A Complete Algebraic Characterization of Behavioral Subtyping

Gary T. Leavens and Don Pigozzi
TR #96-15a

November 1996, Revised November 1999

Keywords: behavioral subtype, subtyping, behavior, realization, observable equivalence, simulation, abstract data type.

1994 CR Categories: D.3.3 [*Programming Languages*] Language Constructs — Abstract data types; F.3.2 [*Logics and Meanings of Programs*] Semantics of Programming Languages — algebraic approaches to semantics; F.3.2 [*Mathematical Logic and Formal Languages*] Mathematical Logic — model theory.

1991 Mathematics Subject Classification. Primary: 68Q65 Secondary: 68N05, 68N15, 68Q60.

To appear in *Acta Informatica*.
Copyright © Gary T. Leavens and Don Pigozzi, 1996.

Department of Computer Science
226 Atanasoff Hall
Iowa State University
Ames, Iowa 50011-1040, USA

Typeset by $\mathcal{A}_{\mathcal{M}}\mathcal{S}$-TEX

A COMPLETE ALGEBRAIC CHARACTERIZATION OF BEHAVIORAL SUBTYPING

Gary T. Leavens and Don Pigozzi

Iowa State University

November 10, 1999

Abstract. We present a model-theoretic study of correct behavioral subtyping for first-order, deterministic, abstract data types with immutable objects. For such types, we give a new algebraic criterion for proving correct behavioral subtyping that is both necessary and sufficient. This proof technique handles incomplete specifications by allowing proofs of correct behavioral subtyping to be based on comparison with one of several paradigmatic models. It compares a model to a selected paradigm with a generalization of the usual notion of simulation relations. This generalization is necessary for specifications that are not term-generated and that use multiple dispatch. However, we also show that the usual notion of simulation gives a necessary and sufficient proof technique for the special cases of term-generated specifications and specifications that only use single dispatch.

1. Introduction

The problem addressed by this paper is to find a sound and complete algebraic condition for correct behavioral subtyping of first-order, deterministic abstract data types (ADTs) with immutable objects. We treat object-oriented (OO) ADTs in the traditional style with single-dispatched methods, as well as a generalization to multiple dispatch. The results are especially interesting and novel for specifications that are not term-generated and that use multiple dispatch, where for completeness one must use simulations that relate whole environments, not just individual objects. However, we also discuss the special cases of term-generated and single-dispatched methods, where our completeness results are not as surprising, but still novel.

1991 *Mathematics Subject Classification.* Primary: 68Q65 Secondary: 68N05, 68N15, 68Q60.
1994 CR Categories. D.3.3[*Programming Languages*] Language Constructs — abstract data types; F.3.2[*Logics and Meanings of Programs*] Semantics of Programming Languages — algebraic approaches to semantics; F.3.2[*Mathematical Logic and Formal Languages*] Mathematical Logic — model theory.

Key words and phrases. behavioral subtype, subtyping, behavior, realization, observable equivalence, simulation, abstract data type, single dispatch, multiple dispatch, multimethod.

The work of both authors is supported in part by the National Science Foundation grants CCR-9593168 and CCR-9803843. Pigozzi's work was also supported in part by the *Institut d'Estudis Catalans, Centre de Recerca Matemática* (Barcelona) under a grant from the *Ministerio de Educacion y Cienca* of the government of Spain. A preliminary report on this work appeared as [19]. Many thanks to the anonymous referees for their help in improving the paper.

The remainder of this introduction gives some background and motivation, an overview of the techniques and results, and a brief comparison to the most important related work.

Motivation. In reasoning about OO programs that use subtyping and dynamic dispatch, one needs a way to deal with subtype polymorphism. The problem is that expressions of a type T may denote objects of any subtype of T, and thus messages sent to the result of such an expression may invoke different pieces of code with potentially different specifications and behavior. One way to deal with this problem is to do a case analysis for each subtype of T. However, this is quite expensive, and requires the reasoning process to be repeated each time a new subtype of T is added to the program.

A better way of dealing with subtyping and dynamic dispatch is to use "supertype abstraction" [22]. In supertype abstraction, one assumes that instances of each subtype of a given type T obey the specification of T's methods; thus one reasons by ignoring the possibility of subtyping, and instead uses the static types of expressions and the specifications associated with those static types. Such reasoning relies on a type system to guarantee that each expression of type T can denote, at run-time, only objects of subtypes of T. Hence, the supertype T's specification is used in reasoning about all objects of its subtypes.

From the description above, one can see that supertype abstraction is a valid reasoning technique only if its main assumption is valid. That is, supertype abstraction is valid only if objects of each subtype of a given type, T, obey the specification of T's methods when manipulated as if they were objects of type T. If an object of a subtype of T did not obey the specification of T's methods, then its behavior would be *surprising* according to the specification of T. Since we wish supertype abstraction to be valid, we use this notion as a test of the soundness of a definition of correct behavioral subtyping [21].

Simplifying Assumptions. One way to study correct behavioral subtyping would be to define a specification language, a programming language, and a verification logic, and then to look at the validity of verification under definitions of correct behavioral subtyping (as in [22]). However, such a study is quite involved, since two languages and a verification logic have to be described, and tied together.

To avoid such complications, and to allow more mathematical tools to be brought to bear on the problem of completeness, we start by abstracting away the specification language. Instead of describing the presentation form of specifications, we use their meaning. For us, the meaning of a specifications of several ADTs is a class of algebraic models. Each such model is a mathematical abstraction of one way of implementing each of the types specified. In particular each type T comes equipped with a *carrier set* that models objects having exactly type T. In the following, whenever we say "specification," what is meant is a class of algebraic models.

To simplify our study further we do not consider mutation of object states. Although mutation is important in practice for OO programs, for this first study of completeness we wish to avoid the semantic complications related to locations, object identities, circular objects, and especially aliasing. This simplification allows us to simplify the treatment of the programming language, since we do not have

to consider statements with side-effects. In particular, a sequence of messages sent to various objects can be modeled by an expression. In the jargon of algebra, such expressions are called *terms*; we also call them *procedures*.

Finally, to avoid the complications of a verification logic, we directly compare the results of sequences of messages sent to different objects (perhaps in different algebraic models). This leads to the notion of behavior.

Behavior and Observations. A specification of an abstract data type allows for many different implementations. Each of these implementations can use distinct data structures and algorithms. The internal state of the object's data structures is therefore hidden. What the specification describes about each such object is its *behavior*: the results of sequences of messages sent to the object. Of course, if the result of some message is itself an object of an abstract type, then its internal state will also be hidden and hence not useful for determining the behavior of the object receiving the message. Therefore, we focus attention on messages that return a value of some pre-defined set of *visible types*, such as `Bool`. Such types are called "visible" because they can be used in both input and output from programs.

Terms having a visible type are *observations*. An observation observes the objects denoted by its free variables, which can be objects of nonvisible, user-defined ADTs. An observation whose free variables have visible types can be considered to be a *program*, because it both inputs and outputs data of visible type. However, because the visible types are not user-defined ADTs, programs are not useful for observing existing objects of user-defined ADTs. More useful are observations that are not programs.

Such observations, terms that have inputs that are nonvisible types, are critical to the study of behavioral subtyping. Recall that supertype abstraction is valid only if objects of each subtype of a given type, T, obey the specification of T's methods, when manipulated as if they were objects of type T. This can be formalized using observations that take inputs of type T. With our simplifying assumptions, the *expected visible behavior* of a type T can be codified as the responses of objects that have exactly type T to a given set of observations that take objects of type T as inputs. If an object of some subtype of T gives a result that is not possible for an object having exactly type T, then its visible behavior is surprising.

In the technical material below, we make use of a notion of comparative behavior. Suppose o_1 and o_2 are objects of some subtype of T. Then o_1 and o_2 have the *same visible behavior* if, for every observation obs that takes T as an argument, $\mathrm{obs}(o_1)$ and $\mathrm{obs}(o_2)$ are the same. In this setting, the visible behavior of objects of a subtype of T is compared with the expected visible behavior of T using objects that are of exactly type T. The expected visible behavior of T is found by looking at all such objects in all models and all observations that take T as an argument. If one can match every object of each subtype of T with some object in the carrier set of type T with the same visible behavior, then the visible behavior of each subtype is as expected, which is correct behavioral subtyping.

It follows that typed homomorphic relations between models of ADTs can be used as part of a technique for proving correct behavioral subtyping [16]. This technique requires one to pick for a given algebraic model, $\boldsymbol{A}$, another model, $\boldsymbol{B}$, such that for each type T, there is a homomorphic relation between $\boldsymbol{A}$ and $\boldsymbol{B}$ that

relates each object of some subtype of T in $\boldsymbol{A}$ to some object having exactly type T in $\boldsymbol{B}$. The relation is homomorphic in the sense that each instance method must preserve the relation at each type. A homomorphic relation with the additional property that it can relate data elements of visible type only to themselves is called a *nominal simulation* of $\boldsymbol{B}$ by $\boldsymbol{A}$. The construction of nominal simulations gives a proof technique for correct behavioral subtyping that is more useful than looking at every observation.

The ability to pick a model, $\boldsymbol{B}$, different from $\boldsymbol{A}$ allows one to work with ADT specifications that are incomplete in the sense that they may have several nonisomorphic models. The problem this solves is that in $\boldsymbol{A}$, objects in the carrier set of a type T may not happen to have the behavior required to be simulated by objects of T's subtypes in $\boldsymbol{A}$.

Soundness and Completeness. Using the criteria of "no surprising visible behavior", we can say that a technique for proving behavioral subtyping is *sound* if, whenever it certifies a subtype relation as correct, then the visible behavior of subtype values, when manipulated as if they were supertype values, will not be surprising. Such a technique is *complete* if whenever subtypes cannot exhibit surprising visible behavior, then it can certify the specified subtype relation as correct.

Techniques for proving correct behavioral subtyping have been studied by several authors [1,3,6,7,16,21,22,23]. While most of these authors have studied the soundness of their techniques, to the best of our knowledge no others have studied their completeness.

In our earlier work we showed that the use of nominal simulations as described above is sound [16]. However, it turns out that this technique is only complete for term-generated specifications and for specifications that do not use multiple dispatch.

Term Generated Specifications. The above notions of behavior do not involve the creation of the objects being observed. One reason for this is that subtype objects contain different information than objects of their supertypes (often more information), and hence they are created using different operations. For example, in an OO language even if `IntSet` has `Interval` as a subtype, one might create an `IntSet` object using a syntax such as `new IntSet()`, but the syntax for creating an `Interval` object might be `new Interval(3,7)`. However, subtyping ensures that `Interval` objects, once created, can respond to the same set of messages (i.e., the same instance protocol) as `IntSet` objects. Thus a procedure that can observe an `IntSet` object can observe an `Interval` object as well.

A specification is *term-generated* if all possible objects can be created using terms. Because the behavior of existing objects, not object creation, is at the heart of behavioral subtyping, it is often convenient to deal with specifications that are not term-generated. For example, a library of OO ADTs typically includes a type `Collection` that is not term-generated. The type `Collection` specifies operations to test membership, and return the size of a collection, but does not specify any particular way to create collections. `Collection` will typically have subtypes like `Set`, `Bag`, `List`, and `Array`. Although these subtypes will typically have primitive constructors, because `Collection` has no primitive constructors, the algebras in such a specification will generally not be term-generated.

Specifications of types that are not term-generated are also often used to describe type parameters. For example, in a polymorphic type such as `Set[T]`, the specification of the type `T`, would simply call for an `eq` operation—there is no need to be able to create objects of type `T` within `Set[T]`'s operations.

Single vs. Multiple Dispatch. Traditional OO languages, such as Smalltalk, C++, and Java use single dispatch to find the code to run in response to a message send. For example, in Java one would write an expression such as `myIntSet.insert(3)` to produce a new `IntSet` containing the elements of `myIntSet` as well as the number 3. In such an expression, `myIntSet` is the *receiver* of the message. The dynamic type of the object that the receiver denotes is used to find a piece of code to run in response to the message. Hence dynamic dispatch in a language with single dispatch does not consider the dynamic type of any arguments other than the receiver.

However, single dispatch creates a number of problems for OO languages. For example, if a class has methods that take additional arguments of the same type, so-called "binary methods" [2], then its subclasses cannot safely be considered subtypes. Various design patterns such as the "visitor pattern" are difficult to write in a language that only has single dispatch.

For these reasons, some OO languages, such as CLOS, Dylan, and Cecil, use a more general technique for doing dynamic dispatch: multiple dispatch. (See Castagna's book [5] for introductory material and other references.) In multiple dispatch the piece of code invoked, called a *multimethod*, can depend on the run-time type of all arguments. For example, with multiple dispatch one can write `eq(arg1, arg2)` and have the code executed depend on the run-time types of both `arg1` and `arg2`. This makes for a more natural encoding of operations such as equality tests, addition, union, and so on. Since in a multiple dispatch language one can make the dispatching depend on any subset of the actual arguments, multiple dispatch is easily able to simulate single dispatch [20].

However, our results show that multiple dispatch can add significant complications to reasoning. In particular, for specifications that are not term-generated, and that use multiple dispatch, the technique of using nominal simulations to prove behavioral subtyping is sound but not complete.

One way to understand this is to think of multiple dispatch as being single dispatch on tuples of objects [15]. In a language with multiple dispatch, the analog of the behavior of an object, in general, is the behavior of a tuple of objects. One reason for the necessity of this generalization is that some objects cannot be observed in isolation. This will be the case if the only methods that can observe such objects are multimethods. (If, on the other hand, the specification is term-generated, then this problem cannot occur.)

To investigate the behavior of tuples of objects from the perspective of a particular tuple of types, it is convenient to introduce the notion of an environment. An environment maps typed variable names to objects; to allow subtyping, the objects may, in general, be of any subtype of the corresponding variable's type. The typing of variables is recorded in a *type context*, which is a map of variable names to types. A type context can be thought as the type of an environment. Given a type

context, H, and an environment $\rho : H$, it must be that for each $x : T$ in H, $\rho(x)$ has a type that is a subtype of T.

The behavior of an environment of type H is the set of all results of observations that type check against the typings recorded in H. As before, we also find it useful to compare the behavior of two environments of the same type, forming a comparative behavior relation indexed by type contexts.

Expected visible behavior for languages with multiple dispatch relies on the concept of a nominal environment. An environment, $\rho : H$, is *nominal* if whenever a variable x has type T in H, then ρ maps x to an object in the carrier set of type T (and not just in the carrier set of some subtype of T). The visible behavior of objects in a nominal environment is thus part of the expected visible behavior of the types of its variables, and the visible behavior of all nominal environments represents the expected visible behavior of the types in a specification.

The generalization of a homomorphic relation on objects to the setting with multiple dispatch is a homomorphic relation on environments. A sound and complete general technique for proving correct behavioral subtyping, which works for specifications that may not be term-generated, is as follows. This technique requires one to pick for a given algebraic model, $\boldsymbol{A}$, another model $\boldsymbol{B}$, such that for each type context H, there is a generalized simulation between $\boldsymbol{A}$ and $\boldsymbol{B}$ that relates each environment of type H over $\boldsymbol{A}$ to some nominal environment of type H over $\boldsymbol{B}$.

Our results show more than this, however. By using the dual of the construction of a comparative behavior relation from our earlier work [17], we show how to construct the appropriate generalized simulation to check the correctness of behavioral subtyping in the general case. We also prove completeness results for the more specific cases of term-generated specifications and specifications without multiple dispatch.

Related Work. The most important previous work that describes comparisons of algebraic models is Schoett's [26]. Schoett studied the problem of when a partial algebra $\boldsymbol{A}$ can be used in place of a paradigm, a partial algebra $\boldsymbol{B}$, without exhibiting surprising behavior. He argues that this will be assured if the two algebras are behaviorally equivalent in the sense that all programs run in the two algebras have the same output. Because of the problem he studied, he assumed that only values of visible types were legitimate input and output for observations. Schoett proved that the existence of a homomorphic relation between $\boldsymbol{A}$ and $\boldsymbol{B}$ that is the identity on visible types is both necessary and sufficient for the behavior of $\boldsymbol{A}$ to be equivalent to the behavior of $\boldsymbol{B}$.

Because Schoett's technique relies on visible inputs, it cannot, in general, settle questions of correctness for ADTs that are not term-generated. In contrast, our techniques allow nonvisible data as input, and so are more suited to the study of behavioral subtyping in OO languages. This difference allows our techniques to work even for ADTs that have no term-generated values. This mimics the way in which one might test "abstract" types in OO programming: first create the objects, and then pass them to the test procedure.

Outline. In what follows, we first describe the mathematical background from [17], extending it with subtyping. In Section 3 we describe the notions of standard and generalized simulations that are the core of our characterization of behavioral subtyping. In Section 4 we extend the notions of behavior and realization from [17], again extending them with subtyping. In Section 5 we give the definition of correct behavioral subtyping and prove the soundness and completeness of our algebraic characterization. Finally in Section 6 we offer some discussion and conclusions.

2. Preliminaries

The syntactic interface of a collection of ADTs is formally described by a signature. These signatures allow for dynamic overloading of operations, as in an OO language with multimethods. Our framework for subtyping is a generalization of Reynold's category-sorted algebras [24]. Let $\mathbb{N} = \{0, 1, 2, \ldots\}$ be the set of natural numbers.

Definition 2.1 (signature with subtyping). A *signature with subtyping* $\Sigma = \langle TYPE, \leq, VIS, OP, ResType \rangle$ consists of:

(i) A nonempty set $TYPE$ of *types*.

(ii) A preordering $\leq$ of $TYPE$. S is called a *subtype* of T and T is a *supertype* of S if $S \leq T$.

(iii) A nonempty subset $VIS \subseteq TYPE$ of *visible types* such that each $V \in VIS$ is minimal with respect to the preordering; i.e., if $V \in VIS$ and $T \leq V$, then $T = V$.

(iv) An $\mathbb{N}$-indexed family $OP = \langle OP_n : n \in \mathbb{N} \rangle$ of *operation symbols*, where OP_0 is nonempty. OP_n is the set of operation symbols of rank n.

(v) A $\mathbb{N}$-indexed family $ResType = \langle ResType_n : n \in \mathbb{N} \rangle$ of partial functions with $ResType_n \colon OP_n \times TYPE^n \to TYPE$ for each $n \in \mathbb{N}$. $ResType$ gives the nominal result type for given operation and argument types. But an interpretation of the operation can yield values in a subtype of the nominal result type.

Each of the functions $ResType_n : OP_n \times TYPE^n \to TYPE$ is monotonic in its second argument in the following sense. Assume $g \in OP_n$, $\langle T_1, \ldots, T_n \rangle \in TYPE^n$, and $\langle S_1, \ldots, S_n \rangle \in TYPE^n$. If $ResType(g, \langle T_1, \ldots, T_n \rangle)$ is defined and $S_1 \leq T_1, \ldots, S_n \leq T_n$, then $ResType(g, \langle S_1, \ldots, S_n \rangle)$ is also defined and

$$ResType(g, \langle S_1, \ldots, S_n \rangle) \leq ResType(g, \langle T_1, \ldots, T_n \rangle). \quad \square$$

Example 2.2. As an example of a signature with subtyping and multimethods, consider Σ_E defined as follows, where all other applications of $ResType$ are undefined. (Think of the types `LO` and `Comp` respectively as short for "Linearly Ordered" and "Comparable".)

$$TYPE_E := \{\texttt{Bool}, \texttt{LO}, \texttt{Comp}\} \qquad OP_0 := \{\texttt{false}\}$$
$$\leq_E := \{\langle T, T\rangle : T \in TYPE_E\} \cup \{\langle \texttt{Comp}, \texttt{LO}\rangle\} \qquad OP_1 := \{\texttt{not}\}$$
$$VIS_E := \{\texttt{Bool}\} \qquad OP_2 := \{\texttt{or}, \texttt{leq}, \texttt{eq}\}$$
$$OP_n := \{\}, \text{ for } n > 2$$

$$ResType_0(\texttt{false}, \langle\rangle) = \texttt{Bool} \qquad ResType_2(\texttt{leq}, \langle \texttt{LO}, \texttt{LO}\rangle) = \texttt{Bool}$$
$$ResType_1(\texttt{not}, \langle \texttt{Bool}\rangle) = \texttt{Bool} \qquad ResType_2(\texttt{leq}, \langle \texttt{LO}, \texttt{Comp}\rangle) = \texttt{Bool}$$
$$ResType_2(\texttt{or}, \langle \texttt{Bool}, \texttt{Bool}\rangle) = \texttt{Bool} \qquad ResType_2(\texttt{leq}, \langle \texttt{Comp}, \texttt{LO}\rangle) = \texttt{Bool}$$
$$ResType_2(\texttt{leq}, \langle \texttt{Comp}, \texttt{Comp}\rangle) = \texttt{Bool}$$
$$ResType_2(\texttt{eq}, \langle \texttt{Comp}, \texttt{Comp}\rangle) = \texttt{Bool}. \quad \square$$

A signature with subtyping is said to be *discrete* if the preordering is discrete, i.e., if $S \leq T$ only if $S = T$. The above example is not discrete. By the *discrete transform of* Σ, in symbols $\widehat{\Sigma}$, we mean the signature obtained from Σ by replacing the partial ordering $\leq$ of types by the identity relation. For example $\widehat{\Sigma_E}$ is the same as Σ_E, except that its preordering does not relate `Comp` to `LO`. The signatures without subtyping considered in [17] can be viewed as signatures with discrete subtyping.

