Acta Informatica 40, 265–301 (2004)
Digital Object Identifier (DOI) 10.1007/s00236-003-0130-y

Specification transformers: a predicate transformer approach to composition

Michel Charpentier[1], **K. Mani Chandy**[2]

[1] University of New Hampshire, Computer Science Department, Nesmith Hall Room 310, Durham NH 03824, USA
(e-mail: charpov@cs.unh.edu)

[2] California Institute of Technology, Computer Science Department, 1200 E. California Boulevard MC 256-80, Pasadena, CA 91125, USA
(e-mail: mani@cs.caltech.edu)

Received: 30 May 2002/ Revised version: 16 August 2003
Published online: 30 October 2003–

Abstract. This paper explores theories that help in (i) proving that a system composed from components satisfies a system specification given only specifications of components and the composition operator, and (ii) deducing desirable properties of components from the system specification and properties of the composition operator. The paper studies compositional systems in general without making assumptions that components are computer programs. The results obtained from such abstract representations are general but also weaker than results that can be obtained from more restrictive assumptions such as assuming that systems are parallel compositions of concurrent programs. Explorations of general theories of composition can help identify fundamental issues common to many problem domains. The theory presented here is based on predicate transformers.

1 Introduction

1.1 Motivation

Engineers compose systems from simpler components. They reason about the behavior of composed systems from the specifications of the components and the manner of composition. We prefer to reason about composed systems from specifications, rather than implementations, of components because working with specifications hides unnecessary details. This paper explores theories for reasoning about composed systems.

Theories that make more restrictive assumptions about compositional operators and components yield stronger results than theories that make few assumptions. In this exploration we make few assumptions. Our goal is to identify fundamental issues in all forms of composition.

In the same spirit, we leave the specification logic undetermined. We assume that systems are specified using some logical form and we rely on predicate transformers (functions from predicates to predicates) [18] to prove properties of composed systems. Though most of the paper deals with composition in the abstract, one section is dedicated to linear temporal logic and to examples of concurrent composition of reactive systems.

In this section, we provide a motivation for an exploration of general theories of composition. Let G and H be systems, and let $\circ$ be a composition operator. Let $G \circ H$ be the system composed from G and H. We assume that the compositional operator $\circ$ is associative but we do not assume that it is symmetric. Let u be a function from systems to some domain and let $u.G$ denote the application of function u to system G. Here $u.G$ is some property of system G such as the weight of G or the volume of G or a vector of weight and volume of G. We are particularly interested in the case where u is a predicate such as: “the weight exceeds 15”.

We assume that the domain of function u is a partial ordering $\preccurlyeq$, and that $\star$ is an operator in this domain. Consider an example where a system designer is required to design a system F that satisfies:

$$\alpha \;\succcurlyeq\; u.F \ ,$$

where α is a given value. For example, u is volume and the requirement for the system F is that it occupies at most volume α. The system designer decides to design system F as a composition of systems G and H, and so the designer's proof obligation becomes:

$$\alpha \;\succcurlyeq\; u\,.\,(G \circ H) \ .$$

One way of discharging this proof obligation is by using a compositional approach, as follows. Prove the following:

i. $\alpha \;\succcurlyeq\; u.G \star u.H$;
ii. $u.G \star u.H \;\succcurlyeq\; u\,.\,(G \circ H)$.

In our example, the double proof obligation in the compositional approach is: (i) prove that the volume occupied by G composed (using $\star$) with the volume occupied by H is at most α, and (ii) the volume occupied by G composed (using $\star$) with the volume occupied by H is at least the volume occupied by component G composed (using $\circ$) with component H.

A central question for the efficacy of composition is: which is easier, discharging the proof obligation $\alpha \succcurlyeq u.(G \circ H)$ directly or by using the compositional approach? A designer who discharges the proof obligation directly works with $G \circ H$. So the designer is working directly with the implementations of G and H. A designer who uses the compositional approach works with the property composition operator $\star$ and properties $u.G$ and $u.H$ rather than with implementations G and H.

Discharging the proof obligation directly is more efficient in many situations because the implementations G and H are the strongest possible specifications of G and H: all properties of G and H, including $u.G$ and $u.H$, can be derived from their implementations. Nevertheless, discharging the proof obligation compositionally is helpful in those cases where the following conditions hold:

i. Properties $u.G$ and $u.H$ have already been computed by the designers of components G and H.
ii. Computing $u.G \star u.H$ is easier than computing $u.(G \circ H)$.
iii. The second part of the obligation, i.e., $u.G * u.H \succcurlyeq u.(G \circ H)$, holds for all G and H so that this obligation need not be discharged.

Let us explore these issues briefly. The designer of a component G will compute $u.G$ so as to reduce the amount of work required by a designer who uses G; so a system designer can presume that $u.G$ and $u.H$ have been computed. If u abstracts relevant information and hides irrelevant implementation details then computing $u.G \star u.H$ will be easier than computing $u.(G \circ H)$. Furthermore, we define the property u to be compositional for a given operator $\star$ exactly when the following holds for all G and H:

$$u.G \star u.H \;\succcurlyeq\; u\,.\,(G \circ H)\;\;.$$

For such compositional properties, discharging the system proof obligation using the compositional approach requires less effort than discharging the proof obligation directly. This paper explores compositional properties at some length.

For example, let G and H be objects that are loaded in bins, let $u.G$ be the number (whole integer) of bins required to store G, let $\star$ be addition, and let $G \circ H$ represent the object obtained by first packing G and then packing H together into as small a bin space as possible. In this case, the above proposition holds because the number of bins required to pack G and H is at most the sum of the number of bins required to pack G and the number required to pack H.

This paper restricts attention to cases where u is a point-wise predicate, $u.G$ is a boolean obtained by applying u to the component G and means that predicate u holds for component G, $\star$ is either conjunction or disjunction, $\circ$ is an associative operator, and $\succcurlyeq$ is implication. For these cases the formula of interest becomes:

$$u.G \vee u.H \;\Rightarrow\; u\,.\,(G \circ H)\;\;,$$
$$u.G \wedge u.H \;\Rightarrow\; u\,.\,(G \circ H)\;\;.$$

The constant α is the boolean constant *true*. Substituting for α, the system proof obligation is:

$$\textit{true} \;\Rightarrow\; u\,.\,(G \circ H) \;\;.$$

The system proof obligation is stated, in a better way, without implication as:

$$u\,.\,(G \circ H) \;\;.$$

The compositional approach proves the following propositions:

i. $u.G \star u.H$;
ii. $u.G \star u.H \;\Rightarrow\; u\,.\,(G \circ H)$.

This last proposition, above, holds for all compositional properties u. Therefore, for compositional properties, the system proof obligation reduces to proving $u.G \star u.H$. So, for compositional properties, the effort required using the compositional approach for discharging system proof obligations is less than the effort required by the direct approach if $u.G \star u.H$ is easier to prove than $u.(G \circ H)$.

For the case that the property composition operator $\star$ is disjunction, the the compositional approach reduces to proving that $u.G$ *or* $u.H$ holds. In most cases, proving that a component of a system has a property is easier than proving that the entire system has that property. For the case where $\star$ is conjunction, the compositional approach reduces to proving $u.G$ *and* $u.H$. Often, proving that all components of a system have a property is at least as easy as proving that the system has that property.

A study of the theorems derivable by making different assumptions about $\star$ is an abstract mathematical exploration and does not address practical engineering design issues adequately. Nevertheless, we are carrying out this exploration with the hope of identifying a few key issues in composition applicable to all domains. Our exploration of basic issues of compositional design follows earlier work by Hoare and He [23], Misra [31], Abadi and al. [1–4], and Meier and Sanders [27], which all explore compositionality in general or specifically for reactive systems. In the remainder of this section, we provide an overview of some of the questions we explore.

Compositional approximations. Now consider the case where u is not compositional. Can designers use compositional properties that are approximations for the non-compositional properties of interest? If they can, then does there exist a closest compositional approximation, and can it be computed?

Assume that $\star$ is a monotonic operator, i.e.,

$$(\alpha \preccurlyeq \alpha') \;\wedge\; (\beta \preccurlyeq \beta') \Rightarrow (\alpha \star \beta \preccurlyeq \alpha' \star \beta') \;\;.$$

Let v be a compositional function where $u \preccurlyeq v$, i.e., for all G, $u.G \preccurlyeq v.G$. We can think of v as being a compositional approximation to u. Then, one way of proving a system property

$$u\,.\,(G \circ H) \;\preccurlyeq\; \alpha$$

is to prove:

$$v\,.\,(G \circ H) \;\preccurlyeq\; \alpha \;\; .$$

One way of proving the above proposition, since v is compositional, is to show:

$$v.G \star v.H \;\preccurlyeq\; \alpha \;\; .$$

We would like to work with function u, but we may choose to work with a compositional approximation v because computing $v.G \star v.H$ is easier than computing $u.(G \circ H)$.

Best approximations. We would like v to be the lowest compositional upper bound to u, i.e., we would like v to be a compositional property for which the following holds:

i. $u \preccurlyeq v$, and
ii. for all compositional functions w such that $u \preccurlyeq w$, $v \preccurlyeq w$.

Least upper bounds may exist for some functions but not for others. We explore whether we can compute the least upper bound for a function u and whether there exists a greatest lower bound, and how these bounds could be useful in design.

Existential properties. Restrict attention to the case where $\succcurlyeq$ is boolean implication. Let the functions u and v be point-wise predicates on systems, and let $u.G$ be the boolean: "system G satisfies predicate u". We use capital letters X and Y for predicates, and hence we use X and Y in place of u and v in the following discussion. We call X and Y properties of systems and when we say that system G has property X, we mean that $X.G$ holds.

Consider the case where $\star$ is boolean disjunction. The proposition of interest is now:

$$X.G \vee X.H \;\Rightarrow\; X\,.\,(G \circ H) \;\; .$$

We define X to be an existential predicate exactly when the above proposition holds for all systems G and H. The word existential is chosen because the composed system has the property if there exists a component that satisfies the property.

We shall show that the weakest existential predicate (similar to least upper bound) and strongest existential predicate for any predicate X exists.

Any system that contains a component that has an existential property also has the same property. No component can violate an existential property of another component. So, an existential property of a component is inherited by all systems that contain that component. For example, consider a component that receives messages into a queue, carries out some finite computation on each message, and sends the message to another queue. We can specify that this component will eventually output each message it receives after suitably modifying the message, independent of other components in the system. Such a specification is existential.

