# EXPLORING POTENTIALS FOR CONSERVATIONAL REASONING USING TOPOLOGIC RULES OF FUNCTION STRUCTURE GRAPHS

**Chiradeep Sen, Joshua D. Summers, and Gregory M. Mocko**
Clemson University

## ABSTRACT
This paper explores the possibility of supporting automated function-based reasoning in the conceptual design phase, specifically, reasoning needed to perform physics-based concept validation. Eleven atomic tasks of topologic reasoning, divided in two categories, connectedness and derivation, are identified that could be used to check graph-based function structures against conservation laws using only the count and types of flows attached to the functions. This reasoning is illustrated by simulating the sequential actions of a designer developing a new mechanical device. The algorithms are validated by implementing them in a function modeling software that first allows the designer to construct a function model and then detects violations of conservation laws in the model appropriately.

*Keywords: mechanical function, formal representation, design reasoning, conceptual design*

## 1 MOTIVATION AND APPROACH
This paper explores the possibility of using formalized function-based representations of mechanical systems to support automated analytical reasoning in conceptual design, specifically those required in physics-based concept validation. As the major objective of conceptual design is to synthesize solutions, much of previous research in function-based reasoning attempt to automate synthesis tasks such as solution search [1, 2] and concept generation [1, 3, 4]. In complement, designers could also benefit from automated *analysis* support at the conceptual level. Conceptual analysis could help with assessing feasibility, exploring working principles or solutions, estimating number of parts, assembly time, and cost, or analyzing the possible modes and probabilities of failure. Presently, these tasks are performed with limited automation, using manual understanding of the physical principles that govern mechanical systems. Answering these questions through automation in the conceptual phase will expose more facts about the merits of a concept early in the process, and thus could enable better informed, more objective, and earlier decisions—a process referred to as "Front Loading" [5]. Exploring and articulating these early reasoning possibilities is the overall aim of this research.

Specifically, this paper identifies reasoning necessary to support checking concept models against the laws of conservation of material and energy. This conservational reasoning could help to determine, for example, (1) if a flow or function that is a necessary part of the concept's functionality has been overlooked in a model (correctness), (2) if the concept or any of its subsets implies more than 100% efficiency (validity), (3) how many residual flows does the concept produce and how much energy or material is wasted through those (emissions), or (4) if the concept is realizable within the given energy or material constraints (feasibility).

In this paper, a set of atomic reasoning actions are first identified and used to support basic, qualitative conservational reasoning. The algorithms are validated by implementing them in a function modeling software that first allows the designer to construct a function model and then detects violations of conservation laws in the model appropriately. Before exploring these possibilities, the advances in function representation and reasoning are briefly reviewed next.

## 2 FUNCTION REPRESENTATIONS AND REASONING IN DESIGN
Historically, research in the function representation has been driven between two viewpoints. In Artificial Intelligence (AI), multiple models exist mainly to support device description and design synthesis. The second viewpoint, named here the *engineering design view*, primarily uses the graph-based function structure to support different design reasoning. These views are briefly discussed next.

## 2.1 The AI Views of Function

The automation of function-based thinking is an active AI research area and multiple approaches, ranging from representations [6, 7], languages [8, 9], ontologies [10, 11], and software tools [12, 13] have been proposed. Device function is described as the interaction between several elements such as the artifact's structure and behavior, the artifact's interaction with its environment, and the user's intent [2, 6, 7, 14, 15]. For example, function is defined as "the relation between the goal of a human user and the behavior of a system" [16]. The Function-Behavior-Structure model [17] describes function as requirements that the device performs. This model has been used to explain creativity [18] and later extended to include situatedness [14]: the dynamic situation where the information available to and represented in design influences the designer's decisions. Similarly, the Function-Behavior-State model defines functions as "a description of (the device's) behavior abstracted by human through recognition of the behavior in order to utilize it" [7]. This model supports problem decomposition and is implemented as the FBS-modeler tool that has been experimentally used in reducing functional redundancy of devices [13]. A related model, Structure-Behavior-Function, defines function as a set of the device's input and output states and the behavior that causes state change [1]. Using design patterns to capture these information, this model simulates "learning of high-level abstractions and their use in reminding and adaptation" [19], thus supporting analogical reasoning. It is implemented in case-based reasoning tools named IDeAL [2, 20] and Kritik [1]. Finally, Functional Representation (FR) defines function as the device's effect on the environment [21] and using a "device-centric view" [6]. This model supports failure diagnosis through causal analysis [22] and was implemented in the Causal Functional Representation Language (CFRL) that describes function as the triple $\{D_F, C_F, G_F\}$, indicating the device, the context of the device's application, and the goal or desire of the user [8]. CFRL can describe how a device works using causal process descriptions [23]. In summary, the AI models include the complex interaction between multiple entities, are primarily descriptive, and are subject to the difficulties of modeling intentionality [24]. By contrast, the engineering design view, described next, adopts a transformational view of functions to support design reasoning.

## 2.2 The Engineering Design View

The engineering design view defines function as transformations of a set of material, energy, or signal flows into another set [25, 26]. A graph-based representation, the function structure, where the nodes are functions and the edges are the flows, is widely used [25-27]. Figure 1 shows a functions structure for a commercial hairdryer product, where the flow name abbreviations are clarified in the legend. This representation is discussed as a potential means to solve conceptual design tasks such as problem decomposition [25, 26], solution search [27], and concept generation [3, 4], and also for reverse and engineering tasks such as design understanding and archiving [28, 29].