To simplify notation we usually write $g \in OP$ as shorthand for $g \in \bigcup_{n\in\mathbb{N}} OP_n$; for the same reason the subscript on $ResType_n$ is normally omitted. We also write $\vec{T} \leq \vec{S}$ as shorthand for $T_1 \leq S_1, \ldots, T_n \leq S_n$, and similar vector abbreviations are used without further explanation.

Operation symbols in OP_0 are called *constants*. For example, `true` is a constant.

Suppose $g \in OP_n$, $\vec{T} \in TYPE^n$, and $S \in TYPE$. The pair $\langle \vec{T}, S\rangle$ is called an *admissible type of* g if $ResType(g, \vec{T})$ is defined and $S = ResType(g, \vec{T})$; admissible types are normally written $\vec{T} \to S$. In this case $\vec{T}$ is called an *admissible type domain of* g. In the signature Σ_E of Ex. 2.2, $\langle \texttt{LO}, \texttt{LO}\rangle \to \texttt{Bool}$ is an admissible type of `leq`; thus $\langle \texttt{LO}, \texttt{LO}\rangle$ is an admissible type domain of `leq`. Note that due to dynamic overloading, an operation symbol may have many admissible type domains. Note also that the condition on subtyping in Def. 2.1 says that, if $\vec{T}$ is an admissible type domain of g, then so is $\vec{S}$ whenever $\vec{S} \leq \vec{T}$. Thus in the example above, $\langle \texttt{Comp}, \texttt{LO}\rangle$ is also an admissible type domain of `leq`, as required because $\langle \texttt{Comp}, \texttt{LO}\rangle \leq_E \langle \texttt{LO}, \texttt{LO}\rangle$.

If g is a constant, we identify $\langle\rangle \to S$ with S. The operation g is *useless* if its set of admissible types is empty, i.e., if $ResType(g, \vec{T})$ is undefined for all $\vec{T} \in TYPE^n$. We assume Σ contains no useless operations.

In the sequel we review some definitions of [17] relevant to the present paper and extend them to signatures with subtyping by applying them to the discrete transform. Definitions, theorems, lemmas, etc. that do not mention the preorder $\leq$, and thus carry over directly from [17], will be indicated by a superscript "D". Keep in mind that discrete signatures can be considered as a special class of signatures with subtyping.

With each signature we associate a unique subsignature of visible types for the purpose of defining observations over Σ. Note that by the condition on subtyping in Def. 2.1(vii) the preordering of VIS induced by the preordering of $TYPE$ is discrete (by definition) and hence Σ_{VIS} can be viewed as a signature without subtyping.

Definition 2.3$^{\mathrm{D}}$ (visible subsignature). Let $\Sigma = \langle TYPE, \leq, VIS, OP, ResType \rangle$ be a signature with subtyping. The *visible subsignature* of Σ,

$$\Sigma_{VIS} = \langle TYPE|_{VIS}, =, VIS, OP|_{VIS}, ResType|_{VIS} \rangle,$$

is defined as follows. $TYPE|_{VIS} = VIS$ and $OP|_{VIS}$ is the set of all operations in OP whose restriction to VIS is nontrivial. More precisely, $g \in OP|_{VIS,n}$ iff $g \in OP_n$ and g has at least one admissible type of the form $\vec{T} \to S$ with $T_1, \ldots, T_n, S \in VIS$. Finally, for $g \in OP_n$ and $\vec{V} \in VIS^n$,

$$ResType|_{VIS}(g, \vec{V}) = \begin{cases} ResType(g, \vec{V}), & \text{if } ResType(g, \vec{V}) \in VIS \\ \text{undefined}, & \text{otherwise.} \end{cases} \quad \square$$

The notion of signature we use here models "multimethods"—the dynamic dispatch of an operation determined by more than one of its arguments. To allow us to study OO languages that do not allow multimethods, we use the following restricted notion of signature. While perhaps overly strong, it does rule out multiple dispatch.

Definition 2.4$^{\mathrm{D}}$ (signature with unary methods). A signature Σ (with or without subtyping) is said to *have only unary methods* if, for every $n \in \mathbb{N}$ and every $g \in OP_n$, if $T_0, \ldots, T_{n-1}$ is a type domain of g, then $T_i \notin VIS$ for at most one $i < n$.

Consider the visible subsignature $(\Sigma_E)_{VIS}$ of the signature Σ_E of Ex. 2.2. Its only type is `Bool` (i.e., $TYPE_E|_{VIS} := \{\texttt{Bool}\}$), and its operations symbols are the symbols for the Boolean operations `false`, `not`, and `or`.

We assume a countably infinite set VAR of variable symbols. *Σ-terms* are formed from a signature, Σ, in the usual way. That is, every variable and constant is a term, and, if $g \in OP_n$ (with $n \geq 1$) and $t_1, \ldots, t_n$ are terms, then $g(t_1, \ldots, t_n)$ is a term. A *ground term* contains no variables, for example, `or(false,false)` is a ground Σ_E-term. We write $t(x_1, \ldots, x_n)$ for a term t when we want to indicate that the variables actually occurring in t must be in the list $x_1, \ldots, x_n$. In this context $t(s_1, \ldots, s_n)$ denotes the result of simultaneously substituting the terms $s_1, \ldots, s_n$ respectively for $x_1, \ldots, x_n$. For signatures with a Boolean type we use that abbreviations `true` for the ground term `not(false)` and `and`(t_1, t_2) for `not(or(not(`t_1`),not(`t_2`)))`.

The following definitions of type context, and the type inference rules, are quoted from [17]. They are applied to the discrete transform and unaffected by the presence of subtyping.

Definition 2.5$^{\mathrm{D}}$ (type context). Let Σ be a signature with subtyping. A finite set, H, of the form $\{\langle x_1, T_1\rangle, \ldots, \langle x_n, T_n\rangle\}$, where $x_1, \ldots, x_n$ are distinct variables and $T_1, \ldots, T_n \in \mathit{TYPE}$, is called a *type context*; i.e., a type context is a finite function from variables to types. The set of variables $\{x_1, \ldots, x_n\}$ of H is denoted by $\mathit{Dom}(H)$ and T_i is denoted by $H(x_i)$. H is *visible* if the type of every variable in H is visible. The set of all type contexts is denoted by TCON and the set of all visible type contexts by $\mathit{TCON}|_{\mathit{VIS}}$. K is a *subcontext* of H if $\mathit{Dom}(K) \subseteq \mathit{Dom}(H)$ and $K(x) = H(x)$ for all $x \in \mathit{Dom}(K)$. □

The type inference rules for terms are given below.

(ident) $$\Sigma; H \vdash x : T, \quad \text{if } H(x) = T,$$

(op-call) $$\frac{\Sigma; H \vdash \vec{t} : \vec{T}}{\Sigma; H \vdash g(\vec{t}) : S}, \quad \text{if } \mathit{ResType}(g, \vec{T}) = S.$$

When $\Sigma; H \vdash t : T$ holds, T is called the *nominal* or *static H-type of t*; it is unique. Thus using the signature Σ_E of Ex. 2.2, $\Sigma_E; \{\langle x_1 : \mathtt{L0}\rangle\} \vdash \mathtt{leq}(x_1, x_1) : \mathtt{Bool}$, and thus the nominal $\{\langle x_1 : \mathtt{L0}\rangle\}$-type of $\mathtt{leq}(x_1, x_1)$ is $\mathtt{Bool}$. Note that if $\Sigma; H \vdash t(x_1, \ldots, x_n) : T$, then $\Sigma; K \vdash t : T$ for every type context K such that $H|_{\{x_1, \ldots, x_n\}} \subseteq K$.

We say that t *is well H-typed* if it has a H-type. When Σ is clear from context we write $H \vdash t : T$. When the type context H is also clear we may speak of t being "well-typed," and of T being the "nominal" or "static" type of t. We will denote the extended type context $H \cup \{\langle x, T\rangle\}$ by $H, x : T$. We further streamline notation by using the expression "$t : T$" when referring to a term t, with the understanding that this automatically entails the assumption t is well-typed and of type T.

It is an easy matter to verify the following lemma by induction on the structure of terms; use the fact that *ResType* is monotonic in its second argument.

Lemma 2.6. *Let H be a type context and let $t(\vec{x} : \vec{S}) : T$ be a well H-typed term, and let $\vec{s} : \vec{U}$ be a sequence of well H-typed terms such that $\vec{U} \leq \vec{S}$. Then $t(\vec{s})$ is well H-typed and $H \vdash t(\vec{s}) : W$ for some type W such that $W \leq T$. Moreover, if T is a visible type, then $W = T$.* □

Mappings between type contexts play an important role in our theory.

Definition 2.7$^{\mathrm{D}}$ (context homomorphism, homomorphic pre-image). Let H and K be type contexts. A mapping $h : \mathit{Dom}(K) \to \mathit{Dom}(H)$ is said to be a *context homomorphism of K to H* if $K \vdash x : T$ implies $H \vdash h(x) : T$ for every $x \in \mathit{VAR}$. K is called the *pre-image* of H under h. □

For example, if $H = \{x : \mathtt{Bool}, y : \mathtt{L0}\}$ and $K = \{x : \mathtt{L0}, y : \mathtt{Bool}, z : \mathtt{Bool}, \}$, and $h(x) = y$ and $h(y) = h(z) = x$, then h is a context homomorphism of K to H, and K is the pre-image of H under h.

If h is a context homomorphism of K to H and $x_1, \ldots, x_n \in \mathit{Dom}(K)$, then for every term $t(x_i, \ldots, x_n)$ and type T

$$K \vdash t(x_1, \ldots, x_n) : T \quad \text{iff} \quad H \vdash t\big(h(x_1), \ldots, h(x_n)\big) : T.$$

In [17] the implementations of an ADT specification without subtyping are modeled by algebras. Here we enrich the algebras by an ordering of type domains to model implementations of ADT specifications with subtyping [16].

Definition 2.8 (Σ-algebra, nominal Σ-algebra). Let Σ be a signature with subtyping. A *Σ-algebra* $\boldsymbol{A} = \langle A, \{ g^{\boldsymbol{A}} : g \in OP \} \rangle$ consists of the following:

(i) A $TYPE$-indexed family of sets $A = \langle A_T : T \in TYPE \rangle$ called the *carrier of* $\mathbf{A}$.

(ii) A partial function $g^{\boldsymbol{A}} \colon \left(\bigcup_{S \in TYPE} A_S\right)^n \to \bigcup_{S \in TYPE} A_S$ for each $n \in \mathbb{N}$ and $g \in OP_n$, called the *interpretation* of g, with the property that, for every admissible type $\langle T_1, \ldots, T_n \rangle \to S$ of g and every $\langle a_1, \ldots, a_n \rangle \in A_{T_1} \times \cdots \times A_{T_n}$, $g^{\boldsymbol{A}}(a_1, \ldots, a_n)$ is defined and contained in $\bigcup_{U \leq S} A_U$.

A Σ-algebra $\boldsymbol{A}$ is *nominal* if for every $g \in OP_n$, admissible type $\langle T_1, \ldots, T_n \rangle \to S$ of g, and $\langle a_1, \ldots, a_n \rangle \in A_{T_1} \times \cdots \times A_{T_n}$, $g^{\boldsymbol{A}}(a_1, \ldots, a_n) \in A_S$. $\square$

An algebra will be called *discrete* if its signature is discrete. Note that every discrete algebra is automatically nominal.

We use the following abbreviations: $\widehat{A}_T$ for $\bigcup_{S \leq T} A_S$ and, if $\vec{T} = T_1 \cdots T_n$, $A_{\vec{T}}$ for $A_{T_1} \times \cdots \times A_{T_n}$ and $\widehat{A}_{\vec{T}}$ for $\widehat{A}_{T_1} \times \cdots \times \widehat{A}_{T_n}$. Expressed in terms of this streamlined notation, the critical property, with respect to the subtyping, of the interpretations of the operations in a Σ-algebra is that $g^{\boldsymbol{A}}(A_{\vec{T}}) \subseteq \widehat{A}_S$ for each admissible type $\vec{T} \to S$ of g; $\boldsymbol{A}$ is nominal if $g^{\boldsymbol{A}}(A_{\vec{T}}) \subseteq A_S$.

Let Σ be a signature with subtyping and $\boldsymbol{A}$ be a Σ-algebra. An element a of $\boldsymbol{A}$ is said to be of *dynamic type* T if $a \in A_T$, and of *virtual type* T if $a \in \widehat{A}_T$, i.e., if it is of dynamic type S for some subtype S of T. Note that, for every $g \in OP_n$ and every admissible type domain $T_1 \cdots T_n$ of g, if a string $a_1 \cdots a_n$ of elements of $\boldsymbol{A}$ is of dynamic type $T_1 \cdots T_n$, then $g^{\boldsymbol{A}}(a_1, \ldots, a_n)$ must be of virtual type S, where S is the nominal type of g under $T_1 \cdots T_n$. If $g^{\boldsymbol{A}}(a_1, \ldots, a_n)$ is always of dynamic type S in this situation, then $\boldsymbol{A}$ is nominal.

Example 2.9[1] Let Σ_E be the signature of Ex. 2.2. The Σ_E-algebra $\boldsymbol{E}$ is defined as follows:

$$E_{\texttt{Bool}} := \{tt, ff\} \qquad E_{\texttt{LO}} := \mathbb{N}$$
$$E_{\texttt{Comp}} := \mathbb{Z} = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$$

$$\texttt{false}^{\boldsymbol{E}}(\langle\rangle) = ff \qquad \texttt{leq}^{\boldsymbol{E}}(\langle n_1, n_2 \rangle) = n_1 \leq n_2, \quad \text{for all } n_1, n_2 \in \mathbb{N}$$
$$\texttt{not}^{\boldsymbol{E}}(\langle b \rangle) = \neg b \qquad \texttt{leq}^{\boldsymbol{E}}(\langle i_1, i_2 \rangle) = i_1 \leq i_2, \quad \text{for all } i_1, i_2 \in \mathbb{Z}$$
$$\texttt{or}^{\boldsymbol{E}}(\langle b_1, b_2 \rangle) = b_1 \vee b_2 \qquad \texttt{eq}^{\boldsymbol{E}}(\langle i_1, i_2 \rangle) = (i_1 = i_2), \quad \text{for all } i_1, i_2 \in \mathbb{Z}.$$

By our abbreviations, $\texttt{true}^{\boldsymbol{E}}(\langle\rangle) = tt$ and $\texttt{and}^{\boldsymbol{E}}(\langle b_1, b_2 \rangle) = b_1 \wedge b_2$. $\square$

Note that $\widehat{E}_{\texttt{LO}} = E_{\texttt{Comp}} = \mathbb{Z}$ but that $\boldsymbol{E}$ is nominal; in fact, every Σ_E-algebra is nominal.

[1] By "$\leq$" in the definitions of Ex. 2.9 we mean the usual ordering of natural numbers, not the subtype ordering of the signature.

We follow Reynolds [24], in contrast to Goguen and Meseguer [9, 12], in not requiring $A_S \subseteq A_T$ when $S \leq T$. However Reynolds handles subtyping by means of an implicit coercion mapping between the domains A_S and A_T when $S \leq T$, and this has essentially the same effect as requiring A_S and A_T to be disjoint when $S \neq T$. We take a middle ground. The domains of distinct types S and T need not be disjoint, nor do they have to be comparable when $S \leq T$, and we do not require coercion functions. But the definitions must be consistent in spite of the dynamic overloading of operators because the operation interpretations are polymorphic. More precisely, if $\vec{T}$ and $\vec{T}'$ are both admissible type domains of g and $\vec{a} \in A_{\vec{T}} \cap A_{\vec{T}'}$, then $g^{\boldsymbol{A}}(\vec{a})$ is uniquely determined by virtue of $g^{\boldsymbol{A}}$ being a partial function. The definition requires in this case that $g^{\boldsymbol{A}}(\vec{a}) \in A_U$ for some $U \leq \mathit{ResType}(g, \vec{T})$ and also that $g^{\boldsymbol{A}}(\vec{a}) \in A_{U'}$ for some $U' \leq \mathit{ResType}(g, \vec{T}')$, and thus that $g^{\boldsymbol{A}}(\vec{a}) \in A_U \cap A_{U'} \subseteq \widehat{A}_{\mathit{ResType}(g,\vec{T})} \cap \widehat{A}_{\mathit{ResType}(g,\vec{T}')}$. This property insures that the discrete transform of $\boldsymbol{A}$, defined below, is in fact well-defined.

For example, in the Σ_E-algebra $\boldsymbol{E}$ defined above, $\texttt{Comp} \leq_E \texttt{LO}$, but $E_{\texttt{Comp}} = \mathbb{Z}$ is neither disjoint nor a subset of $E_{\texttt{LO}} = \mathbb{N}$, in fact $\mathbb{N}$ is a proper subset of $\mathbb{Z}$, so the domain inclusion is opposite from the type ordering. (The reason for this inversion of order will be apparent later.) But the definition of $\texttt{leq}^{\boldsymbol{E}}$ is consistent on the common part of $E_{\texttt{Comp}}$ and $E_{\texttt{LO}}$ since the natural order of $\mathbb{N}$ is a suborder of the natural order of $\mathbb{Z}$.

The key feature of an algebra with subtyping in our sense is the fact that, for any $g \in OP$, if $\vec{a}$ is a sequence of elements of $\boldsymbol{A}$ of dynamic type $\vec{T}$, where $\vec{T}$ is an admissible type domain of g, then $g^{\boldsymbol{A}}(\vec{a})$ can be in any domain whose type is a subtype of the static type S of g for $\vec{T}$, i.e., $g^{\boldsymbol{A}}(\vec{a})$ is only required to be virtually of type S. This would not differ from the case of discrete algebras if each type domain were required to be a subset of the domain of every supertype, as is the case for the order-sorted algebras of Goguen and Meseguer [9, 12]. But it is a definite generalization in our context, and there are many situations where this greater generality seems justified.

Example 2.10. The signature Σ_{II} has the same visible subsignature as Σ_E (Ex. 2.2). Its nonvisible types are `Int`, `IntSet`, and `Interval`. In Σ_{II}, the subtype ordering $\leq_{II}$ is discrete, except that $\texttt{Interval} \leq_{II} \texttt{IntSet}$. The integer type `Int` has the usual constants and operations: 0, 1, $+$, $-$, and $\times$. The set of operations also includes the following.

$$\begin{array}{ll}
\texttt{size} : \langle \texttt{IntSet} \rangle \to \texttt{Int} & \texttt{member} : \langle \texttt{Int}, \texttt{IntSet} \rangle \to \texttt{Bool} \\
\texttt{size} : \langle \texttt{Interval} \rangle \to \texttt{Int} & \texttt{member} : \langle \texttt{Int}, \texttt{Interval} \rangle \to \texttt{Bool} \\
\texttt{choose} : \langle \texttt{IntSet} \rangle \to \texttt{Int} & \texttt{remove} : \langle \texttt{IntSet}, \texttt{Int} \rangle \to \texttt{IntSet} \\
\texttt{choose} : \langle \texttt{Interval} \rangle \to \texttt{Int} & \texttt{remove} : \langle \texttt{Interval}, \texttt{Int} \rangle \to \texttt{IntSet}. \quad \square
\end{array}$$

Example 2.11. The Σ_{II}-algebra $\boldsymbol{II}$ is defined as follows. $II_{\texttt{Bool}} = \{\mathit{ff}\}$, and the operations $\texttt{false}^{\boldsymbol{II}}$, $\texttt{not}^{\boldsymbol{II}}$, and $\texttt{or}^{\boldsymbol{II}}$ are the standard Boolean operations (see Ex. 2.9).

$$II_{\texttt{Int}} := \mathbb{Z} = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\},$$

and $0^{\boldsymbol{II}}$, $1^{\boldsymbol{II}}$, $+^{\boldsymbol{II}}$, $-^{\boldsymbol{II}}$ and $\times^{\boldsymbol{II}}$ have their usual meanings

$$II_{\texttt{IntSet}} := \text{the set of all finite subsets of } \mathbb{Z}$$

$$II_{\texttt{Interval}} := \{\, \langle i, j\rangle : i, j \in \mathbb{Z}, i \le j \,\}$$

$$\texttt{size}^{\boldsymbol{II}}(s) = \begin{cases} |s|, & \text{if } s \in II_{\texttt{IntSet}} \\ j - i + 1, & \text{if } s \in II_{\texttt{Interval}} \text{ and } s = \langle i, j\rangle \end{cases}$$

$$\texttt{choose}^{\boldsymbol{II}}(s) = \begin{cases} \max(s), & \text{if } s \in II_{\texttt{IntSet}} \text{ and } s \neq \{\} \\ 0, & \text{if } s \in II_{\texttt{IntSet}} \text{ and } s = \{\} \\ i, & \text{if } s \in II_{\texttt{Interval}} \text{ and } s = \langle i, j\rangle \end{cases}$$

$$\texttt{member}^{\boldsymbol{II}}(k, s) = \begin{cases} \mathit{tt}, & \text{if } s \in II_{\texttt{IntSet}} \text{ and } i \in s \\ \mathit{tt}, & \text{if } s \in II_{\texttt{Interval}},\ s = \langle i, j\rangle, \text{ and } i \le k \le j \\ \mathit{ff}, & \text{otherwise.} \end{cases}$$

$$\texttt{remove}^{\boldsymbol{II}}(s, k) = \begin{cases} s \setminus \{k\}, & \text{if } s \in II_{\texttt{IntSet}} \\ \langle i, j\rangle, & \text{if } s \in II_{\texttt{Interval}},\ s = \langle i, j\rangle, \\ & \quad \text{and either } k < i \text{ or } j < k \\ \langle i + 1, j\rangle, & \text{if } s \in II_{\texttt{Interval}},\ s = \langle i, j\rangle, \\ & \quad k = i, \text{ and } i < j \\ \langle i, j - 1\rangle, & \text{if } s \in II_{\texttt{Interval}},\ s = \langle i, j\rangle, \\ & \quad k = j, \text{ and } i < j \\ \{n : i \le n \le j \wedge n \neq k\}, & \text{if } s \in II_{\texttt{Interval}},\ s = \langle i, j\rangle, \\ & \quad \text{and no case above holds.} \end{cases}$$ □

The carrier sets $II_{\texttt{Interval}}$ and $II_{\texttt{IntSet}}$ are disjoint, so $\boldsymbol{II}$, like $\boldsymbol{E}$, is not order-sorted in the sense of [9, 12]. Notice also that $\boldsymbol{II}$ is not nominal, because $\texttt{remove}^{\boldsymbol{II}}$ with an `Interval` argument may return an `Interval` instead of an `IntSet`.