Consider queues of passengers for different airlines at an airport. Each airline would like its queue of passengers to satisfy an existential specification that each passenger who enters the queue will get serviced in finite time. Unfortunately, passengers queuing for one airline can be influenced by other activities in the airport. For instance, a security threat at a passenger queue at one airline can shut down processing of passengers at other airlines. Since the behavior of a queue depends on activities elsewhere, an existential specification of that queue must include the assumption that its environment behaves within some constraints. We explore the use of existential properties in the context of interdependent systems by using "guarantees" properties discussed later.

Universal properties. Consider the case where $\star$ is boolean conjunction. The proposition of interest is now:

$$X.G \wedge X.H \;\Rightarrow\; X\,.\,(G \circ H)\ .$$

We define X to be a universal predicate exactly when the above proposition holds for all systems G and H. The word universal is chosen because the composed system has the property if all components satisfy the property.

We shall show that the strongest universal predicate for any predicate X exists, but that the weakest universal predicate does not exist for some predicates X

Consider an example of a universal property. A corporation has many offices and if all the offices satisfy a universal property then the corporation (the composition of offices) also satisfies the property. An example of a universal property is: if the total expenditure is less than R before a transaction then the total expenditure remains less than R after the transaction.

Conjugates and top-down design. The bottom-up part of design is deducing system properties from component properties. Designers can deduce that a system has an existential property if any of its components has that property. They can deduce that a system has a universal property if all of its components have that property.

The top-down aspect of design is deducing requirements of components from specifications of the composed system. Given a property of a composed system, can we deduce that all components of the system must satisfy the property or that some components of the system must satisfy the property? We use conjugate transformers to explore these questions.

Issues in composition. There are many aspects to designing systems by composition. Engineers need the infrastructure that allows them to compose components [32, 28, 25]: this includes tools for specifying and discovering components and for understanding restrictions on types of composition allowed by the components. Engineers should be able to reason about properties of composed systems from properties of the parts. In this paper, our focus is on formal derivation of system properties from component properties, and component properties from system properties. We rely on predicate transformers [18] (functions from predicates to predicates). However, in this paper we also refer to predicate transformers as specification transformers since transformers are applied to predicates that are specifications.

Let X and Y be predicates. We shall say that X "*is stronger than*" Y exactly when:

$$[X \Rightarrow Y] \ .$$

The longer phrase "*is as strong as or is stronger than*" is technically correct because $[X \Rightarrow X]$. We will however use the shorter phrase for the sake of brevity.

1.2 Composition, abstraction and reuse

A compositional framework is often defined as a framework within which system properties can be deduced from components specifications. To design a framework to satisfy this constraint is actually trivial: it suffices to include enough details about components in their specification to make composition deduction possible. As an extreme case, take a component specification to be its implementation ("the source code"). Then, the compositional deduction is obviously possible: given components code, we can prove system properties.

An unstated parameter, which makes the problem much harder, is that we expect specifications to be more abstract than implementations: they should focus on relevant component functionalities and hide as much implementation details as possible. In other words, system designers expect component descriptions to be more concise and more abstract than source code, and they want to use those descriptions and those descriptions only to reason about composition.

The primary reason for this need is reuse. If it is known, from its implementation, that a component will provide a given set of functionalities to a system, we want to reuse that knowledge every time the component itself is reused. A design and verification effort went into the component to make sure that it will provide the desired properties to a system that relies on it. Designers want to reuse that effort and not verify again that the component does what it is supposed to do. The more we hide in a component specification, the more work we reuse when the component is used in a system.

The problem we are faced with, is that specifications that are too abstract do not contain enough information to be composed. If too much of the implementation is hidden in a specification, system designers might not know enough about a component to guarantee the correctness of a system in which the component is used. Therefore, the right balance between abstraction and compositionality must be found.

Figure 1 depicts specifications and proofs in a compositional design. Proofs labeled with 'T' are component-correctness proofs. They are the responsibility of component designers. Proofs labeled with 'C' are proofs of composition, i.e., proofs of system properties from component specifications. Composition will be worthwhile if those C-proofs are kept simple: the effort required to compose components must be less than the effort to design an entire system from scratch. This can be achieved if component designers identify those component properties that they expect systems designers to need, and design their components to have these properties. This is likely to require putting a substantial amount of effort into proving that a component has properties that are useful in composition [24]. However, these T-proofs are achieved at the component level. They are left unchanged through composition and will be reused each time the component is part of a new system. A good framework for composition should allow us to put most of the effort in T-proofs and keep C-proofs as simple as possible. This is greatly influenced by the way components are specified, and especially the level of abstraction of their specifications.

1.3 An illustrative digression

The next few paragraphs give the reader some intuition of the trade-off between too much detail and too little detail in component specification by considering the difference between two kinds of properties of reactive systems: invariant properties and always properties. This issue has been

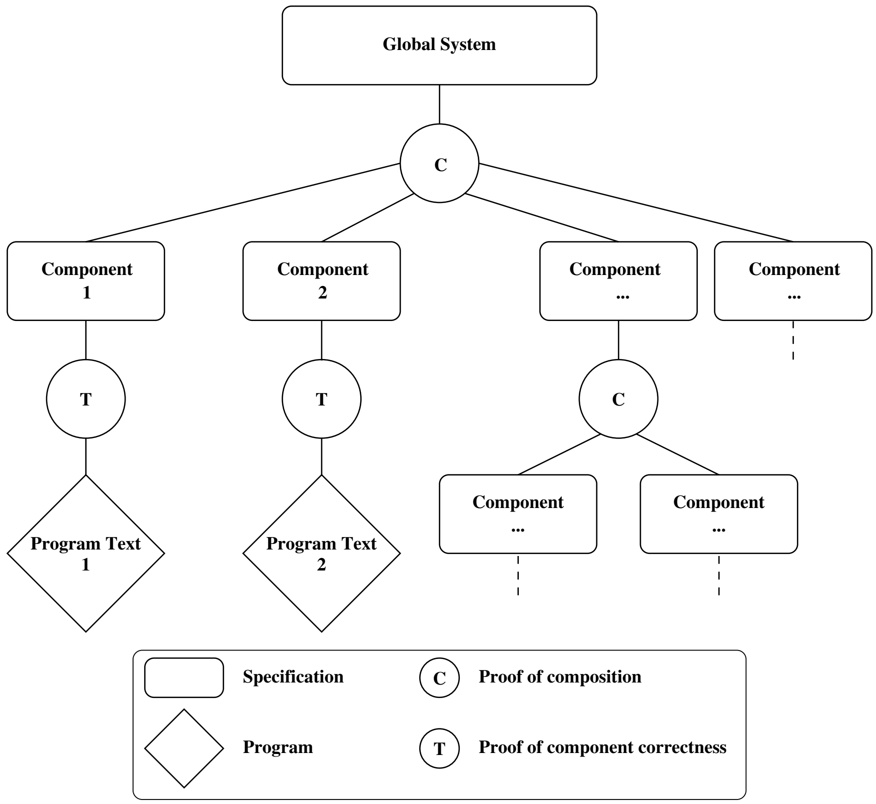

Fig. 1. A compositional design

discussed earlier, for instance in [34, 30], and is investigated more thoroughly in [10].

Reactive systems here are defined in terms of their possible initial states and their atomic transitions. The concurrent (interleaving) composition of several systems is obtained by building the intersection of their possible initial states and the union of their atomic transitions.

In this context, a state predicate is an invariant property of a system exactly when

– the predicate holds in the initial state of all computations of the system, and
– all atomic 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 system exactly when it holds in all states of all computations of the system.

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

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 compositional in the following sense: if all components of a system enjoy an invariant property then the composed system also enjoys that invariant property. By contrast, an always property is not necessarily compositional.

A system in isolation (i.e., a system that is not composed with other systems) may have properties that do not hold when composed with other components. When we study the behavior of a system in isolation, its always properties are relevant, and it is not helpful to know whether an always property is also an invariant property. In this context, always properties offer the appropriate degree of abstraction. 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.

Thus, always properties are too weak to be helpful in composition even though they have the right degree of abstraction for systems in isolation. However, invariant properties are helpful in composition because they are compositional properties. Once composition is achieved, we can weaken invariant properties to obtain desired always properties of the composed system.

While always properties are too weak to be useful in composition, invariant properties might be too strong in the sense that they are less abstract and they do not hide enough of the implementation. If invariant properties are used to specify components, only a very small part of the effort that went into the design of a component can be reused when that component is reused.

1.4 Specification transformers

We consider the following problem. A designer of a component F would like to demonstrate that any system that has F as a component satisfies a specification X. In other words, the component F "brings" the property X into any system that uses F. If F satisfies an existential specification X, then any system that has F as a component will also satisfy X. But what if X is not existential?

In this article, we define a predicate transformer $\mathbb{WE}$ where $\mathbb{WE}.X$ is existential and at least as strong as X. The role of $\mathbb{WE}$ is to *transform* a specification X that is not existential into a stronger existential specification. Now, if we can demonstrate that component F satisfies $\mathbb{WE}.X$, then any system that includes component F would also satisfy specification $\mathbb{WE}.X$, and therefore would also enjoy the weaker property X. Therefore, component designers can ensure that all systems that contain their component have a property X by proving that their component has a stronger existential property $\mathbb{WE}.X$.

What requirements should we place on predicate transformer $\mathbb{WE}$ other than that $\mathbb{WE}.X$ must be stronger than X? The obvious answer is that $\mathbb{WE}.X$ should be as weak as possible. Ideally, it should be the weakest existential property stronger than X, provided 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 $\mathbb{WU}$ with the requirement that $\mathbb{WU}.X$ is universal and stronger than X. If we can prove that all components of a system have property $\mathbb{WU}.X$ then we can conclude that the system enjoys this property and hence also enjoys the weaker property X.

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

In Sect. 3, we define $\mathbb{WE}$ such that $\mathbb{WE}.X$ is the weakest existential specification stronger than X. The main result of this section is an alternate formulation $\mathbb{WE}'$ of $\mathbb{WE}$ that shows that the only way for a component to bring a property X in any system that uses it is to satisfy the specification $\mathbb{WE}.X$. We do not define a transformer $\mathbb{WU}$ in this paper. We only show that $\mathbb{WU}.X$ cannot be defined as the weakest universal property stronger than X.

1.5 Organization of the paper

The remainder of the paper is organized as follows. Section 2 defines our notation and terminology as well as notions of component, composition and specification. Section 3 presents a question about composition and a set of transformers related to this question. As Sect. 3 adopts the point of view of a component designer and the problem of finding suitable compositional descriptions of components, Sect. 4 focuses on the component user and defines a second family of transformers that are related to the issue of using components with non-compositional descriptions. The next section moves from a component-centric point of view to a view that is focused on designing systems by decomposing them into components. It shows how the conjugates of previously defined transformers can be used to switch from a bottom-up situation to a top-down situation. Finally, Sect. 6 tackles the question of *assumption-commitment* specifications by introducing a "$\mathsf{guarantees}$" operator for composition. This sections also gives some insight into how $\mathsf{guarantees}$ can be used along with temporal logic to specify reactive systems in a compositional way. The conclusion discusses the results of this paper in the larger context of a study of composition in general.

