Reihe Informatik

29/95

How to interpret and establish consistency results for semantics of concurrent programming languages

C. Baier, M.E. Majster-Cederbaum

How to interpret and establish consistency results for semantics of concurrent programming languages

Christel Baier, Mila Majster-Cederbaum
Fakultät für Mathematik und Informatik
Universität Mannheim, 68131 Mannheim, Germany
{baier,mcb}@pi1.informatik.uni-mannheim.de

February 14, 1996

Abstract

It is meaningful that a language is provided with several semantic descriptions: e.g. one which serves the needs of the implementor, another one that is suitable for specification and yet another one that will be used to explain the language to the user. In this case one has to guarantee that the various semantics are 'consistent'. The attempt of this paper is to clarify the notion 'consistency' and to present a general framework and theorems for consistency results.

Contents

1 Introduction

Various methods to define the semantics of languages that allow for parallelism, communication and synchronisation have been proposed in the last years. They can be classified by several criteria as e.g. operational versus denotational versus axiomatic methods, interleaving versus true parallelism approaches, branching time versus linear time models, event versus action based models, choice of mathematical discipline to assist the handling of recursion and the solution of domain equations. A variety of semantical desriptions have been proposed for given languages like CCS or $TCSP$ and in the sequel 'consistency' results that relate different semantics in some way have been established (e.g. [3, 6, 10, 14, 18, 25, 29]). The aim of this paper is to clarify these notions of consistency. For this purpose we take the following view of semantics:

The semantics of a (nondeterministic) program P should give a description of the (possible) behaviours of P which provides the possibility to prove that P satisfies its specification, i.e. to verify P. There are different ways to give specifications for programs: In process algebra the languages for specifications and implementations coincide and satisfaction usally means either some kind of equivalence $\equiv$ (e.g. strong or weak bisimulation equivalence [32] or failure equivalence [11]) or some kind of preorder $\sqsubseteq$ (e.g. partial bisimulation [20] or some kind of testing preorder [34]). Then a program P satisfies a specification S iff $P \equiv S$ resp. $P \sqsubseteq S$ or equivalently iff $P \in E_S$ where E_S means the equivalence class of S resp. the set of predecessors of S. In the logical approach a specification is a logical formula (or a set of formulas). The usual way to define what is meant by 'a program P satisfies a formula S (or a set of formulas)' is to fix a semantics α such that the meaning $\alpha(P)$ of P yields an interpretation for the underlying logic. P satisfies S iff $\alpha(P)$ is a model of S, i.e. S evaluates to true under the interpretation $\alpha(P)$. For instance, transition system semantics yield interpretations of modal logic [21], string semantics can be used to interpret linear time temporal logics [26, 38], computation tree semantics for interpreting branching time temporal logics [13], petri net semantics to interpret concurrent temporal logics [40]. Regardless the formalism in which the specification is given a specification S can be considered as a subset E_S of the set of programs $\mathcal{P}$ which is under consideration:

$$E_S = \{ P \in \mathcal{P} : P \text{ satisfies } S \}$$

Given a specification S for a program P, verification of P means proving $P \in E_S$. In the following we assume that for the set $\mathcal{P}$ of programs under consideration there is a collection

Prop of interesting properties which a given program can fulfill or not and a specification is an intersection of such properties. We say a semantics $\alpha : \mathcal{P} \to A$ is 'suitable for checking a property E' iff for all $P \in \mathcal{P}$: $\alpha(P) \in \alpha(E)$ implies $P \in E$ or equivalently iff $\alpha^{-1}(\alpha(E)) = E$. Then instead of proving $P \in E$ one proves $\alpha(P) \in \alpha(E)$ which might be easier since $\alpha(P)$ is an abstraction of P. In this sense 'suitable for checking property E' does not mean that the semantics α really offers a method for testing whether program P has property E or not. This is because it might be the case that the problem whether a given element $a \in A$ is contained in the set $\alpha(E)$ is undecidable. It only ensures that α distinguishes programs satisfying property E from those programs not satisfying property E. This a necessary (but not sufficient) condition for the capability to test property E with help of the semantics α.

By the 'consistency' of two semantics $\alpha_1 : \mathcal{P} \to A_1$, $\alpha_2 : \mathcal{P} \to A_2$ w.r.t. the set *Prop* of properties we mean the equivalence of α_1, α_2 w.r.t. the capability of proving that a program satisfies its specification: We say that α_1, α_2 are consistent w.r.t. *Prop* iff both α_1, α_2 are suitable to check the properties $E \in$ *Prop*. This notion of consistency is equivalent to the existence of a semantics $\beta : \mathcal{P} \to B$ which is suitable for checking the properties $E \in$ *Prop* and functions $f_i : A_i \to B$ such that $f_1 \circ \alpha_1 = f_2 \circ \alpha_2 = \beta$. (Theorem 2.10). Many results in the literature can be interpreted as consistency results in this sense (e.g. [3, 9, 10, 18, 19, 39]). In order to show the consistency w.r.t. a set of properties one has to deal with equations of the form $f \circ \alpha = \beta$ where $\alpha : \mathcal{P} \to A$, $\beta : \mathcal{P} \to B$ are semantics and $f : A \to B$ is a function. In this case we say β is an abstraction of α and α, β are weakly consistent. In the literature a variety of weak consistency results has been established. E.g. [6, 7, 14, 25, 41] contain weak consistency results of the form $f \circ \mathcal{O} = \mathcal{D}$ where $\mathcal{O}$ is an operational, $\mathcal{D}$ a denotational semantics. [4] shows that the pomset semantics of [7] is an abstraction of the denotational prime event structure semantics of [18]. [30] gives a cpo-based denotational semantics $\mathcal{D}_{\mathrm{cpo}}$ and a denotational semantics $\mathcal{D}_{\mathrm{cms}}$ based on the metric approach for *CSP* and establishes a weak consistency result of the form $f \circ \mathcal{D}_{\mathrm{cms}} = \mathcal{D}_{\mathrm{cpo}}$.

This paper investigates the notion of consistency of two (or more) semantics and presents sufficient and necessary conditions for establishing consistency results. Section 2 introduces the several notions of consistency (consistency w.r.t. a set of properties, consistency relative to a more abstract semantics, weak and strong consistency) where no restrictions on the syntax of the language in which the programs are written or on the semantics are made. Theorem 2.4 shows that for a given specification formalism (i.e. a set of properties) there exists a semantics which identifies exactly those programs which cannot be distinguished by any of the given properties. I.e. this semantics is suitable for proving the correctness of programs whose specifications are given in this specification formalism. Theorem 2.11 and Theorem 2.12 show the existence of most concrete common abstractions resp. most abstract common refinements for arbitrary families of semantics. Most concrete common abstractions combine the different views which are given by the semantics since it identifies exactly those programs which are identified by each of the semantics. The most abstract common refinement can be used to check properties which can be checked by at least one of the semantics. Section 3 deals with compositional semantics where the existence of semantic operators is required but no additional structures like metric or partial order is assumed. Theorem 3.9 asserts the existence of a compositional semantics which is suitable for modular verification of programs. We show that the most concrete common abstraction as well as the most abstract refinement of compositional semantics are compositional (Theorem 3.10 and 3.11). In section 4 we show the relation between waek consistency and adequacy and full abstraction. Section 5 extends the results of [4] where a general framework for dealing with denotational semantics in the metric and cpo approach is presented. It presents conditions for establishing weak consistency results $f \circ \alpha = \beta$ where α is an arbitrary compositional semantics and β either a metric semantics (section 5.1) or a cpo semantics (section 5.2). In the appendix we give a categorical characterisation of weak consistency.

2 Several notions of consistency

This section defines the notions consistency w.r.t. a set of properties, consistency relative to a semantics β, strong and weak consistency and presents sufficient and necessary conditions for establishing consistency results in this sense. We assume a fixed set $\mathcal{P}$ of programs where we do not make any restriction on the programming language in which the programs are written. The semantics under consideration are arbitrary functions $\alpha : \mathcal{P} \to A$. A property is any subset of $\mathcal{P}$. A program P has property E iff $P \in E$. A specification formalism is a set *Prop* of properties and a specification (in this specification formalism) is an intersection of properties in *Prop*.

2.1 The existence of a semantics suitable for a given specification formalism

Informally, a semantics $\alpha : \mathcal{P} \to A$ is suitable for a specification formalism iff it distinguishes programs satisfying a certain specification (of this specification formalism) from those not satisfying this specification. Formally:

Definition 2.1 *Let E be a property. A semantics $\alpha : \mathcal{P} \to A$ is* suitable for checking E *iff $\alpha^{-1}(\alpha(E)) = E$. In this case we write $\alpha \models E$.*

If Prop is a specification formalism and α a semantics then α is suitable for Prop iff $\alpha \models E$ for all $E \in$ Prop.

In order to prove that $\alpha \models E$ it is sufficient to show that $\alpha^{-1}(\alpha(E)) \subseteq E$ (since E is always a subset of $\alpha^{-1}(\alpha(E))$.)

In Theorem 2.4 we show that for each specification formalism (i.e. each set of properties) there exists a semantics α which is suitable to verify the correctness of programs w.r.t. to this specification formalism in the sense that for each property E under consideration: $\alpha \models E$. Moreover we show the existence of such a semantics α which is the most 'abstract' one under all those semantics.

Definition 2.2 *Let $\alpha : \mathcal{P} \to A$ and $\beta : \mathcal{P} \to B$ be semantics. β is called an* abstraction *of α iff there exists a function $f : A \to B$ such that $f \circ \alpha = \beta$. In this case we say f is an* abstraction from α to β.

Lemma 2.3 *Let $\alpha : \mathcal{P} \to A$ and $\beta : \mathcal{P} \to B$ be semantics. The following are equivalent:*

(a) β is an abstraction of α.

(b) For all $P, Q \in \mathcal{P}$: $\alpha(P) = \alpha(Q)$ implies $\beta(P) = \beta(Q)$.

(c) For all properties E: $\beta \models E$ implies $\alpha \models E$.

Proof:

(a) $\Longrightarrow$ (c): Let $\beta = f \circ \alpha$ and $\beta \models E$. We have to show that $\alpha^{-1}(\alpha(E)) \subseteq E$. Since $\beta = f \circ \alpha$ and since $f^{-1}(f(\alpha(E))) \supseteq \alpha(E)$ we have:

$$E = \beta^{-1}(\beta(E)) = \alpha^{-1}(f^{-1}(f(\alpha(E)))) \supseteq \alpha^{-1}(\alpha(E))$$

(c) $\Longrightarrow$ (b): Let $\alpha(P) = \alpha(Q)$. We put $E = \beta^{-1}(\beta(P))$. We have to show that $Q \in E$. It is clear that $\beta \models E$. By assumption (c) we have: $\alpha \models E$. Since $P \in E$ and $\alpha(P) = \alpha(Q)$ we have:

$$Q \in \alpha^{-1}(\alpha(P)) \subseteq \alpha(\alpha(E)) = E$$

(b) $\Longrightarrow$ (a): Let $b \in B$ be a fixed element. We define:

$$f : A \to B, \ f(a) = \begin{cases} \beta(P) & : \ \text{if } a = \alpha(P) \\ b : & \ \text{if } a \notin \alpha(\mathcal{P}). \end{cases}$$

f is welldefined by assumption (b) and $f \circ \alpha = \beta$. □

Often a property is described in terms of a semantics $\alpha : \mathcal{P} \to A$, i.e. it is of the form $\alpha^{-1}(A_0)$ where A_0 is a subset of A. In this case the semantics α (and by Lemma 2.3 each semantics for which α is an abstraction) is suitable for checking the property $\alpha^{-1}(A_0)$. For instance, the termination property E for a CCS-like language can easily be defined by a tree semantics: We put $E = \alpha^{-1}(A_0)$ where A_0 is the set of trees of finite height and α a tree semantics in the sense of [32] or [42]. Having a semantics $\alpha : \mathcal{P} \to A$ and a logic L such that formulas of L can be interpreted over the elements of A (e.g. α is a transition system semantics and L a branching time logic like CTL [13] or a modal logic like HML [21]) then every formula φ of L induces a property $E_\varphi = \alpha^{-1}(A_\varphi)$ where A_φ is the set of elements $a \in A$ such that φ evaluates to true under the interpretation a.

Theorem 2.4 *Let Prop be a set of properties. There exists a semantics α such that:*

(1) $\alpha \models E$ for all $E \in Prop$

(2) Whenever β is a semantics also satisfying (1) then α is an abstraction of β.

Proof: If $P \in \mathcal{P}$ then we put $\mathcal{E}_P = \{E \in Prop : P \in E\}$. We define the following equivalence relation on $\mathcal{P}$:

$$P \equiv Q \ :\Longleftrightarrow \ \mathcal{E}_P = \mathcal{E}_Q$$

Let $A = \mathcal{P}/\equiv$ be the set of equivalence classes and let $\alpha : \mathcal{P} \to A$ be the canonical function $\alpha(P) = [P]$ where $[P]$ is the equivalence class of P.

Claim 1: $\alpha \models E$ for all $E \in Prop$.

Proof: If $E \in Prop$ and $P \in \alpha^{-1}(\alpha(E))$ then $\alpha(P) = \alpha(Q)$ for some $Q \in E$. I.e. $P \equiv Q$. Hence $E \in \mathcal{E}_Q = \mathcal{E}_P$ and therefore $P \in E$.

Claim 2: If $\beta : \mathcal{P} \to B$ is also a semantics with $\beta \models E$ for all $E \in Prop$ then α is an abstraction of β.

