Models, Composability, and Validity

Eric Werner Weisel
Old Dominion University
MODELS, COMPOSABILITY, AND VALIDITY

by

Eric Werner Weisel
B.S. May 1988, United States Naval Academy
M.S. May 1995, Florida Institute of Technology

A Dissertation Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirement for the Degree of

DOCTOR OF PHILOSOPHY

ENGINEERING WITH A CONCENTRATION IN
MODELING AND SIMULATION

OLD DOMINION UNIVERSITY
May 2004

Approved by:

Mikel D. Petty (Director)

R. Bowen Loftin (Member)

Roland R. Mielke (Member)

C. Michael Overstreet (Member)

Jack Yellin (Member)
Composability is the capability to select and assemble simulation components in various combinations into simulation systems to satisfy specific user requirements. The defining characteristic of composability is the ability to combine and recombine components into different simulation systems for different purposes. The ability to compose simulation systems from repositories of reusable components has been a highly sought after goal among modeling and simulation developers. The expected benefits of robust, general composability include reduced simulation development cost and time, increased validity and reliability of simulation results, and increased involvement of simulation users in the process. Consequently, composability is an active research area, with both software engineering and theoretical approaches being developed. Composability exists in two forms, syntactic and semantic (also known as engineering and modeling). Syntactic composability is the implementation of components so that they can be connected. Semantic composability answers the question of whether the models implemented in the composition can be meaningfully composed.

This research develops a formal theory for semantic composability of simulation components, drawing upon existing theories, including mathematical logic and computability theory. The theory includes formal definitions of composability and associated concepts, a set of theorems and proofs addressing crucial aspects of semantic composability, and an analysis of what the theoretical results imply for practical composability engineering. Theorems address specific areas of semantic composability research. Validity theorems provide requirements for preserving validity in a composition of valid components. Process complexity theorems address the computational complexity of the composition process.
This work is dedicated to Kathleen, Thomas, Hannah, James, and Alan for an abundance of patience and understanding.
ACKNOWLEDGEMENTS

This work would not have been possible without the help and support of several colleagues. First, I would like to thank Dr. Roland Mielke, Dr. Michael Overstreet, and Dr. Jay Yellen for their guidance in undertaking this research effort. I would particularly like to thank Dr. Jay Yellen for traveling a great distance to provide this support. Additionally, I am grateful to Dr. Bowen Loftin for his guidance and the support of his staff at the Virginia Modeling, Analysis, and Simulation Center. Lastly, I would like to thank my advisor, Dr. Mikel Petty, whose guidance and direction was certainly invaluable in completing this effort.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. COMPOSABILITY CONCEPTS AND LEXICON</td>
<td>1</td>
</tr>
<tr>
<td>1.1. Definitions of composability</td>
<td>1</td>
</tr>
<tr>
<td>1.2. Levels of composability</td>
<td>3</td>
</tr>
<tr>
<td>1.3. Composability as a simulation system requirement</td>
<td>8</td>
</tr>
<tr>
<td>1.4. Types of composability</td>
<td>9</td>
</tr>
<tr>
<td>1.5. Related ideas</td>
<td>10</td>
</tr>
<tr>
<td>2. SURVEY OF COMPOSABILITY RESEARCH</td>
<td>13</td>
</tr>
<tr>
<td>2.1. Syntactic composability</td>
<td>13</td>
</tr>
<tr>
<td>2.2. Semantic composability</td>
<td>28</td>
</tr>
<tr>
<td>2.3. Simulation formalisms</td>
<td>28</td>
</tr>
<tr>
<td>3. DEFINITIONS</td>
<td>36</td>
</tr>
<tr>
<td>3.1. Model</td>
<td>36</td>
</tr>
<tr>
<td>3.2. Simulation</td>
<td>43</td>
</tr>
<tr>
<td>3.3. Validity</td>
<td>47</td>
</tr>
<tr>
<td>3.4. Composition and composability</td>
<td>50</td>
</tr>
<tr>
<td>4. COMPOSITION SUFFICIENCY</td>
<td>55</td>
</tr>
<tr>
<td>4.1. Partially ordered sets</td>
<td>55</td>
</tr>
<tr>
<td>4.2. Vectors and models</td>
<td>58</td>
</tr>
<tr>
<td>4.3. Specifying variable mappings in composite models</td>
<td>59</td>
</tr>
<tr>
<td>4.4. Assembling composite models</td>
<td>62</td>
</tr>
<tr>
<td>4.5. Composition sufficiency theorem</td>
<td>66</td>
</tr>
<tr>
<td>4.6. Consequences of the composition sufficiency theorem</td>
<td>68</td>
</tr>
<tr>
<td>5. VALIDITY OF CLASSES OF MODELS UNDER COMPOSITION</td>
<td>69</td>
</tr>
<tr>
<td>5.1. Classes of models</td>
<td>69</td>
</tr>
<tr>
<td>5.2. Classes of validity relations</td>
<td>70</td>
</tr>
<tr>
<td>5.3. Validity under composition</td>
<td>73</td>
</tr>
<tr>
<td>6. EXAMPLES</td>
<td>89</td>
</tr>
<tr>
<td>6.1. Mod counter example</td>
<td>89</td>
</tr>
<tr>
<td>6.2. First-order differential equation example</td>
<td>95</td>
</tr>
<tr>
<td>7. COMPUTATIONAL COMPLEXITY OF COMPOSITION</td>
<td>98</td>
</tr>
<tr>
<td>7.1. Review of NP-completeness theory</td>
<td>99</td>
</tr>
<tr>
<td>7.2. Component selection problem definitions</td>
<td>101</td>
</tr>
<tr>
<td>7.3. Computational complexity of component selection</td>
<td>107</td>
</tr>
<tr>
<td>7.4. Other computational problems in composability</td>
<td>110</td>
</tr>
<tr>
<td>7.5. Computational complexity summary</td>
<td>112</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8. SUMMARY OF RESULTS AND FUTURE WORK</td>
<td>113</td>
</tr>
<tr>
<td>8.1. Summary of results</td>
<td>113</td>
</tr>
<tr>
<td>8.2. Future work</td>
<td>115</td>
</tr>
<tr>
<td>9. REFERENCES</td>
<td>120</td>
</tr>
<tr>
<td>VITA</td>
<td>132</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Levels of composability</td>
<td>4</td>
</tr>
<tr>
<td>2. Summary of composition validity results</td>
<td>88</td>
</tr>
<tr>
<td>3. Function definition table for MOD2 counter</td>
<td>91</td>
</tr>
<tr>
<td>4. Alternate classes of objective decidability</td>
<td>102</td>
</tr>
<tr>
<td>5. Forms of composition</td>
<td>105</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Notional example of composability</td>
<td>1</td>
</tr>
<tr>
<td>2.</td>
<td>The JMASS system (Meyer, 2003)</td>
<td>16</td>
</tr>
<tr>
<td>3.</td>
<td>The OneSAF Product Line Architecture Framework (Wittman and Harrison, 2001)</td>
<td>19</td>
</tr>
<tr>
<td>4.</td>
<td>JSIMS Execution Overview (Carlisle, Babineau, and Wuerfel, 2003)</td>
<td>22</td>
</tr>
<tr>
<td>5.</td>
<td>CTPS architecture (Petty and Windygga, 1999)</td>
<td>23</td>
</tr>
<tr>
<td>6.</td>
<td>Creating BOM compositions (SISO, 2003)</td>
<td>26</td>
</tr>
<tr>
<td>7.</td>
<td>Two models of height under gravity</td>
<td>37</td>
</tr>
<tr>
<td>8.</td>
<td>The modeling relation with perfect knowledge</td>
<td>38</td>
</tr>
<tr>
<td>9.</td>
<td>The modeling relation without perfect knowledge</td>
<td>39</td>
</tr>
<tr>
<td>10.</td>
<td>Transition graph for Example 3.1</td>
<td>44</td>
</tr>
<tr>
<td>11.</td>
<td>Notional example of simulation execution and associated transition graph</td>
<td>46</td>
</tr>
<tr>
<td>12.</td>
<td>Syntactic composition</td>
<td>51</td>
</tr>
<tr>
<td>13.</td>
<td>Composition of models</td>
<td>51</td>
</tr>
<tr>
<td>14.</td>
<td>Example composite model</td>
<td>65</td>
</tr>
<tr>
<td>15.</td>
<td>Validity of composition of models</td>
<td>74</td>
</tr>
<tr>
<td>16.</td>
<td>MOD2 counter with control</td>
<td>89</td>
</tr>
<tr>
<td>17.</td>
<td>Model of MOD2 counter with control</td>
<td>90</td>
</tr>
<tr>
<td>18.</td>
<td>Labeled transition system for MOD2 counter with control</td>
<td>92</td>
</tr>
<tr>
<td>19.</td>
<td>MOD4 counter with control</td>
<td>93</td>
</tr>
<tr>
<td>20.</td>
<td>Model of MOD4 counter with control</td>
<td>94</td>
</tr>
<tr>
<td>21.</td>
<td>Labeled transition system for MOD4 counter with control</td>
<td>94</td>
</tr>
</tbody>
</table>
Figure | Page
---|---
22. Component selection problems and complexities | 110
23. An example of first-order predicate calculus *(Hein, 2002)* | 117

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
1 COMPOSABILITY CONCEPTS AND LEXICON

Composability is an increasingly important issue in simulation system development. Unfortunately, because modeling and simulation is still a relatively new discipline, terminological discrepancies exist in many of its aspects (Meyer, 1998), and that applies to composability. Different meanings of the term appear in the simulation research literature; they are generally similar in concept but often differ in emphasis or level. The fact that composability means different things in different contexts has been noted previously (Page and Opper, 1998).

![Diagram of composability](image.png)

Figure 1. Notional example of composability.

1.1 Definitions of composability

Composability has been defined in a number of ways. These example definitions of composability from the literature illustrate the variations that can be found:

The ability to rapidly configure, initialize, and test an exercise by logically assembling a simulation from a pool of reusable components (JSIMS Composability Task Force, 1997).

---

1 Citation format for this manuscript is taken from the journal *OPERATIONS RESEARCH*. 

---

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
The ability to create, configure, initialize, test, and validate an exercise by logically assembling a unique simulation execution from a pool of reusable system components in order to meet a specific set of objectives (Harkrider and Lunceford, 1999).

The ability to build new things from existing pieces (Pratt, Ragusa, and von der Lippe, 1999).

The ability to compose models/modules across a variety of application domains, levels of resolution and time scales (Kasputis and Ng, 2000).

Composability means that a highly customized simulation can be created from a pool of reusable elements (Aronson and Wade, 2000).

The following common definition of composability has been proposed: **composability** is the capability to select and assemble simulation components in various combinations into valid simulation systems to satisfy specific user requirements (Petty and Weisel, 2003b).

**Composability** (informal). Composability is the ability to compose, in varying combinations, simulation components into simulation systems to satisfy specific user requirements. More specifically, composability, as generally used, refers to the ability to compose, or select and assemble from a set of available components, a specific simulation application suited to the user’s purpose (Petty and Weisel, 2002).

This definition conveys the intent of composability from a practical point of view, but is insufficiently formal to support a theory of composability. This research will be interested in whether meta-properties of computable functions, i.e., validity, are preserved in composition, which leads to this definition:

**Composability** (semi-formal). A set of valid models (i.e., computable functions) is composable if and only if their composition is valid.

There are two issues with this definition. First, it is only semi-formal, so a more formal definition will be needed. Second, note that under this definition the composability of a given model $M$ is not determinable in isolation; it is only determinable with respect to a specific set of models it may be composed with. Is this fact a problem, in that under this definition, it is impossible to say “Model $M$ is composable” in general? Or is it an insight, in that composability is truly meaningless without reference to the set of models to be composed, and this definition reveals that fact?
Composability is the capability to select and assemble simulation components in various combinations into simulation systems to satisfy specific user requirements (Petty and Weisel, 2003b). The defining characteristic of composability is that different simulation systems can be composed in a variety of ways, each suited to some distinct purpose, and the different possible compositions will be usefully valid. Composability is more than just the ability to assemble simulations from parts; it is the ability to combine and recombine, to configure and reconfigure, sets of components into different simulation systems to meet different needs. Figure 1 suggests the concept. The components to be composed may be drawn from a library or repository of components. That library might include multiple network interfaces, different user interfaces, a range of classes of implemented entity models, a variety of implemented physical models at different levels of fidelity, and so on. Different sets of components from the repository may be composed into different simulation systems. The components may be reused in multiple simulation systems. Indeed, to a certain extent reuse depends on composability (Igarza and Sautereau, 2001).

1.2 Levels of composability

When uses of the term “composability” in the literature are reviewed, it is evident that there is one way in which the meanings often differ; indeed, that difference is hinted at in the quotations given earlier. The uses differ on the question of what is being composed and what is formed by the composition. A number of different answers can be found in the literature; they will be referred to as levels of composability. Nine levels of composability are defined here. These levels have been drawn from various sources, some of which explicitly or implicitly include several of the levels defined here in composability, e.g., (Biddle and Perry, 2000). Composability levels from different sources that were essentially the same have been combined. Those listed here have

---

2 If the compositions aren’t valid, then by definition they aren’t composable.

3 In this list the levels are named after the unit of composition, i.e., the components being composed. Another method of naming the levels is after the result of composition, i.e., what is produced from the components. Other sources have taken both the former approach (Page and Opper, 1998) (JSIMS Composability Task Force, 1997) and the latter approach (Post, 2002).
different meanings and implications, but there may be some overlap in component and scale between them. Table 1 summarizes these composability levels.

<table>
<thead>
<tr>
<th>Components</th>
<th>Composition</th>
<th>Example(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Event</td>
<td>Millennium Challenge (Ceranowicz et al., 2002)</td>
</tr>
<tr>
<td>Parameter</td>
<td>Simulation</td>
<td>Chemical/Biological Dial-A-Sensor (O'Connor et al., 2002)</td>
</tr>
<tr>
<td>Model</td>
<td>Composite model</td>
<td>ModSAF (Ceranowicz, 1994) OneSAF (Henderson and Rodriguez, 2002) (Henderson, 2003)</td>
</tr>
<tr>
<td>Data</td>
<td>Database</td>
<td>Electronic warfare in DIS (Wood and Petty, 1995) SEDRIS (Foley, Mamaghani, and Birkel, 1998)</td>
</tr>
<tr>
<td>Entity</td>
<td>Military unit</td>
<td>ModSAF (Ceranowicz, 1994) WARSIM (National Simulation Center, 2003) OneSAF (Grainger and Henderson, 2003)</td>
</tr>
</tbody>
</table>

*Table 1. Levels of composability.*
1. **Application.** Applications such as simulations, real C4I systems, networks, communications equipment, and auxiliary software components are composed into simulation events, exercises or experiments (Post, 2002). For this to be a level of composability, rather than simply integration, the composition must be done in a way that allows combining and recombining the applications into different systems and events (more on the distinction between composition and integration later). This level of composability has also been called "event-level" (Post, 2002). The Millennium Challenge simulation event may be an example of this level of composability (Ceranowicz et al., 2002).

2. **Federate.** Simulations are composed into network-connected distributed simulation systems that exchange data at run-time. In the terminology of the High Level Architecture (HLA), which is an architecture standard and interoperability protocol for such systems, the simulations are "federates" and the distributed simulation systems are "federations" (Kuhl, Weatherly, and Dahmann, 1999). Here we use those terms with a generic sense analogous to their HLA meanings but denoting distributed simulations in general. At this level of composability, the federates are composed into persistent federations; a federation is persistent if it is reused for a number of different purposes (such as events, exercises, or experiments), though possibly with some changes to the set of federates that have been composed. The composition may be supported by an interoperability protocol designed for that purpose; in addition to HLA, other such protocols include ALSP and DIS.¹ HLA, which is intended to facilitate interoperability among different simulation systems and types and promote reuse of simulation software (Dahmann, Kuhl, and Weatherly, 1998), provides technical capabilities that contribute to composability, but almost entirely at the federate level (Igarza and Sautereau, 2001). The HLA Federation Development and Execution Process provides methodological guidance to federation development that can support composable federates (Lutz et al., 2003). To a large extent, the possible interoperation between federates in a federation is defined by the Federation Object Model (FOM); careful design in FOM development can facilitate federate-level

---

¹ ALSP is Aggregate Level Simulation Protocol and DIS is Distributed Interactive Simulation.
composability (Eiserman, 2003). Examples of persistent federations composed at this level of composability include the Joint Training Confederation (Fischer, 1996) (Tufarolo and Page, 1996) and the Combat Trauma Patient Simulation (Petty and Windyga, 1999). This level of composability has also been called “federation-level” (Post, 2002).

3. **Package.** Pre-assembled packages comprising sets of models that form a consistent subset of the battlespace are composed into simulations (Page and Opper, 1998) (JSIMS Composability Task Force, 1997). This has been achieved for some simulation application domains using common component software frameworks and sets of components designed and developed from the outset to be composable; examples include ESCADRE, which provides composable components (Igarza and Sautereau, 2001), and JMASS, which provides a framework for developing and integrating components (Meyer, 2003).

4. **Parameter.** Parameters are used to configure pre-existing simulations (Page and Opper, 1998) (JSIMS Composability Task Force, 1997). For example, one existing parameter-level composable sensor simulation allows users to provide values for parameters that specify the performance characteristics for existing or proposed chemical/biological sensor systems (O’Connor et al., 2002). The literature also includes in this level of composability, which is sometimes also called “simulation” level, the idea that a limited pool of packages may be composed into simulations. In the lexicon of this paper, that second concept is included in the package level, rather than the parameter level.

5. **Module.** Software modules⁵ are composed into software executables. The executables may be federates in a federation or standalone simulation systems. The OneSAF family of software products has been designed to have this level of

---

⁵ The term “software component” is also used with this meaning, and if used here, would make this the “component” level of composability. However, in this paper the term “component” is used in a general sense as the units of composition at any level.

6. **Model.** Separate models of smaller-scale processes or objects\(^6\) are composed into composite models of larger-scale processes or objects. For example, models of platform/entity sub-systems, such as sensors and weapons, may be composed into composite models of platforms/entities, such as aircraft (Post, 2002). Models of physical processes, such as wind and rainfall, may be composed into composite models of larger-scale physical phenomena, such as weather. The composite models may be implemented as modules or federates. This level of composability is a design goal of both ModSAF (Ceranowicz, 1994) and OneSAF (Henderson and Rodriguez, 2002) (Henderson, 2003). This level of composability has also been called “object-level” (Post, 2002), “component” (JSIMS Composability Task Force, 1997), and “reconfigurable models” (Diaz-Calderon, Paredis, and Khosla, 2000).

7. **Data.** Sets of data are composed into databases. The data sets may be initially distinct because they describe different entities, they are from different sources, or they represent different aspects of some phenomena. Different data sets were composed to represent electronic warfare in DIS (Wood and Petty, 1995). SEDRIS is intended to support such composability for natural environment databases (Foley, Mamaghani, and Birkel, 1998).

8. **Entity.** Platforms/entities are composed into groupings such as military units, force structures, and scenario orders of battle (Post, 2002). This level of composition may be hierarchical, with several layers of groupings composed into higher level groupings. This level of composition is typically done with data, rather than with software, as in ModSAF (Ceranowicz, 1994), WARSIM (National Simulation Center,

---

\(^6\) Here “object” means simulated real-world object, not software object. The former is not always implemented as the latter and assuming so is an oversimplification. Even if a real-world object class, such as an aircraft type, is implemented as a software object class, it is not correct to assume that the sub-components of that real-world object class, such as sensors and weapons, are implemented as sub-classes of the software object class. That is a mixing of is-a and part-of relationships.
2003), and OneSAF (Grainger and Henderson. 2003). This level of composition has also been called “federate-level” (Post, 2002).

9. Behavior. Low-level atomic behaviors are composed into high-level composite behaviors, which are to be executed by autonomous simulation entities in a computer generated forces system or constructive simulation. The behaviors may be expressed in a variety of forms. Examples include hierarchically organized finite state machines as used in ModSAF and its variants (Calder et al., 1993), process flow diagrams (Peters, LaVine, and Napravnik, 2002), and hierarchically organized flowcharts (Grainger and Henderson. 2003).

1.3 Composability as a simulation system requirement

The potential benefits of composability are well known, and do not need extended explanation here. Some of the important benefits that have been asserted in the literature are listed here.

1. Development of user- or use-specific simulation systems without the need to redevelop common components.

2. Reduction of simulation development time.

3. Reduction of simulation development cost.

4. Increased model and results credibility due to the reuse of previously validated models.

The perceived importance of composability is clear from the fact that it has been established as an official requirement for new simulation system development. The best example of this is OneSAF; composability is mentioned numerous times in the OneSAF Operational Requirements Document (U. S. Army, 1998). It appears twice in the shortcomings list, in both the composability item which states that “current SAFs are not composable for specific applications” and in the fidelity in physical models item which requires the “...ability to compose an exercise or study with increased or decreased resolution without modification to the code.” The ORD requires that OneSAF will “...
provide a framework and supporting technology that permits OneSAF components to be
selected, configured, and integrated into a common simulation environment capable of
being tailored to meet requirements of every M&S domain.” Interestingly, the first and
last of these requirements seem to be at the module level, while the second seems to be at
the model level. In response to these requirements, designers of the OneSAF family of
software products have given substantial attention to composability (U. S. Army, 1998)
(Courtemanche and Burch, 2000) (Courtemanche and Wittman, 2002) (Franceschini,

1.4 Types of composability

Composability can be understood from both engineering and modeling perspectives. The
common definition given earlier emphasizes engineering composability. Engineering and
modeling composability have also been called syntactic and semantic composability,
respectively (Pratt, Ragusa, and von der Lippe, 1999) (Ceranowicz, 2002). Engineering
composability, i.e., the actual implementation of composability, requires that the
composable components be constructed so that their implementation details, such as
parameter passing mechanisms, external data accesses, and timing assumptions are
compatible for all of the different configurations that might be composed. The question
in engineering (syntactic) composability is whether the components can be combined.

In contrast, modeling (semantic) composability is a question of whether the models that
make up the composed simulation system can be meaningfully composed, i.e., if their
combined computation is semantically valid. Even if the components can be composed
syntactically, the models may or may not be composable semantically. The term model
is commonly defined as a mathematical or logical representation of some system or
object. Models are often implemented as computer code and executed over time; that
execution is simulation. Models use equations and algorithms that mimic the pertinent

---

7 In this paper we will use the terms engineering/syntactic and modeling/semantic interchangeably,
as best suits the context. In related work we generally use syntactic and semantic composability (Petty and
Weisel, 2003)

8 Variations of this difficulty have been widely recognized, even in contexts not explicitly
cconcerned with composability, e.g., (Dahmann et al., 1999) (Igarza and Sautereau, 2001).
aspects of the system or object. Non-trivial simulations may have many models, organized hierarchically (models invoking sub-models) and collaboratively (models exchanging data with co-models) in intricate ways. When composing models, it is necessary to determine if the inputs each model will receive, which are often outputs of the models it is composed with, will be within its domain of validity. For example, suppose a composite model of an aircraft is composed from two valid component models, one of the aircraft’s jet engine and one of its flight dynamics. Even if both component models are valid, the composition will not be valid if the engine model produces supersonic velocities for the aircraft and the flight dynamics model is only valid for subsonic velocities. Similarly, it is also necessary to determine if the assumptions made in each model of a composition are consistent. A model of aircraft detection composed of components that in some cases assume infra-red signature is independent of aspect angle and in other cases consider aspect angle when calculating infra-red signature will probably not be valid.

1.5 Related ideas

Certain other ideas are closely related to composability; three, interoperability, integration, and configurability, are defined and distinguished from composability.

For simulations, interoperability is the ability of different simulations, connected in a distributed simulation system, to meaningfully collaborate to simulate a common scenario or virtual world. Their collaboration is normally based on the run-time exchange of simulation data or services, typically using an interoperability protocol such as DIS, ALSP, or HLA. Like composability, two types of simulation interoperability can be identified: (1) technical interoperability (ability to physically connect, compatible use of the interoperability protocol, ability to exchange data) and (2) substantive interoperability (exchange of information that is mutually consistent with the interoperating simulations’ model semantics) (Dahmann et al., 1999). Elements contributing to technical interoperability include hardware compatibility, standards compatibility, time management coordination, coordinated use of infrastructure services, and compatible handling of security issues (Dahmann et al., 1999). Substantive interoperability can depend on consistent levels of representation, entity attribution, entity
behaviors, temporal resolution, and spatial resolution (Dahmann et al., 1999), or methods to resolve inconsistencies in those areas, such as those used in multi-resolution simulation (Franceschini, Schricker, and Petty, 1999). Compliance with the HLA protocol primarily establishes technical, not substantive, interoperability. Substantive interoperability has also been called meaningful interoperability (Clark et al., 2001). Research has addressed both types of interoperability (Franceschini et al., 2000).

It can be seen that these two types of interoperability are closely analogous to the definitions given earlier for engineering and modeling (syntactic and semantic) composability. Also, as previously noted, interoperability protocols such as HLA can support composability at the federate level. How do interoperability and composability differ? We identify three differences. First, there is a difference in time; interoperability is the ability to exchange data or services at run-time, whereas composability is the ability to assemble components prior to run-time (Biddle and Perry, 2000). Second, interoperability is narrower in scope than composability; interoperability as usually meant applies to federates, not to the other levels defined earlier for composability, such as models or data. Third, and most important, there is a difference in flexibility and power between interoperability and composability. Interoperability is necessary but not sufficient to provide composability. Composability (engineering and modeling) at the federate level requires interoperability (technical and substantive) because federates that are not interoperable cannot be composed, i.e., interoperability is necessary for composability. However, interoperability is not sufficient to provide composability, i.e., federates may be interoperable but not composable. Recall that an essential aspect of composability is the ability to not just combine components but to combine and recombine them into different simulation systems. Federates that are interoperable in one

9 Instances can be found where interoperability is considered at levels other than federate; e.g., one survey of interoperability research implicitly identified levels of interoperability analogous to five of the levels of composability defined earlier (Franceschini et al., 2000). Such instances are the exception, however.