2 Terminology and notations

2.1 Composition

We postulate a composition operator denoted by $\circ$ and we restrict ourselves to systems built from a finite number of applications of that operator. We assume that $\circ$ is associative. We do not assume other properties such as symmetry or idempotency. We do not interpret systems, and we do not consider how systems are constructed by composing elemental or atomic systems. All that is relevant in this paper is that systems can be composed to obtain systems.

We postulate the existence of a binary relation $\surd$ and we only consider systems $F \circ G$ for those components F and G for which $F \surd G$ holds. We assume that $F \surd G$ denotes that F can be composed with G (in that order).

We assume that if the system $F \circ G \circ H$ can be constructed, then it can be constructed by first composing F with G and then composing the resulting system with H, or by first composing G with H and then composing the resulting system with F on the left. Specifically, we assume the following property of $\surd$, for any F, G and H:

$$F \surd G \wedge (F \circ G) \surd H \;\equiv\; G \surd H \wedge F \surd (G \circ H) \ . \tag{1}$$

We introduce the shortcut $F \surd G \surd H$ to denote either side of this equivalence.

We assume the existence of a UNIT component that satisfies the following axiom for all systems F:

$$(\mathsf{UNIT} \surd F) \ \wedge\ (F \surd \mathsf{UNIT}) \ \wedge\ (\mathsf{UNIT} \circ F = F) \ \wedge\ (F \circ \mathsf{UNIT} = F) \, . \tag{2}$$

For some theorems, we need the additional axiom that the unit system cannot have non-unit components:

$$(F \circ G = \mathsf{UNIT}) \;\equiv\; (F = \mathsf{UNIT}) \wedge (G = \mathsf{UNIT}) \ . \tag{3}$$

Most results presented in this paper do not require this additional axiom. However, some results do. These results are marked with a $^{\circledast}$ sign.

2.2 Membership relation

We introduce a specific notation to denote that a system F is part of a system G:

$$F \triangleleft G \;\triangleq\; \langle \exists H, K : H \surd F \surd K : G = H \circ F \circ K \rangle \ . \qquad (4)$$

Note that, because of the axiom (2) on the UNIT element, $\triangleleft$ is a reflexive operator and $\mathsf{UNIT} \triangleleft F$ is true for any F.

2.3 System properties and specifications

In this paper, "*specification*" and "*system property*" are synonyms. If the context does not allow for ambiguity, "*system property*" may be abbreviated into "*property*".

Specifications are point-wise predicates on systems. In other words, we treat a specification as a total, boolean-valued function with a single argument that is a system. We use a dot notation to denote function application, as in $f.x$ denotes the application of function f to x. Therefore, for any specification X and any system F, the notation $X.F$ denotes the boolean: system F satisfies specification X. The dot has higher precedence than boolean operators (so, for example, $X.F \wedge Y.G = (X.F) \wedge (Y.G)$) and associates to the left ($a.b.c = (a.b).c$). Following [18], we use square brackets to denote that a predicate is "everywhere true". For a system property X, $[X]$ is the boolean: property X holds in all systems.

We introduce two specifications that are specific to the UNIT component, $\mathsf{UNIT}_=$ ("to be the UNIT") and its negation $\mathsf{UNIT}_{\neq}$ ("to be different from the UNIT"). For all systems F:

$$\mathsf{UNIT}_= . F \;\triangleq\; (F = \mathsf{UNIT}) \quad \text{and} \quad \mathsf{UNIT}_{\neq} . F \;\triangleq\; (F \neq \mathsf{UNIT}) \ .$$

2.4 Existential and universal specifications

In this article, we focus our attention on two kinds of compositional specifications: *existential specifications* and *universal specifications*. In Sect. 1.2, we discussed the importance of keeping proofs of composition as simple as possible. Existential and universal specifications lead naturally to simple proofs of composition, which allows us to deal with the complexity of proofs at other levels.

A specification is existential exactly when, for all systems, a system satisfies the specification if there exists a component of the system that satisfies that specification. A specification is universal exactly when, for all systems, a system satisfies the specification if all components of the system satisfy it. Of course, any existential specification is also universal, but universal specifications that are not existential, such as the invariant properties discussed in Sect. 1.3, are helpful in system design.

We denote the fact that a specification X is existential by $\mathsf{exist}.X$. The function exist is a predicate on specifications and $\mathsf{exist}.X$ is the boolean: "specification X is existential". Formally:

$$\mathsf{exist}.X \;\triangleq\; \langle \forall F, G : F \surd G : X.F \vee X.G \Rightarrow X \,.\, F \circ G \rangle \;. \tag{5}$$

In the same way, $\mathsf{univ}.X$ means that specification X is universal:

$$\mathsf{univ}.X \;\triangleq\; \langle \forall F, G : F \surd G : X.F \wedge X.G \Rightarrow X \,.\, F \circ G \rangle \;. \tag{6}$$

The predicate exist (but not univ) can be defined equivalently in term of the membership relation:

$$\mathsf{exist}.X \;\equiv\; \langle \forall F, G : F \triangleleft G : X.F \Rightarrow X.G \rangle \;.$$

2.5 Concrete models

The abstract model defined above can be instantiated with a variety of concrete models. In this paper, we will mainly focus on two: bags of colored balls and UNITY-like [30, 29] transition systems with temporal formulas.

The bags of balls model is a very simple model that emphasizes the generality of our approach. In particular, systems in this model do not have variables, states or computations. Moreover, it leads to specifications that are simple enough for us to confront formal results with our intuition. Composition here corresponds to bag-union and the UNIT element is the empty bag. In other words, the system $F \circ G$ is a bag that contains all balls from bags F and G. Bags can always be composed: the relation $\surd$ is always true.

Transition systems and temporal formulas make a more substantial model and demonstrate the relevance of our work to computer science. This model is important to us because it is used in studying concurrent composition of reactive systems. In this context, F and G are reactive programs (processes) defined by their state signature (local and shared variables), their possible initial states (a state predicate) and their possible transitions (including fairness hypotheses). Composition is concurrent composition (interleaving): the system $F \circ G$ represents $F \| G$. Its initial predicate is the conjunction of the initial predicates of F and G; its set of possible transitions is the union of the sets of transitions of F and G. The compatibility relation $\surd$ takes into account the fact that F cannot write-access G's local variables and vice-versa. UNIT is the empty process with no transitions (other than stuttering) and an initial predicate reduced to *true*. This model is defined more precisely in [7, 12, 10].

Both the bags model and the transition systems model are defined to satisfy all axioms introduced above, including (3). They also enjoy additional properties (both forms of composition are symmetric and concurrent composition of transition systems is idempotent as well), but these will not be referred to. Note that nontrivial models exist that are neither symmetric nor idempotent, such as sequential composition of transformational programs, for instance.

The following specifications of bags of balls illustrate the idea of existential and universal specifications. Note that the two universal specifications in this list are *not* existential.

exist . (at least one blue ball in the bag)
exist . (at least two balls of different colors in the bag)
univ . (no ball in the bag is blue)
univ . (if the bag contains at least 1 red ball, it contains at least 2 colors)

The following specification is neither existential nor universal:

(all balls in the bag are red, or all balls in the bag are blue)

We use the last two examples of specifications in Sects. 3.3 and 3.2, respectively.

3 Weakest specification transformers

3.1 Weakest existential transformer

We define two predicate transformers dealing with weakest existential specifications and then show that the two transformers are equal. We first consider the following equation in predicates. Predicate Z is the unknown and the equation is parametrized by predicate X. For a comprehensive presentation of equations in predicates, the reader is referred to [18, chapter 8].

$$Z : [Z \Rightarrow X] \wedge \mathsf{exist}.Z \tag{7}$$

An equation in predicates such as (7) may have extreme solutions (which are not necessarily defined in terms of fix-points). In particular, they have a weakest solution if and only if the disjunction of all solutions is itself a solution, and in this case the weakest solution is that disjunction. We define $\mathbb{WE}.X$ to be the disjunction of all solutions of (7):

$$\mathbb{WE}.X \;\triangleq\; \langle \exists Z : [Z \Rightarrow X] \wedge \mathsf{exist}.Z : Z \rangle \;\; . \tag{8}$$

Therefore, if $\mathsf{WE}.X$ is a solution of (7), it is necessarily the weakest solution, in other words, it is the weakest existential specification stronger than X. We prove in appendix A.1 that $\mathsf{WE}.X$ is the weakest solution of (7).

We now define a predicate transformer WE' such that:

$$\mathsf{WE}'.X\,.\,F \;\triangleq\; \langle \forall G : F \triangleleft G : X.G \rangle \ . \tag{9}$$

Note that $\mathsf{WE}'.X$ is defined to be existential while $\mathsf{WE}.X$ is defined to be weak. Just as we proved that $\mathsf{WE}.X$ is existential, we can prove that $\mathsf{WE}'.X$ is weak. In appendix A.2, we prove that $\mathsf{WE}'.X$ is stronger than X, is existential, and that any solution of equation (7) is stronger than $\mathsf{WE}'.X$. Therefore, we can conclude that $\mathsf{WE}'.X$ is the weakest existential specification stronger than X and, as a consequence, that $[\mathsf{WE} = \mathsf{WE}']$.

This result should be read as follows. If a component F satisfies $\mathsf{WE}.X$, the weakest existential specification stronger than X, then any system that uses component F satisfies specification X. Conversely, if a component F is designed in such a way that any system that uses it satisfies specification X, then F must satisfy specification $\mathsf{WE}.X$. In other words, $\mathsf{WE}.X$ is the characterization of a component that ensures that any system that contains that component will enjoy property X:

$$\langle \forall F, G : F \triangleleft G : Y.F \Rightarrow X.G \rangle \;\equiv\; [Y \Rightarrow \mathsf{WE}.X] \ . \tag{10}$$

WE provides us with an abstract way of describing components that not only satisfy a given specification X, but export that property into any system they are used in. The specification transformer WE enjoys a number of elementary properties such as:

$$[\mathsf{WE}.X \Rightarrow X],$$

$$\mathsf{exist}.X \;\equiv\; [\mathsf{WE}.X \equiv X],$$

$$[\mathsf{WE}.(X \wedge Y) \;\equiv\; \mathsf{WE}.X \wedge \mathsf{WE}.Y], \tag{11}$$

$$[X \Rightarrow Y] \Rightarrow [\mathsf{WE}.X \Rightarrow \mathsf{WE}.Y], \tag{12}$$

$$[\mathsf{WE}.X] \equiv [X] \ . \tag{13}$$

Property (11) states that the transformer WE is conjunctive. We can show that WE is universally conjunctive but not disjunctive (see Sect. 3.3). Property (12) says that WE is monotonic and property (13) that *true* is the only specification that transforms into *true* through WE.