Any Σ-algebra can be transformed into a $\widehat{\Sigma}$-algebra and hence into an algebra over a discrete signature. This transformation is not faithful in the sense that some information is lost and the original algebra cannot in general be recovered from its transform. But the process of transformation is important because it allows us to extend the results of [17], which are concerned with specifications without subtyping, to specifications with subtyping.

Recall that the definition of a Σ-algebra $\boldsymbol{A}$ insures that the definition of the operation $g^{\boldsymbol{A}}$ must be consistent in spite of its polymorphism. Recall also that the types of $\widehat{\Sigma}$, the discrete transform of Σ, are the same as Σ but are discretely ordered. Recall finally that, for every Σ-algebra $\boldsymbol{A}$ and every type T, $\widehat{A}_T = \bigcup_{U \le T} A_U$.

Definition 2.12 (discrete transform algebra). Let Σ be a signature with subtyping and let $\boldsymbol{A} = \langle A, \{ g^{\boldsymbol{A}} : g \in OP \} \rangle$ be a Σ-algebra. The *discrete transform of* $\boldsymbol{A}$ is the $\widehat{\Sigma}$-algebra $\widehat{\boldsymbol{A}} = \langle \widehat{A}, \{ g^{\boldsymbol{A}} : g \in OP \} \rangle$, where $\widehat{A} = \langle \widehat{A}_T : T \in TYPE \rangle$. □

In forming the discrete transform only the type domains change. The operations stay the same because they are polymorphic to begin with. Note that if Σ is discrete, then $\widehat{\boldsymbol{A}} = \boldsymbol{A}$ for every Σ-algebra $\boldsymbol{A}$.

Consider the discrete transform $\widehat{\boldsymbol{II}}$ of the algebra $\boldsymbol{II}$ of Example 2.11. Its carrier set $\widehat{II}$ is defined as follows. $\widehat{II}_{\texttt{IntSet}} = II_{\texttt{IntSet}} \cup II_{\texttt{Interval}}$, and $\widehat{II}_{\texttt{Interval}}$ is the same as $II_{\texttt{Interval}}$.

By definition $\widehat{\boldsymbol{II}}$ is not subtyped, but if it were allowed to inherit the subtyping of $\boldsymbol{II}$ it would be order-sorted in the sense of [9, 12] because $\widehat{II}_{\texttt{Interval}} \subseteq \widehat{II}_{\texttt{IntSet}}$. This applies to all discrete transforms.

Let Σ be a signature with subtyping. Σ-algebras $\boldsymbol{A}$ and $\boldsymbol{B}$ are *isomorphic* (in symbols $\boldsymbol{A} \cong \boldsymbol{B}$) if their discrete transforms are isomorphic is the normal sense, i.e., if there exists a $TYPE$-indexed bijection $h = \langle h_T : T \in TYPE \rangle$ between $\widehat{A} = \langle \widehat{A}_T : T \in TYPE \rangle$ and $\widehat{B} = \langle \widehat{B}_T : T \in TYPE \rangle$ such that, for every $g \in OP_n$, every admissible type $\vec{T} \to S$ of g, and every $\vec{a} \in \widehat{A}_{\vec{T}}$, we have $h_S(g^{\boldsymbol{A}}(\vec{a})) = g^{\boldsymbol{B}}(h_{\vec{T}}(\vec{a}))$. This looser notion of isomorphism is the appropriate one here because of the polymorphism inherent in operation interpretations.

Definitions, theorems, etc. in [17] that are formulated for discrete algebras can be automatically applied to subtyped algebras by applying them to the discrete transforms (after dropping the discrete ordering on the types). Again these are indicated by a superscript "D".

Definition 2.13$^{\mathrm{D}}$ **(*VIS*-reduct).** The *VIS-reduct* of a Σ-algebra $\boldsymbol{A}$ is the Σ_{VIS}-algebra

$$\boldsymbol{A}|_{VIS} = \langle A|_{VIS}, \{ g^{\boldsymbol{A}|_{VIS}} : g \in OP_{VIS} \} \rangle,$$

where $A|_{VIS} = \langle A_V : V \in VIS \rangle$ and $g^{\boldsymbol{A}|_{VIS}}(\vec{a}) = g^{\boldsymbol{A}}(\vec{a})$ for every admissible type domain $\vec{V}$ of g consisting only of visible types and every $\vec{a} \in A_{\vec{V}}$. □

The notion of a *VIS*-reduct will be used in defining the notion of a specification.

For example, the *VIS*-reduct of the Σ_E-algebra $\boldsymbol{E}$ of Ex. 2.9, $\boldsymbol{E}|_{VIS}$, has only the carrier set of the type `Bool` and the interpretations of the operations `false`, `not`, and `or`. Similarly, $\boldsymbol{II}_{VIS} = \boldsymbol{E}_{VIS}$.

Note that $\widehat{\boldsymbol{A}}|_{VIS} = \boldsymbol{A}|_{VIS}$ because Σ_{VIS} is discretely subtyped.

The notion of an environment over an algebra $\boldsymbol{A}$ with subtyping is the same as that of an of environment over $\widehat{\boldsymbol{A}}$ [17, Def. 1.10].

Definition 2.14 (H-environment, nominal, visible). Let Σ be a signature with subtyping and let $\boldsymbol{A}$ be a Σ-algebra and H a type context.

(i)$^{\mathrm{D}}$ By an *environment of type context* H, or more simply an *H-environment*, over $\boldsymbol{A}$ we mean a mapping ρ of the variables of the domain of H into $\bigcup_{S \in TYPE} A_S$ such that $\rho(x)$ is of virtual type T (i.e., $\rho(x) \in \widehat{A}_T$) if $H(x) = T$.

(ii) An H-environment ρ is *nominal* if $\rho(x)$ is of dynamic type T (i.e., $\rho(x) \in A_T$) whenever $H(x) = T$.

(iii)$^{\mathrm{D}}$ An H-environment ρ is *visible* if H is visible. □

The set of all nominal environments of type context H is denoted by $ENV_H^{\boldsymbol{A}}$. The $TCON$-indexed set $\langle\, ENV_H^{\boldsymbol{A}} : H \in TCON \,\rangle$ is denoted by $ENV^{\boldsymbol{A}}$. Because an ordinary environment is just an environment over the discrete transform, the set of all H-environments (not necessarily nominal) is written $ENV_H^{\widehat{\boldsymbol{A}}}$. If H is clear from context we usually refer to an H-environment simply as an *environment*.

For a discrete signature, every environment is nominal. Every visible environment is also nominal.

Example 2.15. For example, suppose H is the following type context over the signature Σ_E of Ex. 2.2. $H(x_1) = \mathtt{Bool}$, $H(x_2) = \mathtt{LO}$, and $H(x_3) = \mathtt{Comp}$. Then the environment ρ_E defined by $\rho_E(x_1) = tt$, $\rho_E(x_2) = -2$, and $\rho_E(x_3) = -3$ is an H-environment over the algebra $\boldsymbol{E}$ of Ex. 2.9.

Note that ρ_E is not nominal, because $\rho_E(x_2) \notin E_{\mathtt{LO}}$. A nominal H-environment would have to map x_2 to some element of $\mathbb{N}$.

Recall that the notion of a nominal environment will be used in defining the notion of expected behavior, and plays key role in our method of verifying the correctness of behavioral subtyping. By comparing ordinary and nominal environments one can determine if data elements from subtype domains behave like nominal elements.

Let ρ be an H-environment and assume $y \notin Dom(H)$. Let $T \in TYPE$ and $a \in \widehat{A}_T$. The $(H, y:T)$-environment that assigns a to y and $\rho(x)$ to each variable x of H will be denoted by $[y \mapsto a]\rho$.

The notions of pre-image, subcontext and isomorphism can be lifted from type contexts to environments in the obvious way. Let ρ be a H-environment. For every context homomorphism $h\colon K \to H$, let $\rho \circ h\colon Dom(K) \to \bigcup_{S \in TYPE} A_S$ be the composition of the two functions ρ and h. Clearly, $\rho \circ h$ is a K-environment; it is called the *pre-image* of ρ under h. If K is a subcontext of H, then the restriction $\langle \rho(x)\colon x \in Dom(K) \rangle$ of ρ to K is denoted by $\rho|_K$; it is called a *subenvironment* of ρ.

Given an algebra $\boldsymbol{A}$ and environment ρ over it, the meaning of a term is defined by recursion on its structure in the usual way [17, Def. 1.11].

Definition 2.16$^{\mathrm{D}}$ **(meaning).** Assume Σ is a signature with subtyping and $\boldsymbol{A}$ a Σ-algebra. Let H be a type context, $t : T$ a well H-typed term, and ρ an H-environment of $\boldsymbol{A}$. The *meaning,* $[\![\, t : T \,]\!]^{\boldsymbol{A}}\rho$*, of* t *under* ρ is defined as follows:

$$[\![\, t : T \,]\!]^{\boldsymbol{A}}\rho = \begin{cases} \rho(x), & \text{if } t \text{ is a variable } x : T \\ g^{\boldsymbol{A}}\big([\![\, s_1 : S_1 \,]\!]^{\boldsymbol{A}}\rho, \ldots, [\![\, s_n : S_n \,]\!]^{\boldsymbol{A}}\rho\big), & \text{if } t \text{ is } g(s_1, \ldots, s_n). \end{cases} \quad \square$$

An easy inductive proof, based on the recursive definition of $[\![\, t : T \,]\!]^{\boldsymbol{A}}\rho$, shows that the meaning of a term t depends only on the meanings of the variables that actually occur in t. Consequently, if $H \vdash t(x_1, \ldots, x_n) : T$, $\rho \in ENV_H^{\widehat{\boldsymbol{A}}}$, and $\rho(x_1) = a_1, \ldots, \rho(x_n) = a_n$, we can write $t^{\boldsymbol{A}}(a_1, \ldots, a_n)$ in place of $[\![\, t : T \,]\!]^{\boldsymbol{A}}\rho$ without ambiguity. Note that $t^{\boldsymbol{A}}(a_1, \ldots, a_n) = t^{\widehat{\boldsymbol{A}}}(a_1, \ldots, a_n)$.

A Σ-algebra is *term-generated* if every element is of the form $t^{\boldsymbol{A}}$ for some ground term t.

Recall that we are not concerned in this paper with the presentation of specifications. Consequently a specification will be identified with the class of models it denotes. It is customary in the theory of hierarchical specification to take the visible part of an ADT as given a priori and consequently beyond the scope of the specification; furthermore, its structure must be completely accessible. For example, the properties of the Booleans are fixed and each Boolean has a fixed name. Following this custom we assume that the visible reduct of every algebra in the specification class is the same and that it is term-generated. These conditions assure that any isomorphism between any two algebras in the class must be the identity on the visible reduct. Such an isomorphism is called a *VIS-isomorphism*, for emphasis.

Definition 2.17 (specification). Let Σ be a signature with subtyping. By a *specification over* Σ we mean any class of Σ-algebras, usually denoted by SPEC, with the following properties:

(i) For all $\boldsymbol{A}, \boldsymbol{B} \in \mathsf{SPEC}$, $\boldsymbol{A}|_{VIS} = \boldsymbol{B}|_{VIS}$.
(ii) If $\boldsymbol{A}|_{VIS}$ is the common visible reduct of all the members of SPEC, then $\boldsymbol{A}|_{VIS}$ is term-generated.
(iii) SPEC is closed under *VIS*-isomorphism; i.e., if $\boldsymbol{A} \in \mathsf{SPEC}$ and $h\colon \boldsymbol{A} \cong \boldsymbol{B}$ such that h is the identity on $\boldsymbol{A}|_{VIS}$, then $\boldsymbol{B} \in \mathsf{SPEC}$. □

A specification SPEC is *term-generated* if each algebra in SPEC is term-generated. A specification SPEC is said to *have only unary methods* if its signature Σ has only unary methods.

We now give two examples. They are specifications over the signatures Σ_E and Σ_{II}, respectively (Exs. 2.2 and 2.10). In both examples the visible reduct of all models will be the same (as they must be by definition of specification) and in fact will coincide with the common visible reduct of $\boldsymbol{E}$ and $\boldsymbol{II}$ (Exs. 2.9 and 2.11). This is the two-element Boolean algebra with the standard operations. Note that this algebra is term-generated. This will be the visible reduct of all algebras we consider in examples from now on.

Example 2.18. As an example of a specification over Σ_E, with the nonvisible types $\mathtt{LO}$ and $\mathtt{Comp}$, we take SPEC_E to be the class of Σ_E-algebras $\boldsymbol{A}$ such that the following hold.

(i) $\mathtt{leq}^{\boldsymbol{A}}$ is a linear order on $\widehat{A}_{\mathtt{LO}} = A_{\mathtt{LO}} \cup A_{\mathtt{Comp}}$.
(ii) For all $a, b \in A_{\mathtt{Comp}}$, if $\mathtt{leq}^{\boldsymbol{A}}(a, b) = tt$ and $\mathtt{leq}^{\boldsymbol{A}}(b, a) = tt$, then $\mathtt{eq}^{\boldsymbol{A}}(a, b) = tt$.
(iii) $\mathtt{eq}^{\boldsymbol{A}}$ is the identity relation on $A_{\mathtt{Comp}}$. (More precisely, it is the characteristic function of the identity relation.)
(iv) $A_{\mathtt{LO}}$ is well-ordered under the linear order $\mathtt{leq}^{\boldsymbol{A}}$, i.e., every nonempty subset of $A_{\mathtt{LO}}$ has a smallest element. □

It is easy to see that this class of Σ_E-algebras is closed under *VIS*-isomorphisms. Note also that $\boldsymbol{E} \in \mathsf{SPEC}_E$.

Example 2.19. For an example of a specification Σ_{II}, with the nonvisible types `Int`, `IntSet`, `Interval`, let SPEC_{II} be the class of Σ_{II}-algebras $\boldsymbol{A}$ satisfying the following conditions.

(i) the integer operations satisfy the Peano axioms.
(ii) $\mathtt{member}^{\boldsymbol{A}}(k,s) = tt$ for only a finite number of $k \in A_{\mathtt{Int}}$, for each $s \in \widehat{A}_{\mathtt{IntSet}} = A_{\mathtt{IntSet}} \cup A_{\mathtt{Interval}}$.
(iii) $\mathtt{size}^{\boldsymbol{A}}(s) = |\{k : \mathtt{member}^{\boldsymbol{A}}(k,s) = tt\}|$, for each $s \in \widehat{A}_{\mathtt{IntSet}}$.
(iv) $\mathtt{member}^{\boldsymbol{A}}(\mathtt{choose}^{\boldsymbol{A}}(s), s) = tt$, for all $s \in \widehat{A}_{\mathtt{IntSet}}$ such that $\mathtt{size}^{\boldsymbol{A}}(s) > 0$.
(v) $\mathtt{choose}^{\boldsymbol{A}}(s)) = \min(\{k : \mathtt{member}^{\boldsymbol{A}}(k,s) = tt\})$, for all $s \in A_{\mathtt{Interval}}$.
(vi) $\mathtt{member}^{\boldsymbol{A}}(k, \mathtt{remove}^{\boldsymbol{A}}(s,k)) = ff$, for all $s \in \widehat{A}_{\mathtt{IntSet}}$.
(vii) $\mathtt{member}^{\boldsymbol{A}}(k', \mathtt{remove}^{\boldsymbol{A}}(s,k)) = \mathtt{member}^{\boldsymbol{A}}(k', s)$, for all $k' \neq k$ and $s \in \widehat{A}_{\mathtt{IntSet}}$.

□

It is easy to see that $\boldsymbol{II} \in \mathsf{SPEC}_{II}$.

Neither of the two examples SPEC_E or SPEC_{II} is term-generated and neither of them has only unary methods.

Definition 2.20 (discrete transform specification). Let Σ be a signature with subtyping. For any specification SPEC over Σ define $\widehat{\mathsf{SPEC}} = \{\, \widehat{\boldsymbol{A}} : \boldsymbol{A} \in \mathsf{SPEC} \,\}$. $\widehat{\mathsf{SPEC}}$ is called the *discrete transform* of SPEC. □

Lemma 2.21. *Let* SPEC *be a specification over* Σ. *Then* $\widehat{\mathsf{SPEC}}$ *is a specification over* $\widehat{\Sigma}$.

Proof. It is easy to see that $\widehat{\mathsf{SPEC}}$ is closed under *VIS*-isomorphisms, because SPEC is closed under *VIS*-isomorphisms. More precisely, suppose $\boldsymbol{A} \in \mathsf{SPEC}$ and $\widehat{\boldsymbol{A}} \cong \boldsymbol{B}$ with $\boldsymbol{B}$ a $\widehat{\Sigma}$-algebra. Let $h\colon \widehat{\boldsymbol{A}} \to \boldsymbol{B}$ be a *VIS*-isomorphism. Define the Σ-algebra $\boldsymbol{B}'$ as follows: $B'_T = h(A_T)$ for each $T \in \mathit{TYPE}$ and $g^{\boldsymbol{B}'}(\vec{b}) = h\big(g^{\boldsymbol{A}}(h^{-1}(\vec{b}))\big)$ for each $g \in \mathit{OP}$ and each $\vec{b} \in B'_{\vec{T}}$ such that $\vec{T}$ is an admissible type domain of g. It is easy to check that $\boldsymbol{A} \cong \boldsymbol{B}'$ and $\widehat{\boldsymbol{B}'} = \boldsymbol{B}$. So $\boldsymbol{B} \in \widehat{\mathsf{SPEC}}$ since $\boldsymbol{B}' \in \mathsf{SPEC}$ by the assumption that SPEC is closed under *VIS*-isomorphisms.

Also $\widehat{\boldsymbol{A}}|_{\mathit{VIS}} = \boldsymbol{A}|_{\mathit{VIS}} = \boldsymbol{B}|_{\mathit{VIS}} = \widehat{\boldsymbol{B}}|_{\mathit{VIS}}$ for all $\widehat{\boldsymbol{A}}, \widehat{\boldsymbol{B}} \in \widehat{\mathsf{SPEC}}$. □

Throughout the remainder of the paper, when not explicitly indicated otherwise, it is automatically assumed that Σ is a signature with subtyping and SPEC is a specification over Σ.

3. Simulation

A simulation of one Σ-algebra by another is formalized as a TYPE-indexed binary relation between the carriers of the two algebras that preserves the actions of corresponding operations. We also consider a more general notion of simulation that takes the form of a TCON-indexed binary relation between the TCON-indexed sets of environments of the algebras; recall that TCON denotes the set of all type contexts. A standard relation[2] relates individual elements of the algebras and a

[2] In [16], standard relations that allow for subtyping were called "generalized homomorphic relations." They are different than the generalized homomorphic relations used in this paper.

generalized relation relates environments. The formal definition of these two kinds of relations follows.