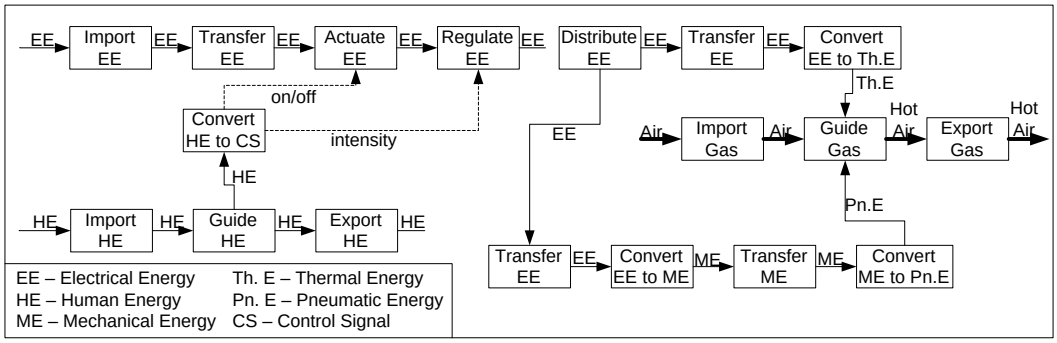



*Figure 1: Function structure of a hairdryer product stored in the Design Repository*[1]

Controlled vocabularies of function terms have been proposed with the aim to improve the consistency of term usage in function structures and the possibility of automated reasoning. Examples are the 46 function verbs and 40 flow adjectives discovered through engineering forensic studies [30], the vocabulary of *motion, control, power, and enclose*, [31], the vocabulary compiled by Szykman et al. [32], and the Functional Basis vocabulary [33, 34], which contains 53 function verbs and 45 flow nouns. The terms in the Functional Basis were identified through tear-down of electromechanical

[1] http://repository.designengineeringlab.org/ accessed on December 31, 2010

consumer products following reverse engineering guidelines [26] and used to catalogue functional information of more than 165 products that are stored in the Design Repository [35]. The function structure shown in Figure 1 is obtained from this archive.

The engineering design viewpoint supports reasoning mainly by reusing knowledge found in previous solutions to solve new problems. For example, the product information archived in the Design Repository has been used to develop a concept generator tool [3], a tool to computes the functional similarity between products [36], a failure-prediction tool [37], a graph grammar tool to synthesize function structures from a black box [4], and a tool to configure components for new design [38].

In summary, two interrelated gaps in function-based reasoning are identified by reviewing these tools. First, reasoning based on the knowledge of the governing physics of engineering systems has not been explored. Rather, the existing tools generally rely on analogical reasoning. For example, the graph grammar tool [4] uses trends of flow transformations found in the Repository models to derive its rules, the concept generation tool [3] uses functions and flows that satisfied a function previously to generate new solutions, and the failure-prediction tool [37] uses the historical failure data of function-flow modules for its algorithms. While this analogical approach may be adequate or even appropriate for these tools, essentially their propositions are rendered acceptable because they are predicted by the trends, rather than because they are mandated by a physical law or its implications. Second, design reasoning based on the meanings or definitions of functional terms is not supported. For example, the above three tools use the names of the functions and flows to establish the trends used for analogy.

These two gaps are interrelated, as physics-based functional reasoning would require capturing the physical laws in a function representation, which in turn would require formalizing the terms used in that representation. However, while such formalization could potentially support advanced physics-based reasoning at the quantitative level, some qualitative yet useful conservation-based reasoning can be performed purely based on the topology of the function structure graphs. Identifying these reasoning needs, reducing them to the elementary reasoning tasks, and algorithmically implementing them in software for demonstration are the major deliverables of this paper.

## 3 TOPOLOGICAL CONSERVATION REASONING ON FUNCTION MODELS

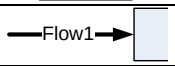
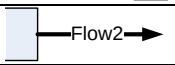
This section identifies atomic reasoning actions that can be combined to check function models for conservational validity at a qualitative level. This reasoning uses only the topological information of the model, i.e., the number and types of flows attached to the functions. Consequently, it supports only qualitative checks. For example, it can reason that if a function consumes some energy, it must also produce some energy or store all of it in a device, requiring a state based representation that is out of scope of this paper. However, it cannot reason that the quantity of energy at input and output must also equate. Consequently, it only provides low-resolution, qualitative reasoning. This topological reasoning is presented below in two parts – **connectedness** and **derivation**.

### 3.1 Connectedness Rules of Topological Conservation

The **connectedness rules** check if the model elements (functions and flows) are adequately connected and provide feedback to the modeler about open-ended elements in three possible ways (Table 1). The first column of Table 1 names these three erroneous situations, the second column shows graphical versions of these errors, while the last column shows anticipated feedback produced by an automated reasoning tool in response to these errors. The first row shows reasoning when a redundant function block is found that is included in the model but is "doing nothing", as it is not transforming any flow. The last two rows show reasoning when the tool finds flows in the model, the origin or destination of which are not identified. As implied by the messages, it is idealized that in a valid model, each flow must originate from and terminate to a function or the device's environment. In this manner, these rules can prevent the accidental modeling of flows with unaccounted existence, thus supporting conservation. Notably, the last two rules have already been implemented in a function modeling tool presented in previous research [39]. However, none of the other reasoning activities under the connectedness and the derivation category are presently supported.