10 Also, federates that are composable are necessarily interoperable.
specific federation or with one specific object model, and cannot be combined and recombined in other ways, are not composable.

Non-persistent federations sometimes provide examples of interoperability without compositability. A non-persistent federation is one that exists for the purpose of supporting a specific event, exercise, or experiment and is not intended to persist beyond that purpose. An example of this type of federation is the Platform Proto-Federation (Harkrider and Petty, 1997). The Platform Proto-Federation was a successful federation; it was the first virtual real-time HLA federation (Harkrider and Petty, 1996). However, the federates were closely bound to each other and could not be immediately reused in other federations without substantial effort. The federates of non-persistent federations are necessarily interoperable, in that they interoperate during execution, but they are composable if and only if the set of federates in the federation can be changed without requiring substantial additional integration effort. The matter of substantial effort is crucial to the third distinction between interoperability and compositability.

Integration is the process of configuring and modifying a set of components to make them interoperable and possibly composable. Essentially, any federate can be integrated into any federation with enough effort, but compositability implies that the changes can be made with little effort. The ability to readily combine and recombine is what distinguishes composable simulations from integrated or interoperable simulations.

Configurability is the ability to include varying numbers of identical federates in a federation, e.g., “generating an exercise with eight rather than four M1A2 tank [simulators]” (Harkrider and Lunceford, 1999) Though related, this capability is not the same as the ability to combine and recombine simulation components into different simulation systems that defines compositability.

Indeed, in the context of accreditation it could be argued that all federations are non-persistent because the accreditation of a federation is by definition for a specific purpose only, and the use of the federation for another purpose requires a new accreditation. However, in this paper “persistent” and “non-persistent” refer to development and integration, not accreditation.
2 SURVEY OF COMPOSABILITY RESEARCH

To date, most composability research and development has been aimed at developing concepts, technologies, tools, protocols, standards, control mechanisms, interfaces, and processes to enable the rapid, efficient, and flexible assembly of simulation systems from components in a practical setting. The overall problem of developing components so that they can be assembled and interoperate, which we refer to as syntactic composability, requires that the components be constructed so that their implementation details are compatible for the different configurations that might be composed. Both the methods employed and the objectives pursued for different syntactic composability research projects have varied, a fact that reflects both the challenge and the relative newness of composability. There has been less semantic composability research, even though the need for such research has been recognized (Davis et al., 2000) (Kasputis and Ng, 2000).

2.1 Syntactic composability

In this section, we survey current work on syntactic composability. Most composability research and development to date has been aimed at developing concepts, technologies, tools, protocols, standards, control mechanisms, interfaces, and processes to enable the rapid, efficient, and flexible assembly of simulation systems from components in a practical setting. The overall problem of developing components so that they can be assembled and interoperate, which we refer to as engineering composability, requires that the components be constructed so that their implementation details are compatible for the different configurations that might be composed. The methods that have been proposed, developed, and tested and the projects in which they have been used are presented. The methods reviewed are the “common library” approach used for JMASS; the “product line” approach, used for OneSAF; the “interoperability protocol” approach used for JSIMS; the “object model” approach, used by the Base Object Models Product Development Group; and the “formal” approach, used for DEVS based systems. In addition to explaining the methods and projects, this section compares the methods to each other in terms of assumptions and capabilities and evaluates their strengths with respect to their objectives.
<table>
<thead>
<tr>
<th>Approach</th>
<th>Level</th>
<th>Example(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common library</td>
<td>package</td>
<td>JMASS (Handley, Shea, and Morano, 2000), (Meyer, 2003)</td>
</tr>
<tr>
<td></td>
<td>module model</td>
<td></td>
</tr>
<tr>
<td>Product line</td>
<td>module model</td>
<td>OneSAF (Wittman and Harrison, 2001), (U.S. Army, 1998), (Courtemanche and</td>
</tr>
<tr>
<td></td>
<td>entity</td>
<td>Burch 2000), (Courtemanche and Wittman, 2002), (Franceschini, Hawkes, and</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Graffuis, 2003), (Henderson and Rodriguez, 2002), (Henderson, 2003),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Grainger and Henderson, 2003)]</td>
</tr>
<tr>
<td>Interoperability</td>
<td>federate</td>
<td>JSIMS (Carlisle, Babineau, and Wuerfel, 2003)</td>
</tr>
<tr>
<td>protocol</td>
<td></td>
<td>CTPS (Petty and Windyga, 1999)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CATT (Marshall, 1999)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CCTT ([Marshall, 1999])</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Joint Training Confederation (Fischer, 1996)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Tufarolo and Page, 1996)</td>
</tr>
<tr>
<td>Formal</td>
<td>model</td>
<td>DEVS-based systems (Zeigler, Praehofer, and Kim, 2000)</td>
</tr>
</tbody>
</table>

Table 2. Summary of engineering approaches to composability

The five approaches to composability surveyed in this paper are listed in Table 2, each with the associated levels of composability and examples of systems and architectures which employ this approach.

2.1.1 Common library approach

The common library approach depends upon an organizing framework for a library of reusable software modules. The software modules are reusable only after modification or
the development of a suitable interface. The library may include components in varying levels of composability. None of the components is a stand-alone simulation. The organizing framework may include tools, services, standards, and interfaces. This library is developed using a common set of assumptions and a common data transfer protocol. The components are written by a team to work together in various combinations. Documentation is required to enable component reuse. Libraries of compliant models may exist as part of the system or may be developed and maintained separately. This approach often utilizes package, module, or model levels of composability. Composition of model, module, or package level components results in composites that are composite models, executables, or simulations.

A number of simulation systems use the common library approach. JMASS is summarized here from (Handley, Shea, and Morano, 2000) and (Meyer, 2003).

The Joint Modeling and Simulation System (JMASS) was developed in the early 1990’s in an effort to bring object-based technologies to bear to reduce development and ownership costs of simulation. JMASS was designed with an open architecture and a well-defined application program interface (API) to enable easy interface with commercial products. JMASS was initially developed to provide high-fidelity analyses for weapon system development and acquisition and specifically intended to reduce simulation development time and difficulty. "JMASS provides tools that permit model developers to concentrate on algorithms instead of software." (Handley, Shea, and Morano, 2000)

JMASS is not a simulation but an architecture and associated toolkit. It is a simulation support environment which consists of:

1. **Architecture.** The architecture includes a simulation engine and services.

2. **Interface standards.** Standards are provided for interface between JMASS architecture and the models in a JMASS simulation.

3. **Tools.** Tools which provide a variety of simulation development and analysis support functions.
A JMASS-based simulation is constructed from JMASS-compliant models within the JMASS architecture. JMASS-based models normally represent real or imagined objects like aircraft, radars, and weapons systems. JMASS compliance, however, does not prove composability.

JMASS provides standardized file formats and APIs that enable model development such that composability is achieved. The JMASS architecture ensures appropriately-formatted data exchange through the “port” mechanism but validity of the exchanged data is not
provided. The JMASS interface standard is enforced by subjecting software in a JMASS-based simulation to a code generation process. JMASS is not "plug and play."

Standard service modules provide scheduling, spatial, data recording, and message logging services.

A sampling of tools include an engineering analysis study manager, Flexible Automated Study Tool (FAST), a visualization tool for spatial behavior, SimView, and a post-processing plotting tool called JPlot.

JMASS has supported simulation development in the acquisition, test and evaluation, and scientific and technical intelligence domains.

Strengths of common library approach that enable these systems to achieve their objectives are:

1. *Open architecture.* JMASS was designed with an open architecture and a well-defined application program interface (API) to enable easy interface with commercial products.

2. *Simulation development system.* The common library approach calls for a simulation development system to ensure interface standard compliance.

3. *Tools, services, standards, and interfaces.* Standardized file formats and APIs enable model development such that composability is achieved. Services and tools exist to allow developers to develop, configure, execute, and analyze compliant models and simulations.

2.1.2 Product line approach

The product line approach provides a contained simulation development system utilizing layers of products for development of specific simulation systems. The simulation development system provides products to allow modification and reuse of components. The simulation development system may include components in varying levels of composability. None of the components is a stand-alone simulation. The simulation development system ensures the appropriate data transfer protocol. The components may
be written by different teams and still work together in various combinations. Documentation, or meta-data, is required to enable component reuse. This approach often utilizes behavior, entity, model, or module levels of composability. Composition of behavior, entity, model, or module level components results in composites that are composite behaviors, military units, composite models, or executables.

A number of simulation systems use the product line approach. The One Semi-automated Forces (OneSAF) Objective System is summarized here from (Wittman and Harrison, 2001). Additional information concerning the One Semi-automated Forces (OneSAF) Objective System is available in (U. S. Army, 1998), (Courtemanche and Burch, 2000), (Courtemanche and Wittman, 2002), (Franceschini, Hawkes, and Graffuis, 2003), (Henderson and Rodriguez, 2002), (Henderson, 2003), (Grainger and Henderson, 2003).

The One Semi-automated Forces (OneSAF) Objective System is a composable, entity-based simulation system. The concept began in early 1996 as an effort to develop a single general-purpose entity-level simulation that would reduce duplication of the U. S. Army’s M&S efforts, provide improved interoperability and reuse, and meet the simulation needs of the Army in the future. Intended uses of OneSAF included the development of advanced concepts for doctrine, tactics, unit commander and staff training across various levels of command, weapon systems development, test, and evaluation, and production of data as input to other simulations. The OneSAF Product Line Architecture Framework (PLAF) is designed to support various user domains with multiple end uses. Variability derives from:

1. **Infrastructure.** The simulation may run on a single processor or may be running as a distributed simulation on multiple processors.

2. **Human interaction.** A range of human interaction is supported from human-in-the-loop, used for training, to simulation with no human interaction, used for analysis.

3. **Application.** Intended use of the specific application.
The OneSAF Product Line Architecture Framework (Wittman and Harrison, 2001)

The PLAF is used to define components, including their services and interface, which allows independent development followed by combination into a number of products and configurations. The PLAF uses a hierarchical composition process. This process builds specific system configurations for uses. Products, like Simulation Generator, are made up of one or more components. Each product is a complete package with respect to functionality. Components can be developed independently so they must have complete service and interface definitions and formal documentation.

Figure 3 shows application, products, and components using layers. The uppermost layer depicts user configuration. Next, products required to compose a complete system configuration are shown. Components, necessary for the support of each product, are
listed under the product. The OneSAF Objective System has various standard components available for each product. Products include:

1. *Military scenario planner product.* The military scenario planner product is used to define a scenario specification for a simulation event.

2. *Model and simulation composer product.* The model and simulation composer product is used to develop composite entities, behaviors, or environmental elements from primitive components.

3. *Simulation generator product.* The OneSAF simulation generator product allows development of the scenario requirements at execution.

4. *Technical manager product.* The OneSAF technical manager product provides mechanisms and services to support exercise configuration and setup.

5. *Simulation core product.* The simulation core product provides simulation services including time management services, random number generation services, and probability distribution services. Additionally, the simulation core product provides modeling capabilities including modeling of units, entities, behaviors, physical models, and the environment.

6. *Simulation controller product.* The simulation controller product provides mechanisms, displays, and devices for interacting with the simulation at runtime.


8. *After-action review product.* The after-action review product allows analysis of data collected during OneSAF execution.

9. *Maintenance environment product.* The maintenance environment product provides an integrated environment for software support services and utilities from requirements development to validation and verification.
The One Semi-automated Forces (OneSAF) Objective System has supported simulation development in the Advanced Concepts and Requirements (ACR), Training, Exercises, and Military Operations (TEMO), and Research, Development, and Acquisition (RDA) domains.

Strengths of product line approach that enable these systems to achieve their objectives are:

1. *Simulation development system.* The product line approach provides a contained simulation development system utilizing layers of products for development of specific simulation systems.

2. *Simulation development products.* A variety of simulation development products enable model development such that composability is achieved. Services and tools exist to allow developers to develop, configure, execute, and analyze models and simulations.

2.1.3 Interoperability protocol approach

The interoperability protocol approach is based on the run-time exchange of simulation data or services, typically using an interoperability protocol such as DIS, ALSP, or HLA. Simulations are composed into network-connected distributed simulation systems that exchange data at run-time. In the terminology of the High Level Architecture (HLA), which is an architecture standard and interoperability protocol for such systems, the simulations are "federates" and the distributed simulation systems are "federations". Components are simulations that can run independently except for sending and receiving data. This approach often utilizes the federate level of composability. Federates are composed into persistent federations; a federation is persistent if it is reused for a number of different purposes (such as events, exercises, or experiments), though possibly with some changes to the set of federates that have been composed.

A number of simulation systems use the interoperability protocol approach. The Joint Simulation System (JSIMS) (Carlisle, Babineau, and Wuerfel, 2003), Combat Trauma Patient Simulation (CTPS) (Petty and Windyga, 1999), Combined Arms Tactical Trainer
(CATT) (Marshall, 1999) and Close Combat Tactical Trainer (CCTT) (Marshall, 1999) are described here. Examples of other persistent federations composed using this approach the Joint Training Confederation (Fischer, 1996) (Tufarolo and Page, 1996).

The Joint Simulation System (JSIMS) Program, summarized here from (Carlisle, Babineau, and Wuerfel, 2003), is a federation of over 30 Service, Agency, and Joint models. The system is designed to provide joint training spanning various phases of military operations. A common simulation architecture is provided by the High Level Architecture (HLA) allowing interoperability and reuse of components. Models representing land, air, maritime, and other military and civilian elements interoperate with each other during JSIMS Exercise Execution. Variation in the type and number of federates is allowed by the JSIMS architecture in a given exercise execution of JSIMS. Additionally, the HLA Specification defines a run-time infrastructure (RTI) which interfaces to each of the JSIMS runtime components. Figure 4 is a notional representation of JSIMS execution.

**Figure 4. JSIMS Execution Overview** (Carlisle, Babineau, and Wuerfel, 2003)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
The Combat Trauma Patient Simulator (CTPS), summarized here from (Petty and Windyga, 1999), is a simulation system for military medical training and analysis. Figure 5 shows the CTPS architecture. Capabilities include simulating the type and severity of casualties, tracking the movement of casualties from the battlefield to the military medical field hospital, logging diagnosis and care given to the patient, and outcome generation. This represents a complete combat casualty cycle for training and analysis. Using off-the-shelf commercial and military technologies, the system provides a realistic assessment of the effect of military combat casualties and provides a dynamic training tool for military medical personnel.

The intentions for system development were to realistically represent battlefield casualties; provide improved training for military medical personnel resulting in better care and higher survival rates for combat wounded; and provide a means for military medical test and evaluation and analysis of issues in combat casualty treatment. The High Level Architecture (HLA) provides an interoperability protocol for the purpose of communication between components of the CTPS system. Since off-the-shelf
commercial and military technologies were used in the development of the system, modifying all of the components to use the HLA was impossible. In many cases, interfaces were constructed to perform that purpose. Each components, then, is able to connect to the run-time infrastructure (RTI). The RTI interfaces convert non-HLA data from each component into the CTPS FOM data format for use by other components via the RTI.

Patients are the main entities in the CTPS system. Different CTPS components are used to simulate various aspects of the patient's condition and care as the patient moves from the battlefield to the military medical field hospital. The HLA coordinates transfer of patient data, using the HLA ownership management services, during simulation execution from one CTPS component to the next.

Interoperability among the CTPS components was necessary for simulation operation. However, none was readily modifiable to provide the desired simulation effect. Composing a number of existing simulations was the best and most efficient method to achieve the desired simulation and meet the goals of the project. Note that each of the components addressed here is a stand-alone simulation. The HLA provides the data transfer protocol only. Semantic composability was achieved manually.

Combined Arms Tactical Trainer (CATT), summarized here from (Marshall, 1999) is a composition of manned simulators and workstations that provide simulation training up to the Task Force command level. The system also includes an after-action review capability. Components provide a variety of simulation effects including natural environmental effects, like weather; and representation of adjacent, supporting, and opposing forces and support elements using computer-generated forced (CGF). Composition is supported by a Distributed Interactive Simulation (DIS) interoperability protocol.

Close Combat Tactical Trainer (CCTT), summarized here from (Marshall, 1999), is a composition of interactive modules and workstations that provide training from the platoon through battalion task force level using simulated armor, cavalry, and mechanized infantry elements. Military units fight on the virtual battlefield in interactive
modules much like the individual crew stations for a variety of military vehicles and weapon systems. A dismounted infantry module (DIM) provides training for infantry decision-making in battlefield leadership situations for computer-generated dismounted infantry forces. Semi-automated forces (SAF) provide a variety of friendly and opposing force entities. Composition is supported by a Distributed Interactive Simulation (DIS) interoperability protocol.

Strengths of interoperability protocol approach that enable these systems to achieve their objectives are:

1. **Open architecture.** A standard interoperability protocol by design provides an open architecture.

2. **Standard interoperability protocol.** Composition is supported by an interoperability protocol designed for that purpose; in addition to HLA, other such protocols include ALSP and DIS.

2.1.4 Object model approach

The object model approach depends upon a standard for model specification. Models are reusable only after modification or the development of a suitable interface. None of the components is a stand-alone simulation. The organizing framework may include tools, services, standards, and interfaces. Documentation is required to enable component reuse. The model specification is designed to work with interoperability protocols such as HLA. This approach utilizes the model level of composability. Composition of model level components results in composites that are composite models.

The Base Object Model (BOM), summarized here from (SISO, 2003), is a “reusable package of information representing an independent pattern of simulation interplay”. Base Object Models (BOM) are meant to improve “interoperability, reuse, and composability, by providing ‘patterns’ and ‘components’ of simulation interplay that can be used as building blocks in the assembly of simulations and enterprises of simulations.” (SISO, 2003) Two types of BOMs are described: Interface (IF) BOMs have messages and triggers related to one or more class of objects and provide a reusable “pattern of
interplay”. An Encapsulated (ECAP) BOM includes additional information like behaviors for modeling.

A simulation or simulation environment is constructed by the composition of individual BOMs resulting in a “Mega-BOM”. Figure 6 is a representation of this type of composition.

![BOM Palette](image)

Choose what fits conceptual model?

compositions

*Figure 6. Creating BOM compositions (SISO, 2003)*

Meta-data in a Mega-BOM includes meta-data associated with individual BOMs as well the relationship between BOMs in the Mega-BOM. Mega-BOMs can be converted to a FOM within the High Level Architecture (HLA) domain to support interoperability through the HLA interoperability protocol. This capability provides improved interoperability throughout the modeling and simulation domain.
Strengths of object model approach that enable this system to achieve their objectives are:

1. **Open architecture.** The Base Object Model (BOM) Template Specification (SISO, 2003) provides an open architecture.

2. **Well-defined model specification.** Base Object Models (BOM) are designed with a well-defined model specification allowing engineering composability.

2.1.5 Formal approach

This section identifies the formal approach to composability. The formal approach depends upon a simulation formalism to define composability in a theoretic or mathematical way. This approach often utilizes model level of composability. DEVS (Zeigler, Praehofer, and Kim, 2000) is an example of a simulation formalism which enables engineering composability through the use of coupled models using ports. The formal approach is unique in its attempt to prove in a formal or mathematical way how models can be composed.

2.1.6 Comparison of methods

Five approaches to composability are surveyed here. The methods reviewed are the “common library” approach used for JMASS; the “product line” approach, used for OneSAF; the “interoperability protocol” approach used for JSIMS; the “object model” approach, used by the Base Object Models Product Development Group; and the “formal” approach, used for DEVS-based systems. Each seeks to solve the problem of developing components so that they can be assembled and interoperate. Each achieves engineering composability by the requiring that components adhere to a protocol for services and data transfer. This allows the components to be constructed so that their implementation details are compatible for the different configurations that might be composed. This protocol is achieved either by a well-defined standard or enforced by requiring components be developed only within a contained development system. None of the methods address semantic composability.
2.2 Semantic composability

There has been less semantic composability research, even though the need for such research has been recognized (Davis et al., 2000) (Kasputis and Ng, 2000).

We are discovering that unless models are designed to work together, they don't (at least not easily and cost effectively). Without a robust, theoretically grounded framework for design, we are consigned to repeat this problem for the foreseeable future (Kasputis and Ng, 2000).

One theoretical look at composability found that the process of selecting the set of components to meet given requirements was unexpectedly complex (Page and Opper, 1999). An effort to define a process maturity model for simulation validation includes at the highest maturity level the beginnings of what is intended to be a formal derivation of validation criteria that are provably necessary and sufficient (Harmon and Youngblood, 2003). Although a valid model is not necessarily a composable one, these ideas contribute to the development of formal approaches to modeling and simulation.

2.3 Simulation formalisms

Formalisms establish well-defined specification for the mathematical object we call a model. This formal definition is necessary for analysis of the model. The theory addressed here depends upon a simulation formalism to define composability in a theoretic or mathematical way. Applicable topics in simulation formalism will be discussed. This section's discussion of mathematical logic, model theory, and simulation theory follows standard texts in those fields (Mendelson, 1997), (Hodges, 1993), (Fishwick, 1995), (Law and Kelton, 2000), (Woods and Lawrence, 1997).

2.3.1 Model theory

There is a branch of mathematical logic, called "model theory", which is concerned with relations between sentences in formal languages and interpretations (assignments of values to variables) that make those sentences true (Hodges, 2000). It may be possible to describe the model and the modeled reality axiomatically and establish validity using model theory. A description of a first order model based on a standard text in mathematical logic is included here (Mendelson, 1997).
A formal theory $K$ consists of:

1. **Alphabet** – a set of abstract symbols.

2. **Grammar** – a set of rules specifying the ways in which the symbols of the alphabet may be formed into finite strings of symbols and the ways in which the strings may be formed into statements. Statements that submit to these grammatical rules are called *well-formed*.

3. **Axioms** – a set of well-formed statements accepted as valid without proof.

4. **Rules of inference** – a set of rules specifying the ways in which axioms and other well-formed statements may be changed into new well-formed statements.

A first-order language $L$ contains the following symbols:

1. The *prepositional connectives* $\rightarrow$ and $\Rightarrow$, and the universal quantifier $\forall$.\(^{13}\)

2. *Punctuation marks*: left parenthesis, right parenthesis, and comma.

3. Denumerably many\(^{14}\) individual *variables* $x_1, x_2, x_3, ...$

4. A finite or denumerable, possible empty, set of *function letters*.

5. A finite or denumerable, possible empty, set of individual *constants*.

6. A non-empty set of *predicate letters*.

Let $L$ be a first-order language. A *first-order theory* in the language $L$ is a formal theory $K$ whose symbols and well-formed statements are the symbols and well-formed statements of $L$ and whose axioms and rules of inference are specified by:

\(^{13}\) Note that the existential quantifier $\exists$ can be constructed from the universal quantifier $(\exists x)P$ instead of $\neg((\forall x)(\neg P))$ (Mendelson, 1997, p. 52).

\(^{14}\) A set is denumerable if it has the same cardinal number as the set of positive integers.
Logical axioms: Let $B$, $C$, and $D$ be well-formed statements in the language $L$.

1. $B \Rightarrow (C \Rightarrow B)$

2. $(B \Rightarrow (C \Rightarrow D)) \Rightarrow ((B \Rightarrow C) \Rightarrow (B \Rightarrow D))$

3. $(\neg C \Rightarrow \neg B) \Rightarrow ((\neg C \Rightarrow B) \Rightarrow C)$

4. $(\forall x_i)B(x_i) \Rightarrow B(t)$

5. $(\forall x_i)(B \Rightarrow C) \Rightarrow (B \Rightarrow (\forall x_i)C)$

Proper axioms: Proper axioms vary from theory to theory\textsuperscript{15}.

Rules of inference:

1. Modus ponens: $C$ follows from $B$ and $B \Rightarrow C$.

2. Generalization: $(\forall x_i)B$ follows from $B$.

Let $L$ be a first-order language. An interpretation $M$ of $L$ consists of the following:

1. Domain – a non-empty set $D$.

2. For each predicate letter $A$ of $L$, an assignment of a relation in $D$.

3. For each function letter $f$ of $L$, an assignment of an operation in $D$.

4. For each constant $a$ of $L$, an assignment of a fixed element of $D$.

Let $K$ be a first-order theory in $L$. In the context of model theory, a model of $K$ is an interpretation of $L$ for which all the axioms of $K$ are true.

---

\textsuperscript{15} A first-order theory with no proper axioms is a first-order predicate calculus. Additionally, every theorem of a first-order predicate calculus is logically valid, and any first-order predicate calculus is consistent.
2.3.2 Discrete Event System Specification (DEVS)

DEVS is a formalism developed in (Zeigler, 1976). In the classic DEVS system specification (Zeigler, Praehofer, and Kim, 2000), a discrete event system specification (DEVS) is a structure:

$$M = (X, S, Y, \delta_{int}, \delta_{ext}, \lambda, \tau)$$

where

- $X$ is the set of input values,
- $S$ is a set of states,
- $Y$ is the set of output values,
- $\delta_{int} : S \rightarrow S$ is the internal transition function,
- $\delta_{ext} : Q \times X \rightarrow S$ is the external transition function

where $Q = \{(s, e) \text{ such that } s \in S, 0 \leq e \leq \tau(s)\}$,

- $\lambda : S \rightarrow Y$ is the output function, and
- $\tau : S \rightarrow R^+$ is the time advance function, where $R^+$ is the set of non-negative real numbers.

The internal and external transition functions, $\delta_{int}$ and $\delta_{ext}$, are sometimes replaced by a single transition function $\delta$.

A number of extensions to DEVS have been proposed. Extensions allow application of DEVS in a new system application. One extension to DEVS allows modeling of dynamic structure systems (Barros, 1998). Other extensions include symbolic DEVS (Zeigler, Praehofer, and Kim, 2000), fuzzy DEVS (Zeigler, Praehofer, and Kim, 2000), real-time DEVS (Zeigler, Praehofer, and Kim, 2000), DEVS-HLA (Kim, Cho, and Kim, 1999), and JDEVS (Filippi, Delhom, and Bernardi, 2002).
2.3.3 The Condition Specification

The Condition Specification is a formalism presented in (Overstreet and Nance, 1985). The Condition Specification (CS) of a model is a collection of three components. The interface specification identifies input and output specification. Model dynamics are specified by object and transition specifications. Each element in the transition specification is a condition action pair (CAP). Finally, a report specification specifies the output. The complete specification of the Condition Specification formalism is found in (Overstreet and Nance, 1985) and (Nance and Overstreet, 1987, 1988).