3.2 Weakest universal transformer

In the same way as WE strengthens a given specification to make it existential and hence compositional, we would like to define a specification transformer

$\mathbb{WU}$ that makes specifications universal. We could then use this transformer to strengthen specifications such as always (see Sect. 1.3 above) to make them universal. Ideally, $\mathbb{WU}.(\mathsf{always}\, p)$ would be universal and stronger than $(\mathsf{always}\, p)$ but weaker and more implementation hiding than $(\mathsf{invariant}\, p)$.

Unfortunately, because disjunctions of universal specifications might not be universal, we cannot reuse the approach applied when defining $\mathbb{WE}$ to define a transformer $\mathbb{WU}$. As the following counter example demonstrates, some specifications do not have a weakest universal specification that strengthens them.

In the model of bags of colored balls, we consider the specification P defined by:

$$\begin{aligned} P_0 &\triangleq (\text{all balls are white}),\\ P_1 &\triangleq (\text{all balls are black}),\\ P &\triangleq P_0 \vee P_1 \ \ . \end{aligned}$$

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 specification stronger than P. Then:

$[P_0 \Rightarrow W]$	because P_0 is universal and stronger than P,
$[P_1 \Rightarrow W]$	because P_1 is universal and stronger than P,
$[P \Rightarrow W]$	from the two above,
$[W \Rightarrow P]$	by definition of W,
$[W \equiv P]$	from the two above.

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

This counterexample tells us that we cannot define $\mathbb{WU}.X$ to be the weakest universal specification stronger than X. Nevertheless, it does not say that a specification transformer $\mathbb{WU}$ should not be defined. We need to find another approach to build weak universal specifications in a systematic way. Several possible definitions of $\mathbb{WU}$ have been considered, and we are still investigating the question.

It should be noted that the counterexample above does not preclude the existence of a weakest universal specification for *some* properties X. In particular, it would be interesting to know if always properties admit a weakest universal strengthening or not. Such a property could be used in place of invariant to specify components. We have indeed been able to define universal specifications that lie between always and invariant, including a form for which we are unable to exhibit anything weaker [10].

3.3 Using $\mathbb{WE}$

The operator $\mathbb{WE}$ transforms a noncompositional specification into a compositional (existential) one by adding to it just what is necessary. If the specification that $\mathbb{WE}$ is applied to is already existential, the transformer does nothing.

An interesting situation is when we can *calculate* $\mathbb{WE}.X$ given X. In that case, a component designer does not need to find out what to prove about a component. Starting from the desired specification that component provides to systems, a proof obligation can be calculated by applying $\mathbb{WE}$ to that specification.

In this section, we show how such a calculation can be carried out on a simple example with bags of balls. We consider the following question: What must be proved on a bag of balls to ensure that any system that contains that bag, if it contains at least one red ball, contains at least two balls of different colors?

Formally, the specification of the component is that any system that contains that component satisfies:

$$(\text{at least 1 red ball}) \Rightarrow (\text{at least 2 colors}) \quad .$$

From (10), we know that the specification for the component is:

$$\mathbb{WE}.((\text{at least 1 red ball}) \Rightarrow (\text{at least 2 colors})) \quad . \tag{14}$$

To calculate the specification above, we rely on the following property of $\mathbb{WE}$ (which is proved in appendix A.3):

$$[X \equiv \mathsf{UNIT}_{\neq}] \vee [\mathbb{WE}.(\mathsf{UNIT}_{=} \vee X) \equiv \mathbb{WE}.X]^{\circledast} \quad . \tag{15}$$

We can then calculate an equivalent formulation of (14):

$$
\begin{array}{ll}
 & \mathbb{WE}.((\text{at least 1 red ball}) \Rightarrow (\text{at least 2 colors})) \\
= & \{\text{Predicate calculus}\} \\
 & \mathbb{WE}.(\mathsf{UNIT}_{=} \vee (\text{at least 1 non red ball})) \\
= & \left\{\begin{array}{l}\text{Assume there exist red balls, hence} \\ \neg[(\text{at least 1 non red ball}) \equiv \mathsf{UNIT}_{\neq}]\text{, apply (15)}\end{array}\right\} \\
 & \mathbb{WE}.(\text{at least 1 non red ball}) \\
= & \{\mathsf{exist}.(\text{at least 1 non red ball})\} \\
 & (\text{at least 1 non red ball}) \quad .
\end{array}
$$

Therefore, we deduce that the necessary and sufficient specification of the component is that it should contain at least one non-red ball. In other words, to ensure that any system that uses F as a component will have at least two balls of different colors provided it has at least one red ball, it is sufficient and necessary that F contains at least one non-red ball. This is consistent with our intuition. However, instead of guessing the desired property of F and then prove that it is both necessary and sufficient, we have *calculated* the property. This provides us with both the property and the proof at the same time, without having to deal explicitly with universal quantification over components. The more WE rules we have (such as (15)), the larger number of specifications of the form WE.X can be calculated.

For some domains, however, it is likely that WE.X cannot be calculated this way. One reason is that WE.X might not be expressible in the language in which X is expressed. For instance, WE applied to a formula of UNITY logic does not lead to a formula of UNITY logic in general. Nevertheless, what is needed in that case is a systematic way to prove WE.X on a component when X is given. Then, WE.X can be added to the component specification without the need for an explicit formulation. The formula for WE.X is irrelevant: a component designer only needs to know how to prove WE.X; a component user only needs to know that a system that uses the component will satisfy specification X.

4 Strongest specification transformers

4.1 Dealing with noncompositional component specifications

In the previous section, the problem we focused on was to find suitable component specifications in order to obtain desired system properties when components are assembled together. Here, we address a different question: given non compositional component specifications, what properties can be deduced about systems that rely on these components? Consider a component specification X. If X is existential, the component can be used in a system through simple proofs of composition. If X is not existential, we have seen that we can strengthen it into an existential property and try to use a component that satisfies WE.X instead. But what if X is not existential and a component satisfies X but not WE.X? What can be said about a system that uses such a component?

In this section, we would like to address this question in the context of our specification transformers.

4.2 Strongest existential transformer

We consider the equation, similar to (7):

$$Z : [X \Rightarrow Z] \wedge \mathsf{exist}.Z \quad . \tag{16}$$

Such an equation in predicates has a strongest solution if and only if the conjunction of all solutions is itself a solution, and in this case the strongest solution is that conjunction [18]. We define $\mathbb{SE}.X$ to be the conjunction of all solutions of (16):

$$\mathbb{SE}.X \ \triangleq\ \langle \forall Z : [X \Rightarrow Z] \wedge \mathsf{exist}.Z : Z \rangle \ .$$

Because any conjunction of existential specifications is existential, we can prove that $\mathbb{SE}.X$ is actually a solution of (16). The proof is similar to what was done in the case of $\mathbb{WE}$ and is not detailed in the paper. As a consequence, $\mathbb{SE}.X$ is the strongest existential property weaker than X.

If a component satisfies specification X, it also satisfies the weaker specification $\mathbb{SE}.X$ and, since the latter is existential, any system that uses this component satisfies $\mathbb{SE}.X$. $\mathbb{SE}.X$ represents the "existential part" of specification X, the part that is inherited by a system with a component that satisfies X. When X is existential, all of it is preserved (and $[\mathbb{SE}.X \equiv X]$). But when specification X is not existential, "only" $\mathbb{SE}.X$ is preserved through composition. In the worst case, $\mathbb{SE}.X$ reduces to *true* (see formula (18)).

Conversely, the strongest property Y that can be deduced of any system built from components, one of which, at least, satisfies X is $\mathbb{SE}.X$:

$$\langle \forall F, G : F \triangleleft G : X.F \Rightarrow Y.G \rangle \ \equiv\ [\mathbb{SE}.X \Rightarrow Y] \ . \tag{17}$$

Equivalently, we can say that $\mathbb{SE}.X$ characterizes those systems that are (or can be) built using a component that satisfies X. For any system F and any specification X:

$$\langle \exists G : G \triangleleft F : X.G \rangle \ \equiv\ \mathbb{SE}.X \,.\, F$$

The transformer $\mathbb{SE}$ is universally disjunctive (hence monotonic) and enjoys properties such as:

$$\begin{gathered} [X \Rightarrow \mathbb{SE}.X], \\ \mathsf{exist}.X \ \equiv\ [\mathbb{SE}.X \equiv X], \\ [\mathbb{SE}.(X \vee Y) \ \equiv\ \mathbb{SE}.X \vee \mathbb{SE}.Y], \\ [X \Rightarrow Y] \Rightarrow [\mathbb{SE}.X \Rightarrow \mathbb{SE}.Y], \\ [\mathbb{SE}.X] \equiv (X \,.\, \mathsf{UNIT})^{\circledast} \ . \end{gathered} \tag{18}$$

Property (18) has an interesting intuitive explanation. $\mathbb{SE}.X$ holds for all systems that have at least one component that satisfies X. Since all systems have UNIT as a component, for all properties X that hold for UNIT, all systems have property $\mathbb{SE}.X$. In other words, those properties that hold for UNIT have no existential part and are entirely lost through composition. (Note that some useful properties fall into that category, such as stability properties for reactive systems.) As expected, designers do not get useful information from knowing that UNIT is a component of their systems.

4.3 Strongest universal transformer

Since conjunctions of universal specifications are universal, a specification transformer $\mathbb{SU}$ can be defined in the same way as $\mathbb{SE}$ was defined. The specification $\mathbb{SU}.X$ is the strongest solution of the equation:

$$Z : [X \Rightarrow Z] \wedge \mathsf{univ}.Z \ ,$$

that is, the conjunction of all its solutions:

$$\mathbb{SU}.X \ \triangleq \ \langle \forall Z : [X \Rightarrow Z] \wedge \mathsf{univ}.Z : Z \rangle \ .$$

For any specification X, $\mathbb{SU}.X$ is the strongest universal specification weaker than X.

