1400244691
Còpia 1

Visible Semantics: An Algebraic Semantics for Automatic Verification of Algorithms

Vicent-Ramon Palasí Lallana

Report LSI-96-26-R

Visible semantics: an algebraic semantics for automatic verification of algorithms

Vicent-Ramon Palasí Lallana
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
e-mail: vicent@goliat.upc.es

Abstract

A new semantics for algebraic specifications, called "visible semantics", is defined. Its most notable property is that it is specially suitable for dealing with the problem of program correctness. Some properties of this semantics are proved and some conclusions are described.

1 Introduction

Of all the theories that attempt to give meaning to algebraic specifications, initial semantics ([GTW75], [GTW78], [TWW79a]) is one of the simplest, most intuitive and most widespread. However, from the very beginning, it was clear that this semantics was too restrictive to model some abstract data types, for instance, stacks, lists, sets, etc.

As a result, behavioural semantics appeared ([SaW83], [SaT85], [Rei81], [Rei84] althought, in this paper the approach in [Niv87] is taken as a starting point). This semantics, which tries to grasp the intuitive notion of "abstraction w.r.t. implementation", is based on dividing the properties of an algebraic specification into observable and nonobservable. Two specifications are considered equivalent if the differences between them are nonobservable.

Though there have been several formulations of behavioural semantics (see [BBK94] or [Kna93] for a survey), none of them is completely suitable for dealing with the problem of proving the correctness of an algorithm w.r.t. an algebraic specification. The following example makes this clear.

Let us suppose that there is an imperative programming language in which the operation of multiplication is not built-in, and, also that a function to compute this operation is programmed. The result could be as follows:

```
function *(a,b:nat) ret c: nat
    c:=0;
    while b > 0 do
        c:=c+a;
        b:=b-1
    endwhile
endfunction
```

If we wish to study the correctness of this function, we have to write a specification of it. Since the specifications that we are considering are the algebraic ones, the result can be as follows (initial semantics is used):

```
spec MULT1 is
    sorts
        nat
    signature
        *: nat nat ⟶ nat
    equations
        ∀ a,b:nat
        *(zero,b)=zero
        *(suc(a),b)=+(*(a,b),b)
endspec
```

We assume that operations suc, zero *and* + *(with prefix notation) have been defined. Their signatures and equations have been omitted for reasons of brevity.*

We shall consider that function "*" is correct if it is equivalent to specification MULT1 according to some reasonable notion of equivalence.

In order to find this notion of equivalence, the most logical choice is to transform the function into an algebraic specification (in this way, we must compare two specifications, which is easier than comparing a program with a specification). To do this, we can apply the algebraic semantics on function "*". The result can be as follows:

```
spec MULT2 is
    sorts
        nat
    signature
        *: nat nat ⟶ nat
    equations
        ∀ a,b:nat
        *(a,b)=eval_function("function *(a,b:nat) ret c: nat ... endfunction",a,b)
endspec
```

Operation eval_function *is part of the algebraic semantics of the language. It is assumed that all the operations of this algebraic semantics have been defined. Their signatures and equations have been omitted for reasons of brevity. (For an exhaustive example of the definition of the algebraic semantics of a language, see* [GoP81] *or* [Wan80]*).*

Now, we can say that function "*" is correct w.r.t. specification MULT1 if specifications MULT1 and MULT2 (which is the algebraic semantics of function "*") are "equivalent". It is easy to see some properties that this equivalence must fulfil.

1. Two specifications can be equivalent despite having different signatures (in our example, MULT2 has at least one operation (eval_function) which does not belong to MULT1).

2. In fact, we can divide operations of a specification into two classes: those that we wish to specify (such as "*", in our example) and those that we describe only because they are required to define the former (such as "eval_function"). We call them "observable" and "hidden", respectively. It is easy to see that two equivalent specifications must have the same observable operations.

3. Finally, it is well known that a term represents a "computation" of the specified software system. Terms that have some hidden operation represent computation states that are not visible in the external behaviour of the system. Therefore, only terms that only have observable operations (hence, "totally observable terms") should be considered to define the equivalence.

Therefore, we can suppose that two specifications are equivalent if their totally observable terms "behave" in the same way. But this approach is not completely correct.

The problem with this approach becomes clear when we wish to specify data types like stacks, lists, sets, etc. For these data structures, it is advisable to obtain an abstraction of the internal implementation similar to that of the conventional behavioural semantics. To do this, the observable sorts[1] must be divided into two classes. The "nonvisible" ones are the sorts that have abstraction of the internal implementation (they will normally represent data structures). The remaining sorts are "visible" sorts (they will normally represent atomic types).[2] Two specifications are equivalent if their visible results are the same, that is, if their totally observable terms of visible sort "behave" in the same way.

To sum up, two levels of "observability" are needed:

1. That which distinguishes between observable and hidden operations. This level is necessary, since, if we want to specify all the data types that are of interest to us

[1]A sort s is observable if there is an observable operation of sort s.

[2]This issue is explained at great length in [Niv87] for conventional behavioural semantics. In this semantics, "visible" and "nonvisible" sorts are called "observable" and "nonobservable" sorts, respectively.

