Theorems about Composition

Michel Charpentier* K. Mani Chandy†

Caltech Technical Report CS–TR–00–02
UNH Technical Report CS–00–01

January 5, 2000

Abstract

Compositional designs require component specifications that can be composed: Designers have to be able to deduce system properties from components specifications. On the other hand, components specifications should be abstract enough to allow component reuse and to hide substantial parts of correctness proofs in components verifications. Part of the problem is that too abstract specifications do not contain enough information to be composed. Therefore, the right balance between abstraction and composability must be found. This paper explores the systematic construction of abstract specifications that can be composed through specific forms of composition called *existential* and *universal*.

*Computer Science Department, University of New Hampshire, Durham, NH 03824. email: charpov@cs.unh.edu.

†Computer Science Department, California Institute of Technology, Pasadena, CA 91125. email: mani@cs.caltech.edu.

Contents

1 Motivations

1.1 Specifications and Proofs in Compositional Design

This paper explores proofs of correctness of systems constructed by composing components. The promise of component technology is that the same component can be used in many systems, and thus the effort that goes into specifying, proving and implementing components can be exploited many times. Compositional design is productive when the effort required to find and compose components is less than the effort required to design an entire system from scratch. Therefore, greater productivity is achieved by using components that embody substantial effort. Rapidly growing commercial efforts into developing libraries of software components (using, for instance, Java beans, Microsoft DNA or CORBA) demonstrate that developing systems by composing existing components is useful.

In this paper, we explore the appropriate level of detail for component specifications. Specifications that are too abstract may be too weak to be useful in composition. Specifications that are too detailed may require systems designers to prove more useful, and more abstract, specifications from the detailed ones. Component designers can help systems designers by employing specifications that deal with exactly those component properties required for proving system properties. So, component designers should identify those component properties that they expect systems designers to need, and design their components to have these properties. Component designers may have to put a substantial amount of effort into proving that their components have properties that are useful in composition.

Figure 1 depicts specifications and proofs in a compositional design.

1.2 Abstract Specifications that Compose

We introduce an informal concept of *compositional properties* to motivate our exploration, and define terms precisely later. Compositional properties are those classes of properties that allow us to deduce system properties from component properties using simple rules. For example, mass is a compositional property because the mass of a system can be deduced in a simple way from the masses of components: the system mass is the sum of component masses. By contrast, heat emitted does not appear to be a compositional property because the heat emitted by a system depends in very complex ways on the shapes, masses, insulation properties and locations of the components.

Engineers have to compute properties of composed systems given properties of components, whether the properties are compositional or not. After all, many engineers have to compute both mass and heat emitted by composed systems. The challenge is to develop theories and methods that help engineers determine all system properties they need.

In this paper, we restrict ourselves to properties that are predicates on systems. A compositional property is a property whose truth can be established

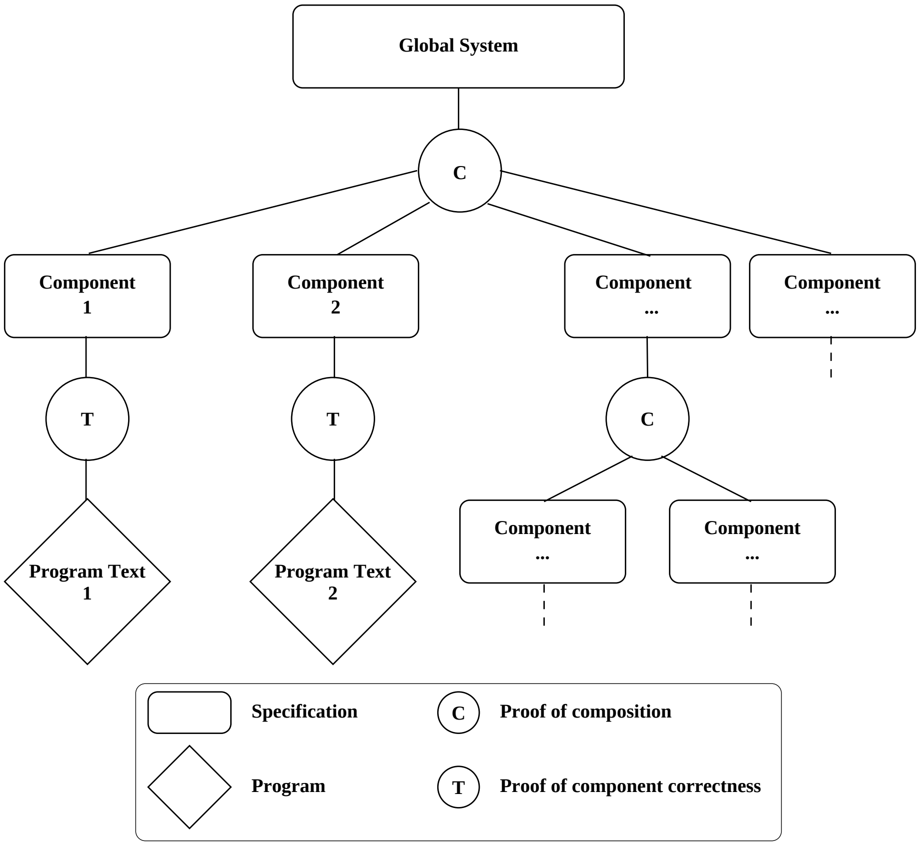

Figure 1: A compositional design

using simple rules from properties of components. The questions that we are exploring are the following:

- What are interesting compositional properties and what are the corresponding proof rules?
- How can we deduce any system property from conjunctions of these compositional properties?

The simplest rules are those that establish that a property X holds for a system given that (i) property X holds for at least one component, or (ii) property X holds for all components. Therefore, in this paper, we focus attention on two kinds of compositional properties: *existential properties* and *universal properties*. A property is an existential property exactly when for all systems, a system has the property if there exists a component of the system that has the property. A property is a universal property exactly when for all systems, a system has the property if all components of the system have the property.

Many interesting properties are neither universal nor existential. Unfortunately, we don't have rules for deducing arbitrary system properties directly from component properties. So, we deduce system properties in two steps:

1. First, we specify components as conjunctions of universal and existential properties so that we can readily derive universal and existential system properties from component properties,
2. and then we derive the system properties we need from these universal and existential system properties.

1.3 An Illustrative Digression

A critical issue in compositional design is the degree of detail in component specifications. Component specifications must be strong enough to allow system specifications to be proved from component specifications. Over-specification makes composition more difficult because designers may have to first prove more abstract and useful specifications from the detailed ones. The next few paragraphs give the reader some intuition of this tradeoff between too much detail and too little detail in component specification by considering the difference between two kinds of properties: invariants and always. This issue has been discussed earlier in [14, 13].