Proof: By Lemma 2.3 it is sufficient to show that $\beta(P) = \beta(Q)$ implies $\alpha(P) = \alpha(Q)$. Let $\beta(P) = \beta(Q)$. Then we have to show that $P \equiv Q$, i.e. $\mathcal{E}_P = \mathcal{E}_Q$. By symmetry it is sufficient to show that $\mathcal{E}_P \subseteq \mathcal{E}_Q$. Let $E \in \mathcal{E}_P$. Then:

$$Q \in \beta^{-1}(\beta(P)) \subseteq \beta^{-1}(\beta(E)) = E$$

Hence $E \in \mathcal{E}_Q$. □

Remark 2.5 If α' is a semantics then: α' fulfills the conditions (1) and (2) of Theorem 2.4 if and only if α' identifies exactly those programs P, Q which cannot be distinguished by any property $E \in Prop$. □

2.2 Consistency w.r.t. a specification formalism

In order to compare two (or more) semantics with regard to their capability of verifying the correctness of the programs we have to look for the properties E which can be checked by them. If both semantics are suitable for checking all properties E of a given specification formalism we consider them as consistent w.r.t. this specification formalism. [23] proposes a notion of consistency relative to a denotational semantics. We generalize this notion and define consistency of two or more semantics α_i relative to another semantics β (not necessary a denotational one) and show that the consistency w.r.t. a specification formalism (a set of properties *Prop*) is equivalent to the consistency relative to a semantics β which is suitable for checking the properties of *Prop* (Theorem 2.10).

Definition 2.6 *Let Prop be a set of properties and $\beta : \mathcal{P} \to B$ a semantics. A family $(\alpha_i)i \in I$ of semantics is called* consistent w.r.t. *Prop iff*

$$\alpha_i \models E$$

*for all $E \in Prop$. $(\alpha_i)_{i \in I}$ is called β-*consistent *iff there exists abstractions f_i from α_i to β.*

Consistency results that assert that a semantics β is an abstraction of another semantics α are special cases of β-consistency results. A list of publications which establish β-consistency results of this form is given in section 2.4. β-consistency results that show the consistency of two semantics relative to a more abstract semantics β are presented e.g. in [3, 9, 10, 18, 19, 39]. In the following two examples we explain in which way the results of [3, 9, 18] can be considered as β-consistency results. In a similar way the results of [10, 19, 39] can be interpreted as β-consistency results.

Example 2.7 [18] and [3] show the consistency of the operational transition system semantics $\mathcal{O}$ of [35] and a denotational prime event structure semantics $\mathcal{D}$ for guarded $TCSP$ in the following sense:

$$\mathcal{O}(P) \;\approx\; \mathit{trans}(\mathcal{D}(P))$$

Here *trans* is a function which assigns a transition system to each prime event structure and $\approx$ denotes weak bisimulation equivalence in the sense of [32]. Let f be the function which maps each transition system to its weak bisimulation equivalence class then the result of [3, 18] says $f \circ \mathcal{O} \;=\; f \circ \mathit{trans} \circ \mathcal{D}$ which is a consistency result in our sense: $\mathcal{O}$ and $\mathcal{D}$ are β-consistent where $\beta \;=\; f \circ \mathcal{O}$. □

Example 2.8 [9] gives a flow event structure semantics $\mathcal{E}$ and a transition system semantics $\mathcal{T}$ for a CCS like language. The transition labels in $\mathcal{T}(P)$ are deterministic non-recursive programs, i.e. programs that are built from atomic actions, sequential and parallel composition. The computations of the event structures are described by transitions

$$E \xrightarrow{p} E'$$

where E, E' are event structures and p a finite pomset (i.e. a flow event structure that describes a deterministic non-recursive process). Such a transition means that E may perform the process described by p such that the behaviour of the remaining process is given by E'. [9] shows the 'adequacy' of $\mathcal{E}$ and $\mathcal{T}$ in the following sense: For each program P:

- If there is a step $P \xrightarrow{w} P'$ w.r.t. the transition system semantics then:

  $$\mathcal{E}(P) \;\xrightarrow{p}\; \mathcal{E}(P')$$

  where $p = \mathcal{E}(w)$.

- If $\mathcal{E}(P) \;\xrightarrow{p}\; E'$ then $P \;\xrightarrow{w}\; P'$ for some deterministic non-recursive program w and some program P' such that $\mathcal{E}(w) = p$ and $\mathcal{E}(P') = E'$.

Let f be the function that assigns to each flow event structure the bisimulation equivalence class of the associated transition system (whose labels are finite pomsets). g denotes the function that assigns to each transition system whose labels a deterministic non-recursive programs the bisimulation equivalence class of the transition system which one gets by substituting the labels w by the pomset $\mathcal{E}(w)$. Then the adequacy result of [9] says: $f \circ \mathcal{E} \;=\; g \circ \mathcal{T}$. Hence $\mathcal{E}$ and $\mathcal{T}$ are β-consistent where $\beta \;=\; f \circ \mathcal{E}$. □

Each family $(\alpha_i)_{i \in I}$ of semantics α_i is β-consistent where $\beta : \mathcal{P} \to B$ is a semantics which maps $\mathcal{P}$ into a one-element domain B. Hence the 'quality' of β-consistency depends on the 'quality' of the semantics β (i.e. the set of properties E with $\beta \models E$). By Lemma 2.3 we get:

Lemma 2.9 *Let $(\alpha_i)_{i \in I}$ be a family of semantics and $\beta : \mathcal{P} \to B$ a semantics. Then $(\alpha_i)_{i \in I}$ is β-consistent if and only if for each property E: $\beta \models E$ implies $\alpha_i \models E$ for all $i \in I$.*

By Lemma 2.9: Let $(\alpha_i)_{i\in I}$ be a family of semantics which is β-consistent. Then $(\alpha_i)_{i\in I}$ is consistent w.r.t. $Prop = \{\beta^{-1}(B_0) : B_0 \subseteq B\}$ where B is the range of β.

Theorem 2.10 *Let Prop be a set of properties and $(\alpha_i)_{i\in I}$ a family of semantics. Then $(\alpha_i)_{i\in I}$ is consistent w.r.t. Prop if and only if there exists a semantics β such that $\beta \models E$ for all $E \in Prop$ and such that $(\alpha_i)_{i\in I}$ is β-consistent.*

Proof: 'if' follows by Lemma 2.9, 'only if' by Theorem 2.4. □

2.3 Most concrete common abstractions and most abstract common refinements

In the situation where for the same language several semantic descriptions are given (e.g. one serving the needs of the implementation, one that is suitable for the verification and another one that explains the language to the user) it might be useful to combine all the different views, i.e. to look for the properties which can be checked by all semantics or to look for the properties which can be checked by at least one the semantics. The following two theorems show that for each family (α_i) of semantics there exist

- a least abstract semantics which is suitable to check **exactly** those properties which can be checked by each of the semantics α_i,
- a most abstract semantics which is suitable to check all those properties which can be checked by some of the semantics α_i.

In other words, there exist a most concrete common abstraction and a most abstract common refinement. Because of Lemma 2.9 the most concrete common abstraction α can be used to show the consistency of the semantics w.r.t. a specification formalism: One has to prove that each property of the specification formalism can be checked by α. The most abstract common refinement can be used to prove additional properties that cannot be expressed by the specification formalism which in general only provides the possibility to describe what a program has to do but not how it has to be done. Hence the most abstract common refinement is suitable to verify properties which assert something about how the program is implemented (e.g. time and space complexity of the implementation).

Theorem 2.11 *Let $(\alpha_i)_{i\in I}$ be a family of semantics. Then there exists a semantics α such that:*

(1) $(\alpha_i)_{i\in I}$ is α-consistent.

(2) Whenever β is a semantics such that (α_i) is β-consistent then there exists a unique abstraction from α to β.

For each semantics α satisfying (1) and (2) and each property E:

$$\alpha \models E \iff \alpha_i \models E \text{ for all } i \in I$$

Proof: Let A_i be the range of α_i. We put

$$A = \left(\biguplus_{i \in I} A_i \right) / \equiv$$

where $\biguplus$ means disjoint union and $\equiv$ the smallest equivalence relation on $\biguplus A_i$ such that:

$$\alpha_i(P) \equiv \alpha_j(P) \ \forall\, i,j \in I,\ P \in \mathcal{P}$$

Let $f_i : A_i \to A$, $f_i(a) = [a]$, where $[a]$ denotes the equivalence class of a w.r.t. $\equiv$. Let $\alpha : \mathcal{P} \to A$ be given by:

$$\alpha(P) = [\ \alpha_i(P)\ ]$$

Then $\alpha = f_i \circ \alpha_i$ for all $i \in I$. Hence (α_i) is α-consistent.

Claim 1: $\alpha \models E \iff \forall\, i \in I \ \ \alpha_i \in E$

Proof: $\Longrightarrow$ by Lemma 2.3. Now we prove $\Longleftarrow$. We assume $\alpha_i \models E$ for all $i \in I$. We have to show that $\alpha^{-1}(\alpha(E)) \subseteq E$. Let $P \in \alpha^{-1}(\alpha(E))$. Then $\alpha(P) = \alpha(Q)$ for some $Q \in E$. If $P = Q$ then $P \in E$. Otherwise there exist $i_0, i_1, \ldots, i_n \in I$ and $Q = P_0, P_1, \ldots, P_n, P_{n+1} = P \in \mathcal{P}$ such that

$$\alpha_{i_k}(P_k) = \alpha_{i_k}(P_{k+1}), \quad k = 0, 1, \ldots, n.$$

Since $Q \in E$ and $\alpha_{i_k} \models E$ it can be shown by induction on k that $P_k \in E$. Hence $P \in E$.

Claim 2: Let $\beta : \mathcal{P} \to B$ be a semantics and $g_i : A_i \to B$ abstractions from α_i to β. Then there exists a unique abstraction $h : A \to B$ from α to β with $h \circ f_i = g_i$.

Proof: In order to define h we need the following:

$$(*) \qquad \text{If } a \in A_i,\ b \in A_j,\ a \equiv b, \text{ then } g_i(a) = g_j(b).$$

Proof of (*): We may assume w.l.o.g. $a \neq b$. Then there exist $i_0, i_1, \ldots, i_n \in I$ and $P = P_0, P_1, \ldots, P_n, P_{n+1} = Q \in \mathcal{P}$ such that:

$$a = \alpha_i(P), \quad b = \alpha_j(Q), \quad \alpha_{i_k}(P_k) = \alpha_{i_k}(P_{k+1}), \ k = 0, 1, \ldots, n.$$

Since $g_{i_k} \circ \alpha_{i_k} = \beta = g_{i_{k+1}} \circ \alpha_{i_{k+1}}$ we get by induction on k: $g_i(a) = g_{i_k}(P_k)$. Hence

$$g_i(a) = g_{i_n}(\alpha_{i_n}(P_n)) = g_j(\alpha_j(Q)) = g_j(b).$$

Definition: Let $h : A \to B$ be given by: $h([a]) = g_i(a)$ if $a \in A_i$. We show that h is the unique abstraction from α to β with $h \circ f_i = g_i$.

By definition of h: $h \circ f_i = g_i$ and

$$h(\alpha(P)) = h([\alpha_i(P)]) = g_i(\alpha_i(P)) = \beta(P)$$

for all $P \in \mathcal{P}$. Let $h' : A \to B$ be also an abstraction from α to β such that $h' \circ f_i = g_i$. Then for all $\xi \in A$: $\xi = [a]$ for some $a \in A_i$, $i \in I$. Hence

$$h'(\xi) = h'(f_i(a)) = g_i(a) = h(\xi).$$

□

Theorem 2.12 *Let $(\alpha_i)_{i \in I}$ be a family of semantics. Then there exists a semantics α such that:*

(1) Each of the semantics α_i, $i \in I$ is an abstraction of α.

(2) Whenever β is a semantics also satisfying (1) then there exists a unique abstraction from β to α.

For each semantics α satisfying (1) and each property E: If $\alpha_i \models E$ for some $i \in I$ then $\alpha \models E$.

Proof: Let A_i be the range of α_i. We put:

$$A = \prod_{i \in I} A_i$$

and $\alpha : \mathcal{P} \to A$, $\alpha(P) = (\alpha_i(P))_{i \in I}$. Then the projections $f_i : A \to A_i$ are abstractions from α to α_i. If $\beta : \mathcal{P} \to B$ is a semantics and $g_i : B \to A_i$ are functions such that $g_i \circ \beta = \alpha_i$ then we show that there exists a unique abstraction $g : B \to A$ such that $f_i \circ g = g_i$.

Let $g : B \to A$ be given by: $g(b) = (\ g_i(b)\ )_{i \in I}$. Then $g \circ \beta = \alpha$ and $f_i \circ g = g_i$. If $g' : B \to A$ is also an abstraction from β to α with $f_i \circ g' = g_i$ then

$$g'(b) = (\ f_i(g'(b))\ )_{i \in I} = (g_i(b))_{i \in I} = g(b)$$

for all $b \in B$.

By Lemma 2.3: If α is a semantics satisfying (1) then: $\alpha_i \models E$ for some $i \in I$ implies $\alpha \models E$. □

One might suppose that the most abstract common refinement α (i.e. a semantics satisfying the conditions (1) and (2) of Theorem 2.12) of a family (α_i) of semantics α_i is suitable for checking a property E if and only if at least one of the semantics α_i is suitable for checking E. In general the implication 'only if' is wrong. For instance, consider two semantics α_1 and α_2 with $\alpha_1(P_1) = \alpha_1(P) \neq \alpha_1(P_2)$ and $\alpha_2(P_2) = \alpha_1(P) \neq \alpha_2(P_1)$ then $\alpha_i \not\models \{P\}$, $i = 1, 2$, whereas the most abstract common refinement α of α_i is sutable for checking $\{P\}$.