(the semicomputable data types), hidden operations are needed (for this issue, see [BeT87]).

2. That which distinguishes between visible observable and nonvisible observable sorts. This level is necessary in order to obtain abstraction of the implementation, when this abstraction is required. (This "observability" is that of the conventional behavioural semantics)

Althought there are several formulations of behavioural semantics, there is none which fulfils all the properties stated so far (see [BBK94]). One of the main problems is that behavioural semantics is traditionally defined between algebras that share the same signature and, as we have seen, this is not acceptable here.

This paper aims to introduce an alternative to behavioural semantics which fulfils all these properties and, therefore, is suitable for the proof of algorithms. (In fact, this paper is part of my future thesis, which deals with the problem of automatic verification of algorithms). We refer to this alternative semantics as "visible semantics".

Behavioural semantics can be defined in two ways: by starting from behavioural equivalence or by relaxing the satisfaction relation. However, visible semantics can only be defined in the first way (the reason for this is given in Section 6).

The structure of this paper is as follows. First, in Section 2, the notation that we use is described. Next, visible semantics is completely defined and, finally, we outline some conclusions and some ideas for future work.

2 Basic concepts

We begin by remembering some basic concepts of the theory of algebraic specification.

Definition 1. A S-set C is a family of sets indexed by S, $C = \{C_s\}_{s \in S}$.

Definition 2. A simple signature Σ is a tuple $\Sigma = (S, F)$ where S is a set whose members are called sorts and F is a $S^* \times S$-set ($F = \{F_{w,s}\}_{(w,s) \in S^* \times S}$), whose members are called function symbols or operations.

If $\sigma \in F_{w,s}$, where $w = w1 \times ... \times wn$ with $w1, ..., wn, s \in S$, we say that σ is an operation of domains $w1, ..., wn$ and sort s. We shall write this as follows: $\sigma \in F_{w1..wn,s}$ or $\sigma : w1 \times ... \times wn \longrightarrow s$.

We refer to S as $sorts(\Sigma)$ and to F as $opns(\Sigma)$. Variables of sort s are referred to as $vars(s)$.

Definition 3. Let there be two simple signatures $\Sigma_1 = (S_1, F_1)$ and $\Sigma_2 = (S_2, F_2)$. We shall write $\Sigma_1 \subseteq \Sigma_2$ if $S_1 \subseteq S_2$ and $F_1 \subseteq F_2$.

Definition 4. Let Σ = (S,F) be a simple signature and X a set of variables. The sets $T_{\Sigma_s}(X)$ are defined as follows:

- If $x \in X$ and $x \in vars(s)$, then $x \in T_{\Sigma_s}(X)$.
- If $\sigma \in F_{\lambda,s}$, then $\sigma \in T_{\Sigma_s}(X)$.
- If $\sigma \in F_{w_1 \ldots w_n,s}$, $t_1 \in T_{\Sigma_{w_1}}, \ldots, t_n \in T_{\Sigma_{wn}}$, then $\sigma(t_1, \ldots, t_n) \in T_{\Sigma_s}$.

The members of $T_{\Sigma_s}(X)$ are called terms of sort s.

Definition 5. Let Σ be a simple signature and X a set of variables. We define $T_\Sigma(X) = \{T_{\Sigma_s}(X)\}_{s \in S}$ and $T_\Sigma = T_\Sigma(\emptyset)$. The members of $T_\Sigma(X)$ are called *terms* and those of T_Σ *ground terms*.

Definition 6. Let Σ=(S,F) be a simple signature. A Σ-algebra A is a tuple (A_S, A_F) where $A_S = \{A_s\}_{s \in S}$ and $A_F = \{\sigma_A\}_{\sigma \in F}$ such that:

- If $\sigma \in F_{\lambda,s}$, then $\sigma_A \in A_s$.
- If $\sigma \in F_{w_1 .. w_n,s}$, then $\sigma_A : A_{s_1}, \ldots, A_{sn} \longrightarrow A_s$.

σ_A is called interpretation of the function symbol σ in A.

Definition 7. Let Σ be a simple signature. Let there be a term $t \in T_\Sigma(X)$ and a Σ-algebra A. We define $\varepsilon_A(t)$ as follows:

- If $\sigma \in F_{\lambda,s}$ where $s \in S$, then $\varepsilon_A(\sigma) = \sigma_A$.
- If $\sigma \in F_{w_1 \ldots w_n,s}$, $t_1 \in T_{\Sigma_{w_1}}, \ldots, t_n \in T_{\Sigma_{wn}}$, then $\varepsilon_A(\sigma(t_1, \ldots, t_n)) = \sigma_A(\varepsilon_A(t_1), \ldots, \varepsilon_A(t_n))$.

$\varepsilon_A(t)$ is called evaluation of t in A.

Definition 8. Suppose a simple signature Σ. Let X be a set of variables. A (2*n+3)-tuple $(X, c_1, d_1, \ldots, c_n, d_n, t_1, t_2)$, where $t_1, t_2 \in T_{\Sigma_s}(X)$; $c_1, d_1 \in T_{\Sigma_{s_1}}(X)$; ...; $c_n, d_n \in T_{\Sigma_{sn}}(X)$, with $s, s_1, \ldots, s_n \in sorts(\Sigma)$ is called Σ-equation of arity n (or Σ-equation with n conditions).