*Table 1: Connectedness rules for topological conservation*

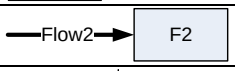
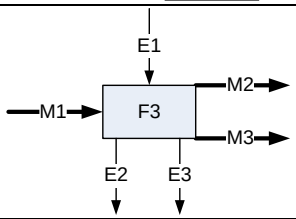
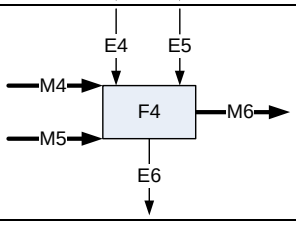
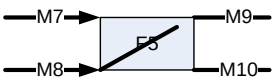
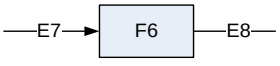
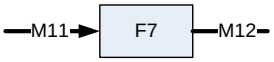
| Situation Name | Model State | Expected Feedback |
|---|---|---|

| Situation Name | Model State | Expected Feedback |
|---|---|---|
| Redundant Function |  | Redundant function: “F1” |
| Dangling tail |  | Dangling tail: Flow1 |
| Dangling head |  | Dangling head: Flow2 |

### 3.2 Derivation Rules of Topological Conservation

The **derivation rules** accounts for how one flow is derived from or into other flows across a function. Table 2 shows eight atomic rules that can be combined to support topological conservation. Since topological reasoning implies checking flow types attached to functions, a vocabulary of flow types is needed. Here, a basic, two-term vocabulary is used to provide the lowest possible resolution in the flow domain: Material (M) and Energy (E), which are distinguished in the figures in the table by thick and thin arrows, respectively. Signal flows are not included here as they are not conservable entities, rather they are captured as parameter values associated with material and energy flows to which an appropriately designed agent shows a specific behavior, which constitutes “sensing”. For example, electrical signals are often interpreted by sensors using its voltage. Without the sensor, the signal does not exist, despite the electrical energy flow, and while a signal is sensed, there is no requirement for the sensor to produce a signal to obey conservation laws.

*Table 2: Derivation rules for topological conservation*

| # | Situation Name | Model State | Expected Feedback |
|---|---|---|---|
| 1 | Orphan flow |  | Orphan flow: Flow1 |
| 2 | Barren flow |  | Barren flow: Flow2 |
| 3 | One In – Many Out |  | Conservation inferred:<br>{M1} → {M2, M3}<br>{E1} → {E2, E3} |
| 4 | Many In – One Out |  | Conservation inferred:<br>{M5, M5} → {M6}<br>{E5, E5} → {E6} |
| 5 | Many In – Many Out |  | Out of date: Unable to infer conservation. Please specify. |
| 6 | Many In – Many Out with User-Defined Derivation |  | Conservation inferred:<br>{M7} → {M10} |
| 7 | Energy Transformation |  | Conservation inferred: {E7} → {E8}<br>Warning: Transformation of energy without residue in “F6”. |
| 8 | Material Transformation |  | Warning: Material transformation may need enabling energy flow. |

The reasoning can be extended to individual flow sub-types within these major flow classes, but that exercise is reserved for future work. However, as indicated earlier, this reasoning does not need or use any function verb vocabulary. Thus, when using the function modeling software tool (Section 5) to construct a model, the modeler must instantiate every flow from one of the two classes, M or E, or their sub-classes, while she can use any plain-text description for the functions.

Here modeling scope is limited to conservation within the major flow types, i.e., material cannot be converted into energy and vice-versa. Thus, all material input must be derived into material output and all energy input must be conserved as energy across a function. A parent-child relation is provided between the flows within a model to describe the derivation relations. For example, when Flow1 is input to a function and is derived into Flow at output, Flow1 is a parent of Flow2 and Flow2 is a child of Flow1. In general, a flow can have one or more children and one or more parents, as long as they are of the same type as the flow itself. The eight reasoning rules are explained next, with reference back to the rows in Table 2.

**(1) Orphan Flow and (2) Barren Flow:** A flow without parents is called an orphan flow (row 1), while a flow without children is called barren (row 2). The orphan flow rule triggers when a function produces an output flow without consuming at least one input flow of the same type. Similarly, the barren flow rule triggers when a function consumes a flow of M or E, but produces no flow of the same type. Accordingly, both F1 and F2 in the table have conservational errors, since F1 produces Flow1 without consuming any flow of the same type, while F2 receives Flow2 but does not produce any flow of that type. The only exceptions to these rules are for a function that supplies a flow (such as Flow2) from storage or stores receives a flow (e.g., Flow1) without any loss. For reference, the Functional Basis vocabulary [34] has two verbs, Store and Supply, for this purpose. These exceptions can be built into the reasoning algorithms as the definitions of these verbs are formalized, which is also reserved for future extensions.

**(3) One In–Many Out:** This rule triggers when a function receives only one input of a given type (M or E) and produces multiple flows of the same type. As a result, assuming that the model is complete with no other input flow of that type, the rule assigns an inferred parent-child relation, where all the output flows of a major type are inferred to be derived from the single input of that type. Since material and energy are conserved separately, both inferences in the third row can be supported from the shown model.

**(4) Many In–One Out:** This rule triggers when a function receives multiple flows of a given type (M or E) and produces only one flow of that type. In this case, the tool infers that all the input flows of a give type join up to form the only output flow of that type.

**(5) Many In–Many Out and (6) User-Defined Derivation:** This rule triggers when a function both receives and produces multiple flows of the same type (M or E). In this condition, an appropriate inference about their derivations cannot be conclusively drawn and hence the rule sets the model state to "out of date" and asks the user to explicitly define the derivation relations. As soon as the user defines enough explicit relations so that inference can be drawn between the remaining flows (row 6), the system draws those inferences and reports as a message. In row 6, the diagonal arrow through the function block implies this user-defined assignment of a derivation relation from M8 to M9.

**(7) Energy Transformation:** This rule triggers whenever a function processes energy flows at input and output. This rule works on the premise that most energy transformations incur some loss. However, this premise is not universally true, as the concept of loss is not formalized and may vary between different applications of a function. Therefore, the message is presented as a warning, rather than an error. The purpose of this rule is to draw the modeler's attention to the possible error, so that the inferences can include both the useful output flow and the residual flow, when applicable.

**(8) Material Transformation:** This rule produces a warning message to alert the modeler that any time material flows are being transformed across a function, the function may need to consume some form of energy to support that action. Similar to the previous rule, the premise of this rule is also not universal, although it is typical. For example, a fan that converts static air to moving air, or a heater that converts cold water to hot water, does so by consuming some energy and adding a converted form of that energy to the material flow.