2.3.4 Structured modeling

Structured modeling is a formalism developed in (Geoffrion, 1987, 1989a). "Structured modeling aims to provide a formal mathematical framework and computer-based environment for conceiving, presenting, and manipulating a wide variety of models." (Geoffrion, 1987) Geoffrion is motivated by an observation that the "discipline of modeling has advanced only slowly compared to the disciplines concerned with analyzing and solving models." (Geoffrion, 1987) Geoffrion details in (Geoffrion, 1987) a number of problems and opportunities facing the field of operations research. Problems include low productivity (multiple model representations required, interface with solver problematic, stove-piped software) and poor managerial acceptance (modeling is not well understood by managers and leaves them with a feeling of loss of control). Opportunities include advances in desktop computing, which have continued well past the publication date of this paper, developments in the field of modeling, advances in database management, and advances in spreadsheet modeling. This list is equally appropriate for simulation modeling.

Geoffrion's answer to these problems and opportunities is a "new generation of modeling systems." (Geoffrion, 1987) Such a system will have a number of desirable features including:

1. a rigorous and coherent conceptual framework for modeling based on a single model representation format suitable for managerial communication, mathematical use, and direct computer execution
2. independence of model representation and model solution, with model interface standards to facilitate building a library of models and of easily accessed solvers for retrieval, systems of simultaneous equations, optimization, and other important manipulations

3. sufficient generality to encompass most of the great modeling paradigms that MS/OR and kindred model-based fields have developed for organizing the complexity of reality (activity analysis, decision trees, flow networks, graphs, Markov chains, queuing systems, etc)

4. usefulness for most phases of the entire life-cycle associated with model-base work

5. representational independence of general model structure and detailed data needed to describe specific model instances

6. desktop implementation with a modern user interface (e.g. visually interactive, directly manipulative, syntactically humane, and with liberal use of graphics and tables)

7. integrated facilities for data management and ad hoc query in the tradition of database systems

8. immediate expression evaluation in the tradition of desktop spreadsheet software (Geoffrion, 1987)

Geoffrion describes a model as giving “sharp definition to ‘knowledge’ about some part of ‘reality’.” (Geoffrion, 1989a) Structured modeling calls such a definition a model element. There are five element types.

1. A primitive entity element is not defined mathematically.

2. A compound entity element is made up of primitive entity elements.

3. An attribute element is made of primitive or compound entity elements with a value.

4. A function element associates a rule.

5. A test element is like a function element with only a [true, false] range.

Desirable properties (correlation, acyclicity, classification, grouping, and hierarchy) are captured by the use of elemental structure, generic structure, and modular structure. An instance is called a structured model. (Geoffrion, 1989a) develops properties and provides a number of elementary results and gives proofs for each.
(Geoffrion, 1989b, 1991, 1992) describe implementing structured modeling in a computer-based modeling environment. (Geoffrion, 1989b) provides general guidance on the proper qualities for a useful modeling environment. The benefits provided by such an environment include increased productivity, increased quality, and increased use of OR/MS practice. A modeling environment should enable the entire modeling life cycle, communicate with decision-makers, not just analysts, support product and hardware evolution, provide paradigm neutral language, and assist good management practices. Challenges include development of a framework for conceptual modeling, availability of executable modeling languages, and software integration. FW/SM, a prototype structured modeling environment (Geoffrion, 1991), implements a language for structured modeling in a desktop modeling environment. Of significant note is the inclusion of off-the-shelf mathematical solvers for applicable tasks within the environment. The Structured Modeling Language (SML) (Geoffrion, 1992) is a system for examining analytic models. Although a single language, SML is presented in four levels of increasing power. Level 1 is a text-based language for representing any graph. Level 2 allows values for vertices and edges. Level 3 provides for organization of the graph structure. Level 4 allows the definition of complex classes of vertices and edges.

Geoffrion's complaints (Geoffrion, 1987) are still valid today. No single formalism is the clear winner. There is still a need for basic research in the area of modeling formalism. Geoffrion's list of desirable characteristics could provide a basis or part of a list of requirements for a suitable modeling formalism. The issues concerning desktop computer user interfaces are largely solved by advances in desktop computing technology since the publication of these papers. The definitions provided in Geoffrion, 1989a) should be individually scrutinized for retention in a modeling formalism. Definitions should be consistent with similar definitions commonly accepted in classical mathematics and other sciences. Having stated this with a visual modeling language in mind, many well-used high-level mathematical and modeling languages such as MATLAB and LINDO have similar textual code necessities. Structured modeling does not address model design for cost effective VV&A. Structured modeling does not directly support recursively-defined relationships. Structured modeling does not support a common formalism for both discrete event and continuous simulation models.
Geoffrion’s development of a modeling system from formalism to modeling environment is noteworthy in two ways. First the scale of achievement is impressive. Second, and perhaps more significantly, the development of SML and the FW/SM modeling environment provided a potential commercial and academic outlet for the use of the structured modeling formalism. Although structured modeling was conceived to model optimization problems, the axiomatic base of the specification should be considered in constructing a formalism for composability. Without a clear winner for a widely accepted and used formalism, this approach should be considered. Additional research concerning structured modeling should include:

1. Research more recent selections and contributions from additional authors.
   Additional research should focus on the formal aspects of structured modeling vice implementation languages or model development systems beyond relevance to formalism.

2. Establish if the structured modeling formalism is consistent.\(^{16}\)

3. Establish the requirements for a satisfactory modeling formalism. Geoffrion’s desirable features for modeling system are germane.

4. Establish the scope and boundaries of a satisfactory modeling formalism, if any.

5. Define the term complete with respect to modeling formalism. Can this term be used in conjunction with modeling formalism with the meaning “fully developed”?\(^{16}\)

6. Is it possible for a formalism to be appropriate and consistent for both discrete event and continuous simulation models?

7. Is it possible for a formalism to be appropriate and consistent over a robust range of model types such as simulation models and optimization models?

\(^{16}\) A formalism is consistent if there is no statement that can be made within the context of the formalism that is both true and false by the rules of inference associated with the formalism. In other words, within a given formalism, a statement and its negation cannot both be theorems.
3 DEFINITIONS

A formal theory of semantic composability is developed here. In general terms, a formal theory has four parts: objects, the things or ideas which are the subject of the theory; axioms, statements about the objects that are accepted as true without proof; rules of inference, which may be used applied to the axioms and previously proven theorems to produce new theorems about the objects; and a goal or purpose for the theory, often to produce a set of interesting or useful theorems (Trudeau, 1993).

As the basis for such a theory, definitions suitable for formal reasoning are stated for model and simulation, which are the objects of the theory, as well as for composability and validity, which are possible attributes of those objects. By their nature, the formal definitions invoke computability theory and mathematical logic, bringing with them the axioms, theorems, and rules of inference of those theories; pertinent aspects of those theories are identified. The formal definitions are compared to the informal definitions in general use and arguments that the definitions are appropriate for the purpose of developing a theory of semantic composability are given. The modeling relation is discussed to motivate the understanding of the definitions.

3.1 Model

At the center of any theory of composability is the notion of a model. Because of its importance, we define model with care and explain the definition at some length. The official definition of model given by the Department of Defense is:

A model is a physical, mathematical, or otherwise logical representation of a system, entity, phenomenon or process (DOD, 1996) (DOD, 1998).

For example, consider the two models of height under gravity in Figure 7; note that both the equation (mathematical model) and the corresponding computer program (computer model) fit the official DOD definition of model.
\[ h = -16t^2 + vt + s \]

/* Height of an object moving under gravity. */
/* Initial height v and velocity s constants. */
main()
{
    float h, v = 100.0, s = 1000.0;
    int t;
    for (t = 0, h = s; h >= 0.0; t++)
    {
        h = (-16.0 * t * t) + (v * t) + s;
        printf("Height at time %d = %f\n", t, h);
    }
}

*Figure 7. Two models of height under gravity*

While intuitive and useful for a number of purposes, the DOD definition is not used in semantic composability theory for two reasons. First, it is insufficiently formal to use as the basis for formal reasoning, in part because it leaves open the form of the model, as shown in Figure 7. Second, in the DOD definition, a model is defined as a representation of a natural system. Thus, the definition assumes that the model has some meaning (i.e., semantics), in that the identity of the natural system is assumed to be known and the model is assumed to be a representation of it. Together, these assumptions imply validity\(^\text{17}\) by definition.

The modeling relation is the relation between a model and the system being modeled. One tool that may be useful in establishing this relationship is model theory. Model

\(^{17}\) We have not given a formal definition of validity yet; here we intend its informal meaning, i.e., a useful degree of closeness between the model and the system it is modeling.
theory establishes a link between a natural system and a formal system. This link is quite useful in the context of simulation since it is our aim to execute a model over time as a simulation. A formal system representation (i.e. computer program) is necessary to achieve this. However, this claim is not without controversy. A debate exists as to the extent models can be used to represent natural systems. The two opposing viewpoints are summarized in the quotations below:

The word ‘model’ has many other uses. For example, model theory is not about scientific theories as models of the world (Hodges, 2000).

In mathematical logic, a model is a structure – an arrangement of objects – which represents a theory expressed as a set of sentences. The various terms of the sentences of the theory are mapped onto objects and their relations in the structure; a model is a structure that makes all of the sentences in the theory true. This specialized notion of model has been adopted by philosophers of science; on a ‘structuralist’ or ‘semantic’ conception, scientific theories are understood as structures which are used to represent real systems in nature. Philosophical debates have arisen regarding the precise extent of the resemblances between scientific models and the natural systems they represent (Lloyd 2000).

![Diagram of Natural System and Formal System](image.png)

*Figure 8. The modeling relation with perfect knowledge*

Figure 8 (Casti, 1994) illustrates this link between a natural system and a formal system for the modeling relation with perfect knowledge (i.e. knowledge of the exact state of the

---

18 Natural systems refer to real or virtual systems we may wish to model.
Figure 9. The modeling relation without perfect knowledge

system with absolute certainty.) In this representation, relationships between observables in the natural system are represented by theorems in the formal system through the process of encoding. Decoding, of course, reverses this process. Figure 9 illustrates this more practical relationship between a natural system and a formal system, this time for
the modeling relation without perfect knowledge. In this representation, relationships between observables in the natural system are represented by theorems in the formal system through the process of encoding as before. However, the perfect model $N^*$ is an abstraction of the natural system $N$, achieved by limiting the number of observables of interest. Additionally, a number of equivalent formal systems are represented.

A definition of model is needed that is both formal and effectively reverses the informal definition's specificity with respect to semantics and ambiguity with respect to form is proposed. In other words, a definition of model that precisely specifies its form and is ambiguous as to semantics will be used. The notion of a model's semantics will be recaptured later through a formal definition of validity.

For semantic composability theory, a model is defined as follows:

**Definition 3.1.** A model $M$ is a computable function

$$M : X \rightarrow Y$$

where

$$X \subseteq S \times I,$$

$$Y \subseteq S \times O,$$

$S$ is a non-empty set of states,

$I$ is a set of inputs, and

$O$ is a set of outputs,

$$\vec{s} \in S, \vec{i} \in I, \text{ and } \vec{o} \in O$$

are vectors of integers.

The justification for this definition is three-fold.

1. The definition of a model as a function allows the use of the existing body of mathematical knowledge on functions. Recall, a function from set $A$ into set $B$ is a
rule $f$ that assigns to every member $a$ of set $A$ a unique member $b = f(a)$ of set $B$. The set $A$ is called the domain of the function and the set $B$ is called the codomain of the function (Hu, 1969). A partial function from set $A$ into set $B$ is defined similarly to a function from set $A$ into set $B$, but the rule $f$ may not be defined for every element of $A$ (Hein, 2002). Specifically for compositability, composition of functions is well defined.

2. Similarly the definition of a model as a computable function allows the use of the existing body of computability theory\textsuperscript{19}. Recall, a function $f : A \rightarrow B$ is computable if there exists a deterministic Turing machine\textsuperscript{20} that, for each element in $A$, halts on an element in $B$. Therefore, for each element of $A$ and each element of $B$ there exists a unique representation\textsuperscript{21} on the Turing machine tape, hence the representation of $\bar{s} \in S$, $\bar{i} \in I$, and $\bar{o} \in O$ as vectors of integers. This definition specifies vectors of integers, instead of single variables, or matrices. All of the values that can be represented on digital computers are integers. The so-called "real numbers" available in most programming languages are in fact integer approximations to real numbers. Theoretically, the restriction to integers is consistent with the assumptions of computability theory (Sommerhalder and van Westrhenen, 1988) (Barrow, 1992). Specifying $\bar{s} \in S$, $\bar{i} \in I$, and $\bar{o} \in O$ as vectors of integers, instead of single integers or matrices, is easy to justify. Vectors of integers can be mapped to single integers using a suitable variant of Cantor's method for mapping rational numbers, which can be represented as vectors with two elements, to single integers (Hein, 2002). A multivariate computable function is representable by superposition and composition of computable functions of one variable. (Brattka, 2000). A deterministic Turing machine (DTM) is a formal model of computation consisting of a finite state control, a read-write head, and a tape constructed from of an infinite sequence of tape squares. A program for a DTM consists of a finite set of tape symbols, a finite set of states, and a transition function. A set is countable if and only if it can be placed in a one-to-one correspondence with , or a subset of , where is the non-negative integers. Examples of countable sets include the integers, the rational numbers, and the set of vectors of integers (or rationals, floating-point values, ...) where each vector has $k$ components. The real numbers are uncountable.
similar argument works for matrices of integers to vectors. Indeed, different presentations of computability theory use different choices; computable functions have been defined using both vector-valued functions (Davis, Sigal, and Weyuker, 1994) and functions on single integers (Sommerhalder and van Westrhenen, 1988). We select vectors, rather than single integers or matrices, because it is simple to distinguish between input, output and state variables as elements of vectors. The models, ultimately, are intended to be implemented and executed as simulations on computers. Computers (at least current non-quantum computers) have computational power equivalent to Turing machines, i.e., they can only compute computable functions. Complex models, e.g., JSAF, may appear to be doing more than this, but because they are computer programs, ultimately they are computable functions. Hence the definition of models as computable functions is a practical matter; models that aren’t computable functions are of little interest to the M&S community. Note that computable functions are a subset of all functions; that is, some functions are not computable as proven by Turing (Turing, 1937) (Davis, 1982). Formal definitions of a computable function are available in the literature (Sommerhalder and van Westrhenen, 1988) (Davis, Sigal, and Weyuker, 1994) (Davis, 1982).

3. A philosophical argument is made in (Barrow, 1992) that the “unreasonable effectiveness of mathematics” in describing the physical world, which is the subject of models, is due to the fact that nature is computable, i.e., the laws of nature are computable. Hence the definition of models as computable functions is consistent with the subject of the models. A definition of model that is unambiguous and is based on existing theory will support the goal to prove results about models.

\( M \) can also be considered a computable partial function \( M : S \times I \rightarrow S \times O \). To avoid confusion, in this paper the word function is used to mean partial function; the term total function is used to indicate a function defined for all elements of the domain.

---

22 Technically, real computers are less powerful than Turing machines, because abstract Turing machines have infinite memory (the “tape” is assumed to be infinitely long) and real computers have finite memory. This means that defining models as computable functions in fact includes models that can’t be run on a real computer.
When convenient, $M$ may be partitioned to form state and output models. Given model $M : S \times I \to S$, called the state model of $M$, is a computable partial function such that for all $\bar{x} \in X$, $M_s(\bar{x}) = \bar{s}$ if and only if $M(\bar{x}) = (\bar{s}, \bar{o})$. Likewise, $M_o : S \times I \to O$, called the output model of $M$, is a computable partial function such that for all $\bar{x} \in X$, $M_o(\bar{x}) = \bar{o}$ if and only if $M(\bar{x}) = (\bar{s}, \bar{o})$. Note that $\tau \in I$, where $\tau$ is defined as the internal or null action. $\tau$ is not observable.

A model $M$ is total if $M$ is defined for all $(\bar{s}, \bar{i}) \in S \times I$. A total model $M$ is a computable function $M : S \times I \to S \times O$. Unless otherwise noted, model refers to the case where $M : S \times I \to S \times O$ is a computable partial function. A model $M$ is inputless if and only if $I = \emptyset$, then $M : S \to S \times O$, $M_s : S \to S$, and $M_o : S \to O$. It is sometimes considered, in this case, that $I = \{\tau\}$.

Other useful definitions include:

**Definition 3.2.** Given model $M : X \to Y$, a sub-model $M' : X' \to Y'$, denoted $M' \subseteq M$, is a computable function defined by $M'(\bar{x}) = M(\bar{x})$ if and only if $\bar{x} \in X'$ where $X' \subseteq X$ and $Y' \subseteq Y$.\(^\text{23}\)

**Definition 3.3.** $M^* : X^* \to Y^*$ is a super-model of $M : X \to Y$ if and only if $M \subseteq M^*$.

### 3.2 Simulation

A labeled transition system (LTS) (Roggenbach and Majster-Cederbaum, 2000) is a concept drawn from theoretical computer science that historically has seen little use in the applied simulation community. However, the LTS provides a useful alternate representation of simulation and is essential to the formal definition of validity presented in the next section.

\(^{23}\) A function is a relation.
Definition 3.4. A labeled transition system (LTS) is a tuple defined by

\[ T = (S, \Sigma, \rightarrow) \]

where

- \( S \) is a set of states,
- \( \Sigma \) is a set of labels, and
- \( \rightarrow \subseteq S \times \Sigma \times S \) is the transition relation.\(^{24}\)

An initial state \( s_0 \in S_0 \subseteq S \) may be considered where \( S_0 \) is a non-empty set of initial states. In this case the LTS is denoted by \( T = (S, s_0, \Sigma, \rightarrow) \). A terminating state \( s_r \in S_T \subseteq S \) may also be considered where \( S_T \) is a set of terminating states. \( S_T \) may be empty. \( S_0 \) and \( S_T \) are implicit in \( \rightarrow \). A labeled transition system is deterministic (DLTS) if \( s \rightarrow s' \) and \( s \rightarrow s'' \) implies \( s' = s'' \).\(^{25}\) A labeled transition system is represented by a directed multigraph \( G(S, \rightarrow) \) called a transition graph.

Example 3.1. Consider the labeled transition system \( T = (S, \Sigma, \rightarrow) \) defined by

\[ S = \{ s_1, s_2, s_3 \} \]
\[ \Sigma = \{ \sigma_1, \sigma_2, \sigma_3, \sigma_4 \} \]
\[ \rightarrow = \{ (s_1, \sigma_1, s_1), (s_1, \sigma_2, s_2), (s_2, \sigma_3, s_3), (s_2, \sigma_4, s_3) \} \]

\[ \begin{align*}
S_1 & \xrightarrow{\sigma_1} S_2 \\
S_2 & \xrightarrow{\sigma_3} S_3 \\
S_2 & \xrightarrow{\sigma_2} S_2 \\
S_3 & \xrightarrow{\sigma_4} S_3
\end{align*} \]

Figure 10. Transition graph for Example 3.1

\(^{24}\) For \( s, s' \in S, \sigma \in \Sigma, s \xrightarrow{\sigma} s' \) means \( (s, \sigma, s') \in \rightarrow \).

\(^{25}\) Unless otherwise noted, we are concerned with deterministic labeled transition systems.
The official definition of simulation utilized by the Department of Defense is:

Simulation is a method for implementing a model over time. Simulation also is a technique for testing, analysis or training in which real world systems are used, or where a model reproduces real world and conceptual systems (DOD, 1996) (DOD, 1998).

In the official definition of simulation, "implementing" actually seems to mean "executing". That is the sense of the term as commonly used; a simulation is an execution of a model over simulated time. For the theory developed here, the following definition of simulation is proposed.

**Definition 3.5.** Simulation is the sequential execution of a model and is represented by a deterministic labeled transition system

\[ L(M) = (S, I, M_s) \]

where

- \( M \) is a model, and
- \( M_s \) is the state model of \( M \).

If an initial state is specified then \( L(M, s_0) = (S, s_0, I, M_s) \). Note that \( L(M) = L(M_s) \).

Like the proposed definition of model, this definition of simulation is stripped of all explicit mention of the simulation representing anything, such as a real-world system. This has been done deliberately because defining a model or simulation as representing something is assuming validity. Validity is a property that models and simulations might or might not possess, not something that they should be defined or assumed to possess. Of course, it is generally intended that a simulation is an execution of a valid model, but that is not where a formal reasoning process should start.

Next, the concept of trajectory is defined.
**Definition 3.6.** A trajectory in $L(M, \bar{s}_0)$ is a sequence of alternating states and inputs beginning with $\bar{s}_0 \in S$ and defined by

\[
\left( \bar{s}_0, \bar{I}_1, \bar{s}_1, \bar{I}_2, ..., \bar{s}_T \right) \text{ or } \left( \bar{s}_0, \bar{I}_1, \bar{s}_1, \bar{I}_2, ... \right)
\]

where $\bar{s}_k = M(\bar{s}_{k-1}, \bar{I}_{k-1})$.

A trajectory is *terminating* if there exists a final state in the trajectory $\bar{s}_T \in S_T$. Otherwise, the trajectory is *non-terminating*.

**Example 3.2.** Two trajectories defined in the labeled transition system in Figure 10 are included below.

\[
\left( s_1, \sigma_2, s_2, \sigma_4, s_3 \right)
\]

\[
\left( s_1, \sigma_2, s_2, \sigma_3, s_1, \sigma_2, s_2, \sigma_4, s_3 \right)
\]

Simulation as a sequence of model executions is shown in Figure 11. This diagram is adapted from a description of the execution of a synchronous system that conveys the sense of simulation (Borål and Stålmarck, 1999) (Halbwachs et al., 1991).

*Figure 11. Notional example of simulation execution and associated transition graph.*
3.3 Validity

A model typically is intended to represent some real-world or notional system. For a given set of initial conditions and a set of inputs, a model for a natural system is to exhibit behavior that is close to the behavior of that system. Intuitively, the validity of the model is based upon how closely these behaviors match. Thus, a formal, quantitative definition of validity must include a measure of this closeness of behaviors. The behavior of the natural system is captured by defining a perfect model. A measure of closeness is achieved by comparing the behavior of the perfect model to the behavior of a candidate model using the concept of bisimulation.

**Definition 3.7.** A natural system $N$ is a real or imagined system. A natural system may be a function or a simulation.

**Definition 3.8.** A model is perfect with respect to a natural system $N$ if and only if

$$L(M^*) = (S^*, I, M^*_{s})$$

represents a system of perfect observations of the natural system $N$. In this case $M^*$ is called the perfect model.

It should be noted that a complete representation of $M^*$ is usually not available; rather, $M^*$ often must be approximated by making observations, either physical or notional, of the natural system. The validity of other models for the natural system is measured with respect to the perfect model. In order for another model $M$ to represent $M^*$, $M$ should exhibit behavior similar to that of $M^*$ for some specified set of initial conditions and some specified input set. A formal method for characterizing similar behavior is to compare the labeled transitions systems for $M$ and $M^*$. Two models exhibit similar behaviors for an input set if they generate similar trajectories when simulated. Milner and Park developed bisimulation to study processes that appear similar by external observation (Milner, 1980) (Park, 1981) (Milner, 1989). The formal approach for comparing all possible trajectories for a given input set is the concept of bisimulation.
The formal definitions of strong and weak bisimulation are as follows.

**Definition 3.9.** Let $T_1 = (P, \Sigma, \rightarrow)$ and $T_2 = (Q, \Sigma, \rightarrow)$ be labeled transition systems. A relation $R \subseteq P \times Q$ is a *strong bisimulation* if and only if for all $(p, q) \in R$, $\sigma \in \Sigma$,

1. if $p \xrightarrow{\sigma} p'$ in $T_1$ then $q \xrightarrow{\sigma} q'$ in $T_2$ and

   $$(p', q') \in R \text{ for some } q' \in Q, \text{ and}$$

2. if $q \xrightarrow{\sigma} q'$ in $T_2$ then $p \xrightarrow{\sigma} p'$ in $T_1$ and

   $$(p', q') \in R \text{ for some } p' \in P.$$ 

**Definition 3.10.** Let $T_1 = (P, \Sigma, \rightarrow)$ and $T_2 = (Q, \Sigma, \rightarrow)$ be labeled transition systems. A relation $R \subseteq P \times Q$ is a *weak bisimulation* if and only if for all $(p, q) \in R$, $\sigma \in \Sigma$,

1. if $p \xrightarrow{\sigma} p'$ in $T_1$ then $q \xrightarrow{\hat{\sigma}} q'$ in $T_2$ and

   $$(p', q') \in R \text{ for some } q' \in Q, \text{ and}$$

2. if $q \xrightarrow{\sigma} q'$ in $T_2$ then $p \xrightarrow{\hat{\sigma}} p'$ in $T_1$ and

   $$(p', q') \in R \text{ for some } p' \in P.$$ 

where $\hat{\sigma}$ denotes the sequence produced by deleting all the $\tau$ actions (internal actions) and

$$s \xrightarrow{\hat{\sigma}} s' \text{ means } s \xrightarrow{\tau'} s'.$$

Note that $s \xrightarrow{\omega_{ab}} s'$ means $s \xrightarrow{\tau'} s'$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
If a strong bisimulation $R$ exists, $T_1$ and $T_2$ are *strongly bisimilar* under $R$, denoted $T_1 \leftrightarrow^R T_2$. If a weak bisimulation $R$ exists, $T_1$ and $T_2$ are *weakly bisimilar* under $R$, denoted $T_1 \leftrightarrow^R T_2$. We will be concerned mostly with the concept of weak bisimulation. Unless otherwise noted, in this paper bisimulation means weak bisimulation.

The behavior of models can be compared using bisimulation.