We refer to the equation $e = (X, c_1, d_1, \ldots, c_n, d_n, t_1, t_2)$ as $c_1 = d_1$ &...& $c_n = d_n \Rightarrow t_1 = t_2$ (or as $e : c_1 = d_1$ &...& $c_n = d_n \Rightarrow t_1 = t_2$) . The set of variables contained in a equation e is called $vars(e)$.

A Σ-equation of arity 0 is called simple equation or equation without conditions.

Definition 9. Let A be a Σ-algebra. Let X be a set of values. An "assignment of values" is an application $v : X \longrightarrow A$.

Definition 10. Given an application v such that $v : X \longrightarrow A$ or $v : X \longrightarrow T_\Sigma$, $v^*(t)$ is defined as follows:

- If $t \in X$, then $v^*(t) = v(t)$.
- If t has the form $\sigma(t_1, ..., t_n)$, with n ≥ 0, then $v^*(t) = \sigma(v^*(t_1), ..., v^*(t_n))$.

Definition 11. A Σ-algebra A satisfies an equation $e : c_1 = d_1 \& ... \& c_n = d_n \Rightarrow t_1 = t_2$ if $\forall v : vars(e) \longrightarrow A$ it is fulfilled that: $\varepsilon_A(v^*(c_1)) = \varepsilon_A(v^*(d_1)) \wedge ... \wedge \varepsilon_A(v^*(c_n)) = \varepsilon_A(v^*(d_n))$ implies $\varepsilon_A(v^*(t_1)) = \varepsilon_A(v^*(t_2))$. If a Σ-algebra A satisfies an equation e, we shall write $A \models e$.

Definition 12. Let E be a set of equations. We shall write $A \models E$ if $\forall e \in E$ it is fulfilled that $A \models e$.

Definition 13. A simple specification is a tuple $SPEC = (\Sigma, E)$ where Σ is a simple signature and E a set of equations. We refer to E as $eqns(SPEC)$.

Definition 14. Given a simple specification $SPEC$, we define $Alg[SPEC] = \{A \mid A \models eqns(SPEC)\}$

3 Visible signatures

We begin by defining the kind of signature suitable for the theory which we wish to expound. As was justified in the introduction, the signature which interests us must be able to express two levels of "observability". The first level has to distinguish between observable and hidden sorts and operations. The second, between visible and nonvisible sorts, among the observable ones. One way to do this is as follows:

Definition 15. A visible signature is a triple $(Vis, \Sigma_{Obs}, \Sigma_{All})$ such that $\Sigma_{Obs} = (S_{Obs}, F_{Obs})$, $\Sigma_{All} = (S, F)$ are simple signatures and, moreover, $Vis \subseteq S_{Obs}$ and $\Sigma_{Obs} \subseteq \Sigma_{All}$.

We refer to the tuple (Vis, Σ_{Obs}) as an observable signature and, therefore, the members of S_{Obs} and of F_{Obs} are called observable sorts and operations, respectively. We call the members of Vis visible sorts.

We call Σ_{All} an extended signature. The members of $(S \backslash S_{Obs})$ and of $(F \backslash F_{Obs})$ are called hidden sorts and operations, respectively[3].

Definition 16. Let $\Sigma = (Vis, \Sigma_{Obs}, \Sigma_{All})$ be a visible signature. We define $sig\_obs(\Sigma) = (Vis, \Sigma_{Obs})$. Given two visible signatures Σ_1 and Σ_2 such that $sig\_obs(\Sigma_1) = (Vis_1, (\Sigma_1)_{Obs})$ and $sig\_obs(\Sigma_2) = (Vis_2, (\Sigma_2)_{Obs})$, we shall write $sig\_obs(\Sigma_1) = sig\_obs(\Sigma_2)$ if $Vis_1 = Vis_2$ and $(\Sigma_1)_{Obs} = (\Sigma_2)_{Obs}$.

Now, we shall define the concept of term in a visible signature, which is simply a term of the extended signature. A term is totally observable of visible sort if all its operations are observable and, moreover, its sort is visible.

Definition 17. Suppose that $\Sigma = (Vis, \Sigma_{Obs}, \Sigma_{All})$ is a visible signature. We say that A is a Σ-algebra if it is a Σ_{All}-algebra.

We define T_Σ, $T_\Sigma(X)$ as $T_{\Sigma_{All}}$, $T_{\Sigma_{All}}(X)$, respectively. Analogously, for each sort $s \in S$, we define $T_{\Sigma_s}(X)$, T_{Σ_s} as $(T_{\Sigma_{All}})_s(X)$, $(T_{\Sigma_{All}})_s$, respectively.

Definition 18. Let $\Sigma = (Vis, \Sigma_{Obs}, \Sigma_{All})$ be a visible signature. We define $TVis_\Sigma = \{t \mid t \in (T_{\Sigma_{Obs}})_s \wedge s \in Vis\}$.