A notable limitation of these derivation rules is seen in the case of **splitting and recombining** of multiple input flows into multiple output flows, as shown in Figure 2. As both input flows are split into multiple components that are subsequently joined to form the two output flows, no inference can be drawn about the conservational validity of the function unless quantitative data about the input, output, and the proportions of the split flows are available. As indicated earlier, this type of reasoning could be addressed if the shown function was decomposed using functions that are instances of

formalized verb classes. However, that extension is out of this paper's scope and is reserved for future extension. In the next section, the topological reasoning rules (connectedness and derivation) are used to illustrate how they could support useful model checking reasoning. Considering three discrete options for flow count (Zero, One, Many) at the two sides of a function (Input, Output), there are only nine permutations possible. While the permutation {Zero input, Zero Output} is covered by the first row of Table 1, the other eight are captured in Table 2.

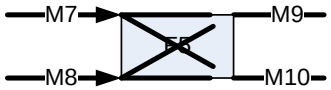



*Figure 2: A topological case unsupported by the qualitative derivation rules*

## 4 ILLUSTRATION OF TOPOLOGICAL CONSERVATION REASONING

The seven steps in Figure 3 represent in-process states of a black-box function structure that is under construction for exploring possible functional architectures, using a function modeling software that captures these reasoning algorithms. A black-box model shows the overall functionality of a design, typically using only one function [25]. The design problem states "Design an air heating device that intakes air from one location in a house and delivers hot air to another location". As the designer adds functions and flows to the model, the software performs real-time model checks using the conservation rules and provides feedback, as described next.

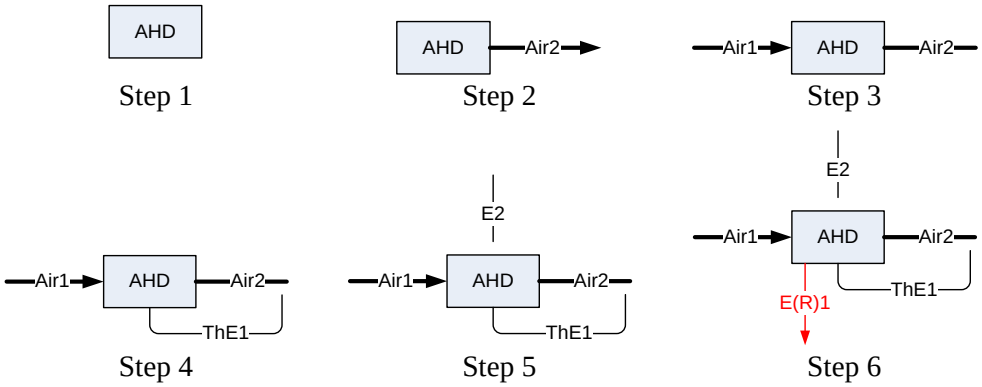



*Figure 3: Modeling steps of an Air Heating Device*

**(1)** **D**esigner adds a function named **AHD** (air heating device), representing the overall action of the design. In response, the system identifies it as a redundant function. At this point, this reasoning is trivial, although valid. The designer ignores this message and proceeds to draw the flows that she thinks necessary.

**(2)** **T**he designer adds a material flow, named **Air2,** to indicate that the function must produce a flow of air. As typical of concept exploration, she may not be concerned about the source of this flow at this time and only wants to model that a flow is necessary. However, the system identifies that Air2 has a dangling head and is an orphan flow. The designer is less concerned about the dangling head message, as she knows that all outgoing flows of a black-box model must be eventually released to the environment, which she saves for later.

**(3)** **T**he designer adds an input material, **Air1,** in response to the orphan flow message. The system infers {Air1} → {Air2} as both are material instances and eliminates the orphan flow message. However, it adds that Air1 has a dangling tail and a warning message that the material transformation from Air1 to Air2 may need input energy. Notably, the parents of Air1 and the children of Air2 are not included in the model, but the system does not identify them as orphan or barren flows. This is because the derivational checks trigger only on the flows that pass the appropriate connectedness checks. For example, the orphan check runs only on flows that do not have a dangling tail and the barren flow check runs on flows that do not have a dangling head, as otherwise many of the orphan or barren messages would be

trivial. The designer ignores the connectedness messages and the warning, and proceeds to the next step.

**(4)** **T**he designer adds an energy flow, **ThE1 (Thermal Energy 1),** in order to express the idea that the overall function must somehow produce heat as output and add that to the outgoing energy flow. Notably, the designer uses the carrier-carried relationship between flows [40] to express this idea. Here the heat (ThE1) is carried by the carrier, Air2. However, in addition to the warning message, the system identifies that ThE1 is an orphan, as no parent flow has been modeled yet. This prompts the designer to model an input energy flow, as shown next.

**(5)** **T**he designer adds **E2, an energy input,** in response to the orphan message. In response, the system eliminates the orphan message and the warning from step 3, and infers ~~{E2}~~ {ThE1}. However, the system now warns the designer that although the inferred relation satisfies conservation of energy, she probably forgot to model residual energy flows during energy transformation. The designer reflects upon this warning and realizes that the system will most likely produce some wasted energy, such as heat, sound, mechanical vibration. Note that heat produced by the device that is not added to the air flow must be modeled as residual.

**(6)** **T**he designer models a **residual flow, E(R)1,** in response to the previous message. The system now only returns the connectedness messages that say that every flow except ThE1 has either a dangling head or a dangling tail, as seen from step 6 in Figure 3.

**(7)** **T**he designer adds **environment instances to the dangling edges** (not shown in Figure 3), in response to the messages. The system now accepts the model as "conservation-wise" valid.