In the context of systems (and components) defined by their infinite computations (reactive systems), a state predicate is an *invariant property* of a program exactly when

- the predicate holds in the initial state of all computations of the program, and
- all transitions from states in which the predicate holds are to states in which the predicate continues to hold.

A state predicate is an *always property* of a program exactly when it holds in all states of all computations of the program.

Invariants tell us about transitions from all states, whether reachable or unreachable. By contrast, always properties tell us about transitions only from reachable states.

All invariant properties are always properties. An always property need not be an invariant property because the system may have a transition from an *unreachable state* for which the property holds to a state in which the property does not hold.

An invariant property is universal. If all components of a system enjoy an invariant property then the composed system (composed using the concurrent composition operator of UNITY for instance) also enjoys that invariant property. By contrast, an always property is not necessarily universal.

Properties relevant to a system in isolation (i.e., a system that is not composed with other systems) are different from properties relevant to components that we expect to compose with other components. When we study the behavior of a system in isolation its always properties are relevant, but it is not helpful to know whether an always property is also an invariant property. This is because transitions from unreachable states are irrelevant in a study of isolated systems. A designer of a system may expect that a specification in terms of always properties is superior to a specification in terms of invariant properties because always properties offer the appropriate degree of abstraction whereas invariant properties are concerned with unnecessary detail about unreachable states.

We cannot, however, deduce system properties from always properties of components. The concurrent composition of systems, all of which have the always property that a bank account is nonnegative, may yield a system in which the bank account does indeed become negative. This is because a state that is unreachable when a system executes in isolation may be reachable when that system is composed with other systems. Thus, always properties are too weak to be helpful in composition even though they have the right degree of abstraction for systems in isolation.

We can, of course, specify a component by its program text. We can deduce all system properties from the implementations (program texts) of its components. This approach is not the most productive way of compositional design because it does not re-use proofs (the T-proofs of fig. 1). Each time we design a system, we carry out proofs starting from implementations instead of starting from something more useful and abstract. Therefore, we need compositional properties that are stronger than always properties and weaker than the program text. Invariants satisfy this requirement.

Invariant properties are helpful in composition because they are universal properties. The concurrent composition of programs, all of which have the invariant property that a bank account is nonnegative, yields a program in which the bank account is guaranteed to remain nonnegative. Invariant properties have the right degree of abstraction for components though they appear to be unnecessarily strong for reasoning about systems in isolation. We can weaken invariants to obtain desired always properties.

In our research, we are searching for simple rules that allow us to prove system properties from component properties, for certain restricted kinds of properties. We don't expect to find simple rules for deducing arbitrary kinds of system properties from arbitrary kinds of component properties. The challenge is to find appropriate kinds of compositional properties, and then deduce desired properties by weakening conjunctions of compositional properties.

1.4 Predicate Transformers for Composition

Consider the following problem. A designer of a component F would like to demonstrate that any system that has F as a component enjoys a property X. If property X is an existential property, and if F has property X, then any system that has F as a component will enjoy property X. What if X is not existential?

If we could define a predicate transformer $\mathcal{E}$ where $\mathcal{E}.X$ is existential and at least as strong as X, and if we could demonstrate that component F has property $\mathcal{E}.X$, then any system that includes component F would also have existential property $\mathcal{E}.X$, and therefore would also enjoy the weaker property X. Therefore, component designers can ensure that all systems that contains their components have a property X by proving that their components have a stronger existential property $\mathcal{E}.X$.

What requirements should we place on predicate transformer $\mathcal{E}$ other than that $\mathcal{E}.X$ must be stronger than X?

The obvious answer is that $\mathcal{E}.X$ should be as weak as possible. Ideally, it should be the weakest existential property stronger than X. Of course we must prove that such a property exists.

Now consider the analogous case for universal properties. We would like to prove that a system has a property X if all its components have property X even though X is not a universal property. So, we attempt to introduce a predicate transformer $\mathcal{U}$ with the requirement that $\mathcal{U}.X$ is universal and stronger than X. If we can then prove that all components of a system have property $\mathcal{U}.X$ then we may conclude that the system enjoys this property and hence also enjoys the weaker property X.

Can we require that $\mathcal{U}.X$ be the *weakest* universal property stronger than X? We can show that we cannot do so because there does not exist, in general, a weakest universal property stronger than X. The idea of a predicate transformer $\mathcal{U}$ where $\mathcal{U}.X$ is universal and is stronger than X will indeed help in engineering systems by composing components, but we must define $\mathcal{U}$ in some way other than being the weakest.

1.5 Overview

Components here are abstract entities. They are not necessarily programs and they may not have "states" or "computations." We consider composition operators that have certain algebraic properties such as associativity and explore theorems that are derived from these properties.

In this paper, we address the following problem: can we find component properties strong enough to be composed, but weak enough to preserve abstraction? More specifically, we focus on two forms of composition, *existential* and *universal*.

In the next section we introduce components, their properties and the composition law. We also introduce a simple model of components that is used as an example throughout the paper, as well as our notation and vocabulary. Section 3 presents definitions of *existential* and *universal* properties. In this section we introduce a special form of existential properties called *guarantees*. The main results of the paper are presented in section 4 where a property transformer $\mathcal{E}$ for existential composition is defined. Theorems with regard to this property transformer are presented, and the reasons why a property transformer for universal composition is *not* defined in the same way are explored. An example, that uses the simple model previously defined, concludes that section. In section

5, the work is compared with other proposed approaches and some remaining questions are formulated. Finally, conclusions are drawn in section 6.

2 Terminology and Notations

2.1 Predicates, Components and Properties

Function application is denoted with a dot. The application of function A to parameter x is denoted by $A.x$. Predicates are boolean valued functions. We denote predicates with the capital letters X, Y, Z, ... Components are denoted with the capital letters F, G, H, ... Properties are predicates on components: $X \cdot F$ is the boolean: property X holds in component F.

2.2 Everywhere Operator

As in Dijkstra and Scholten [10], we use square brackets to denote that a predicate is "*everywhere true*." These brackets are called the *everywhere operator*. For any property X, $[X]$ is the boolean: X holds for all systems.

2.3 Composition

We restrict attention to a single composition operator $\circ$. We postulate the existence of a binary relation $\surd$ between components, and we restrict attention to composition of components that satisfy this relation. We denote by $A\surd B$ the fact that components A and B can be composed, and then their composition is denoted by $A\circ B$.

We assume the existence of a *UNIT* component such that, for any A:

$$UNIT\surd A \wedge A\surd UNIT \wedge (UNIT\circ A = A\circ UNIT = A) \tag{1}$$

Furthermore, we assume that $\circ$ is associative and that, for any A, B and C:

$$A\surd B \wedge (A\circ B)\surd C \;\equiv\; B\surd C \wedge A\surd(B\circ C) \tag{2}$$

Note that we do not assume here any other property of the operator $\circ$, such as symmetry or idempotence.