**Definition 3.11.** Let $M_1$ and $M_2$ be models. $M_1$ is related to $M_2$ if there exists a relation $R$ such that $L(M_1) \leftrightarrow^R L(M_2)$. $M_1$ and $M_2$ are said to be related under $R$, denoted $M_1 \approx^R M_2$.

It is now possible to present a formal and quantitative definition for validity.

**Definition 3.12.** Let $M$ be a model and $M' \subseteq M^*$ where $M^*$ is perfect with respect to a natural system $N$. $M$ is valid if there exists a validity relation $V$ such that $L(M) \leftrightarrow^V L(M')$. $M$ is said to be valid under $V$, denoted $L(M) \Rightarrow^V L(M^*)$.

Zeigler defines two observation frames as *morphic* if their inputs, outputs, and time bases are in correspondence. These two systems are *isomorphic* if their inputs, outputs, and time bases are identical. At the state transition level, two systems $S$ and $S'$ are *homomorphic* if when $S'$ transitions through a sequence of states then $S$ transitions through a corresponding sequence of states (Zeigler, Praehofer, and Kim, 2000).

The concept of homomorphism is related to the concept defined here as validity. The difference, however, is that here transitions are formalized and identified with inputs, as necessary, through the mathematical construct of bisimulation. Bisimulation allows the formal definition of validity, and any number of validity relations, that characterizes the degree and nature of the "closeness" of the systems being compared.
3.4 Composition and composability

Because models have been defined as computable functions, composition of models becomes composition of functions, which is a well-defined mathematical concept. Because models are computable functions and it is known that the set of computable functions is closed under composition (Davis, Sigal, and Weyuker, 1994), any set of models can be composed if the composition exists. However, there is no guarantee that the resulting composite function will be a useful model. The focus of composability in the theory then becomes semantic composability, the question of whether the composite model is valid. The theory will be interested in whether properties of computable functions, such as validity, are preserved in composition.

Recall the definition of function composition.

Definition 3.13. Given functions \( f : A \rightarrow B \) and \( g : C \rightarrow D \), the composition, denoted \( h = f \circ g \), exists if and only if \( f(A) \subseteq C \) where \( f \circ g(x) = f(g(x)) \).

More generally, \( h(x) = f(g_1(x), g_2(x), ..., g_k(x)) \). Note that \( f, g_1, ..., g_k \) may be partial functions. Composition is similarly defined for relations. Suppose \( R_f \subseteq A \times B \) and \( R_g \subseteq C \times D \), then \( R_f \circ R_g \subseteq A \times D \) where \( (a, d) \in R_f \circ R_g \) if and only if \( (a, b) \in R_f \) and \( (b, d) \in R_g \). Note that \( R_f \circ R_g \) exists if and only if \( B \subseteq C \).

Here we focus on function composition of the form \( F \circ G \).

Definition 3.14. Given models \( F : X' \rightarrow Y' \), where \( X' \subseteq S' \times I' \) and \( Y' \subseteq S' \times O' \), and \( G : X \rightarrow Y \), where \( X \subseteq S \times I \) and \( Y \subseteq S \times O \), the syntactic composition, denoted \( F \circ G : X'' \rightarrow Y'' \), exists if and only if \( S = S' \). \( X'' \subseteq S \times I'' \) and \( Y'' \subseteq S \times O'' \) where \( I'' = I \times I' \) and \( O'' = O \times O' \).

Syntactic composability may require the construction of an interface \( W : S \rightarrow S' \).

A notional example of syntactic composition is provided in Figure 12. More complex compositions can be expressed in a similar fashion as in Figure 13.
Figure 12. Syntactic composition

Figure 13. Composition of models

More is needed to specify a composition of models similar to that shown in Figure 13.
Definition 3.15. Given models $M_1 : X_1 \rightarrow Y_1, M_2 : X_2 \rightarrow Y_2, \ldots, M_m : X_m \rightarrow Y_m$, a composition map $C_M$ is a relation such that $(y_{i,p}, x_{j,q}) \in C_M$ if and only if $y_{i,p}$ maps to $x_{j,q}$ where

$$M_i(x_i) = \bar{y}_i, \quad \bar{x}_i = \begin{bmatrix} x_{i,1} \\ x_{i,2} \\ \vdots \\ x_{i,q} \\ \vdots \\ x_{i,n} \end{bmatrix}, \quad \text{and} \quad \bar{y}_i = \begin{bmatrix} y_{i,1} \\ y_{i,2} \\ \vdots \\ y_{i,p} \\ \vdots \\ y_{i,n} \end{bmatrix}, \quad \text{and}$$

$$M_j(x_j) = \bar{y}_j, \quad \bar{x}_j = \begin{bmatrix} x_{j,1} \\ x_{j,2} \\ \vdots \\ x_{j,q} \\ \vdots \\ x_{j,n} \end{bmatrix}, \quad \text{and} \quad \bar{y}_j = \begin{bmatrix} y_{j,1} \\ y_{j,2} \\ \vdots \\ y_{j,p} \\ \vdots \\ y_{j,n} \end{bmatrix}.$$

Note that $C_M$ defines the composite vector $\bar{s}$ and a partial order on $M = (M_1, M_2, \ldots, M_m)$.

This leads to a new formal definition for composability.

Definition 3.16. The set of models $M = (M_1, M_2)$ is semantically composable if

1. $M_1 \circ M_2$ exists

2. A composition map $C_M$ exists such that $C_M$ defines the composite vector $\bar{s}$ and a partial order on $M = (M_1, M_2)$, and

3. $L(M_1 \circ M_2) \Rightarrow L(M^*_1 \circ M^*_2)$
The tuple $M_c = (M, C_M)$ defines a sufficient specification for a composition on $M = (M_1, M_2, ..., M_m)$. The statement $M_c = (M, C_M)$ is semantically composable implies $L(M, C_M) \Rightarrow L(M')$.

**Theorem 3.1.** Given $M_c = (M, C_M)$ where $M_c = (M, C_M)$ is semantically composable, $M = (M_1, M_2, ..., M_m)$, and $C_M$ defines the composite vector $\vec{s}$ and a partial order on $M = (M_1, M_2, ..., M_m)$, then $M_c = (M, C_M)$ is a model.

**Proof.** Let $C_M$ define the composite vector $\vec{s}$ and the partial order $M_1, M_2, ..., M_m$ on $M$. Since $M_c = (M, C_M)$ is semantically composable, then for each adjacent pair of models $M_i, M_j$ in the partial order $M_1, M_2, ..., M_m$, $M_j \circ M_i$ exists and $M_c = \left(M_m \left(M_{m-1} \cdots \left(M_1(\vec{s}, \vec{i}, \vec{i}), \vec{i}\right), \vec{i}\right)\right) = M_m \circ M_{m-1} \circ \cdots \circ M_1$.

Since the composition of computable functions is a computable function, then $M_c = (M, C_M)$ is a computable function (Davis, Sigal, and Weyuker, 1994) and, therefore, a model. ■

**Theorem 3.2.** Given $M_c = (M, C_M)$ where $M_c = (M, C_M)$ is semantically composable, $M = (M_1, M_2, ..., M_m)$, and $C_M$ defines the composite vector $\vec{s}$ and a partial order on $M = (M_1, M_2, ..., M_m)$, then $M_c = (M, C_M)$ generates a labeled transition system $L(M_c) = L(M_m \circ M_{m-1} \circ \cdots \circ M_1)$.

**Proof.** Let $C_M$ define the composite vector $\vec{s}$ and the partial order $M_1, M_2, ..., M_m$ on $M$. Since $M_c = (M, C_M)$ is semantically composable, then for each adjacent pair of models $M_i, M_j$ in the partial order $M_1, M_2, ..., M_m$, $M_j \circ M_i$ exists and $M_c = \left(M_m \left(M_{m-1} \cdots \left(M_1(\vec{s}, \vec{i}, \vec{i}), \vec{i}\right), \vec{i}\right)\right) = M_m \circ M_{m-1} \circ \cdots \circ M_1$.

Define $M'(\vec{x})$ to be a partial function of $\vec{x}$, such that

$$y_r = M'_r(\vec{x}) = \begin{cases} M_r(\vec{x}) & \text{if } M_r(\vec{x}) \text{ exists} \\ \vec{x}_r & \text{otherwise} \end{cases}.$$
The proof is provided by an algorithm for generating the labeled transition system.

1. Write $\bar{x}_1' = \bar{x} = (\bar{s}_{k-1}, \bar{t})$
2. Compute $\bar{y}'_1 = M_1'(\bar{x}_1') = (\bar{s}_1', \bar{o}_1')$
3. For $j \in 2, \ldots, m - 1$, write $\bar{x}_j' = (\bar{s}_{j-1}', \bar{t})$ and compute $\bar{y}_j' = M_j'(\bar{x}_j') = (\bar{s}_j', \bar{o}_j')$
4. Write $\bar{x}_m' = (\bar{s}_{m-1}', \bar{t})$ and compute $\bar{y} = M_m'(\bar{x}_m') = (\bar{s}_k, \bar{o})$ ■
In this section, it will be shown that given any finite set of component models and a specification of how they connect, the use of the combining operation of simple composition, together with a class of computable functions called interfaces, is sufficient to compose any desired composite model from the component models. The result depends only on the assumption that the models pass data from one to another without same-step loops or feedback, i.e., data output by a component model is not somehow also input to the computation that produces it, directly or indirectly. The composition sufficiency result is important because it simplifies the study of the validity of composite models, allowing the assumption that the composite models are assembled with simple composition. In that sense this section is preliminary to the next. As a secondary benefit, the idea of interface functions introduced in this section also eliminates the need to assume that all models have the same number of variables in their input and output vectors.

Proving that simple composition suffices for all composite models depends on some means of specifying any composite model. The specification of the structure of the composite model and the variable passed between the models will be via a relation on the models' input and output variables. That relation will induce a second relation on the models themselves, relating models that produce variables to models that use those variables. We will show that the assumption that there are no loops or feedback in the input/output connections implies that both relations are strict partial orders. Interfaces are necessary to prepare each models' input vector, but they perform no operations beyond rearranging and sub-setting the variables of vectors; in particular, they do not change the value of any of the variables. The proof that any such composite can be assembled as required proceeds by induction on the number of component models. These notions will be made formal in the development that follows.

4.1 Partially ordered sets

Definition 4.1. Let $A$ and $B$ be two sets. A relation $R$ from $A$ to $B$ is any set of pairs $(a, b)$ such that $a \in A$ and $b \in B$. If $(a, b) \in R$, we say that $a$ is related
to \( b \) by \( R \), written \( a \mathrel{R} b \). To express that \( R \) is a relation from \( A \) to \( B \), we write \( R : A \to B \). Note that for \( R : A \to B \), \( R \) is a subset of \( A \times B \). Adapted from (Grassmann and Tremblay, 1996).

**Definition 4.2.** A relation \( R : S \to S \) is called a *strict partial order* if it is irreflexive, antisymmetric, and transitive. A set \( S \) together with a (strict) partial order \( R \) is called a *strict partially ordered set* or a *strict poset*. The strict poset is the set \( S \) together with the strict partial order \( R \), i.e., the pair \((S, R)\). Adapted from (Grassmann and Tremblay, 1996).

**Definition 4.3.** Let \( (S, R) \) be a poset. Then \( y \in S \) is a *maximal* element if there is no \( x \in S \) such that \( x \mathrel{R} y \). Moreover, \( x \in S \) is a *minimal* element if there is no \( y \in S \) such that \( x \mathrel{R} y \). Adapted from (Grassmann and Tremblay, 1996).

**Lemma 4.1. (Poset Maximal Element Lemma)** Any finite strict partially ordered set (poset) \((F, R)\) has a maximal element.

**Proof.** Assume by way of contradiction that \((F, R)\) has no maximal element. Arbitrarily select an element \( f^{(1)} \) in \( F \). By assumption, \( f^{(1)} \) is not a maximal element, so there exists \( f^{(2)} \) such that \( f^{(2)} \mathrel{R} f^{(1)} \) and \( f^{(2)} \neq f^{(1)} \) because a strict poset is irreflexive by definition. By assumption \( f^{(2)} \) is also not a maximal element, so there exists \( f^{(3)} \) such that \( f^{(3)} \mathrel{R} f^{(2)} \), and \( f^{(3)} \neq f^{(2)} \). Also,

---

\[26\text{ It should be clear that a function is a special case of a relation, with the additional constraint that no two pairs } (a, b) \text{ in the relation have the same } a. \text{ For a relation } f : A \to B \text{ that is a function, the notation } b = f(a) \text{ replaces } a \mathrel{f} b. \text{ However, in this section we do not assume or require relations to be functions.}\

\[27\text{ Irreflexive: } (a, a) \notin R. \text{ Antisymmetric: } (a, b) \mathrel{R} (b, a) \notin R. \text{ Transitive: } (a, b) \mathrel{R} (b, c) \mathrel{R} (a, c) \in R.\]

\[28\text{ Note that for a poset like } (\mathbb{N}, <), \text{ these definitions of } \text{maximal} \text{ and } \text{minimal} \text{ are intuitively backwards, but reversing them would cause analogous confusion for } (\mathbb{N}, >). \text{ The definitions have to be one way or the other, so some counter-intuitiveness is unavoidable. They are given here as in (Grassmann and Tremblay, 1996).} \]
$f^{(3)} \neq f^{(1)}$ because otherwise $f^{(3)} R f^{(2)}$ and $f^{(2)} R f^{(3)} (= f^{(1)})$, and a poset is antisymmetric by definition. Continuing in a similar manner from $f^{(3)}$, we must conclude that there exists $f^{(n+1)}$ such that $f^{(n+1)} R f^{(n)}$, and $f^{(n+1)}, f^{(n)}, ..., f^{(2)}, f^{(1)}$ must all be distinct. But $|F| = n$, so $n + 1$ distinct elements in $F$ is a contradiction. Therefore $(F, R)$ has a maximal element. ■

**Lemma 4.2. (Poset Numbering Lemma)** Let $(F, \rightarrow)$ be a strict partially ordered set (poset). Then the elements of $F$ can be numbered $f^{(1)}, f^{(2)}, ..., f^{(n)}$ such that $i < j$ implies $f^{(i)} \notin f^{(j)}$. \[29\]

**Proof.** We give an algorithm to number the elements as required.

1. Set $i = 1$.
2. Select a maximal element $f \in F$.
3. Assign number $i$ to element $f$, i.e., denote it $f^{(i)}$.
4. Set $i = i + 1$.
5. Set $F = F - \{f^{(i)}\}$.
6. If $F = \emptyset$ then stop, else go to Step 2.

The algorithm terminates because in each iteration $F$, which is finite, is reduced in size by 1. The algorithm correctly numbers the elements because in each iteration the next available number is given to maximal element remaining, so no element that proceeds it in the order can be given a lower number. A maximal element must exist in each iteration by the Lemma 4.1 (Poset Maximal Element Lemma). ■

---

29 The $\rightarrow$ symbol for the poset relation is meant to suggest data being passed from one model to another, but another symbol might be preferable to avoid overloading $\rightarrow$, which is also used in function definitions. Note that the conclusion of the Lemma 4.2 (Poset Numbering Lemma) can't be written "$i < j$ implies $(f^{(i)}, f^{(j)}) \notin \rightarrow$", because not all elements of a poset are necessarily comparable.
4.2 Vectors and models

**Definition 4.4.** Let $Z^k$ be the set of vectors of integers of size $k$, i.e., with $k$ variables. $Z^*$ is the set of all finite vectors of integers of any finite size $k \geq 1$; $Z^* = Z^1 \cup Z^2 \cup ...$

**Definition 4.5.** Let $F = \{f_1, f_2, ..., f_n\}$ be a set of models, i.e., computable functions, with $f_i : Z^* \to Z^*$ for $1 \leq i \leq n$. We call the models in $F$ *component models*; they will be composed into *composite models*.

**Definition 4.6.** Model $f_u \in F$ has input and output vectors $\bar{x}_u$ and $\bar{y}_u$ respectively, i.e., $\bar{y}_u = f_u(\bar{x}_u)$. Recall that $\bar{x}_u, \bar{y}_u \in Z^*$; denote their individual variables as

$$\bar{x}_u = [x_{u,1}, x_{u,2}, ..., x_{u,|\bar{x}_u|}]$$

$$\bar{y}_u = [y_{u,1}, y_{u,2}, ..., y_{u,|\bar{y}_u|}]$$

Other vectors are also input to or output from the composite model:

$m = [m_1, m_2, ..., m_{|m|}]$ State vector passed from previous simulation step.

$m_{\text{next}} = [m_{\text{next},1}, m_{\text{next},2}, ..., m_{\text{next},|m_{\text{next}}|}]$ State vector passed to next simulation step.

$i = [i_1, i_2, ..., i_{|i|}]$ Interactive input.

$o = [o_1, o_2, ..., o_{|o|}]$ Interactive output.
4.3 Specifying variable mappings in composite models

Definition 4.7. Let $X_F$ be the set of all input variables and $Y_F$ be the set of all output variables for the models of $F$. Treating the vectors $\bar{x}_u$ and $\bar{y}_u$ as sets of variables, then $X_F$ and $Y_F$ are initially defined as:

$$X_F = \bar{x}_1 \cup \bar{x}_2 \cup ... \cup \bar{x}_n$$

$$Y_F = \bar{y}_1 \cup \bar{y}_2 \cup ... \cup \bar{y}_n$$

Let $M$ be a relation from $Y_F$ to $X_F$, i.e., $M : Y_F \rightarrow X_F$. $M$ specifies how the variables output by models are later input to other models in the composite model. $M$ is a set of pairs $(\bar{y}_{u,j}, \bar{x}_{v,k})$ with $y_{u,j} \in Y_F$ and $x_{v,k} \in X_F$. A pair $(\bar{y}_{u,j}, \bar{x}_{v,k}) \in M$ specifies that value of output variable $\bar{y}_{u,j}$, produced by model $f_u$, is needed as the value of input variable $\bar{x}_{v,k}$, used by model $f_v$. Thus $M$ can be understood as a mapping from outputs to inputs, specifying how the variables’ values are to be passed from model to model in composite model $M$.

By placing a restriction on $M$ we formalize the informal assumption that the models in $F$ pass data from one to another without loops or feedback. To do so it is necessary to relate each variable in $X_F$ and $Y_F$ to its model, i.e., the model that inputs or outputs it.

Let function $g : X_F \cup Y_F \rightarrow F$ specify the model for each variable. Then the restriction on $M$ is as follows:

$$\neg \exists \bar{x}_1, \bar{y}_1, \bar{x}_2, \bar{y}_2, ..., \bar{x}_j, \bar{y}_j, 1 \leq j \leq n, \text{ such that } (\bar{y}_1, \bar{x}_1), (\bar{y}_2, \bar{x}_2), ..., (\bar{y}_j, \bar{x}_j) \in M,$$

$$g(\bar{x}_i) = g(\bar{y}_{i+1}) \text{ for } 1 \leq i \leq j, \text{ and } g(\bar{y}_j) = g(\bar{y}_1).$$

In other words, $M$ is noncyclic; there is no cycle of variables in $M$ that begins and ends at the same model. This restriction is all that is needed.

In addition to the model output variables in $Y_F$ as initially defined, the values of the variables of $\bar{m}$ (the state vector from the previous simulation step) and $\bar{t}$ (the interactive
input to the composite model) may be needed as the value of some model's input variable. Similarly, in addition to the model input variables in $X_F$ as initially defined, the variables of $\overline{m}_{next}$ (the state vector to be passed to the next simulation step) and $\overline{o}$ (the interactive output of the composite model) may need to receive the value of a model's output variable. Therefore, those variables are also included in the final definitions of $X_F$ and $Y_F$:

$$X_F = \overline{x}_1 \cup \overline{x}_2 \cup \ldots \cup \overline{x}_n \cup \overline{m}_{next} \cup \overline{o}$$

$$Y_F = \overline{y}_1 \cup \overline{y}_2 \cup \ldots \cup \overline{y}_n \cup \overline{m} \cup \overline{t}$$

The change to the definitions to $X_F$ and $Y_F$ do not change the definition of relation $M$. As before, $M$ maps from $Y_F$ to $X_F$ and has no cycles.

$M$ specifies the connections between the models that must be made when composing the composite model. $M$ relates variables that are either available to the composite model from the outside (passed from the previous simulation step or interactively input) or output by the models to variables that are either input to the models are made available by the composite model to the outside (passed to the next simulation step or interactively output). Relation $M$ therefore specifies the structure of the composite model to be assembled from the component models in $F$.

In addition to the relation $M$ between specific variables of the models of $F$, we are interested also in the relation between the models of $F$ induced by $M$. Informally, if any output variable of model $f_u$ is later input to model $f_v$, then $f_u$ will be related to $f_v$. This relation will eventually determine the sequence in which the models of $F$ are composed.

**Definition 4.8.** Relation $M'$ is a relation on the set of models, i.e., $M' : F \rightarrow F$.

$M'$ is induced from $M$ so that if at least one variable output from model $f_u$ is mapped to at least one variable input to model $f_v$ by $M$, then $f_u$ is related to $f_v$ by $M'$. Formally:
\[(f_u, f_v) \in M' \Leftrightarrow \exists j \exists k \text{ such that } (\overline{v}_{u,j}, \overline{x}_{v,k}) \in M.\]

Note that \(M'\) is not transitive, that is \((f_u, f_v)(f_v, f_w) \in M'\) does not necessarily imply \((f_u, f_w) \in M',\) because \(f_w\) may not necessarily input any variable output by \(f_u.\) Because a strict partial order on the models of \(F\) will be needed, then the transitive closure of \(M'\) is defined.

**Definition 4.9.** Relation \(M^+\) is the transitive closure of \(M'.\) In other words, \(M' \subseteq M^+\) and \((f_u, f_v)(f_v, f_w) \in M^+ \Rightarrow (f_u, f_w) \in M^+.\)

**Lemma 4.3. (Model Set Ordering Lemma)** The transitive closure \(M^+\) of the relation \(M'\) induced by relation \(M,\) when \(M\) is restricted to be noncyclic, is a strict partial order on \(F.\)

**Proof.** For \(M^+\) to be a strict partial order, it must be irreflexive, antisymmetric, and transitive on \(F.\) First, to show that \(M^+\) is irreflexive, assume by way of contradiction that it is reflexive. Then there exists \(f_u\) such that \((f_u, f_u) \in M^+.\)

\((f_u, f_u) \in M^+\) implies that there exists \(f_u\) such that \((f_u, f_u) \in M',\) because the transitive closure would not have added \((f_u, f_u)\) to \(M^+.\) \((f_u, f_u) \in M'\) implies that there exists \(\overline{y}_{u,j}, \overline{x}_{u,k}\) such that \((\overline{y}_{u,j}, \overline{x}_{u,k}) \in M.\) But \(\overline{g} (\overline{y}_{u,j}) = \overline{g} (\overline{x}_{u,k}),\) which contradicts the noncyclic restriction on \(M.\) Thus \(M^+\) must be irreflexive. By similar reasoning, assuming that \(M^+\) is symmetric implies a prohibited cycle in \(M,\) so \(M^+\) must be antisymmetric. Finally, \(M^+\) is transitive by definition as a transitive closure. Thus \(M^+\) is a strict partial order on \(F.\)

If the transitive closure \(M^+\) of the relation \(M'\) induced by relation \(M\) is a strict partial order on \(F,\) we say that \(M\) induces a strict partial order on \(F.\)

---

\(^{30}\) The use of the + superscript to indicate transitive closure is from (Grassmann and Tremblay, 1996).
4.4 Assembling composite models

The models in $F$ are sequenced in the composition first-to-last (inner-to-outer) based on the strict partial order induced by $M$. The composite function is assembled from the models in $F$ using simple composition.

Each model $f_v$ in $F$ takes a vector $\bar{x}_v$ as input and produces a vector $\bar{y}_v$ as output.

$$\bar{y}_v = f_v(\bar{x}_v)$$

Interfaces are used to provide the needed variables to each model; they provide each model a "customized" input by selecting and rearranging the variables each model needs from among those available. For model $f_v$ with input $\bar{x}_v$, interface $w_v$ must have access to (as input) all the variables that could possibly be input to $f_v$. The possible input variables include the previous state vector $\bar{m}$, the interactive input $\bar{i}$, and the outputs $\bar{y}_1, \bar{y}_2, \ldots, \bar{y}_{v-1}$ of the models that precede $f_v$ in the partial order. Interface $w_v$ selects from among those variables just the variables needed by $f_v$ and rearranges them to assemble $\bar{x}_v$; it does not change any of the variables' values.

$$\bar{x}_1 = w_1(\bar{m}, \bar{i})$$
$$\bar{x}_2 = w_2(\bar{m}, \bar{i}, \bar{y}_1)$$
$$\bar{x}_3 = w_3(\bar{m}, \bar{i}, \bar{y}_1, \bar{y}_2)$$

A specialized interface $w_v$ is needed for each model $f_v$, but we will show that all such interfaces are computable.

**Definition 4.10.** An *interface* is a function $w: (\mathbb{Z}^*)^n \to \mathbb{Z}^*$. It takes as input a finite number of finite-sized vectors of integers and produces as output a single finite-sized vector of integers. An interface $w$ copies variables from the input
vector to the output vector, possibly reordering the input variables or omitting a subset of them, but not changing any of their values.\textsuperscript{31}

\textbf{Lemma 4.4. (Interface Computability Lemma)} Any interface \( w : (Z^*)^r \rightarrow Z^* \) is computable.