Obviously, $TVis_\Sigma$ is a subset of T_Σ. The members of $TVis_\Sigma$ are called totally observable (ground) terms of visible sort.

As was justified in the introduction, a totally observable term of visible sort represents a computation result which is externally visible (that is, visible "from outside" the specified system).

Definition 19. Given a Σ-algebra A, we define $A_{TVis} = \{\varepsilon_A(t) \mid t \in TVis_\Sigma\}$.

Now, we shall introduce the category suitable for dealing with the visible semantics. To do this, first we introduce the concepts of visible morphism and visible equivalence.

Definition 20. Let Σ_A and Σ_B be two visible signatures such that $sig\_obs(\Sigma_A) = sig\_obs(\Sigma_B)$. Let A be a Σ_A-algebra and B a Σ_B-algebra. A visible morphism is a function f between A_{TVis} and B_{TVis} such that, for any $t \in TVis_{\Sigma_A}$ it is fulfilled that: $f(\varepsilon_A(t)) = \varepsilon_B(t)$.

If this application is bijective, it is called visible isomorphism.

Lemma 21. Let us suppose that Σ_A and Σ_B are two visible signatures such that $sig\_obs(\Sigma_A) = sig\_obs(\Sigma_B)$. Let A be a Σ_A-algebra and B a Σ_B-algebra. Only one of the

[3]On occassions, we shall use "nonobservable" as a synonym of "hidden"

following two cases may occur:

1. There is no visible morphism between A and B.
2. There is one single visible morphism between A and B.

Proof. The proof is in the appendix of this paper.

Notice that, in order to define a visible morphism between two algebras, they need not have the same signature; only the same observable *signature. This is coherent with our philosophy which states that hidden sorts and operations are only useful for defining the behaviour of the observable one, but are not visible "from the outside". The same is true of the following category.*

Definition 22. The category $Visible(Vis, \Sigma_{Obs})$ contains all the Σ-algebras such that $sig\_obs(\Sigma) = (Vis, \Sigma_{Obs})$ as objects, and the visible morphisms as morphisms.

Now we shall define the visible equivalence between two algebras. This concept is also defined between algebras that share the same observable signature.

Definition 23. Let Σ_A and Σ_B be two visible signatures such that $sig\_obs(\Sigma_A) = sig\_obs(\Sigma_B)$. Let A be a Σ_A algebra and B a Σ_B-algebra. A and B are visibly equivalent (and we shall write $A \equiv_V B$) if there is a visible isomorphism between A and B.

In the introduction, it was stated that the visible equivalence must take into account only the totally observable terms of visible sort. This can be seen in the following lemma.

Lemma 24. Let Σ_A and Σ_B be two visible signatures such that $sig\_obs(\Sigma_A) = sig\_obs(\Sigma_B)$. Let A be a Σ_A-algebra and B a Σ_B-algebra. A and B are visibly equivalent if and only if [4]:

$$\forall\, t_1, t_2 \in TVis_{\Sigma_A} \text{ it is fulfilled that } A \models t_1 = t_2 \text{ if and only if } B \models t_1 = t_2$$

Proof. The proof is in the appendix of this paper.

4 Visible satisfaction of an equation

Following the philosophy of visible semantics, an equation is satisfied if all its totally observable consequences of visible sort are so. In order to formalize this intuitive idea, first we must state the following definitions:

[4]On the other hand, notice that $TVis_{\Sigma_A} = TVis_{\Sigma_B}$.

Definition 25. Let us suppose that Σ is a visible signature and X is a set of variables. We refer to a Σ_{All}-equation as a Σ-equation.

Definition 26. Let Σ be a visible signature. A visible Σ-context of sort s is a term $vc[z]$ belonging to $T_\Sigma(z)$ where z is a variable of sort s.

Definition 27. Given a term $t \in T_\Sigma(X)$ and a visible Σ-context $vc[z]$, we refer to the term obtained by replacing z in $vc[z]$ by t as $vc[t]$ (that is to say, $vc[t] = vc[z \leftarrow t]$).

$vc[t]$ is any term for which t is a subterm. A totally observable consequence of visible sort (of a term t) can be defined as a totally observable term of visible sort which can be derived from t (or from contexts of t). Therefore, visible satisfaction can be defined as follows:

Definition 28. Let Σ be a visible signature. Let $e : t_1 = t_2$ be a *ground* Σ-equation (without conditions) of sort s (that is, $t_1, t_2 \in T_{\Sigma_s}$). A Σ-algebra A visibly satisfies e (and we shall write $A \models_V e$) if, for any visible context $vc[z]$ of sort s, it is fulfilled that:

$$\forall\, l_1, l_2 \in TVis_\Sigma \quad (l_1 \equiv_A vc[t_1]) \wedge (l_2 \equiv_A vc[t_2]) \text{ implies } l_1 \equiv_A l_2$$

Definition 29. Let Σ be a visible signature. Let $e : c_1 = d_1 \,\&...\&\, c_n = d_n \Rightarrow t_1 = t_2$, be a Σ-equation. A Σ-algebra A visibly satisfies e (and we shall write $A \models_V e$) if:

$$\forall\, v : vars(e) \longrightarrow T_\Sigma \text{ it is fulfilled that}$$
$$(A \models_V v^*(c_1) = v^*(d_1)) \wedge ... \wedge (A \models_V v^*(c_n) = v^*(d_n)) \text{ implies } (A \models_V v^*(t_1) = v^*(t_2))$$

Analogously to initial and behavioural semantics, we can define the concept of theory taking as a starting point the satisfaction of an equation.