Definition 3.1$^{\mathrm{D}}$ **(standard and generalized relations).** Let $\boldsymbol{A}, \boldsymbol{B} \in \mathsf{SPEC}$.

(i) A *standard relation* between $\boldsymbol{A}$ and $\boldsymbol{B}$ is a $TYPE$-indexed family of binary relations $\mathcal{R} := \langle \mathcal{R}_T \colon T \in TYPE \rangle$ such that $\mathcal{R}_T \subseteq \widehat{A}_T \times \widehat{B}_T$ for every $T \in TYPE$.

(ii) A *generalized relation* between $\boldsymbol{A}$ and $\boldsymbol{B}$ is a $TCON$-indexed family of binary relations $\mathcal{G} := \langle \mathcal{G}_H \colon H \in TCON \rangle$ such that $\mathcal{G}_H \subseteq ENV_H^{\widehat{\boldsymbol{A}}} \times ENV_H^{\widehat{\boldsymbol{B}}}$ for every $H \in TCON$. $\square$

By the *Cartesian product* of the two $TYPE$-indexed sets $A = \langle\, A_T : T \in TYPE \,\rangle$ and $B = \langle\, B_T : T \in TYPE \,\rangle$ we mean the $TYPE$-indexed set $\langle\, A_T \times B_T : T \in TYPE \,\rangle$; it is denoted by $A \times B$. Similarly, $ENV^{\boldsymbol{A}} \times ENV^{\boldsymbol{B}} = \langle\, ENV_H^{\boldsymbol{A}} \times ENV_H^{\boldsymbol{B}} : H \in TCON \,\rangle$. Thus $\mathcal{R}$ is a standard relation between $\boldsymbol{A}$ and $\boldsymbol{B}$ if $\mathcal{R} \subseteq A \times B$, and $\mathcal{G}$ is a generalized relation if $\mathcal{G} \subseteq ENV^{\boldsymbol{A}} \times ENV^{\boldsymbol{B}}$.

$\mathcal{R}$ is a standard relation between $\boldsymbol{A}$ and $\boldsymbol{B}$ iff it is a standard relation between the discrete $\widehat{\Sigma}$-algebras $\widehat{\boldsymbol{A}}$ and $\widehat{\boldsymbol{B}}$ in the sense of [17, Def. 2.1]. It can be identified with the indexed subsets of the Cartesian product $\widehat{A} \times \widehat{B}$. Similar remarks hold for generalized relations.

The empty relation ($\mathcal{R}_T = \emptyset$ for all T) and the universal relation ($\mathcal{R}_T = \widehat{A}_T \times \widehat{B}_T$ for all T) are two trivial examples of standard relations. Note that $\mathcal{R}_T$ may relate an element of dynamic type T to one of only virtual type T. We also have the empty generalized relation and the universal generalized relation ($\mathcal{G}_H = ENV_H^{\widehat{\boldsymbol{A}}} \times ENV_H^{\widehat{\boldsymbol{B}}}$ for all H).

Standard or generalized relations between Σ-algebras that are preserved under the operations of an algebra, in a sense made precise in the following definitions, are called homomorphic.[3]

The various notions of one algebra simulating another are defined in terms of relations of this kind.

Definition 3.2$^{\mathrm{D}}$ **(standard homomorphic relation).** Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be Σ-algebras and let $\mathcal{R}$ be a standard relation between $\boldsymbol{A}$ and $\boldsymbol{B}$. $\mathcal{R}$ is a *standard homomorphic relation*, or simply *homomorphic*, if it satisfies the following condition:

(SHR1) For every $g \in OP_n$ and admissible type $\vec{T} \to S$ of g,

$$\vec{a}\ \mathcal{R}_{\vec{T}}\ \vec{b} \text{ implies } g^{\boldsymbol{A}}(\vec{a})\ \mathcal{R}_S\ g^{\boldsymbol{B}}(\vec{b}). \quad \square$$

Recall the notion of a homomorphism between type contexts that was defined in Sec. 2 (Def. 2.7).

[3]Homomorphic relations are called *logical relations* when extended to higher types. An independent generalization of logical relations that appears to be closely related to our notion of homomorphic generalized relations is considered in [14].

Definition 3.3$^{\mathrm{D}}$ (generalized homomorphic relation). Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be Σ-algebras and let $\mathcal{G}$ be a generalized relation between $\boldsymbol{A}$ and $\boldsymbol{B}$. $\mathcal{G}$ is a *generalized homomorphic relation*, or simply *homomorphic*, if the following conditions are satisfied:

(GHR1) Let H be a type context and let $\vec{x} \in Dom(H)^n$ and $\vec{T} \in TYPE^n$ such that $H \vdash \vec{x} : \vec{T}$. Let $g \in OP_n$ and let type $\vec{T} \to S$ be an admissible type of g. Then for any $y \in VAR \setminus Dom(H)$ and any pair of H-environments ρ and σ in $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively,

$$\rho\ \mathcal{G}_H\ \sigma \quad \text{implies} \quad [y \mapsto g^{\boldsymbol{A}}(\rho(\vec{x}))]\rho\ \mathcal{G}_{H,y:S}\ [y \mapsto g^{\boldsymbol{B}}(\sigma(\vec{x}))]\sigma.$$

(GHR2) For all type contexts H and K and every context homomorphism h from K to H,

$$\rho\ \mathcal{G}_H\ \sigma \text{ implies } \rho \circ h\ \mathcal{G}_K\ \sigma \circ h. \quad \square$$

$\mathcal{R}$ is a standard homomorphic relation between $\boldsymbol{A}$ and $\boldsymbol{B}$ exactly when it is a standard homomorphic relation between $\widehat{\boldsymbol{A}}$ and $\widehat{\boldsymbol{B}}$ is the sense of [17, Def. 2.6]; similarly for generalized homomorphic relations ([17, Def. 2.7]). Both notions of homomorphic relation are symmetric, that is, if $\mathcal{R}$ ($\mathcal{G}$) is a standard (generalized) homomorphic relation between $\boldsymbol{A}$ and $\boldsymbol{B}$, then its converse $\breve{\mathcal{R}}$ ($\breve{\mathcal{G}}$) is a standard (generalized) homomorphic relation between $\boldsymbol{B}$ and $\boldsymbol{A}$.

Let $\mathcal{R}$ be a standard homomorphic relation between $\boldsymbol{A}$ and $\boldsymbol{B}$. For each type context $H = \{ x_1 : T_1, \ldots, x_n : T_n \}$, define $\mathcal{R}^+_H \subseteq ENV^{\widehat{\boldsymbol{A}}}_H \times ENV^{\widehat{\boldsymbol{B}}}_H$ by the condition

$$\rho\ \mathcal{R}^+_H\ \sigma \quad \text{iff} \quad \rho(x_i)\ \mathcal{R}_{T_i}\ \sigma(x_i) \text{ for all } i = 1, \ldots, n,$$

and set $\mathcal{R}^+ := \langle \mathcal{R}^+_H : H \in TCON \rangle$. $\mathcal{R}^+$ is called the *pointwise extension* of $\mathcal{R}$. It is shown in [17, Thm. 4.3] that if $\mathcal{R}$ is homomorphic, then so is $\mathcal{R}^+$.

Conversely, every generalized relation $\mathcal{G}$ restricts to a standard relation in the following way. Let $\mathcal{G}$ be a generalized relation between $\boldsymbol{A}$ and $\boldsymbol{B}$. For each type T, define $\mathcal{G}^-_T \subseteq A \times B$ by the condition $a\ \mathcal{G}^-_T\ b$ iff there exist $H \in TCON$, $\rho, \sigma \in ENV^{\boldsymbol{A} \times \boldsymbol{B}}_H$, and $x \in Dom(H)$ with $H \vdash x : T$ such that $\rho(x) = a$, $\sigma(x) = b$, and $\rho\ \mathcal{G}_H\ \sigma$. Set $\mathcal{G}^- := \langle \mathcal{G}^-_T : T \in TYPE \rangle$. $\mathcal{G}^-$ is called the *projective restriction* of $\mathcal{G}$.

It is easy to check that $\mathcal{R}^{+-} = \mathcal{R}$ for every standard relation $\mathcal{R}$ and $\mathcal{G}^{-+} \supseteq \mathcal{G}$ for every generalized relation. Moreover, if $\mathcal{R}$ is homomorphic, then so is $\mathcal{R}^+$, but it is not the case that every homomorphic generalized relation is of the form $\mathcal{R}^+$ for some homomorphic standard relation $\mathcal{R}$; see [17, Ex. A.5]. The projective restriction $\mathcal{G}^-$ of a homomorphic generalized relation $\mathcal{G}$ is not in general homomorphic as a standard relation; a counterexample can be found in Ex. 3.12. There is one important situation however in which $\mathcal{G}$ homomorphic always implies $\mathcal{G}^-$ is homomorphic, namely when the signature has only unary methods:

Theorem 3.4$^{\mathrm{D}}$. *Assume Σ is a signature with only unary methods and let $\boldsymbol{A}$ and $\boldsymbol{B}$ be Σ-algebras. Let $\mathcal{G}$ be a VIS-identical generalized homomorphic relation between $\boldsymbol{A}$ and $\boldsymbol{B}$. Then $\mathcal{G}^-$ is a VIS-identical standard homomorphic relation between $\boldsymbol{A}$ and $\boldsymbol{B}$.*

Proof. Let $g \in OP_n$ and let $T_1, \ldots, T_n \to S$ be an admissible type of g. Let $\langle a_i, b_i \rangle \in A_{T_i} \times B_{T_i}$ such that $a_i \; \mathcal{G}_T^- \; b_i$ for all $i \leq n$. Then for each $i \leq n$ there is an $H_i \in TCON$, a $\langle \rho_i, \sigma_i \rangle \in \mathcal{G}_{H_i}$, and an x_i with $H_i \vdash x_i : T$ such that $\rho_i(x_i) = a_i$ and $\sigma_i(x_i) = b_i$. Since Σ is a signature with unary methods by assumption, there is at most one $j \leq n$ such that $T_j \notin VIS$; without loss of generality we may assume that $j = 1$. Since $\mathcal{G}$ is *VIS*-identical, $a_i = b_i$ for every $2 \leq i \leq n$. Thus, since the visible sorts are all term-generated, there exists for each $2 \leq i \leq n$ a nullary term t_i of type T_i such that $t_i^{\boldsymbol{A}} = t_i^{\boldsymbol{B}} = a_i \; (= b_i)$. Let $y_2, \ldots, y_n \in VAR \setminus Dom(H_n)$, and let

$$\hat{\rho}_n = [y_2 \to t_2^{\boldsymbol{A}}] \cdots [y_n \to t_n^{\boldsymbol{A}}] \rho_n \quad \text{and} \quad \hat{\sigma}_n = [y_2 \to t_2^{\boldsymbol{B}}] \cdots [y_n \to t_n^{\boldsymbol{B}}] \sigma_n .$$

By the substitution property for generalized homomorphic relations, (GHR1), we have

$$\hat{\rho}_n \;\; \mathcal{G}_{H_n, y_2 : T_2, \ldots, y_n : T_n} \;\; \hat{\sigma}_n .$$

Note that $\hat{\rho}_n(y_i) = \hat{\sigma}_n(y_i) = a_i \; (= b_i)$ for each $2 \leq i \leq n$, and $\hat{\rho}_n(x_1) = a_1$ and $\hat{\rho}_n(x_1) = b_1$. Applying (GHR1) again we get

$$[z \mapsto g^{\boldsymbol{A}}(a_1, \ldots, a_n)] \hat{\rho}_n \;\; \mathcal{G}_{H_n, y_2 : T_2, \ldots, y_n : T_n} \;\; [z \mapsto g^{\boldsymbol{B}}(b_1, \ldots, b_n)] \hat{\sigma}_n ,$$

where $z \in VAR \setminus Dom(H_n, y_2 : T_2, \ldots, y_n : T_n)$. Thus

$$g^{\boldsymbol{A}}(a_1, \ldots, a_n) = \hat{\rho}_n(z) \; \mathcal{G}_S^- \; \hat{\sigma}_n(z) = g^{\boldsymbol{A}}(b_1, \ldots, b_n).$$

So $\mathcal{G}^-$ is a standard homomorphic relation between $\boldsymbol{A}$ and $\boldsymbol{B}$, and it is clearly *VIS*-identical. □

The following property of generalized homomorphic relations is an immediate consequence of (GHR2).

(GHR3) $\rho \; \mathcal{G}_H \; \sigma$ implies $\rho|_K \; \mathcal{G}_K \; \sigma|_K$ for all type contexts H and K such that K is a subcontext of H.

Properties (SHR1) and (GHR1) are called *substitution* properties. Properties (GHR2) and (GHR3) are respectively called the *pre-image* and the *subcontext* properties.

By a straightforward induction on the structure of terms it can be shown that the substitution property (GHR1) extends to terms in the following sense (cf. [17, Lem. 2.9]).

Lemma 3.5$^{\mathrm{D}}$. *Let $\boldsymbol{A}, \boldsymbol{B} \in \mathsf{SPEC}$ and $\mathcal{G}$ be a generalized homomorphic relation between $\boldsymbol{A}$ and $\boldsymbol{B}$. Let H be a type context and let ρ and σ be H-environments of $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively. If $\rho\ \mathcal{G}_H\ \sigma$, then for every H-typed term $t : T$ and every variable y not in the domain of H we have*

$$[y \mapsto [\![t : T]\!]^{\boldsymbol{A}} \rho]\rho\ \mathcal{G}_{H,y:T}\ [y \mapsto [\![t : T]\!]^{\boldsymbol{B}} \sigma]\sigma. \quad \square$$

For any algebra $\boldsymbol{A}$, the *TYPE*-indexed family of identity relations on the carrier sets of $\widehat{\boldsymbol{A}}$ is a standard homomorphic relation between $\boldsymbol{A}$ and $\boldsymbol{A}$, and the *TCON*-indexed family of identity relations on environments over $\widehat{\boldsymbol{A}}$ is a generalized homomorphic relation between $\boldsymbol{A}$ and $\boldsymbol{A}$. Slightly more interesting examples are the relations between two arbitrary algebras $\boldsymbol{A}$ and $\boldsymbol{B}$ in SPEC that coincide with the identity on visible type domains and visible environments and are empty otherwise. These relations always exist because of the assumption that $\boldsymbol{A}|_{VIS} = \boldsymbol{B}|_{VIS}$ when $\boldsymbol{A}$ and $\boldsymbol{B}$ are models of the same specification.

Definition 3.6$^{\mathrm{D}}$ (visible identity relations). Let $\boldsymbol{A}, \boldsymbol{B} \in \mathsf{SPEC}$.

(i) The *visible standard identity relation* $\mathcal{I}$ between $\boldsymbol{A}$ and $\boldsymbol{B}$ is defined by $\mathcal{I} := \langle \mathcal{I}_T : T \in TYPE \rangle$, where $\mathcal{I}_T = \{ \langle a, a \rangle : a \in A_T \}$ for all $T \in VIS$, and $\mathcal{I}_T = \emptyset$ for all $T \notin VIS$.

(ii) The *visible generalized identity relation* $\mathcal{I}^+$ between $\boldsymbol{A}$ and $\boldsymbol{B}$ is defined by $\mathcal{I}^+ = \langle \mathcal{I}^+_H : H \in TCON \rangle$, where $\mathcal{I}^+_H = \{ \langle \rho, \rho \rangle : \rho \in ENV^{\boldsymbol{A}}_H \}$ for $H \in TCON|_{VIS}$, and $\mathcal{I}^+_H = \emptyset$ for $H \notin TCON|_{VIS}$. $\square$

$\mathcal{I}$ and $\mathcal{I}^+$ coincide respectively with the visible standard and generalized identity relations on the discrete algebras $\widehat{\boldsymbol{A}}$ and $\widehat{\boldsymbol{B}}$ defined in [17, Def. 2.2]. They will prove useful when we define the notion of simulation below.

Standard and generalized relations are *VIS-identical* if their visible parts coincide respectively with $\mathcal{I}$ and $\mathcal{I}^+$. More formally we have:

Definition 3.7$^{\mathrm{D}}$ (*VIS*-identical relations). Let $\boldsymbol{A}, \boldsymbol{B} \in \mathsf{SPEC}$, and let $\mathcal{R} \subseteq \widehat{A} \times \widehat{B}$ and $\mathcal{G} \subseteq ENV^{\widehat{\boldsymbol{A}}} \times ENV^{\widehat{\boldsymbol{B}}}$.

(i) $\mathcal{R}$ is *VIS-identical* if $\mathcal{R}|_{VIS} = \mathcal{I}$.

(ii) $\mathcal{G}$ is *VIS-identical* if $\mathcal{G}|_{VIS} = \mathcal{I}^+$. $\square$

Neither the empty nor the universal relations are *VIS*-identical.

$\mathcal{R}$ and $\mathcal{S}$ are *VIS*-identical iff they are *VIS*-identical relations between the discrete $\widehat{\Sigma}$-algebras $\widehat{\boldsymbol{A}}$ and $\widehat{\boldsymbol{B}}$ in the sense of [17, Def. 2.11].

It is reasonable to require that a visible data element simulate only itself [26], so a standard homomorphic relation is said to be a *weak simulation*[4] between discrete algebras $\boldsymbol{A}$ and $\boldsymbol{B}$ if it relates every visible data element only to itself. (Recall our assumption that the visible parts of any two algebras in the specification are the same.) Schoett [26] proves this weak notion of simulation is both necessary and sufficient to insure behavioral equivalence with regard to visible data. But Schoett's notion of simulation turns out to be inadequate when nonvisible data is taken into account. We now define formally two notions of simulation, nominal

[4] See [17, Def. 2.14]; since this relation is symmetric it is called a bisimulation in [17].

standard simulation and nominal generalized simulation, both of which are strong enough to insure correct behavioral subtyping. However, only the latter turns out to be both necessary and sufficient for this purpose for signatures that may contain multimethods.

By a standard simulation between algebras with subtyping, we mean a standard simulation between their discrete transforms in the sense of [17, Def. 2.11]. This means that a *VIS*-identical standard homomorphic relation $\mathcal{R}$ between two algebras $\boldsymbol{A}$ and $\boldsymbol{B}$ is a *standard simulation of* $\boldsymbol{B}$ *by* $\boldsymbol{A}$ if, for each type T and each element a of $\boldsymbol{A}$ of virtual type T, a is $\mathcal{R}_T$-related to some element b of $\boldsymbol{B}$ that is also of virtual type T. The significant notion of simulation from the standpoint of behavioral subtyping is that of nominal standard simulation, where each element a of $\boldsymbol{A}$ of virtual type T is required to be $\mathcal{R}_T$-related to some element b of $\boldsymbol{B}$ of dynamic type T.

Definition 3.8 (nominal standard simulation). Let $\boldsymbol{A}, \boldsymbol{B} \in \mathsf{SPEC}$. A *nominal standard simulation of* $\boldsymbol{B}$ *by* $\boldsymbol{A}$ is a standard relation $\mathcal{R} \subseteq \widehat{A} \times \widehat{B}$ that satisfies the following conditions.

(i) $\mathcal{R}$ is *VIS*-identical and homomorphic.

(ii) For every $T \in TYPE$ and $a \in \widehat{A}_T$, there exists a $b \in B_T$ such that $a \mathcal{R}_T b$.

$\mathcal{R}$ is a *nominal standard bisimulation between* $\boldsymbol{A}$ *and* $\boldsymbol{B}$ if both $\mathcal{R}$ and its converse $\breve{\mathcal{R}}$ are nominal standard simulations. □

We call the property (ii) the *coercion property.*

We say that *a simulates b at type T* (*under* $\mathcal{R}$) if $\langle a, b\rangle \in \mathcal{R}_T$. Note that the requirement that $\mathcal{R}$ be *VIS*-identical ($\mathcal{R} \mid_{VIS} = \mathcal{I}$) means that for each visible type V, each visible element of type V in $\mathbf{A}$ simulates just itself at type V.

Example 3.9. Let $\boldsymbol{II}$ be the algebra of Ex. 2.11, with the nonvisible types `Int`, `IntSet` and `Interval`. Then $\boldsymbol{II}$ does not simulate itself by means of any nominal standard simulation. This is because $\texttt{choose}^{\boldsymbol{II}}$ returns the least element of an `Interval` value, but the greatest element of an `IntSet` value.