If a system is built from components that all satisfy specification X, this system satisfies $\mathbb{SU}.X$. Actually, $\mathbb{SU}.X$ is the strongest property Y that can be deduced that way:

$$\begin{array}{c} \langle \forall A,B,C,\cdots : A\surd B\surd C\surd \cdots \wedge X.A \wedge X.B \wedge X.C \wedge \cdots : Y.A\circ B\circ C\circ \cdots \rangle \\ \equiv \\ [\mathbb{SU}.X \Rightarrow Y] \, . \end{array} \tag{19}$$

Equivalently, $\mathbb{SU}.X$ characterizes those systems that can be built from components that all satisfy specification X. For any system F and any specification X:

$$\begin{array}{c} \langle \exists A,B,C,\cdots : A\surd B\surd C\surd \cdots \wedge X.A \wedge X.B \wedge X.C \wedge \cdots : F = A\circ B\circ C\circ \cdots \rangle \\ \equiv \\ \mathbb{SU}.X \, . \, F \, . \end{array}$$

Among interesting properties concerning $\mathbb{SU}$, we can prove:

$$[X \Rightarrow \mathbb{SU}.X],$$

$$\mathsf{univ}.X \ \equiv \ [\mathbb{SU}.X \equiv X],$$

$$[X \Rightarrow Y] \Rightarrow [\mathbb{SU}.X \Rightarrow \mathbb{SU}.Y],$$

$$[\mathbb{SU}.X \Rightarrow \mathbb{SE}.X], \tag{20}$$

$$[\mathbb{SU}.X] \Rightarrow (X \, . \, \mathsf{UNIT})^{\circledast} \ . \tag{21}$$

Property (20) comes from the fact that $\mathbb{SE}.X$ is universal (from being existential) and weaker than X. Property (21) is a direct consequence of (18) and (20). It is only an implication rather than an equivalence because $\mathbb{SU}.X$ is, in general, strictly stronger than $\mathbb{SE}.X$ (we learn more from knowing that *all* components of a system satisfy X than we do from knowing that at least one component does). Note that $\mathbb{SU}$ is monotonic but neither conjunctive, nor disjunctive (not even finitely).

5 Conjugates of specification transformers

5.1 Top-down reasoning in system design

With the transformers $\mathbb{WE}$, $\mathbb{SE}$ and $\mathbb{SU}$, we have been focusing on the deduction of system properties from component properties: assuming that one or more components satisfy properties such as X or $\mathbb{WE}.X$, we deduce that a system that is built from these components satisfies X, $\mathbb{SE}.X$ or $\mathbb{SU}.X$. While most of the literature on composition consider that form of deduction (from components to systems) to be the definition of compositionality, we believe reasoning from system specifications to component properties to be an important question as well.

If composition is to become a primary paradigm in the construction of software systems, designers need not only to be able to assemble components and reason about the resulting system, but they also need a way to identify suitable components from their system description. The design of a large system starts with a specification of that system, not specifications of its components. Designer have to use that system specification to deduce specifications of components they might use. This is a top-down aspect of composition that we want to study in this section.

5.2 All-components *and* some-components *specifications*

We define two additional forms of composition as the duals of existential and universal composition. A property is *all-components* if and only if its negation is existential:

$$\mathsf{all\text{-}c}.X \;\triangleq\; \mathsf{exist}.(\neg X) \ .$$

Unfolding the definition of exist, we find that a property is an all-components property if and only if, when it holds for any system, it holds for all components of that system:

$$\mathsf{all\text{-}c}.X \;\equiv\; \langle \forall F, G : F \surd G : X \,.\, F \circ G \Rightarrow X.F \wedge F.G \rangle \ .$$

In the same way, we define *some-component* properties as the duals of universal properties:

$$\mathsf{some\text{-}c}.X \;\triangleq\; \mathsf{univ}.(\neg X) \ .$$

A property is a some-component property if and only if, when it holds for any system, it holds for at least one component of that system:

$$\mathsf{some\text{-}c}.X \;\equiv\; \langle \forall F, G : F \surd G : X \,.\, F \circ G \Rightarrow X.F \vee F.G \rangle \ .$$

Existential and universal properties are used for bottom-up reasoning. Their dual are used in top-down reasoning.

– **Bottom-up.** Given components that have existential or universal properties, designers can deduce properties of systems composed from these components using very simple rules.
– **Top-down.** Given that a system should satisfy an all-components property, designers know that they must design all components to also satisfy that property. Likewise, given that a system should have a some-component property, designers know that they must design at least one component to have that property.

5.3 The conjugate transformer $\mathbb{WE}^*$

Every predicate transformer $\mathcal{T}$ has a unique conjugate $\mathcal{T}^*$ defined by $[\mathcal{T}^*.X \equiv \neg\mathcal{T}.(\neg X)]$. Duality provides us with a way of deducing properties of $\mathcal{T}^*$ from properties of $\mathcal{T}$, and vice-versa. For instance, $\mathcal{T}$ is monotonic iff $\mathcal{T}^*$ is monotonic. In the same way, $\mathcal{T}$ is conjunctive (resp. disjunctive) iff $\mathcal{T}^*$ is disjunctive (resp. conjunctive). In this section, we focus on the conjugates of $\mathbb{WE}$, $\mathbb{SE}$ and $\mathbb{SU}$.

We define $\mathbb{WE}^*$ as the conjugate of the property transformer $\mathbb{WE}$:

$$\mathbb{WE}^*.X \ \triangleq\ \neg\mathbb{WE}.(\neg X) \ .$$

Because the dual of existential properties are all-components properties, we can easily deduce that $\mathbb{WE}^*.X$ is the strongest all-components property weaker than property X. Moreover, applying the duality principle to formula (10), we can deduce that $\mathbb{WE}^*.X$ is the strongest property that can be deduced of all components of any system that satisfies X:

$$\langle \forall F, G : F \triangleleft G : X.G \Rightarrow Y.F \rangle \ \equiv\ [\mathbb{WE}^*.X \Rightarrow Y] \ .$$

An equivalent interpretation of $\mathbb{WE}^*.X$ is that it characterizes those components that can be part of a system that satisfies specification X. In other words, if a component does not satisfy $\mathbb{WE}^*.X$, it cannot be used to build a system with the property X.

5.4 Other conjugate transformers

Specification transformers $\mathbb{SE}^*$ and $\mathbb{SU}^*$ can de defined in a way similar to the definition of $\mathbb{WE}^*$. Given a specification X, $\mathbb{SE}^*.X$ is the weakest all-components property stronger than X and $\mathbb{SU}^*.X$ is the weakest some-component property stronger than X.

Applying a duality principle to formulas (17) and (19), we deduce that $\mathbb{SE}^*.X$ is the weakest property that must be proved on a system to ensure that all components of the system satisfy X:

$$\langle \forall F, G : F \triangleleft G : Y.G \Rightarrow X.F \rangle \ \equiv\ [Y \Rightarrow \mathbb{SE}^*.X] \ ,$$

and $\mathbb{SU}^*.X$ is the weakest property that must be proved on a system to ensure that at least one component of the system satisfies X:

$$\langle \forall A,B,C,\cdots : A\surd B\surd C\surd\cdots \wedge Y.A\circ B\circ C\circ\cdots : X.A\vee X.B\vee X.C\vee\cdots\rangle$$
$$\equiv$$
$$[Y \Rightarrow \mathbb{SU}^*.X]\,.$$

Note that, as a system specification, $\mathbb{SE}^*.X$ means that, in any decomposition of the system, all components satisfy X, while $\mathbb{SU}.X$ means that the system *can* be decomposed in such a way that all components satisfy X. Similarly, $\mathbb{SU}^*.X$ (in any decomposition, at least one component satisfies X) must not be confused with $\mathbb{SE}.X$ (there exists at least one decomposition in which at least one component satisfies X).

5.5 Comparison of the six transformers

Table 1 summarizes the intuitive interpretation for each one of the six property transformers that we have defined. For the "strongest transformers", an alternative interpretation is given. Each transformer corresponds to a different view of composition and top-down results are derived at no cost from corresponding bottom-up formulas.

Table 1. Comparison of the six transformers

$\mathbb{WE}.X$	What must be proved on a component to ensure that any system that contains that component satisfies X.
$\mathbb{SE}.X$	i) What can be deduced of any system that contains at least one component that satisfies X or, equivalently, ii) a characterization of those systems that can be built using a component that satisfies X.
$\mathbb{SU}.X$	i) What can be deduced of any system that contains only components that satisfy X or, equivalently, ii) a characterization of those systems that can be built using only components that satisfy X.
$\mathbb{WE}^*.X$	i) What can be deduced on all components of any system that satisfies X or, equivalently, ii) a characterization of those components that can be used to build a system that satisfies X.
$\mathbb{SE}^*.X$	What must be proved on a system to ensure that all components satisfy X.
$\mathbb{SU}^*.X$	What must be proved on a system to ensure that at least one component satisfies X.

6 Assumption-commitment specifications

6.1 Dealing with environments

We used invariant properties as an example of useful universal specifications. One might wonder, however, if there are many useful existential specifications. After all, what can a component assert without *any* knowledge of its environment?

In this section, we define an operator called guarantees. This operator uses two specifications as parameters and builds a form of existential assumption-commitment specification: one parameter is used to describe what is expected from the environment and the other to represent what is provided to the system by the component. The main advantage of this approach is that guarantees always leads to existential specifications, no matter what kind of properties are used as parameters. As a result, guarantees can be used as a form of assumption-commitment reasoning while keeping proofs of composition relatively simple.

6.2 A guarantees *operator*

The guarantees operator was originally defined in [8]. The definition that was used there, adapted to the notations used in this paper, is:

$$X \text{ guarantees } Y\,.\,F \;\triangleq\; \langle \forall G : F \triangleleft G : X.G \Rightarrow Y.G \rangle \;\; .$$

An important point of this definition is that the assumption part (the left-hand side of $\Rightarrow$) is $X.G$, not $X.(\mathit{env}.F)$. In other words, assumptions are made on the complete system, including the component under consideration, not just the component's environment. This choice allows us not to mention environments at all and to get around a well-known circularity problem [1, 3] generated by the fact that components are environments of each other. This results in a much simpler situation that can be summarized this way: X guarantees Y is an existential specification, regardless of what specifications X and Y actually are:

$$\mathsf{exist}.(X \text{ guarantees } Y) \;\; .$$

The following result, which is a direct consequence of (9) and the equivalence between $\mathbb{WE}$ and $\mathbb{WE}'$, tells us that X guarantees Y is a *weak* existential property:

$$[X \text{ guarantees } Y \;\equiv\; \mathbb{WE}.(X \Rightarrow Y)] \;\; . \qquad (22)$$

This means that X guarantees Y is the weakest existential strengthening of $X \Rightarrow Y$.