Definition 30. Let us suppose that Σ is a visible signature and that M is a set of Σ-algebras. We refer to the set of all the Σ-equations that are visibly satisfied by all the algebras of M as the visible theory of M (and we shall write $VisTheo(M)$).

$$VisTheo(M) = \{e \mid \forall\, A \in M \quad A \models_V e\}$$

If A is a Σ-algebra, we refer to $VisTheo(\{A\})$ as $VisTheo(A)$.

5 Visible specifications

Given the concept of the visible signature, that of the visible specification is easily definable.

Definition 31. A visible specification is a tuple $SPEC = (\Sigma, E)$, where $\Sigma = (Vis, \Sigma_{Obs}, \Sigma_{All})$ is a visible signature and E is a set of Σ-equations. We define $sig(SPEC) = \Sigma$, $sig\_obs(SPEC) =$

(Vis, Σ_{Obs}) and $eqns(SPEC) = E$.

Definition 32. Let $SPEC$ be a visible specification such that $sig(SPEC) = \Sigma$. We refer to T_Σ as $T_{\Sigma_{SPEC}}$. Analogously, we define $T_{\Sigma_{SPEC}}(X)$ and $(T_{\Sigma_{SPEC}})_s$. We refer to $TVis_\Sigma$ as $TVis_{SPEC}$.

Definition 33. Let us suppose that $SPEC = (\Sigma, E)$ is a visible specification where $\Sigma = (Vis, \Sigma_{Obs}, \Sigma_{All})$. We define the following concepts:

1. $\equiv_{SPEC}$ *(congruence defined by specification SPEC)*
2. $[t]_{\equiv_{SPEC}}$ *(class of equivalence of term t, according to this congruence)*
3. T_{SPEC} *(initial algebra of SPEC)*

as, respectively,

1. $\equiv_{SPEC'}$
2. $[t]_{\equiv_{SPEC'}}$
3. $T_{SPEC'}$

where SPEC' is the simple specification defined as $SPEC' = (\Sigma_{All}, E)$.

6 Semantics of a visible specification

In this section, we shall define the visible semantics of a specification. To do this, we shall start from the principle that two visibly equivalent algebras should be considered equal according to visible semantics.

Definition 34. Let $SPEC$ be a visible specification. We define the loose visible semantics of $SPEC$ as:

$$Visible[SPEC] = \{A \mid \exists B \in Alg[SPEC] \text{ such that } B \equiv_V A\}$$

That is to say, the algebras belonging to Visible[SPEC] are those that are visibly equivalent to the algebras that fulfil the equations of SPEC.

In behavioural semantics, there is the "good" property that two algebras are equivalent if they share the same theory. This enables us to define the (loose) behavioural semantics of an algebraic specification as the set of algebras that behaviourally satisfy their equations.

However, in visible semantics, two equivalent algebras need not satisfy the same equations since they can have different signatures and, consequently, the equations of one algebra need not make sense in the signature of the other. Therefore, the only way to define the (loose) visible semantics of a specification is as we did it: starting from the notion of visible equivalence.

We also define the initial and final visible semantics starting from this notion, as we shall see below.

Definitiôn 35. The initial and final visible semantics are defined respectively as:

$$Vis - I[SPEC] = \{A \mid A \equiv_V T_{SPEC}\}$$
$$Vis - F[SPEC] = \{A \mid A \equiv_V F[SPEC]\}$$

, where T_{SPEC} and $F[SPEC]$ are the initial and the final algebras of $SPEC$, respectively.

7 Conclusions

We have seen which properties are required for an algebraic semantics to be able to deal with the problem of program correctness. A new kind of semantics (called "visible") which fulfils all these properties has been defined. We have described the new concepts of visible morphism, visible category, visible equivalence, visible satisfaction of an equation and loose, initial and final visible semantics of a specification. It has been explained that the concept of satisfaction does not serve to define the visible semantics of an algebraic specification, unlike in behavioural semantics.

The lines of future research are twofold. Firstly, visible semantics will be extended to support generic specifications such as "STACK[T]", where T can be any sort (for more details of this kind of specifications, see [TWW79b]).

Secondly, this paper is part of my future thesis, which propounds a method to automatically deduce the correctness of a program w.r.t. an algebraic specification. The notion of correctness that we shall use in this thesis will be defined starting from visible semantics, which has been described here.

8 References

[BBK94] BERNOT, G. BIDOIT, M.; KNAPIK, T. *Behavioural approaches to algebraic specifications.* Acta Informatica, 31 (1994), pp. 651-671.