The above steps illustrate, through an example modeling session, how real-time model checks could assist the exploration of design concepts that are conservation-wise valid. The implementation of the software is likely to involve peripheral issues such as the distraction from conceptual thinking caused by the software's messages. These issues can be addressed during implementation by providing the flexibility to turn off real-time feedback and allowing on-request model checking, where all the checks would be executed on the current state of the model.

Figure 4 shows a partially developed concept of the air heater, derived by decomposing the black-box model using the software. The steps of this decomposition are omitted for conciseness. However, the topological rules (connectedness and derivation) could be applied to each step of this process. The messages produced by these rules run on this model are shown to the right of the model and can be verified by checking the model against the rules in Table 2. As seen here, the model is accepted by the system, given the inferred conservation relations. Thus, it is illustrated that the topological rules not only apply to simple models such as a single-function black-box, but also suffices to check the validity of larger, decomposed models. The limitation arising from the splitting and recombining situation is anticipated—although not proven—as rare and therefore these rules form a comprehensive set of topological validation rules for the majority of modeling situations.

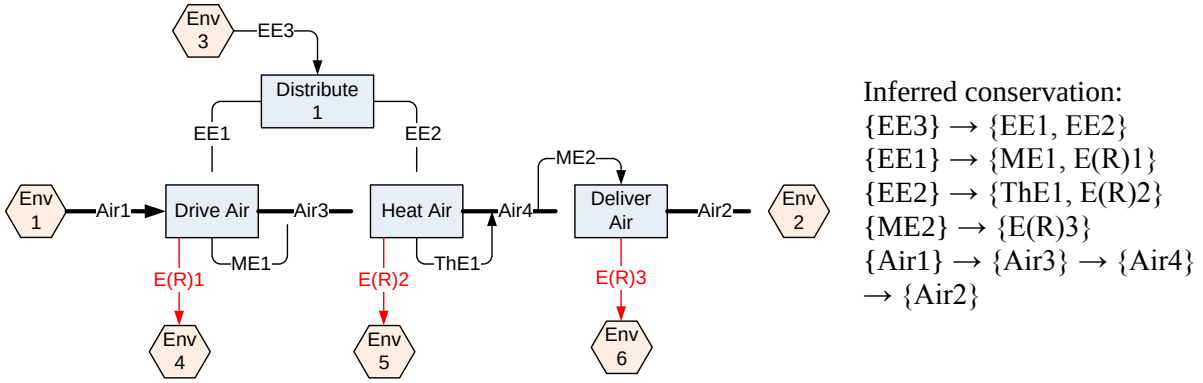



*Figure 4: Partially developed model of the air heating device and output messages*

Further, the benefit of this topological reasoning toward validating concepts against the conservation laws can be illustrated by comparing this model (Figure 4) with the model of the hairdryer (Figure 1), which is also essentially an air heating device with similar sub-functions. Notably, none of the functions in Figure 1 show any residual flow, thus implying 100% efficiency. Further, the Guide Gas function in Figure 1, analogous to the Deliver Air function in Figure 4, violates conservation laws, as its two energy input flows (ThE and PnE) are not balanced at its output. Overall, the model in Figure 1 does not objectively describe that the incoming flow of EE (top left corner of Figure 1) is ultimately manifested as two usable energy flows (ThE and PnE) that are carried away by the outgoing Hot Air flow, plus residual flows lost through the system boundary. The model in Figure 1 was created through reverse engineering, where the physical product was examined in order to identify its functions, and yet, the residual flows were omitted. It is anticipated that by using a reasoning system such as the topological reasoning mentioned here, these accidental omission can be avoided and the fidelity of the reverse engineered models can be improved.

By contrast, Figure 4 would explicitly account for energy conservation through the inferences, even during concept exploration in forward design. Since the four air flows are connected by the derivation relation (the last inference in Figure 4), it is possible to reason that the air flow, which is identified by four different names at four functional states, picks up ME1 from the Drive Air function, picks up ThE1 from the Heat Air function, and loses ME2 to the Deliver Air function. The last component, ME2 is the part of the flow energy in Air4 that is spent to drive the air against the frictional resistance of the delivery conduit and ultimately released to the environment as frictional heat (ER3). Thus, the energy added to Air2, compared to Air 1, must be (ME1.Energy + ThE1.Energy – ME2.Energy). Here, the suffix ".Energy" indicates the energy parameter of the preceding flow name. Further, since all functions are modeled with residual flows, it is further computable from the first three inferences that the energy content of ME1 is (EE1.Energy – ER1.Energy) and the energy content of ThE1 is (EE2.Energy – ER2.Energy). Finally, since {EE3} → {EE1, EE2}, it can be reasoned that the overall efficiency of this device concept is (ME1 + ThE1 – ME2).Energy / EE3.Energy.

## 5 VALIDATION THROUGH SOFTWARE IMPLEMENTATION

The purpose of the software implementation is only to demonstrate that the reasoning algorithms discussed above are actually implementable and do return the anticipated results. The usability of the tool is not under investigation yet and hence the user interface and graphics of the software are not examined. For the same reason, designer studies to test the usability and overall design benefit of this reasoning are reserved for the future. Figure 5 shows the modeling interface, while Figure 6 shows the reasoning output. Although the modeling steps discussed earlier are used to obtain Figure 5, for brevity, only the final model is shown. The forms for creating function and instances are also omitted. As can be inferred from Figure 5, function names, environment names, flow names, flow types within brackets, and the residual status of flows in red arrows are user input. The reasoning function is called from the menus, and returns Figure 6. As seen here, each inference discussed in Figure 4 is returned by the software. In addition, the function Distribute1 is identified to have 100% efficiency, on the ground that while this function operates on energy, no residue is produced. This detection is actually realistic, since if this function is executed through a junction box, some energy will be lost through Joule heating. This warning message can be removed by showing a residual energy flow from the Distribute1 function (not shown in Figure 5).