However, because SPEC_{II}, the specification given in Ex. 2.19, puts no restriction on which element of an `IntSet` value $\texttt{choose}^{\boldsymbol{II}}$ returns, it is possible to find another algebra $\widetilde{\boldsymbol{II}} \in \mathsf{SPEC}_{II}$ such that $\boldsymbol{II}$ does simulate $\widetilde{\boldsymbol{II}}$ by a nominal standard simulation. In fact, for every $\boldsymbol{A} \in \mathsf{SPEC}_{II}$ there exists an $\widetilde{\boldsymbol{A}} \in \mathsf{SPEC}_{II}$ such that $\boldsymbol{A}$ simulates $\widetilde{\boldsymbol{A}}$ by a nominal standard simulation. $\widetilde{\boldsymbol{A}}$ can be taken to be identical to $\boldsymbol{A}$, except that the carrier set of `IntSet` is defined to be $\widetilde{A}_{\texttt{IntSet}} = A_{\texttt{IntSet}} \cup A_{\texttt{Interval}}$. Note the $\widetilde{\boldsymbol{A}}$ is identical to $\widehat{\boldsymbol{A}}$ is all respects except that the subtyping of $\boldsymbol{A}$ is retained ($\widehat{\boldsymbol{A}}$ is discrete). The identity standard relation $\mathcal{R} \subseteq \widehat{A} \times \widetilde{A}$ is a nominal standard simulation of $\widetilde{\boldsymbol{A}}$ by $\boldsymbol{A}$; more precisely, $\mathcal{R}$ is defined as follows.

$$\mathcal{R}_{\texttt{Bool}} = \{\langle b, b\rangle : b \in A_{\texttt{Bool}}\} \qquad \mathcal{R}_{\texttt{IntSet}} = \{\langle s, s\rangle : s \in \widehat{A}_{\texttt{IntSet}}\}$$
$$\mathcal{R}_{\texttt{Int}} = \{\langle i, i\rangle : i \in A_{\texttt{Int}}\} \qquad \mathcal{R}_{\texttt{Interval}} = \{\langle s, s\rangle : s \in A_{\texttt{Interval}}\}. \quad \square$$

This construction of a nominal standard simulation of the model $\widetilde{\boldsymbol{A}}$ of SPEC_{II} by $\boldsymbol{A}$ seems to use a trick, and it does. The trick is that $\widetilde{\boldsymbol{A}}$ is obtained from $\boldsymbol{A}$

simply by respecifying each data element of $\boldsymbol{II}$ of dynamic type `Interval`, which is of virtual type `IntSet`, to also be of dynamic type `IntSet`. So when this element simulates itself at type `IntSet` as an element of $\widetilde{\boldsymbol{A}}$, the simulation is nominal.

The procedure is not as arbitrary as it appears because, for the trick to work, it is essential that, if $\boldsymbol{A}$ is in SPEC_{II}, then $\widetilde{\boldsymbol{A}}$ is also in SPEC_{II}. This is indeed true for SPEC_{II} because the specification is so loose it puts no condition on the `choose` operation, when applied to a value a of `IntSet`, other than that it must return a value of `Int` that is a member a.

More to the point, the fact that respecification of `Interval` data elements as `IntSet` data elements does not take us out of SPEC_{II} verifies an important property of the specification SPEC_{II}: after we have established the precise link between nominal simulation and behavioral subtyping in Cor. 5.5, we will be able to conclude that the subtype relation in SPEC_{II} is a correct behavioral subtype relation. Intuitively this means that there can be no surprising behavior with respect to subtyping in any model of SPEC_{II} (see Def. 5.1).

Example 3.10. For the algebra $\boldsymbol{E}$ of Example 2.9, with the nonvisible types `LO` and `Comp`, there is also no nominal standard simulation of $\boldsymbol{E}$ by $\boldsymbol{E}$. Suppose such a simulation, say $\mathcal{R}$, exists. Since $\mathcal{R}$ is nominal, for each integer $k \in E_{\texttt{Comp}}$ there is a natural number $n_k \in E_{\texttt{LO}}$ such that $k\ \mathcal{R}_{\texttt{LO}}\ n_k$. For each pair k, l of integers such that $k < l$ (where $<$ is the natural order of the integers) we have $\texttt{leq}^{\boldsymbol{E}}(k,l) = \mathit{tt}$ and $\texttt{leq}^{\boldsymbol{E}}(l,k) = \mathit{ff}$, and thus, since $\mathcal{R}$ is a simulation, $\texttt{leq}^{\boldsymbol{E}}(n_k, n_l)\ \mathcal{R}_{\texttt{Bool}}\ \mathit{tt}$ and $\texttt{leq}^{\boldsymbol{E}}(n_l, n_k)\ \mathcal{R}_{\texttt{Bool}}\ \mathit{ff}$. Hence $\texttt{leq}^{\boldsymbol{E}}(n_k, n_l) = \mathit{tt}$ and $\texttt{leq}^{\boldsymbol{E}}(n_l, n_k) = \mathit{ff}$ since $\mathcal{R}$ is *VIS*-identical. Thus $n_k < n_l$ (where $<$ is now the natural order of the natural numbers). This implies that the natural numbers contains an infinite, strictly descending sequence, a contradiction. (Compare [17, Ex. A.1].)

The trick we used in Ex. 3.9 does not work here. If we respecify each integer to be of dynamic type `LO` we obtain a Σ_E-algebra outside the specification SPEC_E because of the condition that the set of `LO` data elements is well-ordered under the linear ordering (Def. 2.18(iv)). The argument used above to show that $\boldsymbol{E}$ does not nominally simulate itself can be used to show that $\boldsymbol{E}$ does not nominally simulate any model of SPEC_E. This illustrates the inadequacy of standard simulations for characterizing correct behavioral subtyping for such specifications and shows the necessity of the notion of generalized simulation defined below. □

In [17] we considered a generalized notion of simulation that related entire environments rather than individual data elements. This allows for a more refined analysis of the comparative behavior of individual data elements by taking into account all the various contexts in which they can occur. A generalized simulation with subtyping is a generalized simulation of the discrete transforms in the sense of [17, Def. 2.13]. In detail, a *VIS*-identical generalized homomorphic relation $\mathcal{G}$ between two algebras $\boldsymbol{A}$ and $\boldsymbol{B}$ is a *generalized simulation of* $\boldsymbol{B}$ *by* $\boldsymbol{A}$ if, for every type context H, each H-environment over $\boldsymbol{A}$ is $\mathcal{G}_H$-related to some H-environment over $\boldsymbol{B}$. From the standpoint of behavioral subtyping, the more significant notion is that of a nominal generalized simulation: each H-environment over $\boldsymbol{A}$ is $\mathcal{G}_H$-related to some nominal H-environment over $\boldsymbol{B}$. The formal definition is as follows.

Definition 3.11 (nominal generalized simulation). Let $\boldsymbol{A}, \boldsymbol{B} \in \mathsf{SPEC}$. A *nominal generalized simulation of* $\boldsymbol{B}$ *by* $\boldsymbol{A}$ is a relation $\mathcal{G} \subseteq ENV^{\widehat{\boldsymbol{A}}} \times ENV^{\widehat{\boldsymbol{B}}}$ that satisfies the following condition.

(i) $\mathcal{G}$ is *VIS*-identical and homomorphic

(ii) for every $H \in TCON$ and $\rho \in ENV^{\widehat{\boldsymbol{A}}}_H$, there exists a nominal $\sigma \in ENV^{\boldsymbol{B}}_H$ such that $\rho \, \mathcal{G}_H \, \sigma$,

$\mathcal{G}$ is a (*nominal*) *generalized bisimulation* between $\boldsymbol{A}$ and $\boldsymbol{B}$ if both $\mathcal{G}$ and $\breve{\mathcal{G}}$ are generalized simulations. □

Again, we call the property (ii) the *coercion property.*

If $\mathcal{R}$ is a nominal standard simulation, then its pointwise extension $\mathcal{R}^+$ is a nominal generalized simulation. In particular, the pointwise extension of the nominal standard simulation $\mathcal{R}$ of $\boldsymbol{A}$ in $\widetilde{\boldsymbol{A}}$ given in Ex. 3.9 is a nominal generalized simulation. But generalized simulations are actually more general.

Example 3.12. Although there is no nominal standard simulation of $\boldsymbol{E}$ by itself, as observed in Ex. 3.10, we will now construct a nominal generalized simulation $\mathcal{G}^{\boldsymbol{E}}$ of $\boldsymbol{E}$ by itself. This will give an example of a homomorphic generalized relation whose projective restriction is not a homomorphic standard relation. For if $\mathcal{G}^{\boldsymbol{E}}{}^-$ were homomorphic, then it is easy to see it would be a nominal standard simulation of $\boldsymbol{E}$ by itself, which is impossible.

The construction of $\mathcal{G}^{\boldsymbol{E}}$ is adapted from one given in Exs. 2.8 and 2.16 of [17].

Let $H \in TCON$ and $\langle \rho, \sigma \rangle \in ENV^{\widehat{\boldsymbol{E}}}_H \times ENV^{\widehat{\boldsymbol{E}}}_H$ be given. We say that $\langle \rho, \sigma \rangle$ is *finite partial order isomorphism* if the following holds. For all $x, y \in Dom(H)$ such that $H \vdash x : \mathtt{LO}$ and $H \vdash y : \mathtt{LO}$, $\rho(x) \leq \rho(y)$ iff $\sigma(x) \leq \sigma(y)$. (This makes sense since $\rho(x)$, $\rho(y)$, $\sigma(x)$, and $\sigma(y)$ are all in $\widehat{E}_{\mathtt{LO}} = \mathbb{Z}$.) Let $\mathcal{G}^{\boldsymbol{E}}$ be the set of all finite partial order isomorphisms such that $\rho(x) = \sigma(x)$ whenever $H \vdash x : \mathtt{Comp}$ or $H \vdash x : \mathtt{Bool}$. It is easy to check the $\mathcal{G}^{\boldsymbol{E}}$ satisfies the substitution, pre-image, and subcontext properties that define a generalized homomorphic relation (see Ex. 2.8 of [17] for details). Moreover, it also satisfies the coercion property of a nominal generalized simulation. To see this let H be a type context $\rho \in ENV^{\widehat{\boldsymbol{E}}}_H$. We must show there is a nominal $\sigma \in ENV^{\boldsymbol{E}}_H$ such that $\langle \rho, \sigma \rangle \in \mathcal{G}^{\boldsymbol{E}}$. In this context "$\sigma$ is nominal" means that $\sigma(x) \geq 0$ whenever $H \vdash x : \mathtt{LO}$. σ can be obtained from ρ by shifting all the negative integers of the form $\rho(x)$, where x has H-type $\mathtt{LO}$, by the same amount M far enough in the positive direction to make them nonnegative. Since environments are finite mappings, there is a smallest nonnegative integer with this property.

$$M := \max\{\, |\rho(x)| : x \in Dom(H),\ H \vdash x : \mathtt{LO}, \text{ and } \rho(x) < 0 \,\}.$$

Define $\sigma \in \mathcal{G}^{\boldsymbol{E}}_H$ by the conditions that

- $\sigma(x) = \rho(x)$ for all x such that $H \vdash x : \mathtt{Bool}$ or $H \vdash x : \mathtt{Comp}$, and
- $\sigma(x) = \rho(x) + M$ for all x such that $H \vdash x : \mathtt{LO}$.

Clearly, σ is nominal and $\langle \rho, \sigma \rangle \in \mathcal{G}^{\boldsymbol{E}}$.

It is not difficult to see how the construction of $\mathcal{G}^{\boldsymbol{E}}$ can be modified to give, for every model $\boldsymbol{A}$ of the specification SPEC_E (Ex. 2.18), a nominal generalized simulation of $\boldsymbol{E}$ by $\boldsymbol{A}$. The key to the construction is fact that, by the specification, $A_{\mathtt{Comp}}$ is linearly ordered by $\mathtt{leq}^{\boldsymbol{A}}$, and every finite linearly ordered set is order-isomorphic to a subset of the natural numbers, and hence to $E_{\mathtt{Comp}}$ under $\mathtt{leq}^{\boldsymbol{E}}$. □

As in the case of SPEC_E, we will be able to conclude by means of Cor. 5.5 below that the subtype relation in SPEC_{II} is a correct behavioral subtyping. However there is a significant difference between SPEC_{II} and SPEC_E from our point of view. The correct behavioral subtyping of SPEC_{II} is established by nominal standard simulations, and in the case of SPEC_E we used nominal generalized simulations. As previously observed, the correct behavioral subtyping of SPEC_E cannot be verified by nominal standard simulations.

4 Behavior and Realization

The concept of behavior, in particular visible behavior, and the closely related notion of realization, underlies much of the work on semantics of ADTs. By *visible behavior* we mean, informally, the printed or returned results of observations. By *realization* we mean data that produce some desired behavior. *Surprising behavior* contradicts the predictions of a specification. To handle multiple dispatch, we formalize the notion of the realization of an element a of an Σ-algebra as an environment ρ over the algebra with the property that a is the result of executing one of the set of allowed procedures in the environment ρ; symmetrically, the behavior of an environment ρ is any element a such that ρ realizes it in the above sense [17]. The assumption is that there is a well-defined class of *procedures* associated with any signature that can take nonvisible input data (represented in the present context in the form of an environment) and output a nonvisible element of the algebra. The term *observation* is reserved for those procedures that output visible data. In our earlier paper, and also in the present one, we restrict ourselves to simple functional procedures that can be identified with terms of signature Σ. (However, this restriction is not essential for our methods to work.)

We formalize the above remarks in the following definition. (Note that the term t in the following definition is not necessarily ground.)

Definition 4.1 (procedure, observation, program). Let Σ be a signature with subtyping and H a type context. Any well H-typed term t will be called an *H-procedure*. If the H-type of t is visible, t will be called an *H-observation*. If both H and the H-type of t are visible, t will be called an *H-program*. □

Among the procedures are the variables $x : T$ and the ground terms $t : T$ of arbitrary type T. These are called the *projection* and *ground procedures*, respectively. If T is visible, then the ground procedures are just the programs that require no input to run. Because of the assumption that the visible reduct of each $\boldsymbol{A}$ in SPEC is term-generated, there is, for each visible element a, a ground program with output a.

For an example, recall the signature Σ_E of Ex. 2.2 with the type `LO`. Let the type context H be such that $H(x_1) = \texttt{LO}$. Then x_1 is an H-procedure, and $\texttt{leq}(x_1, x_1)$ is an H-observation, as it has type `Bool`.

Let $t(x_1, \ldots, x_n)$ be a K-procedure (K-observation, K-program), and let h be a context homomorphism from K to H. Then $t\big(h(x_1)\ldots h(x_n)\big)$ is a H-procedure (H-observation, H-program).

The following is the main definition of this section.

Definition 4.2$^{\mathrm{D}}$ **(behavior-realization relation).** Let Σ be a signature with subtyping, let $\mathbf{A}$ be a Σ-algebra, and let $H \in TCON$ and $T \in TYPE$. Let $\rho \in ENV_H^{\widehat{\boldsymbol{A}}}$, $a \in \widehat{A}_T$, and let t be an H-procedure of type T. Then ρ *realizes* a *under* t, and a is the *behavior* of ρ *under* t, if $[\![\, t : T \,]\!]^{\boldsymbol{A}}\rho = a$. The element a is a *visible behavior* of ρ *under* t if $a \in A_T$ for some $T \in VIS$. □

Example 4.3. For example, recall the Σ_E-algebra $\boldsymbol{E}$ of Ex. 2.9. Let the type context H be such that $H(x_1) = \texttt{LO}$. Let ρ be an H-environment such that $\rho(x_1) = -3$. Let s be the H-procedure $\texttt{leq}(x_1, x_1)$. Then ρ realizes tt under s, and tt is the behavior of ρ under s, because $[\![\, s : \texttt{Bool} \,]\!]^{\boldsymbol{E}}\rho = tt$. □

An ADT is often specified by specifying the acceptable behavior of those H-environments ρ that are meaningful to the programmer. One way of doing this is by focusing on the *behavior function* of ρ, that is, the function that maps each H-procedure t to the behavior $[\![\, t : T \,]\!]^{\boldsymbol{A}}\rho$ of ρ under t, and then specifying the family of acceptable functions of this kind, say by means of some formal specification language. An alternate approach, and the one we take here, is to compare the behavior of ρ in $\boldsymbol{A}$ to its behavior in some paradigmatic algebra $\boldsymbol{B}$, or some class of such paradigms. This presumes of course that ρ is an H-environment in both $\boldsymbol{A}$ and $\boldsymbol{B}$. This will be true for $\boldsymbol{A}, \boldsymbol{B} \in \mathsf{SPEC}$ if ρ is visible because $\boldsymbol{A}|_{VIS} = \boldsymbol{B}|_{VIS}$. It is not true in general for nonvisible environments, and so we must consider a more general relation that compares the behaviors of different environments. We therefore shift the focus from the behavior function to the *comparative behavior relation*. This is the standard relation between $\boldsymbol{A}$ and $\boldsymbol{B}$ that compares, for each H-procedure t, the behavior under t of a given H-environment ρ of $\boldsymbol{A}$ with the behavior under the same t of a given H-environment σ of $\boldsymbol{B}$. These considerations lead to the following definitions.

Definition 4.4$^{\mathrm{D}}$ **(behavior and realization relations).** Let $\boldsymbol{A}, \boldsymbol{B} \in \mathsf{SPEC}$.

(i) Let $H \in TCON$ and $\langle \rho, \sigma \rangle \in ENV_H^{\widehat{\boldsymbol{A}}} \times ENV_H^{\widehat{\boldsymbol{B}}}$. By the *comparative behavior* of ρ and σ we mean the standard relation between $\boldsymbol{A}$ and $\boldsymbol{B}$ defined by

$$\mathcal{BE}(\rho, \sigma)_T :=$$
$$\{\langle a, b \rangle : a = [\![\, t : T \,]\!]^{\boldsymbol{A}}\rho,\ b = [\![\, t : T \,]\!]^{\boldsymbol{B}}\sigma,\ t \text{ is an } H\text{-procedure of type } T\}.$$

(ii) Let $T \in TYPE$ and $\langle a, b \rangle \in \widehat{A}_T \times \widehat{B}_T$. By the *comparative realization* of a and b we mean the generalized relation between $\boldsymbol{A}$ and $\boldsymbol{B}$ defined by

$$\mathcal{RE}(a, b)_H :=$$
$$\{\langle \rho, \sigma \rangle : [\![\, t : T \,]\!]^{\boldsymbol{A}}\rho = a,\ [\![\, t : T \,]\!]^{\boldsymbol{B}}\sigma = b,\ t \text{ is an } H\text{-procedure of type } T\}.$$ □

Note that for all $\langle a, b\rangle \in \widehat{A}_T \times \widehat{B}_T$ and $\langle \rho, \sigma\rangle \in ENV_H^{\widehat{\boldsymbol{A}}} \times ENV_H^{\widehat{\boldsymbol{B}}}$,

$$a\ \mathcal{BE}(\rho,\sigma)_T\ b \quad \text{iff} \quad \rho\ \mathcal{RE}(a,b)_H\ \sigma. \tag{4.1}$$

In the sequel we usually speak simply of the *behavior* of a pair of environments instead of their comparative behavior, and similarly of the *realization* of a pair of data elements.

These definitions of behavior and realization between subtyped algebras $\boldsymbol{A}$ and $\boldsymbol{B}$ are the same notions of behavior and realization between their discrete transforms $\widehat{\boldsymbol{A}}$ and $\widehat{\boldsymbol{B}}$ defined in [17, Def. 3.2].

The following definition extends the notion of comparative behavior to a family of pairs of environments in the natural way; that is, it associates a standard relation with each generalized relation between $\boldsymbol{A}$ and $\boldsymbol{B}$. Similarly, the comparative realization of a pair of data elements is extended so as to associate a generalized relation with each standard one.

Definition 4.5$^{\mathrm{D}}$ **(behavior-realization operators).** Let $\boldsymbol{A}, \boldsymbol{B} \in \mathsf{SPEC}$.

(i) For each generalized relation $\mathcal{G} \subseteq ENV^{\widehat{\boldsymbol{A}}} \times ENV^{\widehat{\boldsymbol{B}}}$ between $\boldsymbol{A}$ and $\boldsymbol{B}$, define $\mathcal{BE}(\mathcal{G}) := \langle \mathcal{BE}(\mathcal{G})_T : T \in TYPE\rangle$, where

$$a\ \mathcal{BE}(\mathcal{G})_T\ b \quad \text{iff} \quad \exists_{H \in TCON} \exists_{\langle \rho,\sigma\rangle \in \mathcal{G}_H} \big(a\ \mathcal{BE}(\rho,\sigma)_T\ b\big).$$

$\mathcal{BE}(\mathcal{G})$ is called the *behavior of* $\mathcal{G}$ and $\mathcal{BE}(\mathcal{G})|_{VIS}$ is the *visible behavior of* $\mathcal{G}$. $\mathcal{BE}$ is a function from the generalized to the standard relations between $\boldsymbol{A}$ and $\boldsymbol{B}$. It is called the *behavior operator* on $\boldsymbol{A} \times \boldsymbol{B}$.