[BeT87] BERGSTRA, J.A. TUCKER, J.V. *Algebraic Specifications of Computable and Semicomputable Data Types.* Theoretical Computer Science, 50 (1987), pp. 186-200.

[GoP81] GOGUEN, J.A. PARSAYE-GHOMI, K. *Algebraic Denotational Semantics Using Parameterized Abstract Modules.* Formalization of Programming Concepts, LNCS 107 (1981), pp. 292-309.

[GTW75] GOGUEN, J.A. THATCHER, J.W. WAGNER, E.G. WRIGHT, J.B. *Abstract data types as initial algebras and correctness of data representations.* Proc. ACM Conference on Computer Graphics, Pattern and Data Structure, New York (1975), pp. 89-93.

[GTW78] GOGUEN, J.A. THATCHER, J.W. WAGNER, E.G. *An initial algebra approach to the specification, correctness and implementations of abstract data types.*, in: R.T.Yeh, ed., Current Trends in Programming Methodology; IV Data Structuring (Prentice-Hall, Englewood Clifts, NJ, 1978) pp. 80-149.

[Kna93] KNAPIK, T. *Spécifications algébriques observationnelles modulaires: une semantique fondée sur une relation de satisfaction observationnelle.* Thèse de l'Université de Paris-Sud, Orsay 1993.

[Niv87] NIVELA, P. *Semántica de Comportamiento en Lenguajes de Especificación.* PhD thesis, directed by Fernando Orejas Valdés, Barcelona, 1987. Universitat Politècnica de Catalunya. Facultat d'Informàtica.

[Rei81] REICHEL, H. *Behavioural equivalence – a unifying concept for initial and final specification methods.* Proceedings third Hungarian Computer Science Conference. Budapest (1981), pp. 27-39.

[Rei84] REICHEL, H. *Behavioral validity of equations in abstract data types.* Contributions to General Algebra 3, Proceedings of the Vienna Conference, Verlag B. G. Teubner, Stuttgart (1985), pp. 301-324.

[SaT85] SANNELLA, D. TARLECKY, A. *On observational equivalence and algebraic specification.* Journal of Computer and System Science, 34 (1987), pp. 150-178.

[SaW83] SANNELLA, D. WIRSING, M. *A kernel language for algebraic specification and implementation.* Proceedings International Conference on Foundations of Computation Theory. Sweden. Springer LNCS 158 (1983), pp. 413-427.

[TWW79a] THATCHER, J.W. WAGNER, E.G. WRIGHT, J.B. *Specifications of abstract data types using condicional axioms.* IBM Research Report RC 6214, Yorktown Heigths, NY, 1979.

[TWW79b] THATCHER, J.W WAGNER, E.G WRIGHT, J.B *Data type specifications: para-metrization and the power of specifications techniques.* IBM Research Report RC

7757, Yorktown Heights, NY, 1979.

[Wan80] WAND, M. *First-Order Identities as a Defining Language.* Acta Informatica, 14 (1980), pp. 337-357.

A Appendix: Proof of lemmas 21 and 24

In the present appendix, we include the proofs of lemmas 21 and 24, which were been stated in Section 3.

A.1 Proof of lemma 21

A priori, it seems that there are three possible cases:

1. There is no visible morphism between A and B.
2. There is one single visible morphism between A and B.
3. There are several morphisms between A and B.

Consequently, we must prove that cases 1 and 2 may occur and case 3 may not.

1. The following example shows that case 1 may occur:

   Let us suppose a visible signature $\Sigma = (Vis, \Sigma_{Obs}, \Sigma_{All})$ such that:

   $Vis = sorts(\Sigma_{Obs})$
   $\Sigma_{Obs} = \Sigma_{All}$
   $sorts(\Sigma_{All}) = \{sort1\}$
   $opns(\Sigma_{All}) = \{op1, op2\}$ where $op1, op2 :\longrightarrow sort1$

   It is easy to see that totally observable terms of visible sort are op_1 and op_2 in this signature. Now, let us consider the Σ-algebras A and B such that:

$A_{sort1} = \{\bullet\}$	$B_{sort1} = \{*, \#\}$
$op1_A = \bullet$	$op1_B = *$
$op2_A = \bullet$	$op2_B = \#$

   It is obvious that no visible morphism can be defined between A and B, since the only mapping between A and B which fulfils $f(\varepsilon_A(t)) = \varepsilon_B(t)$ (for any $t \in TVis_{\Sigma_A}$) is not an application because "$\bullet$" has two possible images in B ("$*$" and "$\#$").

2. Now, let us see that case 2 may occur. The example in which $A = B$ shows this. In this example, a visible morphism is a function which fulfils $f(\varepsilon_A(t)) = \varepsilon_A(t)$, (for any $t \in TVis_{\Sigma_A}$). It is obvious that this morphism exists and that it is the identity function.