2.4 Bags of Colored Balls

To illustrate the results presented in this paper, we use a model for components defined as follows:

- components are bags of colored balls;
- bags can always be composed ($F\surd G$ for all F and G) and composition is the union of contents of the component bags in the composed bag;
- the *UNIT* element is the empty bag.

Note that properties of the form "*all balls in the bag are red*" hold in *UNIT*.

3 Existential and Universal Composition

We start our exploration of compositional properties by studying properties that obey certain rules of universal and existential quantification.

3.1 Existential Properties

A property X is existential (denoted by the boolean $exist.X$) exactly when

$$exist.X \;\triangleq\; \langle \forall F, G : F \surd G : X \cdot F \lor X \cdot G \Rightarrow X \cdot F \circ G \rangle \tag{3}$$

A system enjoys an existential property if it has a component that enjoys that property.

3.2 Universal Properties

A property X is universal (denoted by the boolean $univ.X$) exactly when:

$$univ.X \;\triangleq\; \langle \forall F, G : F \surd G : X \cdot F \land X \cdot G \Rightarrow X \cdot F \circ G \rangle \tag{4}$$

A system enjoys a universal property if all its components enjoy that property. Note that any existential property is also universal:

$$exist.X \;\Rightarrow\; univ.X$$

3.3 "*guarantees*" Properties

In this section, we show that there is a systematic way to build existential properties from any property.

We introduce a function *guarantees*, from properties × properties to properties:

$$\begin{array}{c} X \;\; guarantees \;\; Y \cdot F \\ \triangleq \\ \langle \forall H, K : H \surd F \land H \circ F \surd K : X \cdot H \circ F \circ K \Rightarrow Y \cdot H \circ F \circ K \rangle \end{array} \tag{5}$$

Properties of the form X *guarantees* Y are called guarantees *properties*. An important result about *guarantees* properties is that, for any X and Y, X *guarantees* Y is existential:

Proposition 1 $\quad exist.(X \;\; guarantees \;\; Y)$

Proof: See corollary of proposition 12 page 16. □

3.4 Basic Rules

In this section, we give basic rules relating existential properties, universal properties and *guarantees* [3]. Let E and E' be existential properties and U and U' be universal properties. Then:

$$\begin{array}{ll} exist\cdot(E \wedge E') & exist\cdot(E \vee E') \\ univ\cdot(U \wedge U') & univ\cdot(U \vee E) \end{array}$$

Note that the disjunction of universal properties is not universal in general, and that strengthening or weakening existential or universal properties does not preserve their existential or universal characteristics.

Furthermore:

$$\begin{gathered} E \cdot UNIT \;\equiv\; [E \equiv true] \\ [X \Rightarrow Y] \equiv [X \;\; guarantees \;\; Y] \\ [(X \;\; guarantees \;\; Y) \Rightarrow (X \Rightarrow Y)] \\ [(X \;\; guarantees \;\; Y) \wedge (Y \;\; guarantees \;\; Z) \Rightarrow (X \;\; guarantees \;\; Z)] \\ [(X \;\; guarantees \;\; Y) \wedge (X' \;\; guarantees \;\; Y') \Rightarrow (X \wedge X' \;\; guarantees \;\; Y \wedge Y')] \\ [(X \;\; guarantees \;\; Y) \wedge (X' \;\; guarantees \;\; Y') \Rightarrow (X \vee X' \;\; guarantees \;\; Y \vee Y')] \end{gathered}$$

3.5 Example

In the bag of colored balls model:

exist · (bag contains at least 3 balls)
exist · (bag contains at least 2 red balls and at least 1 blue ball)
exist · (bag contains at least 1 red ball) *guarantees* (bag contains at least 2 colors)
univ · (all balls in bag are black)
univ · (all balls in bag are red, or bag contains at least 1 blue ball)

The following properties are neither existential, nor universal:

(all balls in the bag are red, or all balls in the bag are blue)
(if the bag contains at least 1 red ball, then the bag contains at least 2 colors)

4 The Property Transformer $\mathcal{E}$

4.1 $\mathcal{E}$ and $\mathcal{E}'$

In this section, we show that, for any property X, there exists a weakest existential property stronger than X, denoted by $\mathcal{E}.X$. Note that X holds in any system containing a component for which $\mathcal{E}.X$ holds. This is because $\mathcal{E}.X$ holds for the system since $\mathcal{E}.X$ is existential, and there exists a component for which $\mathcal{E}.X$ holds, and since $\mathcal{E}.X$ is stronger than X we conclude that X holds for the system.

We provide two equivalent formulations for $\mathcal{E}$:

$$\mathcal{E}.X \;\triangleq\; \langle \exists Y : [Y \Rightarrow X] \wedge exist.Y : Y \rangle \tag{6}$$

$$\mathcal{E}'.X \cdot F \;\triangleq\; \langle \forall H, K : H \surd F \wedge H \circ F \surd K : X \cdot H \circ F \circ K \rangle \tag{7}$$

Instead of proving directly that $[\mathcal{E} = \mathcal{E}']$, we prove that $\mathcal{E}.X$ is the weakest existential property stronger than X, and that $\mathcal{E}'.X$ also is the weakest existential property stronger than X. From the uniqueness of such a weakest property, we conclude that $[\mathcal{E} = \mathcal{E}']$.

Note that by construction $\mathcal{E}.X$ is weaker than any existential property stronger than X, but we have to prove that $\mathcal{E}.X$ is existential. On the other hand $\mathcal{E}'.X$ is defined to be existential, but we have to prove than it is the weakest existential property stronger than X.

4.2 $\mathcal{E}.X$ is the Weakest Existential Property Stronger than X

We consider the equation (in predicates):

$$Y : [Y \Rightarrow X] \wedge \mathit{exist}.Y \tag{8}$$

It is well known [10] that such an equation has a weakest solution exactly when the disjunction of all solutions is itself a solution, and then this disjunction is the weakest solution.