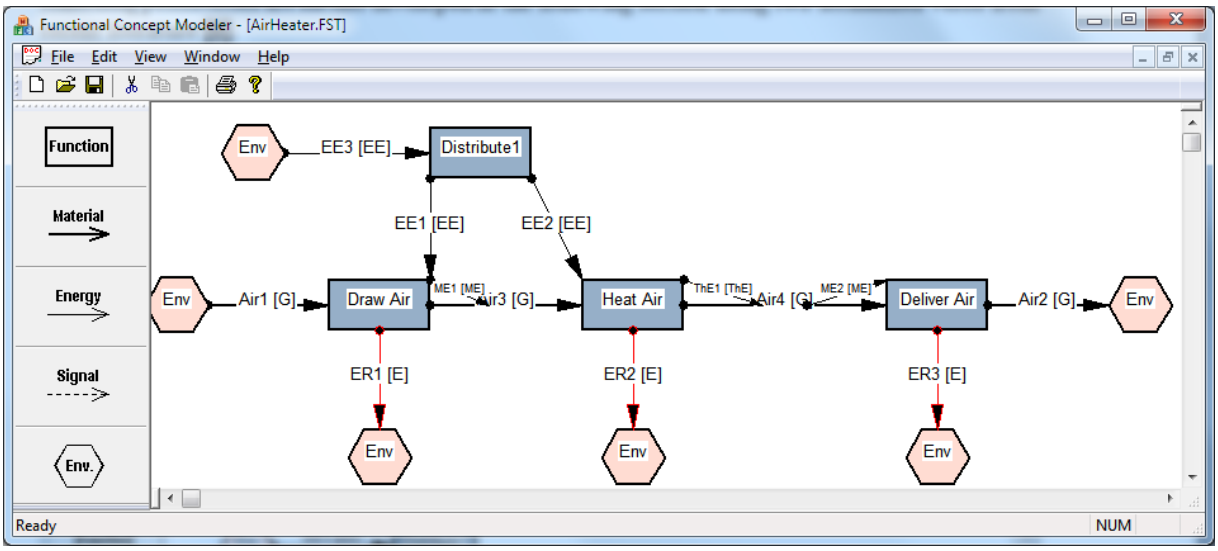



*Figure 5: Software implementation for demonstration of topologic reasoning*

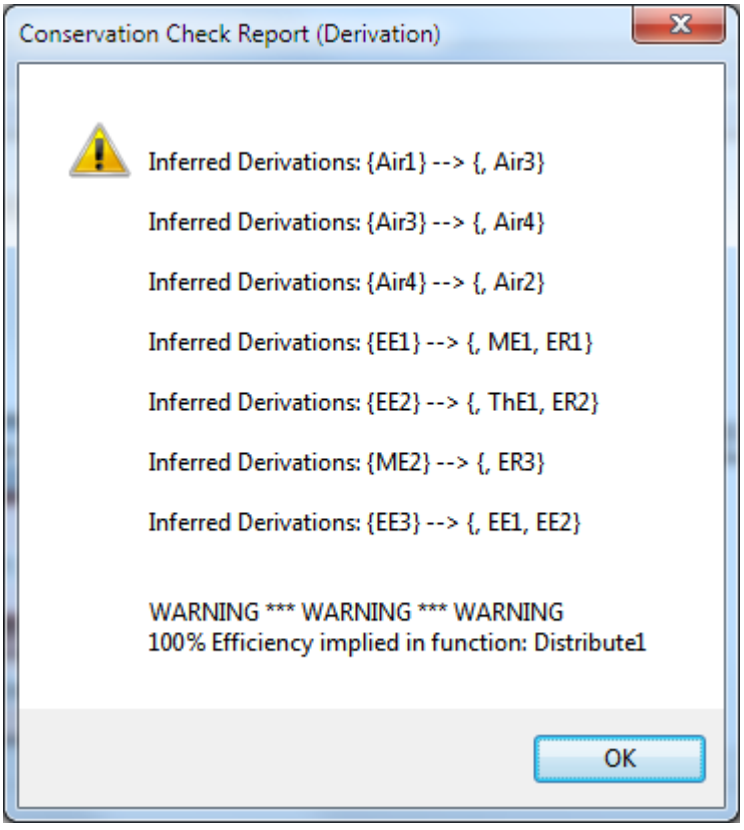



*Figure 6: Topologic Conservation Reasoning Results from Software Implementation*

## 6 CLOSURE AND FUTURE WORK

This paper illustrates the ability to support qualitative conservation-based reasoning suitable for early stage mechanical design using topological rules for function structure graphs. To illustrate this reasoning, the paper also presents a software implementation of the algorithms. Three connectedness reasoning rules and eight derivation reasoning rules are defined and illustrated. These qualitative reasoning rules are not predicated on any formal vocabulary for function verbs. This is a first step towards providing computational reasoning for concept analysis in engineering design.

As indicated previously, this paper presents a first step in this research project, which is to identify the possibility of supporting automated conservation reasoning in early design. An immediate next step is to extend this reasoning to the quantitative level. This paper identifies that the function model and its underlying function representation must capture quantifiable flow attributes for to energy and mass

properties in order to support quantified calculations and error-checking. Development of this formal representation of function verbs is currently underway. With this formalization, additional reasoning could be written to perform decomposition validation and detect if the quantity of input and output energy flows obey the balance laws of transport phenomena (quantitative model checking). For example, Figure 7 shows a possible decomposition of the Heat Air function from Figure 4.