3. Finally, let us see that case 3 may not occur. We shall prove it by contradiction. Let us suppose that there are two visible morphisms between A and B, called f_1 and f_2. By applying the definition, for any $t \in TVis_{\Sigma_A}$, it is fulfilled that:

   $$f_1(\varepsilon_A(t)) = \varepsilon_B(t)$$
   $$f_2(\varepsilon_A(t)) = \varepsilon_B(t)$$

   By properties of equality, this means that, for any $t \in TVis_{\Sigma_A}$, it is fulfilled that:

   $$f_1(\varepsilon_A(t)) = f_2(\varepsilon_A(t))$$

   That is to say, the two morphisms are the same, and, therefore, uniqueness is proved.
   □

A.2 Proof of lemma 24

Since lemma 24 contains an "if and only if", we must prove both "directions" of the double implication:

- First, let us prove that, if A and B are visibly equivalent, then:

  $$\forall\, t_1, t_2 \in TVis_{\Sigma_A} \text{ it is fulfilled that } A \models t_1 = t_2 \text{ if and only if } B \models t_1 = t_2$$

  Let us suppose that A and B are visibly equivalent. This means that there is a visible isomorphism f between A and B. Since f is a bijection, it is fulfilled that:

  $$\forall\, a_1, a_2 \in A_{TVis} \quad a_1 = a_2 \text{ if and only if } f(a_1) = f(a_2)$$

  By the definition of A_{TVis}, $x \in A_{TVis}$ if and only if $\exists\, t \in TVis_{\Sigma_A}$ such that $\varepsilon_A(t) = x$. Then, the above statement is equivalent to the following one:

  $$\forall\, t_1, t_2 \in TVis_{\Sigma_A} \quad \varepsilon_A(t_1) = \varepsilon_A(t_2) \text{ if and only if } f(\varepsilon_A(t_1)) = f(\varepsilon_A(t_2))$$

  Now, since f is a visible morphism between A and B, for any $t \in TVis_{\Sigma_A}$, $f(\varepsilon_A(t)) = \varepsilon_B(t)$. Therefore, we obtain:

  $$\forall\, t_1, t_2 \in TVis_{\Sigma_A} \quad \varepsilon_A(t_1) = \varepsilon_A(t_2) \text{ if and only if } \varepsilon_B(t_1) = \varepsilon_B(t_2)$$

By applying the definition of satisfaction of an equation in a given algebra and the fact that t_1 and t_2 are ground terms[5], this is equivalent to the following statement:

$$\forall t_1, t_2 \in TVis_{\Sigma_A} \text{ it is fulfilled that } A \models t_1 = t_2 \text{ if and only if } B \models t_1 = t_2$$

, which is what we wished to prove.

- Let us prove the reciprocal implication, that is, that the above statement implies visible,equivalence between A and B. By applying the definition of satisfaction of an equation in a given algebra and the fact that t_1 and t_2 are ground terms, this statement is equivalent to the following one (which we shall call "statement α"):

$$\forall t_1, t_2 \in TVis_{\Sigma_A} \quad \varepsilon_A(t_1) = \varepsilon_A(t_2) \text{ if and only if } \varepsilon_B(t_1) = \varepsilon_B(t_2)$$

Consequently, we must proof that, if statement α is fulfilled, there is a visible isomorphism between A and B.

To do this, let us suppose that statement α is fulfilled. We define f as the mapping which fulfils $f(\varepsilon_A(t)) = \varepsilon_B(t)$. If we prove that f is a bijection, we shall prove that A and B are visibly equivalent. We divide the proof into three parts:

  - First, let us see that f is an application, that is,

$$\forall a_1, a_2 \in A_{TVis} \quad a_1 = a_2 \text{ implies } f(a_1) = f(a_2)$$

By the definition of A_{TVis}, $x \in A_{TVis}$ if and only if $\exists t \in TVis_{\Sigma_A}$ such that $\varepsilon_A(t) = x$. Then, the previous statement is equivalent to the following one:

$$\forall t_1, t_2 \in TVis_{\Sigma_A} \quad \varepsilon_A(t_1) = \varepsilon_A(t_2) \text{ implies } f(\varepsilon_A(t_1)) = f(\varepsilon_A(t_2))$$

Now, since f is defined as $f(\varepsilon_A(t)) = \varepsilon_B(t)$, this means that:

$$\forall t_1, t_2 \in TVis_{\Sigma_A} \quad \varepsilon_A(t_1) = \varepsilon_A(t_2) \text{ implies } \varepsilon_B(t_1) = \varepsilon_B(t_2)$$

which is fulfilled, by statement α.

  - Now, let us see that f is injective, that is,

$$\forall a_1, a_2 \in A_{TVis} \quad f(a_1) = f(a_2) \text{ implies } a_1 = a_2$$

[5]And, consequently, for any assignment of values v, it is fulfilled that $v^*(t_1) = t_1$ and $v^*(t_2) = t_2$.

By the definition of A_{TVis}, $x \in A_{TVis}$ if and only if $\exists\, t \in TVis_{\Sigma_A}$ such that $\varepsilon_A(t) = x$. Therefore, the above statement is equivalent to the following one:

$$\forall\, t_1, t_2 \in TVis_{\Sigma_A} \quad f(\varepsilon_A(t_1)) = f(\varepsilon_A(t_2)) \text{ implies } \varepsilon_A(t_1) = \varepsilon_A(t_2)$$

Now, since f is defined as $f(\varepsilon_A(t)) = \varepsilon_B(t)$, this means that:

$$\forall\, t_1, t_2 \in TVis_{\Sigma_A} \quad \varepsilon_B(t_1) = \varepsilon_B(t_2) \text{ implies } \varepsilon_A(t_1) = \varepsilon_A(t_2)$$

which is fulfilled, by statement α.