\textbf{Proof.} The input vectors can be concatenated into a single vector of finite size; denote its size as \( k \). Let \( \bar{x} = [x_1, x_2, \ldots, x_k] \) be the input vector and \( \bar{y} = [y_1, y_2, \ldots, y_m] \) be the output vector. Let \( (j_1, j_2, \ldots, j_m) \) denote the indices of the input variables to be copied to the output vector as \( y_i = \bar{x}_{j_i}, 1 \leq i \leq m \). Then the output vector can be computed as a concatenation of projection functions\textsuperscript{32} as \( \bar{y} = [\pi_{j_1}, \pi_{j_2}, \ldots, \pi_{j_m}] \). Thus any interface \( w \) can be expressed as a composition of only concatenation and projection functions. Concatenation and projection functions are defined as computable (Sommerhalder and van Westrhenen, 1988), and any composition of computable functions is computable (Davis, Sigal, and Weyuker, 1994), so any interface \( w \) is computable. ■

To illustrate how the models and interfaces are composed into composite models, the composite models for \( f_1 \) to \( f_3 \) are shown.

\[
\begin{align*}
\bar{y}_1 &= f_1(\bar{x}_1) = f_1(w_1(m, \bar{i})) \\
\bar{y}_2 &= f_2(\bar{x}_2) = f_2(w_2(m, \bar{i}, f_1(w_1(m, \bar{i}))))) \\
\bar{y}_3 &= f_3(\bar{x}_3) = f_3(w_3(m, \bar{i}, f_1(w_1(m, \bar{i})), f_2(w_2(m, \bar{i}, f_1(w_1(m, \bar{i})))))
\end{align*}
\]

To simplify these expressions, the model invocations can be replaced with their output vectors.

\textsuperscript{31} By definition, any function that changes the value of a variable is a \textit{model}, not an \textit{interface}.

\textsuperscript{32} The projection function \( \pi_j^k \) is defined as returning variable \( j \) from a vector of size \( k \) without changing its value (Sommerhalder and van Westrhenen, 1988).
\[
\begin{align*}
\bar{y}_1 &= f_1(\bar{x}_1) = f_1(w_1(\bar{m}, \bar{i})) \\
\bar{y}_2 &= f_2(\bar{x}_2) = f_2(w_2(\bar{m}, \bar{i}, \bar{y}_1)) \\
\bar{y}_3 &= f_3(\bar{x}_3) = f_3(w_3(\bar{m}, \bar{i}, \bar{y}_1, \bar{y}_2))
\end{align*}
\]

More generally, each composite model is assembled from the composite models for all models that precede it in the partial order.

\[
\begin{align*}
\bar{y}_{v-1} &= f_{v-1}(\bar{x}_{v-1}) = f_{v-1}(w_{v-1}(\bar{m}, \bar{i}, \bar{y}_1, \bar{y}_2, ..., \bar{y}_{v-1})) \\
\bar{y}_v &= f_v(\bar{x}_v) = f_v(w_v(\bar{m}, \bar{i}, \bar{y}_1, \bar{y}_2, ..., \bar{y}_{v-1})) \\
\bar{y}_{v+1} &= f_{v+1}(\bar{x}_{v+1}) = f_{v+1}(w_{v+1}(\bar{m}, \bar{i}, \bar{y}_1, \bar{y}_2, ..., \bar{y}_v))
\end{align*}
\]

Note that the output \( \bar{y}_v \) of \( f_v \) does not necessarily include all of the variables in \( \bar{m}, \bar{i}, \bar{y}_1, \bar{y}_2, ..., \bar{y}_{v-1} \), so all of those vectors will have to be input again to \( w_{v+1} \) along with \( \bar{y}_v \) for possible inclusion by \( w_{v+1} \) in \( \bar{x}_{v+1} \) for \( f_{v+1} \). For example, that is why \( \bar{y}_1 \) is repeated in the input to \( w_3 \) for \( f_3 \), even though it is also in the input to \( w_2 \) for \( f_2 \) that also is input to \( w_3 \).

Because \( F \) has contains \( n \) models, model output \( \bar{y}_v \) is input \( n-v \) times, to \( w_{v+1}, w_{v+2}, ..., w_n \). These multiple inputs of \( \bar{y}_v \) would in an implementation likely be from a single copy of \( \bar{y}_v \) that had been computed once and saved. However, in the formal development of the Composition Sufficiency Theorem (to come) there is no notion of memory, so the references to \( \bar{y}_v \) should be understood as placeholders for the model invocations \( f_v(w_v(\bar{m}, \bar{i}, \bar{y}_1, \bar{y}_2, ..., \bar{y}_{v-1})) \) in every case.

Any of the models’ output variables, and indeed even variables from \( \bar{m} \) and \( \bar{i} \), may be output from the composite model as \( \bar{m}_{\text{next}} \) or \( \bar{o} \). Two final interfaces assemble \( \bar{m}_{\text{next}} \) and \( \bar{o} \) by selecting from and rearranging the available variables.

\[
\begin{align*}
\bar{m}_{\text{next}} &= w_{n+1}(\bar{m}, \bar{i}, \bar{y}_1, \bar{y}_2, ..., \bar{y}_n) \\
\bar{o} &= w_{n+2}(\bar{m}, \bar{i}, \bar{y}_1, \bar{y}_2, ..., \bar{y}_v)
\end{align*}
\]
Example 4.1. For example, Figure 14 illustrates one possible composite model. In the figure, the sets and relations defined so far have these values:

\[
F = \{f_1, f_2, f_3\}
\]

\[
\bar{x}_1 = [x_{1,1}, x_{1,2}, x_{1,3}]
\]

\[
\bar{y}_1 = [y_{1,1}, y_{1,2}, y_{1,3}, y_{1,4}]
\]

\[
\bar{x}_2 = [x_{2,1}, x_{2,2}, x_{2,3}, x_{2,4}]
\]

\[
\bar{y}_2 = [y_{2,1}, y_{2,2}, y_{2,3}, y_{2,4}]
\]

\[
\bar{x}_3 = [x_{3,1}, x_{3,2}, x_{3,3}, x_{3,4}, x_{3,5}, x_{3,6}, x_{3,7}]
\]

\[
\bar{y}_3 = [y_{3,1}, y_{3,2}, y_{3,3}, y_{3,4}, y_{3,5}, y_{3,6}, y_{3,7}, y_{3,8}]
\]
\[ \overline{m} = [m_1, m_2, m_3, m_4, m_5] \]
\[ \overline{m}_{\text{next}} = [m_{\text{next},1}, m_{\text{next},2}, m_{\text{next},3}, m_{\text{next},4}, m_{\text{next},5}] \]
\[ \bar{i} = [i_1, i_2, i_3, i_4] \]
\[ \bar{o} = [o_1, o_2, o_3, o_4, o_5, o_6, o_7, o_8] \]
\[ X_F = \{ x_{1,1}, x_{1,2}, x_{1,3}, x_{2,1}, x_{2,2}, x_{2,3}, x_{3,1}, x_{3,2}, x_{3,3}, x_{3,4}, x_{3,5}, x_{3,6}, x_{3,7} \} \]
\[ Y_F = \{ y_{1,1}, y_{1,2}, y_{1,3}, y_{1,4}, y_{2,1}, y_{2,2}, y_{2,3}, y_{2,4}, y_{3,1}, y_{3,2}, y_{3,3}, y_{3,4}, y_{3,5}, y_{3,6}, y_{3,7}, y_{3,8} \} \]
\[ M = \left\{ \begin{array}{l}
(i_1, x_{1,1}, m_{1}, x_{1,2}, y_{1,1}, y_{1,2}, y_{1,3}, y_{1,4}, y_{2,1}, y_{2,2}, y_{2,3}, y_{2,4}, y_{3,1}, m_{\text{next},1}, y_{3,2}, m_{\text{next},2}, y_{3,3}, m_{\text{next},3}, y_{3,4}, m_{\text{next},4}, y_{3,5}, m_{\text{next},5}, y_{3,6}, y_{3,7}, y_{3,8})
\end{array} \right\} \]

### 4.5 Composition sufficiency theorem

With the preliminaries established, we now turn to the important question of this section. Are simple composition and interfaces sufficient to assemble any composite model, given a set of component models and a specification of their connections? The answer is yes.

**Theorem 4.1. (Composition Sufficiency Theorem)** Given any finite set \( F = \{f_1, f_2, \ldots, f_n\} \) of models and any relation \( M \) on the variables of \( F \), if \( M \) induces a strict partial ordering of \( F \), then there exists a simple composition of the models in \( F \) and interfaces that satisfies \( M \) and is computable.

**Proof.** Assume without loss of generality that the \( n \) models in \( F \) have been numbered as per Lemma 4.2 (Poset Numbering Lemma) so that for \( f_u, f_v \in F \), \( u < v \) implies \( f_u \prec M' f_v \). The proof is by induction on model number. At each step \( j, 1 \leq j \leq n \), it must be shown that a composite model of the first \( j \) models
in \( F \) meets three criteria: (1) it can be assembled using simple composition of the models and interfaces, (2) it satisfies \( M \) by providing access to all variables needed as input by each model, and (3) it is computable. For brevity, we describe a composite model that meets these three criteria as "well-formed".

**Inductive base:**\(^{33}\) For \( j = 1 \), the composite model is \( f_1(w_1(\overline{m}, \overline{i})) \). The composite model can be seen by inspection to be assembled by simple composition of models and interfaces. Because it is first in the ordering on \( F \), \( f_1 \) has no inputs from other models in \( F \), so it can take inputs only from \( \overline{m} \) and \( \overline{i} \), which are available to it via interface \( w_1 \). The composite model is computable because the component models are computable by definition, the interfaces are computable by the Lemma 4.4 (Interface Computability Lemma), and computable functions are known to be closed under composition (Davis, Sigal, and Weyuker, 1994). Therefore the composite model for \( j = 1 \) is well-formed.

**Inductive hypothesis:** Assume the composite model for \( j, 1 \leq j < n \), is well-formed.

**Inductive step:** For \( j + 1, j < j + 1 \leq n \), the composite model is \( f_{j+1}(w_{j+1}([m, i, f_1(x_1), f_2(x_2), \ldots, f_j(x_j)]) \). The composite model can be seen by inspection to be assembled by simple composition of models and interfaces. Because of the strict partial ordering of \( F \), \( f_{j+1} \) requires as input only variables in \( \overline{m}, \overline{i} \), and in \( \overline{y_1}, \overline{y_2}, \ldots, \overline{y_j} \) produced from models \( f_1, f_2, \ldots, f_j \). All of these variables are available in the composite model for \( f_{j+1} \) by invocation of \( f_1, f_2, \ldots, f_j \) all of which have been shown to be well-formed. Interface \( w_{j+1} \) selects and arranges the variables for \( \overline{x_{j+1}} \) as needed by \( f_{j+1} \). The composite

---

\(^{33}\) The number and names of the steps of the mathematical induction proof method vary by source, though all are essentially similar, of course. Here we follow the abbreviated form of (Grassmann and Tremblay, 1996).
model is computable because all of its components are computable. Therefore \( w_{j+1} \) is well formed.

**Conclusion:** Because the inductive base and the inductive step are established, a well-formed composite model can be assembled for any finite \( F \) and any \( M \) that induces a strict partial order on \( F \). ■

### 4.6 Consequences of the composition sufficiency theorem

Theorem 4.1 (Composition Sufficiency Theorem) as given shows that any composite model can be composed using simple composition of models and interfaces. The most important consequence of this result is that theorems concerning validity under composition can assume that simple composition only is used to compose the models, thus simplifying the proofs, yet still cover all possible composite models.

This theorem, and the notion of interfaces, also eliminates the need to require that all models have the same size input and output vectors. Inside the composite model, the mapping \( M \) and the interfaces \( w_1, w_2, \ldots, w_n \) adjust the vector sizes as needed. Outside the composite model, interfaces \( w_{n+1} \) and \( w_{n+2} \) adjust the vector sizes for \( \bar{m}_{\text{res}} \) and \( \bar{o} \) respectively.

It should be easy to show, using an approach much like the Theorem 4.1 (Composition Sufficiency Theorem), that simple composition of models and interfaces are sufficient to compose a model equivalent to any simulation, at least in terms of producing the final state.
5 VALIDITY OF CLASSES OF MODELS UNDER COMPOSITION

In a previous section, validity was defined as a measure of the closeness of behaviors between a model and some real-world or notional system. In this section, classes of models and classes of validity relations are defined and then the validity of composition of models is considered.

5.1 Classes of models

Several classes of models are defined here, others are possible.

**Definition 5.1.** A model $M$ is linear if and only if

1. $M(x' + x") = M(x') + M(x")$, and
2. $M(\alpha \bar{x}) = \alpha M(\bar{x})$

for all $x, x', \bar{x} \in S \times I$ and scalars $\alpha$.

A linear model can be represented by $M(\bar{x}) = A\bar{x}$ where $A$ is a matrix.

**Example 5.1.** Consider a linear model $M$.

$$M \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 3x_1 - 2x_2 \\ 2x_2 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 3 & -2 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$= \begin{bmatrix} 4 & -2 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

**Definition 5.2.** A model $M$ is affine if and only if there exists a matrix $A$ and constant vector $\bar{c}$ such that

$$M(\bar{x}) = A\bar{x} + \bar{c}$$

for all $\bar{x} \in S \times I$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Example 5.2. Consider an affine model \( M \).

\[
M \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 4x_1 + 3x_2 + 2 \\ 2x_1 + 2x_2 + 1 \end{bmatrix}
= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 4 & 3 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 2 \\ 1 \end{bmatrix}
= \begin{bmatrix} 5 & 3 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 2 \\ 1 \end{bmatrix}
\]

Definition 5.3. A model \( M \) is algebraic if and only if \( M \) is composed using only a finite number of elementary operations (addition, subtraction, multiplication, division, and rational root extraction) and inverses of similar development.

Definition 5.4. A model \( M \) is elementary if and only if \( M \) is composed using only a finite number of constant, algebraic, exponential, and logarithmic functions and inverses of similar development. Included in the elementary functions are the trigonometric and hyperbolic functions.

Definition 5.5. A model \( M \) is computable if and only if \( M \) is a computable function. This is the class of all models.

Definition 5.6. A model \( M \) is enumerated if and only if \( M \) is defined only by enumerating changes in the state of the model.

5.2 Classes of validity relations

Quality of closeness is achieved by requiring the bisimulation to have special properties.

Definition 5.7. A relation \( R \subseteq P \times Q \) is an equivalence relation if and only if

1. \( (p, p) \in R \)
2. \( (p, q) \in R \Rightarrow (q, p) \in R \), and
3. \( (p, q) \in R \) and \( (q, r) \in R \Rightarrow (p, r) \in R \).
Given $L(M_1) \Leftrightarrow_r L(M_2)$, if $R$ is an equivalence relation, $M_1$ and $M_2$ are said to be equivalent, denoted $M_1 =_R M_2$. Given $L(M) \Leftrightarrow_v L(M^*)$, if $V$ is an equivalence relation, $M$ is said to be valid under equivalence, denoted $M =_V M^*$.

The equivalence relation establishes a very strong constraint for validity; often a less constraining criterion is acceptable. Another useful relation that preserves the notion of closeness is the metric relation.

**Definition 5.8.** A metric is a function $u : P \times Q \to \mathbb{Z}^+$ satisfying

1. $u(p, p) = 0$
2. $u(p, q) = u(q, p)$
3. $u(p, q) = 0 \rightarrow p = q$, and.
4. $u(p, q) + u(q, r) \geq u(p, r)$.

**Example 5.3.** For vectors $\vec{p}$ and $\vec{q}$. Let $u(\vec{p}, \vec{q}) = \|\vec{p} - \vec{q}\|$.  

1. $u(\vec{p}, \vec{p}) = \|\vec{p} - \vec{p}\| = 0 \rightarrow u(\vec{p}, \vec{p}) = 0$
2. $u(\vec{p}, \vec{q}) = \|\vec{p} - \vec{q}\| = \|\vec{q} - \vec{p}\| = u(\vec{q}, \vec{p})$
3. $u(\vec{p}, \vec{q}) = 0 \rightarrow \|\vec{p} - \vec{q}\| = 0 \rightarrow \vec{p} = \vec{q}$, and.
4. $u(\vec{p}, \vec{q}) + u(\vec{q}, \vec{r}) = \|\vec{p} - \vec{q}\| + \|\vec{q} - \vec{r}\| \geq \|(\vec{p} - \vec{q}) + (\vec{q} - \vec{r})\| = \|\vec{p} - \vec{r}\| = u(\vec{p}, \vec{r})$

**Example 5.4.** For vectors $\vec{p}$ and $\vec{q}$. Let $u(\vec{p}, \vec{q}) = \begin{cases} 0 & \text{if } \vec{p} = \vec{q} \\ 1 & \text{otherwise} \end{cases}$.

1. $\vec{p} = \vec{p} \rightarrow u(\vec{p}, \vec{p}) = 0$
2. if $\vec{p} = \vec{q}$ then $u(\vec{p}, \vec{q}) = 0 = u(\vec{q}, \vec{p})$
if $p \neq q$ then $u(p, q) = 1 = u(q, p)$

3. $u(p, q) = 0 \Rightarrow p = q$, and

4. if $p = q$, $q = r$, $p = r$ then $u(p, q) + u(q, r) = 0 + 0 \geq 0 = u(p, r)$

if $p \neq q$, $q = r$, $p \neq r$ then $u(p, q) + u(q, r) = 0 + 1 \geq 1 = u(p, r)$

if $p \neq q$, $q \neq r$, $p \neq r$ then $u(p, q) + u(q, r) = 1 + 1 \geq 1 = u(p, r)$

if $p \neq q$, $q \neq r$, $p = r$ then $u(p, q) + u(p, r) = 1 + 1 \geq 1 = u(p, r)$

if $p = q$, $q \neq r$, $p = r$ then $u(p, q) + u(p, r) = 1 + 1 \geq 1 = u(p, r)$

if $p = q$, $q = r$, $p \neq r$ then $u(p, q) + u(p, r) = 1 + 1 \geq 1 = u(p, r)$

**Definition 5.9.** A relation $R \subseteq P \times Q$ is an **metric relation with parameter $\delta$** if and only if for all $(p, q) \in R$, $u(p, q) \leq \delta$.

Given $L(M) \Rightarrow M'(M^*)$, if $V$ is a metric relation, $M$ is said to be **valid under metric $u$ with parameter $\delta$**. A variation of the metric relation is the **iteration specific metric relation** in which $u(p, q) \leq \delta(k)$ where $\delta(k)$ is a function of the number of iterations. Figure 15 is an notional example of validity under a metric.

---

Here equivalence relations and metric relations are defined, but others are possible.
Zeigler discusses the concept of *approximate homomorphism* when the states in the sequence may differ. There is additional discussion on bounds on error (Zeigler, Praehofer, and Kim, 2000).

The difference, however, is that in this work the difference or error is formalized by well-defined classes of validity relations that characterize the degree and nature of the "closeness" of the systems being compared. This mathematical definition using validity relations allows formal reasoning on the systems being compared. In the next section, this formal definition is applied to reason about the composability of compositions of models from various model classes.

### 5.3 Validity under composition

Generalizing the question of validity under composition, an important purpose of semantic composability theory is to establish the validity of compositions of models for different classes of models and validity relations. For some classes of models and relations, it is possible to prove that validity is preserved when valid models are composed. Clearly, results of this type could be of considerable value in practical applications of composability. Classes of models being studied include linear functions, affine functions, algebraic functions, elementary functions, and computable functions; classes of relations are equivalence relations and metric relations. Here, only validity of compositions of similarly developed models and validity relations are considered. Future research will consider more complex compositions.

The objective in the proof of compositions of models that are valid under equivalence relations is to show that given $L(F) \Rightarrow_{V} L(F^*)$ and $L(G) \Rightarrow_{V} L(G^*)$ where $V_F$ and $V_G$ are equivalence relations, $L(F \circ G) \Rightarrow_{V_{F \circ G}} L(F^* \circ G^*)$ where $V_{F \circ G}$ is an equivalence relation.

The objective in the proof of compositions of models that are valid under iteration-specific metric relations is as follows. Given $L(F) \Rightarrow_{V_F} L(F^*)$, where $V_F$ is a iteration-specific metric relation such that for all $(\tilde{s}_k, \tilde{s}_k^*) \in V_F$, $u(\tilde{s}_k, \tilde{s}_k^*) \leq \delta_F(k)$ where
\( \delta_z(k) = O(t(k)) \), and \( F_k(x_{k-1}) = s_k \), and \( L(G) \Rightarrow_{V_{G}} L(G^*) \), where \( V_{G} \) is a iteration-specific metric relation such that for all \( (s_k, s_k^*) \in V_{G} \), \( u(s_k, s_k^*) \leq \delta_i(k) \) 34 where \( \delta_i(k) = O(t(k)) \) 35, and \( G_k(x_{k-1}) = s_k \), \( L(F \circ G) \Rightarrow_{V_{FG}} L(F^* \circ G^*) \) where \( V_{FG} \) is a iteration-specific metric relation such that for all \( (s_k, s_k^*) \in V_{FG} \), \( u(s_k, s_k^*) \leq \delta_{FG}(k) \) where \( \delta_{FG}(k) = O(t(k)) \).

Figure 16 describes the objective in the proof of compositions of models that are valid under equivalence relations.

---

34 The metric \( u(s_k, s_k^*) \) and the bound on \( \delta(k) \), \( t(k) \), are the same for \( L(F) \), \( L(G) \), and \( L(F \circ G) \).

35 Note that \( \delta(k) = O(t(k)) \) means for sufficiently large \( k \), \( \delta(k) \leq c(t(k)) \) where \( c \) is a constant.
The objective of the theorem below is to show that the set of models valid under equivalence relations is closed under composition.

**Theorem 5.1.** Given \( L(F) \Rightarrow_{V_F} L(F^*) \) and \( L(G) \Rightarrow_{V_G} L(G^*) \) where \( V_F \) and \( V_G \) are equivalence relations, \( L(F \circ G) \Rightarrow_{V_{F \circ G}} L(F^* \circ G^*) \) where \( V_{F \circ G} \) is an equivalence relation.

**Proof.** If \( R \) is an equivalence relation then \((a,b) \in R \) and \((b,c) \in R \implies (a,c) \in R\). Since \( L(F) \Rightarrow_{V_F} L(F^*) \) and \( L(G) \Rightarrow_{V_G} L(G^*) \) where \( V_F \) and \( V_G \) are equivalence relations, then \((\bar{s}, \bar{s}') \in V_F, V_G \) and \((\bar{s}', \bar{s}^*) \in V^* \implies (\bar{s}, \bar{s}^*) \in V_{F \circ G}\).

Therefore, there exists \( V_{F \circ G} \) such that \( L(F \circ G) \Rightarrow_{V_{F \circ G}} L(F^* \circ G^*) \). ■

The objective of the next set of theorems is to show that the set of models valid under iteration-specific metric relations is not closed under composition. Compositions of models valid under iteration-specific metric relations are valid only for limited classes of models. First, the general case is proven. Next, compositions of several sub-classes of models are shown to be valid under iteration-specific metric relations here, others are possible.