2.4 Weak and strong consistency

Theorem 2.10 shows that one way to establish consistency results w.r.t. a specification formalism of semantics α_i, $i \in I$, is to establish abstractions from the semantics α_i to some other semantics β. As a matter of fact, many 'consistency' results consist of defining an abstraction from one semantics to the other. This motivates the following definition:

Definition 2.13 *Let $\alpha : \mathcal{P} \to A$ and $\beta : \mathcal{P} \to B$ be semantics.*

(a) α and β are called weakly consistent *iff α is an abstraction of β or β an abstraction of α.*

(b) α and β are called strongly consistent *iff α is an abstraction of β and β an abstraction of α.*

For instance weak consistency results are established in [4, 6, 7, 14, 23, 24, 25, 30, 33, 36]. The simplest form of strong consisteny results is the equality of semantics. E.g. [6, 7, 14, 25] contain strong consistency results of the form $\mathcal{O} = \mathcal{D}$ where $\mathcal{O}$ is an operational, $\mathcal{D}$ a denotational semantics. A strong consistency result for different semantics is established in [41] which shows the strong consistency of Plotkins transitions system semantics for CSP [37] and a denotational semantics for CSP defined in the style of [8].

Lemma 2.3 contains a characterisation of weakly consistent semantics. The following lemma follows immediately by Lemma 2.3. It shows that strongly consistent semantics α, β are fully equivalent w.r.t. the capability of testing properties (independent of any specification formalism).

Lemma 2.14 *Let $\alpha : \mathcal{P} \to A$ and $\beta : \mathcal{P} \to B$ be semantics. The following are equivalent:*

(a) α and β are strongly consistent.

(b) For all $P, Q \in \mathcal{P}$: $\alpha(P) = \alpha(Q)$ if and only if $\beta(P) = \beta(Q)$.

(c) For all properties E: $\alpha \models E$ if and only if $\beta \models E$.

Remark 2.15 The semantics α in Theorem 2.11 and Theorem 2.12 are unique up to strong consistency. □

Lemma 2.16 *Let $\beta : \mathcal{P} \to B$ be a semantics and $f : A \to B$ a surjection. Then there exists a semantics $\alpha : \mathcal{P} \to A$ such that α and β are strongly consistent.*

Proof: Since f is surjective there exists $g : B \to A$ such that $f \circ g = id_B$. We put $\alpha = g \circ \beta$. Then g is an abstraction from β to α. Since $f \circ \alpha = f \circ g \circ \beta = \beta$, f is an abstraction from α to β. □

If $\alpha_1 : \mathcal{P} \to A_1$ and $\alpha_2 : \mathcal{P} \to A_2$ are weakly consistent, e.g. $f \circ \alpha_1 = \alpha_2$, then α_1, α_2 are consistent w.r.t. $Prop = \{\alpha_2^{-1}(A_0) : A_0 \subseteq A\}$ and β-consistent w.r.t. every abstraction β of α_2. The following diagram shows the hierarchy of consistency results where the arrows denote implication:

$$\begin{tikzcd}
& \alpha\text{-consistency} & \\
\beta\text{-consistency} \arrow[ur] \arrow[rr, leftrightarrow] & & \text{consistency w.r.t. } Prop \arrow[ul] \\
& \text{weak consistency} \arrow[ul] \arrow[ur] & \\
& \text{strong consistency} \arrow[u] &
\end{tikzcd}$$

where β is a most abstract semantics suitable for checking all properties in *Prop* (Theorem 2.4) and α an abstraction of β. β-consistency is a weaker notion than weak consistency. E.g. one can provide a pomset semantics α_1 and a synchronization tree semantics α_2 for a language which includes nondeterministic choice and parallel composition that are β-consistent where β maps a process to a set of traces. α_1 and α_2 cannot be weakly consistent as α_1 is a linear time true concurrency model and α_2 a branching time interleaving model.

2.5 Application: Verification and stepwise refinement

Weak consistency results can help to prove the correctness of implementations which are generated by stepwise refinement. We show that under the assumption that a given program P which is written in a high-level language $\mathcal{P}$ meets the specification it can be concluded that its refinement $i(P)$ (which is written in a lower-level language $\mathcal{P}'$) also meets the specification. The programs $P \in \mathcal{P}$ are considered as algorithms. We assume a mapping $i : \mathcal{P} \to \mathcal{P}'$ which assigns to each $P \in \mathcal{P}$ a program $i(P) \in \mathcal{P}'$ which we call the implementation of P.

We assume semantics $\alpha : \mathcal{P} \to A$ and $\alpha' : \mathcal{P}' \to A'$ and a 'consistency result' of the form $\alpha = f \circ \alpha' \circ i$ where $f : A' \to A$ is a function. The reason why we assume that f maps A' to A and not vice versa is that in general $i(P)$ (and then also $\alpha'(i(P))$) contains more details. E.g. it might be the case that in $\mathcal{P}$ there is an atomic statement a which stands for the sorting of a finite sequence of natural numbers. The function i might map this atomic statement to a program which implements Quicksort (or another sorting algorithm).

$$\begin{tikzcd}
\mathcal{P} \arrow[r, "i"] \arrow[d, "\alpha"'] & \mathcal{P}' \arrow[d, "\alpha'"] \\
A \arrow[r, "f"'] & A'
\end{tikzcd}$$

The assumption $\alpha = f \circ \alpha' \circ i$ can be considered as a weak consistency result for the semantics α and $\alpha' \circ i$ for the language $\mathcal{P}$.

Lemma 2.17 *For each property $E \subseteq \mathcal{P}$ with $\alpha \models E$: If the algorithm P has property E then its implementation $i(P)$ has property $i(E)$.*

If $E = \alpha^{-1}(A_0)$ where $A_0 \subseteq A$ and $E' = \alpha'^{-1}(A'_0)$ where $A_0 \subseteq A'$ and $f^{-1}(A_0) \subseteq A'_0$ then:

If the algorithms P has property E then its implementation $i(P)$ has property E'.

Proof: Since f is an abstraction from $\alpha' \circ i$ to α we get by Lemma 2.3: $\alpha' \circ i \models E$. Since α' is an abstraction from i to $\alpha' \circ i$ we get (again by Lemma 2.3): $i \models E$. Hence for each algorithm $P \in \mathcal{P}$: $i(P) \in i(E)$ if and only if $P \in E$.

Let $E = \alpha^{-1}(A_0)$, $E' = \alpha'^{-1}(A'_0)$ and $f^{-1}(A_0) \subseteq A'_0$. Then $\alpha \models E$ and $\alpha' \models E'$. Since $\alpha = f \circ \alpha' \circ i$ we have:

$$E = \alpha^{-1}(A_0) = i^{-1}(\ \alpha'^{-1}(f^{-1}(A_0))\ ) \subseteq i^{-1}(\ \alpha'^{-1}(A'_0)\ ) = i^{-1}(E')$$

Therefore $i(E) \subseteq E'$. Hence: If $P \in E$ then $i(P) \in i(E) \subseteq E'$. $\square$

This result can be used to verify the correctness of the implementation w.r.t. a specification written in some logic. In addition to the assumptions of above we assume that the specifications are given as a set of logical formulas of some logic L. We assume that there are functions $F : A \to I$, $F' : A' \to I'$ where I, I' are sets of interpretations for L such that for all $a' \in A'$ and all formulas φ of L:

If $F(f(a'))$ is a model of φ then $F'(a')$ is a model of φ.

We say $P \in \mathcal{P}$ satisfies φ iff $F(\alpha(P))$ is a model of φ and similary, $P' \in \mathcal{P}'$ satisfies φ iff $F'(\alpha'(P'))$ is a model of φ.

Lemma 2.18 *With the notions and assumptions of above: If the algorithm P satisfies φ then the implementation $i(P)$ satisfies φ.*

Proof: Let A_0 resp. A'_0 be the set of elements $a \in A$ resp. $a' \in A'$ such that $F(a)$ resp. $F'(a')$ is a model of φ. Then the assumption above says that $A'_0 \supseteq f^{-1}(A_0)$. If P satisfies φ (i.e. $F(\alpha(P))$ is a model of φ) then $P \in \alpha^{-1}(A_0)$. Hence by Lemma 2.17 $i(P) \in \alpha'^{-1}(A'_0)$, i.e. $F'(\alpha'(i(P)))$ is a model of φ which means that $i(P)$ satisfies φ. □

Hence under the assumptions of above the verification problem is reduced to prove that

- the algorithm meets the specification
- if $F(f(a'))$ is a model of φ then $F'(a')$ is a model of φ.

The simplest case in which the second condition is satisfied is $I' = I$, $F \circ f = f'$.

3 Compositional semantics

In compositional semantics the meaning of a composite program $P = \omega(P_1, \ldots, P_n)$ can be computed by composing the meanings of the modules $P_1, \ldots, P_n$ via a semantic operator ω_A on the underlying semantic domain A. Compositionality yields tools to compute the semantics and for modular verification. Modular verification means that in order to prove the correctness of a program P each of the program modules is verified separately and the correctness of P is proved using the specifications of the modules but without using any knowledge about how the modules are implemented. This implies that also the specifications should be modular: for each property E under consideration and each n-ary operator ω of the underlying language there exist a set $\mathcal{E}_{E,\omega}$ consisting of n-tuples $(E_1, \ldots, E_n)$ of properties such that the composite program $P = \omega(P_1, \ldots, P_n)$ has property E if and only if there is some of these tuples where P_i has property E_i, $i = 1, \ldots, n$. In this case, if α is a compositional semantics suitable for checking all properties in the underlying specification formalism then

$$\alpha(P) \in \alpha(E) \iff \exists\ (E_1, \ldots, E_n) \in \mathcal{E}_{E,\omega}\ :\ \alpha(P_i) \in \alpha(E_i),\ i = 1, \ldots, n$$

I.e. α is suitable for modular verification.

In this and the following sections we deal with a language $\mathcal{P}$ with abstract operator symbols and recursion (modelled by declarations). The syntax is as in [4]:

Notation 3.1 *A* symbol-algebra *(or* signature*) is a pair $\Sigma = (Op, |\cdot|)$ consisting of a set Op of operator symbols and a function $|\cdot| : Op \to \mathbb{N}_0$ which assigns the arity $|\omega|$ to each operator symbol ω. Operator symbols of arity 0 are called* constant symbols. *The set $\mathcal{L}(\Sigma, Idf)$ of* statements *over Σ and Idf (where Idf is a fixed set of variables) is given by the production system*

$$s \ ::= \ a \ | \ x \ | \ \omega(s_1, \ldots, s_n)$$

where a is a constant symbol, $x \in Idf$, ω an operator symbol with $|\omega| = n \geq 1$, $s_1, \ldots, s_n \in \mathcal{L}(\Sigma, Idf)$. A declaration *is a mapping $\sigma : Idf \to \mathcal{L}(\Sigma, Idf)$. A* program *over (Σ, Idf) is a pair $< \sigma, s >$ consisting of a declaration σ and a statement s.*

If $P =< \sigma, s >$ is a program then the behaviour of P is given by s where each occurrence of a variable x in s is interpreted as a recursive call of the procedure $\sigma(x)$. In the following we fix a declaration σ and we define $\mathcal{P}$ to be the set of programs $< \sigma, s >$ where $s \in \mathcal{L}(\Sigma, Idf)$. The operator symbols are considered as operators on $\mathcal{P}$ where we put

$$\omega(< \sigma, s_1 >, \ \ldots, \ < \sigma, s_n > \ ) \ = < \ \sigma, \ \omega(s_1, \ldots, s_n) \ > .$$

We write x to denote the recursive program $< \sigma, x >$, $\sigma(x)$ instead of $< \sigma, \sigma(x) >$ and a instead of $< \sigma, a >$ for each constant symbol a.

Example 3.2 *CCS* [31, 32] without recursion is associated with the symbol-algebra Σ_{CCS} which consists of the following operator symbols: the constant symbol *nil*, for each action $a \in Act$ an operator symbol γ_a of the arity 1, $\gamma_a(s) = a.s$, modelling prefixing, the binary operator symbols $+$ and $|$, modelling nondeterminism resp. parallelism with possible communication on complementary actions, for each $L \subseteq Act \setminus \{\tau\}$ an operator symbol ρ_L, $\rho_L(s) = s \setminus L$, of the arity 1 for modelling restriction to actions $\notin L \cup \overline{L}$ and for each relabelling function $\lambda : Act \to Act$ an operator symbol ℓ_λ of the arity 1, $\ell_\lambda(s) = s\{\lambda\}$, which is used for renaming the actions. Here *Act* is a nonempty set of actions which contains an internal action denoted by τ. We assume a function $\overline{(\cdot)} : Act \to Act$, $a \mapsto \overline{a}$, such that $\overline{\tau} = \tau$ and $\overline{\overline{a}} = a$ for each action $a \in Act$. If $L \subseteq Act$ then $\overline{L} = \{\overline{a} : a \in L\}$. A relabelling function is a function $\lambda : Act \to Act$ with $\lambda(\tau) = \tau$ and $\lambda(\overline{a}) = \overline{\lambda(a)}$. □

Example 3.3 The language $\mathcal{L}_0 = \mathcal{L}_0(\Sigma_0, Idf)$ which is considered e.g. in [7] is given by the symbol-algebra Σ_0 which contains a nonempty set *Act* of actions as the constant symbols and binary operator symbols $+$, $||$ and ; for modelling nondeterminism, parallelism (with or without synchronisation or communication) resp. sequential execution. □

As usual interpretations of symbol-algebras are given by Σ-algebras. If $\Sigma = (Op, |\cdot|)$ is a symbol-algebra then a Σ-algebra is a pair (A, Op_A) consisting of a set A and a set Op_A of operators $Op_A = \{\omega_A : \omega \in Op\}$ such that $\omega_A : A^n \to A$ is a function where $|\omega| = n$. (In the case $n = 0$ $\omega_A \in A$.) In the following we omit the operator set Op_A of a Σ-algebra and we shortly write A instead of (A, Op_A). If A and B are Σ-algebras then a function $f : A \to B$ is called a homomorphism iff

$$f \, (\, \omega_A(\xi_1, \ldots, \xi_n) \,) \ = \ \omega_B \, (\, f(\xi_1), \ldots, f(\xi_n) \,)$$

for each $\omega \in Op$, $|\omega| = n$.