(ii) For each standard relation $\mathcal{R} \subseteq \widehat{A} \times \widehat{B}$ between $\boldsymbol{A}$ and $\boldsymbol{B}$, define $\mathcal{RE}(\mathcal{R}) := \langle \mathcal{RE}(\mathcal{R})_H : H \in TCON\rangle$, where

$$\rho\ \mathcal{RE}(\mathcal{R})_H\ \sigma \quad \text{iff} \quad \exists_{T \in TYPE} \exists_{\langle a,b\rangle \in \mathcal{R}_T} \big(\rho\ \mathcal{RE}(a,b)_H\ \sigma\big).$$

$\mathcal{RE}(\mathcal{R})$ is called the *realization* of $\mathcal{R}$. $\mathcal{RE}$ is a function from the standard to the generalized relations between $\boldsymbol{A}$ and $\boldsymbol{B}$. It is called the *realization operator* on $\boldsymbol{A} \times \boldsymbol{B}$. □

Example 4.6. As an example of the behavior operator, consider the nominal generalized relation $\mathcal{G}^{\boldsymbol{E}}$ between $\boldsymbol{E}$ and itself defined in Ex. 3.12. We describe the relation $\mathcal{BE}(\mathcal{G}^{\boldsymbol{E}})_T$ for each type T in the signature Σ_E. In this example we take H to be an arbitrary type context and $\langle \rho, \sigma\rangle$ an arbitrary member of $\mathcal{G}_H^{\boldsymbol{E}}$. Note that $\langle \rho, \sigma\rangle$ is a finite partial order isomorphism (see Ex. 3.12).

$\mathcal{BE}(\mathcal{G}^{\boldsymbol{E}})_{\mathtt{Bool}} = \{\, \langle b, b\rangle : b \in \{tt, ff\} \,\}$, i.e., $\mathcal{BE}(\mathcal{G}^{\boldsymbol{E}})$ is *VIS*-identical. The reason for this is that $\mathcal{G}^{\boldsymbol{E}}$ itself is *VIS*-identical, so $\rho(x) = \sigma(x)$ for all x such that $H \vdash x{:}\mathtt{Bool}$. Furthermore, all observations, apart from projections, are of the form $\mathtt{leq}(x, y)$, and $[\![\, \mathtt{leq}(x,y){:}\mathtt{Bool} \,]\!]^{\boldsymbol{E}}\rho = [\![\, \mathtt{leq}(x,y){:}\mathtt{Bool} \,]\!]^{\boldsymbol{E}}\sigma$ because, since $\langle \rho, \sigma\rangle$ is a finite partial order isomorphism, $\rho(x) \leq \rho(y)$ iff $\sigma(x) \leq \sigma(y)$.

The only procedures of type $\mathtt{Comp}$ or type $\mathtt{L0}$ are projections. Thus $\mathcal{BE}(\mathcal{G}^{\boldsymbol{E}})_{\mathtt{Comp}} = \{\, \langle n, n\rangle : n \in \mathbb{Z} \,\}$ since $\rho(x) = \sigma(x)$ for every x such that $H \vdash x : \mathtt{Comp}$. Finally, $\mathcal{BE}(\mathcal{G}^{\boldsymbol{E}})_{\mathtt{L0}} = \{\, \langle n, m\rangle : n \in \mathbb{Z}, m \in \mathbb{N}, n \leq m \,\}$. The reason for this, is that, for every x such $H \vdash x : \mathtt{L0}$, $\sigma(x)$ is obtained by shifting $\rho(x)$ in the positive direction at least

far enough so that all the integers in the range of ρ of this form are nonnegative. Since the shift can be arbitrarily large, $\sigma(x)$ can also be arbitrarily large. □

It turns out that the realization operator $\mathcal{RE}$ is of less interest to us than its *dual* $\mathcal{RE}^\partial$. Roughly speaking, $\rho\ \mathcal{RE}(\mathcal{R})\ \sigma$ iff there exists at least one $\langle a, b\rangle \in \mathcal{R}$ such that $\langle a, b\rangle$ is a behavior of $\langle \rho, \sigma\rangle$, while $\rho\ \mathcal{RE}^\partial(\mathcal{R})\ \sigma$ iff for every $\langle a, b\rangle$, if $\langle a, b\rangle$ is a behavior of $\langle \rho, \sigma\rangle$, then $\langle a, b\rangle \in \mathcal{R}$. This is defined as follows, where for any two generalized relations $\mathcal{G}$ and $\mathcal{G}''$, $\mathcal{G} \setminus \mathcal{G}''$ is the set-theoretical complement of $\mathcal{G}''$ relative to $\mathcal{G}$.

Definition 4.7$^{\mathrm{D}}$ **(dual realization operator).** Let $\boldsymbol{A}, \boldsymbol{B} \in \mathsf{SPEC}$ and $\mathcal{R} \subseteq \widehat{A} \times \widehat{B}$. Define

$$\mathcal{RE}^\partial(\mathcal{R}) := -\mathcal{RE}(-\mathcal{R}) = ENV^{\widehat{\boldsymbol{A}}} \times ENV^{\widehat{\boldsymbol{B}}} \setminus \big(\mathcal{RE}((\widehat{A} \times \widehat{B}) \setminus \mathcal{R})\big).$$

Equivalently, for all $\rho, \sigma \in ENV_H^{\widehat{\boldsymbol{A}}} \times ENV_H^{\widehat{\boldsymbol{B}}}$,

$$\rho\, \mathcal{RE}^\partial(\mathcal{R})_H\, \sigma \quad \text{iff} \quad \forall_{T \in TYPE} \big(\mathcal{BE}(\rho, \sigma)_T \subseteq \mathcal{R}_T\big).$$

$\mathcal{RE}^\partial(\mathcal{R})$ is called the *dual realization* of $\mathcal{R}$, and $\mathcal{RE}^\partial$ is called the *dual realization operator*. □

The dual behavior of a generalized relation is also definable in the same way [17], but it will play no role in this paper.

As an example of the dual realization operator, we will take the dual realization of the maximal standard *VIS*-identical relation, which is denoted $\mathcal{I}^*$. This relation plays a key role in the construction of generalized simulations. To define $\mathcal{I}^*$, recall that $\mathcal{I}$ is the standard visible identity relation between $\boldsymbol{A}$ and $\boldsymbol{B}$, i.e., the standard identity relation on $\boldsymbol{A}|_{VIS} = \boldsymbol{B}|_{VIS}$ (Def. 3.6). Recall also that $\mathcal{R} \subseteq \widehat{A} \times \widehat{B}$ is *VIS*-identical if $\mathcal{R} \mid_{VIS} = \mathcal{I}$ (Def. 3.7).

Definition 4.8 (maximal *VIS*-identical relation). Let $\boldsymbol{A}, \boldsymbol{B} \in \mathsf{SPEC}$. Then $\mathcal{I}^*$ is the standard relation between $\boldsymbol{A}$ and $\boldsymbol{B}$ defined by

$$\mathcal{I}^*_T = \begin{cases} \mathcal{I}_T\ (= \{\, \langle a, a\rangle : a \in A_T \,\}), & \text{if } T \in VIS \\ \widehat{A}_T \times \widehat{B}_T, & \text{if } T \in TYPE \setminus VIS. \end{cases} \quad \square$$

Clearly $\mathcal{I}^*$ is the maximal standard *VIS*-identical relation between $\boldsymbol{A}$ and $\boldsymbol{B}$ in the following sense.

$$\mathcal{R} \mid_{VIS} \subseteq \mathcal{I} \quad \text{iff} \quad \mathcal{R} \subseteq \mathcal{I}^*, \quad \text{for every } \mathcal{R} \subseteq \widehat{A} \times \widehat{B}. \tag{4.2}$$

The following example hints at the utility of $\mathcal{RE}^\partial(\mathcal{I}^*)$.

Example 4.9. Let $\boldsymbol{E}$ be the Σ_E-algebra of Ex. 2.9. Let H be any type context and $\rho, \sigma \in ENV_H^{\widehat{\boldsymbol{E}}}$. Then $\rho \, \mathcal{RE}^{\partial}(\mathcal{I}^*)_H \, \sigma$ iff $\langle \rho, \sigma \rangle$ is a finite partial order isomorphism and $\rho(x) = \sigma(x)$ whenever $H \vdash x{:}\mathtt{Comp}$ or $H \vdash x{:}\mathtt{Bool}$. Thus $\mathcal{RE}^{\partial}(\mathcal{I}^*) = \mathcal{G}^{\boldsymbol{E}}$, where $\mathcal{G}^{\boldsymbol{E}}$ is the nominal generalized relation between $\boldsymbol{E}$ and itself defined in Ex. 3.12 and further considered in Ex. 4.6. □

The elementary part of the theory of the operators $\mathcal{BE}$ and $\mathcal{RE}^{\partial}$ is developed in detail in [17]. The main result obtained there is that $\mathcal{BE}$ and $\mathcal{RE}^{\partial}$ form a Galois connection when viewed as mappings between the partially ordered set of standard relations (under set-theoretical inclusion) and the dual partially ordered set of generalized relations. This is expressed in the following equivalences which together we refer to simply as the *basic adjunction*. It has both a local and a global form. For the proof see [17, Thm. 3.9 and Cor. 3.10].

Theorem 4.10$^{\mathrm{D}}$ **(Basic Adjunction).** *Let* $\boldsymbol{A}, \boldsymbol{B} \in \mathsf{SPEC}$.

(i) *Let* $H \in TCON$. *Then, for every* $\mathcal{R} \subseteq \widehat{A} \times \widehat{B}$ *and every* $\langle \rho, \sigma \rangle \in ENV_H^{\widehat{\boldsymbol{A}}} \times ENV_H^{\widehat{\boldsymbol{B}}}$,

$$\mathcal{BE}(\rho, \sigma) \subseteq \mathcal{R} \quad \textit{iff} \quad \rho \, \mathcal{RE}^{\partial}(\mathcal{R})_H \, \sigma.$$

(ii) *For every* $\mathcal{R} \subseteq A \times B$ *and every* $\mathcal{G} \subseteq ENV^{\widehat{\boldsymbol{A}}} \times ENV^{\widehat{\boldsymbol{B}}}$,

$$\mathcal{BE}(\mathcal{G}) \subseteq \mathcal{R} \quad \textit{iff} \quad \mathcal{G} \subseteq \mathcal{RE}^{\partial}(\mathcal{R}). \quad \square$$

The global form of the basic adjunction (part (ii)) can be paraphrased in the following way. For every standard relation $\mathcal{R}$, its dual realization $\mathcal{RE}^{\partial}(\mathcal{R})$ is the largest generalized relation whose behavior is included in $\mathcal{R}$, and for every generalized relation $\mathcal{G}$, its behavior $\mathcal{BE}(\mathcal{G})$ is the smallest standard relation whose dual realization includes $\mathcal{G}$.

In most practical situations one is interested in the visible behavior of H-environments, that is, the function that assigns to each H-observation $t : V$ the value $[\![t : V]\!]^{\boldsymbol{A}} \rho$. We refine the notion of behavioral equivalence accordingly.

Definition 4.11$^{\mathrm{D}}$ **(*VIS*-behavioral equivalence).** Let $\boldsymbol{A}, \boldsymbol{B} \in \mathsf{SPEC}$. Let $H \in TCON$ and $\langle \rho, \sigma \rangle \in ENV_H^{\widehat{\boldsymbol{A}}} \times ENV_H^{\widehat{\boldsymbol{B}}}$. Then ρ and σ are *VIS-behaviorally equivalent* iff $\mathcal{BE}(\rho, \sigma)$ is *VIS*-identical, i.e., $\mathcal{BE}(\rho, \sigma)|_{VIS} = \mathcal{I}$. □

Note that, because there is a ground program with output a for every visible element a, we always have $\mathcal{I} \subseteq \mathcal{BE}(\rho, \sigma)$. So

$$\mathcal{BE}(\rho, \sigma)|_{VIS} = \mathcal{I} \quad \text{iff} \quad \mathcal{BE}(\rho, \sigma)|_{VIS} \subseteq \mathcal{I}. \tag{4.3}$$

Thus ρ and σ are *VIS*-behaviorally equivalent iff

$$[\![t : V]\!]^{\boldsymbol{A}} \rho = [\![t : V]\!]^{\boldsymbol{B}} \sigma \text{ for every } V \in VIS \text{ and } H\text{-program } t : V. \tag{4.4}$$

We now show how the notion of *VIS*-behavioral equivalence fits into our general framework by characterizing it in terms of the dual realization of the standard relation $\mathcal{I}^*$ (defined in Ex. 4.8).

Theorem 4.12$^{\mathrm{D}}$. *Let $\boldsymbol{A}, \boldsymbol{B} \in \mathsf{SPEC}$. Let $H \in TCON$, $\rho \in ENV_H^{\widehat{\boldsymbol{A}}}$, and $\sigma \in ENV_H^{\widehat{\boldsymbol{B}}}$. Then ρ and σ are VIS-behaviorally equivalent iff $\rho\, \mathcal{RE}^{\partial}(\mathcal{I}^*)_H\, \sigma$.*

Proof. By the basic adjunction, Thm. 4.10, we have $\mathcal{BE}(\rho,\sigma) \subseteq \mathcal{I}^*$ iff $\rho\, \mathcal{RE}^{\partial}(\mathcal{I}^*)_H\, \sigma$. But by formula (4.2), $\mathcal{BE}(\rho,\sigma)|_{VIS} \subseteq \mathcal{I}$ iff $\mathcal{BE}(\rho,\sigma) \subseteq \mathcal{I}^*$. Thus, by formula (4.3),

$$\mathcal{BE}(\rho,\sigma)|_{VIS} = \mathcal{I} \quad \text{iff} \quad \rho\, \mathcal{RE}^{\partial}(\mathcal{I}^*)_H\, \sigma. \quad \square$$

An immediate consequence of this theorem is that $\mathcal{RE}^{\partial}(\mathcal{I}^*)$ is the largest generalized relation between $\boldsymbol{A}$ and $\boldsymbol{B}$ whose behavior is *VIS*-identical. This fact is formalized in the following two corollaries.

Corollary 4.13$^{\mathrm{D}}$. *Let $\boldsymbol{A}, \boldsymbol{B} \in \mathsf{SPEC}$. Then $\mathcal{RE}^{\partial}(\mathcal{I}^*)$ is VIS-identical.* $\square$

Corollary 4.14$^{\mathrm{D}}$. *Let $\boldsymbol{A}, \boldsymbol{B} \in \mathsf{SPEC}$ and $\mathcal{G} \subseteq ENV^{\widehat{\boldsymbol{A}}} \times ENV^{\widehat{\boldsymbol{B}}}$. Then $\mathcal{BE}(\mathcal{G})|_{VIS} = \mathcal{I}$ implies $\mathcal{G} \subseteq \mathcal{RE}^{\partial}(\mathcal{I}^*)$.* $\square$

Let SPEC be a specification and $\boldsymbol{A}, \boldsymbol{B} \in \mathsf{SPEC}$. $\boldsymbol{A}$ is *VIS-behaviorally reducible* to $\boldsymbol{B}$ if the discrete transform of $\boldsymbol{A}$ is *VIS*-behaviorally reducible to the discrete transform of $\boldsymbol{B}$ in the sense of [17, Def. 3.15]; that is, $\boldsymbol{A}$ is *VIS*-behaviorally reducible to $\boldsymbol{B}$ if the following condition holds.

- For every H-environment ρ over $\boldsymbol{A}$, there is an H-environment σ over $\boldsymbol{B}$ such that ρ and σ are *VIS*-behaviorally equivalent.

This applies to all environments, whether visible or nonvisible. If H is visible, then the *VIS*-behavioral equivalence of ρ and σ implies that they are in fact equal, since in this case the H-projections are observations, in fact, programs. (Recall that $\boldsymbol{A}|_{VIS} = \boldsymbol{B}|_{VIS}$ since $\boldsymbol{A}$ and $\boldsymbol{B}$ are models of the same specification.) Thus *VIS*-behavioral reducibility implies that every visible environment is *VIS*-behaviorally equivalent to itself, when viewed first as an environment over $\boldsymbol{A}$ and then over $\boldsymbol{B}$. This latter, weaker condition turns out to be equivalent to Schoett's [26] notion of behavioral equivalence. The stronger notion of behavioral equivalence considered in [17], which deals with nonvisible as well as visible environments, is thus a natural extension of Schoett's notion.

These notions can also be applied to subtyped algebras by applying them to their discrete transforms. However, to allow reasoning that uses static type information to be valid in the presence of subtyping, we need a still stronger notion of behavioral reducibility; what is needed is that the equivalence or reducibility must be to nominal data (i.e., to data of the static type). These considerations lead to the following definition.

Definition 4.15 (nominal *VIS*-behavioral reducibility and equivalence). Let SPEC be a specification with subtyping and let $\boldsymbol{A}, \boldsymbol{B} \in \mathsf{SPEC}$. $\boldsymbol{A}$ is *nominally VIS-behaviorally reducible* to $\boldsymbol{B}$ if the following condition holds.

(i) For every $H \in TCON$ and $\rho \in ENV_H^{\widehat{\boldsymbol{A}}}$, there exists a nominal environment $\sigma \in ENV_H^{\boldsymbol{B}}$ such that ρ and σ are *VIS*-behaviorally equivalent.

The Σ-algebras $\boldsymbol{A}$ and $\boldsymbol{B}$ are *nominally VIS-behaviorally equivalent* if each of $\boldsymbol{A}$ and $\boldsymbol{B}$ is nominally *VIS*-behaviorally reducible to the other. $\square$

5 Correct Behavioral Subtyping

In this section we define the main notion of the paper, Def. 5.1, and prove the main result, Thm. 5.4. The definition says what correct behavioral subtyping means for a specification. This definition requires that there are "no surprising visible behavior" in the following sense: assume a set of static types is given along with data elements from some subtype domains. Then, relative to the observations over the static types, the data elements can behave no differently from a suitably chosen set of nominal elements of the static types. The main result is that a specification has correct behavioral subtyping if and only if each algebra of the specification can be nominally simulated by some algebra of the specification. The requirement that the simulation be nominal is key to preventing surprising behavior; it contains the idea of a coercion found in other work on behavioral subtyping [1, 3, 7, 23].

Definition 5.1 (correct behavioral subtype relation). Let Σ be a signature with subtyping having $\leq$ as its subtyping preorder. Let SPEC be a specification over Σ. Then $\leq$ is a *correct behavioral subtype relation* for SPEC if and only if, for each $\boldsymbol{A} \in \mathsf{SPEC}$, there is some $\boldsymbol{B} \in \mathsf{SPEC}$ such that $\boldsymbol{A}$ is nominally *VIS*-behaviorally reducible to $\boldsymbol{B}$. □

Both the specifications SPEC_E and SPEC_{II} have correct behavioral subtype relations. This will be proved below, using the technique to which we now turn.

The main result, Thm. 5.4 below, is that nominal generalized simulation is both a sound and complete technique for establishing that there are no subtyping surprises. More precisely, nominal generalized simulation means that, for each algebra $\boldsymbol{A}$ in the specification, there is an algebra $\boldsymbol{B}$ in the specification and a nominal generalized simulation of $\boldsymbol{B}$ by $\boldsymbol{A}$. Hence, our main result is that nominal generalized simulation is an exact criterion for nominal *VIS*-behavioral reduction. The key lemma we will need for the proof of this result is the fact that the dual realization of every standard relation is homomorphic. For discrete signatures this is proved in [17, Thm. 4.10]. Since these notions are discrete, the theorem of [17] applies automatically to the subtyped case.

Theorem 5.2$^{\mathrm{D}}$. *Let $\boldsymbol{A}, \boldsymbol{B} \in \mathsf{SPEC}$ and $\mathcal{R} \subseteq \widehat{A} \times \widehat{B}$. Then $\mathcal{RE}^{\partial}(\mathcal{R})$ is a generalized homomorphic relation.* □

It turns out that, by the basic adjunction, the dual realization of the visible identity relation, i.e., $\mathcal{RE}^{\partial}(\mathcal{I}^*)$, is the largest *VIS*-identical generalized relation between $\boldsymbol{A}$ and $\boldsymbol{B}$ (see Cors. 4.13, 4.14 above). The idea behind the proof of the soundness and completeness result, Thm. 5.4 below, is now easy to see. Suppose $\boldsymbol{A}$ is nominally *VIS*-behaviorally reducible to $\boldsymbol{B}$. Then for every H-environment ρ of $\boldsymbol{A}$ there exists a nominal H-environment σ of $\boldsymbol{B}$ such that ρ and σ are *VIS*-behaviorally equivalent, i.e., $\rho\, \mathcal{RE}^{\partial}(\mathcal{I}^*)\, \sigma$. Thus $\mathcal{RE}^{\partial}(\mathcal{I}^*)$ itself is a nominal generalized simulation of $\boldsymbol{B}$ by $\boldsymbol{A}$. Conversely, if a nominal generalized simulation of $\boldsymbol{B}$ by $\boldsymbol{A}$ exists, then it must be included in $\mathcal{RE}^{\partial}(\mathcal{I}^*)$ and hence $\boldsymbol{A}$ is nominally *VIS*-behaviorally reducible to $\boldsymbol{B}$. This is the outline of the proof. We now give the details, including the following auxilary theorem.