– Let us see that f is exhaustive, that is,

$$\forall\, b \in B_{TVis} \quad \exists\, a \in A_{TVis} \text{ such that } f(a) = b$$

By the definition of B_{TVis}, $x \in B_{TVis}$ if and only if $\exists\, t \in TVis_{\Sigma_B}$ such that $\varepsilon_B(t) = x$. Therefore, the above statement is equivalent to the following one (remember that $TVis_{\Sigma_A} = TVis_{\Sigma_B}$):

$$\forall\, t \in TVis_{\Sigma_A} \quad \exists\, a \in A_{TVis} \text{ such that } f(a) = \varepsilon_B(t)$$

Now, by the definition of f, we know that $f(\varepsilon_A(t)) = \varepsilon_B(t)$. Moreover, by the definition of A_{TVis}, $\varepsilon_A(t) \in A_{TVis}$. Consequently, if we make $\varepsilon_A(t)$ be a, the above statement is fulfilled and, therefore, exhaustivity is proved. □

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

Research Reports – 1996

LSI-96-1-R "(Pure) Logic out of Probability", Ton Sales.

LSI-96-2-R "Automatic Generation of Multiresolution Boundary Representations", C. Andújar, D. Ayala, P. Brunet, R. Joan-Arinyo, and J. Solé.

LSI-96-3-R "A Frame-Dependent Oracle for Linear Hierarchical Radiosity: A Step towards Frame-to-Frame Coherent Radiosity", Ignacio Martin, Dani Tost, and Xavier Pueyo.

LSI-96-4-R "Skip-Trees, an Alternative Data Structure to Skip-Lists in a Concurrent Approach", Xavier Messeguer.

LSI-96-5-R "Change of Belief in SKL Model Frames (Automatization Based on Analytic Tableaux)", Matías Alvarado and Gustavo Núñez.

LSI-96-6-R "Compressibility of Infinite Binary Sequences", José L. Balcázar, Ricard Gavaldà, and Montserrat Hermo.

LSI-96-7-R "A Proposal for Word Sense Disambiguation using Conceptual Distance", Eneko Agirre and German Rigau.

LSI-96-8-R "Word Sense Disambiguation Using Conceptual Density", Eneko Agirre and German Rigau.

LSI-96-9-R "Towards Learning a Constraint Grammar from Annotated Corpora Using Decision Trees", Lluis Màrquez and Horacio Rodríguez.

LSI-96-10-R "POS Tagging Using Relaxation Labelling", Lluís Padró..

LSI-96-11-R "Hybrid Techniques for Training HMM Part-of-Speech Taggers", Ted Briscoe, Greg Grefenstette, Lluís Padró, and Iskander Serail.

LSI-96-12-R "Using Bidirectional Chart Parsing for Corpus Analysis", A. Ageno and H. Rodríguez.

LSI-96-13-R "Limited Logical Belief Analysis", Antonio Moreno.

LSI-96-14-R "Logic as General Rationality: A Survey", Ton Sales.

LSI-96-15-R "A Syntactic Characterization of Bounded-Rank Decision Trees in Terms of Decision Lists", Nicola Galesi.

LSI-96-16-R "Algebraic Transformation of Unary Partial Algebras I: Double-Pushout Approach", P. Burmeister, F. Rosselló, J. Torrens, and G. Valiente.

LSI-96-17-R "Rewriting in Categories of Spans", Miquel Monserrat, Francesc Rosselló, Joan Torrens, and Gabriel Valiente.

LSI-96-18-R "Strong Law for the Depth of Circuits", Tatsuie Tsukiji and Fatos Xhafa.

LSI-96-19-R "Learning Causal Networks from Data", Ramon Sangüesa i Solé.

LSI-96-20-R "Boundary Generation from Voxel-based Volume Representations", R. Joan-Arinyo and J. Solé.

LSI-96-21-R "Exact Learning of Subclasses of CDNF Formulas with Membership Queries", Carlos Domingo.

LSI-96-22-R "Modeling the Thermal Behavior of Biosphere 2 in a Non-Controlled Environment Using Bond Graphs", Angela Nebot, François E. Cellier, and Francisco Mugica.

LSI-96-23-R "Obtaining Synchronization-Free Code with Maximum Parallelism", Ricard Gavaldá, Eduard Ayguadé, and Jordi Torres.

LSI-96-24-R "Memoisation of Categorial Proof Nets: Parallelism in Categorial Processing", Glyn Morrill.

LSI-96-25-R "Decision Trees Have Approximate Fingerprints", Víctor Lavín and Vijay Raghavan.

LSI-96-26-R "Visible Semantics: An Algebraic Semantics for Automatic Verification of Algorithms", Vicent-Ramon Palasí Lallana.