Intuitively, equivalence (22) can be interpreted by the following observation: guarantees relies on logical implication to build an assumption-commitment contract and on $\mathbb{WE}$ to make the resulting property compositional. The occurrence of $\mathbb{WE}$ in the alternate definition of guarantees emphasizes the fact that specifications built through guarantees are not made unnecessarily strong so as to be compositional. The use of logical implication for assumption-commitment specifications has been discussed and criticized, in particular in [1]. A potential issue is that a component could satisfy its specification by corrupting the system so that its environment ceases to satisfy its part of the contract (i.e., the right-hand side of the implication), possibly at a later time. Another weakness of logical implication is that it does not allow some forms of circular reasoning. Obviously, $X \wedge Y$ cannot be deduced from $X \Rightarrow Y$ and $Y \Rightarrow X$. Such circular deductions, however, are indeed valid when X and Y are safety properties, but require an operator other than logical implication, which can distinguish between safety and liveness specifications. Because our theory is not specific to reactive systems described in terms of traces, we cannot rely on such an operator. When the systems under consideration are indeed reactive systems described in terms of traces, one can apply $\mathbb{WE}$ to a more powerful operator than logical implication. The following section, however, discusses an example that relies on guarantees as it is defined here, using logical implication.

6.3 Application to temporal logic

In [12,7], guarantees is used, along with UNITY logic [30,29], to specify and verify a resource allocation system with distributed clients. At some point in the study, there is one component for each client and one component for the resource allocator (before the allocator itself is split into a composed system). An important part of the resource allocator component specification relies on guarantees and is written formally:

$$\forall i :: \mathit{ask}_i \nearrow \ \wedge\ \mathit{rel}_i \nearrow \tag{23a}$$

$$\wedge \quad \text{always } \langle \forall i, k :: \mathit{ask}_i[k] \leqslant \mathit{NbT} \rangle \tag{23b}$$

$$\wedge \quad \forall i, h :: h_i \sqsubseteq \mathit{giv}_i \wedge h_i \sqsubseteq_{\geqslant} \mathit{ask}_i \mapsto \mathit{Tokens}.\mathit{rel}_i \geqslant \mathit{Tokens}.h_i \tag{23c}$$

$$\text{guarantees}$$

$$\forall i, h :: h \sqsubseteq \mathit{ask}_i \mapsto h \sqsubseteq_{\leqslant} \mathit{giv}_i \tag{23d}$$

We will not define the notations that were used in that example. Instead, we will give an informal interpretation of specification (23): (23a) says that messages from clients to the resource allocator cannot be taken back (a message that is sent, is sent); (23b) says that any request by any client is always smaller than the total number of resource in the system; (23c) says that if clients were given enough resources to satisfy a request, they return those resources within finite time ($\mapsto$ is UNITY *leads-to* temporal operator). Conjuncts (23a), (23b) and (23c) together represent the resource allocator's assumptions on its environment. The right-hand side of guarantees (23d) is the allocator commitment to the system: any request will be satisfied eventually.

The presence of a progress property in the left-hand side of this specification is extremely beneficial. Much work on composition of reactive systems has focused on safety assumptions [1,2,26,35,17,15,16,20,5]. If assumptions are reduced to safety, the resource allocator specification can be written as follows:

"messages sent are not taken back"
Some Assumption-Commitment Operator
"Enough resources locally available for the first pending request"
$\mapsto$
"First pending request is satisfied"

Using such a specification, we would still be able to derive the correctness of the global system from components specifications. The reasoning involved is:

1. Apply client's commitment to return resources and to never request for more resources than there are in the system. Either the allocator has enough resources for the first pending request, or it will get more resources coming back.
2. By induction, enough resources are eventually available to answer the first pending request.
3. By induction again, other successive requests will be answered in the same way.

Note that when guarantees is used, the proof outlined above still has to be done. But it is achieved in the verification of the resource allocator component, when proving (23), not in the proof of composition. Therefore, it has to be done only once and will be reused each time the allocator component is reused. This is an important benefit since such progress proofs are usually difficult. In this example, that's two proofs by induction that we are able to move from C-proof to T-proof (see fig. 1) thanks to guarantees.

7 Discussion

This paper explores theories of reasoning about composition. We started from studying a formula:

$$u\,.\,(G \circ H) \;\preccurlyeq\; u.G \star u.H \;\;,$$

where G and H are components, $\circ$ is a composition operator on components, u is a property of components, and $\star$ is a composition operator on properties. An exploration of this formula is done, more conveniently, using abstract properties u rather than properties specifically of processes as is done in CCS [21] and CSP [22]. In this paper we restricted attention to the case where u is a predicate, $\succcurlyeq$ is logical implication, $\circ$ is associative, and $\star$ is either conjunction or disjunction:

$$u\,.\,(G \circ H) \;\Leftarrow\; u.G \wedge u.H \;\;,$$
$$u\,.\,(G \circ H) \;\Leftarrow\; u.G \vee u.H \;\;.$$

Properties that satisfy such a formula are compositional in the sense that using one of these formula to reason about composed systems is straightforward. We explored ways of dealing with non-compositional properties by working with stronger or weaker compositional properties.

An exploration of the above formulas makes no assumptions about components or the law of composition. This leads to general results and (we hope) some insight into fundamental issues in compositional design. Such an exploration of formulas suffers, however, from not producing results for specific domains such as concurrent programming or structural engineering design. More results can be obtained about composition of programs and about composition of mechanical structures than about composition of objects that may be programs or mechanical structures. In particular, a weakness of this approach is that the above formulas do not deal explicitly with time, neither physical time nor the "*eventually*" of temporal logic. For instance, we cannot develop one set of results for safety and a different set of results for liveness because the formulas have no notion of time, fairness, computations or the interleaving of steps, as in research on open systems [1, 2,5].

Six specification transformers are defined in this paper. They were originally introduced in [14] (where $\mathbb{WE}$ is called $\mathcal{E}$ and axiom (3) is not used) and [13]. Their role is to assert relationships between existential or universal specifications and specifications that do not enjoy these characteristics. Some transformers offer systematic ways of building existential and universal specifications, while other describe how composition affects non existential and non universal specifications. Furthermore, conjugates transformers are used to derive component specifications from system specifications in a top-down way. Together, these transformers provide us with a logical language in which to express compositional properties (given or expected) of system and component specifications.

Compositional techniques are useful in systems design, whether components are used in many systems or used only in a single system, because abstraction and hiding unnecessary detail are helpful. The work expended in designing and specifying a component is amortized over all the systems in which the component is used, and so component designers put more work into designing reusable components so that systems designers can put in less work each time they compose components into systems. Specifications and notations that simplify proofs of composition reduce the effort required from systems designers though they may increase the effort required by component designers. Systems designers have the options of designing their systems from scratch or of using existing components. They will not use components unless they can save time by doing so. Compositional properties, such as existential and universal properties, help systems designers save time.

The compositional framework presented in this paper can be extended in several directions. An obvious extension is to consider the case where several laws of composition are used together [9]. Researchers can also study component properties other than predicates and they can explore property composition operators other than conjunction and disjunction. They can consider the case where $u.G$ is a vector of attributes of a component such as its shape, size, color, weight, amount of power consumed, and so on. Such an exploration will help in understanding the key issues of composition.

A different direction is to extend the theory proposed here to specific, and less general, models suitable only for narrower domains. An important domain deals with system behavior over time. Time can be physical time in which each operation at a component occurs at a specific clock value, or it can be defined in terms of temporal logic operators.. Models that study interleaving of steps of components allows for open computations where environment steps are interleaved with component steps. We have started to explore the use of our transformers in conjunction with temporal logics. Several guarantees-based specifications and proofs have been written in a UNITY-like linear temporal logic [11, 12, 7] and in the branching-time logic CTL [6]. In order to do this, one has to define proof rules for specifications of the form $\mathbb{WE}.X$ within the chosen logic. Rules relative to UNITY-like logics have been defined in [10, 12]. They have also been mechanized with the higher-order theorem prover *Isabelle*, and hand-written proofs from [11, 12, 14] have been redone using the prover [33, 19]. We now need to further investigate the use of transformers other than $\mathbb{WE}$, as well as to explore application to other specification languages.

References

1. Abadi M, Lamport L (1993) Composing specifications. ACM Transactions on Programming Languages and Systems 15(1): 73–132
2. Abadi M, Lamport L (1995) Conjoining specifications. ACM Transactions on Programming Languages and Systems 17(3): 507–534

3. Abadi M, Merz S (1995) An abstract account of composition. In: Wiedermann J, Hajek P (eds) Mathematical Foundations of Computer Science, vol 969. Lecture Notes in Computer Science, pp 499–508. Springer, Berlin Heidelberg New York
4. Abadi M, Plotkin G (1993) A logical view of composition. Theoretical Computer Science 114(1): 3–30
5. Alur R, Henzinger TA, Kupferman O (1997) Alternating-time temporal logic. In: 38th Annual Symposium on Foundations of Computer Science, pp 100–109. IEEE Computer Society Press
6. Andrade H, Sanders BA (2002) An approach to compositional model checking. In: International Parallel and Distributed Processing Symposium. Workshop on Formal Methods for Parallel Programming: Theory and Applications (FMPPTA'02). IEEE
7. Chandy KM, Charpentier M (2002) An experiment in program composition and proof. Formal Methods in System Design 20(1): 7–21
8. Chandy KM, Sanders BA Reasoning about program composition. http://www.cise.ufl.edu/∼sanders/pubs/composition.ps
9. Charpentier M (2002) An approach to composition motivated by wp. In: Kutsche RD, Weber H (eds) Fundamental Approaches to Software Engineering (FASE'2002), vol 2306 of Lecture Notes in Computer Science, pp 1–14. Springer, Berlin Heidelberg New York
10. Charpentier M (2003) Composing invariants. In: Araki K, Gnesi S, Mandrioli D (eds) 12th International Symposium of Formal Methods Europe (FME'2003), vol 2805 of Lecture Notes in Computer Science. Springer, Berlin Heidelberg New York
11. Charpentier M, Chandy KM (1999) Examples of program composition illustrating the use of universal properties. In: Rolim J (ed) International workshop on Formal Methods for Parallel Programming: Theory and Applications (FMPPTA'99), vol 1586 of Lecture Notes in Computer Science, pp 1215–1227. Springer, Berlin Heidelberg New York
12. Charpentier M, Chandy KM (1999) Towards a compositional approach to the design and verification of distributed systems. In: Wing J, Woodcock J, Davies J (eds) World Congress on Formal Methods in the Development of Computing Systems (FM'99), (Volume I), vol 1708 of Lecture Notes in Computer Science, pp 570–589. Springer, Berlin Heidelberg New York
13. Charpentier M, Chandy KM (2000) Reasoning about composition using property transformers and their conjugates. In: van Leeuwen J, Watanabe O, Hagiya M, Mosses PD, Ito T (eds) Theoretical Computer Science: Exploring New Frontiers of Theoretical Informatics (IFIP-TCS'2000), vol 1872 of Lecture Notes in Computer Science, pp 580–595. Springer, Berlin Heidelberg New York
14. Charpentier M, Chandy KM (2000) Theorems about composition. In: Backhouse R, Nuno Oliveira J (eds) International Conference on Mathematics of Program Construction (MPC'2000), vol 1837 of Lecture Notes in Computer Science, pp 167–186. Springer, Berlin Heidelberg New York
15. Collette P (1994) Composition of assumption-commitment specifications in a UNITY style. Science of Computer Programming 23: 107–125
16. Collette P (1994) Design of Compositional Proof Systems Based on Assumption-Commitment Specifications. Application to UNITY. Doctoral thesis, Faculté des Sciences Appliquées, Université Catholique de Louvain
17. Collette P, Knapp E (1995) Logical foundations for compositional verification and development of concurrent programs in UNITY. In: International Conference on Algebraic Methodology and Software Technology, vol 936 of Lecture Notes in Computer Science, pp 353–367. Springer, Berlin Heidelberg New York
18. Dijkstra EW, Scholten CS (1990) Predicate calculus and program semantics. Texts and monographs in computer science. Springer, Berlin Heidelberg New York