Theorem 5.3. *Let* $\boldsymbol{A}, \boldsymbol{B} \in \mathsf{SPEC}$ *and* $\mathcal{G} \subseteq ENV^{\widehat{\boldsymbol{A}}} \times ENV^{\widehat{\boldsymbol{B}}}$. *Assume* $\mathcal{G}$ *is homomorphic. If* $\mathcal{G}$ *is VIS-identical, then so is its behavior* $\mathcal{BE}(\mathcal{G})$.

Proof. Let a and b be visible elements of the same type T. Suppose $a\ \mathcal{BE}(\mathcal{G})_T\ b$. Then, by definition, there are H-environments ρ and σ of $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively, and an H-observation t of type T such that $[\![t : T]\!]^{\boldsymbol{A}}\rho = a$ and $[\![t : T]\!]^{\boldsymbol{B}}\sigma = b$. By hypothesis, $\mathcal{G}$ is homomorphic, thus by Lem. 3.5, for any variable y not in the domain of H,

$$[y \mapsto a]\rho\ \mathcal{G}_{H,y:T}\ [y \mapsto b]\sigma.$$

Then by (GHR2), the pre-image property of homomorphic generalized relations, $[y \mapsto a]\ \mathcal{G}_{y:T}\ [y \mapsto b]$. But since T is visible, $[y \mapsto a]$ and $[y \mapsto b]$ are visible environments of the same type context $y : T$. Thus $[y \mapsto a]$ and $[y \mapsto b]$ must be equal by the assumption that $\mathcal{G}$ is *VIS*-identical. So $a = b$ and hence $\mathcal{BE}(\mathcal{G})$ is also *VIS*-identical. □

As a criterion for establishing *VIS*-behavioral reducibility the existence of a generalized simulation is both complete and sound. This is proved in [17], Thm. 4.13, in the context of discrete specifications, and it can be applied to subtyped specifications via the discrete transform. The next theorem however, which is the main result of the paper, extends the completeness and soundness of *VIS*-behavioral reducibility to subtyped specifications in a way that takes into account static type information in reasoning about subtyping.

Theorem 5.4 (Soundness and Completeness). *Let* SPEC *be a specification with subtyping, and let* $\boldsymbol{A}, \boldsymbol{B} \in \mathsf{SPEC}$. $\boldsymbol{A}$ *is nominally VIS-behaviorally reducible to* $\boldsymbol{B}$ *iff there exists a nominal generalized simulation of* $\boldsymbol{B}$ *by* $\boldsymbol{A}$. $\boldsymbol{A}$ *and* $\boldsymbol{B}$ *are nominally VIS-behaviorally equivalent iff there is a nominal generalized bisimulation between* $\boldsymbol{A}$ *and* $\boldsymbol{B}$.

Proof. Assume $\boldsymbol{A}$ is nominally *VIS*-behaviorally reducible to $\boldsymbol{B}$. Then it follows from Def. 4.15(i) that, for every $H \in TCON$ and $\rho \in ENV_H^{\widehat{\boldsymbol{A}}}$, there is a nominal environment $\sigma \in ENV_H^{\boldsymbol{B}}$ such that ρ and σ are *VIS*-behaviorally equivalent, and hence, by Thm. 4.12, $\rho\ \mathcal{RE}^{\partial}(\mathcal{I}^*)_H\ \sigma$. We also have $\mathcal{RE}^{\partial}(\mathcal{I}^*)$ is homomorphic by Thm. 5.2 and *VIS*-identical by Cor. 4.13. Hence $\mathcal{RE}^{\partial}(\mathcal{I}^*)$ is a nominal generalized simulation of $\boldsymbol{B}$ by $\boldsymbol{A}$.

Suppose now that $\mathcal{G}$ is a nominal generalized simulation of $\boldsymbol{B}$ by $\boldsymbol{A}$. Then by definition (Def. 3.11(i)), $\mathcal{G}$ is homomorphic and *VIS*-identical. So its behavior is *VIS*-identical by Thm. 5.3. Thus $\mathcal{G} \subseteq \mathcal{RE}^{\partial}(\mathcal{I}^*)$ by Cor. 4.14. That $\boldsymbol{A}$ is nominally *VIS*-behaviorally reducible to $\boldsymbol{B}$ now follows easily. For suppose $\rho \in ENV_H^{\widehat{\boldsymbol{A}}}$. Then by the assumption that $\mathcal{G}$ is a nominal generalized simulation, there is a nominal environment $\sigma \in ENV_H^{\boldsymbol{B}}$ such that $\rho\,\mathcal{G}_H\,\sigma$. Thus $\rho\,\mathcal{RE}^{\partial}(\mathcal{I}^*)\,\sigma$, and hence ρ and σ are *VIS*-behaviorally equivalent by Thm. 4.12.

The second part of the theorem follows immediately from the first. □

Corollary 5.5. *Let* Σ *be a signature with subtyping having* $\leq$ *as its subtyping preorder. Let* SPEC *be a specification over* Σ. *Then* $\leq$ *is a correct behavioral subtype relation for* SPEC *iff, for each* $\boldsymbol{A} \in \mathsf{SPEC}$, *there is some* $\boldsymbol{B} \in \mathsf{SPEC}$ *such that there exists a nominal generalized simulation of* $\boldsymbol{B}$ *by* $\boldsymbol{A}$. □

Example 5.6. This corollary can be used to show that the subtypings in the specifications SPEC_E of Ex. 2.18 and SPEC_{II} of Ex. 2.19 are correct behavioral subtype relations. In Ex. 3.9 we showed that, for every $\boldsymbol{A} \in \mathsf{SPEC}_{II}$, there is an $\widetilde{\boldsymbol{A}} \in \mathsf{SPEC}_{II}$ such that $\boldsymbol{A}$ simulates $\widetilde{\boldsymbol{A}}$ by a nominal standard simulation $\mathcal{R}$. Then $\boldsymbol{A}$ simulates $\widetilde{\boldsymbol{A}}$ by the nominal generalized simulation $\mathcal{R}^+$, the pointwise extension $\mathcal{R}$.

In Ex. 3.12 we observed that every algebra in SPEC_E simulates the algebra $\boldsymbol{E}$ of Ex. 2.9 by a nominal generalized simulation, so that in this case $\boldsymbol{E}$ can serve as the paradigm that all the other algebras in the specification can be compared with. But, in contrast to case of SPEC_{II}, we see from the discussion in Ex. 3.10 that the correct subtyping of SPEC_E cannot be established by means of nominal standard simulations. □

Specifications that are term generated or have unary methods. The fact that the correct behavioral subtyping of SPEC_E cannot be established by means of nominal standard simulations shows that "nominal generalized simulation" can not be replaced by "nominal standard simulation" in Thm. 5.4 or Cor. 5.5. On the other hand, the existence of a standard homomorphic relation between $\boldsymbol{A}$ and $\boldsymbol{B}$ is both necessary and sufficient for the weaker notion of behavioral equivalence considered by Schoett [26]. What makes Schoett's behavioral equivalence result possible is that fact that the visible reduct of every algebra in SPEC is term-generated. It turns out that the same reasoning can be used to show that, if SPEC itself is either term-generated or has only unary methods, then nominal standard simulations are both necessary and sufficient for nominal *VIS*-behavioral reducibility, and hence for correct behavioral subtyping. The key to the proof of the first result is the following theorem, which involves the notion a pseudo-transitive relation.

A standard relation $\mathcal{R}$ between $\boldsymbol{A}$ and $\boldsymbol{B}$ is said to be *pseudo-transitive* if $(\mathcal{R}\,;\breve{\mathcal{R}}\,;\mathcal{R}) \subseteq \mathcal{R}$; that is, for all $\langle a, b\rangle \in \widehat{A}_T \times \widehat{B}_T$, if there is a $\langle c, d\rangle \in \widehat{A}_T \times \widehat{B}_T$ such that $\langle a, d\rangle, \langle c, d\rangle, \langle c, b\rangle \in \mathcal{R}$, then $\langle a, b\rangle \in \mathcal{R}$.

Theorem 5.7$^{\mathrm{D}}$**.** *Assume* SPEC *is term-generated. Let* $\boldsymbol{A}, \boldsymbol{B} \in \mathsf{SPEC}$*, and let* $\mathcal{R} \subseteq \widehat{A} \times \widehat{B}$*. If* $\mathcal{R}$ *is pseudo-transitive, then* $\mathcal{BE}\big(\mathcal{RE}^{\partial}(\mathcal{R})\big)$ *is homomorphic.* □

The proof of this theorem is presented in the appendix.

Theorem 5.8. *Assume* SPEC *is a term-generated specification with subtyping. Then* $\boldsymbol{A}$ *is nominally VIS-behaviorally reducible to* $\boldsymbol{B}$ *iff there exists a nominal standard simulation of* $\boldsymbol{B}$ *by* $\boldsymbol{A}$*.* $\boldsymbol{A}$ *and* $\boldsymbol{B}$ *are nominally VIS-behaviorally equivalent iff there is a nominal standard bisimulation between* $\boldsymbol{A}$ *and* $\boldsymbol{B}$*.*

Proof. Assume $\boldsymbol{A}$ is nominally *VIS*-behaviorally reducible to $\boldsymbol{B}$. We will show that the desired simulation of $\boldsymbol{B}$ by $\boldsymbol{A}$ is $\mathcal{BE}\big(\mathcal{RE}^{\partial}(\mathcal{I}^*)\big)$. Observe first of all that $\mathcal{I}^*$ is obviously pseudo-transitive and hence, by Lemma 5.7, $\mathcal{BE}\big(\mathcal{RE}^{\partial}(\mathcal{I}^*)\big)$ is homomorphic. Since $\mathcal{RE}^{\partial}(\mathcal{I}^*)$ is homomorphic by Thm. 5.2 and *VIS*-identical by Cor. 4.13, $\mathcal{BE}\big(\mathcal{RE}^{\partial}(\mathcal{I}^*)\big)$ is *VIS*-identical by Thm. 5.3. It remains only to verify that $\mathcal{BE}\big(\mathcal{RE}^{\partial}(\mathcal{I}^*)\big)$ satisfies the coercion property (Def. 3.8(ii)). Let $T \in \mathit{TYPE}$ and $a \in \widehat{A}_T$. Let $H = \{\langle x, T\rangle\}$ be a type context and let $\rho = [x \mapsto a]$ be an H-environment of $\boldsymbol{A}$. By assumption there is a nominal H-environment σ of $\boldsymbol{B}$ such

that ρ is *VIS*-behaviorally equivalent to σ. Thus by Thm. 4.12, $\rho\ \mathcal{RE}^{\partial}(\mathcal{I}^*)_H\ \sigma$. Let $t(x)$ be the projection procedure $x:T$. Then by definition of behavior:

$$a = [\![\, t:T \,]\!]^{\boldsymbol{A}}\rho\ \ \mathcal{BE}\left(\mathcal{RE}^{\partial}(\mathcal{I}^*)\right)_T\ \ [\![\, t:T \,]\!]^{\boldsymbol{A}}\sigma = b.$$

Hence $a\ \mathcal{BE}\left(\mathcal{RE}^{\partial}(\mathcal{I}^*_T)\right)\ b$ and, since σ is nominal, $b \in B_T$. So $\mathcal{BE}\left(\mathcal{RE}^{\partial}(\mathcal{I}^*)\right)$ is a nominal standard simulation of $\boldsymbol{B}$ by $\boldsymbol{A}$.

Assume now that there exists a nominal standard simulation $\mathcal{R}$ of $\boldsymbol{B}$ by $\boldsymbol{A}$. $\mathcal{R}$ is *VIS*-identical, so $\mathcal{R} \subseteq \mathcal{I}^*$. Let $\mathcal{R}^+$ be the pointwise extension of $\mathcal{R}$. We want to show that

$$\mathcal{BE}(\mathcal{R}^+) \subseteq \mathcal{R}. \tag{5.1}$$

Let $T \in TYPE$ and assume $a\ \mathcal{BE}(\mathcal{R}^+)_T\ b$ for some $\langle a, b\rangle \in \widehat{A}_T \times \widehat{B}_T$. Then by definition there is an $H \in TCON$, an H-procedure $t{:}T$, and $\langle \rho, \sigma\rangle \in ENV_H^{\widehat{\boldsymbol{A}}} \times ENV_H^{\widehat{\boldsymbol{B}}}$ such that $\rho\ \mathcal{R}^+_H\ \sigma$, $a = [\![\, t:T \,]\!]^{\boldsymbol{A}}\rho$, and $b = [\![\, t:T \,]\!]^{\boldsymbol{B}}\sigma$. Since t is a H-procedure, $t = t(x_1, \ldots, x_n)$ with $x_1, \ldots, x_n \in Dom(H)$, and $a = t^{\boldsymbol{A}}\left(\rho(x_1), \ldots, \rho(x_n)\right)$ and $b = t^{\boldsymbol{B}}(\sigma(x_1), \ldots, \sigma(x_n))$. For each i, if x_i is of type T_i, then $\rho(x_i)\ \mathcal{R}_{T_i}\ \sigma(x_i)$, because $\rho\ \mathcal{R}^+\ \sigma$, and hence $a\ \mathcal{R}\ b$ because $\mathcal{R}$ is homomorphic. This establishes (5.1).

It now follows directly from the basic adjunction and the fact that $\mathcal{R} \subseteq \mathcal{I}^*$ that

$$\mathcal{R}^+ \subseteq \mathcal{RE}^{\partial}(\mathcal{I}^*). \tag{5.2}$$

Let $H \in TCON$ and $\rho \in ENV_H^{\widehat{\boldsymbol{A}}}$. Let $H = \{\, x_1{:}T_1, \ldots, x_n{:}T_n \,\}$ and let $\sigma \in ENV_H^{\boldsymbol{B}}$ be a nominal environment such that, for each $i = 1, \ldots, n$, $\rho(x_i)\ \mathcal{R}_{T_i}\ \sigma(x_i)$; such a nominal σ exists, because $\mathcal{R}$ satisfies condition (ii) of Def. 3.8. Then $\rho\ \mathcal{R}^+_H\ \sigma$ and hence $\rho\ \mathcal{RE}^{\partial}(\mathcal{I}^*)_H\ \sigma$ by (5.2). Thus by Thm. 4.12, ρ is *VIS*-behaviorally equivalent to σ. So by definition $\boldsymbol{A}$ *VIS*-behaviorally reduces to $\boldsymbol{B}$.

The second part of the theorem is an immediate consequence of the first. □

Theorem 5.9. *Assume* SPEC *is a specification with subtyping that has only unary methods. Then* $\boldsymbol{A}$ *is nominally VIS-behaviorally reducible to* $\boldsymbol{B}$ *iff there exists a nominal standard simulation of* $\boldsymbol{B}$ *by* $\boldsymbol{A}$. $\boldsymbol{A}$ *and* $\boldsymbol{B}$ *are nominally VIS-behaviorally equivalent iff there is a nominal standard bisimulation between* $\boldsymbol{A}$ *and* $\boldsymbol{B}$.

Proof. Let $\mathcal{G}$ be a nominal generalized simulation of $\boldsymbol{B}$ by $\boldsymbol{A}$. By Theorem 3.4 $\mathcal{G}^-$ is a *VIS*-identical standard homomorphic relation between $\boldsymbol{A}$ and $\boldsymbol{B}$. Let $T \in TYPE$ and $a \in \widehat{A}_T$. Let H be any type context and $\rho \in ENV_H^{\widehat{\boldsymbol{A}}}$ such that $\rho(x) = a$ for some $\langle x, T\rangle \in H$. Since $\mathcal{G}$ is a nominal simulation, there is a nominal environment $\sigma \in ENV_H^{\boldsymbol{B}}$ such that $\rho\ \mathcal{G}_H\ \sigma$. Then $a\ \mathcal{G}^-_T\ \sigma(x)$, and $\sigma(x) \in B_T$ since σ is nominal simulation. This show that $\mathcal{G}^-$ is a nominal generalized simulation of $\boldsymbol{B}$ by $\boldsymbol{A}$.

That $\mathcal{G}^-$ is a standard nominal bisimulation when $\mathcal{G}$ is a nominal generalized simulation follows immediately from the first part of the proof. □

6. Discussion

In this section we discuss related work, future work, and offer some conclusions.

Related work. Our decision to formulate our results as a *behavior-realization adjunction* was considerably influenced by Goguen's categorical theory of automata [8] and its subsequent extension to (discrete) algebras by Goguen and Meseguer [10, 11]. Our notion of dual realization can be viewed in a loose sense as a generalization of Goguen and Meseguer's realization of the behavior of an automaton or algebra, although strictly speaking the two notions are incomparable. In their work they speak of the behavior of an entire algebra, which they take as the system of visible input-output functions one obtains by running each admissible program in the algebra. Thus a realization of a given system of input-output functions is any algebra for which this system is the behavior. In our case the structure of the algebra is not explicitly taken into account and the focus is shifted to environments. Since Goguen and Meseguer deal only with visible input and output, only the reachable, i.e., the term-generated part of the algebra is relevant to its behavior. In our context that would be roughly equivalent to restricting attention to the behavior of the empty environment. Furthermore, we focus on the relationship between *comparative* behavior and realization by restricting attention in effect to the product of two algebras. The connections between the two notions of realization are discussed in more detail in [17].

The models of data types used by Cardelli [4], and those in order-sorted algebra [12, 13], require that a subtype's carrier set be a subset of its supertype's carrier set. When an operation of the supertype is applied to a subtype object o, in such a model, the results are identical whether o is regarded as an element of the supertype or as an element of the subtype. Thus the existence of such a model is a sufficient criteria for correct behavioral subtyping, but it is not necessary as was shown in Ex. 3.10, for the specification of SPEC_E of Ex. 2.18. Forcing the subtype's carrier set be a subset of the supertype's carrier set may take the algebra outside the specification. However, when it works, the "trick" of constructing such a model is a good proof technique, as we discussed in Ex. 3.9.

Bruce and Wegner [3] define what we would call correct behavioral subtyping using a generalization of order-sorted algebras where there are coercion functions from each subtype's carrier set to the carrier set of the corresponding supertype. Such coercion functions are a special case of nominal standard simulations, and are thus sufficient to guarantee correct behavioral subtyping, but not necessary, even for term-generated specifications.

The technique we reported on in [16] built on the work of Bruce and Wegner, and the category sorted algebras of Reynolds [24]. In [16] we used what are called nominal standard simulation relations in the present paper, and proved that the existence of such a simulation was a sufficient condition for correct behavioral subtyping. We have shown that the technique is only complete for term-generated specifications and for specifications that do not use multiple dispatch. For completeness with non-term-generated specifications that use multiple dispatch one needs to use nominal generalized simulations.

Future work. It should be relatively straightforward to extend the results in this paper and in [17] to higher-order terms, using logical relations. Jung and Tiuryn [14] use a generalized notion of logical relation, they call them "Kripke logical relations of varying arity", to study lambda definabilty in Henkin models of the simply typed lambda calculus; the idea for such logical relations originated with Sieber [25]. They appear to be closely related to our generalized homomorphic relations (but do not consider subtyping).

We also plan to extend our results to higher-order terms in the presence of nondeterminism, as was done in [16].

How useful are the results of this paper to software engineers? In applying Cor. 5.5 to verify the correct behavioral subtyping of a given specification, one would in theory have to check that each model simulates, by a nominal generalized relation, some other model of the specification, possibly chosen from some restricted class of paradigmatic models. This is in general a difficult problem, but we have seen that in certain cases, in particular the simple test specifications SPEC_E and SPEC_{II} considered above, it is possible to verify this fact by a relatively simple argument. However, it seems that the main use of our results would be to validate the soundness and relative completeness of proof-theoretic techniques for settling questions of correct behavioral subtyping. We are presently investigating such proof-theoretic methods.

Another extension planned is to adapt our results to the study of OO ADTs with mutable objects (i.e., objects with time-varying state) [6, 7, 23].

Additional questions to investigate are proof-theoretic conditions for behavioral reduction and equivalence, especially for subtyping, and how to generalize our results by means of category theory.

Conclusions. We have presented a sound and complete model-theoretic technique for verifying the correctness of a specification with subtyping. These results are based on the theory of the adjunction formed by behavior and dual realization developed in [17]. The key ideas of this theory, which are essential for completeness in the non-term-generated case, are the generalization of the notion of observation, which allows nonvisible data to be compared, and the notion of a generalized homomorphic relation. We showed that standard homomorphic relations are too strong to exactly characterize correct behavioral subtyping.

Because of these technical innovations, our techniques apply not only to term-generated and complete specifications, but also to non-term-generated and incomplete specifications. As far as we know, these results are the first exact algebraic characterization of correct behavioral subtyping for ADTs with immutable objects.

7. Appendix

The purpose of this appendix is to prove Thm. 5.7, the main technical result we needed to prove that, in term-generated specifications, standard bisimulation is necessary as well as sufficient for correct behavior subtyping (Thm. 5.8).

