b, c \rangle, \ldots, \langle a, d \rangle, \ldots)
\]

\(^4\) Formally the connection between \( \Phi_1 \) and \( \Phi_2 \) is modelled by token functions instances of \( \Phi_1 \) and \( \Phi_2 \) that have the right shared token flows.
which means that the type function $\Phi$ of transforming electrical energy flows to rotational energy flows has as a part the type function $\Phi_2$ of transforming thermal energy flows to chemical energy flows. This result generalises quickly: any basic type function $\Phi = \{a,d, \ldots\}$ has many other basic type functions $\Phi_2 = \{b,c, \ldots\}$ as its parts, where the token flows $b$ and $c$ can be flows on any type different to the types of flows of $a$ and $d$.

**VOLATING GROUND MEREOLOGY**

The last result can be used to prove that the parthood relation for type functions violates ground mereology. By this result the type function $\Phi$ and the token function $\Phi_2$ have as a part the type function $\Phi_3$ which means that the type function $\Phi$ has as a part the type function $\Phi_2$ which has as a part the type function $\Phi_3$.

For deriving Eqs. (24) and (25), one can extend the previous case to one with two functional compositions. Consider the token flows $a$, $b$, $c$, $d$, $a'$ and $d'$ and the token functions $\phi_1 = \langle a,b \rangle$, $\phi_2 = \langle c,d \rangle$, $\phi_3 = \langle b, a' \rangle$, $\phi_4 = \langle a',d' \rangle$ and $\phi_5 = \langle d',c \rangle$. The flows $a$ and $a'$ are flows of electrical energy of the same type, $b$ is a flow of thermal energy, $c$ is a flow of chemical energy, $d$ and $d'$ are flows of rotational energy of the same type.

The functions $\phi_1$, $\phi_2$, $\phi_3$, $\phi_4$ and $\phi_5$ compose to the token function $\phi_6 = \text{Comp}(\phi_1, \phi_2, \phi_3, \phi_4, \phi_5) = \langle b,c \rangle$, which gives the part-of relation:

$$P(\Phi, \Phi_6)$$

Together with the part-of relation $P(\Phi_2, \Phi)$ given in Eqn. (24) this leads to a violation of the ground mereological postulate of antisymmetry, Eqn. (2): this postulate, applied to Eqns. (24) and (25), requires that $\Phi_2$ is identical to $\Phi$, which is not the case.

For deriving Eqs. (24) and (25), one can extend the previous case to one with two functional compositions. Consider the token flows $a$, $b$, $c$, $d$, $a'$ and $d'$ and the token functions $\phi_1 = \langle a,b \rangle$, $\phi_2 = \langle c,d \rangle$, $\phi_3 = \langle b, a' \rangle$, $\phi_4 = \langle a',d' \rangle$ and $\phi_5 = \langle d',c \rangle$. The flows $a$ and $a'$ are flows of electrical energy of the same type, $b$ is a flow of thermal energy, $c$ is a flow of chemical energy, $d$ and $d'$ are flows of rotational energy of the same type.

The functions $\phi_1$, $\phi_2$, $\phi_3$, $\phi_4$ and $\phi_5$ compose to the token function $\phi_6 = \text{Comp}(\phi_1, \phi_2, \phi_3, \phi_4, \phi_5) = \langle b,c \rangle$, which gives the part-of relation:

$$P(\Phi, \Phi_6) = P(\langle a',d' \rangle, \langle b,c \rangle)$$

and thus the part-of relation Eqn. (25). The functions $\phi_1$, $\phi_2$, $\phi_3$, $\phi_4$ and $\phi_5$ compose to $\phi = \text{Comp}(\phi_1, \phi_2, \phi_3, \phi_4, \phi_5) = \langle a,d \rangle$, which again gives the part-of relations Eqns. (23) and (24), see Fig. 6:

![Figure 6: \(\Phi IS PART OF \Phi_2 AND \Phi_2 IS PART OF \Phi\)](image)

The parthood relation for token functions can also violate ground mereology when considering two compositions simultaneously. Take a case with three token functions $\phi_1 = \langle a,b \rangle$, $\phi_2 = \langle c,d \rangle$ and $\phi_3 = \langle d,c \rangle$. The first two functions compose to the token function $\phi_4 = \text{Comp}(\phi_1, \phi_2) = \langle a,c \rangle, \langle b,d \rangle$. This new function and the function $\phi_5$ compose to $\text{Comp}(\phi_4, \phi_5) = \langle a,b \rangle$, see Fig. 7. With condition Eqn. (12) one obtains:

$$P(\phi_1, \phi_4), P(\phi_2, \phi_4), P(\phi_5, \phi_4) \text{ and } P(\phi_4, \phi_5)$$

For letting this set of part-of relations meet the postulates of reflexivity, Eqn. (1), and transitivity, Eqn. (3) or ground mereology, additional part-of relations have to be added. Yet already with the four relations given in Eqn. (27) a violation with the antisymmetry postulate, Eqn. (2), can be detected; given $P(\phi_1, \phi_4)$ and $P(\phi_4, \phi_5)$, this postulate requires that $\phi_1 = \phi_4$, which is not the case.