19. Ehmety SO, Paulson LC (2002) Program composition in isabelle/UNITY. In: International Parallel and Distributed Processing Symposium. Workshop on Formal Methods for Parallel Programming: Theory and Applications (FMPPTA'02). IEEE
20. Fiadeiro JL, Maibaum T (1995) Verifying for reuse: foundations of object-oriented system verification. In: Makie I, Hankin C, Nagarajan R (eds) Theory and Formal Methods, pp 235–257. World Scientific Publishing Company
21. Hennessy M, Milner R (1985) Algebraic laws for nondeterminism and concurrency. Journal of the ACM 32(1): 137–161
22. Hoare CAR (1984) Communicating Sequential Processes. Prentice-Hall International, Englewoood Cliffs, NJ
23. Hoare CAR, He J (1998) Unifying theories of programming (first edn). Prentice Hall, London New York
24. Lamport L (1997) Composition: A way to make proofs harder. In: de Roever W-P, Langmaack H, Pnueli A (eds) Compositionality: The Significant Difference. International Symposium, COMPOS'97, vol 1536 of Lecture Notes in Computer Science, pp 402–423. Springer, Berlin Heidelberg New York
25. Leavens GT, Sitaraman M (eds) (2000) Foundations of Component-Based Systems. Cambridge University Press, Cambridge (UK) New York
26. Manna Z, Pnueli A (1992) The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer, Berlin Heidelberg New York
27. Meier D, Sanders B (2000) Composing leads-to properties. Theoretical Computer Science 243(1–2): 339–361
28. Microsoft Corporation (1997) The Component Object Model specification.
29. Misra J (1995) A logic for concurrent programming: Progress. Journal of Computer and Software Engineering 3(2): 273–300
30. Misra J (1995) A logic for concurrent programming: Safety. Journal of Computer and Software Engineering 3(2): 239–272
31. Misra J (2001) A discipline of multiprogramming: programming theory for distributed applications. Monographs in Computer Science. Springer, Berlin Heidelberg New York
32. Object Management Group. The Common Object Request Broker: Architecture and Specification. Version 2.0.
33. Paulson LC (2001) Mechanizing a theory of program composition for UNITY. ACM Transactions on Computational Logic (To appear)
34. Sanders BA (1991) Eliminating the substitution axiom from UNITY logic. Formal Aspects of Computing 3(2): 189–205
35. Udink RT (1995) Program Refinement in UNITY-like Environments. PhD thesis, Utrecht University

A Proofs

A.1 Proofs related to **WE**

Proposition 1 $[\mathrm{WE}.X \Rightarrow X]$

Proof. $\mathrm{WE}.X$

$= \{\text{Definition of WE (8)}\}$

$\langle \exists Z : [Z \Rightarrow X] \wedge \mathsf{exist}.Z : Z \rangle$

$\Rightarrow \{[Z \Rightarrow X] \wedge \mathsf{exist}.Z \Rightarrow [Z \Rightarrow X]$, and $\exists Z$ is monotonic$\}$

$$
\begin{array}{ll}
 & \langle \exists Z : [Z \Rightarrow X] : Z \rangle \\
\Rightarrow & \{[[Z \Rightarrow X] \wedge Z \Rightarrow X], \text{ and } \exists Z \text{ is monotonic}\} \\
 & \langle \exists Z :: X \rangle \\
= & \{\text{No free } Z \text{ in } X\} \\
 & X
\end{array}
$$
□

Proposition 2 $\mathsf{exist}.(\mathbb{WE}.X)$

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

$$
\begin{array}{ll}
 & \mathbb{WE}.X\ .\ F \wedge F \surd G \\
= & \{\text{Definition of } \mathbb{WE}\ (8)\} \\
 & \langle \exists Z : [Z \Rightarrow X] \wedge \mathsf{exist}.Z : Z \rangle\ .\ F \wedge F \surd G \\
= & \{\text{Predicate calculus}\} \\
 & \langle \exists Z : [Z \Rightarrow X] \wedge \mathsf{exist}.Z : Z.F \wedge F \surd G \rangle \\
= & \{\text{Duplicate and expand } \mathsf{exist}.Z\ (5)\} \\
 & \langle \exists Z : [Z \Rightarrow X] \wedge \mathsf{exist}.Z : \\
 & \quad \langle \forall F', G' : F' \surd G' : Z.F' \vee Z.G' \Rightarrow Z.F' \circ G' \rangle \wedge Z.F \wedge F \surd G \rangle \\
\Rightarrow & \{\text{Choose } F' := F \text{ and } G' := G\} \\
 & \langle \exists Z : [Z \Rightarrow X] \wedge \mathsf{exist}.Z : \\
 & \quad (F \surd G \Rightarrow (Z.F \vee Z.G \Rightarrow Z.F \circ G)) \wedge Z.F \wedge F \surd G \rangle \\
\Rightarrow & \{\text{Modus ponens}\} \\
 & \langle \exists Z : [Z \Rightarrow X] \wedge \mathsf{exist}.Z : Z.F \circ G \rangle \\
= & \{\text{Predicate calculus}\} \\
 & \langle \exists Z : [Z \Rightarrow X] \wedge \mathsf{exist}.Z : Z \rangle\ .\ F \circ G \\
= & \{\text{Definition of } \mathbb{WE}\ (8)\} \\
 & \mathbb{WE}.X\ .\ F \circ G
\end{array}
$$
□

A.2 Proofs related to $\mathbb{WE}'$

Proposition 3 $[\mathbb{WE}'.X \Rightarrow X]$

Proof.

$$
\begin{array}{ll}
 & \mathbb{WE}'.X\ .\ F \\
= & \{\text{Definition of } \mathbb{WE}'\ (9)\} \\
 & \langle \forall G : F \triangleleft G : X.G \rangle \\
= & \{\text{Definition of } \triangleleft\ (4), \text{ predicate calculus}\} \\
 & \langle \forall H, K : H \surd F \surd K : X\ .\ H \circ F \circ K \rangle
\end{array}
$$

$\Rightarrow$ {Choose $H := \mathsf{UNIT}$ and $K := \mathsf{UNIT}$}
$(\mathsf{UNIT} \surd F \surd \mathsf{UNIT} \Rightarrow X \,.\, \mathsf{UNIT} \circ F \circ \mathsf{UNIT})$
$=$ {Axiom about UNIT (2)}
$X \,.\, F$ $\square$

Proposition 4 $\mathsf{exist}.(\mathrm{WE}'.X)$

Proof. $\mathsf{exist}.(\mathrm{WE}'.X)$
$=$ {Definition of existential properties}
$\langle \forall F, G : F \surd G : \mathrm{WE}'.X \,.\, F \vee \mathrm{WE}'.X \,.\, G \Rightarrow \mathrm{WE}'.X \,.\, F \circ G \rangle$
$=$ {Predicate calculus}
$\langle \forall F, G : F \surd G : \mathrm{WE}'.X \,.\, F \Rightarrow \mathrm{WE}'.X \,.\, F \circ G \rangle$
$\wedge \langle \forall F, G : F \surd G : \mathrm{WE}'.X \,.\, G \Rightarrow \mathrm{WE}'.X \,.\, F \circ G \rangle$

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

$$\mathrm{WE}'.X \,.\, F \Rightarrow \mathrm{WE}'.X \,.\, F \circ G$$

and then that

$$\mathrm{WE}'.X \,.\, G \Rightarrow \mathrm{WE}'.X \,.\, F \circ G \ .$$

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

$\mathrm{WE}'.X \,.\, F \ \wedge \ F \surd G$
$=$ {Definition of WE' (9)}
$\langle \forall G' : F \lhd G' : X.G' \rangle \wedge F \surd G$
$=$ {Definition of $\lhd$ (4), predicate calculus}
$\langle \forall H, K : H \surd F \surd K : X \,.\, H \circ F \circ K \rangle \wedge F \surd G$
$\Rightarrow$ {For K' s.t. $G \surd K'$, replace K with $G \circ K'$}
$\langle \forall H, K' : G \surd K' \wedge H \surd F \surd G \circ K' : X \,.\, H \circ F \circ G \circ K' \rangle \wedge F \surd G$
$=$ {Axiom about $\surd$ (1), using the shortcut}
$\langle \forall H, K' : F \surd G \wedge H \surd F \circ G \surd K' : X \,.\, H \circ F \circ G \circ K' \rangle \wedge F \surd G$
$\Rightarrow$ {Modus ponens}
$\langle \forall H, K' : H \surd (F \circ G) \surd K' : X \,.\, H \circ (F \circ G) \circ K' \rangle$
$=$ {Definition of $\lhd$ (4), predicate calculus}
$\langle \forall G' : F \circ G \ \lhd \ G' : X \,.\, G' \rangle$
$=$ {Definition of WE' (9)}
$\mathrm{WE}'.X \,.\, F \circ G$

$\mathrm{WE}'.X \,.\, G \ \wedge \ F \surd G$
$=$ {Definition of WE' (9)}

$$
\begin{array}{ll}
 & \langle \forall G' : G \lhd G' : X.G' \rangle \wedge F \surd G \\
= & \{\text{Definition of } \lhd \text{ (4), predicate calculus}\} \\
 & \langle \forall H, K : H \surd G \surd K : X\,.\,H \circ G \circ K \rangle \wedge F \surd G \\
\Rightarrow & \{\text{For } H' \text{ s.t. } H' \surd F\text{, replace } H \text{ with } H' \circ F\} \\
 & \langle \forall H', K : H' \surd F \wedge H' \circ F \surd G \surd K : X\,.\,H' \circ F \circ G \circ K \rangle \wedge F \surd G \\
= & \{\text{Axiom about } \surd \text{ (1), using the shortcut}\} \\
 & \langle \forall H', K : F \surd G \wedge H' \surd F \circ G \surd K : X\,.\,H' \circ F \circ G \circ K \rangle \wedge F \surd G \\
\Rightarrow & \{\text{Modus ponens}\} \\
 & \langle \forall H', K : H' \surd (F \circ G) \surd K : X\,.\,H' \circ (F \circ G) \circ K \rangle \\
= & \{\text{Definition of } \lhd \text{ (4), predicate calculus}\} \\
 & \langle \forall G' : F \circ G \;\lhd\; G' : X\,.\,G' \rangle \\
= & \{\text{Definition of } \mathsf{WE}' \text{ (9)}\} \\
 & \mathsf{WE}'.X\,.\,F \circ G
\end{array}
$$
□

This completes the proof that $\mathsf{WE}'.X$ is solution of equation (7). We now prove that any other solution of (7) is stronger than $\mathsf{WE}'.X$.