Figure 17 is a notional example of validity under metric for an affine model.

```
Figure 17. Validity under metric for affine model
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Consider as an example, \( F^* = A \bar{x} \), \( G^* = A^{-1} \bar{x} \), \( F = PA \bar{x} \), \( G = A^{-1} \bar{x} \), where \( P \) is a permutation matrix. Here \( L(F^* \circ G^*) \) cycles between \( \bar{s}_0^* \) and \( \bar{s}_1^* \). Now select \( \bar{s}_0^* \) such that \( \bar{e}_{\text{max}} = \max \{ F(\bar{s}_0^*) - F^*(\bar{s}_0^*) \} \). It is clear to see that

\[
\bar{E}_k = \bar{s}_k - \bar{s}_k^*
\]

\[
= F(\bar{s}_{k-1}) - F^*(\bar{s}_{k-1}^*)
\]

\[
= F(\bar{s}_{k-1} + \bar{E}_{k-1}) - F^*(\bar{s}_{k-1}^*)
\]

\[
= A(\bar{s}_{k-1} + \bar{E}_{k-1}) + \bar{c} - F^*(\bar{s}_{k-1}^*)
\]

\[
= A\bar{s}_{k-1}^* + A\bar{E}_{k-1} + \bar{c} - F^*(\bar{s}_{k-1}^*)
\]

\[
= F(\bar{s}_{k-1}) - F^*(\bar{s}_{k-1}) + A\bar{E}_{k-1}
\]

\[
\leq \bar{e}_{\text{max}} + A\bar{E}_{k-1}
\]

\[
= \bar{e}_{\text{max}} + A\bar{e}_{\text{max}} + A^2\bar{e}_{\text{max}} + \cdots + A^k\bar{E}_0
\]

\[
= (I + A + A^2 + \cdots + A^{k-1})\bar{e}_{\text{max}} + A^k\bar{E}_0
\]

Therefore, \( \| \bar{E}_k \| \leq (I + A + A^2 + \cdots + A^{k-1})\bar{e}_{\text{max}} + A^k\bar{E}_0 \leq \delta_F \).

If \( \bar{e}_{\text{max}} \) is defined such that \( \bar{E}_0 \leq \bar{e}_{\text{max}} \) then \( \| \bar{E}_k \| \leq (I + A + A^2 + \cdots + A^{k-1})\bar{e}_{\text{max}} \leq \delta_F \).

Two cases of metric relations are developed. First, the step metric requires that at each model execution the assignment \( \bar{s}_{k-1}^* = \bar{s}_{k-1} \) is made. This assignment results in \( \bar{E}_{k-1} = \bar{s}_{k-1} - \bar{s}_{k-1}^* = \bar{0} \). In the trajectory metric, no such assignment is required. Here \( \| \bar{E}_{k-1} \| \geq 0 \).

**Theorem 5.2.** Given affine model \( F = A \bar{x} + \bar{c} \) and model \( G \), such that \( L(F) \Rightarrow_{V_F} L(F^*) \) and \( L(G) \Rightarrow_{V_G} L(G^*) \) where \( V_F \) and \( V_G \) are step metric relations such that for all, \( (\bar{s}_{k-1}, \bar{s}_{k-1}^*) \in V_F \), \( \| \bar{s}_k - \bar{s}_k^* \| \leq \delta_F \), and for all
\[(\bar{s}_{k-1}, \bar{s}^*_k) \in V_G, \|\bar{s}_k - \bar{s}^*_k\| \leq \delta_G. \quad L(F \circ G) \Rightarrow_{V_{FG}} L(F^* \circ G^*) \text{ where } V_{FG} \text{ is an step metric relation such that for all } (\bar{s}_{k-1}, \bar{s}^*_k) \in V_{FG}, \|\bar{s}_k - \bar{s}^*_k\| \leq \delta_{FG}.

Here \(u(\bar{s}_k, \bar{s}^*_k) = \|\bar{s}_k - \bar{s}^*_k\|\).

**Proof.** Let \(\bar{E}_k\) be error after step \(k\). Assign \(\bar{s}^*_{k-1} = \bar{s}_{k-1}\). Then \(\bar{E}_{k-1} = \bar{s}_{k-1} - \bar{s}^*_{k-1} = 0\). For \(F\), \(\bar{E}_k = \bar{s}_k - \bar{s}^*_k = \bar{e}_k\). Now if \(\forall k, \|\bar{E}_k\| \leq \delta_F\), then \(\|\bar{e}_{max}\| \leq \delta_F\). Likewise, for \(G\), \(\bar{E}_k = \bar{s}_k - \bar{s}^*_k = \bar{y}_k\). Now if \(\forall k, \|\bar{E}_k\| \leq \delta_G\), then \(\|\bar{y}_{max}\| \leq \delta_G\). So for \(F \circ G\),

\[
\bar{E}_k = \bar{s}_k - \bar{s}^*_k
= F(G(\bar{s}_{k-1})) - F^*(G^*(\bar{s}_{k-1}))
= F(\bar{s}'_{k-1} + \bar{y}_k) - F^*(\bar{s}'_{k-1})
= A(\bar{s}'_{k-1} + \bar{y}_k) + \bar{c} - F^*(\bar{s}'_{k-1})
= A\bar{s}'_{k-1} + A\bar{y}_k + \bar{c} - F^*(\bar{s}'_{k-1})
= (A\bar{s}'_{k-1} + \bar{c}) - F^*(\bar{s}'_{k-1}) + A\bar{y}_k
= F(\bar{s}'_{k-1}) - F^*(\bar{s}'_{k-1}) + A\bar{y}_k
= \bar{e}_k + A\bar{y}_k
\]

\[
\|\bar{E}_k\| = \|\bar{e}_k + A\bar{y}_k\|
\leq \|\bar{e}_k\| + \|A\bar{y}_k\|
\leq \|\bar{e}_{max}\| + constant \cdot \|\bar{y}_{max}\|
\leq \delta_F + constant \cdot \delta_G
\]

Therefore \(V_{FG}\) is an step metric relation such that for all \((\bar{s}_{k-1}, \bar{s}^*_k) \in V_{FG}, \|\bar{s}_k - \bar{s}^*_k\| \leq \delta_{FG}\) and \(L(F \circ G) \Rightarrow_{V_{FG}} L(F^* \circ G^*)\). \(\blacksquare\)

**Corollary 5.3.** Given linear model \(F = A\bar{x}\) and model \(G\), such that \(L(F) \Rightarrow_{V_F} L(F^*)\) and \(L(G) \Rightarrow_{V_G} L(G^*)\) where \(V_F\) and \(V_G\) are step metric
relations such that for all, \((\bar{s}_{k-1}, \bar{s}_{k-1}^*) \in V_F, \|\bar{s}_k - \bar{s}_k^*\| \leq \delta_F\), and for all \((\bar{s}_{k-1}, \bar{s}_{k-1}^*) \in V_G, \|\bar{s}_k - \bar{s}_k^*\| \leq \delta_G\). Let \(L(F \circ G) \Rightarrow_{V_F \circ G} L(F^* \circ G^*)\) where \(V_F \circ G\) is an step metric relation such that for all \((\bar{s}_{k-1}, \bar{s}_{k-1}^*) \in V_F \circ G, \|\bar{s}_k - \bar{s}_k^*\| \leq \delta_{F \circ G}.

Here \(u(\bar{s}_k, \bar{s}_k^*) = \|\bar{s}_k - \bar{s}_k^*\|\).

**Proof.** A linear model is a special case of the affine model class. The result follows directly from the proof for Theorem 5.2.

**Theorem 5.4.** Given algebraic model \(F\) and model \(G\), such that \(L(F) \Rightarrow_{V_F} L(F^*)\) and \(L(G) \Rightarrow_{V_G} L(G^*)\) where \(V_F\) and \(V_G\) are step metric relations such that for all, \((\bar{s}_{k-1}, \bar{s}_{k-1}^*) \in V_F, \|\bar{s}_k - \bar{s}_k^*\| \leq \delta_F\), and for all \((\bar{s}_{k-1}, \bar{s}_{k-1}^*) \in V_G, \|\bar{s}_k - \bar{s}_k^*\| \leq \delta_G\). There is no \(V_F \circ G\) such that \(L(F \circ G) \Rightarrow_{V_F \circ G} L(F^* \circ G^*)\) where \(V_F \circ G\) is an step metric relation such that for all \((\bar{s}_{k-1}, \bar{s}_{k-1}^*) \in V_F \circ G, \|\bar{s}_k - \bar{s}_k^*\| \leq \delta_{F \circ G}.

Here \(u(\bar{s}_k, \bar{s}_k^*) = \|\bar{s}_k - \bar{s}_k^*\|\).

**Proof.** The proof will be by counter-example. Let \(E_k\) be error after step \(k\).

Assign \(\bar{s}_k = \bar{s}_{k-1}\). Then \(E_k = \bar{s}_{k-1} - \bar{s}_{k-1}^* = 0\). For \(F\), \(E_k = \bar{s}_k - \bar{s}_k^* = \bar{s}_k\). Now if \(\forall k, \|E_k\| \leq \delta_F\), then \(\|\bar{s}_k\| \leq \delta_F\). Likewise, for \(G\), \(E_k = \bar{s}_k - \bar{s}_k^* = \bar{G}_k\). Now if \(\forall k, \|E_k\| \leq \delta_G\), then \(\|\bar{G}_k\| \leq \delta_G\). So for \(F \circ G\),

\[
E_k = \bar{s}_k - \bar{s}_k^* = F(G(\bar{s}_{k-1})) - F^*(G^*(\bar{s}_{k-1})) = F(\bar{s}_{k-1} + \bar{G}_k) - F^*(\bar{s}_{k-1}^*)
\]

Let \(F^* = s^2, F = s^2 + \varepsilon, G^* = s, G = s + \gamma\). Then
\[ E_k = s_k - s_k^* \]
\[ = F(G(s_{k-1}))-F^*(G^*(s_{k-1})) \]
\[ = F(s_{k-1} + \gamma) - F^*(s_{k-1}) \]
\[ = (s_{k-1} + \gamma)^2 + \varepsilon - F^*(s_{k-1}) \]
\[ = (s_{k-1}^2 + 2s_{k-1}\gamma + \gamma^2) + \varepsilon - F^*(s_{k-1}) \]
\[ = s_{k-1}^2 + \varepsilon + 2s_{k-1}\gamma + \gamma^2 - F^*(s_{k-1}) \]
\[ = F(s_{k-1}) - F^*(s_{k-1}) + 2s_{k-1}\gamma + \gamma^2 \]
\[ = \varepsilon + 2s_{k-1}\gamma + \gamma^2 \]

Since \( E_k = s_k - s_k^* \) is a function of \( s_{k-1} \) then there is no \( V_{F\circ G} \) such that
\[ L(F \circ G) \Rightarrow_{V_{F\circ G}} L(F^* \circ G^*) \] where \( V_{F\circ G} \) is an step metric relation such that for all
\[ (\tilde{s}_{k-1}, \tilde{s}_{k-1}^*) \in V_{F\circ G}, \| \tilde{s}_k - \tilde{s}_k^* \| \leq \delta_{F\circ G}. \]

**Corollary 5.5.** Given elementary or computable model \( F \) and model \( G \), such that \( L(F) \Rightarrow_{\tilde{V}_F} L(F^*) \) and \( L(G) \Rightarrow_{\tilde{V}_G} L(G^*) \) where \( V_F \) and \( V_G \) are step metric relations such that for all, \( (\tilde{s}_{k-1}, \tilde{s}_{k-1}^*) \in V_F \), \( \| \tilde{s}_k - \tilde{s}_k^* \| \leq \delta_F \), and for all \( (\tilde{s}_{k-1}, \tilde{s}_{k-1}^*) \in V_G \), \( \| \tilde{s}_k - \tilde{s}_k^* \| \leq \delta_G \). There is no \( V_{F\circ G} \) such that
\[ L(F \circ G) \Rightarrow_{V_{F\circ G}} L(F^* \circ G^*) \] where \( V_{F\circ G} \) is an step metric relation such that for all
\[ (\tilde{s}_{k-1}, \tilde{s}_{k-1}^*) \in V_{F\circ G}, \| \tilde{s}_k - \tilde{s}_k^* \| \leq \delta_{F\circ G}. \]

**Proof.** A algebraic model is a special case of the elementary and computable model classes. It follows that a counter-example for the algebraic case also provides a counter-example for the elementary and computable cases. □

**Theorem 5.6.** Given linear models \( F = Ax \) and \( G = Bx \), \( L(F) \Rightarrow_{\tilde{V}_F} L(F^*) \) and \( L(G) \Rightarrow_{\tilde{V}_G} L(G^*) \) where \( V_F \) and \( V_G \) are iteration-specific trajectory metric relations defined by

1. For all, \( (\tilde{s}, \tilde{s}^*) \in V_F, u(\tilde{s}_k, \tilde{s}_k^*) \leq \delta_F(k) \) where \( \delta_F(k) = O(t(k)) \), and
2. For all \((\bar{s}, \bar{s}^*) \in V_G\), \(u(\bar{s}_k, \bar{s}_k^*) \leq \delta_G(k)\) where \(\delta_G(k) = O(t(k))\).

There is no \(V_{F \circ G}\) such that \(L(F \circ G) \Rightarrow_{v_{x_0}} L(F^* \circ G^*)\) where \(V_{F \circ G}\) is an iteration-specific trajectory metric relation such that for all \((\bar{s}, \bar{s}^*) \in V_{F \circ G}\), \(u(\bar{s}_k, \bar{s}_k^*) \leq \delta_{F \circ G}(k)\) where \(\delta_{F \circ G}(k) = O(t(k))\).

Proof. The proof will be by counter-example. Let \(\bar{E}_k\) \(36\) be error after step \(k\). For \(F\),

\[
\bar{E}_k = \bar{s}_k - \bar{s}_k^*
= F(\bar{s}_{k-1}) - F^*(\bar{s}_{k-1}^*)
= F(\bar{s}_{k-1} + \bar{E}_{k-1}) - F^*(\bar{s}_{k-1}^*)
= A(\bar{s}_{k-1} + \bar{E}_{k-1}) - F^*(\bar{s}_{k-1}^*)
= A\bar{s}_{k-1} + A\bar{E}_{k-1} - F^*(\bar{s}_{k-1}^*)
= A\bar{s}_{k-1} - F^*(\bar{s}_{k-1}^*) + A\bar{E}_{k-1}
= F(\bar{s}_{k-1}^*) - F^*(\bar{s}_{k-1}^*) + A\bar{E}_{k-1}
= \bar{e}_k + A\bar{E}_{k-1}
= \bar{e}_k + A\bar{e}_{k-1} + A^2\bar{e}_{k-2} + \cdots + A^i\bar{E}_0
\]

Likewise, for \(G\)

\[
\bar{E}_k = \bar{y}_k + B\bar{E}_{k-1}
= \bar{y}_k + B\bar{y}_{k-1} + B^2\bar{y}_{k-2} + \cdots + B^i\bar{E}_0
\]

Then for \(F \circ G\),

---

\(^{36}\) Note that model error \(\bar{e}_k\) is a function of \(\bar{s}_{k-1}^*\), \(\bar{e}_k = F(\bar{s}_{k-1}^*) - F^*(\bar{s}_{k-1}^*)\). Model error reflects the error generated by evaluating the model at \(\bar{s}_{k-1}^*\). \(A\bar{E}_{k-1}\) is the accumulated error.
\[E_k = \bar{\epsilon}_k + A\bar{\gamma}_k + AB\bar{E}_{k-1}\]
\[= \bar{\epsilon}_k + A\bar{\gamma}_k + AB(\bar{\epsilon}_{k-1} + A\bar{\gamma}_{k-1})\]
\[+ (AB)^2(\bar{\epsilon}_{k-2} + A\bar{\gamma}_{k-2}) + \cdots\]
\[+ (AB)^{t+1}(\bar{\epsilon}_1 + A\bar{\gamma}_1) + (AB)^{t} \bar{E}_0\]

Let \(F = A\bar{x}\), \(G = B\bar{x}\), \(F^* = A\bar{x}\), and \(G^* = B\bar{x}\) where
\[A = \begin{bmatrix} 0.9 & 1 \\ 0 & 0.9 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 0.9 & 0 \\ 1 & 0.9 \end{bmatrix}.

Note that \(F = F^* = A\bar{x}\) and \(G = G^* = B\bar{x}\) so for all \(k\), model error,
\[\bar{\epsilon}_k = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \text{and} \quad \bar{\gamma}_k = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.

Therefore, for \(F = A\bar{x}\),
\[
\|E_k\| = \|\bar{\epsilon}_k - \bar{\gamma}^*\| \\
= \|\bar{\epsilon}_k + A\bar{E}_{k-1}\| \\
= \|\bar{\epsilon}_k + A\bar{\epsilon}_{k-1} + A^2\bar{\epsilon}_{k-2} + \cdots + A^t\bar{E}_0\| \\
\leq \|A^t\| \cdot \|\bar{E}_0\|
\]

And, for \(G = B\bar{x}\),
\[
\|E_k\| = \|\bar{\gamma}_k - \bar{\epsilon}^*\| \\
= \|\bar{\gamma}_k + B\bar{E}_{k-1}\| \\
= \|\bar{\gamma}_k + B\bar{\gamma}_{k-1} + B^2\bar{\gamma}_{k-2} + \cdots + B^t\bar{E}_0\| \\
= \|B^t\bar{E}_0\| \\
\leq \|B^t\| \cdot \|\bar{E}_0\|
\]
Since the eigenvalues of \( A = \begin{bmatrix} 0.9 & 1 \\ 0 & 0.9 \end{bmatrix} \) are \( \lambda = 0.9 \), and \( \lambda = 0.9 \), then the spectral radius\(^{37}\) of \( A \), \( \rho(A) < 1 \). Likewise, since the eigenvalues of \( B = \begin{bmatrix} 0.9 & 0 \\ 1 & 0.9 \end{bmatrix} \) are \( \lambda = 0.9 \), and \( \lambda = 0.9 \), then the spectral radius of \( B \), \( \rho(B) < 1 \).

Note that \( \|A\|_2 = 1.5296 \) and \( \|B\|_2 = 1.5296 \). Since \( A^k \tilde{x} \) converges to \( 0 \) for every \( \tilde{x} \) if and only if \( \rho(A) < 1 \), then for \( F = A \tilde{x} \), \( \|E_k\| = \|A^k \tilde{E}_0\| \) tends to zero. Likewise, for \( G = B \tilde{x} \), \( \|E_k\| = \|B^k \tilde{E}_0\| \) tends to zero. Therefore, for all, \( (\tilde{s}, \tilde{s}^*) \in V_F \), \( u(\tilde{s}_k, \tilde{s}_k^*) \leq \delta_F(k) \) where \( \delta_F(k) = O(1) \) and for all \( (\tilde{s}, \tilde{s}^*) \in V_G \), \( u(\tilde{s}_k, \tilde{s}_k^*) \leq \delta_G(k) \) where \( \delta_G(k) = O(1) \). For example, let \( \tilde{E}_0 = \begin{bmatrix} 0 \\ 0.01 \end{bmatrix} \), then

\[
A^{1000} \tilde{E}_0 = 1 \times 10^{-44} \begin{bmatrix} 0.2951 \\ 0.0027 \end{bmatrix} \quad \text{and} \quad B^{1000} \tilde{E}_0 = 1 \times 10^{-47} \begin{bmatrix} 0 \\ 0.1748 \end{bmatrix}.
\]

Now for \( F \circ G \),

\[
\|\tilde{E}_k\| = \|\tilde{s}_k - \tilde{s}_k^*\| = \|\tilde{e}_k + A\tilde{y}_k + AB\tilde{E}_{k-1}\|
\]

\[
\leq \|\tilde{e}_k + A\tilde{y}_k + AB(\tilde{e}_{k-1} + A\tilde{y}_{k-1}) + (AB)^2(\tilde{e}_{k-2} + A\tilde{y}_{k-2}) + \ldots + (AB)^{k-1}(\tilde{e}_1 + A\tilde{y}_1) + (AB)^k \tilde{E}_0\|
\]

\[
\leq \|AB\|^k \|\tilde{E}_0\|
\]

But, \( AB = \begin{bmatrix} 0.9 & 1 \\ 0 & 0.9 \end{bmatrix} \begin{bmatrix} 0.9 & 0 \\ 1 & 0.9 \end{bmatrix} = \begin{bmatrix} 1.81 & 0.9 \\ 0.9 & 0.81 \end{bmatrix} \). Since the eigenvalues of \( AB \)

---

\(^{37}\)The spectrum of a matrix \( A \) is the set of all eigenvalues \( \lambda(A) \). The spectral radius of a matrix \( A \) is the maximum modulus of the eigenvalues of the matrix \( \rho(A) = \max \{ |\lambda| : \lambda \in \lambda(A) \} \).
are \( \lambda = 2.3396 \), and \( \lambda = 0.2804 \), then the spectral radius of \( AB \), \( \rho(AB) > 1 \).

Note that \( \|AB\|_2 = 2.3396 \). Since \( \|AB\|_{\infty} \) tends to infinity for some \( \bar{x} \) if and only if \( \rho(AB) > 1 \), then for \( F \circ G \), \( \|E_0\| = \|(AB)^k E_0\| \) tends to infinity for some \( E_0 \). For example, let \( E_0 = \begin{bmatrix} 0 \\ 0.01 \end{bmatrix} \), then

\[
(AB)^{10} E_0 = \begin{bmatrix} 3650 & 2174 \\ 2174 & 1264 \end{bmatrix} \begin{bmatrix} 0 \\ 0.01 \end{bmatrix} = \begin{bmatrix} 21.4 \\ 12.6 \end{bmatrix}
\]

\[
(AB)^{11} E_0 = \begin{bmatrix} 8538 & 5024 \\ 5024 & 2956 \end{bmatrix} \begin{bmatrix} 0 \\ 0.01 \end{bmatrix} = \begin{bmatrix} 50.2 \\ 29.6 \end{bmatrix}
\]

Therefore, there is no \( V_{F \circ G} \) such that \( L(F \circ G) \Rightarrow_{V_{F \circ G}} L(F^* \circ G^*) \) where \( V_{F \circ G} \) is an iteration-specific trajectory metric relation such that for all \( (\bar{s}, \bar{s}^*) \in V_{F \circ G} \), \( u(\bar{s}_k, \bar{s}_k^*) \leq \delta_{F \circ G}(k) \) where \( \delta_{F \circ G}(k) = O(t(k)) \).

**Corollary 5.7.** Given affine, algebraic, elementary, or computable models \( F \) and \( G \), \( L(F) \Rightarrow_{V_F} L(F^*) \) and \( L(G) \Rightarrow_{V_G} L(G^*) \) where \( V_F \) and \( V_G \) are iteration-specific trajectory metric relations defined by

1. For all \( (\bar{s}, \bar{s}^*) \in V_F \), \( u(\bar{s}_k, \bar{s}_k^*) \leq \delta_F(k) \) where \( \delta_F(k) = O(t(k)) \), and

2. For all \( (\bar{s}, \bar{s}^*) \in V_G \), \( u(\bar{s}_k, \bar{s}_k^*) \leq \delta_G(k) \) where \( \delta_G(k) = O(t(k)) \).

There is no \( V_{F \circ G} \) such that \( L(F \circ G) \Rightarrow_{V_{F \circ G}} L(F^* \circ G^*) \) where \( V_{F \circ G} \) is an iteration-specific trajectory metric relation such that for all \( (\bar{s}, \bar{s}^*) \in V_{F \circ G} \), \( u(\bar{s}_k, \bar{s}_k^*) \leq \delta_{F \circ G}(k) \) where \( \delta_{F \circ G}(k) = O(t(k)) \).

**Proof.** A linear model is a special case of the affine, algebraic, elementary, and computable model classes. It follows that a counter-example for the linear case also provides a counter-example for the affine, algebraic, elementary, and computable cases. ■
**Theorem 5.8.** Given affine models \( F = A\bar{x} + \bar{c} \) and \( G = B\bar{x} + d \), \( A \) and \( B \) are square and where \( \|A\| < 1 \) and \( \|B\| < 1 \), \( L(F) \Rightarrow_{\nu_f} L(F^*) \) and \( L(G) \Rightarrow_{\nu_g} L(G^*) \) where \( V_F \) and \( V_G \) are iteration-specific trajectory metric relations defined by

1. For all \( (\bar{s}, \bar{s}^*) \in V_F \), \( \|\bar{s}_k - \bar{s}^*_k\| \leq \delta_F(k) \leq \delta_F \) where \( \delta_F(k) = O(1) \), and

2. For all \( (\bar{s}, \bar{s}^*) \in V_G \), \( \|\bar{s}_k - \bar{s}^*_k\| \leq \delta_G(k) \leq \delta_G \) where \( \delta_G(k) = O(1) \).

\[ L(F \circ G) \Rightarrow_{\nu_{F \circ G}} L(F^* \circ G^*) \] where \( V_{F \circ G} \) is an iteration-specific trajectory metric relation such that for all \( (\bar{s}, \bar{s}^*) \in V_{F \circ G} \), \( \|\bar{s}_k - \bar{s}^*_k\| \leq \delta_{F \circ G}(k) \leq \delta_{F \circ G} \)

\[
\delta_{F \circ G}(k) = \left\| \sum_{i=0}^{k-1} (AB)^i (\bar{e} + A\bar{y}) + (AB)^k \bar{E}_0 \right\| \leq \delta_{F \circ G}
\]

where \( \delta_{F \circ G}(k) = O(1) \).

**Proof.** Let \( \bar{E}_k \) be error after step \( k \). For \( F \),

\[
\bar{E}_k = \bar{s}_k - \bar{s}^*_k
\]

\[
= F(\bar{s}_{k-1}) - F^*(\bar{s}^*_{k-1})
\]

\[
= F(\bar{s}^*_{k-1} + \bar{E}_{k-1}) - F^*(\bar{s}^*_{k-1})
\]

\[
= A(\bar{s}^*_{k-1} + \bar{E}_{k-1}) + \bar{c} - F^*(\bar{s}^*_{k-1})
\]

\[
= A\bar{s}^*_{k-1} + A\bar{E}_{k-1} + \bar{c} - F^*(\bar{s}^*_{k-1})
\]

\[
= (A\bar{s}^*_{k-1} + \bar{c}) - F^*(\bar{s}^*_{k-1}) + A\bar{E}_{k-1}
\]

\[
= F(\bar{s}^*_{k-1}) - F^*(\bar{s}^*_{k-1}) + A\bar{E}_{k-1}
\]

\[
= \bar{e}_k + A\bar{E}_{k-1}
\]

\[
= \bar{e}_k + A\bar{e}_{k-1} + A^2\bar{e}_{k-2} + \cdots + A^k\bar{E}_0
\]

\[38 \text{ The models } F:S \rightarrow S \quad G:S \rightarrow S \text{ are state models. Here we provide proof for the inputless case only. A similar argument for the more general case allowing input requires the development of interfaces to insure that } A \text{ and } B \text{ are square.} \]
Consider as an example, $F^* = A\tilde{x}$, $G^* = A^{-1}\tilde{x}$, $F = PA\tilde{x}$, $G = A^{-1}\tilde{x}$. Here $L(F^* \circ G^*)$ cycles between $\tilde{x}_0^*$ and $\tilde{x}_1^*$. Now select $\tilde{x}_0^*$ such that $\tilde{\epsilon}_{\text{max}} = \max\{F(\tilde{x}_0^*) - F(\tilde{x}_0^*)\}$. It is clear to see that

$$\bar{E}_k = \tilde{\epsilon}_{\text{max}} + A\bar{E}_{k-1}$$

$$= \tilde{\epsilon}_{\text{max}} + A\tilde{\epsilon}_{\text{max}} + A^2\tilde{\epsilon}_{\text{max}} + \cdots + A^k\bar{E}_0$$

$$= (1 + A + A^2 + \cdots + A^{k-1})\tilde{\epsilon}_{\text{max}} + A^k\bar{E}_0$$

Therefore, $\|\bar{E}_k\| = \|(1 + A + A^2 + \cdots + A^k)\tilde{\epsilon}_{\text{max}}\| \leq \delta_f$.

If $\tilde{\epsilon}_{\text{max}}$ is defined such that $\bar{E}_0 \leq \tilde{\epsilon}_{\text{max}}$, then $\|\bar{E}_k\| = \|(1 + A + A^2 + \cdots + A^k)\tilde{\epsilon}_{\text{max}}\| \leq \delta_f$.

Likewise, for $G$

$$\bar{E}_k = \tilde{\gamma}_{\text{max}} + B\bar{E}_{k-1}$$

$$= \tilde{\gamma}_{\text{max}} + B\tilde{\gamma}_{\text{max}} + B^2\tilde{\gamma}_{\text{max}} + \cdots + B^k\bar{E}_0$$

$$= (1 + B + B^2 + \cdots + B^k)\tilde{\gamma}_{\text{max}} + B^k\bar{E}_0$$

Therefore, $\|\bar{E}_k\| = \|(1 + B + B^2 + \cdots + B^k)\tilde{\gamma}_{\text{max}} + B^k\bar{E}_0\| \leq \delta_f$. If $\tilde{\gamma}_{\text{max}}$ is defined such that $\bar{E}_0 \leq \tilde{\gamma}_{\text{max}}$, then $\|\bar{E}_k\| = \|(1 + B + B^2 + \cdots + B^k)\tilde{\gamma}_{\text{max}}\| \leq \delta_f$.

Then for $F \circ G$

$$\bar{E}_k = \bar{\epsilon}_k + A\tilde{\gamma}_k + AB(\bar{\epsilon}_{k-1} + A\tilde{\gamma}_{k-1})$$

$$+ (AB)^2(\bar{\epsilon}_{k-2} + A\tilde{\gamma}_{k-2}) + \cdots$$

$$+ (AB)^{k-1}(\bar{\epsilon}_1 + A\tilde{\gamma}_1) + \cdots + (AB)^k\bar{E}_0$$

$$= \tilde{\epsilon}_{\text{max}} + A\tilde{\gamma}_{\text{max}} + AB(\tilde{\epsilon}_{\text{max}} + A\tilde{\gamma}_{\text{max}})$$

$$+ (AB)^2(\tilde{\epsilon}_{\text{max}} + A\tilde{\gamma}_{\text{max}}) + \cdots$$

$$+ (AB)^{k-1}(\tilde{\epsilon}_{\text{max}} + A\tilde{\gamma}_{\text{max}}) + (AB)^k\bar{E}_0$$

$$\|\bar{E}_k\| = \|(1 + AB + (AB)^2 + \cdots + (AB)^{k-1})(\tilde{\epsilon}_{\text{max}} + A\tilde{\gamma}_{\text{max}}) + (AB)^k\bar{E}_0\|$$
It is well-known that $\rho(A) \leq \|A\|$. Since $\|A\| < 1$ then $\rho(A) < 1$. Likewise since $\|B\| < 1$ then $\rho(B) < 1$. Also $\|AB\| \leq \|A\| \cdot \|B\|$ therefore $\|AB\| < 1$ and $\rho(AB) < 1$.

Note that if $\rho(A) < 1$,

$$(I - A)^{-1} = I + A + A^2 + \cdots + A^k + A^{k+1} + \cdots$$

$$I + A + A^2 + \cdots + A^k = (I - A)^{-1} - A^{k+1}(I + A + A^2 + \cdots)$$

$$\|I + A + A^2 + \cdots + A^k\| = \|(I + A + A^2 + \cdots) - A^{k+1}(I + A + A^2 + \cdots)\|$$

$$\|I + A + A^2 + \cdots + A^k\| \leq \|(I + A + A^2 + \cdots)\|$$

Likewise if $\rho(AB) < 1$,

$$(I - AB)^{-1} = I + AB + (AB)^2 + \cdots + (AB)^k + (AB)^{k+1} + \cdots$$

$$I + AB + (AB)^2 + \cdots + (AB)^k = (I - AB)^{-1} - (AB)^{k+1}(I + AB + (AB)^2 + \cdots)$$

$$\|I + AB + (AB)^2 + \cdots + (AB)^k\| \leq \|I + AB + (AB)^2 + \cdots\|$$

$$-\|(AB)^{k+1}\| \cdot \|(I + AB + (AB)^2 + \cdots)\|$$

$$\leq \|I + AB + (AB)^2 + \cdots\| = \|(I - AB)^{-1}\|$$

Therefore,

$$\|E_A\| = \|\left(I + AB + (AB)^2 + \cdots + (AB)^{k-1}\right)(\bar{e}_{\max} + A \bar{y}_{\max}) + (AB)^k \bar{E}_0\|$$

$$\leq \left\|\left(I + AB + (AB)^2 + \cdots + (AB)^{k-1}\right)(\bar{e}_{\max} + A \bar{y}_{\max}) + (AB)^k \bar{E}_0\right\|$$

Since $\bar{E}_0 \leq \bar{e}_{\max}$ and $\bar{E}_0 \leq \bar{y}_{\max}$ then $\bar{E}_0 \leq \bar{e}_{\max} + A \bar{y}_{\max}$. Then,

$$\|E_A\| = \|\left(I + AB + (AB)^2 + \cdots + (AB)^{k-1}\right)(\bar{e}_{\max} + A \bar{y}_{\max}) + (AB)^k \bar{E}_0\|$$

$$\leq \left\|\left(I + AB + (AB)^2 + \cdots + (AB)^{k-1}\right)(\bar{e}_{\max} + A \bar{y}_{\max}) + (AB)^k \bar{E}_0\right\|$$

$$\leq \|c_1 \delta + c_2 \delta_0\|$$
Therefore, $L\left(F \circ G\right) \Rightarrow_{\nu_{FG}} L\left(F^* \circ G^*\right)$ where $V_{FG}$ is an iteration-specific trajectory metric relation such that for all $(\bar{s}, \bar{s}^*) \in V_{FG}$, $\left\|\bar{s}_k - \bar{s}_k^*\right\| \leq \delta_{FG}(k) \leq \delta_{FG}$. 

$$\delta_{FG}(k) = \left\| \sum_{i=0}^{k-1} (AB)^i (\bar{e} + A\bar{\gamma}) + (AB)^k \bar{E}_0 \right\| \leq \delta_{FG}$$

where $\delta_{FG}(k) = O(1)$. 

**Corollary 5.9.** Given linear models $F = Ax$ and $G = B\bar{x}$, where $A$ and $B$ are square and where $\|A\| < 1$ and $\|B\| < 1$, $L\left(F\right) \Rightarrow_{\nu_F} L\left(F^*\right)$ and $L\left(G\right) \Rightarrow_{\nu_G} L\left(G^*\right)$ where $V_F$ and $V_G$ are iteration-specific trajectory metric relations defined by

3. For all $(\bar{s}, \bar{s}^*) \in V_F$, $\left\|\bar{s}_k - \bar{s}_k^*\right\| \leq \delta_{F}(k) \leq \delta_{F}$ where $\delta_{F}(k) = O(1)$, and

4. For all $(\bar{s}, \bar{s}^*) \in V_G$, $\left\|\bar{s}_k - \bar{s}_k^*\right\| \leq \delta_{G}(k) \leq \delta_{G}$ where $\delta_{G}(k) = O(1)$.

$L\left(F \circ G\right) \Rightarrow_{\nu_{FG}} L\left(F^* \circ G^*\right)$ where $V_{FG}$ is an iteration-specific trajectory metric relation such that for all $(\bar{s}, \bar{s}^*) \in V_{FG}$, $\left\|\bar{s}_k - \bar{s}_k^*\right\| \leq \delta_{FG}(k) \leq \delta_{FG}$

$$\delta_{FG}(k) = \left\| \sum_{i=0}^{k-1} (AB)^i (\bar{e} + A\bar{\gamma}) + (AB)^k \bar{E}_0 \right\| \leq \delta_{FG}$$

where $\delta_{FG}(k) = O(1)$.

**Proof.** A linear model is a special case of the affine model class. The result follows directly from the proof for **Theorem 5.8.**

Current results from this work are tabulated in Table 3. A “Yes” in Table 3 indicates that a composition of valid models from the function class indicated by the column label is provably valid under the relation indicated by the row label, whereas “No” indicates that such preservation of validity cannot be proven, in general, for those classes. Additional function classes and relations will be identified as the theory is applied to develop component libraries for specific application domains.
<table>
<thead>
<tr>
<th>Relation</th>
<th>Model</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Affine (∥A∥ &lt; 1)</td>
<td>Affine</td>
<td>Algebraic</td>
<td>Elementary</td>
<td>Computable</td>
</tr>
<tr>
<td>Equivalence</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Step metric</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Trajectory metric</td>
<td>Yes</td>
<td>No</td>
<td>No\textsuperscript{30}</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

\begin{table}[H]
\centering
\begin{tabular}{|l|c|c|c|c|c|}
\hline
Relation & Model & Affine (∥A∥ < 1) & Affine & Algebraic & Elementary & Computable \\
\hline
Equivalence & Yes & Yes & Yes & Yes & Yes \\
Step metric & Yes & Yes & No & No & No \\
Trajectory metric & Yes & No & No\textsuperscript{30} & No & No \\
\hline
\end{tabular}
\end{table}

\textit{Table 3. Summary of composition validity results}

\textsuperscript{30} Conditions may exist for algebraic, elementary, or computable models not previously noted in the linear or affine classes of models which allow semantic composability for a conditional sub-class of models.
6 EXAMPLES

In this section, several examples are described and then used to illustrate many of the composability concepts introduced in Sections 3 through 5.

6.1 Mod counter example

In this section, several modulo counters are described. We begin by describing a mod2 counter with control.

6.1.1 MOD2 counter with control

As the first example, a model is developed for a MOD2 counter with control. The natural system that represents the MOD2 counter is shown in Figure 18. This diagram consists of the usual block diagram representation of a system.