![Figure 7: \(\phi_1 IS PART OF \phi_4 AND \phi_4 IS PART OF \phi_1\)](image)

An example of this final case can be the transport of two flows $a$ and $c$ of materials from one container to flows $b$ and $d$ in a second container, after which an unwanted flow $c$ is returned as flow $b$ in the first container.

In terms of the arrow-box representations given in Figs. 6 and 7, the violations of ground mereology by the parthood relations for token and type functions may be seen as due to that more complex composites of arrows and boxes do not necessarily represent more complex functions. Parthood relations for arrows and boxes therefore do not provide a proper parthood relation for the functions they represent.

A last observation is that in the proof that the parthood relation for token functions violates ground mereology, two functions $\phi_2$ and $\phi_3$ are considered that compose to a function with no net input and no net output. This composite $\text{Comp}(\phi_2, \phi_3)$ is not explicitly considered in the proof. When composing a set of functions, such subsets that compose to no-input-and-no-output components are formally possible. Conditions Eqns. (6) and (11) allow for the case too. But from an engineering point of view the case may be unreasonable.

**CONCLUSION AND DISCUSSION**

In this paper a proof is given that the relation between technical functions and their subfunctions cannot be taken as a formal parthood relation. The structure of this proof is as follows. Mereology provides the postulates of *reflexivity, antisymmetry and transitivity* as postulates any formal parthood relation should meet [7,8,9]. Two classes of *token technical
functions and type technical functions are defined and the functions from these classes used in the proof, are specific enough to be proper instances of functions on many engineering accounts of functions [1,10,12,33,34,35,36,37,38]. The result of the proof is in this way general: it holds for all formal parthood relations, and for all of these accounts of functions.

The generality of the proof is achieved at the expense of the realism involved in the modelling of technical functions: the token and type functions considered in this paper and the cases used in the proof are not representative to typical engineering functional descriptions; they may be taken as ‘toy examples’. This observation allows for challenging the result by requiring a proof that is based on more realistic examples of functions. Such a challenge is reasonable from an engineering point of view, and probably also effective. Yet arguing that in engineering only single compositions and decompositions are considered is less realistic: more complex cases are used in engineering (see, e.g., [10], figure 2.3). Arguing that functional decompositions in engineering are limited to only decompositions of functions into a finite set of basic functions (e.g., [1,6,10,34,36]) may also not do the job: the functions and functional compositions considered in the proof are rather simple. But one can argue against the last proof for token functions, by holding that engineers would typically not compose functions \( \{ \phi_1, \phi_2, \phi_3 \} \) where some of the functions compose to a function \( \text{Comp}(\phi_2, \phi_3) \) that has no input and no output. Finally, as a last resort, one can hold that engineers need not accept a formal parthood relation.

This line of reasoning is not tenable when considering engineering ontologies. Parthood relations are in ontology taken as fundamental, general and well-defined relations which do meet the postulates of (ground) mereology. The proof given in this paper shows that this fundamental, general and well-defined relation is not to be used for modelling the relation between subfunctions and functions. Also the charge that the proof is based on toy examples loses its thrust in ontology. Engineering ontologies are developed in part with the aim of enabling automated reasoning, including functional reasoning. Automated functional reasoning is already being developed also outside of the formal confines of ontologies: an example in design methodology is [41]. Algorithms that help guide that reasoning now may easily generate the very toy examples that were considered in this paper, including cases \( \{ \phi_1, \phi_2, \phi_3 \} \) with \( \text{Comp}(\phi_2, \phi_3) \) being a function with no net input an no net output. Such ontological reasoning will run into precisely such contradictions as given in this paper, when subfunctions are taken as parts of the functions they compose.

ACKNOWLEDGMENTS

The modelling of functions builds on joint work with Stefano Borgo, Massimiliano Carrara and Pawel Garbacz. Research for this paper was supported by the Netherlands Organisation for Scientific Research (NWO).

REFERENCES