Definition 3.4 *A* compositional semantics *for $\mathcal{P}$ is a function $\alpha : \mathcal{P} \to A$ where A is Σ-algebra and which satisfies the following conditions:*

(I) $\alpha(\omega(P_1, \dots, P_n)) \;=\; \omega_A(\alpha(P_1), \dots, \alpha(P_n))$
for all $\omega \in Op$, $|\omega| = n \geq 0$ and $P_1, \dots, P_n \in \mathcal{P}$.

(II) α satisfies the recursion condition: $\alpha(x) \;=\; \alpha(\;\sigma(x)\;)$ for all $x \in Idf$.

In [4] we gave examples which show that for given A there might be either no compositional semantics or more than one. A general characterisation of possible meaning functions is given in the following lemma:

Lemma 3.5 *(see* [4]*) Let A be a Σ-algebra and $\alpha : \mathcal{P} \to A$ a semantics. Then: α is a compositional if and only if α is a fixed point of the operator*

$$\Phi^A \;:\; (\mathcal{P} \to A) \;\to\; (\mathcal{P} \to A)$$

where Φ^A is defined by structural induction:

(1) $\Phi^A(f)(a) \;=\; a_A$ *for each constant symbol a*

(2) $\Phi^A(f)(x) \;=\; f(\;\sigma(x)\;)$ *for each variable $x \in Idf$*

(3) $\Phi^A(f)(\;\omega(P_1, \dots, P_n)\;) \;=\; \omega_A(\;\Phi^A(f)(P_1), \dots, \Phi^A(f)(P_n)\;)$
for all $P_1, \dots, P_n \in \mathcal{P}$, $\omega \in Op$, $|\omega| = n \geq 1$.

Remark 3.6 Let $f : A \to B$ be a homomorphism between two Σ-algebras A and B and let $\overline{f} \;:\; (\mathcal{P} \to A) \;\to\; (\mathcal{P} \to B)$ be defined by $\overline{f}(g) \;=\; f \circ g$. Then f is an abstraction from α to β if and only if $\overline{f}(\alpha) \;=\; \beta$. It is easy to see that $\overline{f} \circ \Phi^A \;=\; \Phi^B \circ \overline{f}$ where Φ^A and Φ^B are defined as in Lemma 3.5. □

3.1 The existence of a semantics suitable for modular verification

We extend the result of Theorem 2.4 and present a condition for the existence of a compositional semantics which is suitable for modular verification. The condition which is needed is the modularity of the properties:

Definition 3.7 *Let Prop be a set of properties. Prop is called* modular *iff the following conditions are satisfied:*

1. *For each property $E \in Prop$ and each n-ary operator symbol ω $(n \geq 1)$ there exists a set $\mathcal{E}_{E,\omega}$ consisting of n-tuples $(E_1, \dots, E_n)$ of properties $E_i \in Prop$ such that for all $P_1, \dots, P_n \in \mathcal{P}$:*

$$\omega(P_1, \dots, P_n) \;\in\; E \quad\Longleftrightarrow\quad \exists\; (E_1, \dots, E_n) \in \mathcal{E}_{E,\omega} \;:\; P_i \in E_i,\; i = 1, \dots, n$$

2. For all variables $x \in \mathcal{P}$ and all properties $E \in$ Prop:

$$x \in E \iff \sigma(x) \in E$$

If a specification formalism is modular then the first condition asserts the soundness and completeness of modular verification: the implication $\Leftarrow$ reflects the soundness of modular verification (if the modules fulfill certain properties then the composite program satisfies its specification), the implication $\Rightarrow$ stands for the completeness of modular verification (if the composite program is correct then the correctness can be shown by proving that its modules have certain properties). The second condition is needed to ensure that the recursive program x which is assumed to consist of a recursive call of $\sigma(x)$ cannot be distinguished from $\sigma(x)$ by any property.

Lemma 3.8 *Let Prop be a set of properties satisfying condition 2 of Definition 3.7 and let $\equiv$ be the induced equivalence relation, i.e.*

$$P \equiv Q \iff \forall\, E \in \mathit{Prop}:\ P \in E \text{ iff } Q \in E.$$

Then:

(a) If Prop is modular then $\equiv$ is a congruence relation on $\mathcal{P}$.

(b) If $\equiv$ is a congruence relation on Prop and Prop $= \mathcal{P}/\equiv$ then Prop is modular.

Proof: Let *Prop* be modular. We show that $\equiv$ is a congruence relation on $\mathcal{P}$. If ω is an n-ary operator symbol, $n \geq 1$, then we have for all $P_1, \ldots, P_n, Q_1, \ldots, Q_n \in \mathcal{P}$ with $P_i \equiv Q_i$:

$$\begin{aligned}
\omega(P_1,\ldots,P_n) \in E &\iff \exists\,(E_1,\ldots,E_n) \in \mathcal{E}_{E,\omega} : P_i \in E_i,\ i = 1,\ldots,n \\
&\iff \exists\,(E_1,\ldots,E_n) \in \mathcal{E}_{E,\omega} : Q_i \in E_i,\ i = 1,\ldots,n \\
&\iff \omega(Q_1,\ldots,Q_n) \in E
\end{aligned}$$

Here $E \in$ *Prop* and $\mathcal{E}_{E,\omega}$ are as in Definition 3.7. Hence $P_i \equiv Q_i$, $i = 1, \ldots, n$ implies $\omega(P_1, \ldots, P_n) \equiv \omega(Q_1, \ldots, Q_n)$.

Now we suppose that $\equiv$ is a congruence relation on $\mathcal{P}$ and *Prop* $= \mathcal{P}/\equiv$. We show that *Prop* satisfies the first condition of Definition 3.7. Let $E \in$ *Prop* and ω an n-ary operator symbol in Σ. We define

$$\mathcal{E}_{E,\omega} = \{\ ([P_1],\ldots,[P_n]) : \omega(P_1,\ldots,P_n) \in E\ \}$$

where $[P]$ denotes the $\equiv$-equivalence class of P. Since *Prop* $= \mathcal{P}/\equiv$ the equivalence class $[P]$ is a property in *Prop*. Then:

1. If $\omega(P_1, \ldots, P_n) \in E$ then $([P_1], \ldots, [P_n]) \in \mathcal{E}_{E,\omega}$. Hence $P_i \in E_i$, $i = 1, \ldots, n$ for some $(E_1, \ldots, E_n) \in \mathcal{E}_{E,\omega}$.

2. Let $(E_1, \dots, E_n) \in \mathcal{E}_{E,\omega}$ and $P_i \in E_i$, $i = 1, \dots, n$, and let $(Q_1, \dots, Q_n)$ be a tuple of programs with $E_i = [Q_i]$ and $\omega(Q_1, \dots, Q_n) \in E$. Then $P_i \equiv Q_i$, $i = 1, \dots, n$. Since $\equiv$ is a congruence relation on $\mathcal{P}$ we have:

$$\omega(P_1, \dots, P_n) \equiv \omega(Q_1, \dots, Q_n)$$

Since E is the equivalence class of $\omega(Q_1, \dots, Q_n)$ we get $\omega(P_1, \dots, P_n) \in E$. □

By Lemma 3.8(b), every congruence relation on a process algebra induces a modular specification formalism. For instance, bisimulation equivalence $\sim$ is a congruence on CCS [32] and hence the specification formalism *Prop* $= CCS/\sim$ a modular specification formalism.

The next lemma shows that for every modular specification formalism *Prop* there exists a most abstract compositional semantics which is suitable for checking all properties of *Prop*, i.e. which is suitable for modular verification.

Theorem 3.9 *Let Prop be a modular set of properties. Then there exists a surjective compositional semantics α which satisfies:*

(1) $\alpha \models E$ for all $E \in Prop$

(2) Whenever β is a semantics satisfying (1) then α is an abstraction of β.

Proof: Let α be defined as in the proof of Theorem 2.4. Then α is surjective and satisfies (1) and (2). We show the compositionality of α: For every n-ary operator symbol ω in Σ we may define:

$$\omega_A([P_1], \dots, [P_n]) = [\ \omega(P_1, \dots, P_n)\ ]$$

(ω_A is welldefined because of Lemma 3.8). Hence A is a Σ-algebra. By the second condition in Definition 3.7 we get:

$$\alpha(x) = [x] = [\sigma(x)] = \alpha(\sigma(x))$$

Hence α is compositional. □

3.2 Most concrete common abstractions and most abstract common refinements

The following theorems show that the results of Theorem 2.11 and 2.12 carry over to surjective compositional semantics:

Theorem 3.10 *Let $(\alpha_i)_{i \in I}$ be a family of surjective compositional semantics. Then there exists a surjective compositional semantics α such that:*

(1) $(\alpha_i)_{i \in I}$ is α-consistent.

(2) Whenever β is a semantics satisfying (1) then β is an abstraction of α.

Proof: Let A_i be the range of α_i and let $\alpha : \mathcal{P} \to A$ be as in the proof of Theorem 2.11. Then α is surjective (since each of the semantics α_i is surjective) and satisfies (1) and (2). We show the compositionality of α:

Claim: Let ω be an n-ary operator symbol, $n \geq 1$, and $i,\ j \in I$, $a_1, \ldots, a_n \in A_i$, $b_1, \ldots, b_n \in A_j$ such that $a_k \equiv b_k$, $k = 1, \ldots, n$. Then:

$$\omega_{A_i}(a_1, \ldots, a_n) \equiv \omega_{A_j}(b_1, \ldots, b_n)$$

Proof: Let $i \in I$ be a fixed index and $a_k = \alpha_i(P_k)$, $k = 1, \ldots, n$. (Note that α_i is surjective. Hence each element $a \in A$ is of the form $a = \alpha_i(P)$ for some $P \in \mathcal{P}$.)

We show by induction on k $(0 \leq k \leq n)$ that for all $j \in I$:

$$\omega_{A_i}(a_1, \ldots, a_n) \equiv \omega_{A_j}(b_1, \ldots, b_k, \alpha_j(P_{k+1}), \ldots, \alpha_j(P_n))$$

for all $b_1, \ldots, b_k \in A_j$ with $a_l \equiv b_l$, $l = 1, \ldots, k$.

Basis of induction ($k = 0$):

$$\begin{aligned} \omega_{A_i}(a_1, \ldots, a_n) &= \omega_{A_i}(\ \alpha_i(P_1), \ldots, \alpha_i(P_n)\ ) \\ &= \alpha_i(\ \omega(P_1, \ldots, P_n)\ ) \\ &\equiv \alpha_j(\ \omega(P_1, \ldots, P_n)\ ) \\ &= \omega_{A_j}(\ \alpha_j(P_1), \ldots, \alpha_j(P_n)\ ) \end{aligned}$$

Induction step $k-1 \implies k$ $(1 \leq k \leq n)$:

Let $j \in I$, $b_1, \ldots, b_k \in A_j$, $a_l \equiv b_l = \alpha_j(Q_l)$, $l = 1, \ldots, k$. Then there exist

$$i = i_1,\ i_2,\ \ldots,\ i_m = j \ \in\ I$$

and $P_k = R_1,\ R_2,\ \ldots,\ R_m,\ R_{m+1} = Q_k \in \mathcal{P}$ such that $\alpha_{i_r}(R_r) = \alpha_{i_r}(R_{r+1})$, $r = 1, \ldots, m$. By induction hypothesis:

$$\omega_{A_i}(a_1, \ldots, a_n) \equiv \omega_{A_{i_1}}(c_1, \ldots, c_{k-1}, \alpha_{i_1}(P_k), \ldots, \alpha_{i_1}(P_n))$$

where $c_l = \alpha_{i_1}(Q_l)$, $l = 1, \ldots, k-1$. Note that $\alpha_{i_1}(Q_l) \equiv \alpha_j(Q_l) = b_l \equiv a_l$. Then:

$$\begin{aligned} & \omega_{A_i}(a_1, \ldots, a_n) \\ \equiv\ & \omega_{A_{i_1}}(\alpha_{i_1}(Q_1), \ldots, \alpha_{i_1}(Q_{k-1}), \alpha_{i_1}(R_1), \alpha_{i_1}(P_{k+1}), \ldots, \alpha_{i_1}(P_n)) \\ =\ & \omega_{A_{i_1}}(\alpha_{i_1}(Q_1), \ldots, \alpha_{i_1}(Q_{k-1}), \alpha_{i_1}(R_2), \alpha_{i_1}(P_{k+1}), \ldots, \alpha_{i_1}(P_n)) \\ =\ & \alpha_{i_1}(\ \omega(Q_1, \ldots, Q_{k-1}, R_2, P_{k+1}, \ldots, P_n)\ ) \\ \equiv\ & \alpha_{i_2}(\ \omega(Q_1, \ldots, Q_{k-1}, R_2, P_{k+1}, \ldots, P_n)\ ) \\ =\ & \omega_{A_{i_2}}(\alpha_{i_2}(Q_1), \ldots, \alpha_{i_2}(Q_{k-1}), \alpha_{i_2}(R_2), \alpha_{i_2}(P_{k+1}), \ldots, \alpha_{i_2}(P_n)) \\ =\ & \omega_{A_{i_2}}(\alpha_{i_2}(Q_1), \ldots, \alpha_{i_2}(Q_{k-1}), \alpha_{i_2}(R_3), \alpha_{i_2}(P_{k+1}), \ldots, \alpha_{i_2}(P_n)) \end{aligned}$$

By induction we get:

$$\begin{aligned} & \omega_{A_i}(a_1, \ldots, a_n) \\ \equiv\ & \omega_{A_{i_m}}(\alpha_{i_m}(Q_1), \ldots, \alpha_{i_m}(Q_{k-1}), \alpha_{i_m}(R_{m+1}), \alpha_{i_m}(P_{k+1}), \ldots, \alpha_{i_m}(P_n)) \\ =\ & \omega_{A_j}(\alpha_j(Q_1), \ldots, \alpha_j(Q_{k-1}), \alpha_j(Q_k), \alpha_j(P_{k+1}), \ldots, \alpha_j(P_n)) \\ =\ & \omega_{A_j}(b_1, \ldots, b_{k-1}, b_k, \alpha_j(P_{k+1}), \ldots, \alpha_j(P_n)) \end{aligned}$$

For each constant symbol a we put $a_A = [a]$. For each n-ary operator symbol ω, $n \geq 1$, we put:

$$\omega_A(\xi_1, \ldots, \xi_n) = [\, \omega_{A_i}(a_1, \ldots, a_n) \,]$$

where $i \in I$ and $a_1, \ldots, a_n \in A_i$, $[a_l] = \xi_l$, $l = 1, \ldots, n$. Note that this operator ω_A is welldefined because of the claim and the assumption that α_i is surjective. Hence A is a Σ-algebra with $\alpha(x) = [\alpha_i(x)] = [\alpha_i(\sigma(x))] = \alpha(\sigma(x))$ and

$$\begin{aligned}\alpha(\, \omega(P_1, \ldots, P_n) \,) &= [\, \alpha_i(\omega(P_1, \ldots, P_n)) \,] \\ &= [\, \omega_{A_i}(\alpha_i(P_1), \ldots, \alpha_i(P_n)) \,] \\ &= \omega_A(\alpha(P_1), \ldots, \alpha(P_n)).\end{aligned}$$

Hence α is compositional. □

Theorem 3.11 *Let $(\alpha_i)_{i \in I}$ be a family of compositional semantics. Then there exists a compositional semantics α such that:*

(1) Each of the semantics α_i is an abstraction of α.