![Figure 18. MOD2 counter with control](image)

The purpose of the system is to count the number of 1's present in the signal. The counter has value 0 or 1, which is displayed by a single binary digit on the face of the counter. The control has value ON or OFF and is used to either enable counting (ON) or disable counting (OFF). The reset has value RESET, to indicate the occurrence of a counter reset (1 → 0 transition), or RESET otherwise.

6.1.2 Model

The model for this system is a computable function $M : X \rightarrow Y$ where $X \subseteq S \times I$ and $Y \subseteq S \times O$. Figure 19 is a graphical representation of $M$.
Now let:

\[
\bar{s} = \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} \text{count} \\ \text{reset} \end{bmatrix}
\text{where } \bar{s} \in S = \left\{ \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right\},
\]

\[
\bar{i} = \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} = \begin{bmatrix} \text{control} \\ \text{signal} \end{bmatrix}
\text{where } \bar{i} \in I = \left\{ \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\},
\]

\[
\bar{o} = [o_1] = \begin{bmatrix} \text{display} \end{bmatrix}
\text{where } \bar{o} \in O = \{0, 1\},
\]

\[
\bar{x} = \begin{bmatrix} s_1 \\ s_2 \\ i_1 \\ i_2 \end{bmatrix}, \text{ and } \bar{y} = \begin{bmatrix} s'_1 \\ s'_2 \\ o_1 \end{bmatrix}.
\]

\(M\) is expressed in tabular form in Table 4.
Table 4. Function definition table for MOD2 counter

After iteration \( k \), the state and output models are given analytically by:

\[
\begin{bmatrix}
\text{count} \\
\text{reset}
\end{bmatrix}_k = M_S \begin{bmatrix}
\text{count} \\
\text{reset} \\
\text{control} \\
\text{signal}
\end{bmatrix}_{k-1}, \text{ where }
\]

\( \text{count}_k = \text{count}_{k-1} \land \neg \text{control}_{k-1} \lor \text{count}_{k-1} \land \neg \text{signal}_{k-1} \lor \neg \text{count}_{k-1} \land \text{control}_{k-1} \land \text{signal}_{k-1} \)

\( \text{reset}_k = \text{count}_{k-1} \land \text{control}_{k-1} \land \text{signal}_{k-1} \)

\[
\begin{bmatrix}
\text{display}_k
\end{bmatrix} = M_O \begin{bmatrix}
\text{count} \\
\text{reset} \\
\text{control} \\
\text{signal}
\end{bmatrix}_{k-1}, \text{ where } \text{display}_k = \text{count}_k
\]

and \( M \) is given analytically by:
where \( \wedge, \lor, \text{ and } (\overline{\cdot}) \) are the boolean operations of AND, OR, and NOT, respectively.

6.1.3 Labeled transition system

The labeled transition system corresponding to the MOD2 counter with control model is shown in Figure 20. In this figure, vertices indicate state and labels represent input in the format (control, signal).

For the MOD2 Counter model defined in Table 4 and having labeled transition system shown in Figure 20, suppose the model is to be simulated beginning in state 0. Then, for the input sequence \( \langle(0,0),(1,1),(1,0),(1,1)\rangle \), the resulting trajectory consists of an alternating
sequence of state values and input values and is given by 
\[ \left\{ \begin{array}{c} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{array} \right\}. \]

6.1.4 MOD4 counter with control

A MOD4 counter with control can be constructed by cascading two MOD2 counters together. The block diagram for this new system is shown in Figure 21. The purpose of the system is to count the number of 1's present in the signal. Each MOD2 counter has value 0 or 1, which is displayed by a single binary digit on the face of the counter. The combined effect is a two-digit binary display. The control has value ON or OFF and is used to either enable counting (ON) or disable counting (OFF). The reset has value RESET, to indicate the occurrence of a counter reset (1 \to 0 transition), or \text{RESET} otherwise.

![Diagram of MOD4 counter with control](image)

Figure 21. MOD4 counter with control

The composite model for the MOD4 counter is shown in Figure 22. This model \( M_{\text{composite}} \) consists of the composition of two component models, \( M_1 \) and \( M_2 \), where each is a MOD2 counter.
Figure 22. Model of MOD4 counter with control

Figure 23. Labeled transition system for MOD4 counter with control
In this diagram, the rightmost column of digits represents the state vector of \( M_1 \) and the leftmost column of digits represents the state vector of \( M_2 \).

### 6.2 First-order differential equation example

Consider the first-order ordinary differential equation

\[
\dot{x} + x = 0
\]

where \( x(t) \) is a function of time, and \( \dot{x} \) is the first derivative of \( x \) with respect to time

with boundary condition \( x(0) = 1 \).

It is well-known that the analytic solution \( x(t) \) to the general equation is \( x(t) = Ae^{-t} \). Evaluation at the boundary condition results in a solution of

\[
x(t) = e^{-t}
\]

Now consider a model based on the Runge-Kutta method. Consider a system of ordinary differential equations of the form

\[
y' = f(t, y)
\]

The Runge-Kutta method is implemented in the form

\[
y_{k+1} = y_k + \frac{1}{2}(c_1 + c_2)
\]

where

\[
c_1 = h_k \cdot f(t_k, y_k), \text{ and}
\]

\[
c_2 = h_k \cdot f(t_k + h_k, y_k + c_1).
\]
For this example, let \( h = 0.25 \). After iteration \( k \), the state model is given by:

\[
\begin{bmatrix}
  x \\
  t \\
  h_{k+1}
\end{bmatrix} = M_S \begin{bmatrix}
  x \\
  t \\
  h_k
\end{bmatrix},
\]

where

\[
x_{k+1} = x_k + \frac{1}{2}(h^2 - 2h)x_k
\]

\[
t_{k+1} = t_k + h
\]

Note that \( h \) is included in the state vector since \( h \) is often modified by the model during simulation. A comparison of the values generated by the analytic solution and the model are provide in Figure 24. Figure 25 shows the error calculated by the metric

\[
\overline{E}_k = |\hat{x}_k - \hat{s}_k|.
\]

In this example, the analytic solution represents the perfect model. This example, though simple, demonstrates that models defined as they are here are useful to represent both discrete event and continuous-time systems.

![Analytic solution vs model](image)

*Figure 24. Analytic solution vs model for Runge-Kutta model*
Figure 25. Error for Runge-Kutta model
7 COMPUTATIONAL COMPLEXITY OF COMPOSITION

The computational complexity of the problem of selecting a set of components that meet a set of objectives is examined. In earlier work, Page and Opper (Page and Opper, 1999) defined four variants of this component selection problem based on two forms of objectives decidability (bounded and unbounded) and two forms of composition (emergent and non-emergent). They gave a proof that the bounded non-emergent variant of the component selection problem is \textit{NP-complete}. In this paper, an additional form of composition (anti-emergent) is defined, leading to two additional variants of the problem. Then a general form of the component selection problem that subsumes all six variants is defined. The general component selection problem is proven to be \textit{NP-complete} even if the objectives met by a component or composition are known. Several related but different problems, including determining the objectives met by a component and determining the validity of a proposed composition are defined, and conjectures for their complexity are given.

Composability, considered as a process, has a number of computational problems embedded within it that are important enough to warrant formal study. Here one of the most basic questions of composability is examined: given a set of components and a set of objectives, how difficult is it to select a subset of the components that meet the objectives? The computational complexity of the component selection problem is established, following and generalizing earlier work done on the problem by Page and Opper (Page and Opper, 1999). Several other computational problems within composability are defined and their complexity is considered.

This section begins with a brief background review of NP-completeness theory. Then the component selection problem is defined formally, first in several variants, then in a general form. Next, the computational complexity of the general component selection problem is established. Following that, several other problems inherent in composability are defined and conjectures given for their computational complexity.
7.1 Review of NP-completeness theory

NP-completeness theory is concerned with the computational complexity of decision problems (Garey and Johnson, 1979). A decision problem is defined in two parts. The first part is a formal specification of the information (such as sets, graphs, matrices, or numbers) that is the subject of the problem. The specification is given in precise yet general terms, for example, calling for "a graph of \( n \) vertices" rather than some specific graph. An instance of a decision problem is a specific set of information that complies with the information specification. The second part of a decision problem is a question, which can be answered "yes" or "no" (hence "decision problem"), about the properties of an instance. A solution to a decision problem is with respect to a specific instance; the solution is "yes" if the instance satisfies the question, "no" if it does not. For decision problem \( U \), the set \( Y_U \) is the set of all instances of \( U \) for which the solution is "yes". For an instance \( I \) of problem \( U \), \( I \in Y_U \) if and only if the solution to instance \( I \) is "yes".\(^{40}\)

In computational complexity theory, problems are categorized based on their upper bound on time \( O(f(n)) \), where \( n \) is the size of the instance. Those problems where \( f(n) \) is a polynomial function on \( n \) (e.g., \( f(n) = n^2 \)) on a deterministic computer belong to set \( P \). Problems for which the time function \( f(n) \) of the best known algorithm is an exponential function on \( n \) (e.g., \( f(n) = 2^n \)) belong to set \( NP \). (Problems in \( NP \) can be solved in polynomial time on a nondeterministic computer.) Though it remains unproven, it is widely assumed that \( P \neq NP \).\(^{41}\) \( NP\)-complete problems\(^{42}\) are those

\(^{40}\)Problems of other types, such as search or optimization, can generally be shown to be no easier than their corresponding decision problems, so proofs about the difficulty of the decision problems apply to the other types as a lower bound. Surprisingly, problems of the seemingly more difficult types can also be shown in many cases to be no harder than their corresponding decision problems (Preparata and Shamos, 1988).

\(^{41}\)Settling the question of whether \( P = NP \) has been called "the most important open question of either mathematics or computer science" (Homer and Selman, 2001).
problems in NP such that every instance of any NP-complete problem can be transformed into an equivalent instance of any other NP-complete problem by some process that requires \( O(f(n)) \) time, where \( f(n) \) is polynomial on \( n \).

Given a decision problem \( V \), \( V \) can be shown to be NP-complete using a two-step procedure. The first shows that the problem is in NP. The second shows that it is at least as difficult as other NP-complete problems. The steps together imply that a problem is NP-complete.

Show that \( V \) is in NP, by giving a polynomial time algorithm to check a solution for \( V \).

Transform a known NP-complete problem \( U \) to \( V \), as follows:

1. Define a transformation function \( h \) from an instance \( I \) of \( U \) to an instance \( h(I) \) of \( V \).43
2. Show that \( h \) requires polynomial time in \( n \), the size of \( I \).
3. Show that \( I \in Y_U \) if and only if \( h(I) \in Y_V \).

Showing a problem to be NP-complete has practical value. Once a problem is proven to be NP-complete it is known to be as hard as all other NP-complete problems. The failure, to date, to find a polynomial algorithm for any NP-complete problem suggests that finding one for the given problem may be problematic.

As mentioned earlier, to be NP-complete a decision problem must be in NP (step 1 in the basic proof procedure). NP-completeness theory may also provide information about

---

42 Because NP-complete is a set of problems it is perhaps more precise to say "problems in NP-complete" than "NP-complete problems". However, the latter formulation is ubiquitous and we follow the convention.

43 The transformation is more commonly known as \( f \), rather than \( h \) (Garey and Johnson. 1979). We use the latter to avoid name conflict with the time function in the conventional order notation \( O(f(n)) \).
decision problems that are not in \( NP \). Suppose decision problem \( W \) is not in \( NP \), but a
known \( NP\)-complete problem \( U \) can be transformed into \( W \) so that a solution to \( W \) is a
solution to \( U \) (step 2 in the basic proof procedure). Then \( W \) is at least as hard as the
\( NP\)-complete problems, and is not solvable in polynomial time unless \( P = NP \). Such
problems are referred to as \( NP\)-hard (Garey and Johnson. 1979).

NP-completeness theory includes the idea of oracles (also known as oracle Turing
machines or oracle functions) (Garey and Johnson. 1979). An oracle is essentially a
hypothetical or notional computational procedure that can perform an arbitrary
computation in one computational or time step. Oracles can be used to study separately
the computational complexity of problems that may have other computational problems
embedded within them or connected to them. For example, if problem \( U \) has problem
\( V \) embedded within it, then an oracle to solve \( V \) could be assumed and the complexity
of \( U \) studied. This allows the determination of how difficult \( U \) might be even if the
related problem \( V \) were solved.

7.2 Component selection problem definitions

In this section, the computational problem that is the focus of this paper, namely,
component selection, is defined. Informally, component selection is the problem of
selecting components to compose in order to meet a simulation's objectives. Somewhat
less informally, component selection is the problem of selecting from a repository
containing a given set of components a subset of those components to be composed such
that the composition will meet a given set of objectives. In this section the problem is
defined formally. In 1999, Page and Opper (Page and Opper, 1999) formally defined
four variants of the component selection problem based on two forms of objectives
decidability and two forms of composition (Page and Opper, 1999). First, those
definitions are examined and two additional variants of the problem, based on a third
form of composability, are defined. Then a general form of the component selection
problem that subsumes all six variants is defined.

44 They actually referred to the problem as “composability”, rather than “component selection”.
Because there are other computational problems inherent in composability, we use the latter term.
Let $O = \{o_1, o_2, \ldots, o_n\}$ be a set of objectives. Let $C = \{c_1, c_2, \ldots, c_m\}$ be a set of components. A simulation system $S$ is a subset of $C$, i.e., $S \subseteq C$. If $|S| > 1$ then $S$ is a composition. Let the symbol $\circ$ denote composition of components, e.g., $(c_j \circ c_k)$ is a composition of two components. Let $\Rightarrow$ denote satisfying an objective, i.e., $c_j \Rightarrow o_i$ means component $c_j$ satisfies objective $o_i$, and $c_j \nRightarrow o_i$ means that it does not. The $\Rightarrow$ and $\nRightarrow$ operators may also be applied to compositions and sets of objectives, e.g., $c_j \circ c_k \Rightarrow o_i$ and $S \nRightarrow O$ have the expected meanings. A simulation system $S \Rightarrow O$ if and only if $S \Rightarrow o_i$ for every $o_i \in O$.

In their definitions, Page and Opper (Page and Opper, 1999) first considered the decidability of objectives, defining two forms: bounded and unbounded (Page and Opper, 1999). If, for every objective $o_i$ in a given set of objectives $O$, it is possible to decide in polynomial time that $o_i$ is satisfied, then $O$ is bounded.

<table>
<thead>
<tr>
<th>Decidability of objective</th>
<th>Decidable in polynomial time</th>
<th>Decidable but not in polynomial time</th>
<th>Undecidable</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Page and Opper, 1999)</td>
<td>Bounded</td>
<td>Unbounded</td>
<td>Unbounded</td>
</tr>
<tr>
<td>Alternate</td>
<td>Decidable</td>
<td>Decidable</td>
<td>Undecidable</td>
</tr>
</tbody>
</table>

*Table 5. Alternate classes of objective decidability*

---

45 The notation introduced here follows that of Page and Opper (Page and Opper, 1999); for the most part it is the same, but there have been some changes and additions.

46 The symbol "⇒" normally means "logically implies", as in $A \Rightarrow B$. To avoid confusion, we will not use it in that sense here.
If $O$ contains at least one objective $o_i$ for which polynomial time is not sufficient to decide if $o_i$ is satisfied, then $O$ is unbounded. The question of whether a component halts is undecidable (Page and Opper, 1999) (Turing, 1937) and so is an example of the latter case. Note that the decidability of the objectives is defined independently of any particular component or composition.

These definitions are helpful and draw attention to the potential difficulty of deciding if an objective is satisfied. Two comments regarding them should be noted. First, as given the definitions refer to deciding in polynomial time if an objective is satisfied, but they do not state explicitly which problem parameter the decision time must be polynomial in; should it be polynomial in the length of the objective’s encoding, in the size of the objective set, or something else? The former is assumed. Second, it is possible to partition the objectives into three decidability classes, instead of two: decidable in polynomial time, decidable but not in polynomial time, and undecidable. Page and Opper (Page and Opper, 1999) group the second and third of these three into the “unbounded” class in their definitions; an alternate classification would be to group the first and second into “decidable”, leaving the third as “undecidable”. These classifications are summarized in Table 5. Though the alternate grouping has some uses, for the remainder of this paper the Page and Opper (Page and Opper, 1999) terms and definitions for objectives decidability are retained.

Page and Opper (Page and Opper, 1999) second considered forms of composition, also defining two forms: emergent and non-emergent (Page and Opper, 1999). The idea is that the objectives met by a composition of components may not necessarily be simply the union of the objectives met by the components individually. Suppose $c_j, c_k \in C$ are components and $o_i \in O$ is an objective. If $c_j \rightarrow o_i$ and $c_k \rightarrow o_i$ and $c_j \circ c_k \rightarrow o_i$, then the composition is non-emergent. If $c_j \rightarrow o_i$ and $c_k \rightarrow o_i$ but $c_j \circ c_k \rightarrow o_i$, then the composition is emergent. In the latter case, the components combine to satisfy some objective that neither satisfies separately.
Based on their two forms of objectives decidability (bounded and unbounded) and two forms of composition (emergent and non-emergent), Page and Opper defined four variants of the component selection problem: \textit{bounded emergent}, \textit{unbounded emergent}, \textit{bounded non-emergent}, and \textit{unbounded non-emergent} (Page and Opper, 1999).

We define an additional form of composition: \textit{anti-emergent}. If $c_j \Rightarrow o_i$ or $c_k \Rightarrow o_i$ but $c_j \circ c_k \not\Rightarrow o_i$, then the composition is anti-emergent. The idea is that the components in a composition could interfere with each other in such a way that the composition fails to satisfy objectives that the components satisfy separately. It is easy to imagine examples of anti-emergence; a terrain database component and an intervisibility determination component may separately satisfy objectives, but if they are based on different terrain database formats, their composition will likely be anti-emergent.

The three possible combinations already listed do not exhaust the possibilities. Table 6 lists all eight possible logical combinations of $c_j \Leftrightarrow o_i$, $c_k \Leftrightarrow o_i$, and $c_j \circ c_k \Leftrightarrow o_i$. Those due to (Page and Opper, 1999) are so noted. The combinations all fall into one of the three composition forms defined (non-emergent, emergent, anti-emergent).

The anti-emergent form of composition adds two new variants of the component selection problem to the four defined earlier: bounded anti-emergent and unbounded anti-emergent. The computational complexity of each problem variant could be studied separately. Indeed, Page and Opper considered one problem variant, giving a proof that the bounded non-emergent variant of the component selection problem is \textit{NP-complete} (Page and Opper, 1999). However, we prefer to separate the problem of determining which objectives a component or composition satisfies from the problem of selecting a set of components to meet the objectives; we will do so by assuming an oracle for the former problem. We also prefer to consider the component selection problem in general, rather than in terms of variants.
<table>
<thead>
<tr>
<th>Separately</th>
<th>Composed</th>
<th>Form of composition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_j \Rightarrow o_i$</td>
<td>$c_k \Rightarrow o_i$</td>
<td>$c_j \circ c_k \Rightarrow o_i$</td>
</tr>
<tr>
<td>$c_j \nRightarrow o_i$</td>
<td>$c_k \Rightarrow o_i$</td>
<td>$c_j \circ c_k \Rightarrow o_i$</td>
</tr>
<tr>
<td>$c_j \Rightarrow o_i$</td>
<td>$c_k \nRightarrow o_i$</td>
<td>$c_j \circ c_k \Rightarrow o_i$</td>
</tr>
<tr>
<td>$c_j \nRightarrow o_i$</td>
<td>$c_k \nRightarrow o_i$</td>
<td>$c_j \circ c_k \Rightarrow o_i$</td>
</tr>
<tr>
<td>$c_j \Rightarrow o_i$</td>
<td>$c_k \Rightarrow o_i$</td>
<td>$c_j \circ c_k \nRightarrow o_i$</td>
</tr>
<tr>
<td>$c_j \nRightarrow o_i$</td>
<td>$c_k \Rightarrow o_i$</td>
<td>$c_j \circ c_k \nRightarrow o_i$</td>
</tr>
<tr>
<td>$c_j \Rightarrow o_i$</td>
<td>$c_k \nRightarrow o_i$</td>
<td>$c_j \circ c_k \nRightarrow o_i$</td>
</tr>
<tr>
<td>$c_j \nRightarrow o_i$</td>
<td>$c_k \nRightarrow o_i$</td>
<td>$c_j \circ c_k \nRightarrow o_i$</td>
</tr>
</tbody>
</table>

Table 6. Forms of composition

The general component selection problem is defined as follows, following the problem definition conventions of NP-completeness theory (Garey and Johnson, 1979):
COMPONENT SELECTION

INSTANCE: Set $C = \{c_1, c_2, ..., c_n\}$ of components, set $O = \{o_1, o_2, ..., o_m\}$ of objectives, oracle function $\sigma: \text{power}(C) \rightarrow \text{power}(O)$, positive integer $K \leq |C|$.

QUESTION: Does $C$ contain a composition $S$ that satisfies $O$ of size $K$ or less, i.e., a subset $S \subseteq C$ with $|S| \leq K$ such that $O \subseteq \sigma(S)$?

In this problem definition, the purpose of $C$ as the set of components and $O$ as the set of objectives in the instance is straightforward. Oracle function $\sigma$ determines what objectives any component or composition of components satisfies. To understand the definition of $\sigma$, recall that the power set of a set is the set of all its subsets, so a function from the power set of $C$ (written $\text{power}(C)$) to the power set of $O$ (written $\text{power}(O)$) can specify for any subset of $C$, i.e., any component or composition of components, what subset of objectives from among $O$ that subset of $C$ satisfies. Because $\sigma$ is an oracle function, it can be assumed to answer the question in one step, allowing examination of the difficulty of component selection independent of the difficulty of objectives decidability. Integer $K$ in the instance is the maximum size of the composition allowed; the limit is required to make this a decision problem. The deceptively similar-sounding problem “What is the smallest subset of $C$ that satisfies $O$?” is an optimization problem and not within the bounds of NP-completeness theory. However, note that the decision problem subsumes the also similar-sounding existence problem; if the question is “Does there exist a subset of $C$ that satisfies $O$?” then setting $K$ to $|C|$ expresses that existence problem as a decision problem. Finally, note that the condition $O \subseteq \sigma(S)$ in the question could equivalently have been written using the notation introduced earlier as $S \Rightarrow O$; the former form was chosen to have the problem definition use only standard set notation.

The important thing to observe about this problem is that it subsumes all six variants of the component selection problem defined earlier. The emergent, non-emergent, and anti-emergent forms of composability are covered by specifying different mappings for
function \(\sigma\) in the instance. The bounded and unbounded forms of objectives decidability are covered by making \(\sigma\) an oracle function.

7.3 Computational complexity of component selection