The key to the proof of Thm. 5.7 is Thm. 7.3 below, which says that, if $\boldsymbol{A}$ and $\boldsymbol{B}$ are members of a term-generated specification with subtyping and $\mathcal{R}$ is a standard relation satisfying a certain very weak condition, then the dual realization

$\mathcal{RE}^{\partial}(\mathcal{R})$ of $\mathcal{R}$ is completely determined by its pointwise restriction $\mathcal{RE}^{\partial}(\mathcal{R})^{-}$, which is defined below. Two technical lemmas are required to establish this result.

Recall that a standard relation $\mathcal{R}$ between $\boldsymbol{A}$ and $\boldsymbol{B}$ is *pseudo-transitive* if $\mathcal{R}\,;\breve{\mathcal{R}}\,;\mathcal{R} \subseteq \mathcal{R}$. If $\mathcal{R}$ is pseudo-transitive then $(\mathcal{R}\,;\breve{\mathcal{R}})\,;(\mathcal{R}\,;\breve{\mathcal{R}}) \subseteq \mathcal{R}\,;\breve{\mathcal{R}}$ and $(\mathcal{R}\,;\breve{\mathcal{R}})^{\smile} = \breve{\mathcal{R}}\,;\breve{\mathcal{R}} \subseteq \mathcal{R}\,;\breve{\mathcal{R}}$. Thus $\mathcal{R}\,;\breve{\mathcal{R}}$ is transitive and symmetric and will be an equivalence relation if for every $T \in TYPE$ and every $a \in \widehat{A}_T$ there exists a $b \in \widehat{B}_T$ such that $a\ \mathcal{R}_T\ b$.

Lemma 7.1$^{\mathrm{D}}$**.** *Let $\boldsymbol{A}, \boldsymbol{B} \in \mathsf{SPEC}$ and let $\mathcal{R} \subseteq A \times B$. If $\mathcal{R}$ is pseudo-transitive, then so is $\mathcal{RE}^{\partial}(\mathcal{R})$.*

Proof. Assume $\mathcal{R}$ is pseudo-transitive. Let $H \in TCON$ and let ρ and σ be H-environments in $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively. Assume

$$\rho\ \mathcal{RE}^{\partial}(\mathcal{R})_H\,;\,\mathcal{RE}^{\partial}(\mathcal{R})_H^{\smile}\,;\,\mathcal{RE}^{\partial}(\mathcal{R})_H\ \sigma.$$

Then there exists a $\langle \nu, \mu \rangle \in ENV_H^{\widehat{\boldsymbol{A}}} \times ENV_H^{\widehat{\boldsymbol{B}}}$ such that

$$\rho\ \mathcal{RE}^{\partial}(\mathcal{R})_H\ \mu\ \mathcal{RE}^{\partial}(\mathcal{R})_H^{\smile}\ \nu\ \mathcal{RE}^{\partial}(\mathcal{R})_H\ \sigma.$$

Let $t : T$ be an H-procedure. Then by the definition of dual realization,

$$[\![\, t : T \,]\!]^{\boldsymbol{A}}\rho\ \mathcal{R}_T\ [\![\, t : T \,]\!]^{\boldsymbol{B}}\mu\ \mathcal{R}_T^{\smile}\ [\![\, t : T \,]\!]^{\boldsymbol{A}}\nu\ \mathcal{R}_T\ [\![\, t : T \,]\!]^{\boldsymbol{B}}\sigma.$$

Thus $[\![\, t : T \,]\!]^{\boldsymbol{A}}\rho\ \mathcal{R}_T\ [\![\, t : T \,]\!]^{\boldsymbol{B}}\sigma$ since $\mathcal{R}$ is pseudo-transitive. Since this holds for every H-procedure, if follows that $\rho\ \mathcal{RE}^{\partial}(\mathcal{R})_H\ \sigma$. □

Lemma 7.2$^{\mathrm{D}}$**.** *Let $\mathcal{R}$ be a standard relation between $\boldsymbol{A}, \boldsymbol{B} \in \mathsf{SPEC}$. Let $K \in TCON$ and $\langle \rho, \sigma \rangle \in ENV_H^{\widehat{\boldsymbol{A}}} \times ENV_H^{\widehat{\boldsymbol{B}}}$. Let $t_1 : T_1, \ldots, t_n : T_n$ be ground terms and let $y_1, \ldots, y_n \in VAR \setminus Dom(K)$. Finally, let $H = K \cup \{\, y_1 : T_1, \ldots, y_n : T_n \,\}$, and take ρ^+ and σ^+ be the H-environments*

$$\rho^+ = [y_1 \mapsto t_1^{\boldsymbol{A}}] \ldots [y_n \mapsto t_n^{\boldsymbol{A}}]\rho, \qquad \sigma^+ = [y_1 \mapsto t_1^{\boldsymbol{B}}] \ldots [y_n \mapsto t_n^{\boldsymbol{B}}]\sigma.$$

If $\rho\ \mathcal{RE}^{\partial}(\mathcal{R})_K\ \sigma$, then $\rho^+\ \mathcal{RE}^{\partial}(\mathcal{R})_H\ \sigma^+$.

Proof. Assume $\rho\ \mathcal{RE}^{\partial}(\mathcal{R})_K\ \sigma$. Let $s(x_1, \ldots, x_m, y_1, \ldots, y_n) : S$ be an H-procedure where $Dom(H) = \{\, x_1, \ldots, x_m \,\}$. Let $r(x_1, \ldots, x_m) = s(x_1, \ldots, x_m, t_1, \ldots, t_n)$. Then $K \vdash r : S$, i.e., $r : S$ is a K-procedure, and

$$[\![\, s : S \,]\!]^{\boldsymbol{A}}\rho^+ = [\![\, r : S \,]\!]^{\boldsymbol{A}}\rho\ \mathcal{R}_S\ [\![\, r : S \,]\!]^{\boldsymbol{B}}\sigma = [\![\, s : S \,]\!]^{\boldsymbol{B}}\sigma^+.$$

Hence $\rho^+\ \mathcal{RE}^{\partial}(\mathcal{R})_H\ \sigma^+$. □

Let $\mathcal{G}$ be a generalized relation between $\boldsymbol{A}$ and $\boldsymbol{B}$. For each type T, define the standard relation $\mathcal{G}_T^{-} \subseteq A \times B$ by the condition

- $a\ \mathcal{G}_T^{-}\ b$ iff there exist $H \in TCON$, $\rho, \sigma \in ENV_H^{\widehat{\boldsymbol{A}}} \times ENV_H^{\widehat{\boldsymbol{B}}}$, and x with $H \vdash x : T$ such that $\rho(x) = a$, $\sigma(x) = b$, and $\rho\ \mathcal{G}_H\ \sigma$.

Set $\mathcal{G}^{-} := \langle \mathcal{G}_T^{-} \colon T \in TYPE \rangle$. $\mathcal{G}^{-}$ is called the *projective restriction* of $\mathcal{G}$. It is easy to check that $\mathcal{R}^{+-} = \mathcal{R}$ for every standard relation $\mathcal{R}$ and $\mathcal{G}^{-+} \supseteq \mathcal{G}$ for every generalized relation. Another important property of the projective restriction is that, if $\mathcal{G}$ is a homomorphic generalized relation, then $\mathcal{G}^{-} = \mathcal{BE}(\mathcal{G})$. This is established in [17], Proposition 4.1.

Theorem 7.3$^{\mathrm{D}}$. *Assume* SPEC *is term-generated and* $\boldsymbol{A}, \boldsymbol{B} \in$ SPEC. *Let* $\mathcal{R} \subseteq \widehat{A} \times \widehat{B}$. *If* $\mathcal{R}$ *is pseudo-transitive, then* $\mathcal{RE}^{\partial}(\mathcal{R}) = \mathcal{RE}^{\partial}(\mathcal{R})^{-+}$.

Proof. If $\mathcal{RE}^{\partial}(\mathcal{R}) = \emptyset$, then conclusion of the theorem obviously holds; so we assume $\mathcal{RE}^{\partial}(\mathcal{R}) \neq \emptyset$. Since $\mathcal{RE}^{\partial}(\mathcal{R}) \subseteq \mathcal{RE}^{\partial}(\mathcal{R})^{-+}$ in general, the theorem follows immediately from the following lemma.

(7.1) For all $H \in TCON$, $\rho \in ENV_H^{\widehat{\boldsymbol{A}}}$, and $\sigma \in ENV_H^{\widehat{\boldsymbol{B}}}$,
if $\rho(x)\ \mathcal{RE}^{\partial}(\mathcal{R})^{-}_{H(x)}\ \sigma(x)$ for all $x \in Dom(H)$, then $\rho\ \mathcal{RE}^{\partial}(\mathcal{R})_H\ \sigma$.

This lemma is proved by induction on the number n of variables in $Dom(H)$. If $n = 0$, i.e., H is the empty type context, then, since $\mathcal{RE}^{\partial}(\mathcal{R}) \neq \emptyset$, $\mathcal{RE}^{\partial}(\mathcal{R})_H = ENV_H^{\widehat{\boldsymbol{A}}} \times ENV_H^{\widehat{\boldsymbol{B}}} = \{\langle\langle\rangle, \langle\rangle\rangle\}$. Thus the conclusion of (7.1) clearly holds.

Assume now that $n = 1$. Let $H = \{x : T\}$. Then the conclusion of (7.1) follows immediately from the definition of pointwise restriction; we have $\rho(x)\ \mathcal{RE}^{\partial}(\mathcal{R})^{-}\ \sigma(x)$ iff $\rho\ \mathcal{RE}^{\partial}(\mathcal{R})_H\ \sigma$.

Assume now that $n > 1$ and that $\rho(x)\ \mathcal{RE}^{\partial}(\mathcal{R})^{-}_{H(x)}\ \sigma(x)$ for all $x \in Dom(H)$. Choose $x, y \in Dom(H)$ with $x \neq y$. Fix $\rho \in ENV_H^{\widehat{\boldsymbol{A}}}$ and $\sigma \in ENV_H^{\widehat{\boldsymbol{B}}}$. Since $\boldsymbol{A}$ and $\boldsymbol{B}$ are term-generated by hypothesis, there exist ground terms t_x, t_y, s_x, and s_y such that

$$\rho(x) = t_x^{\boldsymbol{A}}, \quad \rho(y) = t_y^{\boldsymbol{A}}, \quad \sigma(x) = s_x^{\boldsymbol{B}}, \quad \sigma(y) = s_y^{\boldsymbol{B}}.$$

Let $H(x) = T$ and $H(y) = S$, and let K be the subcontext of H obtained by removing the type assignments $x : T$ and $y : S$. Let $\rho|_K$ and $\sigma|_K$ be the restrictions of ρ and σ to K, respectively. By the induction hypothesis,

(7.2) $$[x \mapsto t_x^{\boldsymbol{A}}]\rho|_K\ \mathcal{RE}^{\partial}(\mathcal{R})_{K,x:T}\ [x \mapsto s_x^{\boldsymbol{B}}]\sigma|_K,$$

(7.3) $$[y \mapsto t_y^{\boldsymbol{A}}]\rho|_K\ \mathcal{RE}^{\partial}(\mathcal{R})_{K,y:T}\ [y \mapsto s_y^{\boldsymbol{B}}]\sigma|_K.$$

We can now verify the following string of relationships.

$$\begin{aligned}
\rho &= [x \mapsto t_x^{\boldsymbol{A}}][y \mapsto t_y^{\boldsymbol{A}}]\rho|_K \\
&\ \ \mathcal{RE}^{\partial}(\mathcal{R})_H\ [x \mapsto s_x^{\boldsymbol{B}}][y \mapsto t_y^{\boldsymbol{B}}]\sigma|_K, && \text{by (7.2) and Lem. 7.2} \\
&\ \ \mathcal{RE}^{\partial}(\mathcal{R})_H^{\smile}\ [x \mapsto s_x^{\boldsymbol{A}}][y \mapsto t_y^{\boldsymbol{A}}]\rho|_K, && \text{by Lem. 7.2} \\
&\ \ \mathcal{RE}^{\partial}(\mathcal{R})_H\ [x \mapsto s_x^{\boldsymbol{B}}][y \mapsto s_y^{\boldsymbol{B}}]\sigma|_K, && \text{by (7.3) and Lem. 7.2} \\
&\ \ = \sigma.
\end{aligned}$$

Thus $\rho\ \mathcal{RE}^{\partial}(\mathcal{R})_H\ ;\ \mathcal{RE}^{\partial}(\mathcal{R})_H^{\smile}\ ;\ \mathcal{RE}^{\partial}(\mathcal{R})_H\ \sigma$. So, since $\mathcal{R}$ is pseudo-transitive by hypothesis, we have $\rho\ \mathcal{RE}^{\partial}(\mathcal{R})_H\ \sigma$ by Lem. 7.1. This completes the proofs of both (7.1) and the theorem. □

Proof of Thm. 5.7. By the above theorem, $\mathcal{RE}^{\partial}(\mathcal{R})^{-+} = \mathcal{RE}^{\partial}(\mathcal{R})$. Thus $\mathcal{RE}^{\partial}(\mathcal{R})^{-+}$ is homomorphic by Thm. 5.2. But, if the pointwise extension of a standard relation is homomorphic, then the relation itself is homomorphic. (This is easy to show; see [17], Theorem 4.3 for details.) Hence $\mathcal{RE}^{\partial}(\mathcal{R})^{-}$ is homomorphic. But, since $\mathcal{RE}^{\partial}\ \mathcal{R}$ is homomorphic (by Thm. 5.2), $\mathcal{BE}\left(\mathcal{RE}^{\partial}(\mathcal{R})\right) = \mathcal{RE}^{\partial}(\mathcal{R})^{-}$ by Proposition 4.1 of [17]. Thus $\mathcal{BE}\left(\mathcal{RE}^{\partial}(\mathcal{R})\right)$ is homomorphic. □

References

1. Pierre America, *Designing an Object-Oriented Programming Langauge with Behavioral Subtyping*, Foundations of Object-Oriented Languages, REX School/Workshop, Noordijkerhout, The Netherlands, May/June 1990 (J. W. de Bakker and W. P. de Roever and G. Rozenberg, eds.), Lecture Notes in Computer Science, vol. 489, Springer-Verlag, New York, 1991, pp. 60–90.
2. Kim Bruce, Luca Cardelli, Giuseppe Castagna, The Hopkins Object Group, Gary T. Leavens, and Benjamin Pierce, *On Binary Methods*, Theory and Practice of Object Systems **1** (1995), no. 3, John, Wiley and Sons, Inc., 221-242.
3. Kim B. Bruce and Peter Wegner, *An Algebraic Model of Subtype and Inheritance*, Advances in Database Programming Languages (F. Bançilhon and P. Buneman, eds.), Addison-Wesley, Reading, Mass., 1990, pp. 75–96.
4. Luca Cardelli, *A Semantics of Multiple Inheritance*, Information and Computation **76** (1988), 138–164.
5. Giuseppe Castagna, *Object-Oriented Programming: A Unified Foundation*, Progress in Theoretical Computer Science Series, Birkhauser, Boston, 1997.
6. Krishna Kishore Dhara and Gary T. Leavens, *Weak Behavioral Subtyping for Types with Mutable Objects*, Mathematical Foundations of Programming Semantics, Eleventh Annual Conference (S. Brookes, M. Main, A. Melton and M. Mislove, eds.), Electronic Notes in Theoretical Computer Science, vol. 1, Elsevier, 1995.
7. Krishna Kishore Dhara and Gary T. Leavens, *Forcing Behavioral Subtyping Through Specification Inheritance*, Proceedings of the 18th International Conference on Software Engineering, Berlin, Germany, IEEE Computer Society Press, 1996, pp. 258–267.
8. Joseph A. Goguen, *Realization is Universal*, Math. Systems Theory **6** (1973), 359–374.
9. Joseph A. Goguen, *Order Sorted Algebras*, Technical Report 14, UCLA Computer Science Department (1978), Semantics and Theory of Computation Series.
10. Joseph Goguen and José Meseguer, *Universal Realization, Persistent Interconnection and Implementation of Abstract Modules*, Proceedings, 9th International Conference on Automata, Languages and Programming (M. Nielsen and E.M. Schmidt, eds.), Lecture Notes in Computer Science, vol. 140, Springer-Verlag, New York, 1982, pp. 265–281.
11. Joseph Goguen and José Meseguer, *Initiality, Induction and Computability*, Algebraic Methods in Semantics (M. Nivat and J. Reynolds, eds.), Cambridge University Press, Cambridge, 1985, pp. 459–541.
12. Joseph Goguen and José Meseguer, *Order-Sorted Algebra I: Equational Deduction for Multiple Inheritance, Overloading, Exceptions and Partial Operations*, Theor. Comp. Sci. **105** (1987), 217–273.
13. Joseph Goguen and José Meseguer, *Order-Sorted Algebra Solves the Constructor-Selector, Multiple Representation and Coercion Problems*, Symposium on Logic in Computer Science, Ithaca, NY, IEEE, 1987, pp. 18–29.
14. A. Jung and J. Tiuryn, *A new characterization of lambda definability*, Typed lambda calculus and applications. International Conference on Typed Lambda Calculus and Applications, TLCA '93 March 1993, Utrecht, The Netherlands Proceedings. (M. Bezem and J. F. Groote, eds.), Lecture Notes in Computer Science, vol. 598, Springer-Verlag, Berlin, 1993, pp. 245–257.
15. Gary T. Leavens and Todd D. Millstein, *Multiple Dispatch as Dispatch on Tuples*, OOPSLA '98 Conference Proceedings, ACM SIGPLAN Notices, vol. 33, ACM, New York, 1998, pp. 374-387.
16. Gary T. Leavens and Don Pigozzi, *Typed Homomorphic Relations Extended with Subtypes*, Mathematical Foundations of Programming Semantics '91 (S. Brookes, ed.), Lecture Notes in Computer Science, vol. 598, Springer-Verlag, New York, 1992, pp. 144–167.
17. Gary T. Leavens and Don Pigozzi, *The Behavior-Realization Adjunction and Generalized Homomorphic Relations*, Theor. Comp. Sci. **177** (1997), 183–216, An extended version is available as [18].
18. Gary T. Leavens and Don Pigozzi, *The Behavior-Realization Adjunction and Generalized Homomorphic Relations*, Department of Computer Science, Iowa State University, TR #94-18b.

Available by anonymous ftp from ftp.cs.iastate.edu and by e-mail from alamanc@cs.iastate.edu (September 1994, revised September 1994, July 1996).

19. Gary T. Leavens and Don Pigozzi, *An exact algebraic characterization of behavioral subtyping*, Preprint núm. 315, Centre de Recerca Matemática, Istitut d'Estudis Catalans (Desembre 1995).
20. Gary T. Leavens and Don Pigozzi, *Class-Based and Algebraic Models of Objects*, US–Brazil Joint Workshops on the Formal Foundations of Software Systems (Rance Cleaveland, Michael Mislove and Philip Mulry, eds.), Electronic Notes in Theoretical Computer Science, vol. 14, Elsevier, 1999, pp. 183–216, http://www.elsevier.nl/locate/entcs/volume14.html.
21. Gary T. Leavens and William E. Weihl, *Reasoning about Object-oriented Programs that use Subtypes (extended abstract)*, ECOOP/OOPSLA '90 Proceedings (N. Meyrowitz, ed.), ACM SIGPLAN Notices, vol. 25, ACM, October, 1990, pp. 212–223.
22. Gary T. Leavens and William E. Weihl, *Specification and Verification of Object-Oriented Programs Using Supertype Abstraction*, Acta Informatica **32** (1995), no. 8, 705–778.
23. Barbara Liskov and Jeannette Wing, *A Behavioral Notion of Subtyping*, ACM Transactions on Programming Languages and Systems **16** (1994), no. 11, 1811–1841.
24. John C. Reynolds, *Using Category Theory to Design Implicit Conversions and Generic Operators*, Semantics-Directed Compiler Generation, Proceedings of a Workshop, Aarhus, Denmark (N. D. Jones, ed.), Lecture Notes in Computer Science, vol. 94, Springer-Verlag, New York, 1980, pp. 211–258.
25. K. Sieber, *Reasoning about sequential functions via logical relations*, Applications of Categories in Computer Science. Proceedings of the LMS Symposium, Durham 1991 (M. P. Fourman, P. T. Johnstone, and A. M. Pitts, eds.), London Mathematical Society Lecture Note Series, vol. 177, Cambridge University Press, Cambridge, 1992, pp. 258–269.
26. Oliver Schoett, *Behavioural Correctness of Data Representations*, Science of Computer Programming **14** (June, 1990), no. 1, 43–57.
27. Philip Wadler and Stephen Blott, *How to make ad-hoc Polymorphism less ad hoc*, Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, ACM, New York, 1989, pp. 60–76.

Department of Computer Science, Iowa State University, Ames, Iowa 50011-1040 USA
E-mail address: leavens@cs.iastate.edu

Department of Mathematics, Iowa State University, Ames, Iowa 50011 USA
E-mail address: dpigozzi@iastate.edu