(2) Whenever β is a semantics also satisfying (1) then α is an abstraction of β.

Proof: Let $\alpha : \mathcal{P} \to A$ be defined as in the proof of Theorem 2.12. Then α satisfies (1) and (2). We show the compositionality of α: If a is a constant symbol then we put $a_A = (\, a_{A_i} \,)_{i \in I}$. If ω is an n-ary operator symbol ($n \geq 1$) then we put:

$$\omega_A(\xi_1, \ldots, \xi_n) = (\, \omega_{A_i}(f_i(\xi_1), \ldots, f_i(\xi_n)) \,)_{i \in I}$$

Then A is a Σ-algebra and it is easy to see that α is compositional. □

3.3 Weak consistency of compositional semantics

The following lemma shows that in order to establish an abstraction between two compositional semantics one has to look for a homomorphism [28].

Lemma 3.12 *Let $\alpha : \mathcal{P} \to A$, $\beta : \mathcal{P} \to B$ be compositional semantics and $f : A \to B$ an abstraction from α to β. If α is surjective then f is a homomorphism.*

Proof: For each constant symbol a: $f(a_A) = f(\alpha(a)) = \beta(a) = a_B$. For each n-ary operator symbol ω: If $\xi = \omega_A(\xi_1, \ldots, \xi_n)$ then $\xi_i = \alpha(P_i)$ for some program P_i. Then

$$\beta(P_i) = f(\alpha(P_i)) = f(\xi_i).$$

Since α is compositional:

$$\xi = \omega_A(\alpha(P_1), \ldots, \alpha(P_n)) = \alpha(\, \omega(P_1, \ldots, P_n) \,)$$

Since β is compositional and $f \circ \alpha = \beta$:

$$\begin{aligned}f(\xi) &= f(\alpha(\omega(P_1, \ldots, P_n)) = \beta(\omega(P_1, \ldots, P_n)) \\ &= \omega_B(\beta(P_1), \ldots, \beta(P_n)) = \omega_B(f(\xi_1), \ldots, f(\xi_n))\end{aligned}$$

I.e. f is a homomorphism. □

Lemma 3.13 *Let A and B two Σ-algebras and $f : A \rightarrow B$ a homomorphism from A to B. If $\alpha : \mathcal{P} \rightarrow A$ is compositional then $f \circ \alpha$ is a compositional semantics on B.*

Proof: by structural induction. □

In the next section we make use of Lemma 3.12 and Lemma 3.13 and present criterions such that for given compositional semantics $\alpha : \mathcal{P} \rightarrow A$ and $\beta : \mathcal{P} \rightarrow B$ and a given homomorphism $f : A \rightarrow B$ the semantics $f \circ \alpha$ agrees with β thus yielding a weak consistency result for α and β.

Lemma 3.14 *Let $\beta : \mathcal{P} \rightarrow B$ be a compositional semantics and $f : A \rightarrow B$ a surjection where A is an arbitrary set. Then there exist semantic operators on A which turn A into a Σ-algebra and f into a homomorphism and a compositional semantics $\alpha : \mathcal{P} \rightarrow A$ on A such that α and β are strongly consistent.*

Proof: As stated in the proof of Lemma 2.16 there exist a function $g : B \rightarrow A$ with $f \circ g = id_B$ and the semantics $\alpha = g \circ \beta$ and $\beta = f \circ \alpha$ are strongly consistent.

For each n-ary operator symbol ω we put:

$$\omega_A : A^n \rightarrow A, \quad \omega_A(\xi_1, \ldots, \xi_n) \; = \; g(\omega_B(f(\xi_1,), \ldots, f(\xi_n)) \;)$$

Then A is a Σ-algebra, f a homorphism and

$$\alpha(\sigma(x)) \; = \; g(\beta(\sigma(x))) \; = \; g(\beta(x)) \; = \; \alpha(x)$$

and for each n-ary operator symbol ω:

$$\begin{aligned} \alpha(\omega(P_1, \ldots, P_n)) &= g(\beta(\omega(P_1, \ldots, P_n))) \\ &= g(\omega_B(\beta(P_1), \ldots, \beta(P_n))) \\ &= g(\omega_B(f(\alpha(P_1)), \ldots, f(\alpha(P_n)))) \\ &= \omega_A(\alpha(P_1), \ldots, \alpha(P_n)). \end{aligned}$$

Hence α is compositional. □

Lemma 3.15 *Let $\alpha : \mathcal{P} \rightarrow A$ be a compositional semantics and $f : A \rightarrow B$ be a surjection such that for each n-ary operator symbol ω:*

$$\textit{If } f(\xi_i) = f(\xi_i'),\ i = 1, \ldots, n, \textit{ then } \omega_A(\xi_1, \ldots, \xi_n) \; = \; \omega_A(\xi_1', \ldots, \xi_n').$$

Then there exist semantic operators on B which turn B into a Σ-algebra and f into a homomorphism. Hence by Lemma 3.13 $f \circ \alpha$ is a compositional semantics on B (which is an abstraction of α).

Proof: If ω is an n-ary operator symbol then we put:

$$\omega_B(\eta_1, \ldots, \eta_n) \; = \; f(\omega_A(\xi_1, \ldots, \xi_n))$$

where $\xi_i \in f^{-1}(\eta_i)$, $i = 1, \ldots, n$. Note that by the assumptions about f the operators ω_B are welldefined. It is easy to see that f is a homomorphism. □

4 Adequacy and full abstractness

Often when relating an operational with a denotational semantics of a language $\mathcal{P}$ as in section 3 the notions of adequacy and full abstraction are employed. We extend these notions to arbitrary semantics α and β.

Definition 4.1 *A semantics α is called* adequate *w.r.t. a semantics β iff for all programs P, Q:*

$$\alpha(P) = \alpha(Q) \text{ implies } \forall\, C[\cdot] \;:\; \beta(\, C[P]\,) \;=\; \beta(\, C[Q]\,).$$

α is called fully abstract *w.r.t. β iff*

$$\forall\, C[\cdot] \;:\; \beta(\, C[P]\,) \;=\; \beta(\, C[Q]\,) \text{ implies } \alpha(P) = \alpha(Q).$$

Here $C[\cdot]$ ranges over the set of contexts for $\mathcal{P}$, i.e. the set of 'programs' in $\mathcal{P}$ containing one or more holes. $C[P]$ is the program which is obtained by substituting P for the holes in $C[\cdot]$.

Theorem 4.2 *Let α be a compositional semantics. Then: α is adequate w.r.t. β if and only if β is an abstraction of α.*

Proof: Let A be the range of α. Every context $C[\cdot]$ induces a mapping $C_A : A \to A$ with $\alpha(\, C[P]\,) \;=\; C_A(\alpha(P))$ for all $P \in \mathcal{P}$. If $f \circ \alpha \;=\; \beta$ and $\alpha(P) = \alpha(Q)$ then for all contexts $C[\cdot]$:

$$\beta(C[P]) \;=\; f(\alpha(C[P])) \;=\; f(C_A(\alpha(P))) \;=\; f(C_A(\alpha(Q))) \;=\; f(\alpha(C[Q])) \;=\; \beta(C[Q]).$$

If α is adequate w.r.t. β then for all P and Q: If $\alpha(P) = \alpha(Q)$ then we consider the context $C[\cdot]$ consisting of a whole (i.e. $C[S] = S$ for all $S \in \mathcal{P}$) and get $\beta(P) = \beta(Q)$. Hence β is an abstraction of α. $\square$

The next theorem shows the existence of a compositional semantics which is adequate and fully abstract w.r.t. a given semantics β under the assumption that β does not distinguish between a procedure name x and its body $\sigma(x)$ in every context.

Theorem 4.3 *For each semantics β there exists a unique (up to strong consistency) semantics α which is adequate and fully abstract w.r.t. β.*

If $\beta(C[x]) = \beta(C[\sigma(x)])$ for all variables x and all contexts $C[\cdot]$ then there exists a compositional semantics α which is adequate and fully abstract w.r.t. β.

Proof: First we show the uniqueness up to strong consistency: If α and α' are semantics which are adequate and fully abstract w.r.t. β then α and α' identify exactly the same programs. Hence α and α' are strongly consistent.

α can be constructed with help of Theorem 2.4: Let $\alpha : \mathcal{P} \to \mathcal{P}/\equiv_\beta$ be the canonical function where

$$P \equiv_\beta Q \quad\Longleftrightarrow\quad \forall\, C[\cdot] \;:\; \beta(C[P]) \;=\; \beta(C[Q]).$$

(Then α is the most abstract semantics which is suitable for the specification formalism $Prop_\beta \;=\; \mathcal{P}/\equiv_\beta$.) α identifies two programs P and Q if and only if $P \equiv_\beta Q$, i.e. α is adequate and fully abstract w.r.t. β. Semantic operators on the range $A = \mathcal{P}/\equiv_\beta$ of α can be defined by:

$$\omega_A(\, [P_1]_{\equiv_\beta}, \ldots, [P_n]_{\equiv_\beta}\,) = [\, \omega(P_1, \ldots, P_n)\,]_{\equiv_\beta}$$

where ω is an n-ary operator symbol. It is easy to see that α is compositional if $\beta(C[x]) = \beta(C[\sigma(x)])$. $\square$.

5 Weak consistency results for denotational semantics

As we have seen before, establishing some form of consistency result always involves the construction of an abstraction from one semantics to another. This is by definition true for weak (strong) consistency, β-consistency and by Theorem 2.10 for consistency w.r.t. a specification formalism. Hence the task of establishing a weak consistency result plays a central role. In this section we show how weak consistency results can be obtained systematically in a certain setting. We assume that we are given a language $\mathcal{P}$ with signature Σ together with a denotational semantics α and present conditions for the existence of an abstraction from a semantics α' to α. By Lemma 3.12 every abstraction between denotational (compositional) semantics is a homomorphism. Hence in order to establish weak consistency results for denotational semantics one has to look for homomorphisms between the underlying Σ-algebras.

Two main strategies for defining denotational semantics for a language that includes some notion of recursion can be distinguished: one is based on partial orders, the other on metric spaces. The idea of the partial order resp. metric approach is to ensure the existence of a least resp. a unique fixed point of the operator Φ^A of Lemma 3.5 by Tarski's resp. Banach's fixed point theorem. Then this fixed point is the least resp. unique compositional semantics on the underlying Σ-algebra. In order to apply the fixed point theorems of Tarski or Banach one has to ensure the continuity of Φ^A resp. that Φ^A is a contraction. The usual way to do so is to require that the semantic operators are continuous resp. non-distance-increasing/contracting.

If there are two cpo's $(D, \sqsubseteq)$ and $(D, \sqsubseteq')$ with the semantic operators being continuous with respect to both orderings then the least meaning functions with range $(D, \sqsubseteq)$ resp. $(D, \sqsubseteq')$ will be different in general. On the other hand if we have complete metric spaces (M, d) and (M, d') such that the operators are non-distance-increasing respectively contracting on both then the meaning functions with range (M, d) resp. (M, d') coincide, i.e. the metric plays a lesser role. This is the reason for the different behaviour of the metric and partial order approach with respect to weak consistency results which we discuss in this section. Given a compositional semantics α on an arbitrary Σ-algebra A and a homomorphism $f : A \to M$ where M is a complete metric space as above then $f \circ \alpha$ equals α^M, the unique compositional semantics on M (Theorem 5.7). In order to obtain the corresponding result $f \circ \alpha = \alpha^D$ (where $f : A \to D$ is a homomorphism and α^D is the least compositional semantics on the cpo D) we need additional assumptions either about A and f or about D which ensure that f preserves corresponding fixed points. We discuss the cases that A is cpo (Theorem 5.14) or a complete metric space (Theorem 5.18) and the case that D can be endowed with a suitable metric (Theorem 5.26).

5.1 Denotational semantics in the metric approach

In this section we present the concept of a Σ-complete-metric-space (Σ-cms for short) as it was introduced in [4]. The concept of Σ-cms's constitutes an abstraction of all those properties that are necessary to define semantics on the basis of complete metric spaces and Banach's fixed point theorem. Given any Σ-cms M there is by Theorem 5.6 an automatic way to obtain a unique compositional semantics α^M with range M. Hence providing a semantics using the metric approach really means constructing a suitable Σ-cms.

Notation 5.1 *A symbol-algebra with* guardedness conditions *is a symbol-algebra Σ which is endowed with a function $deg : Op \to I\!N_0$ such that $0 \leq deg(\omega) \leq |\omega|$ for each operator symbol ω. $deg(\omega)$ is called the* degree of guardedness *of ω. The set of* guarded statements *over Σ is given by the production system:*

$$\zeta \quad ::= \quad a \mid \omega(\zeta_1, \ldots, \zeta_{n-k}, s_1, \ldots, s_k)$$

where a is a constant symbol, ω an n-ary operator symbol with $n \geq 1$ and $k = deg(\omega)$, $s_1, \ldots, s_k$ are statements and $\zeta_1, \ldots, \zeta_{n-k}$ guarded statements.

In the following we assume that for the declaration $\sigma : Idf \to \mathcal{L}(\Sigma, Idf)$ under consideration the statements $\sigma(x)$ are guarded.

Example 5.2 The guardedness conditions in the symbol-algebra Σ_{CCS} as defined in Example 3.2 are given by: The prefixing operator symbols γ_a are guarded operator symbols, i.e. $deg(\gamma_a) = |\gamma_a| = 1$ for all $a \in Act$. The other operator symbols have 0 as the degree of guradedness, i.e. $deg(+) = deg(|) = deg(\rho_L) = deg(\ell_\lambda) = 0$. Hence a statement $s \in \mathcal{L}(\Sigma_{CCS}, Idf)$ is guarded if and only if each occurrence of an identifier x in s is in the scope of a prefixing operator symbol. This is equivalent to Milners definition of guarded CCS-terms without recursion (see [32]). □

Example 5.3 Guardedness of statements of the language $\mathcal{L}_0$ (see Example3.3) in the sense of [7] can be obtained when we define $deg(+) = deg(\|) = 0$ and $deg(;) = 1$. A statement $s \in \mathcal{L}_0$ is guarded if and only if each occurrence of an identifier x in s is contained in the second argument of a sequential composition, i.e. there exists a subterm $s_1; s_2$ of s such that the occurrence of x is in s_2. □

Definition 5.4 *A* Σ-cms *is a Σ-algebra M which is endowed with a metric δ $(0 \leq \delta \leq 1)$ such that (M, δ) is a complete metric space and such that for each operator symbol ω with $|\omega| = n \geq 1$, $deg(\omega) = k$ the associated operator $\omega_M : M^n \to M$ is non-distance-increasing and contracting in its last k arguments:*

$$\delta(\ \omega_M(\xi_1, \ldots, \xi_n),\ \omega_M(\xi'_1, \ldots, \xi'_n)\ ) \ \leq\ \max\left\{ \max_{1 \leq i \leq n-k} \delta(\xi_i, \xi'_i),\ \frac{1}{2} \cdot \max_{n-k+1 \leq j \leq n} \delta(\xi_j, \xi'_j) \right\}$$

for all $\xi_1, \ldots, \xi_n,\ \xi'_1, \ldots, \xi'_n \in M$.

In [4] it was shown that the function Φ^M (which is defined as in Lemma 3.5) is a contracting self-mapping of the complete metric space $\mathcal{P} \to M$. By Banach's fixed point theorem Φ^M has exactly one fixed point in $\mathcal{P} \to M$.

Definition 5.5 *Let σ be a declaration M an Σ-cms. The unique fixed point of Φ^M is denoted by α^M.*

Theorem 5.6 (see [4]) *Let M be a Σ-cms. Then $\alpha^M : \mathcal{P} \to M$ is the unique compositional semantics on M.*

In [4] we established the following consistency result: Given two Σ-cms's N and M and a homomorphism $f : N \to M$ then $f \circ \alpha^N = \alpha^M$. In Theorem 5.7 we extend this result in the form that N may be an arbitrary Σ-algebra.

Theorem 5.7 *Let M be a Σ-cms, A a Σ-algebra, $\alpha : \mathcal{P} \to A$ a compositional semantics and $f : A \to M$ a homomorphism from A to M. Then f is an abstraction from α to α^M, i.e. $f \circ \alpha = \alpha^M$.*

Proof: By Theorem 3.13 we have that $f \circ \alpha$ is a compositional semantics. By the uniqueness of the semantics in Theorem 5.6 we get $f \circ \alpha = \alpha^M$. □

Note that in Theorem 5.7 A is an arbitrary Σ-algebra and there is no restriction on the way in which we got the semantics on A. α could be defined by the metric or by the cpo approach or in another way.

Example 5.8 The trace resp. tree semantics of [6] on the complete metric spaces $\mathcal{P}_{\mathrm{nc}}(Act^\infty)$ and M_{cl} can be related by Theorem 5.7: Let $N = \mathcal{P}_{\mathrm{nc}}(Act^\infty)$ be the collection of all nonempty and closed subsets of Act^∞ (finite or infinite sequences over Act) endowed with the Haussdorff-metric and M_{cl} the unique complete metric space satisfying the domain equation $M \simeq \mathcal{P}_{\mathrm{closed}}(Act \times M)$. Then N and M_{cl} endowed with suitable semantic operators are Σ_0-cms and Σ_{CCS}-cms. The trace resp. tree semantics on N and M_{cl} of [6] coincide with the unique compositional semantics on N resp. M_{cl} in the sense of Theorem 5.6. In the following we will denote them by $trace_{\mathrm{cms}}$ resp. $tree_{\mathrm{cms}}$. Let $e : M_{\mathrm{cl}} \to \mathcal{P}_{\mathrm{closed}}(Act \times M_{\mathrm{cl}})$ be an isometry and let $f : M_{\mathrm{cl}} \to \mathcal{P}_{\mathrm{nc}}(Act^\infty)$ be the unique function $M_{\mathrm{cl}} \to \mathcal{P}_{\mathrm{nc}}(Act^\infty)$ satisfying the equation

$$f(X) = \{ \, a.f(Y) \; : \; (a,Y) \in e(X) \, \}.$$

The existence and uniqueness of such a function f can be concluded by the fact that $M_{\mathrm{cl}} \to \mathcal{P}_{\mathrm{nc}}(Act^\infty)$ is a complete metric space and that f is the unique fixed point of the contracting operator

$$\Psi : (M_{\mathrm{cl}} \to \mathcal{P}_{\mathrm{nc}}(Act^\infty)) \to (M_{\mathrm{cl}} \to \mathcal{P}_{\mathrm{nc}}(Act^\infty))$$

which is given by

$$\Psi(F)(X) = \{ \, a.F(Y) \; : \; (a,Y) \in e(X) \, \}.$$

It is easy to see that f is homomorphism and we get $f \circ tree_{\mathrm{cms}} = trace_{\mathrm{cms}}$.

Theorem 5.7 can also be applied to show the consistency of two noninterleaving metric semantics: an event structure semantics in the style of [3, 18] and the pomset semantics of [7] (cf. [4]). □

5.2 Denotational semantics in the cpo approach

In this section we interprete the cpo approach in our algebraic context. As it was shown in [4] the semantics on some Σ-cpo D can be defined as the least compositional semantics on D. In contrast to the metric case it is possible that there exist other compositional semantics. Hence we cannot guarantee the consistency of the semantics of homomorphic Σ-algebras.

We assume the reader to be familar with basic notions of domain theory which can be found e.g. in [2, 17]. In order to prevent confusions we explain the notion of a cpo (complete partial order) as it is used here. By a cpo we mean a partially ordered set with a bottom element such that each monotone sequence has a least upper bound. A function f between cpo's is called continuous iff f is monotone and preserves least upper bounds of monotone sequences.

Definition 5.9 *A* Σ-cpo *is a* Σ*-algebra* D *endowed with a partial order* $\sqsubseteq$ *on* D *such that* $(D, \sqsubseteq)$ *is a cpo and such that for each operator symbol* $\omega \in Op$, $|\omega| = n \geq 1$, *the associated operator* $\omega_D : D^n \to D$ *is continuous with respect to* $\sqsubseteq$.

In [4] it was shown that for each Σ-cpo D the function Φ^D is a continuous self-mapping of the cpo $\mathcal{P} \to D$. By Tarski's fixed point theorem the least fixed point of Φ^D exists.

Definition 5.10 *Let* D *be a* Σ*-cpo. The least fixed point of* Φ^D *is denoted by* α^D.

Theorem 5.11 (see [4]) *Let* D *be a* Σ*-cpo. Then* $\alpha^D : \mathcal{P} \to D$ *is the least compositional semantics on* D.

Example 5.12 In [42] resp. [43] the concept of Σ-cpo's is used to define denotational semantics for CCS on trees and labelled (prime) event structures. In the following *Tree* resp. *PrimeEv* denote the Σ_{CCS}-cpo of (synchronisation) trees resp. labelled prime event structures in the sense of [42] resp. [43]. $tree_{\mathrm{cpo}}$ resp. ev_{cpo} denote the least compositional semantics on *Tree* resp. *PrimeEv* for the language $\mathcal{P}(\Sigma_{CCS}, Idf)$.

Also a trace semantics for programs over Σ_0 or Σ_{CCS} in the style of [22] can be defined using the Σ-cpo concept. The underlying Σ-cpo is $\mathcal{P}\downarrow(Act^*_{\surd})$, the collection of all nonempty and leftclosed (w.r.t. the prefixing order) subsets of

$$Act^*_{\surd} \quad = \quad Act^* \cup \{t\surd : t \in Act^*\}$$

endowed with suitable semantic operators and the subset-relation as partial order (see [22]). Here $\surd$ is a 'new' symbol (not contained in *Act*) for representing successfull termination. □

In the metric approach we obtained as a special case of Theorem 5.7 the following consistency result: Whenever f is a homomorphism between two Σ-cms's then the semantics are consistent. In [4] we gave examples which show that this result is wrong when we deal with Σ-cpo's even if we require that f is cpo-continuous or that f is the least homomorphism between the given Σ-cpo's. The reason for this difference of the metric and partial order setting is founded in the fact that the choice of the metric does not influence the meaning function whereas in the cpo approach the meaning function depends on the underlying partial order.

Remark 5.13 By Lemma 3.12 and Theorem 5.11 we get: If D is a Σ-cpo, $\alpha : \mathcal{P} \rightarrow A$ a compositional semantics and $f : A \rightarrow D$ a homomorphism then:

$$\alpha^D \sqsubseteq f \circ \alpha$$

This is not a consistency result in our sense. Nevertheless it might be useful for verifying the correctness of programs:

1. If the elements on D can be considered as processes and if the partial order $\sqsubseteq$ on D can be interpreted in such a way that $\xi \sqsubseteq \xi'$ implies that ξ' simulates ξ (which means that each possible behaviour of ξ' is a possible behaviour of ξ) then the result $\alpha^D \sqsubseteq f \circ \alpha$ might be helpful for verifying safety properties (which assert that 'nothing bad happens'):

   We assume $\alpha(P)$ to be a high-level representation of P (e.g. the implementation) and $\alpha^D(P)$ to be a semantics for which it is shown that it meets the specification (or we assume $\alpha^D(P)$ to be the specification itself). Then $\alpha^D(P) \sqsubseteq f(\alpha(P))$ asserts that each possible behaviour of the implementation is allowed by the specification. Here we assume that $f(\alpha(P))$ can be considered as a simulation of $\alpha(P)$. In general this is not enough to verify the correctness of the implementation. In addition one has to check some liveness (progress) properties (which assert that 'something good will happen').

   As far as we know the only denotational cpo semantics which allows such an interpretation of the partial order is the failure semantics of [11]. In most other cases the partial order has the opposite meaning, i.e. $\xi \sqsubseteq \xi'$ implies that ξ simulates ξ' and not vice versa (e.g. Winskels partial order on trees [42] or prime event structures [43]). If the underlying semantic domain D is a complete lattice (instead of a cpo) then one can deal with greatest instead of least fixed points. This leads to a greatest compositional semantics α^D and hence to a consistency result $f \circ \alpha \sqsubseteq \alpha^D$ which might be helpful to prove safety properties as described above.

2. Another situation where the result $\alpha^D \sqsubseteq f \circ \alpha$ might be useful is the following: we assume a logic L and functions $F : D \rightarrow I$, $F' : A \rightarrow I'$ where I, I' are sets of interpretations for L such that:

   - $\xi \sqsubseteq \xi'$ implies that for each formula φ of L: If ξ satisfies φ then ξ' satisfies φ. More precisely: If $F(\xi)$ is a model for φ then $F(\xi')$ is a model of φ.
   - For each formula φ of L and each element $a \in A$: If $f(a)$ satisfies φ then a satisfies φ. More precisely: If $F(f(a))$ is a model of φ then $F'(a)$ is a model of φ.

   The second condition can be omitted in the case $I = I'$ and $F \circ f = F'$.

   Assume that the specification of a program P is a set of formulas of L and that $\alpha^D(P)$ meets the specification, i.e. $F(\alpha^D(P))$ is a model for all such formulas. Then by the first assumption: $F(f(\alpha(P)))$ satisfies all formulas of the specification. And hence by the second assumption: $\alpha(P)$ meets the specification. Hence instead of proving the correctness of P via the semantics α one might use the more abstract semantics α^D to prove the correctness of P.

An example for a cpo which satisfies the condition of above is the treelike domain D of [1]. [1] shows that $\xi \sqsubseteq \xi'$ implies $HML_\omega(\xi) \subseteq HML_\omega(\xi')$ where $HML_\omega(\xi)$ denotes the set of formulas φ of the Hennessy-Milner logic (without negation, infinite conjunction or disjunction) such that the transition system of ξ is a model of φ. In our terminology: L is the Hennessy-Milner logic HML_ω and I the set of transition systems in the sense of [1]. The function F can be assumed to be the inclusion $D \to I$ since the elements of D are special kinds of labelled trees which can be considered as transition systems in the sense of [1]. Then $\xi \sqsubseteq \xi'$ implies that for each Hennessy-Milner formula φ: If ξ satisfies φ (i.e. $F(\xi)$ is a model of φ) then ξ' satisfies φ (i.e. $F(\xi')$ is a model of φ). □

In the rest of this section we show how weak consistency results of the form $f \circ \alpha = \alpha^D$ can be established where α^D is a cpo semantics. We deal with the cases that α is a cpo semantics (Theorem 5.14), a metric semantics (Theorem 5.18) or an arbitrary compositional semantics (Theorem 5.20 and 5.26).

Theorem 5.14 (see [4]) *Let D, E be Σ-cpo's. If there exists a strict and continuous homomorphism $f : D \to E$ then $f \circ \alpha^D = \alpha^E$.*

Example 5.15 By Theorem 5.14 we may conclude the weak consistency of Winskels partial order semantics $tree_{\mathrm{cpo}}$ and ev_{cpo} (see Example 5.12). Let $f : PrimeEv \to Tree$ denote the least function with

$$f(\mathcal{E}) = \sum \{ act(e).f(\mathcal{E} \setminus e) : e \in \mathcal{E}[1] \}.$$

Here $\sum T_i$ means the tree which we get by taking the union of nodes and edges where we assume that all trees T_i have the same root and no other node in common. $\alpha.T$ means prefixing. If $\mathcal{E}$ is a prime event structure then $\mathcal{E}[1]$ denotes the set of events of depth 1, i.e. the set of events without any predecessor. If $e \in \mathcal{E}[1]$ then $\mathcal{E} \setminus e$ means the prime event structure which arises from $\mathcal{E}$ by removing e and all events in conflict with e. act denotes the labelling function on $\mathcal{E}$, i.e. $act(e)$ is the action which is represented by e. The existence of f can be seen as follows: $PrimeEv \to Tree$ is a cpo and the operator

$$\Psi : (PrimeEv \to Tree) \to (PrimeEv \to Tree),$$

$$\Psi(F)(\mathcal{E}) = \sum \{ act(e).F(\mathcal{E} \setminus e) : e \in \mathcal{E}[1] \}$$

is continuous. Hence there exists a least fixed point f of Ψ. It is easy to see that f is a homomorphism which is strict and continuous. Hence we conclude by Theorem 5.14 $f \circ ev_{\mathrm{cpo}} = tree_{\mathrm{cpo}}$. □

Next we present conditions which ensure the weak consistency of a given metric semantics α^M and a given cpo semantics α^D. As a special case of Theorem 5.7 we have: If $f : D \to M$ is a homomorphism then $f \circ \alpha^D = \alpha^M$.

Example 5.16 This result can be applied to show the weak consistency of Winskels partial order tree semantics $tree_{\mathrm{cpo}}$ (Example 5.12) and the treelike semantics $tree_{\mathrm{cms}}$ on the complete metric space M_{cl} (Example 5.8): Since $\mathit{Tree} \to M_{\text{cl}}$ is a complete metric space and since

$$\Psi \;:\; (\mathit{Tree} \to M_{\text{cl}}) \;\to\; (\mathit{Tree} \to M_{\text{cl}}),$$

$$\Psi(F)(T) \;=\; e^{-1}(\;\{\;(a, F(S)) \;:\; T \xrightarrow{a} S\;\}\;)$$

is contracting there exists a unique fixed point f of Ψ. Here $e : M_{\text{cl}} \to \mathcal{P}_{\text{closed}}(\mathit{Act} \times M_{\text{cl}})$ is an isometry and $T \xrightarrow{a} S$ means that there exists a son w of the root v of T such that the edge $< v, w >$ is labelled by a and S is the subtree of T with root w. It is easy to see that $f : \mathit{Tree} \to M_{\text{cl}}$ is a homomorphism from the Σ_{CCS}-cpo *Tree* into the Σ_{CCS}-cms M_{cl}. Hence we get: $f \;\circ\; \mathit{tree}_{\text{cpo}} \;=\; \mathit{tree}_{\text{cms}}$. □

Our aim is to find conditions such that for a given homomorphism $f : M \to D$ the weak consistency result $f \circ \alpha^M \;=\; \alpha^D$ can be guaranteed. By Remark 3.6: $\overline{f} \circ \Phi^M \;=\; \Phi^D \circ \overline{f}$ This leads to the general situation: Let $f : M \to D$ be a function such that $f \circ \psi \;=\; \phi \circ f$ where $\psi \;:\; M \;\to\; M$ is a contracting selfmapping of a complete metric space M and $\phi : D \to D$ is a continuous selfmapping of a cpo D. Then we have to ensure that the image of the unique fixed point $\mathit{fix}(\psi)$ under f is the least fixed point $\mathit{lfp}(\phi)$ of ϕ.

Lemma 5.17 *Let M be a complete metric space, $\psi : M \to M$ a contracting function, D a cpo, $\phi : D \to D$ a continuous function and $f \;:\; M \to D$ a function which satisfies the following conditions:*

(i) $\perp_D \;\in f(M)$

(ii) If $(\xi_n)_{n\geq 1}$ is a Cauchy sequence in M such that $(f(\xi_n))_{n\geq 1}$ is a monotone sequence in D then $f(\lim\, \xi_n) \;=\; \bigsqcup\, f(\xi_n)$.

(iii) $f \circ \psi \;=\; \phi \circ f$

Then $f(\;\mathit{fix}(\psi)\;) \;\;=\;\; \mathit{lfp}(\phi)$ *where $\mathit{fix}(\psi)$ denotes the unique fixed point of ψ in M and $\mathit{lfp}(\phi)$ denotes the least fixed point of ϕ in D.*

Proof: Let $\xi \in M$, $f(\xi) = \perp_D$ (condition (i)). Then $f(\psi^n(\xi)) \;=\; \phi^n(\perp)$ by condition (iii). By Banach's resp. Tarski's fixed point theorem and condition (ii):

$$f(\;\mathit{fix}(\psi)\;) \;=\; f\left(\lim_{n\to\infty}\; \psi^n(\xi)\right) \;=\; \bigsqcup_{n\geq 0}\; f(\psi^n(\xi)) \;=\;=\; \bigsqcup_{n\geq 0}\; \phi^n(\xi) \;=\; \mathit{lfp}(\phi).$$

□

By Lemma 5.17 (with $\mathcal{P} \to M$ instead of M and $\mathcal{P} \to D$ instead of D) and Remark 3.6 we get:

Theorem 5.18 *Let M be a Σ-cms, D a Σ-cpo and $f \;:\; M \to D$ a homomorphism such that $\perp_D \in f(M)$ and $f(\;\lim x_n\;) \;=\; \bigsqcup\; f(x_n)$ for each Cauchy sequence (x_n) in M such that $(f(x_n))$ is a monotone sequence in D. Then*

$$f \circ \alpha^M \;=\; \alpha^D.$$

Example 5.19 The trace semantics $trace_{\text{cpo}}$ of [22] on the Σ_0-cpo $D = \mathcal{P}\downarrow(Act^*_{\surd})$ (Example 5.12) and the trace semantics $trace_{\text{cms}}$ of [6] on the Σ_0-cms $M = \mathcal{P}_{\text{nc}}(Act^{\infty})$ (Example 5.8) can be related by Theorem 5.18: Let $f : M \to D$ be given by

$$f(X) = X\downarrow \cup \{ t\surd : t \in X \cap Act^* \}$$

where $X\downarrow = \{ t \in Act^* : \exists t' \in X \; t \sqsubseteq t' \}$ and where $\sqsubseteq$ denotes the prefix ordering. Then f is a homomorphism with $f(\{\emptyset\}) = \{\emptyset\} = \bot_D$ and $f(\lim X_n) = \bigsqcup f(X_n)$. Hence we get $f \circ trace_{\text{cms}} = trace_{\text{cpo}}$. □

Having in mind the consistency result of Theorem 5.7 involving metric, one way of obtaining a consistency result involving cpo could be to 'construct' a metric on a given cpo and then proceeding in analogy to Theorem 5.7. The notion of partial order on D itself is too weak to provide an interesting metric on D. If, however, we add some kind of length or rank information to $(D, \sqsubseteq)$ as it has been done in [5, 29] then interesting and relevant metrics arise for a subspace M of D. If we are able to provide a rank for a Σ-cpo $(D, \sqsubseteq)$ then we obtain a result that is analogous to Theorem 5.7.

Theorem 5.20 *Let D be a Σ-cpo and M a subset of D with $\bot_D \in M$ and which is closed w.r.t. the semantic operators ω_D, $\omega \in \Sigma$. If there exists a metric d on M which turns M into an Σ-cms (where the underlying semantic operators on M are the restrictions of the semantic operators ω_D on M) and such that $\lim \xi_n = \bigsqcup \xi_n$ for each Cauchy sequence (ξ_n) in M which is monotone in D then*

$$\alpha^M = \alpha^D.$$

In this case: If $\alpha : \mathcal{P} \to A$ is a compositional semantics and $f : A \to D$ a homomorphism with $f(A) \subseteq M$ then

$$f \circ \alpha = \alpha^D.$$

Proof: follows by Theorem 5.18 (where $f : M \to D$ is the inclusion) and by Theorem 5.7. □

As a conclusion of Theorem 5.20 we get:

Theorem 5.21 *Let A be a Σ-algebra. If there exists a partial order $\sqsubseteq$ on A and a metric d on A such that A endowed with $\sqsubseteq$ and d is both, a Σ-cpo and a Σ-cms, and such that $\lim \xi_n = \bigsqcup \xi_n$ for each monotone Cauchy sequence (ξ_n) in A then*

$$\alpha^A_{\text{cms}} = \alpha^A_{\text{cpo}}.$$

Here $\alpha^A_{\text{cms}} : \to A$ denotes the unique compositional semantics on A as a Σ-cms (Theorem 5.6) and $\alpha^A_{\text{cpo}} : \mathcal{P} \to A$ the least compositional semantics on A as a Σ-cpo (Theorem 5.11).

In the following we present conditions to define a metric on a subspace M of a Σ-cpo D which turns M into a Σ-cms and such that the limit of a monotone Cauchy sequence in M equals its least upper bound in D. Then the 'cpo semantics' on D as a Σ-cpo and the 'metric semantics' on M as an Σ-cms coincide (Theorem 5.20). We use the concept of pseudo rank ordered cpo's which are introduced in [5] The concept of (pseudo) rank orderings is closely related to the rankings considered in [12] and the projection spaces introduced in [15].

Definition 5.22 *Let D be a cpo. A* pseudo rank ordering *on D is a family $\tilde{\pi} = (\pi_n)_{n\geq 0}$ consisting of continuous function $\pi_n : D \to D$ such that*

(i) $\pi_0(\xi) = \bot_D$ *for all* $\xi \in D$

(ii) $\pi_n \circ \pi_m = \pi_m \circ \pi_n = \pi_n$ *for all* $m \geq n \geq 0$

(iii) $\pi_n \sqsubseteq id_D$ *for all* $n \geq 0$

An element $\xi \in D$ is called approximable *(w.r.t. $\tilde{\pi}$) iff $\xi = \bigsqcup_{n\geq 0} \pi_n(\xi)$. $\mathcal{M}(D, \tilde{\pi})$ denotes the set of approximable elements, i.e.*

$$\mathcal{M}(D, \tilde{\pi}) = \left\{ \xi \in D \ : \ \xi = \bigsqcup_{n\geq 0} \pi_n(\xi) \right\}.$$

A pseudo rank ordering $\tilde{\pi}$ on D is called a rank ordering *on D iff $\mathcal{M}(D, \tilde{\pi}) = D$. A* pseudo rank ordered cpo *is a pair $(D, \tilde{\pi})$ consisting of a cpo D and a pseudo rank ordering $\tilde{\pi}$ on D.*

In [5] it is shown that for each pseudo rank ordered cpo $(D, \tilde{\pi})$: $\mathcal{M}(D, \tilde{\pi})$ is a cpo which contains $\bot$ and the sets $\pi_n(D)$. The restrictions of the functions π_n on $\mathcal{M}(D, \tilde{\pi})$ yield a rank ordering on $\mathcal{M}(D, \tilde{\pi})$. The following lemma shows that the completeness in the partial order setting enforces the completeness in the metric setting and that the limits of monotone Cauchy sequences coincide with the least upper bounds. We omit the proof which is given in [5].

Lemma 5.23 *Let $(D, \tilde{\pi})$ be a pseudo rank ordered cpo. Then $M = \mathcal{M}(D, \tilde{\pi})$ endowed with the distance*

$$d[\tilde{\pi}](\xi_1, \xi_2) = \inf \left\{ \frac{1}{2^n} : \pi_n(\xi_1) = \pi_n(\xi_2) \right\}$$

is a complete ultrametric space. For each Cauchy sequence (ξ_n) in M which is monotone in D we have $\lim \xi_n = \bigsqcup \xi_n$.

If $\lambda : D^m \to D$ is a function with $\lambda(M^m) \subseteq M$ and $\pi_n \circ \lambda = \pi_n \circ \lambda \circ (\pi_n^{m-k}, \pi_{n-1}^k)$ for all $n \geq 1$ then $\lambda | M^m \to M$ is non-distance-increasing and contracting in its last k arguments. Here $m \geq 1$, $0 \leq k \leq m$.

In order to apply Theorem 5.20 we have to ensure that $\mathcal{M}(D, \tilde{\pi})$ is a Σ-cms. This means that we have to require that the semantic operators on D preserve approximibility and that the induced operators on $\mathcal{M}(D, \tilde{\pi})$ are non-distance-increasing resp. contracting.

Definition 5.24 *Let $(D, \tilde{\pi})$ be a pseudo rank ordered cpo. Then $(D, \tilde{\pi})$ is called a* Σ-pro-cpo *(Σ-pseudo-rank-ordered-cpo) iff D is a Σ-cpo such that for each operator symbol ω, $|\omega| = m$, $deg(\omega) = k$:*

- $\omega_D(\mathcal{M}(D, \tilde{\pi})^m) \subseteq \mathcal{M}(D, \tilde{\pi})$
- $\pi_n \circ \omega_D = \pi_n \circ \omega_D \circ (\pi_n^{m-k}, \pi_{n-1}^k)$ *for all* $n \geq 1$.