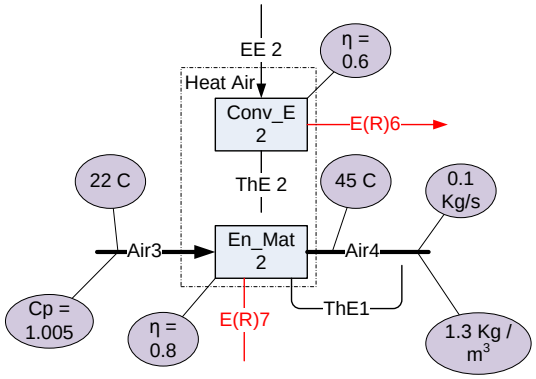



*Figure 7: Decomposition of the Heat Air subfunction of Figure 4*

The two functions are instances of two verbs to be formalized, Convert_Energy and Energize_Material and describe the conversion of EE into ThE in a heater and the addition of that heat to the Air3 flow, thus producing Air4 at a different state. The designer assigns numeric values of function and flow attributes that are known or under examination for feasibility. In Figure 7, these values are the efficiencies of the two functions (0.6 and 0.8), the temperatures of the incoming and the outgoing air flows (22C and 45C), the mass flow rate of the air (0.1 Kg/s), and some thermodynamic properties such as specific heat (Cp = 1.005 KJ/KgK) and density (1.3 Kg/$m^3$). Using this model, the following information can now be automatically computed by the software.

**(1)** **The software checks for decomposition validity** between the Heat Air function of Figure 4 and the overall model in Figure 7. This would detect that the flows attached to the boundary of both models are identical, except that both functions in the decomposed version (Convert_E2 and En_Mat2) incur residual flows. Thus, by comparison, E(R)2.Energy = E(R)6.Energy + E(R)7.Energy can be inferred.

**(2)** **The software computes the energy needed in EE2**, using the following reasoning.

$$
\begin{aligned}
ThE1.Energy &= Air3.Mass \times Air3.C_p \times \left(Air4.Temp - Air3.Temp\right) \\
&= 0.1 \times 1.005 \times (45 - 22) KJ / s \\
&= 2.31 KW \\
\therefore ThE2.Energy &= ThE1.Energy / En\_Mat2.\eta \\
&= \left(2.31 / 0.8\right) KW = 2.89 KW \\
\therefore EE2.Energy &= ThE2.Energy / Conv\_E2.\eta \\
&= \left(2.89 / .6\right) KW = 4.82 KW
\end{aligned}
$$

Quantitative calculations of the above type would require, in addition to flow attributes (e.g., mass, energy, specific heat) and function attributes (e.g., efficiency), formal representation and algorithmic application of the laws of thermodynamics and transport phenomena. While the example only illustrates reasoning for assessing feasibility or power requirements, the existing formalism of function modeling can be significantly enhanced to support formal reasoning in conceptual design by (1) first developing formal representations of the function verbs and flow nouns and (2) then extending that representation by adding information elements such as function and flow attributes. Work in direction is currently underway.

## REFERENCES


[1] A. Goel, S. Bhatta, and E. Stroulia, "Kritik: An Early Case-Based Design System," in *Issues and Applications of Case-Based Reasoning in Design*, M. L. Maher and P. Pu, Eds., ed Mahwah, NJ: Erlbaum, 1997, pp. 87-132.

[2] A. K. Goel and S. R. Bhatta, "Use of design patterns in analogy-based design," *Advanced Engineering Informatics*, vol. 18, pp. 85-94, 2004.

[3] J. Vucovich, N. Bhardwaj, H. H. Ho, M. Ramakrishna, M. Thakur, and R. Stone, "Concept Generation Algorithms for Repository-Based Early Design," presented at the ASME 2006 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Philadelphia, PA, USA, 2006.

[4] P. Sridharan and M. I. Campbell, "A study on the grammatical construction of function structures," *Artificial Intelligence for Engineering Design, Analysis & Manufacturing*, vol. 19, pp. 139-160, 2005.

[5] S. Thomke and T. Fujimoto, "The Effect of “Front-Loading” Problem-Solving on Product Development Performance," *Journal of Product Innovation Management*, vol. 17, pp. 128-142, 2000.

[6] B. Chandrasekaran and J. R. Josephson, "Function in Device Representation," *Engineering with Computers*, vol. 16, pp. 162-177, 2000.

[7] Y. Umeda, H. Takeda, T. Tomiyama, and H. Yoshikawa, "Function, Behavior, and Structure," in *Applications of Artificial Intelligence V, Vol 1: Design*, J. S. Gero, Ed., ed Boston, MA: Springer Verlag, , 1990, pp. 177-193.

[8] M. Vescovi, Y. Iwasaki, R. Fikes, and B. Chandrasekaran, "CFRL: A Language for Specifying the Causal Functionality of Engineered Devices," presented at the Eleventh National Conference on Artificial Intelligence, Washington, D.C., 1993.

[9] M. Sasajima, Y. Kitamura, M. Ikeda, and R. Mizoguchi, "FBRL:A Function and Behavior Representation Language," presented at the International Joint Conferences on Artificial Intelligence, Montreal, Quebec, Canada, 1995.

[10] P. Garbacz, "Towards a standard taxonomy of artifact functions," *Applied Ontology*, vol. 1, pp. 221–236, 2005-2006.

[11] Y. Kitamura and R. Mizoguchi, "Ontology-based description of functional design knowledge and its use in a functional way server," *Expert Systems with Applications*, vol. 24, pp. 153-166, 2003.

[12] R. H. Bracewell and J. E. E. Sharpe, "Functional descriptions used in computer support for qualitative scheme generation—“Schemebuilder”," *Artificial Intelligence for Engineering Design, Analysis & Manufacturing*, vol. 10, pp. 333-345, 1996.