Proposition 5 $\mathsf{exist}.X \;\equiv\; [\mathsf{WE}'.X \equiv X]$

Proof. $\boxed{\Leftarrow}$

$$
\begin{array}{ll}
 & [\mathsf{WE}'.X \equiv X] \\
= & \{\text{From Proposition 4}\} \\
 & [\mathsf{WE}'.X \equiv X] \wedge \mathsf{exist}.(\mathsf{WE}'.X) \\
\Rightarrow & \{\text{Leibniz}\} \\
 & \mathsf{exist}.X
\end{array}
$$

$\boxed{\Rightarrow}$ Assume $\mathsf{exist}.X$, prove that $X\,.\,F \equiv \mathsf{WE}'.X\,.\,F$, for any F.

$$
\begin{array}{ll}
 & X\,.\,F \\
= & \{\text{Introduce and expand } \mathsf{exist}.X\} \\
 & X\,.\,F \wedge \langle \forall H, G : H \surd G : X\,.\,H \vee X\,.\,G \Rightarrow X\,.\,H \circ G \rangle \\
\Rightarrow & \{\text{Choose } G := F\} \\
 & X\,.\,F \wedge \langle \forall H : H \surd F : X\,.\,H \vee X\,.\,F \Rightarrow X\,.\,H \circ F \rangle \\
\Rightarrow & \{\text{Modus ponens}\} \\
 & \langle \forall H : H \surd F : X\,.\,H \circ F \rangle \\
= & \{\text{Introduce and expand } \mathsf{exist}.X\} \\
 & \langle \forall H : H \surd F : X.H \circ F \wedge \langle \forall G, K : G \surd K : X.G \vee X.K \Rightarrow X.G \circ K \rangle \rangle \\
\Rightarrow & \{\text{Choose } G := H \circ F\} \\
 & \langle \forall H : H \surd F : \\
 & \quad X\,.\,H \circ F \wedge \langle \forall K : H \circ F \surd K : X\,.\,H \circ F \vee X\,.\,K \Rightarrow X\,.\,H \circ F \circ K \rangle \rangle
\end{array}
$$

$=$ {Move $\forall K$ outside}
$\langle \forall H, K : H \surd F :$
$X . H \circ F \wedge (H \circ F \surd K \Rightarrow (X . H \circ F \vee X . K \Rightarrow X . H \circ F \circ K))\rangle$
$\Rightarrow$ {Predicate calculus, $\forall H, K$ is monotonic}
$\langle \forall H, K : H \surd F \wedge H \circ F \surd K : X . H \circ F \circ K\rangle$
$=$ {Axiom about $\surd$ (1), introducing the shortcut}
$\langle \forall H, K : H \surd F \surd K : X . H \circ F \circ K\rangle$
$=$ {Definition of $\triangleleft$ (4), predicate calculus}
$\langle \forall G : F \triangleleft G : X . G\rangle$
$=$ {Definition of WE' (9)}
$\mathrm{WE}'.X . F$

Since $[\mathrm{WE}'.X \;\Rightarrow\; X]$ (Prop. 3), this completes the proof of $X.F \equiv \mathrm{WE}'.X.F$. $\square$

Proposition 6 (WE' is universally conjunctive) *For any set S:*

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

Proof. $\quad \mathrm{WE}'.\langle \forall X : X \in S : X\rangle . F$
$=$ {Definition of WE' (9)}
$\langle \forall G : F \triangleleft G : \langle \forall X : X \in S : X\rangle.G\rangle$
$=$ {Definition of $\triangleleft$ (4), predicate calculus}
$\langle \forall H, K : H \surd F \surd K : \langle \forall X : X \in S : X\rangle . H \circ F \circ K\rangle$
$=$ {Predicate calculus}
$\langle \forall H, K : H \surd F \surd K : \langle \forall X : X \in S : X . H \circ F \circ K\rangle\rangle$
$=$ {Interchange of universal quantifiers}
$\langle \forall X : X \in S : \langle \forall H, K : H \surd F \surd K : X . H \circ F \circ K\rangle\rangle$
$=$ {Definition of $\triangleleft$ (4), predicate calculus}
$\langle \forall X : X \in S : \langle \forall G : F \triangleleft G : X.G\rangle\rangle$
$=$ {Definition of WE' (9)}
$\langle \forall X : X \in S : \mathrm{WE}'.X . F\rangle$
$=$ {Predicate calculus}
$\langle \forall X : X \in S : \mathrm{WE}'.X\rangle . F$ $\square$

Proposition 7 (WE' is monotonic)

$$[X \Rightarrow Y] \Rightarrow [\mathrm{WE}'.X \Rightarrow \mathrm{WE}'.Y]$$

Proof. $\quad [X \Rightarrow Y]$
$=$ {Predicate calculus}

$$
\begin{aligned}
& [X \equiv X \wedge Y] \\
\Rightarrow\ & \{\text{Leibniz}\} \\
& [\mathsf{WE}'.X \equiv \mathsf{WE}'.(X \wedge Y)] \\
=\ & \{\mathsf{WE}' \text{ is conjunctive from Proposition 6}\} \\
& [\mathsf{WE}'.X \equiv \mathsf{WE}'.X \wedge \mathsf{WE}'.Y] \\
=\ & \{\text{Predicate calculus}\} \\
& [\mathsf{WE}'.X \Rightarrow \mathsf{WE}'.Y] \qquad \square
\end{aligned}
$$

From Propositions 3 and 4, we know that $\mathsf{WE}'.X$ is solution of equation (7). It remains to show that any solution Z of (7) is stronger than $\mathsf{WE}'.X$.

Proof.

$$
\begin{aligned}
& [Z \Rightarrow X] \wedge \mathsf{exist}.Z \\
=\ & \{\mathsf{exist}.Z \equiv [\mathsf{WE}'.Z \equiv Z] \text{ from Proposition 5}\} \\
& [Z \Rightarrow X] \wedge [\mathsf{WE}'.Z \equiv Z] \\
\Rightarrow\ & \{\mathsf{WE}' \text{ is monotonic from Proposition 7}\} \\
& [\mathsf{WE}'.Z \Rightarrow \mathsf{WE}'.X] \wedge [\mathsf{WE}'.Z \equiv Z] \\
\Rightarrow\ & \{\text{Leibniz}\} \\
& [Z \Rightarrow \mathsf{WE}'.X] \qquad \square
\end{aligned}
$$

A.3 Proof of Formula (15)

$$[X \equiv \mathsf{UNIT}_{\neq}] \vee [\mathsf{WE}.(\mathsf{UNIT}_{=} \vee X) \equiv \mathsf{WE}.X]^{\circledast} \ . \tag{15}$$

Proof.
Assume $\neg[X \equiv \mathsf{UNIT}_{\neq}]$, and prove $[\mathsf{WE}.(\mathsf{UNIT}_{=} \vee X) \equiv \mathsf{WE}.X]$.
First case: $F \neq \mathsf{UNIT}$

$$
\begin{aligned}
& \mathsf{WE}.(\mathsf{UNIT}_{=} \vee X)\ .\ F \\
=\ & \{[\mathsf{WE} = \mathsf{WE}'], \text{ definition of } \mathsf{WE}' \text{ (9)}\} \\
& \langle \forall G : F \triangleleft G : (\mathsf{UNIT}_{=} \vee X)\ .\ G\rangle \\
=\ & \{\text{Definition of } \triangleleft \text{ (4), predicate calculus}\} \\
& \langle \forall H, K : H \surd F \surd K : (\mathsf{UNIT}_{=} \vee X)\ .\ H \circ F \circ K\rangle \\
=\ & \{\text{From hypothesis } F \neq \mathsf{UNIT} \text{ and axiom (3)}, \neg(\mathsf{UNIT}_{=}\ .\ H \circ F \circ K)\} \\
& \langle \forall H, K : H \surd F \surd K : X\ .\ H \circ F \circ K\rangle \\
=\ & \{\text{Definition of } \triangleleft \text{ (4), predicate calculus}\} \\
& \langle \forall G : F \triangleleft G : X\ .\ G\rangle \\
=\ & \{\text{Definition of } \mathsf{WE}' \text{ (9)}, [\mathsf{WE} = \mathsf{WE}']\} \\
& \mathsf{WE}.X\ .\ F
\end{aligned}
$$

Second case: $F = \mathsf{UNIT}$

$$
\begin{array}{ll}
 & \mathrm{WE}.(\mathsf{UNIT}_{=} \vee X)\ .\ \mathsf{UNIT} \\
= & \{\mathsf{exist}.(\mathrm{WE}.Y) \text{ and } \mathsf{exist}.Z \ \wedge\ Z.\mathsf{UNIT} \ \equiv\ [Z \equiv \mathit{true}]\} \\
 & [\mathrm{WE}.(\mathsf{UNIT}_{=} \vee X) \ \equiv\ \mathit{true}] \\
= & \{[\mathrm{WE}.Y] \equiv [Y] \text{ from (13)}\} \\
 & [\mathsf{UNIT}_{=} \vee X \ \equiv\ \mathit{true}] \\
= & \{\text{Predicate calculus: either } \mathsf{UNIT} \text{ satisfies } X \text{ or it doesn't}\} \\
 & [X \equiv \mathit{true}] \vee [X \equiv \mathsf{UNIT}_{\neq}] \\
= & \{\text{Hypothesis } \neg[X \equiv \mathsf{UNIT}_{\neq}]\} \\
 & [X \equiv \mathit{true}] \\
= & \{[\mathrm{WE}.Y] \equiv [Y] \text{ from (13)}\} \\
 & [\mathrm{WE}.X \equiv \mathit{true}] \\
= & \{\mathsf{exist}.Z \ \wedge\ Z.\mathsf{UNIT} \ \equiv\ [Z \equiv \mathit{true}]\} \\
 & \mathrm{WE}.X\ .\ \mathsf{UNIT} \qquad \square
\end{array}
$$