Lemma 5.25 *Let $(D, \tilde{\pi})$ be a Σ-pro-cpo. Then $M = \mathcal{M}(D, \tilde{\pi})$ is a Σ-cms and we have:*

$$\alpha^M = \alpha^D$$

Proof: follows by Theorem 5.20 and Lemma 5.23. □

Theorem 5.26 *Let $(D, \tilde{\pi})$ be a Σ-pro-cpo, A a Σ-algebra and $\alpha : \mathcal{P} \to A$ a compositional semantics. If $f : A \to D$ is a homomorphism with $f(A) \subseteq \mathcal{M}(D, \tilde{\pi})$ then*

$$f \circ \alpha = \alpha^D.$$

Proof: Since f is a homorphism from the Σ-algebra A into the Σ-algebra D, f is also a homorphism from the Σ-algebra A into the Σ-cms $\mathcal{M}(D, \tilde{\pi})$. By Theorem 5.7 we get that $f \circ \alpha$ is the unique compositional semantics on $\mathcal{M}(D, \tilde{\pi})$. By Lemma 5.25: $f \circ \alpha = \alpha^D$. □

Example 5.27 The concept of pseudo rank ordered cpo's can be applied to Winskels cpo's of trees and prime event structures. Here $\pi_n(\xi)$ is the n-cut of ξ (i.e. the tree/prime event structure which arises from ξ by removing all nodes/events of depth $> n$). Then all elements are approximable. It is easy to see that Winskels semantic operators satisfy the conditions of Definition 5.24. Hence Winskels semantic domains of trees resp. prime event structures are Σ_{CCS}-pro-cpo's. As an application of Theorem 5.26 one gets the weak consistency result of Example 5.15 by establishing a homomorphism from prime event structures to trees. Applying Theorem 5.26 instead of Theorem 5.14 has the advantage that there is no need to show the continuity or strictness of this homomorphism. □

6 Conclusion and related work

We introduced the notions of weak, strong and β-consistency and consistency w.r.t. a specification formalism, i.e. a set *Prop* of properties. It is shown that the various 'consistency' results from the literature fit into our framework. The connection between β-consistency and consistency w.r.t. a specification formalism is studied (Theorem 2.10). We showed that there is always a unique most abstract semantics suitable for verification (Theorem 2.4). Under the assumption that the underlying specification formalism allows for modular verification this semantics is compositional (Theorem 3.9). We also gave special attention to compositionality and verification of programs that are built by stepwise refinement. We showed that each (operational) semantics β with $\beta(C[x]) = \beta(C[\sigma[x])$ possesses a unique denotational semantics which is adequate and fully abstract w.r.t. β (Theorem 4.3). Finally, we presented conditions for establishing weak consistency results for denotational semantics.

As far as we know the only other work which deals with a general framework for establishing weak consistency result is [25] which presents a method for proving that an operational semantics is an abstraction of a denotational semantics defined in the metric approach.

Larsen [27] presents a general framework to compare the expressivity of specification formalisms. The notion of a specification formalism used in [27] differs a little bit from ours:

A specification formalism in the sense of [27] can be identified with the strong consistency equivalence classes of semantics in our sense. Each strong consistency equivalence class $\mathcal{S} = [\alpha]$ can be described by the set of properties E with $\alpha \models E$:

$$Prop(\mathcal{S}) = \{ E \subseteq \mathcal{P} : \mathcal{S} \models E \}$$

where $\mathcal{S} \models E$ iff $\alpha \models E$ for all/some semantics $\alpha \in \mathcal{S}$ (Lemma 2.14). 'Relative expressivity' in the sense of [27] adapted to our framework is weak consistency: Following [27], we say that $\mathcal{S}$ is at last as expressive as $\mathcal{S}'$ iff $\mathcal{S}' \models E$ implies $\mathcal{S} \models E$. With Lemma 2.3: The strong consistency equivalence class $[\alpha]$ is as least as expressive as $[\alpha']$ if and only if α' is an abstraction of α. Larsen's notion of 'supporting decomposition' is closely related to our notion of modularity of a specification formalism where [27] deals with contexts instead of operator symbols.

A A categorical characterization of consistency

The results of Theorem 2.4, 2.11, 2.12, 3.9, 3.10 and 3.11 can be formulated in terms of category theory:

Notation A.1 *$Sem(\mathcal{P})$ denotes the category of* **surjective** *semantics (as objects) and abstractions (as morphisms). If Prop is a specification formalism then $Sem(\mathcal{P}, Prop)$ denotes the subcategories of semantics α such that $\alpha \models E$ for all $E \in Prop$. If $\mathcal{P}$ is a set of programs as in section 3 then $Comp(\mathcal{P})$ denotes the category of surjective compositional semantics (as objects) and homomorphisms (as morphisms) and $Comp(\mathcal{P}, Prop)$ the subcategory of compositional semantics suitable for checking all properties in Prop.*

The surjectivity is not really a restriction since every semantics can be identified with the surjection one gets by shrinking the range. The surjectivity implies that for given semantics α, β there exists at most one arrow from α to β. Hence one might think of 'abstraction' as a preorder on semantics. If the semantics α in Lemma 2.3 is surjective then condition (a) in Lemma 2.3 can be replaced by

(a') There exists a unique abstraction from α to β.

Requiring that α and β are surjective condition (a) in Lemma 2.14 can be replaced by

(a') α and β are isomorphic.

Hence isomorphism in $Sem(\mathcal{P})$ is strong consistency. Establishing a weak consistency result for semantics α, β means defining a morphism between α and β.

Theorem A.2 *The categories $Sem(\mathcal{P})$ and $Comp(\mathcal{P})$ have products and coproducts for arbitrary families of (compositional) semantics.*

Proof: see Theorem 2.11, Theorem 3.10, Theorem 2.12 and Theorem 3.11. The product is the most abstract common refinement, the coproduct the most concrete common abstraction. □

Theorem A.3 *For each specification formalism Prop the category Sem($\mathcal{P}$, Prop) has a final object. If Prop is modular then Comp($\mathcal{P}$, Prop) has a final object.*

Proof: follows by Theorem 2.4 and Theorem 3.9. □

If α is a semantics that is fully abstract and adequate w.r.t. β then α is the most abstract semantics which is adequate w.r.t. β (i.e. it is the final object in the subcategory of semantics which are adequate w.r.t. β) and the most concrete semantics which is fully abstract w.r.t. β (i.e. it is the initial object in the subcategory of semantics which are fully abstract w.r.t. β).

It is an open problem if similar results can be established for suitable categories of metric semantics or cpo semantics.

References

[1] S. Abramsky: A Domain Equation for Bisimulation, Information and Computation, Vol. 92, 1991.

[2] S. Abramsky, A. Jung: Domain Theory, In S. Abramsky, D.M. Gabbay and T.S.E. Maibaum, editors, Handbook of Logic in Computer Science, Vol. 3, Clarendon Press, 1994.

[3] C. Baier, M.E. Majster-Cederbaum: The Connection between an Event Structure Semantics and an Operational Semantics for *TCSP*, Acta Informatica 31, 1994.

[4] C. Baier, M.E. Majster-Cederbaum: Denotational Semantics in the cpo and Metric Approach, Theoretical Computer Science, Vol. 135, 1994.

[5] C. Baier, M.E. Majster-Cederbaum: Construction of a CMS on a given CPO, Techn. Report 28/95, Universität Mannheim, 1994, submitted for publication.

[6] J.W. de Bakker, J.J.Ch. Meyer: Metric Semantics for Concurrency, Report CS-R8803, Centre for Mathematics and Computer Science, Amsterdam, 1988.

[7] J.W. de Bakker, J.H.A. Warmerdam: Metric Pomset Semantics for a Concurrent Language with Recursion, Report CS-R9033, Centre for Mathematics and Computer Science, Amsterdam, July 1990.

[8] J.W. de Bakker, J.I.Zucker: Processes and the Denotational Semantics of Concurrency, Information and Control, Vol.54, No. 1/2, 1982.

[9] G. Boudol, I.Castellani: Concurrency and Atomicity, Theoretical Computer Science, Vol. 59, 1988.

[10] G. Boudol, I.Castellani: Three Equivalent Semantics for *CCS*, Lecture Notes in Computer Science 469, Springer-Verlag, 1990.

[11] S.D. Brookes, C.A.R. Hoare, A.W. Roscoe: A Theory of Communicating Sequential Processes, Journal of the ACM, Vol. 31, No. 3, July 1984.

[12] K. Bruce, J.C. Mitchell: PER Models of Subtyping, Recursive Types and Higher-order Polymorphism, Proc. 19th ACM Symp. on Principles of Programming Languages, pp 316-327, 1992.

[13] E.M. Clarke, E.A. Emerson: Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic, Proc. Workshop on Logics of Programs, Lecture Notes in Computer Science 131, 1981.

[14] P. Degano, R. De Nicola, U. Montanari: On the Consistency of 'Truly Concurrent' Operational and Denotational Semantics, Proc. Symposium on Logic in Computer Science, Edinburgh, 1988.

[15] H. Ehrig, F. Parisi-Presicce, P. Boehm, C. Rieckhoff, C. Dimitrovici, M. Große-Rohde: Combining Data Type Specifications using Projection Algebras, Theoretical Computer Science, Vol. 71, pp 347-380, 1990.

[16] R. Engelking: General Topology, Sigma Series in Rure Mathematics, Vol. 6, Heldermann Verlag Berlin, 1989.

[17] G. Gierz, H. Hofmann, K. Keimel, J. Lawson, M. Mislove, D. Scott: A Compendium of Continuous Lattices, Springer-Verlag, 1980.

[18] U. Goltz, R. Loogen: Modelling Nondeterministic Concurrent Processes with Event Structures, Fundamenta Informaticae, Vol. 14, No.1, 1991.

[19] U. Goltz, A. Mycroft: On the Relationship of *CCS* and Petri Nets, Proc. ICALP 84, Lecture Notes in Computer Science 172, Springer-Verlag, 1984.

[20] M. Hennessy: Axiomatising Finite Delay Operators, Acta Informatica, Vol. 21, 1984.

[21] M. Hennessy, R. Milner: On Observing Nondeterminism and Concurrency, in Automata, Languages and Programming, Lecture Notes in Computer Science 85, 1980.

[22] C.A.R. Hoare: Communicating Sequential Processes, Prentice Hall, 1985.

[23] C.A.R. Hoare, P.E. Lauer: Consistent and Complementary Formal Theories of the Semantics of Programming Languages, Acta Informatica, Vol. 3, 1974.

[24] E. Horita: A Fully Abstract Model for a Nonuniform Concurrent Language with Parametrization and Locality, in Proc. REX Workshop '92, Lecture Notes in Computer Science 666, Springer-Verlag 1993.

[25] J.N. Kok, J.J.M.M. Rutten: Contractions in Comparing Concurrency Semantics, Report CS-R8755, Centre for Mathematics and Computer Science, November 1987.

[26] L. Lamport: Specifying Concurrent Program Modules, ACM Transactions on Programming Languages and Systems, Vol. 5, No. 2, 1983.

[27] K. Larsen: Ideal Specification Formalism = Expressivity + Compositionality + Decidability + Testability + ..., Proc. CONCUR'90, Theories of Concurrency: Unification and Extension, Lecture Notes in Computer Science 458, pp 33-56, 1990.

[28] M. Majster-Cederbaum: General Properties of Semantics, Habilitation Thesis, Technische Universität München, 1983.

[29] M. Majster-Cederbaum, C. Baier: Metric Completion versus Ideal Completion, to appear in Theoretical Computer Science, Vol. 170.

(Extended abstract in Proc. STRICT 95, J. Desel (ed.), Springer-Verlag.)

[30] M. Majster-Cederbaum, F. Zetzsche: The Comparison of a CPO-Based with a CMS-Based Semantics for *CSP*, Theoretical Computer Science, Vol. 124, 1994.

[31] R. Milner: A Calculus of Communicating Systems, Lecture Notes in Computer Science 92, Springer-Verlag, 1980.

[32] R. Milner: Communication and Concurrency, Prentice Hall, 1989.

[33] M.W. Mislove, F.J. Oles: Full Abstraction and Unnested Recursion, in Proc. REX Workshop '92, Lecture Notes in Computer Science 666, Springer-Verlag, 1993.

[34] R. de Nicola, M. Hennessy: Testing Equivalences for Processes, Theoretical Computer Science, Vol. 34, 1984.

[35] E.R. Olderog: *TCSP*: Theory of Communicating Sequential Processes, Advances in Petri-Nets 1986, Lecture Notes in Computer Science 255, Springer-Verlag, 1987.

[36] E.R. Olderog: Operational Petri-Net Semantics for *CCSP*, Advances in Petri-Nets 1987, Lecture Notes in Computer Science 266, Springer-Verlag, 1987.

[37] G.D. Plotkin: An Operational Semantics for *CSP*, Formal Description of Programming Concepts II, D. Bjorner, North Holland, 1983.

[38] A. Pnueli: The Temporal Logic of Programs, 18th Ann. Symp. on Foundations of Computer Science, Providence, 1977.

[39] W. Reisig: Partial Order Semantics versus Interleaving Semantics for *CSP*-like Languages and its Impact on Fairness, Proc. ICALP 84, Lecture Notes in Computer Science 172, Springer-Verlag, 1984.

[40] W. Reisig: Elements of a Temporal Logic Coping with Concurrency, SFB-Bericht 342/23, 92A, Techn. Universität München, 1992.

[41] M. Spanier: Vergleich zweier Theorien nebenläufiger Prozesse, Ph.D.Thesis, Universität Mannheim, 1993.

[42] G. Winskel: Synchronisation Trees, Theoretical Computer Science, Vol. 34, pp 33-82, 1984.

[43] G. Winskel: An introduction to event structures, Lecture Notes in Computer Science 354, pp 364-397, Springer-Verlag, 1988.