[13] Y. Umeda, M. Ishii, M. Yoshioka, Y. Shimomura, and T. Tomiyama, "Supporting Conceptual Design Based on the Function-Behavior-State Modeler," *Artificial Intelligence for Engineering Design, Analysis & Manufacturing*, vol. 10, pp. 275-288, 1996.

[14] J. S. Gero and U. Kannengiesser, "The situated function-behaviour-structure framework," in *Artificial Intelligence in Design*, J. S. Gero, Ed., ed Norwell, MA, USA: Kluwer Academic Publishers, 2002, pp. 89-104.

[15] B. Chandrasekaran and J. R. Josephson, "Representing Function as Effect," presented at the Fifth International Workshop on Advances in Functional Modeling of Complex Technical Systems, Paris, France, 1997.

[16] D. G. Bobrow, "Qualitative reasoning about physical systems: an introduction," *Artificial Intelligence*, vol. 24, pp. 1-5, 1984.

[17] J. S. Gero, "Design prototypes: a knowledge representation schema for design," *AI Magazine*, vol. 11, pp. 26-36, 1990.

[18] J. S. Gero, "Creativity, emergence and evolution in design," *Knowledge-Based Systems*, vol. 9, pp. 435-448, 1996.

[19] S. R. Bhatta and A. K. Goel, "A Functional Theory of Design Patterns," presented at the 15th international joint conference on Artifical intelligence - Volume 1, Nagoya, Japan, 1997.

[20] S. Bhatta, A. Goel, and S. Prabhakar, "Innovation in Analogical Design: A Model-Based Approach," presented at the Artificial Intelligence in Design, Dordrecht, The Netherlands, 1994.

[21] B. Chandrasekaran, "Representing function: relating functional representation and functional modeling research streams," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 19, pp. 65-74, 2005.

[22] V. Sembugamoorthy and B. Chandrasekaran, "Functional Representation of Devices and Compilation of Diagnostic Problem-Solving Systems," in *Experience, Memory, and Reasoning*, J. Kolodner and C. K. Riesbeck, Eds., ed Hillsdale, NJ: Lawrence Erlbaum Associates, 1986, pp. 47-53.
[23] Y. Iwasaki, R. Fikes, M. Vescovi, and B. Chandrasekaran, "How Things Are Intended to Work: Capturing Functional Knowledge in Device Design," presented at the International Joint Conference on Artificial Intelligence, Menlo Park, CA, 1993.
[24] P. Kroes, "Formalization of Technical Functions: Why is that so Difficult?," presented at the Tools and Methods of Competitive Engineering, TMCE-2010, Ancona, Italy, 2010.
[25] G. Pahl, W. Beitz, J. Feldhusen, and K. H. Grote, *Engineering Design: A Systematic Approach*, 3rd. ed. London: Springer-Verlag London Limited, 2007.
[26] K. N. Otto and K. L. Wood, *Product Design Techniques in Reverse Engineering and New Product Development*. Upper Saddle River, NJ: Prentice Hall, 2001.
[27] D. G. Ullman, *The Mechanical Design Process*. New York: McGraw-Hill, 1992.
[28] M. R. Bohm and R. B. Stone, "Product Design Support: Exploring a Design Repository System," presented at the ASME International Mechanical Engineering Congress, Anaheim, CA, USA, 2004.
[29] M. R. Bohm and R. B. Stone, "Representing Functionality to Support Reuse: Conceptual and Supporting Functions," presented at the ASME 2004 Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Salt Lake City, UT, USA, 2004.
[30] J. A. Collins, B. T. Hagan, and H. M. Bratt, "Failure-Experience Matrix - a Useful Design Tool," *Journal of Engineering for Industry*, vol. B 98, pp. 1074-1079, 1976.
[31] C. F. Kirschman and G. M. Fadel, "Classifying Functions for Mechanical Design," *Journal of mechanical design*, vol. 120, pp. 475-482, 1998.
[32] S. Szykman, J. W. Racz, and R. D. Sriram, "The Representation of Function in Computer-Based Design," presented at the 1999 ASME Design Engineering Technical Conferences, Las Vegas, NV, USA, 1999.
[33] R. B. Stone and K. L. Wood, "Development of a Functional Basis for Design," *Journal of mechanical design*, vol. 122, pp. 359-370, 2000.
[34] J. Hirtz, R. B. Stone, D. A. McAdams, S. Szykman, and K. L. Wood, "A Functional Basis for Engineering Design: Reconciling and Evolving Previous Efforts," *Research in engineering design*, vol. 13, pp. 65-82, 2002.
[35] M. R. Bohm, R. B. Stone, and S. Szykman, "Enhancing virtual product representations for advanced design repository systems," *Journal of Computing and Information Science in Engineering*, vol. 5, pp. 360-72, 2005.
[36] D. A. McAdams and K. Wood, "A Quantitative Similarity Metric for Design-by-Analogy," *Journal of mechanical design*, vol. 124, pp. 173-182, 2002.
[37] I. Y. Tumer and R. B. Stone, "Analytical Methods to Evaluate Failure Potential during High-Risk Component Development," presented at the 2001 ASME Design Engineering Technical Conferences, Pittsburgh, PA, USA, 2001.
[38] T. Kurtoglu, A. Swantner, and M. I. Campbell, "Automating the conceptual design process: From black box to component selection," *Artificial Intelligence for Engineering Design, Analysis & Manufacturing*, vol. 24, pp. 49-62, 2010.
[39] R. L. Nagel, K. L. Perry, R. B. Stone, and D. A. McAdams, "FunctionCAD: A Functional Modeling Application Based on the Function Design Framework," presented at the International Design Engineering Technical Conferences, San Diego, CA, 2009.
[40] R. L. Nagel, M. R. Bohm, R. B. Stone, and D. A. McAdams, "A Representation of Carrier Flows for Functional Design," presented at the International Conference on Engineering Design ICED 07, Paris, France.