From definition (6), $\mathcal{E}.X$ is the disjunction of all the solutions of equation (8). Therefore, $\mathcal{E}.X$ is the weakest existential property stronger than X if and only if $\mathcal{E}.X$ is a solution of equation (8). The proof obligation is:

$$[\mathcal{E}.X \Rightarrow X] \wedge \mathit{exist}.(\mathcal{E}.X)$$

Proposition 2 $$[\mathcal{E}.X \Rightarrow X]$$

Proof:

$$\begin{array}{ll}
 & \mathcal{E}.X \\
= & \{\text{Definition of } \mathcal{E}\ (6)\} \\
 & \langle \exists Y : [Y \Rightarrow X] \wedge \mathit{exist}.Y : Y \rangle \\
\Rightarrow & \{[Y \Rightarrow X] \wedge \mathit{exist}.Y \Rightarrow [Y \Rightarrow X] \text{ and } \exists Y \text{ is monotonic}\} \\
 & \langle \exists Y : [Y \Rightarrow X] : Y \rangle \\
\Rightarrow & \{[[Y \Rightarrow X] \wedge Y \Rightarrow X] \text{ and } \exists Y \text{ is monotonic}\} \\
 & \langle \exists Y :: X \rangle \\
= & \{\text{No free } Y \text{ in } X\} \\
 & X
\end{array}$$

□

Proposition 3 $$\mathit{exist}.(\mathcal{E}.X)$$

Proof: We consider two components F and G such that $F\surd G$ and we prove that $\mathcal{E}.X \cdot F \Rightarrow \mathcal{E}.X \cdot F{\circ}G$. By a similar argument, $\mathcal{E}.X \cdot G \Rightarrow \mathcal{E}.X \cdot F{\circ}G$, and therefore we deduce that $\mathcal{E}.X \cdot F \vee \mathcal{E}.X \cdot G \Rightarrow \mathcal{E}.X \cdot F{\circ}G$, i.e., that $\mathcal{E}.X$ is existential.

$$
\begin{array}{ll}
& \mathcal{E}.X \cdot F \wedge F\surd G \\
= & \{\text{Definition of } \mathcal{E} \ (6)\} \\
& \langle \exists Y : [Y \Rightarrow X] \wedge \mathit{exist}.Y : Y \rangle \cdot F \wedge F\surd G \\
= & \{\text{Predicate calculus}\} \\
& \langle \exists Y : [Y \Rightarrow X] \wedge \mathit{exist}.Y : Y \cdot F \wedge F\surd G \rangle \\
= & \{\text{Duplicate and expand } \mathit{exist}.Y \ (3)\} \\
& \langle \exists Y : [Y \Rightarrow X] \wedge \mathit{exist}.Y : \\
& \quad \langle \forall F', G' : F'\surd G' : Y \cdot F' \vee Y \cdot G' \Rightarrow Y \cdot F' \circ G' \rangle \wedge Y \cdot F \wedge F\surd G \rangle \\
\Rightarrow & \{\text{Choose } F' = F \text{ and } G' = G\} \\
& \langle \exists Y : [Y \Rightarrow X] \wedge \mathit{exist}.Y : (F\surd G \Rightarrow (Y \cdot F \vee Y \cdot G \Rightarrow Y \cdot F \circ G)) \wedge Y \cdot F \wedge F\surd G \rangle \\
\Rightarrow & \{\text{Modus ponens}\} \\
& \langle \exists Y : [Y \Rightarrow X] \wedge \mathit{exist}.Y : Y \cdot F \circ G \rangle \\
= & \{\text{Predicate calculus}\} \\
& \langle \exists Y : [Y \Rightarrow X] \wedge \mathit{exist}.Y : Y \rangle \cdot F \circ G \\
= & \{\text{Definition of } \mathcal{E} \ (6)\} \\
& \mathcal{E}.X \cdot F \circ G
\end{array}
$$

□

Proposition 4 *For any property X, there exists a weakest existential property stronger than X and it is $\mathcal{E}.X$.*

Proof: From propositions 2 and 3 and the characterization of extreme solutions of equations in predicates. □

4.3 $\mathcal{E}'.X$ is the Weakest Existential Property Stronger than X

In this section, we prove that $\mathcal{E}'.X$ also is the weakest existential property stronger than X. We prove first that $\mathcal{E}'.X$ is solution of equation (8), and then that any other solution is stronger than $\mathcal{E}'.X$.

Proposition 5 $$[\mathcal{E}'.X \Rightarrow X]$$

Proof:

$$
\begin{array}{ll}
& \mathcal{E}'.X \cdot F \\
= & \{\text{Definition of } \mathcal{E}' \ (7)\} \\
& \langle \forall H, K : H\surd F \wedge H \circ F\surd K : X \cdot H \circ F \circ K \rangle \\
\Rightarrow & \{\text{Choose } H = K = \mathit{UNIT}\} \\
& (\mathit{UNIT}\surd F \wedge \mathit{UNIT} \circ F\surd \mathit{UNIT} \Rightarrow X \cdot \mathit{UNIT} \circ F \circ \mathit{UNIT}) \\
= & \{\text{Axiom on } \mathit{UNIT} \ (1)\} \\
& X \cdot F
\end{array}
$$

□

Proposition 6 $$\mathit{exist}.(\mathcal{E}'.X)$$

Proof:

$$
\begin{array}{ll}
& exist.(\mathcal{E}'.X) \\
= & \{\text{Definition of existential properties}\} \\
& \langle \forall F, G : F\surd G : \mathcal{E}'.X \cdot F \vee \mathcal{E}'.X \cdot G \Rightarrow \mathcal{E}'.X \cdot F{\circ}G \rangle \\
= & \{\text{Predicate calculus}\} \\
& \quad \langle \forall F, G : F\surd G : \mathcal{E}'.X \cdot F \Rightarrow \mathcal{E}'.X \cdot F{\circ}G \rangle \\
& \wedge \quad \langle \forall F, G : F\surd G : \mathcal{E}'.X \cdot G \Rightarrow \mathcal{E}'.X \cdot F{\circ}G \rangle
\end{array}
$$

In order to prove this two proof obligations, we choose two components F and G such that $F\surd G$, and we prove first that

$$\mathcal{E}'.X \cdot F \Rightarrow \mathcal{E}'.X \cdot F{\circ}G$$

and then that

$$\mathcal{E}'.X \cdot G \Rightarrow \mathcal{E}'.X \cdot F{\circ}G$$

(The two proofs are different, because $\surd$ and $\circ$ may not be symmetric.)

$$
\begin{array}{ll}
& \mathcal{E}'.X \cdot F \;\wedge\; F\surd G \\
= & \{\text{Definition of } \mathcal{E}' \text{ (7)}\} \\
& \langle \forall H, K : H\surd F \wedge H{\circ}F\surd K : X \cdot H{\circ}F{\circ}K \rangle \wedge F\surd G \\
\Rightarrow & \{\text{For } K' \text{ s.t. } G\surd K'\text{, substitute } K \text{ with } G{\circ}K'\} \\
& \langle \forall H, K' : G\surd K' \wedge H\surd F \wedge H{\circ}F\surd G{\circ}K' : X \cdot H{\circ}F{\circ}G{\circ}K' \rangle \wedge F\surd G \\
= & \{\text{Axiom on } \surd \text{ (2) with } A = H{\circ}F,\ B = G,\ C = K'\} \\
& \langle \forall H, K' : H\surd F \wedge H{\circ}F\surd G \wedge H{\circ}F{\circ}G\surd K' : X \cdot H{\circ}F{\circ}G{\circ}K' \rangle \wedge F\surd G \\
= & \{\text{Axiom on } \surd \text{ (2) with } A = H,\ B = F,\ C = G\} \\
& \langle \forall H, K' : F\surd G \wedge H\surd F{\circ}G \wedge H{\circ}F{\circ}G\surd K' : X \cdot H{\circ}F{\circ}G{\circ}K' \rangle \wedge F\surd G \\
\Rightarrow & \{\text{Modus ponens}\} \\
& \langle \forall H, K' : H\surd (F{\circ}G) \wedge H{\circ}(F{\circ}G)\surd K' : X \cdot H{\circ}(F{\circ}G){\circ}K' \rangle \\
= & \{\text{Definition of } \mathcal{E}' \text{ (7)}\} \\
& \mathcal{E}'.X \cdot F{\circ}G
\end{array}
$$

$$
\begin{array}{ll}
& \mathcal{E}'.X \cdot G \;\wedge\; F\surd G \\
= & \{\text{Definition of } \mathcal{E}' \text{ (7)}\} \\
& \langle \forall H, K : H\surd G \wedge H{\circ}G\surd K : X \cdot H{\circ}G{\circ}K \rangle \wedge F\surd G \\
\Rightarrow & \{\text{For } H' \text{ s.t. } H'\surd F\text{, substitute } H \text{ with } H'{\circ}F\} \\
& \langle \forall H', K : H'\surd F \wedge H'{\circ}F\surd G \wedge H'{\circ}F{\circ}G\surd K : X \cdot H'{\circ}F{\circ}G{\circ}K \rangle \wedge F\surd G \\
= & \{\text{Axiom on } \surd \text{ (2) with } A = H',\ B = F,\ C = G\} \\
& \langle \forall H', K : F\surd G \wedge H'\surd F{\circ}G \wedge H'{\circ}F{\circ}G\surd K : X \cdot H'{\circ}F{\circ}G{\circ}K \rangle \wedge F\surd G \\
\Rightarrow & \{\text{Modus ponens}\} \\
& \langle \forall H', K : H'\surd (F{\circ}G) \wedge H'{\circ}(F{\circ}G)\surd K : X \cdot H'{\circ}(F{\circ}G){\circ}K \rangle \\
= & \{\text{Definition of } \mathcal{E}' \text{ (7)}\} \\
& \mathcal{E}'.X \cdot F{\circ}G
\end{array}
$$

□

This completes the proof that $\mathcal{E}'.X$ is solution of equation (8). We now prove that any other solution of (8) is stronger than $\mathcal{E}'.X$.

Proposition 7 $\quad exist.X \;\equiv\; [\mathcal{E}'.X \equiv X]$

Proof:

$\boxed{\Leftarrow}$

$$
\begin{array}{ll}
 & [\mathcal{E}'.X \equiv X] \\
= & \{\text{From proposition 6}\} \\
 & [\mathcal{E}'.X \equiv X] \wedge exist.(\mathcal{E}'.X) \\
\Rightarrow & \{\text{Leibniz}\} \\
 & exist.X
\end{array}
$$

$\boxed{\Rightarrow}$ Assume $exist.X$, prove that $X \cdot F \equiv \mathcal{E}'.X \cdot F$, for any F.

$$
\begin{array}{ll}
 & X \cdot F \\
= & \{\text{Introduce and expand } exist.X\} \\
 & X \cdot F \wedge \langle \forall H, G : H \surd G : X \cdot H \vee X \cdot G \Rightarrow X \cdot H \circ G \rangle \\
\Rightarrow & \{\text{Choose } G = F\} \\
 & X \cdot F \wedge \langle \forall H : H \surd F : X \cdot H \vee X \cdot F \Rightarrow X \cdot H \circ F \rangle \\
\Rightarrow & \{\text{Modus ponens}\} \\
 & \langle \forall H : H \surd F : X \cdot H \circ F \rangle \\
= & \{\text{Introduce and expand } exist.X\} \\
 & \langle \forall H : H \surd F : X \cdot H \circ F \wedge \langle \forall G, K : G \surd K : X \cdot G \vee X \cdot K \Rightarrow X \cdot G \circ K \rangle \rangle \\
\Rightarrow & \{\text{Choose } G = H \circ F\} \\
 & \langle \forall H : H \surd F : X \cdot H \circ F \wedge \langle \forall K : H \circ F \surd K : X \cdot H \circ F \vee X \cdot K \Rightarrow X \cdot H \circ F \circ K \rangle \rangle \\
= & \{\text{Move } \forall K \text{ outside}\} \\
 & \langle \forall H, K : H \surd F : X \cdot H \circ F \wedge (H \circ F \surd K \Rightarrow (X \cdot H \circ F \vee X \cdot K \Rightarrow X \cdot H \circ F \circ K)) \rangle \\
\Rightarrow & \{\text{Boolean calculus}\} \\
 & \langle \forall H, K : H \surd F \wedge H \circ F \surd K : X \cdot H \circ F \circ K \rangle \\
= & \{\text{Definition of } \mathcal{E}' \text{ (7)}\} \\
 & \mathcal{E}'.X \cdot F
\end{array}
$$

Since $[\mathcal{E}'.X \Rightarrow X]$ (prop. 5), this completes the proof of $X \cdot F \equiv \mathcal{E}'.X \cdot F$. $\square$

Proposition 8 ($\mathcal{E}'$ is universally conjunctive) *For any set S:*

$$[\mathcal{E}'.\langle \forall X : X \in S : X \rangle \;\equiv\; \langle \forall X : X \in S : \mathcal{E}'.X \rangle]$$

Proof:

$$
\begin{array}{ll}
 & \mathcal{E}'.\langle \forall X : X \in S : X \rangle \cdot F \\
= & \{\text{Definition of } \mathcal{E}' \text{ (7)}\} \\
 & \langle \forall H, K : H \surd F \wedge H \circ F \surd K : \langle \forall X : X \in S : X \rangle \cdot H \circ F \circ K \rangle
\end{array}
$$

$= \{\text{Predicate calculus}\}$
$\langle \forall H, K : H \surd F \wedge H \circ F \surd K : \langle \forall X : X \in S : X \cdot H \circ F \circ K \rangle \rangle$
$= \{\text{Interchange of universal quantifiers}\}$
$\langle \forall X : X \in S : \langle \forall H, K : H \surd F \wedge H \circ F \surd K : X \cdot H \circ F \circ K \rangle \rangle$
$= \{\text{Definition of } \mathcal{E}' \ (7)\}$
$\langle \forall X : X \in S : \mathcal{E}'.X \cdot F \rangle$
$= \{\text{Predicate calculus}\}$
$\langle \forall X : X \in S : \mathcal{E}'.X \rangle \cdot F$ $\square$

Proposition 9 ($\mathcal{E}'$ is monotonic)

$$[X \Rightarrow Y] \Rightarrow [\mathcal{E}'.X \Rightarrow \mathcal{E}'.Y]$$

Proof:

$[X \Rightarrow Y]$
$= \{\text{Predicate calculus}\}$
$[X \equiv X \wedge Y]$
$\Rightarrow \{\text{Leibniz}\}$
$[\mathcal{E}'.X \equiv \mathcal{E}'.(X \wedge Y)]$
$= \{\mathcal{E}' \text{ is conjunctive from prop. 8}\}$
$[\mathcal{E}'.X \equiv \mathcal{E}'.X \wedge \mathcal{E}'.Y]$
$= \{\text{Predicate calculus}\}$
$[\mathcal{E}'.X \Rightarrow \mathcal{E}'.Y]$ $\square$

Proposition 10 *For any property X, there exists a weakest existential property stronger than X and it is $\mathcal{E}'.X$.*

Proof: From propositions 5 and 6, we know that $\mathcal{E}'.X$ is solution of equation (8). It remains to show that any solution Z of (8) is stronger than $\mathcal{E}'.X$:

$[Z \Rightarrow X] \wedge \mathit{exist}.Z$
$= \{[\mathcal{E}'.Z \equiv Z] \text{ from prop. 7}\}$
$[Z \Rightarrow X] \wedge [\mathcal{E}'.Z \equiv Z]$
$\Rightarrow \{\mathcal{E}' \text{ is monotonic from prop. 9}\}$
$[\mathcal{E}'.Z \Rightarrow \mathcal{E}'.X] \wedge [\mathcal{E}'.Z \equiv Z]$
$\Rightarrow \{\text{Leibniz}\}$
$[Z \Rightarrow \mathcal{E}'.X]$ $\square$

Proposition 11 $[\mathcal{E} \ = \ \mathcal{E}']$

Proof: From the uniqueness of a weakest element, when it exists. $\square$

4.4 Relationship with "*guarantees*"

Proposition 12 $[X \ \mathit{guarantees}\ Y \ \equiv\ \mathcal{E}'.(X \Rightarrow Y)]$

Proof:

$$
\begin{array}{ll}
& X\ \mathit{guarantees}\ Y \cdot F \\
= & \{\text{Definition of } \mathit{guarantees} \text{ (def. 5)}\} \\
& \langle \forall H, K : H\surd F \wedge H \circ F \surd K : X \cdot H \circ F \circ K \Rightarrow Y \cdot H \circ F \circ K \rangle \\
= & \{\text{Predicate calculus}\} \\
& \langle \forall H, K : H\surd F \wedge H \circ F \surd K : (X \Rightarrow Y) \cdot H \circ F \circ K \rangle \\
= & \{\text{Definition of } \mathcal{E}'\} \\
& \mathcal{E}'.(X \Rightarrow Y) \cdot F
\end{array}
$$

□

Corollary: $\mathit{exist}.(X\ \mathit{guarantees}\ Y)$

4.5 A Property Transformer $\mathcal{U}$?

The reason why, for any property X, there exists a weakest existential property stronger than X, which allowed us to define the property transformer $\mathcal{E}$, is that any disjunction of existential properties in existential. This is not true for universal properties. Indeed, we can demonstrate a property X such that there is no unique weakest universal property stronger than X. This fact prevents us from defining a property transformer $\mathcal{U}$ in the same manner as we defined $\mathcal{E}$.

Proposition 13 (No $\mathcal{U}$ transformer) *There exists nontrivial models in which some properties do not have a unique weakest universal property stronger than them.*

Proof: To prove this claim, we use the model of bags of colored balls and we consider the property P defined by:

$$
\begin{array}{l}
P_0 \triangleq \text{(all balls white)} \\
P_1 \triangleq \text{(all balls black)} \\
P \ \triangleq P_0 \vee P_1
\end{array}
$$

Clearly, P_0 and P_1 are universal and, if black and white balls exist at all, P is not. Let W, if it exists, be the weakest universal property stronger than P. Then:

$$
\begin{array}{ll}
[P_0 \Rightarrow W] & \text{because } P_0 \text{ is universal and stronger than } P, \\
[P_1 \Rightarrow W] & \text{because } P_1 \text{ is universal and stronger than } P, \\
[P \Rightarrow W] & \text{from the two above,} \\
[W \Rightarrow P] & \text{by definition of } W, \\
[W \equiv P] & \text{from the two above.}
\end{array}
$$

But W is universal (by definition) and P is not, which leads us to a contradiction. Therefore, no such W exists. □

4.6 Example

Let's consider the following question: In the world of bags of colored balls, what is the property: (at least 1 red ball) *guarantees* (at least 2 colors)? Obvioulsy, the answer is: $\mathcal{E}$.((at least 1 red ball) $\Rightarrow$ (at least 2 colors)), but can we find a simpler, equivalent, formulation?

In this part, we prove the following equivalence:

$$[\mathcal{E}.((\text{at least 1 red ball}) \Rightarrow (\text{at least 2 colors})) \;\equiv\; (\text{at least 1 non red ball})]^1 \quad (9)$$

We introduce two specific properties, $UNIT_=$ ("being the unit") and its negation $UNIT_{\neq}$ ("not being the unit"):

$$UNIT_= \cdot F \;\triangleq\; (F = UNIT)$$
$$UNIT_{\neq} \cdot F \;\triangleq\; (F \neq UNIT)$$

Then, equivalence (9) follows from the following proposition:

Proposition 14 $\neg exist.UNIT_{\neq} \vee [X \equiv UNIT_{\neq}] \vee [\mathcal{E}.(UNIT_= \vee X) \equiv \mathcal{E}.X]$

Proof:
Assume $exist.UNIT_{\neq}$ and $\neg[X \equiv UNIT_{\neq}]$, and prove $[\mathcal{E}.(UNIT_= \vee X) \equiv \mathcal{E}.X]$.
First case: $F \neq UNIT$

$$
\begin{array}{ll}
 & \mathcal{E}.(UNIT_= \vee X) \cdot F \\
= & \{[\mathcal{E} = \mathcal{E}'] \text{ from prop. 11, definition of } \mathcal{E}' \text{ (7)}\} \\
 & \langle \forall H, K : H \surd F \wedge H \circ F \surd K : (UNIT_= \vee X) \cdot H \circ F \circ K \rangle \\
= & \{\text{From hypotheses } F \neq UNIT \text{ and } exist.UNIT_{\neq}, \neg(UNIT_= \cdot H \circ F \circ K\} \\
 & \langle \forall H, K : H \surd F \wedge H \circ F \surd K : X \cdot H \circ F \circ K \rangle \\
= & \{\text{Definition of } \mathcal{E}' \text{ (7)}, [\mathcal{E} = \mathcal{E}'] \text{ from prop. 11}\} \\
 & \mathcal{E}.X \cdot F
\end{array}
$$

Second case: $F = UNIT$

$$
\begin{array}{ll}
 & \mathcal{E}.(UNIT_= \vee X) \cdot UNIT \\
= & \{exist.(\mathcal{E}.Y) \text{ from prop. 3, basic rules of existential properties, page 10}\} \\
 & [\mathcal{E}.(UNIT_= \vee X) \;\equiv\; true] \\
= & \{[\mathcal{E}.true \equiv true] \text{ and } [E.Y \Rightarrow Y] \text{ from prop. 2}\} \\
 & [UNIT_= \vee X \;\equiv\; true] \\
= & \{\text{Predicate calculus } (UNIT \text{ is the only component where } X \textit{ may} \text{ not hold})\} \\
 & [X \equiv true] \vee [X \equiv UNIT_{\neq}] \\
= & \{\text{Hypothesis } \neg[X \equiv UNIT_{\neq}]\} \\
 & [X \equiv true]
\end{array}
$$

[1] This example also shows that $\mathcal{E}$ is not disjunctive, i.e., $\mathcal{E}.X \vee \mathcal{E}.Y$ is, in general, strictly stronger that $\mathcal{E}.(X \vee Y)$.

$$
\begin{aligned}
&= \{[\mathcal{E}.true \equiv true] \text{ and } [E.Y \Rightarrow Y] \text{ from prop. 2}\} \\
&\quad [\mathcal{E}.X \equiv true] \\
&= \{exist.(\mathcal{E}.Y) \text{ from prop. 3, basic rules of existential properties, page 10}\} \\
&\quad \mathcal{E}.X \cdot UNIT \qquad \square
\end{aligned}
$$

We can now apply proposition 14 to prove formula (9):

$$
\begin{aligned}
&\quad \mathcal{E}.((\text{at least 1 red ball}) \Rightarrow (\text{at least 2 colors})) \\
&= \{\text{Predicate calculus}\} \\
&\quad \mathcal{E}.(UNIT_{=} \vee (\text{at least 1 non red ball})) \\
&= \left\{\begin{array}{l} exist.UNIT_{\neq} \text{ in bags model, assume there exist red balls, hence} \\ \neg[(\text{at least 1 non red ball}) \equiv UNIT_{\neq}], \text{ apply prop. 14} \end{array}\right\} \\
&\quad \mathcal{E}.(\text{at least 1 non red ball}) \\
&= \{exist.(\text{at least 1 non red ball})\} \\
&\quad (\text{at least 1 non red ball})
\end{aligned}
$$

5 Related Work

5.1 Existential/Universal versus Assumption-Commitment

Traditional "*assumption-commitment*" (or "*rely-guarantee*" approach to composition of concurrent systems [1, 2, 12, 16, 9, 7, 8, 11] relies on an explicit specification of a component's possible environments. It is defined in terms of "open system computations", in which some steps are labeled "environment steps". In some sense, components are "prepared" to be composed, by leaving room for interaction with the outside world. Interaction with the environment is present from the start. If nothing is specified about the environment, few component properties can be proved. Properties like *always* are meaningless in this context,

The approach presented here is dual to assumption-commitment. In contrast to much of the work in assumption-commitment, we deal with properties of components, not components coupled to specific environments. There are no "environment steps" in our theory. Indeed, we want to deal with systems in which steps and computations do not exist, as we saw from the example of bags of colored balls. So, we do not use automata-based models or process models. Nor do we assume specific forms of computations such as open-system computations.

In our theory, we deal with component properties and composition in the abstract. We postulate simple rules (such as associativity and the existence of a unit element) for the composition operator, and then base a theory on these rules. We explore proofs of all kinds of properties (predicates on components) including properties such as color and mass. Here too, we postulate rules enjoyed by such properties and then prove theorems from these rules. We study composition and properties in the abstract, and not in terms of specific languages or logics such as TLA [1, 2], linear temporal logic [12, 11], Unity [16, 9, 7, 8] CSP, or process algebras.

A potential weakness of our approach is that by exploiting only theorems that we can prove from a limited set of rules (associativity, existential and universal properties) we obtain less useful results than by working with a specific programming language and its associated logic (say CSP or Unity). Despite this weakness, we believe that explorations such as these can help to identify the relationships between central results about compositional design and the postulates about composition and component properties.

5.2 The Benefits of "*guarantees*"

Existential and universal properties, as well as the *guarantees* operator were first introduced in [3] using slightly different definitions. The *guarantees* operator has several advantages compared to corresponding operators in the assumption-commitment theory.

Firstly, because guarantees properties do not reference an "environment", but only deal with component and (global) system properties, *guarantees* avoids a well-known circularity problem (due to the fact that components are environments of each other) in traditional approaches. The price we pay is that we cannot assume a property X on the environment to prove the same property X on the system, and such assumptions have proved useful in assumption-commitment specifications. To describe such behavior, we do not use *guarantees* and we rely on universal properties as in [5].

Secondly, because X *guarantees* Y is existential, regardless of the properties X and Y, *guarantees* can be used with *progress* properties in its left-hand side. Indeed, system proofs can be simplified considerably by using guarantees properties with progress properties on the left-hand side [6]. For instance, a useful property of a distributed resource manager is: All clients *eventually* return the resources they are given *guarantees* the server *eventually* satisfies all requests from clients. By contrast, much of the literature on assumption-commitment specifications deal only with assumptions that are safety properties [1, 2].

An advantage of having progress properties on the left-hand side of *guaranteesis* that component designers can prove complex properties that can be used directly in proving composed systems; thus, the effort in developing proofs of a component is amortized over many system proofs

5.3 $\mathcal{E}$ and $\mathcal{E}'$ versus "*guarantees*"

A simple, but important theorem, is that the property X *guarantees* Y is merely the application of predicate transformer $\mathcal{E}$ to the property $X \Rightarrow Y$. It says that guarantees is the weakest property stronger than implication. Indeed, this theorem is why guarantees enjoys so many of the theorems that implication does.

Also, this theorem shows why guarantees is so useful in compositional design. There are many cases in which a designer of a component needs to prove that all systems containing that component have the property X implies Y for some X and Y. Usually, such a property is not existential, and therefore the designer develops a component with the weakest existential property stronger than the desired property. In other words, the designer develops a component that satisfies the property X guarantees Y.

Two equivalent formulations of $\mathcal{E}$ are given in this report. The first is in terms of an extreme solution to an equation in predicates. This form is useful to deduce theorems about existential properties and *guarantees*. The second form uses an explicit quantification over components. This form has been useful in deducing proof rules for *guarantees* properties in the context of concurrent programs. (These rules are not given here.)

5.4 Predicate Transformers and Universal Properties

In section 4.5, we proved that, for some properties, there does not exist a weakest universal strengthening. Consequently, it is impossible to define a predicate transformer $\mathcal{U}$ that would be to universal composition what the transformer $\mathcal{E}$ is to existential composition.

Universal properties are useful where *guarantees* properties cannot be used. However, elementary properties like *always* are not universal. Therefore, we are interested in studying universal properties that are stronger than specific properties such as *always* that have been shown to be useful in domains such as concurrent programming. *Invariant* is one possible strengthening of *always*, but *invariant* properties are too strong. Intermediate universal properties, between *invariant* and *always*, can be defined [8, 16], but we need to experiment with using these properties on proving systems to determine if they have the appropriate level of abstraction. The existence of a weakest universal property stronger than *always* is still an open question.

6 Conclusions

Component technology allows designers to develop systems by composing subsystems. Compositional design is simpler than designing systems from scratch if designers have access to large libraries of components, appropriate components in the library can be discovered easily, and there are simple rules for deducing system properties from component properties. This paper reports on an ongoing exploration of rules for deducing system properties from component properties. In this paper we discussed universal and existential properties, and a predicate transformer $\mathcal{E}$.

We have explored properties of composition that can be deduced from algebraic properties of the composition operator: associativity and the existence of a unit component. Though the exploration reported in this study is abstract and is independent of specific types of systems, we are particularly concerned with applying the results to parallel and sequential composition of programs. Since Hoare triples and weakest preconditions handle sequential composition very well, the practical results of our explorations are particularly important for concurrent composition. The concurrent composition operator, as defined in UNITY, is symmetric (commutative) and idempotent in addition to being associative. This allows us to prove additional theorems about $\mathcal{E}$ for this special case and, for instance, to deduce proof rules for *guarantees* that are specific to the UNITY model. We have applied these rules and the theorems discussed here to the compositional design of several message-passing programs [6] and shared-memory programs [5].

Requiring components to be specified in terms of conjunctions of existential and universal properties may seem too restrictive. We have found, however, that this requirement is at the appropriate level of specificity for compositional design. Designers must specify their components to help those who assemble them to reason about composed systems. There may well be property types, in addition to universal and existential, that have simple rules for composition. We have found, however, that universal and existential are adequate for compositional design in many cases. Existential properties are a surprisingly rich class of properties, especially since many properties can be expressed as guarantees properties.

In many message-passing examples, components can be specified in terms of guarantees properties where the left-hand side of the guarantees is *true*. Such properties are existential properties in the sense that these properties hold for a component regardless of the system in which the component is embedded. For example, we specify a first-in first-out single input, single output, message channel by the characteristic that the sequence of messages output "follows" [4, 15, 6] the sequence of messages input, in the sense that an existential invariant of the system is that the output sequence is a prefix of the input sequence, and existential progress property is that any prefix of the input sequence is eventually a prefix of the output sequence.

We have shown that there are some properties X for which there does not exist a weakest universal property stronger than X. For specific properties of interest in specific domains, however, there may exist weakest universal properties stronger than them. For example, there may exist a weakest universal property stronger than any always property. We wish to explore such strengthening properties because always properties are important for state-transition systems.

The goal of our exploration is to understand the theorems that we can prove from simple postulates about composition and properties. This paper reports on an early step in that exploration.

References

[1] Martín Abadi and Leslie Lamport. Composing specifications. *ACM Transactions on Programming Languages and Systems*, 15(1):73–132, January 1993.

[2] Martín Abadi and Leslie Lamport. Conjoining specifications. *ACM Transactions on Programming Languages and Systems*, 17(3):507–534, May 1995.

[3] K. Mani Chandy and Beverly Sanders. Reasoning about program composition. Submitted for publication. . http://www.cise.ufl.edu/~sanders/pubs/composition.ps.

[4] Michel Charpentier. *Assistance à la Répartition de Systèmes Réactifs*. Thèse de doctorat, Institut National Polytechnique de Toulouse, November 1997.

[5] Michel Charpentier and K. Mani Chandy. Examples of program composition illustrating the use of universal properties. In J. Rolim, editor, *International workshop on Formal Methods for Parallel Programming: Theory and Applications (FMPPTA'99)*, volume 1586 of *Lecture Notes in Computer Science*, pages 1215–1227. Springer-Verlag, April 1999.

[6] Michel Charpentier and K. Mani Chandy. Towards a compositional approach to the design and verification of distributed systems. In J. Wing, J. Woodcock, and J. Davies, editors, *World Congress on Formal Methods in the Development of Computing Systems (FM'99), (Vol. I)*, volume 1708 of *Lecture Notes in Computer Science*, pages 570–589. Springer-Verlag, September 1999.

[7] Pierre Collette. Composition of assumption-commitment specifications in a UNITY style. *Science of Computer Programming*, 23:107–125, 1994.

[8] Pierre Collette. *Design of Compositional Proof Systems Based on Assumption-Commitment Specifications. Application to* UNITY. Doctoral thesis, Faculté des Sciences Appliquées, Université Catholique de Louvain, June 1994.

[9] Pierre Collette and Edgar Knapp. Logical foundations for compositional verification and development of concurrent programs in UNITY. In *International Conference on Algebraic Methodology and Software Technology*, volume 936 of *Lecture Notes in Computer Science*, pages 353–367. Springer-Verlag, 1995.

[10] Edsger W. Dijkstra and Carel S. Scholten. *Predicate calculus and program semantics*. Texts and monographs in computer science. Springer-Verlag, 1990.

[11] J.L. Fiadeiro and T. Maibaum. Verifying for reuse: foundations of object-oriented system verification. In I. Makie C. Hankin and R. Nagarajan, editors, *Theory and Formal Methods*, pages 235–257. World Scientific Publishing Company, 1995.

[12] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.

[13] Jayadev Misra. A logic for concurrent programming: Safety. *Journal of Computer and Software Engineering*, 3(2):239–272, 1995.

[14] Beverly A. Sanders. Eliminating the substitution axiom from UNITY logic. *Formal Aspects of Computing*, 3(2):189–205, April–June 1991.

[15] Paolo A. G. Sivilotti. *A Method for the Specification, Composition, and Testing of Distributed Object Systems*. PhD thesis, California Institute of Technology, 256-80 Caltech, Pasadena, California 91125, December 1997.

[16] Rob T. Udink. *Program Refinement in UNITY-like Environments*. PhD thesis, Utrecht University, September 1995.