In this section the computational complexity of the general component selection problem, defined in the previous section, is established. As mentioned, Page and Opper gave a proof that that one variant of the component selection problem is \(NP\)-complete (Page and Opper, 1999). It would be reasonable to assume that the general problem, which includes the variants, is also \(NP\)-complete (or \(NP\)-hard). As expected, it will be shown that the general problem, even given the oracle function to determine the objectives satisfied by a composition, remains \(NP\)-complete.

The proof of \(NP\)-completeness will use the \textit{MINIMUM COVER} problem, known to be \(NP\)-complete (Garey and Johnson, 1979) (Karp, 1972). That problem is defined as follows:

\textbf{MINIMUM COVER}\textsuperscript{47}

INSTANCE: Collection \(C\) of subsets of a finite set \(O\), positive integer \(K \leq |C|\).

QUESTION: Does \(C\) contain a cover for \(O\) of size \(K\) or less, i.e., a subset \(C' \subseteq C\) with \(|C'| \leq K\) such that every element of \(O\) belongs to at least member of \(C'\)?

The definition of \textit{COMPONENT SELECTION} is repeated here for convenience:

\textsuperscript{47} In (Garey and Johnson, 1979), the definition for \textit{MINIMUM COVER} uses \(S\) for the set to be covered, rather than \(O\). Because \(S\) is already used for the subset in \textit{COMPONENT SELECTION} and the set to be covered in \textit{MINIMUM COVER} corresponds to \(O\), not \(S\), in that problem, using \(O\) here makes the notation of the proof less confusing.
COMPONENT SELECTION

INSTANCE: Set $C = \{c_1, c_2, \ldots, c_n\}$ of components, set $O = \{o_1, o_2, \ldots, o_n\}$ of objectives, oracle function $\sigma : \text{power}(C) \to \text{power}(O)$, positive integer $K \leq |C|$.

QUESTION: Does $C$ contain a composition $S$ that satisfies $O$ of size $K$ or less, i.e., a subset $S \subseteq C$ with $|S| \leq K$ such that $O \subseteq \sigma(S)$?

We now establish the complexity of component selection.

Theorem. COMPONENT SELECTION is NP-complete.

Proof. By transformation from MINIMUM COVER. First it must be shown that COMPONENT SELECTION is in NP. Given a subset $S$ of $C$, determining if $O \subseteq \sigma(S)$ can be done by searching in $\sigma(S)$ for each element of $O$. A naïve algorithm to do so requires $O(mn)$ time (recall that computing $\sigma(S)$ requires only one step), which is clearly polynomial in the length of the instance, thus COMPONENT SELECTION is in NP.

The transformation function $h$ from any instance $I$ of MINIMUM COVER to an instance $h(I)$ of COMPONENT SELECTION is now defined. Because the instances of MINIMUM COVER and COMPONENT SELECTION have like-named items, subscripts will be used where needed to distinguish them, e.g., $C_{MC}$ is the set $C$ from instance $I$ of MINIMUM COVER, whereas $C_{CS}$ is the set $C$ from the instance $h(I)$ of COMPONENT SELECTION. The function $\sigma$ in $h(I)$ will be defined by generating its mapping as ordered pairs. The transformation $h$ is as follows:

For every $c_i = \{o_{i,1}, o_{i,2}, \ldots, o_{i,n_i}\} \in C_{MC}$, generate $c_i \in C_{CS}$, where the elements of $C_{MC}$ are sets and those of $C_{CS}$ are simply elements.
For every \( c_i = \{o_{i,1}, o_{i,2}, ..., o_{i,n}\} \in C_{MC} \), generate \((c_i, \{o_{i,1}, o_{i,2}, ..., o_{i,n}\})\) \( \in \sigma \), where the first element of each ordered pair is an element of \( C_{CS} \) and the second element is the corresponding element of \( C_{MC} \), i.e., a subset of \( O_{MC} \).

Copy \( O_{MC} \) to \( O_{CS} \).

Copy \( K_{MC} \) to \( K_{CS} \).

Step 1 requires time in \( O(m) \), step 2 in \( O(mn) \), step 3 in \( O(n) \), and step 4 in \( O(1) \), so the overall time complexity of transformation \( h \) is \( O(mn) \), which is polynomial in the length of the input.

It must now be shown that \( I \in Y_{MC} \) if and only if \( h(I) \in Y_{CS} \).

Only if. Assume \( I \in Y_{MC} \). Then there exists subset \( C' \subseteq C_{MC} \) such that \( |C'| \leq K_{MC} \) and \( O_{MC} \subseteq \bigcup_{c \in C'} c \). Let \( S = \{c_i \in C_{CS} \text{ such that } c_i \in C'\} \). Then \( \sigma(S) = \bigcup_{c \in C'} \sigma(c) = \bigcup_{c \in C'} c \) by \( h \).

Because \( O_{CS} = O_{MC} \) by \( h \) and \( O_{MC} \subseteq \bigcup_{c \in C'} c = \sigma(S) \), then \( O_{CS} \subseteq \sigma(S) \). Because \( K_{CS} = K_{MC} \) by \( h \) and \( |S| = |C'| \leq K_{MC} \), then \( |S| \leq K_{CS} \). Therefore \( h(I) \in Y_{CS} \).

If. Assume \( h(I) \in Y_{CS} \). Then there exists subset \( S \subseteq C_{CS} \) such that \( |S| \leq K_{CS} \) and \( O_{CS} \subseteq \sigma(S) \). Let \( C' = \{\sigma(c) \text{ such that } c \in S\} \). Then \( \bigcup_{c \in C'} c = \bigcup_{c \in C'} \sigma(c) = \sigma(S) \) by \( h \).

Because \( O_{MC} = O_{CS} \) by \( h \) and \( O_{CS} \subseteq \sigma(S) = \bigcup_{c \in C'} c \), then \( O_{MC} \subseteq \bigcup_{c \in C'} c \). Because \( K_{MC} = K_{CS} \) by \( h \) and \( |C'| = |S| \leq K_{CS} \), then \( |C'| \leq K_{MC} \). Therefore \( I \in Y_{MC} \).

Thus \( I \in Y_{MC} \) if and only if \( h(I) \in Y_{CS} \), and therefore \textit{COMPONENT SELECTION} is \textit{NP-complete}. \( \blacksquare \)

This result shows that general component selection problem is \textit{NP-complete} even if the objectives met by each component and composition are known. If the objectives met by the components and composition are not known and must be determined as part of
assembling a simulation system by composition, then the combined problem is at least as hard component selection, i.e., the combined problem of component selection and objectives determination is \( NP\text{-hard} \).

Figure 26 summarizes the complexity results of this paper. It shows the six variants of the component selection problem previously defined, the general component selection problem, and their computational complexities.

![Component selection problems and complexities](image)

Figure 26. Component selection problems and complexities

7.4 Other computational problems in composability

\( COMPONENT\ SELECTION \) is not the only computational problem inherent in composability. In this section several related but different problems are informally defined and conjectures for their complexity are discussed.

\( COMPONENT\ OBJECTIVES \). What objectives are satisfied by a component? This problem has already been mentioned in previous sections. Knowing the objectives met by a component is a necessary precursor to solving the component selection problem. To determine what objectives a component satisfies, the component itself must be analyzed;
that might be done by processing the component as programming language source code or by processing a meta-model specification of the component's semantics. Unfortunately, a procedure to determine in general which objectives a component satisfies may not be in $NP$ or even decidable, as observed by Page and Opper (Page and Opper, 1999), because one objective could be that the component halt for arbitrary input and the halting problem is undecidable (Hein, 2002) (Turing, 1937). Even the simpler-seeming problem of determining if a given query objective is satisfied by a component may be likewise undecidable for the same reason. However, it will be necessary to make determinations of the objectives satisfied by components for composability to be a practical reality. Such determinations will either have to be made by non-algorithmic methods and/or only for decidable objectives.

**COMPOSITION OBJECTIVES.** What objectives are satisfied by a composition? This problem was also mentioned previously, but was conflated with the component objectives problem. In fact, it may be separable. If the objectives met by each component in a composition are already known, and the form of composition is known, then that information could be used as the basis for an algorithm to determine what objectives are satisfied by a composition, or if a given query objective is satisfied by a composition. We conjecture that this question, given the information mentioned, is decidable.

**VALIDITY DETERMINATION.** Is a component valid? Here, validity is defined as a special case of quantifiable consistency between models, where one of the models is defined as "perfect" and the other may or may not be valid with respect to it. Using that definition, we conjecture that determining the validity of a model is decidable but $NP$-complete or $NP$-hard.

**COMPOSITION COMPLEXITY.** Given the computational complexities (time and/or space) of the components, what is the computational complexity of their composition? We conjecture that this problem is not only decidable, but in $P$, solvable by analysis of a formal specification of the component, such as programming language source code or formal semantic meta-model. However, a practical algorithm to make the determination could be difficult to develop.
7.5 Computational complexity summary

The computational complexity of the component selection problem was examined. Page and Opper defined four variants of the problem and gave a proof that the one variant is \textit{NP-complete}. Two additional variants of the problem were defined. Then a general form of the component selection problem that subsumes all six variants was given. The general component selection problem was proven to be \textit{NP-complete} even if the objectives met by a component or composition are available. Several other computational problems inherent in composability, including \textit{COMPONENT OBJECTIVES}, \textit{COMPOSITION OBJECTIVES}, \textit{VALIDITY DETERMINATION}, and \textit{COMPOSITION COMPLEXITY} were informally defined.

The main result, that component selection is \textit{NP-complete} even if the objectives satisfied by a component or composition are available, suggests the underlying computational complexity of composability. The component selection problem is conceptually simple compared to issues of actually implementing composability in either syntactic or semantic forms.

Of course, an \textit{NP-complete} problem is not unsolvable; rather, if a problem is \textit{NP-complete} there is no efficient (polynomial time) algorithm to solve it, unless $P = NP$. However, establishing a problem as \textit{NP-complete} does not free developers from the need to solve it in practical applications. \textit{NP-complete} problems are regularly solved by algorithms that are not efficient but have run-times that can be tolerated, or by heuristics that are not guaranteed to find an optimum solution but give a reasonable approximation for a useful range of cases. Clearly, if composability is to become a practical reality, the component selection problem will have to dealt with in some way.
8 SUMMARY OF RESULTS AND FUTURE WORK

The summary of results and future work are presented in this section. First, the summary of results and related publications are presented. Next future work is discussed.

8.1 Summary of results, contributions, and related publications

This section presents the summary of results and related publications. First, the summary of results is presented. Next related publications are summarized.

8.1.1 Summary of results

Composability is the capability to select and assemble simulation components in various combinations into simulation systems to satisfy specific user requirements. The defining characteristic of composability is the ability to combine and recombine components into different simulation systems for different purposes. Composability could have many benefits for the practice of simulation. The ability to compose simulation systems from repositories of reusable components, i.e., composability, has lately been a highly sought after goal among modeling and simulation developers. The expected benefits of robust, general composability include reduced simulation development cost and time, increased validity and reliability of simulation results, and increased involvement of simulation users in the process. Consequently, composability is an active research area, with both software engineering and theoretical approaches being developed.

Composability exists in two forms, syntactic and semantic (also known as engineering and modeling). Syntactic composability is the implementation of components so that they can be connected. Semantic composability is the question of whether the models implemented in the composed components can be meaningfully composed, i.e., is their combined computation valid? A theory of semantic composability has been developed that examines the semantic composability of models using formal definitions and reasoning.

This research develops a formal theory for semantic composability of simulation components, drawing upon existing theories, including mathematical logic and computability theory. The theory includes formal definitions of composability and
associated concepts, a set of theorems and proofs addressing crucial aspects of semantic composability, and an analysis of what the theoretical results imply for practical composability engineering. Theorems address specific areas of semantic composability research. Validity theorems provide requirements for preserving validity in a composition of valid components. Process complexity theorems address the computational complexity of the composition process.

8.1.2 Contributions

*Formal definitions.* Formal definitions for model, simulation, validity, and composability are developed.

*Application of existing theories.* Theoretical constructs such as labeled transition systems and bisimulation are applied to formally analyze validity and composability.

*Classes of models and validity relations.* Classes of models and validity relations characterizing the degree to which a model is valid are developed.

*Validity under composition.* Conditions under which certain classes of models are or are not valid when composed are determined.

8.1.3 Related publications

Though composability is widely understood at the conceptual level, many different specific meanings and levels have been associated with the term; those levels have been clarified, composability has been distinguished from related ideas such as interoperability, and composability research has been briefly surveyed (Petty and Weisel, 2003b). The contents of Section 1 were presented at the Spring 2003 Simulation Interoperability Workshop (Petty and Weisel, 2003b). Mikel Petty was the primary author of this paper. Formal definitions for model, simulation, and validity have been given as the basis for a theory of semantic composability (Petty and Weisel, 2003), and some of the definitions have been updated and made more general, especially validity (Petty, Weisel, and Mielke, 2003). Classes of models and validity relations are defined and the question of whether validity is preserved under composition for those classes has been studied (Weisel, Petty, and Mielke, 2003). The contents of Section 2 were
presented at the Spring 2004 Simulation Interoperability Workshop (Weisel, Petty, and Mielke, 2004). The contents of Section 7 were presented at the Fall 2003 Simulation Interoperability Workshop (Petty, Weisel, and Mielke, 2003b). Mikel Petty was the primary author of this paper. Mikel Petty was the primary author of Section 4. These sections are included for continuity.

8.2 Future work

Future work is addressed in this section. Several open theoretical questions are introduced. The composability research and development program is outlined.

8.2.1 Open theoretical questions

This section introduces several open theoretical questions for future research. Generalizing the question of validity under composition remains to be solved. Another question that follows directly from the computational complexity results presented earlier is to study the computational complexity of additional computational problems in composability.

8.2.1.1 Validity under composition

Generalizing the question of validity under composition, an important purpose of semantic composability theory is to establish the validity of compositions of models for different classes of models and validity relations. For some classes of models and relations, it is possible to prove that validity is preserved when valid models are composed. Clearly, results of this type could be of considerable value in practical applications of composability. Classes of models being studied include linear functions, affine functions, algebraic functions, elementary functions, and computable functions; classes of relations are equivalence relations and metric relations. Here, only validity of compositions of similarly developed models and validity relations are considered. Future research will consider more complex compositions.

8.2.1.2 Computational complexity questions

One line of future work that follows directly from the computational complexity results presented earlier is to study the computational complexity of the computational problems
in composability other than component selection, i.e., the problems identified earlier. We also anticipate working towards the development of formal semantic meta-models and algorithms to process them for validity determination.

8.2.2 Composability research program

We envision a four-phase research program to develop and exploit a formal theory of semantic composability. The current work described in this report is Phase 1 within that program. The complete research program, in outline, is as follows:

**Phase 1, Theory.** Develop a formal theory of semantic composability. Starting from formal definitions of key terms, such as model, simulation, and valid, develop a formal theory of semantic composability. This phase is the subject of this dissertation.

**Phase 2, Meta-models.** Develop semantic meta-model formalisms and algorithms, and compare the theory of phase 1 with related theories.

**Phase 3, Architecture.** Develop architecture and framework standards for model composition that are based on the semantic composability theory of Phase 1 and the semantic meta-models of Phase 2.

**Phase 4, Environment.** Develop a simulation software development environment and simulation component repository tool based on the results of the first three phases.

8.2.3 Semantic meta-models

Meta-models are descriptions of models; semantic meta-models are descriptions of model semantics. We believe that semantic meta-models can be used to determine if the semantics of a composition of models are valid.

This idea is not unique to this research; indeed, it was mentioned several times in different situations at the Spring 2003 Simulation Interoperability Workshop and the DMSO-organized Composable Mission Space Environments Workshop. What

---

48 Many mentions of this idea used the term “meta-data” instead of “meta-model”. Because we are more interested in specifying the semantics of models than of data, we use the latter term.
distinguishes the concept of semantic meta-models here is our recognition that to support algorithmic validity determination they must be formal. Forms such as UML, XML, or even structured documentation have been suggested for meta-models. However, this research seeks to determine if models can be composed and remain semantically valid in an algorithmic way. To do so, the semantic meta-models, i.e., the specifications of model semantics, must be unambiguous and formal. Therefore we favor formal meta-model forms such as first-order predicate calculus (Hein, 2002) and the formal software specification language Z (Potter, Sinclair, and Till, 1991).

\[
\begin{align*}
\forall x(B(x) & \rightarrow \neg L(x)) \\
\forall x(C(x) & \rightarrow \neg D(x)) \\
\forall x(-L(x) & \rightarrow D(x)) \\
\text{Therefore} & \\
\forall x(B(x) & \rightarrow \neg C(x))
\end{align*}
\]

Figure 27. An example of first-order predicate calculus (Hein, 2002)

<table>
<thead>
<tr>
<th>System</th>
<th>filters : P Filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipes : P Pipe</td>
<td></td>
</tr>
<tr>
<td>\forall c_1, c_2 : filters \cdot c_1.filter_id = c_2.filter_id \Leftrightarrow c_1 = c_2</td>
<td></td>
</tr>
<tr>
<td>\forall p : pipes \cdot p.source_filter \in filters \land p.sink_filter \in filters</td>
<td></td>
</tr>
<tr>
<td>\forall f : filters, pt : PORT</td>
<td>pt \in f.in_ports \cdot # { p : pipes</td>
</tr>
<tr>
<td>\forall f : filters, pt : PORT</td>
<td>pt \in f.out_ports \cdot # { p : pipes</td>
</tr>
</tbody>
</table>

Figure 28. An example of Z (Potter, Sinclair, and Till, 1991)
Figure 27 and Figure 28 are small examples of these formalisms.\textsuperscript{49}

Such meta-model forms are admittedly not as accessible to human users as UML, for example. However, we are convinced that less formal forms would not allow algorithmic determination of validity for a proposed composition of models. Ultimately a simulation development environment could make the formal meta-models accessible to humans via an interface or interpreter, but that is an issue for a future phase of the research program.

We prefer to use an existing formalism, such as those mentioned, for use as semantic meta-models, rather than develop a new formalism. We believe that an existing formalism will serve for this purpose and we can exploit existing theory and software for that formalism. We would develop a method of repeatably and unambiguously expressing model semantics in the meta-model formalism. Such meta-models will most likely include specifications of the limits of the subset of the input domain for which the model is valid.

Given semantic meta-models for a set of models, we would like to determine if their composition is valid. This determination will require algorithms to process the semantic meta-models. The next phase of research will develop algorithms to process semantic meta-models, determine if the composition is valid, and if possible, calculate input domain subset for which the composition is valid.

A number of aspects of a model have meaning, or semantics. The semantics of the state space (input and output), as well as the components themselves must be encoded in a way that can be automatically processed by the simulation development system. The encoded semantics of a component or model is called component meta-data. This component meta-data must be encoded in a way to allow automatic analysis of the semantics of the component or model. This is necessary to ensure that when components are composed, not only is the composition syntactically acceptable, but semantically meaningful as well. In order to allow formal proof of theories concerning the meta-data, the meta-data will be

\textsuperscript{49} The figures are not intended to show how model semantics might be represented in FOPC or Z; they are simply examples of the formalisms.
encoded as a formal theory, also called a formal system. For the meta-data for a given component, we seek an interpretation of the meta-data such that all the axioms of the formal system are true\(^{50}\). Other formal systems have been considered for encoding semantics within the context of simulation.

8.2.4 Comparison to other simulation and related formalisms

Other non-VMASC research efforts have developed, or are developing, theories or approaches that have some relationship to the semantic composability theory under development at VMASC. Those efforts include:

1. Semantic descriptors (Kasputis)
2. Wymorian systems engineering (Alessi, Wymore)
3. DEVS (Zeigler)
4. Base Object Models (SISO BOM PDG)
5. Denotational semantics (Mosses, Gunter, Scott)

We have reviewed each of these theories and have met (separately) with Kasputis, Alessi and Wymore, and the SISO BOM PDG, to discuss composability. It is clear that none of these theories is redundant with semantic composability theory. Each is approaching a different problem set with different theoretical and practical constructs. Nevertheless, it would be quite useful to examine these theories for contributions that they can make to semantic composability theory, as well as vice versa. We will show formally the differences between each with respect to composability, and integrate valuable ideas from each into semantic composability theory.

\[^{50}\text{Let } K \text{ be a first-order theory in } L . \text{ In the context of model theory, a model of } K \text{ is an interpretation of } L \text{ for which all the axioms of } K \text{ are true. Note that this definition of model is not the same as commonly used in the modeling and simulation community.}\]
9 REFERENCES


Trudeau, R. J. 1993. *Introduction to Graph Theory*. Dover, Mineola, N.Y.


CURRICULUM VITA
for
ERIC WERNER WEISEL

DEGREES:
Doctor of Philosophy (Engineering with a Concentration in Modeling and Simulation), Old Dominion University, Norfolk, VA, 2004
Master of Science (Operations Research), Florida Institute of Technology, Melbourne, FL, 1995
Bachelor of Science (Mathematics), United States Naval Academy, Annapolis, MD, 1988

PROFESSIONAL CHRONOLOGY:
WernerAnderson, Inc., Gloucester, VA
President, March 2002 – present
Virginia Modeling, Analysis, and Simulation Center, Old Dominion University, Norfolk, VA
Project Scientist, August 2001 – July 2003
United States Navy
Submarine Officer, May 1988 – June 2001

SCIENTIFIC AND PROFESSIONAL SOCIETIES MEMBERSHIP:
Institute for Operations Research and the Management Sciences

SCHOLARLY ACTIVITIES COMPLETED:


