The ς-Semantics: A Comprehensive Semantics for Functional Programs

Olaf Chitil

Lehrstuhl für Informatik II, Aachen University of Technology
Ahornstraße 55, 52056 Aachen, Germany

chitil@informatik.rwth-aachen.de
http://www-i2.informatik.RWTH-Aachen.de/~chitil

Abstract

A comprehensive semantics for functional programs is presented, which generalizes the well-known call-by-value and call-by-name semantics. By permitting a separate choice between call-by value and call-by-name for every argument position of every function and parameterizing the semantics by this choice we abstract from the parameter-passing mechanism. Thus common and distinguishing features of all instances of the ς-semantics, especially call-by-value and call-by-name semantics, are highlighted. Furthermore, a property can be validated for all instances of the ς-semantics by a single proof. This is employed for proving the equivalence of the given denotational (fixed-point based) and two operational (reduction based) definitions of the ς-semantics. We present and apply means for very simple proofs of equivalence with the denotational ς-semantics for a large class of reduction-based ς-semantics. Our basis are simple first-order constructor-based functional programs with patterns.

Keywords: functional programming language, denotational and operational semantics, call-by-value, call-by-name, strictness, pattern-matching.

1 Introduction

Two essentially different parameter-passing-mechanisms exist for the evaluation of functional programs: call-by-value (cbv) and call-by-name (cbn) (Call-by-need evaluation is only a more efficient semantic equivalent of cbn evaluation. This efficiency aspect will just be touched in Chapter 10). Unfortunately, formal definitions of the two arising semantics are generally rather independent of each other: Firstly, the leftmost-innermost and the leftmost-outermost reduction strategy, which are often associated with cbv and cbn evaluation, respectively, look rather incomparable. Secondly, denotational definitions of cbv semantics are usually inelegant modifications of their cbn counterpart ([11]). This independence is in contrast with the fact that many modern functional programming languages even use a mixture of both parameter-passing-mechanisms. SCHEME for instance is based on cbv evaluation but includes constructs (`delay` and

`force`) for simulating call-by-need evaluation and the cbn language HASKELL permits cbv annotations for efficiency reasons ([6, 25]).

Therefore we define one unifying, generalized semantics by permitting a separate choice between cbv and cbn for every parameter position of every function and every data constructor. This semantics, named ς-semantics[1], is parameterized by this choice (ς) and thus includes the well-known cbv and cbn semantics as special instances.

Hereby we see that the mentioned association with leftmost-innermost and leftmost-outermost reduction is rather misleading. Instead we will see that the two semantics use different instances of the program for their operational semantics and that leftmost-innermost reduction is even a kind of outermost-reduction. The ς-semantics highlights the true differences between cbv and cbn evaluation. Even more important is that common features are stressed. This improves our understanding of the mentioned prevailing mixtures of cbv and cbn semantics in existing languages. Furthermore, abstracting from the parameter-passing-mechanism provides the means for stating, analysing, and proving validity of semantic properties of functional programs in general. Here we use this to prove the well-definedness and the equivalence of the three definitions, one denotational and two operational, which we give for the ς-semantics (hence in particular cbv and cbn semantics are dealt with by the same proofs). In the proof of equivalence we employ rather general techniques which even provide the means for simple proofs of equivalence with the denotational semantics for operational, reduction based ς-semantics which may be defined in the future, for instance for increased efficiency.

1.1 Functional Programs and Data-Types

For ease of presentation (see also the Conclusion) our basis are simple first-order constructor-based functional programs with patterns like the following:

```
add(x,Zero)      →  x
add(x,Succ(y))   →  Succ(add(x,y))

mult(x,Zero)     →  Zero
mult(x,Succ(y))  →  add(x,mult(x,y))
```

We regard a program as a specification of a data-type, which in the first-order case is simply an algebra consisting of a carrier set and a set of operations. One part of the data-type is already defined by the signature and the semantics of the programming language and not by the particular program. In the example we assume the carrier set to contain the constructor terms `Zero`, `Succ`(`Zero`), ..., and the constructor symbols to denote term construction operations (free interpretation). Only the meaning of the symbols `add` and `mult` is given by the program. Consequently, the semantics of a programming language defines a base data-type BDT = $\langle$carrier, base operations$\rangle$, and the semantics of a particular program P is the extension of this data-type by additional operations: DT(P) = $\langle$carrier, base operations $\cup$ defined operations$\rangle$ The ς-semantics is modular, that is extending a program does not modify but only extend its data-type.

1.2 Constructors and Pattern Matching

We use constructor-based (also called algebraic) data-types, because on the one hand only potentially non-flat data-types fully expose the relationship between cbv and cbn

[1] ς is a variation of the Greek letter sigma.

parameter-passing, and on the other hand all data-types used in modern functional programming languages are covered[2]. These languages permit the user only to define new constructor-based data-types and also the predefined data-types like lists, characters and numbers can be regarded as constructor-based (with some additional base operations which are not considered here), being implemented in a special, more efficient way.

If a function is defined by several reduction rules, then common functional programming languages use priority rules for selecting one rule (first-fitting with left-to-right argument matching in HASKELL, best-fitting in HOPE; [11]) This leads to complicated semantic definitions and makes equational reasoning rather difficult ([30, 31]). We avoid this problem by taking the term rewriting approach of confluent reduction rules.

1.3 Related Work

An implementation of a cbn language represents an argument of an operation as a boxed value, that is as a pointer to the (unboxed) actual value or to a delayed computation. If an operation is strict in an argument, then the unboxed value can be passed directly. Launchbury and others ([26, 19]) developed a typed higher-order language to express this kind of optimizations inside a cbn language. The language handles both boxed and unboxed values, unfortunately with several syntactic restrictions: unboxed expressions as arguments have to be in head normal-form, recursive types like lists cannot be unboxed, and polymorphic expressions cannot be unboxed. In contrast, we give a uniform presentation of cbv and cbn evaluation. Our basis is the abstract notion of strictness instead of the representation of values in an implementation. However, the resulting denotational semantics are quite similar.

Danvy and Hatcliff describe an implementation technique for a λ-calculus with mixed strictnesses ([8]). They define a transformation which maps such programs to an evaluation order independent λ-calculus code in continuation-passing-style. The transformation is based on the simulation of cbn evaluation in a cbv language by using explicit suspensions. Then the well-known cbv continuation-passing-style transformation is applied. For the purpose of implementation it does not make a difference that their strictness annotations are meant to be the result of a strictness analysis while our choice of parameter-passing mechanism, ς, affects the semantics. However, the transformation is not intended for comparing cbv and cbn parameter passing and giving a uniform semantic framework.

1.4 Starting-Points

The denotational semantics we define is a fixed-point semantics like those for recursive applicative program schemes ([32, 33, 23, 10, 4, 12, 7]). While the concept of a base data-type is central to this field, viewing the semantics of a whole program as a data-type is inspired by algebraic specifications ([35]). We define reduction semantics as operational semantics, but due to pattern-matching we cannot use the simple reduction semantics of recursive applicative program schemes. Instead, we apply notions and results from term rewriting systems ([9, 18, 13, 15, 14]), especially for proving the equivalence of the denotational and operational ς-semantics. For proving the equivalence of the two

[2]However, some non-free data-types like for example sets cannot be completely modelled by constructor-based data-types.

operational ς-semantics we generalize the proofs in [24] and [2] that parallel-outermost and leftmost-outermost reductions are terminating whenever possible.

1.5 Structure of the Paper

In Section 2 we introduce preliminary concepts and notations. Afterwards, the syntax of our programs is defined in Section 3. Subsequently, we consider in Section 4 properties we expect any semantics to possess, and, to prepare the generalization, we define the standard cbv and cbn semantics for our programs in Section 5. In Section 6 the ς-semantics is defined. Then we prove in Section 7 the equivalence of the three given definitions of the ς-semantics. Subsequently we prove in Section 8 some properties of our ς-semantics, especially those we consider desirable in Section 4. In Section 9 we discuss the use of ς-semantics for modelling the mixed strictnesses of modern functional programming languages and in Section 10 we consider more efficient reduction strategies than those given, as basis for realistic implementations. We conclude with a summary and some remarks in Section 11.

To avoid tiresome details many proofs are only sketched. Full proofs and additional examples are given in [5].

2 Basic Definitions and Properties

We denote the natural numbers by $\mathbb{N}$, the positive natural numbers by $\mathbb{N}_+$ and the set $\{1, 2, \ldots, n\}$ by $[n]$ for any $n \in \mathbb{N}$. $\mathbb{B} = \{\mathsf{tt}, \mathsf{ff}\}$ is the set of boolean values. If M is a set, then M^* denotes the set of words over M with ε as empty word.

The notions partial order, mapping over partial orders, algebra, ordered algebra, and finite and infinite term are standard and may be found for example in [34].

Both [9] and [18] give comprehensive surveys of term rewriting systems, the latter also of almost orthogonal term rewriting systems. However, our presentation is quite different in order to introduce the concept of an instance of a term rewriting system, which will be the basis of our operational ς-semantics, in a simple way.

2.1 Partial Orders

A *partial order* $\mathfrak{A} = \langle A, \leq_{\mathfrak{A}} \rangle$ consists of a non-empty set A and a reflexive, antisymmetric, and transitive relation $\leq_{\mathfrak{A}} \subseteq A \times A$. We use various symbols like $\leq, \subseteq, \preceq, \sqsubseteq$, and $\trianglelefteq$ for different partial order relations.

Let T be a subset of A. In case of their existence, the *least element of* T is denoted by $\mathrm{Least}_{\mathfrak{A}}(T)$ and the *least upper bound* of T by $\bigsqcup_{\mathfrak{A}} T$. T is an *(ω-)chain* iff it is non-empty,finite or countably infinite, and totally ordered. We often write chains as sequences $T = (a_i)_{i \in \mathbb{N}}$ such that $a_0 \leq a_1 \leq a_2 \leq \ldots$. The partial order $\mathfrak{A}$ is an *(ω-) complete partial order (a cpo)* iff every chain $T \subseteq A$ and the empty set have a least upper bound. Then $\mathfrak{A}$ has the least element $\bot_{\mathfrak{A}} := \bigsqcup \emptyset$.

A subset $T \subseteq A$ is *cofinal in a subset* $T' \subseteq A$ iff for all $t \in T$ there exists a $t' \in T'$ such that $t \leq t'$. If $\mathfrak{A}$ is a cpo, $T, T' \subseteq A$ are chains, and T is cofinal in T', then $\bigsqcup T \leq \bigsqcup T'$.

Let $\mathfrak{A}$ be a cpo. An element a of A is *ω-compact* iff for every chain $T \subseteq A$ the property $a \leq \bigsqcup T$ implies the existence of an $a' \in T$ such that $a \leq a'$. A cpo $\mathfrak{A}$ is *ω-inductive* iff for every element $a \in A$ there exists a chain $T \subseteq A$ of ω-compact elements such that $a = \bigsqcup T$.

2.2 Mappings over Partial Orders

If A is a set and $\mathfrak{B} = \langle B, \leq_{\mathfrak{B}} \rangle$ a partial order, then the *canonical partial order of the mapping space*, $\langle (A \to B), \preceq \rangle$, is defined by $\varphi \preceq \psi :\Longleftrightarrow \forall a \in A.\ \varphi(a) \leq_{\mathfrak{B}} \psi(a)$.

Assuming $\mathfrak{A}$ and $\mathfrak{B}$ are partial orders, a mapping $\varphi : A \to B$ is *monotonic* iff $a \leq_{\mathfrak{A}} a'$ implies $\varphi(a) \leq_{\mathfrak{B}} \varphi(a')$.

Let $\mathfrak{A}$ and $\mathfrak{B}$ be cpos. A mapping $\varphi : A \to B$ is *strict* iff $\varphi(\bot_{\mathfrak{A}}) = \bot_{\mathfrak{B}}$. A mapping φ is *(ω-)continuous* iff for every chain $T \subseteq A$ the least upper bound of $\varphi(T) := \{\varphi(t) \mid t \in T\}$ exists and $\bigsqcup \varphi(T) = \varphi(\bigsqcup T)$. The set of ($\omega$-)continuous mappings is denoted by $[A \to B]$. The *canonical partial order of (ω-)continuous mappings*, $\langle [A \to B], \preceq \rangle$, is complete.

Let φ be a mapping from a set A to itself. A *fixed-point* of φ is an element $a \in A$ such that $\varphi(a) = a$. The well-known fixed-point theorem of Knaster and Tarski states that if $\mathfrak{A}$ is a cpo and $\varphi : A \to A$ is continuous, then φ has a least fixed point given by $\mathrm{Fix}(\varphi) := \bigsqcup_{i \in \mathbb{N}} \varphi^i(\bot_{\mathfrak{A}})$.

Until here we considered only mappings with one argument. However, for $\varphi, \psi : A_1 \times \ldots \times A_n \to A$ we define $\varphi \preceq \psi :\Longleftrightarrow \forall a_1 \in A_1, \ldots, a_n \in A_n.\ \varphi(a_1, \ldots, a_n) \leq_{\mathfrak{A}} \psi(a_1, \ldots, a_n)$ and then the generalization of other previously defined notions is straightforward.

2.3 Algebras

A *signature* Σ is a set of operation symbols. Associated with every $f \in \Sigma$ is a natural number denoting its *arity*. We write $f^{(n)}$ for an operation symbol f of arity n and $\Sigma_n \subseteq \Sigma$ is the subset of all such operation symbols. A *constant* is an operation symbol of arity 0.

A *(Σ-)algebra* $\mathfrak{A} = \langle A, \alpha \rangle$ consists of a non-empty set A and a mapping $\alpha : \bigcup_{n \in \mathbb{N}} \Sigma_n \to (A^n \to A)$ which assigns a total mapping (an operation) of arity n to every operation symbol of arity n. We generally write $f^{\mathfrak{A}}$ for $\alpha(f)$. The class of all (Σ-)algebras is denoted by Alg_Σ.

Let $\mathfrak{A}$ and $\mathfrak{B}$ be Σ-algebras. A mapping $h : A \to B$ is a *homomorphism* from $\mathfrak{A}$ to $\mathfrak{B}$ iff it preserves operations, that is $h(f^{\mathfrak{A}}(a_1, \ldots, a_n)) = f^{\mathfrak{B}}(h(a_1), \ldots, h(a_n))$ for all $a_1, \ldots, a_n \in A$ and $f^{(n)} \in \Sigma$. $\mathfrak{A}$ and $\mathfrak{B}$ are *isomorphic* iff such a homomorphism h and its inverse h^{-1} exist.

Let K be a class of Σ-algebras and $X \subseteq A$ for an algebra $\mathfrak{A} \in K$. $\mathfrak{A}$ is *free in K relative to X* iff every mapping φ from X to an algebra $\mathfrak{B}$ of K can uniquely be extended to a homomorphism from $\mathfrak{A}$ to $\mathfrak{B}$. In the case $X = \emptyset$ we say that $\mathfrak{A}$ is *initial in K*. Since all algebras relatively free in K to X are isomorphic, we do not distinguish them in the following and just speak of the relatively free to X and the initial algebra in K (if any exists at all). For $K = \mathrm{Alg}_\Sigma$ we speak of *the absolutely free and the absolutely initial Σ-algebra*.

2.4 Ordered Algebras

The triple $\mathfrak{A} = \langle A, \leq, \alpha \rangle$ is an *ordered (Σ-)algebra* iff $\langle A, \leq \rangle$ is a partial order and $\langle A, \alpha \rangle$ a Σ-algebra such that all operations $\alpha(f)$ are monotonic w.r.t. the partial order. An ordered algebra may be viewed both as algebra and as partial order by ignoring $\leq$ and α, respectively.

The set of all ordered Σ-algebras with partial order $\langle A, \leq \rangle$ is denoted by $\mathrm{Alg}_\Sigma \langle A, \leq \rangle$. Its *canonical partial order* $\langle \mathrm{Alg}_\Sigma \langle A, \leq \rangle, \sqsubseteq \rangle$ is given by $\mathfrak{A} \sqsubseteq \mathfrak{B} :\Longleftrightarrow \forall f \in \Sigma.\ f^{\mathfrak{A}} \preceq f^{\mathfrak{B}}$.

An ordered Σ-algebra $\langle A, \leq, \alpha\rangle$ is *(ω-)complete* (also called *(ω-)continuous*) iff $\langle A, \leq\rangle$ is complete and the operations $f^{\mathfrak{A}}$ are continuous for all $f \in \Sigma$. The class of all complete Σ-algebras is denoted by $\mathrm{Alg}^{\infty}_{\Sigma,\perp}$. Let $\mathfrak{A}$ and $\mathfrak{B}$ be complete ordered Σ-algebras. An *(ω-)morphism* from $\mathfrak{A}$ to $\mathfrak{B}$ is a strict continuous homomorphism from $\mathfrak{A}$ to $\mathfrak{B}$. For complete algebras the notions of *isomorphism*, *freedom* and *initiality* are defined w.r.t. morphisms instead of homomorphisms.

2.5 Finite and Infinite Terms

In the following Σ is a signature with at least one constant and X a finite or countably infinite *set of variables* with $X \cap \Sigma = \emptyset$.

We define the *Σ-term algebra* $\mathcal{T}_\Sigma(X) = \langle \mathrm{T}_\Sigma(X), \tau\rangle$ to be the free Σ-algebra in Alg_Σ relative to X. The *Σ-ground term algebra* $\mathcal{T}_\Sigma = \langle \mathrm{T}_\Sigma, \tau\rangle := \mathcal{T}_\Sigma(\emptyset)$ is the initial Σ-algebra in Alg_Σ. Similarly we define the *algebra of infinite partial Σ-terms* $\mathcal{T}^{\infty}_{\Sigma,\perp}(X) = \langle \mathrm{T}^{\infty}_{\Sigma,\perp}(X), \trianglelefteq, \tau^{\infty}\rangle$ to be the free (ω-complete) Σ-algebra in $\mathrm{Alg}^{\infty}_{\Sigma,\perp}$ to X, and $\mathcal{T}^{\infty}_{\Sigma,\perp} = \langle \mathrm{T}^{\infty}_{\Sigma,\perp}, \trianglelefteq, \tau^{\infty}\rangle := \mathcal{T}^{\infty}_{\Sigma,\perp}(\emptyset)$ is the *initial Σ-algebra in* $\mathrm{Alg}^{\infty}_{\Sigma,\perp}$.

These abstract definitions make us independent of any concrete term representations. However, we generally write terms in the common prefix notation with parenthesis, for example: $f(g(x), a)$. Infinite partial terms may be visualized as particular partial mappings $\mathbb{N}^*_+ \rightharpoonup \Sigma \dot\cup X \dot\cup \{\perp\}$ (cf. positions below).

In the following we use the meta symbols t, f, x, and their variations for respectively terms ($\mathrm{T}^{\infty}_{\Sigma,\perp}(X)$), signature symbols and variables. We often write tuples of terms $(t_1, \ldots, t_n) \in (\mathrm{T}^{\infty}_{\Sigma,\perp}(X))^n$ as vectors $\vec{t}$, using the same meta symbol (here t) for the vector $\vec{t}$ and its indexed elements t_i. Moreover we abbreviate terms $f(t_1, \ldots, t_n)$ by $f(\vec{t})$.

The freeness of $\mathcal{T}^{\infty}_{\Sigma,\perp}(X)$ in $\mathrm{Alg}^{\infty}_{\Sigma,\perp}$ guarantees unique decomposability of terms, that is $t = \perp$ or $t = x$ or there exist $f^{(n)} \in \Sigma$ and terms $t_1, \ldots, t_n$ so that $t = f(\vec{t})$.

The order $\langle \mathrm{T}^{\infty}_{\Sigma,\perp}(X), \trianglelefteq\rangle$ of infinite partial terms is characterized by the properties $\perp \trianglelefteq t$ and $t'_1 \trianglelefteq t''_1, \ldots, t'_n \trianglelefteq t''_n \Longrightarrow f(\vec{t'}) \trianglelefteq f(\vec{t''})$.

Due to the existing unique homomorphism from $\mathcal{T}_\Sigma(X)$ to $\mathcal{T}^{\infty}_{\Sigma,\perp}(X)$ the set of terms $\mathrm{T}_\Sigma(X)$ can be considered as subset of $\mathrm{T}^{\infty}_{\Sigma,\perp}(X)$. Defining the set of *finite partial Σ-terms* $\mathrm{T}_{\Sigma,\perp}(X) := \mathrm{T}_{\Sigma \dot\cup \{\perp\}}(X)$ we have $\mathrm{T}_\Sigma(X) \subset \mathrm{T}_{\Sigma,\perp}(X) \subseteq \mathrm{T}^{\infty}_{\Sigma,\perp}(X)$. The terms of $\mathrm{T}_{\Sigma,\perp}(X)$ are exactly the ω-compact elements of $\mathrm{T}^{\infty}_{\Sigma,\perp}(X)$ and $\mathcal{T}^{\infty}_{\Sigma,\perp}(X)$ is ω-inductive.

Let $t, t' \in \mathrm{T}^{\infty}_{\Sigma,\perp}(X)$ be terms. The *set of variables appearing in a term t*, that is the least subset Y of X with $t \in \mathrm{T}^{\infty}_{\Sigma,\perp}(Y)$, is denoted by $\mathrm{Var}(t)$. A list $u \in \mathbb{N}^*_+$ of positive integers is called *position*. The *set of positions of a term t* is the least subset $\mathrm{Pos}(t)$ of $\mathbb{N}^*_+$ with $\varepsilon \in \mathrm{Pos}(t)$ and $t = f^{(n)}(\vec{t}), u \in \mathrm{Pos}(t_i), i \in [n] \Longrightarrow i.u \in \mathrm{Pos}(t)$. The *symbol at a position $u \in \mathrm{Pos}(t)$ in a term t*, $t(u)$, is defined by[3] $\perp(\varepsilon) := \perp$, $x(\varepsilon) := x$, $f(\vec{t})(\varepsilon) := f$ and $f(\vec{t})(i.u) := t_i.u$. The *subterm of a term t at a position $u \in \mathrm{Pos}(t)$*, t/u, is given by $t/\varepsilon := t$ and $f(\vec{t})/i.u := t_i/u$. The term $t[u \leftarrow t']$, created by *inserting the term t' at the position $u \in \mathbb{N}^*_+$ in t*, is defined by $t[\varepsilon \leftarrow t'] := t'$, $f^{(n)}(\vec{t})[i.u \leftarrow t'] := f(t_1, \ldots, t_i[u \leftarrow t'], \ldots, t_n)$ if $i \in [n]$, and $t[i.u \leftarrow t'] := t$ otherwise. Finally the *set of positions of a term t with a symbol $g \in \Sigma \dot\cup X \dot\cup \{\perp\}$* is defined by $\mathrm{Pos}(g, t) := \{u \in \mathrm{Pos}(t) \mid t(u) = g\}$. A term t is *linear* iff it does not contain multiple occurrences of the same variable, that is $|\mathrm{Pos}(x, t)| \leq 1$ for all $x \in X$. We employ the operations just defined for tuples as well. Since $\mathrm{Pos}(\vec{t}) := \{i.u_i \in \mathbb{N}^*_+ \mid u_i \in \mathrm{Pos}(t_i)\}$, we have $\varepsilon \notin \mathrm{Pos}(\vec{t})$ and hence $\mathrm{T}^{\infty}_{\Sigma,\perp}(t)$ and $(\mathrm{T}^{\infty}_{\Sigma,\perp}(t))^1$ have to be distinguished.

[3] This notation is inspired by the interpretation of terms as partial mappings $\mathbb{N}^*_+ \rightharpoonup \Sigma \dot\cup X \dot\cup \{\perp\}$.

We define two partial orders over positions: The *prefix order* $\langle \mathbb{N}_+^*, \leq \rangle$ is given by $u \leq v :\Longleftrightarrow \exists w \in \mathbb{N}_+^*.\ u.w = v.$ and the *lexicographic order* $\langle \mathbb{N}_+^*, \leq_{\text{lex}} \rangle$ by $u \leq_{\text{lex}} v :\Longleftrightarrow u \leq v \vee \exists w, u', v' \in \mathbb{N}_+^*.\ \exists i, j \in \mathbb{N}_+.\ i < j \wedge u = w.i.u' \wedge v = w.j.v'$. Two positions u and v are *independent*, written $u \parallel v$, iff neither $u \leq v$ nor $v \leq u$.

A mapping $\sigma : X \to \mathrm{T}_\Sigma(X)$ with $\sigma(x) \neq x$ for only finitely many $x \in X$ is a *substitution*. We write $[t_1/x_1, \ldots, t_n/x_n]$ for σ if $\sigma(x_i) = t_i$ for all $i \in [n]$, and similarly $[t/Y]$ if $\sigma(x) = t$ for all $x \in Y \subseteq X$, under the condition that $\sigma(x) = x$ for all other $x \in X$. Its unique extension to a homomorphism from $\mathcal{T}_\Sigma(X)$ to $\mathcal{T}_\Sigma(X)$ is denoted by σ as well and we usually write $t\sigma$ for the *instance* $\sigma(t)$ of t. If $t\sigma$ is a ground term, that is contains no variables, it is called *ground instance* of t.

With regard to operational semantics, the reader should note that we define substitutions only for finite terms and that applying a substitution is therefore an effective operation.

2.6 Term Rewriting Systems

A *rewrite rule* $l \to r$ is a pair of terms with $l \in \mathrm{T}_\Sigma(X)$ and $r \in \mathrm{T}_\Sigma(\mathrm{Var}(l))$. A *term rewriting system (TRS)* is a finite set of rewrite rules R. A left-hand side of a rule is called *redex scheme* and the set of all redex schemes of a TRS R is denoted by $\mathrm{RedS_R}$. If $l \in \mathrm{RedS_R}$ and $\sigma : \mathrm{Var}(t) \to \mathrm{T}_\Sigma$, then $l\sigma$ is a *redex of the TRS* R. Red_R is the set of all redexes of R and $\mathrm{RedPos}(t) := \{u \in \mathrm{Pos}(t) \mid t/u \in \mathrm{Red}_R\}$ is the *set of all redex positions of a term* $t \in \mathrm{T}_\Sigma$.

A term $t \in \mathrm{T}_\Sigma$ is *rewritable at a position* $u \in \mathrm{Pos}(t)$ *by a rewrite rule* $l \to r \in R$ *in a single step to a term* $t' \in \mathrm{T}_\Sigma$ iff there exists a substitution $\sigma : \mathrm{Var}(l) \to \mathrm{T}_\Sigma$ with $t/u = l\sigma$ and $t' = t[u \leftarrow r\sigma]$. Both t' and σ are uniquely determined by t, u, and $l \to r$. Hence a *(simple) reduction* is a tuple $A = \langle t, u, l \to r \rangle$, written as $A = (t \xrightarrow[l \to r]{u} t')$ or just $A = (t \xrightarrow[R]{u} t')$.

Let ϱ be a set of reductions, the ϱ-reductions. If ϱ is a proper subset of all reductions, then we mark ϱ-reductions by ϱ, that is we write $A = t \xrightarrow[l \to r, \varrho]{u} t'$ or $A = t \xrightarrow[R, \varrho]{u} t'$. The *$\varrho$-reduction relation* $\xrightarrow[R,\varrho]{} \subseteq \mathrm{T}_\Sigma \times \mathrm{T}_\Sigma$ is defined by $t \xrightarrow[R,\varrho]{} t' :\Longleftrightarrow \exists u \in \mathrm{RedPos}_R(t). \exists (l \to r) \in R :\ (t \xrightarrow[l \to r]{u} t'$ is a ϱ-reduction). A *sequential ϱ-reduction strategy* is a deterministic ϱ-reduction relation, that is where for every $t \in \mathrm{T}_\Sigma$ there is at most one $t' \in \mathrm{T}_\Sigma$ such that $t \xrightarrow[R,\varrho]{} t'$. We write $A = t_1 \xrightarrow[R,\varrho]{} t_2 \xrightarrow[R,\varrho]{} t_3 \xrightarrow[R,\varrho]{} \ldots$ for a (finite or infinite) *ϱ-reduction sequence* $A = t_1 \xrightarrow[R,\varrho]{} t_2, t_2 \xrightarrow[R,\varrho]{} t_3, t_3 \xrightarrow[R,\varrho]{} t_4, \ldots$.

$A' := \langle t, \{\langle u_i, l_i \to r_i \rangle \mid i \in [n]\} \rangle$ is a *(parallel) reduction*, written as $A' = t \xrightarrow[R]{U} t'$, iff $A = (t = t_1 \xrightarrow[l_1 \to r_1]{u_1} t_2 \xrightarrow[l_2 \to r_2]{u_2} \ldots t_n = t')$ is a reduction sequence and $U := \{u_1, \ldots, u_n\} \subseteq \mathrm{RedPos}_R(t)$ a set of mutually independent redex positions of t. This implies $t' = t[u_1 \leftarrow r_1\sigma_1] \ldots [u_n \leftarrow r_n\sigma_n]$ with the order of the replacement being irrelevant due to the independence of the redex positions. The definitions of *parallel ϱ-reduction*, $A' = t \xrightarrow[R,\varrho]{U} t'$, *parallel ϱ-reduction relation* $\xrightarrow[R,\varrho]{}$, and *parallel ϱ-reduction sequence* and *strategy* are straightforward.

Let $\xrightarrow[\varrho]{} \subseteq T \times T$ be an arbitrary relation over a set T. The *reflexive and transitive closure* of $\xrightarrow[\varrho]{}$ is denoted by $\xrightarrow[\varrho]{*}$. The relation $\xrightarrow[\varrho]{}$ is *confluent* iff for all elements $t, t', t'' \in T$ with $t \xrightarrow[\varrho]{*} t'$ and $t \xrightarrow[\varrho]{*} t''$ there exists a $\hat{t} \in T$ such that $t' \xrightarrow[\varrho]{*} \hat{t}$ and $t'' \xrightarrow[\varrho]{*} \hat{t}$. An element $t \in T$ is a *ϱ-normal-form* iff there is no $t' \in T$ with $t' \neq t$

and $t \xrightarrow[\varrho]{} t'$. An element $t' \in T$ is a *ϱ-normal-form of* $t \in T$ iff $t \xrightarrow[\varrho]{*} t'$ and t' is a ϱ-normal-form. It follows that if $\xrightarrow[\varrho]{}$ is confluent, then the ϱ-normal-form of an element $t \in T$ is unique (if it exists).

We use the properties just defined for the relations $\xrightarrow[R,\varrho]{}$ and $\xrightarrow[R,\varrho]{}\!\!\!\!\!\rightarrow$. A unique ϱ-normal-form of a term $t \in \mathrm{T}_\Sigma$ is denoted by $t{\downarrow_{R,\varrho}}$. A term $t \in \mathrm{T}_\Sigma$ is a normal-form iff $\mathrm{RedPos}_R(t) = \emptyset$.

We overload notation in that respect that for instance $t \xrightarrow[R,\varrho]{} t'$ may denote both a proposition concerning the relation $\xrightarrow[R,\varrho]{}$ and a reduction (a tuple). The meaning should be clear from the context.

2.7 Almost Orthogonal TRS

An *almost orthogonal TRS* R is a TRS which is left-linear, that is all its redex schemes are linear, and which fulfills the following condition of uniqueness concerning overlays of redexes:

> If $l{\rightarrow}r, l'{\rightarrow}r' \in R, u \in \mathrm{Pos}(l)$ with $l/u \notin X$ and $\sigma, \sigma' : X{\rightarrow}\mathrm{T}_\Sigma$, then $(l/u)\sigma = l'\sigma'$ implies $u = \varepsilon$ and $r\sigma = r'\sigma'$.

Almost orthogonal TRSs have the important property that in a reduction $t \xrightarrow[l\longrightarrow r]{u} t'$ the term t and the position u determine uniquely — not $l{\rightarrow}r$ and σ, but — the rewrite rule instance $l\sigma{\rightarrow}r\sigma$ and thereby t'. Let R be an almost orthogonal TRS. The following rather technical lemma is used in many proofs. It states that two redex schemes which are above each other in a term do not overlap.

Lemma 2.1
Let $t \in \mathrm{T}_\Sigma$, $u, v \in \mathrm{RedPos}_R(t)$ with $u < v$, and $l \in \mathrm{RedS_R}$ with $t/u = l\sigma$ for a substitution σ. Then there exists a unique $w \in \mathbb{N}_+^*$ such that $u \leq u.w \leq v$ and $l/w \in X$.

Proof idea: Show that $l/w \notin X$ for all w with $u \leq u.w \leq v$ contradicts the condition of uniqueness of almost orthogonal TRSs. □

Almost orthogonal TRSs were introduced by Rosen ([29]) and were treated in detail by O'Donnell ([24]). However, both references actually do not consider TRSs but so called subtree replacement systems which are special, generally infinite sets of rewrite rules without variables. Nonetheless, all ground instances of all reduction rules of an almost orthogonal TRS R together, that is all $l\sigma{\rightarrow}r\sigma$ with $l{\rightarrow}r \in R$ and $l\sigma, r\sigma \in \mathrm{T}_\Sigma$, form such a subterm replacement system. Subterm replacement systems suggest to use almost orthogonal TRSs in a more general way than usual.

An *instance I of an almost orthogonal TRS R* is a subset of the set of all ground instances of the rewrite rules of R. It is uniquely determined by the *set of redexes of the instance* $\mathrm{Red}_{R,I} \subseteq \mathrm{Red}_R$, because variables which occur in a right-hand side occur also in the left-hand side and because of the condition of uniqueness (cf. instance predicate Q of rule schemata in [24]). For $\mathrm{Red}_{R,I} = \mathrm{Red}_R$ we obtain the commonly considered canonical instance of a TRS. For a term $t \in \mathrm{T}_\Sigma$ we define its *I-redex positions* $\mathrm{RedPos}_{R,I}(t) := \{u \in \mathrm{RedPos}_R(t) \mid t/u \in \mathrm{Red}_{R,I}\}$. A reduction $A = t \xrightarrow[R]{u} t'$ is an *I-reduction*, that is a reduction in the instance I, iff $u \in \mathrm{RedPos}_{R,I}(t)$. I-reductions are a special kind of ϱ-reductions defined in the previous subsection. Consequently we

write $A = t \xrightarrow[R,I]{u} t'$ and have the notions of sequential and parallel I-reduction relation, I-reduction strategy and I-reduction sequence.

For the study of many properties of reduction sequences it is important to follow how redexes are rearranged in a reduction. The *residual map* maps a redex position v of a term t to its residuals, that is redex positions of a term t' which are copies of t/v under the rearrangement caused by the reduction $t \xrightarrow[l \to r]{u} t'$ (cf. [14]):

$$v \setminus (t \xrightarrow[l \to r]{u} t') := \begin{cases} \emptyset & \text{, if } v = u; \\ \{v\} & \text{, if } v \parallel u \text{ or } v < u; \\ \{u.w'.v' \mid v = u.w.v', r/w' = l/w \in X\} & \text{, if } v > u. \end{cases}$$

Example 2.1
Let $R := \{f(x,y) \to x, g(x) \to f(x,x), b \to a\}$ and consider the reduction $A := f(g(b), g(a)) \xrightarrow{1} f(f(b,b), g(a))$.

The redex positions of the first term are $\varepsilon, 1, 1.1, 2$ and those of the second term are $\varepsilon, 1, 1.1, 1.2, 2$. The rearrangement of redex positions is described by $\varepsilon \setminus A = \{\varepsilon\}, 1 \setminus A = \emptyset, 1.1 \setminus A = \{1.1, 1.2\}, 2 \setminus A = \{2\}$. □

Although the definition uses the rewrite rule $l \to r$, it actually depends only on v, t and u:

Lemma 2.2
If $A = t \xrightarrow[l \to r]{u} \hat{t}$ and $A' = t \xrightarrow[l' \to r']{u} \hat{t}$ are reductions and $v \in \mathrm{RedPos}_R(t)$, then $v \setminus A = v \setminus A'$.

Proof idea: Simple but tiresome, using Lemma 2.1. □

Reductions preserve the independence of redex positions:

Lemma 2.3
If $t \xrightarrow{w} t'$ is a reduction and $u, v \in \mathrm{RedPos}_R(t)$, then $u \parallel v$ implies $(u \setminus t \xrightarrow{w} t') \parallel (v \setminus t \xrightarrow{w} t')$.

Proof idea: Case analysis on the relative order of u, v and w, using Lemma 2.1. □

Because of this lemma the following *extension of the residual map to parallel reductions* is well-defined:

$$u \setminus t \xrightarrow{V} t' := \begin{cases} \{u\} & \text{, if for all } v \in V \text{ either } v \parallel u \text{ or } u < v; \\ u \setminus t \xrightarrow{v} t'' & \text{, if } v \in V \text{ exists such that } v \leq u. \end{cases}$$

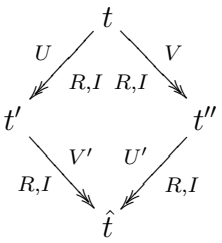
Furthermore we define the *residual map for sets of redex positions*:

$$\begin{aligned} U \setminus t \xrightarrow[R,I]{} t' &:= \bigcup \{u \setminus t \xrightarrow[R,I]{} t' \mid u \in U\}, \\ U \setminus t \xrightarrow[R,I]{} t' &:= \bigcup \{u \setminus t \xrightarrow[R,I]{} t' \mid u \in U\}. \end{aligned}$$

A set of I-redexes $\mathrm{Red}_{R,I}$ is *residually closed* iff for all I-reductions $t \xrightarrow[R,I]{} t'$ and all positions $u \in \mathrm{RedPos}_{R,I}(t)$ the inclusion $u \setminus t \xrightarrow[R,I]{} t' \subseteq \mathrm{RedPos}_{R,I}(t')$ holds. Obviously this property is not fulfilled for arbitrary sets of I-redexes. However, without it the notion of residual map is rather useless.

Lemma 2.4 *Residual closure of* Red_R
The set of all redexes of an almost orthogonal TRS R, Red_R, is residually closed. In particular we have $t/v = t'/v'$ for all reductions $t \xrightarrow{u} t'$ and $v \in \mathrm{RedPos}_R(t)$ with $v \not< u$ and all $v' \in v \setminus t \xrightarrow{u} t'$.
Proof idea: Case analysis on the relative order of u and v, using Lemma 2.1. □

Lemma 2.5 *Parallel moves lemma*
Let $\mathrm{Red}_{R,I}$ be residually closed. If $t \xrightarrow[R,I]{U}\!\!\!\!\rightarrow\, t'$ and $t \xrightarrow[R,I]{V}\!\!\!\!\rightarrow\, t''$ are reductions, then exists a unique $\hat{t} \in \mathrm{T}_\Sigma$ such that $t' \xrightarrow[R,I]{V'}\!\!\!\!\rightarrow\, \hat{t}$ and $t'' \xrightarrow[R,I]{U'}\!\!\!\!\rightarrow\, \hat{t}$ are reductions with $U' := U \setminus t \xrightarrow[R,I]{V}\!\!\!\!\rightarrow\, t''$ and $V' := V \setminus t \xrightarrow[R,I]{U}\!\!\!\!\rightarrow\, t'$, that is:



Proof: [24, 5]. □

Consequently $\xrightarrow[R,I]{}$ and $RP_{R,I}$ are confluent for residually closed redex sets $\mathrm{Red}_{R,I}$.

Finally an I-redex position of a term is *innermost/outermost* iff it is maximal/minimal w.r.t. the prefix order $\langle \mathbb{N}_+^*, \leq \rangle$ in the set of all I-redex positions of the term. The *leftmost-innermost/leftmost-outermost I-redex position of a term* is its least innermost/outermost I-redex position w.r.t. the lexicographic order $\langle \mathbb{N}_+^*, \leq_{\mathrm{lex}} \rangle$. We write $t \xrightarrow[R,I,\mathrm{li}]{} t'$, $t \xrightarrow[R,I,\mathrm{lo}]{} t'$, $t \xrightarrow[R,I,\mathrm{no}]{}$ for a reduction $t \xrightarrow[R,I]{} t'$ where u is the leftmost-innermost, the leftmost-outermost, not an outermost I-redex position of t, respectively, and we use corresponding notation for the associated I-reduction relations.

3 Abstract Syntax

Here we define the syntax of simple constructor-based first-order functional programs. Since we later define numerous different semantics for this syntax, we obtain numerous different functional programming languages.

We use a signature which distinguishes between constructor and (definable) function symbols.

Definition 3.1
Let $\mathcal{C}$ be a signature of *constructor symbols* with $\mathcal{C}_0 \neq \emptyset$ and $\mathcal{F}$ be a signature of *function symbols* so that $\mathcal{C} \cap \mathcal{F} = \emptyset$. Then $\Sigma = (\mathcal{C}, \mathcal{F})$ is a *program signature*. □

We often regard the program signature plainly as signature $\Sigma = \mathcal{F} \dot{\cup} \mathcal{C}$ with every operation symbol $g \in \Sigma$ nevertheless uniquely identifiable as constructor or function symbol. In all our examples constructor symbols start with a capital letter and function symbols with a small one, as in Miranda[4] and Haskell. We use the same convention

[4]Miranda is a trademark of Research Software Ltd.

for meta variables denoting signature symbols, using small letters for meta variables ranging over the whole program signature.

Definition 3.2
Let Σ be a program signature. An ordered pair of terms

$$f(p_1, \ldots, p_n) \;\; \rightarrow \;\; r$$

with $f^n \in \mathcal{F}$, $p_1, \ldots, p_n \in \mathrm{T}_{\mathcal{C}}(X)$, $f(p_1, \ldots, p_n)$ linear, and $r \in \mathrm{T}_{\Sigma}(\mathrm{Var}(\vec{p}))$ is a *program rule* and $\vec{p}$ is called a *pattern*. A finite set P of program rules which fulfills the *condition of uniqueness*

$$l_1 \rightarrow r_1, l_2 \rightarrow r_2 \in P,\; l_1\sigma_1 = l_2\sigma_2 \implies r_1\sigma_1 = r_2\sigma_2$$

$(\sigma_1, \sigma_2 : X \rightarrow \mathrm{T}_{\Sigma})$ is a *program*. We sometimes write P_{Σ} to state that P is a program over the signature Σ, since this information cannot be deduced from the pure set of program rules. The set of all programs over Σ is denoted by Prog_{Σ}. □

An example of a program was already given in the introduction. As explained there, we do not want to use any priority rule for pattern matching. Using the first-fitting rule would even be impossible since it assumes an order which is supplied by a concrete program text but not by our abstract definition of a program as a set of rules.

Under these circumstances the condition of uniqueness is needed to guarantee that a program specifies (deterministic) functions. Its significance and that of the left-linearity condition become fully clear in the proof of well-definedness of the semantics in Subsections 6.2 and 6.4.

The reader should notice that patterns do not need to be *exhaustive* in the sense that for all $f^n \in \mathcal{F}$ and $t_1, \ldots, t_n \in \mathrm{T}_{\mathcal{C}}$ there would be $f(\vec{p}) \rightarrow r \in P$ and a substitution σ with $f(\vec{t}) = f(\vec{p})\sigma$. There may even be function symbols without any program rule.

Obviously the definition of programs implies:

Corollary 3.1
A program is an almost orthogonal TRS. □

This gives us a confluent reduction relation $\xrightarrow[P]{}$ for the operational semantics and enables us to exploit many further known properties and proof methods.

In the following P is an arbitrary program over an arbitrary program signature $\Sigma = (\mathcal{C}, \mathcal{F})$.

4 Kinds of Semantics

4.1 Defining Semantics

For defining a semantics we first define a base data-type $\mathrm{BDT}_{\mathcal{C}}$. Then we use it and the rules of a program $P \in \mathrm{Prog}_{\Sigma}$ for defining the data-type $\mathrm{DT}(P)$ of the program.

The principal idea of a denotational fixed-point semantics is to associate a transformation $\Phi_P : \mathrm{Alg}_{\Sigma} \rightarrow \mathrm{Alg}_{\Sigma}$ with the program, which fixes the function operators by doing the pattern matching and interpreting the right-hand sides of the program rules. Due to the recursive nature of the program rules the transformation Φ_P needs an algebra as input and the data-type is defined as a fixed-point of Φ_P.

As usual all operators are continuous mappings over cpos, which is justified by interpreting the partial order as an order of information content (cf. Appendix B.2 of [11]). Hence all used algebras are complete. Since the data-type shall be an extension of the base data-type, the transformation Φ_P is actually only defined over a set of admissible data-types, the *interpretations* $\mathrm{Int}_\Sigma := \{\mathfrak{A} \in \mathrm{Alg}^\infty_{\Sigma,\bot} \mid \mathfrak{A}|_{(\mathcal{C},\emptyset)} = \mathrm{BDT}_\mathcal{C}\}$. Together with the canonical partial order $\sqsubseteq$ of complete algebras we get the cpo $\langle \mathrm{Int}_\Sigma, \sqsubseteq\rangle$. The data-type $\mathrm{DT}(P)$ is defined as the least fixed-point of the continuous transformation Φ_P.

An operational reduction semantics is based on the computation on terms using a reduction relation $\xrightarrow[P]{}$. Therefore it does not (directly) assign a data-type to a program but a *term semantics* $[\![\cdot]\!]_P : \mathrm{T}_\Sigma \to A$, a mapping from program terms T_Σ to a *computation domain* A. This computation domain shall be identical with the carrier set of the data-type $\mathrm{DT}(P) = \langle A, \alpha\rangle$.

A data-type fixes a term semantics as well: If $\mathfrak{A} = \langle A, \alpha\rangle \in \mathrm{Alg}_\Sigma$, $Y \subseteq X$, and $\beta : Y \to A$ is a valuation, then the *algebraic term semantics* $[\![\cdot]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} : \mathrm{T}_\Sigma(Y) \to A$ is the unique homomorphism from $\mathrm{T}_\Sigma(Y)$ to $\mathfrak{A}$ which extends β. Conversely, an algebra $\mathfrak{A}_{[\![\cdot]\!]} = \langle A, \alpha\rangle$ is a *data-type of the term semantics* $[\![\cdot]\!] : \mathrm{T}_\Sigma \to A$, if $f^{\mathfrak{A}_{[\![\cdot]\!]}}(a_1, \ldots a_n) = [\![f(t_1, \ldots, t_n)]\!]$ for all $f^{(n)} \in \Sigma$, $t_1, \ldots, t_n \in \mathrm{T}_\Sigma$, and $a_1, \ldots, a_n \in A$ with $[\![t_1]\!] = a_1, \ldots, [\![t_n]\!] = a_n$. Evidently we have $[\![\cdot]\!] = [\![\cdot]\!]^{\mathrm{alg}}_{\mathfrak{A}_{[\![\cdot]\!]}}$.

In general there are several data-types for one term semantics, because there may not be for every $a \in A$ a $t \in \mathrm{T}_\Sigma$ with $[\![t]\!] = a$. We see in Section 7 that for the ς-semantics there is a one-to-one relationship though, that is the ς-data-type could be defined using only the ς-term semantics. Nonetheless, there exist many reduction semantics for one fixed-point semantics, since many different reduction strategies fix the same term semantics.

4.2 Properties of Semantics

A term semantics $[\![\cdot]\!] : \mathrm{T}_\Sigma \to A$ is *compositional* (or invariant) iff for all $t', t'' \in \mathrm{T}_\Sigma$ and $t \in \mathrm{T}_\Sigma(\{\Box\})$ we have $[\![t']\!] = [\![t'']\!] \Longrightarrow [\![t[t'/\Box]]\!] = [\![t[t''/\Box]]\!]$. Compositionality reflects the nature of terms as compound objects and only a compositional term semantics engenders a semantic equality ($t \simeq t' \Longleftrightarrow [\![t]\!] = [\![t']\!]$) which is a congruence. Algebraic term semantics are compositional, because they are homomorphisms. Likewise, data-types of term semantics $[\![\cdot]\!]$ exist only for compositional $[\![\cdot]\!]$, since otherwise the $f^{\mathfrak{A}_{[\![\cdot]\!]}}$ are not well-defined. Hence, an operational semantics needs to be compositional so that an equivalent denotational semantics can exist.

The following term semantics $[\![\cdot]\!]^{\mathrm{nf}}_P : \mathrm{T}_\Sigma \to \mathrm{T}_\mathcal{C} \dot\cup \{\bot\}$, naïvely defined on the basis of normal-forms, is not compositional:

$$[\![t]\!]^{\mathrm{nf}}_P := \begin{cases} t\!\downarrow_P & \text{, if } t\!\downarrow_P \text{ exists and } t\!\downarrow_P \in \mathrm{T}_\mathcal{C}; \\ \bot & \text{, otherwise.} \end{cases}$$

Example 4.1
In this and several further examples we use the well-known 'sugared' syntax for lists instead of $\mathtt{Nil}^{(0)}$ and $\mathtt{Cons}^{(2)}$.

```
list1       →  []:list1
list2       →  [[]]:list2
head(x:xs)  →  x
```

The term `head([])` which is in normal-form but shall denote $\bot$ demonstrates the necessity of the condition $t\!\downarrow_P \in \mathrm{T}_{\mathcal{C}}$ in the definition of $[\![\cdot]\!]^{\mathrm{nf}}_P$.

We see that neither `list1` nor `list2` have a normal-form and hence $[\![\texttt{list1}]\!]^{\mathrm{nf}}_P = [\![\texttt{list2}]\!]^{\mathrm{nf}}_P = \bot$. Nonetheless we have

$$\begin{array}{llllll} \texttt{head(}\underline{\texttt{list1}}\texttt{)} & \xrightarrow[P]{} & \underline{\texttt{head}}\texttt{([]:list1)} & \xrightarrow[P]{} & \texttt{[]} \\ \texttt{head(}\underline{\texttt{list2}}\texttt{)} & \xrightarrow[P]{} & \underline{\texttt{head}}\texttt{([[]]:list2)} & \xrightarrow[P]{} & \texttt{[[]]} \end{array}$$

and hence $[\![\texttt{head(list1)}]\!]^{\mathrm{nf}}_P = \texttt{[]} \neq \texttt{[[]]} = [\![\texttt{head(list2)}]\!]^{\mathrm{nf}}_P$. This shows that $[\![\cdot]\!]^{\mathrm{nf}}_P$ is not compositional. □

Finally a property deserves attention although it is guaranteed by every (reasonable) reduction or fixed-point semantics: modularity. Any program can be extended by new function symbols together with respective program rules. We expect the data-type assigned to the extended program to be an extension of the data-type of the original program, just as that one is an extension of the base data-type (cf. hierarchical specifications in Subsection 5.4 of [35]). This also reveals the natural characterization of the base data-type as the data-type of the empty program: $\mathrm{BDT}_{\mathcal{C}} = \mathrm{DT}(\emptyset_{\mathcal{C}})$.

Formally we call a program $P_{(\mathcal{C},\mathcal{F}')}$ an *extension* of a program $P_{(\mathcal{C},\mathcal{F})}$ iff $P_{(\mathcal{C},\mathcal{F})} \subseteq P_{(\mathcal{C},\mathcal{F}')}$ and for all rules $f(\vec{p})\to r \in P_{(\mathcal{C},\mathcal{F}')} \setminus P_{(\mathcal{C},\mathcal{F})}$ we have $f \in \mathcal{F}' \setminus \mathcal{F}$. A semantics $\mathrm{DT} : \mathrm{Prog}_\Sigma \to \mathrm{Alg}_\Sigma$ is *modular* iff $\mathrm{DT}(P_{(\mathcal{C},\mathcal{F}')})|_{(\mathcal{C},\mathcal{F})} = \mathrm{DT}(P_{(\mathcal{C},\mathcal{F})})$ for all programs $P_{(\mathcal{C},\mathcal{F})}$ and all its extensions $P_{(\mathcal{C},\mathcal{F}')}$.

A compositional reduction semantics defines modular data-types, because the term semantics of a term is not changed by additional program rules for new function symbols. Similarly, any semantics defined as least fixed-point of a transformation Φ_P which uses only the program rules $f(\vec{p})\to r \in P$ for fixing the operation of the function symbol f is modular.

A well-known semantics which is not modular is the *quotient algebra semantics* ([9]), which defines the data-type of a program to be the free term algebra $\mathcal{T}_\Sigma$ modulo the equations (rules) of the program. Extending a program may even not only change the (old) operations of the data-type but also the carrier set.

5 Cbv and Cbn Semantics

Preparing the generalization to ς-semantics in the next section, we define here the well-known cbv and cbn semantics. Some amendments are necessary due to the constructors and the pattern-matching.

We do not give any statement about well-definedness or equivalence of the reduction and fixed-point semantics, because that is done for the more general ς-semantics in Sections 6 and 7.

5.1 Cbv Reduction Semantics

We start with the reduction semantics, since cbv and cbn are rather operational notions.

Definition 5.1
Let $t \in \mathrm{T}_\Sigma$ be a term.

- An position $u \in \mathrm{RedPos}_P(t)$ is an *innermost* redex position of the term t* iff $t/u = f(c_1,\ldots,c_n)$ with $c_1,\ldots,c_n \in \mathrm{T}_{\mathcal{C}}$.

- The *leftmost-innermost** *(*li**) redex position of the term* t is the least innermost* redex position of t w.r.t. the lexicographical order.
- A reduction $A = t \xrightarrow[l \to r]{u} t'$ is a *leftmost-innermost** *reduction*, written $A = t \xrightarrow[P,\mathrm{li}^*]{} t'$, iff u is the li*redex position of t. Hereby the *leftmost-innermost** *reduction strategy* $\xrightarrow[P,\mathrm{li}^*]{}$ is defined as well.

□

Definition 5.2
The *li** *reduction* or *cbv reduction semantics* $[\![\cdot]\!]^{\mathrm{red}}_{P,\mathrm{cbv}} : \mathrm{T}_\Sigma \to \mathrm{T}_\mathcal{C} \dot\cup \{\bot\}$ is defined by

$$[\![t]\!]^{\mathrm{red}}_{P,\mathrm{cbv}} := \begin{cases} t{\downarrow_{P,\mathrm{cbv}}} & \text{, if } t{\downarrow_{P,\mathrm{cbv}}} \text{ exists and } t{\downarrow_{P,\mathrm{cbv}}} \in \mathrm{T}_\mathcal{C}; \\ \bot & \text{, otherwise.} \end{cases}$$

□

We avoid the name leftmost-innermost reduction, since the li*reduction strategy is not identical with the leftmost-innermost reduction strategy. The leftmost-innermost redex position of a term is not always a li*redex position (the converse is true), due to the possibility of patterns which are not exhaustive.

Example 5.1
Let $P = \{\mathtt{f(x)} \to \mathtt{A}\}$ be a program with the function symbols $\mathtt{f}^{(1)}$ and $\mathtt{a}^{(0)}$. Then the term `f(a)` is still reducible by leftmost-innermost reduction[5]: $\underline{\mathtt{f(a)}} \xrightarrow[\mathrm{li}]{} \mathtt{A}$, but `f(a)` does not contain any li*redex. Hence $[\![\mathtt{f(a)}]\!]^{\mathrm{red}}_{P,\mathrm{cbv}} = \bot$. □

In contrast to the normal-form semantics the cbv semantics is compositional:

Example 5.2
Using the program of Example 4.1 we still have $[\![\mathtt{list1}]\!]^{\mathrm{red}}_{P,\mathrm{cbv}} = [\![\mathtt{list2}]\!]^{\mathrm{red}}_{P,\mathrm{cbv}} = \bot$, but also

$$\begin{array}{lllll} \mathtt{head}(\underline{\mathtt{list1}}) & \xrightarrow[P,\mathrm{li}^*]{} & \mathtt{head([]:}\underline{\mathtt{list1}}\mathtt{)} & \xrightarrow[P,\mathrm{li}^*]{} & \ldots \\ \mathtt{head}(\underline{\mathtt{list2}}) & \xrightarrow[P,\mathrm{li}^*]{} & \mathtt{head([[]]:}\underline{\mathtt{list2}}\mathtt{)} & \xrightarrow[P,\mathrm{li}^*]{} & \ldots \end{array}$$

and hence $[\![\mathtt{head(list1)}]\!]^{\mathrm{red}}_{P,\mathrm{cbv}} = \bot = [\![\mathtt{head(list2)}]\!]^{\mathrm{red}}_{P,\mathrm{cbv}}$ □

5.2 Cbv Fixed-Point Semantics

Definition 5.3
The ordered $\mathcal{C}$-algebra $\mathrm{BDT}_{\mathcal{C},\mathrm{cbv}} := \mathcal{T}^\bot_\mathcal{C} := \langle \mathrm{T}^\bot_\mathcal{C}, \trianglelefteq, \tau \rangle$ with τ given by $\langle \mathrm{T}_\mathcal{C}, \trianglelefteq, \tau \rangle = \mathcal{T}_\mathcal{C}$ and $\mathrm{T}^\bot_\mathcal{C} := \mathrm{T}_\mathcal{C} \dot\cup \{\bot\}$ is the *cbv base data-type* over the constructor symbols $\mathcal{C}$. □

To distinguish syntax and semantics we underline meta-variables ranging over the carrier set of base types[6], for example $\underline{t} \in \mathrm{T}^\bot_\mathcal{C}$.

[5] We underline the redexes reduced in a reduction.

[6] In a few cases like the reduction cbv semantics of Subsection 5.1 this distinction is not possible or too awkward.

Definition 5.4
The set $\mathrm{Int}_{\Sigma,\mathrm{cbv}} := \{\mathfrak{A} \in \mathrm{Alg}^{\infty}_{\Sigma,\perp} \mid \mathfrak{A}|_{(\mathcal{C},\emptyset)} = \mathrm{BDT}_{\mathcal{C},\mathrm{cbv}}\}$ is the set of *cbv interpretations* and $\langle \mathrm{Int}_{\Sigma,\mathrm{cbv}}, \sqsubseteq \rangle$ is the canonical cpo of cbv interpretations with least element $\perp_{\mathrm{cbv}} := \perp_{\mathrm{Int}_{\Sigma,\mathrm{cbv}}}$. □

Note that $\perp_{\mathrm{cbv}}$ and $\mathrm{BDT}_{\mathcal{C},\mathrm{cbv}}$ are not identical but $\perp_{\mathrm{cbv}}|_{(\mathcal{C},\emptyset)} = \mathrm{BDT}_{\mathcal{C},\mathrm{cbv}}$.

Definition 5.5
The *cbv transformation* of $P \in \mathrm{Prog}_{\Sigma}$, $\Phi_{P,\mathrm{cbv}} : [\mathrm{Int}_{\Sigma,\mathrm{cbv}} \to \mathrm{Int}_{\Sigma,\mathrm{cbv}}]$, is defined by

$$f^{\Phi_{P,\mathrm{cbv}}(\mathfrak{A})}(\vec{\underline{t}}) := \begin{cases} [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} & , \text{ if } f(\vec{p}) \to r \in P \text{ and } \beta : \mathrm{Var}(\vec{p}) \to \mathrm{T}_{\mathcal{C}} \text{ exist} \\ & \quad \text{with } [\![p_1]\!]^{\mathrm{alg}}_{(\mathrm{BDT}_{\mathcal{C},\mathrm{cbv}}),\beta} = \underline{t}_1, \ldots, [\![p_n]\!]^{\mathrm{alg}}_{(\mathrm{BDT}_{\mathcal{C},\mathrm{cbv}}),\beta} = \underline{t}_n; \\ \perp & , \text{ otherwise;} \end{cases}$$

for all $f^{(n)} \in \mathcal{F}$, $\vec{\underline{t}} \in (\mathrm{T}^{\perp}_{\mathcal{C}})^n$, and $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\mathrm{cbv}}$. □

In the cbv transformation pattern-matching is performed by the equations $[\![p_i]\!]^{\mathrm{alg}}_{(\mathrm{BDT}_{\mathcal{C},\mathrm{cbv}}),\beta} = \underline{t}_i$. Since patterns consist only of constructor symbols, the base data-type $\mathrm{BDT}_{\mathcal{C},\mathrm{cbv}}$ suffices here; using $\mathfrak{A}$ instead is possible but would suggest a non-existing dependence. Restricting the range of the valuation β to $\mathrm{T}_{\mathcal{C}}$ instead of $\mathrm{T}^{\perp}_{\mathcal{C}}$ assures that pattern matching never succeeds if $\underline{t}_i = \perp$ for some $i \in [n]$. This reflects the fact that all operations are strict in the cbv semantics.

Definition 5.6
The *cbv fixed-point data-type* $\mathrm{DT}^{\mathrm{fix}}_{\mathrm{cbv}}(P) \in \mathrm{Int}_{\Sigma,\mathrm{cbv}}$ is defined by

$$\mathrm{DT}^{\mathrm{fix}}_{\mathrm{cbv}}(P) := \mathrm{Fix}(\Phi_{P,\mathrm{cbv}}) = \bigsqcup_{i \in \mathbb{N}} (\Phi_{P,\mathrm{cbv}})^i(\perp_{\mathrm{cbv}})$$

and the *cbv fixed-point (term) semantics* $[\![\cdot]\!]^{\mathrm{fix}}_{P,\mathrm{cbv}} : \mathrm{T}_{\Sigma} \to \mathrm{T}^{\perp}_{\mathcal{C}}$ by

$$[\![t]\!]^{\mathrm{fix}}_{P,\mathrm{cbv}} := [\![t]\!]^{\mathrm{alg}}_{\mathrm{DT}^{\mathrm{fix}}_{\mathrm{cbv}}(P)}$$

□

Example 5.3 *Determining a cbv fixed-point data-type*
Taking the program of Example 4.1, its cbv fixed-point data-type is determined by means of the following table. Let $\mathfrak{A}_i := (\Phi_{P,\mathrm{cbv}})^i(\perp_{\mathrm{cbv}})$ for all $i \in \mathbb{N}$ and $\mathfrak{A}_{\infty} := \mathrm{DT}^{\mathrm{fix}}_{\mathrm{cbv}}(P)$.

	$i = 0$	$i = 1$	$\ldots$	$i = \infty$
$\mathtt{list1}^{\mathfrak{A}_i}()$	$\perp$	$[\![\mathtt{[]:list1}]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbv}},(\mathtt{list1} \mapsto \perp)} = \perp$	$\ldots$	as for $i = 1$
$\mathtt{list2}^{\mathfrak{A}_i}()$	$\perp$	$[\![\mathtt{[[]]:list2}]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbv}},(\mathtt{list2} \mapsto \perp)} = \perp$	$\ldots$	as for $i = 1$
$\mathtt{head}^{\mathfrak{A}_i}$	$\underline{t} \mapsto \perp$	$\begin{pmatrix} \perp \mapsto \perp \\ \mathtt{[]} \mapsto \perp \\ \underline{t}_1 \mathtt{:} \underline{t}_2 \mapsto \underline{t}_1 \end{pmatrix}$	$\ldots$	as for $i = 1$

with $\underline{t} \in \mathrm{T}^{\perp}_{\mathcal{C}}$ and $\underline{t}_1, \underline{t}_2 \in \mathrm{T}_{\mathcal{C}}$. □

5.3 Cbn Fixed-Point Semantics

While the cbv semantics enforces strictness of all operations, the cbn semantics defines non-strict constructor operations, thus permitting partial and infinite data structures.

Definition 5.7
The ordered $\mathcal{C}$-algebra $\mathrm{BDT}_{\mathcal{C},\mathrm{cbn}} := \mathcal{T}^{\infty}_{\mathcal{C},\perp}$ is the *cbn base data-type* over the constructor symbols $\mathcal{C}$. □

Definition 5.8
The set $\mathrm{Int}_{\Sigma,\mathrm{cbn}} := \{\mathfrak{A} \in \mathrm{Alg}^{\infty}_{\Sigma,\perp} \mid \mathfrak{A}|_{(\mathcal{C},\emptyset)} = \mathrm{BDT}_{\mathcal{C},\mathrm{cbn}}\}$ is the set of *cbn interpretations* and $\langle \mathrm{Int}_{\Sigma,\mathrm{cbn}}, \sqsubseteq \rangle$ is the canonical cpo of cbv interpretations with least element $\perp_{\mathrm{cbn}} := \perp_{\mathrm{Int}_{\Sigma,\mathrm{cbn}}}$. □

Definition 5.9
The *cbn transformation* of $P \in \mathrm{Prog}_{\Sigma}$, $\Phi_{P,\mathrm{cbn}} : [\mathrm{Int}_{\Sigma,\mathrm{cbn}} \to \mathrm{Int}_{\Sigma,\mathrm{cbn}}]$, is defined by

$$f^{\Phi_{P,\mathrm{cbn}}(\mathfrak{A})}(\vec{\underline{t}}) := \begin{cases} [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} & , \text{ if } f(\vec{p}) \to r \in P \text{ and } \beta : \mathrm{Var}(\vec{p}) \to \mathrm{T}^{\infty}_{\mathcal{C},\perp} \text{ exist} \\ & \quad \text{with } [\![p_1]\!]^{\mathrm{alg}}_{(\mathrm{BDT}_{\mathcal{C},\mathrm{cbn}}),\beta} = \underline{t}_1, \ldots, [\![p_n]\!]^{\mathrm{alg}}_{(\mathrm{BDT}_{\mathcal{C},\mathrm{cbn}}),\beta} = \underline{t}_n; \\ \perp & , \text{ otherwise;} \end{cases}$$

for all $f^{(n)} \in \mathcal{F}$, $\vec{\underline{t}} \in (\mathrm{T}^{\infty}_{\mathcal{C},\perp})^n$ and $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\mathrm{cbn}}$. □

Note that the valuation β, which is used for pattern matching in the cbn transformation, ranges over the full computation domain $\mathrm{T}^{\infty}_{\mathcal{C},\perp}$, so that function operations can be non-strict.

Definition 5.10
The *cbn fixed-point data-type* $\mathrm{DT}^{\mathrm{fix}}_{\mathrm{cbn}}(P) \in \mathrm{Int}_{\Sigma,\mathrm{cbn}}$ is defined by

$$\mathrm{DT}^{\mathrm{fix}}_{\mathrm{cbn}}(P) := \mathrm{Fix}(\Phi_{P,\mathrm{cbn}}) = \bigsqcup_{i \in \mathbb{N}} (\Phi_{P,\mathrm{cbn}})^i(\perp_{\mathrm{cbn}})$$

and the *cbn fixed-point (term) semantics* $[\![\cdot]\!]^{\mathrm{fix}}_{P,\mathrm{cbn}} : \mathrm{T}_{\Sigma} \to \mathrm{T}^{\infty}_{\mathcal{C},\perp}$ by

$$[\![t]\!]^{\mathrm{fix}}_{P,\mathrm{cbn}} := [\![t]\!]^{\mathrm{alg}}_{\mathrm{DT}^{\mathrm{fix}}_{\mathrm{cbn}}(P)}$$

□

Example 5.4 *Determining a cbn fixed-point data-type*
Taking anew the program of Example 4.1, its cbn fixed-point data-type is determined by means of the following table. Let $\mathfrak{A}_i := (\Phi_{P,\mathrm{cbn}})^i(\perp_{\mathrm{cbn}})$ for all $i \in \mathbb{N}$ and $\mathfrak{A}_{\infty} = \mathrm{DT}^{\mathrm{fix}}_{\mathrm{cbn}}(P)$.

	$i = 0$	$i = 1$	$i = 2$	$\ldots$	$i = \infty$
$\mathtt{list1}^{\mathfrak{A}_i}()$	$\perp$	$\mathtt{[]} : \perp$	$\mathtt{[]} : \mathtt{[]} : \perp$	$\ldots$	$\mathtt{[]} : \mathtt{[]} : \mathtt{[]} : \ldots$
$\mathtt{list2}^{\mathfrak{A}_i}()$	$\perp$	$\mathtt{[[]]} : \perp$	$\mathtt{[[]]} : \mathtt{[[]]} : \perp$	$\ldots$	$\mathtt{[[]]} : \mathtt{[[]]} : \mathtt{[[]]} : \ldots$
$\mathtt{head}^{\mathfrak{A}_i}$	$\underline{t} \mapsto \perp$	$\perp \mapsto \perp$ $\mathtt{[]} \mapsto \perp$ $\underline{t}_1 : \underline{t}_2 \mapsto \underline{t}_1$	as for $i = 1$	$\ldots$	as for $i = 1$

with $\underline{t}, \underline{t}_1, \underline{t}_2 \in \mathrm{T}^{\infty}_{\mathcal{C},\perp}$.

The equations $[\![\mathtt{head(list1)}]\!]^{\mathrm{fix}}_{P,\mathrm{cbn}} = \mathtt{[]}$ and $[\![\mathtt{head(list2)}]\!]^{\mathrm{fix}}_{P,\mathrm{cbn}} = \mathtt{[[]]}$ do not violate compositionality, because $[\![\mathtt{list1}]\!]^{\mathrm{fix}}_{\mathrm{P},\mathrm{cbn}}$ and $[\![\mathtt{list2}]\!]^{\mathrm{fix}}_{\mathrm{P},\mathrm{cbn}}$ differ. □

5.4 Cbn Reduction Semantics

In general the leftmost-outermost (lo) reduction strategy is associated with cbn semantics, but due to the patterns and the non-flat base data-type this strategy is not complete for our cbn semantics:

Example 5.5 *Incompleteness of lo reduction*
Considering the program

$$\begin{array}{lcl} \texttt{f(x, A)} & \to & \texttt{A} \\ \texttt{a} & \to & \texttt{A} \\ \texttt{undef} & \to & \texttt{undef} \end{array}$$

we have $[\![\texttt{f(undef,a)}]\!]^{\mathrm{fix}}_{P,\mathrm{cbn}} = \texttt{f}^{\mathrm{DT}^{\mathrm{fix}}_{\mathrm{cbn}}}(\bot, \texttt{A}) = \texttt{A}$, but

$$\texttt{f(}\underline{\texttt{undef}}\texttt{,a)} \xrightarrow[P,\mathrm{lo}]{} \texttt{f(}\underline{\texttt{undef}}\texttt{,a)} \xrightarrow[P,\mathrm{lo}]{} \ldots .$$

□

In HASKELL the patterns of function definitions are translated into `case` expressions for which lo reduction is applicable. However, such a translation does not only complicate the semantics but is unfeasable for our language, since our language permits the definition of non-sequential operations (see Sections 8 and 10). We use the parallel-outermost (po) reduction strategy.

Definition 5.11
A reduction $A = t \xrightarrow[P]{U}\!\!\!\!\rightarrow\, t'$ is a *po reduction*, written $A = t \xrightarrow[P,\mathrm{po}]{} t'$, iff U is the set of all outermost redex positions of t. This also defines the po reduction strategy $\xrightarrow[P,\mathrm{po}]{}\!\!\!\!\!\rightarrow$.
□

Example 5.6
Using the program of Example 5.5 above, we obtain

$$\texttt{f(}\underline{\texttt{undef}}\texttt{,}\underline{\texttt{a}}\texttt{)} \xrightarrow[P,\mathrm{po}]{}\!\!\!\!\rightarrow\, \underline{\texttt{f}}\texttt{(undef,A)} \xrightarrow[P,\mathrm{po}]{}\!\!\!\!\rightarrow\, \texttt{A}$$

as desired. □

Simply using the po normal-form to define the cbn reduction semantics — analogously to the cbv semantics — is not sufficient, due to the computation domain $\mathrm{T}^{\infty}_{\mathcal{C},\bot}$. Obviously an infinite constructor term can never be the result of a computation, but it can be approximated to arbitrary precision.

Definition 5.12
The algebraic term semantics with respect to the algebra $\bot_{\mathrm{cbn}}$,

$$[\![\cdot]\!]^{\mathrm{alg}}_{\bot_{\mathrm{cbn}}} : \mathrm{T}_{\Sigma} \to \mathrm{T}^{\infty}_{\mathcal{C},\bot},$$

is called *semantic cbn approximation.* □

Definition 5.13
The *po reduction* or *cbn reduction semantics* $[\![\cdot]\!]^{\mathrm{po}}_{P,\mathrm{cbn}} : \mathrm{T}_\Sigma \to \mathrm{T}^\infty_{\mathcal{C},\perp}$ is defined by

$$[\![t]\!]^{\mathrm{po}}_{P,\mathrm{cbn}} := \bigsqcup \{ [\![t']\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbn}}} \mid t \xrightarrow[P,\mathrm{po}]{*} t' \}.$$

□

Example 5.7 *Approximating a cbn reduction semantics*
Using the program of Example 5.5, we have the following approximation

$$\begin{array}{ccccccccc}
\texttt{list1} & \xrightarrow[P,\mathrm{po}]{} & \texttt{[]:list1} & \xrightarrow[P,\mathrm{po}]{} & \texttt{[]:[]:list1} & \xrightarrow[P,\mathrm{po}]{} & \texttt{[]:[]:[]:list1} & \xrightarrow[P,\mathrm{po}]{} & \cdots \\
\downarrow{\scriptstyle [\![\cdot]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbn}}}} & & \downarrow{\scriptstyle [\![\cdot]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbn}}}} & & \downarrow{\scriptstyle [\![\cdot]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbn}}}} & & \downarrow{\scriptstyle [\![\cdot]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbn}}}} & & \\
\perp & & \texttt{[]:}\perp & & \texttt{[]:[]:}\perp & & \texttt{[]:[]:[]:}\perp & & \cdots
\end{array}$$

□

The reader should notice that the semantic cbn approximation is computable, because it is characterized by

$$\begin{aligned}
[\![G(t_1,\ldots,t_n)]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbn}}} &= G([\![t_1]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbn}}},\ldots,[\![t_n]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbn}}}) \\
[\![f(t_1,\ldots,t_n)]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbn}}} &= \perp
\end{aligned}$$

for all $G^{(n)} \in \mathcal{C}$ and $f^{(n)} \in \mathcal{F}$.

Also, if for a $t \in \mathrm{T}_\Sigma$ we have $[\![t]\!]^{\mathrm{po}}_{P,\mathrm{cbn}} = \underline{t} \in \mathrm{T}_\mathcal{C}$, then the po normal-form $t{\downarrow_{P,\mathrm{po}}}$ exists and $\underline{t} = t{\downarrow_{P,\mathrm{po}}}$, as we prove in Section 8.

6 Definition of the ς-Semantics

Looking at the definitions of the cbv and the cbn fixed-point semantics the fact stands out that both definitions consist of two almost independent parts: the base data-type and the transformation. The idea to exchange these between cbv and cbn semantics immediately suggests itself. The cbn transformation can be applied to cbv interpretations with only little adaptions, likewise the cbv transformation can be applied to cbn interpretations.

The two resulting mixed semantics are not even that unusual. The first one is more closely related to the cbn semantics of recursive applicative program schemes than our cbn semantics. In the theory of recursive applicative program schemes 'cbn' refers just to the evaluation of function operations. The major part of the literature ([32, 21, 23, 10, 20]) considers only flatly ordered base data-types (interpretations), because the flat order permits simpler proofs.

The second mixture has even been implemented in versions of the functional programming language Hope ([11]), thereby combining the expressiveness of infinite data structures with the efficiency of cbv evaluation.

However, defining these additional semantics would have the unpleasant consequence that we would have to prove all properties of semantics four times, for example the equivalence of fixed-point and reduction semantics. Besides, finding reduction semantics for the two new semantics would not be easy.

The solution is a further generalization, blurring the dividing-lines. We introduce a new parameter ς which states for every argument position of every operation symbol, if the operation shall be strict at that argument position.

Definition 6.1
A mapping $\varsigma : \bigcup_{n\in\mathbb{N}} \Sigma_n \to \mathbb{B}^n$ is a *forced strictness* for the signature Σ. For $g \in \Sigma$ the boolean vector $\varsigma(g)$ is called *forced strictness* of g. The symbol $g^{(n)} \in \Sigma$ is *forcedly strict* in $\vec{\underline{t}} \in (\mathrm{T}^\infty_{\mathcal{C},\bot})^n$ iff there exists an $i \in [n]$ with $\varsigma(g)_i = \mathtt{tt}$ and $\underline{t}_i = \bot$. □

The forced strictness ς gives us an arbitrary number of semantics, depending on the signature, but they can be handled simultaneously in a simple, uniform way.

In all the following sections ς is an arbitrary forced strictness.

6.1 ς-Fixed-Point Semantics

The definition is completely analogous to that of the cbv and cbn fixed-point semantics. The exact relationship is discussed in Subsection 6.5.

Definition 6.2
The ordered algebra $\mathrm{BDT}_{\mathcal{C},\varsigma} := \mathrm{T}_{\mathcal{C},\varsigma} := \langle \mathrm{T}_{\mathcal{C},\varsigma}, \trianglelefteq, \tau\rangle$ with

- $\mathrm{T}_{\mathcal{C},\varsigma}$ being the least subset of $\mathrm{T}^\infty_{\mathcal{C},\bot}$ satisfying
  - $G^{(n)} \in \mathcal{C}$, $\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$, G not forcedly strict for $\vec{\underline{t}} \implies G(\vec{\underline{t}}) \in \mathrm{T}_{\mathcal{C},\varsigma}$ and
  - $T \subseteq \mathrm{T}_{\mathcal{C},\varsigma}$ is a chain $\implies \bigsqcup T \in \mathrm{T}_{\mathcal{C},\varsigma}$, and
- τ defined by

$$\tau(G)(\vec{\underline{t}}) = \begin{cases} G(\vec{\underline{t}}) & \text{, if } G \text{ is not forcedly strict for } \vec{\underline{t}}; \\ \bot & \text{, otherwise;} \end{cases}$$

  for all $G^{(n)} \in \mathcal{C}$, $\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$

is the ς-*base data-type* over the constructor symbols $\mathcal{C}$. □

Using simply $\mathrm{T}^\infty_{\mathcal{C},\bot}$ instead of $\mathrm{T}_{\mathcal{C},\varsigma}$ is an alternative but would disagree with our aim that the cbv semantics shall be an instance of the ς-semantics. Besides, the elements of $\mathrm{T}^\infty_{\mathcal{C},\bot} \setminus \mathrm{T}_{\mathcal{C},\varsigma}$ could never be denoted by syntactic terms.

Definition 6.3
The set $\mathrm{Int}_{\Sigma,\varsigma} := \{\mathfrak{A} \in \mathrm{Alg}^\infty_{\Sigma,\bot} \mid \mathfrak{A}|_{(\mathcal{C},\emptyset)} = \mathrm{BDT}_{\mathcal{C},\varsigma}\}$ is the set of ς-*interpretations* and $\langle \mathrm{Int}_{\Sigma,\varsigma}, \sqsubseteq\rangle$ is the canonical cpo of ς-interpretations with least element $\bot_\varsigma := \bot_{\mathrm{Int}_{\Sigma,\varsigma}}$. □

Since pattern matching is more complicated in the context of forced strictness, we define it separately.

Definition 6.4
A term tuple $\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$ is *semantically* ς-*matchable* with a redex scheme $f^{(n)}(\vec{p}) \in \mathrm{RedS}_P$ by a variable mapping $\beta : \mathrm{Var}(\vec{p}) \to \mathrm{T}_{\mathcal{C},\varsigma}$ iff

(i) f is not forcedly strict for $\vec{\underline{t}}$,

(ii) $p_1[\bot/\mathrm{Var}(p_1)] \trianglelefteq \underline{t}_1, \ldots, p_n[\bot/\mathrm{Var}(p_n)] \trianglelefteq \underline{t}_n$, and

(iii) $[\![p_1]\!]^{\mathrm{alg}}_{(\mathrm{BDT}_{\mathcal{C},\varsigma}),\beta} = \underline{t}_1, \ldots, [\![p_n]\!]^{\mathrm{alg}}_{(\mathrm{BDT}_{\mathcal{C},\varsigma}),\beta} = \underline{t}_n$.

□

The necessity of the new *order condition* (ii) is explained in the next subsection.

Definition 6.5
The ς-*transformation* of $P \in \mathrm{Prog}_\Sigma$, $\Phi_{P,\varsigma} : [\mathrm{Int}_{\Sigma,\varsigma} \to \mathrm{Int}_{\Sigma,\varsigma}]$, is defined by

$$f^{\Phi_{P,\varsigma}(\mathfrak{A})}(\vec{\underline{t}}) := \begin{cases} [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} & , \text{ if } \vec{\underline{t}} \text{ is semantically } \varsigma\text{-matchable with} \\ & \quad \text{the left-hand side of a program rule } f(\vec{p}) \to r \in P \\ & \quad \text{by a valuation } \beta : \mathrm{Var}(\vec{p}) \to \mathrm{T}_{\mathcal{C},\varsigma}; \\ \bot & , \text{ otherwise;} \end{cases}$$

for all $f^{(n)} \in \mathcal{F}$, $\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$ and $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$. □

Definition 6.6
The ς-*fixed-point data-type* $\mathrm{DT}^{\mathrm{fix}}_\varsigma(P) \in \mathrm{Int}_{\Sigma,\varsigma}$ is defined by

$$\mathrm{DT}^{\mathrm{fix}}_\varsigma(P) := \mathrm{Fix}(\Phi_{P,\varsigma}) = \bigsqcup_{i \in \mathbb{N}} (\Phi_{P,\varsigma})^i(\bot_\varsigma)$$

and the ς-*fixed-point (term) semantics* $[\![\cdot]\!]^{\mathrm{fix}}_{P,\varsigma} : \mathrm{T}_\Sigma \to \mathrm{T}_{\mathcal{C},\varsigma}$ by

$$[\![t]\!]^{\mathrm{fix}}_{P,\varsigma} := [\![t]\!]^{\mathrm{alg}}_{\mathrm{DT}^{\mathrm{fix}}_\varsigma(P)}$$

□

6.2 Well-Definedness of the ς-Fixed-Point Semantics

When proving the well-definedness of the ς-fixed-point semantics we also justify some details of the given definition which are not straightforward or where alternatives are feasible.

Lemma 6.1 *Continuity of forced strictness*
Let $g^{(n)} \in \Sigma$, $T = (\vec{\underline{t}}_j)_{j \in \mathbb{N}}$ a chain with $\underline{t}_{i,j} \in \mathrm{T}_{\mathcal{C},\varsigma}$ and $\vec{\underline{t}} = \bigsqcup T$. The symbol g is forcedly strict for $\vec{\underline{t}}$ iff g is forcedly strict for all $\vec{\underline{t}}_j \in T$.

Proof idea: Case analysis on g being forcedly strict and g not being forcedly strict for $\vec{\underline{t}}$. □

Corollary 6.2
The constructor operations of the ς-base data-type are continuous, that is $\mathrm{BDT}_{\mathcal{C},\varsigma} \in \mathrm{Alg}^\infty_{\Sigma,\bot}$. □

Lemma 6.3
The canonically ordered set of ς-interpretations $\langle \mathrm{Int}_{\Sigma,\varsigma}, \sqsubseteq \rangle$ is a cpo.

Proof idea: Every ς-interpretation is a cpo and all have the same carrier set and order. □

Now we prove that — similar to operational confluence — the function operations resulting from an application of the ς-transformation are true mappings, that is exactly one result term is respectively associated with every argument term tuple. Here the order condition of semantic ς-matching proves to be necessary. The following example illustrates this.

Example 6.1
We regard the program

$$\begin{array}{lcl} \mathtt{f(G(x))} & \to & \mathtt{A} \\ \mathtt{f(H(x))} & \to & \mathtt{B} \end{array}$$

Since

$$\mathtt{G(x)}[\bot/\mathtt{x}] = \mathtt{G}(\bot) \not\trianglelefteq \bot \quad \text{and} \quad \mathtt{H(x)}[\bot/\mathtt{x}] = \mathtt{H}(\bot) \not\trianglelefteq \bot$$

the order condition assures that $\bot$ cannot semantically ς-match neither $\mathtt{f(G(x))}$ nor $\mathtt{f(H(x))}$. However, with the forced strictness $\varsigma(\mathtt{f}) := (\mathtt{ff})$ and $\varsigma(\mathtt{G}) := \varsigma(\mathtt{H}) := (\mathtt{tt})$ and the valuation $\beta(\mathtt{x}) := \bot$ we have

$$[\![\mathtt{G(x)}]\!]^{\mathrm{alg}}_{(\mathrm{BDT}_{\mathcal{C},\varsigma}),\beta} = \mathtt{G}^{\mathrm{BDT}_{\mathcal{C},\varsigma}}(\bot) = \bot \quad \text{and} \quad [\![\mathtt{H(x)}]\!]^{\mathrm{alg}}_{(\mathrm{BDT}_{\mathcal{C},\varsigma}),\beta} = \mathtt{H}^{\mathrm{BDT}_{\mathcal{C},\varsigma}}(\bot) = \bot.$$

Therefore, without the ordering condition $\bot$ would semantically ς-match both redex schemes. We would get the contradiction

$$\mathtt{A} = [\![\mathtt{A}]\!]^{\mathrm{alg}}_{\bot_\varsigma,\beta} = \mathtt{f}^{\Phi_{P,\varsigma}(\bot_\varsigma)}(\bot) = [\![\mathtt{B}]\!]^{\mathrm{alg}}_{\bot_\varsigma,\beta} = \mathtt{B}$$

because $\mathtt{f}$ is not forcedly strict for the argument term $\bot$. □

Lemma 6.4 *Semantic unification implies syntactic unification*
Let $p, p' \in \mathrm{T}_{\mathcal{C}}(X)$ be linear patterns with $\mathrm{Var}(p) \cap \mathrm{Var}(p') = \emptyset$, $\beta : \mathrm{Var}(p) \to \mathrm{T}_{\mathcal{C},\varsigma}$, and $\beta' : \mathrm{Var}(p') \to \mathrm{T}_{\mathcal{C},\varsigma}$ valuations, and $\underline{t} \in \mathrm{T}_{\mathcal{C},\varsigma}$. If

$$p[\bot/\mathrm{Var}(p)] \trianglelefteq \underline{t}, \quad p'[\bot/\mathrm{Var}(p')] \trianglelefteq \underline{t} \quad \text{and} \quad [\![p]\!]^{\mathrm{alg}}_{(\mathrm{BDT}_{\mathcal{C},\varsigma}),\beta} = \underline{t} = [\![p']\!]^{\mathrm{alg}}_{(\mathrm{BDT}_{\mathcal{C},\varsigma}),\beta}$$

then exist two substitutions

$$\sigma : \mathrm{Var}(p) \to \mathrm{T}_{\mathcal{C}}(\mathrm{Var}(p) \,\dot\cup\, \mathrm{Var}(p')) \quad \text{and} \quad \sigma' : \mathrm{Var}(p') \to \mathrm{T}_{\mathcal{C}}(\mathrm{Var}(p) \,\dot\cup\, \mathrm{Var}(p')),$$

and a valuation

$$\hat\beta : (\mathrm{Var}(p) \,\dot\cup\, \mathrm{Var}(p')) \to \mathrm{T}_{\mathcal{C},\varsigma}$$

so that

$$\begin{array}{rcl} \forall x \in \mathrm{Var}(p).\ \beta(x) & = & [\![x\sigma]\!]^{\mathrm{alg}}_{(\mathrm{BDT}_{\mathcal{C},\varsigma}),\hat\beta} \\ \forall x \in \mathrm{Var}(p').\ \beta'(x) & = & [\![x\sigma']\!]^{\mathrm{alg}}_{(\mathrm{BDT}_{\mathcal{C},\varsigma}),\hat\beta} \\ p\sigma & = & p'\sigma'. \end{array}$$

Proof idea: Parallel structural induction on p and p'. □

Corollary 6.5
The function operations resulting from a ς-transformation are well-defined, that is if $\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$ with $n \in \mathbb{N}$ and $\vec{\underline{t}}$ is semantically ς-matchable with the left-hand side of a reduction rule $f^{(n)}(\vec{p}) \to r \in P$ by $\beta : \mathrm{Var}(\vec{p}) \to \mathrm{T}_{\mathcal{C},\varsigma}$ and with the left-hand side of a reduction rule $f^{(n)}(\vec{p}') \to r' \in P$ by $\beta' : \mathrm{Var}(\vec{p}') \to \mathrm{T}_{\mathcal{C},\varsigma}$, then $[\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = [\![r']\!]^{\mathrm{alg}}_{\mathfrak{A},\beta'}$ for all $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$. □

Finally, we have to prove that the algebras resulting from a ς-transformation are continuous, that is they are ς-interpretations, and that the ς-transformation is continuous. First we show the continuity of semantic ς-matching.

Lemma 6.6 *Characterization of semantic ς-matching*
The term tuple $\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$ is semantically ς-matchable with $f(\vec{p}) \in \mathrm{RedS}_P$ by $\beta : \mathrm{Var}(\vec{p}) \to \mathrm{T}_{\mathcal{C},\varsigma}$ iff conditions (i) and (ii) of semantic ς-matchability (Def. 6.4) are fulfilled and $\beta(x) = \vec{\underline{t}}/u$ with u given by $\{u\} = \mathrm{Pos}(x, \vec{p})$ for all $x \in \mathrm{Var}(\vec{p})$.
Proof idea: Structural induction on the patterns $\vec{p}$. □

The reader should notice that u is well-defined only because patterns are linear.

Lemma 6.7 *Continuity of semantic ς-matching*
Let $f^{(n)}(\vec{p}) \in \mathrm{RedS}_P$, $T = (\vec{\underline{t}}_j)_{j \in \mathbb{N}}$ a chain with $\underline{t}_{i,j} \in \mathrm{T}_{\mathcal{C},\varsigma}$, and $\vec{\underline{s}} := \bigsqcup T$.

- The term tuple $\vec{\underline{s}}$ is semantically ς-matchable with $f(\vec{p})$ iff a $\underline{t}_k \in T$ exists which is semantically ς-matchable with $f(\vec{p})$.
- If all $\vec{\underline{t}}_j \in T$ are semantically ς-matchable with $f(\vec{p})$ by corresponding $\beta_j : \mathrm{Var}(\vec{p}) \to \mathrm{T}_{\mathcal{C},\varsigma}$then $\vec{\underline{s}}$ is semantically ς-matchable with $f(\vec{p})$ by $\beta = \bigsqcup_{j \in \mathbb{N}} \beta_j$.

Proof idea: Straightforward, using the previous lemma. □

The following two rather general lemmas are the basis for the last two lemmas of this subsection.

Lemma 6.8 *Continuity of the algebraic term semantics w.r.t. the valuation*
Let Σ be a signature, $\langle A, \leq \rangle$ a cpo, $\mathfrak{A} \in \mathrm{Alg}^{\infty}_{\Sigma,\perp}(\langle A, \leq \rangle)$ and $t \in \mathrm{T}_\Sigma(X)$. Then the algebraic term semantics $[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A},\cdot} : (X \to A) \to A$ is continuous w.r.t the canonical (pointwise) cpo on valuations $\langle (X \to A), \leq \rangle$.
Proof idea: Structural induction on t. □

Lemma 6.9 *Continuity of the algebraic term semantics w.r.t the algebra*
Let Σ be a signature and $\langle A, \leq \rangle$ a cpo. Let $I \subseteq \mathrm{Alg}^{\infty}_{\Sigma,\perp}(\langle A, \leq \rangle)$ such that the canonical partial order $\langle I, \sqsubseteq \rangle$ is continuous. Let $\beta : X \to A$ and $t \in \mathrm{T}_\Sigma(X)$. Then the algebraic term semantics $[\![t]\!]^{\mathrm{alg}}_{\cdot,\beta} : I \to A$ is continuous w.r.t. the cpo of algebras $\langle I, \sqsubseteq \rangle$.
Proof idea: Structural induction on t. □

Lemma 6.10 *Continuity of the results of a ς-transformation*
$\Phi_{P,\varsigma}(\mathfrak{A}) \in \mathrm{Int}_{\Sigma,\varsigma}$ for all $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$.
Proof idea: Show continuity of the function operations of $\Phi_{P,\varsigma}(\mathfrak{A})$ by using Lemma 6.8. □

Lemma 6.11 *Continuity of the ς-transformation*
The ς-transformation is continuous, that is $\Phi_{P,\varsigma} : [\mathrm{Int}_{\Sigma,\varsigma} \to \mathrm{Int}_{\Sigma,\varsigma}]$.

Proof idea: Show that $f^{\bigsqcup \Phi_{P,\varsigma}(T)}(\vec{\underline{t}})$ exists and $f^{\bigsqcup \Phi_{P,\varsigma}(T)}(\vec{\underline{t}}) = f^{\Phi_{P,\varsigma}(\bigsqcup T)}(\vec{\underline{t}})$ for all $f^{(n)} \in \mathcal{F}$, $\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$, and chains $T = (\mathfrak{A}_j)_{j\in\mathbb{N}} \subseteq \mathrm{Int}_{\Sigma,\varsigma}$ using Lemma 6.9. □

Looking at the definition of the ς-transformation the question arises, why we set the result of a function operation to $\bot$, when the argument terms are not semantically ς-matchable with any left-hand side. Simply leaving this value unchanged is an obvious alternative. We define $\Phi^*_{P,\varsigma}$ by

$$f^{\Phi^*_{P,\varsigma}(\mathfrak{A})}(\vec{\underline{t}}) := \begin{cases} [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} & , \text{ if } \vec{\underline{t}} \text{ is semantically } \varsigma\text{-matchable with} \\ & \quad \text{the left-hand side of a program rule } f(\vec{p}) \to r \in P \\ & \quad \text{by a valuation } \beta : \mathrm{Var}(\vec{p}) \to \mathrm{T}_{\mathcal{C},\varsigma}; \\ f^{\mathfrak{A}}(\vec{\underline{t}}) & , \text{ otherwise.} \end{cases}$$

It is easy to prove that $\bigsqcup \Phi^*_{P,\varsigma}(\bot_\varsigma)$ exists and even equals the least fixed-point of $\Phi_{P,\varsigma}$. However, $\Phi^*_{P,\varsigma}$ does not map ς-interpretations to ς-interpretations; the resulting algebra may be more general. To remedy this, we can define $\mathrm{Int}^*_{\Sigma,\varsigma}$ just like $\mathrm{Int}_{\Sigma,\varsigma}$, with the exception that operations do not need to be continuous. Using $\mathrm{Int}^*_{\Sigma,\varsigma}$ as domain and range of $\Phi^*_{P,\varsigma}$ solves the problem, but then $\Phi^*_{P,\varsigma}$ is no longer continuous and we cannot apply the fixed-point theorem of Tarski. Actually $\bigsqcup \Phi^*_{P,\varsigma}(\bot_\varsigma) = \mathrm{Fix}(\Phi_{P,\varsigma})$ exists as mentioned and can even be proved to be the least fixed-point of $\Phi^*_{P,\varsigma}$, but simple proofs of this repeatedly refer to $\Phi_{P,\varsigma}$, so that we better persevere with it.

6.3 ς-Reduction Semantics

For the cbv and the cbn semantics we were able to fall back on the well-known innermost and outermost reduction strategies. However, ς-semantics mixes the two argument evaluation mechanisms and hence there is no obvious suitable reduction strategy. Especially the variable strictness of the constructor operations are irritating, because there are no reduction rules for constructor symbols.

We approach the problem by examining which kinds of reductions are sound w.r.t. the ς-fixed-point semantics.

Definition 6.7
A *reduction* $t \xrightarrow[l\to r]{u} t'$ *is sound* w.r.t. a term semantics $[\![\cdot]\!] : \mathrm{T}_\Sigma \to A$ iff $[\![t]\!] = [\![t']\!]$. □

Example 6.2
Using the program

```
positive(Succ(x))  →  Succ(Zero)
inf                →  Succ(inf)
```

the outermost reduction $\underline{\texttt{positive}}\texttt{(Succ(inf))} \longrightarrow \texttt{Succ(Zero)}$ is *not* sound w.r.t. the cbv fixed-point semantics: $[\![\texttt{positive(Succ(inf))}]\!]^{\mathrm{fix}}_{P,\mathrm{cbv}} = \bot \neq \texttt{Succ(Zero)} = [\![\texttt{Succ(Zero)}]\!]^{\mathrm{fix}}_{P,\mathrm{cbv}}$. □

The example illustrates what can easily be gained from the definition of the ς-transformation: Only forced strictness may cause a reduction to be unsound w.r.t. a ς-fixed-point semantics.

Obviously, reducing a redex $f(\vec{\underline{t}})$ is sound if $[\![t_i]\!]^{\mathrm{fix}}_{P,\sigma} \neq \bot$ for all forcedly strict argument positions i. Naturally this condition is sufficient but not necessary; the instantiated right-hand side of a program rule may have the value $\bot$ as well and for general reductions the context of the redex is moreover relevant. For our purposes considering only redexes suffices.

Unfortunately $[\![t_i]\!]^{\mathrm{fix}}_{P,\sigma} \neq \bot$ cannot be decided and therefore cannot be used for defining a sound reduction strategy. However, a sufficient approximation of the value $[\![t_i]\!]^{\mathrm{fix}}_{P,\sigma}$ is easy to find:

Definition 6.8
The algebraic term semantics with respect to the algebra $\bot_\varsigma$, $[\![\cdot]\!]^{\mathrm{alg}}_{\bot_\varsigma} : \mathrm{T}_\Sigma \to \mathrm{T}_{\mathcal{C},\varsigma}$, is called *semantic ς-approximation*. □

Since the semantic ς-approximation has a purely syntactic definition (cf. the cbn approximation in Subsection 5.4), it can be employed in an operational semantics.

Due to the monotonicity of the algebraic term semantics w.r.t. the algebra in accordance with Lemma 6.9, $\underline{t} \trianglelefteq [\![t]\!]^{\mathrm{alg}}_{\bot_\varsigma}$ implies $t \trianglelefteq [\![t]\!]^{\mathrm{fix}}_{P,\varsigma}$. Hence we can define a meaningful notion of syntactic ς-matching in analogy to semantic ς-matching (Def. 6.4). Using that, we define the set of ς-redexes, whose reduction is sound w.r.t. the ς-fixed-point semantics. This soundness is proved in Subsection 7.2.

Definition 6.9
A term tuple $\vec{t} \in (\mathrm{T}_\Sigma)^n$ is *syntactically ς-matchable* with a redex scheme $f^{(n)}(\vec{p}) \in \mathrm{RedS}_P$ by a substitution $\sigma : \mathrm{Var}(\vec{p}) \to \mathrm{T}_{\mathcal{C},\varsigma}$ iff

(i) f is not forcedly strict for $([\![t_1]\!]^{\mathrm{alg}}_{\bot_\varsigma} \ldots [\![t_n]\!]^{\mathrm{alg}}_{\bot_\varsigma})$,

(ii) $p_1[\bot/\mathrm{Var}(p_1)] \trianglelefteq [\![t_1]\!]^{\mathrm{alg}}_{\bot_\varsigma}, \ldots, p_n[\bot/\mathrm{Var}(p_n)] \trianglelefteq [\![t_n]\!]^{\mathrm{alg}}_{\bot_\varsigma}$, and

(iii) $f(\vec{t}) = f(\vec{p})\sigma$.

□

Definition 6.10 (Cf. V- and N-redexes in [7])
A term $f(\vec{t}) \in \mathrm{T}_\Sigma$ is a *ς-redex* of a redex scheme $f(\vec{p}) \in \mathrm{RedS}_P$ iff $\vec{t}$ is syntactically ς-matchable with $f(\vec{p})$ by some substitution σ. The set of all ς-redexes of a program P is denoted by $\mathrm{Red}_{P,\varsigma}$. □

The set of ς-redexes defines an instance of a program viewed as TRS. This instance is denoted by ς as well. Thus we obtain notions like ς-redex position, ς-reduction and ς-reduction relation.

We employ these for a very simple definition of ς-reduction semantics. We define our ς-reduction semantics like the po reduction semantics as least upper bound of results of finite reduction sequences. Only, instead of po reduction sequences all ς-reduction sequences are considered.

Definition 6.11
The *global* ς*-reduction semantics*, $[\![\cdot]\!]^{\mathrm{red}}_{P,\varsigma} : \mathrm{T}_\Sigma \to \mathrm{T}_{\mathcal{C},\varsigma}$, is defined by

$$[\![t]\!]^{\mathrm{red}}_{P,\varsigma} := \bigsqcup \{ [\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma} \mid t \xrightarrow[P,\varsigma]{*} t' \}.$$

□

The global ς-reduction semantics will be of central importance in the equivalence proofs of the next section. However, it is not suitable for practical purposes, since it does not give a deterministic reduction strategy.

Example 6.3 *Non-determinism*
Taking the program of Example 4.1 we compute the global cbn semantics of `head(list1)`:

$$\texttt{head(list1)} \xrightarrow[P,\ \mathrm{cbn}]{} \texttt{head([]:list1)} \xrightarrow[P,\ \mathrm{cbn}]{} \texttt{head([]:[]:list1)} \xrightarrow[P,\ \mathrm{cbn}]{} \cdots$$
$$\qquad\qquad\qquad\qquad\quad \Big\downarrow{\scriptstyle P,\mathrm{cbn}} \qquad\qquad\qquad\quad \Big\downarrow{\scriptstyle P,\mathrm{cbn}}$$
$$\qquad\qquad\qquad\qquad\quad \texttt{[]} \qquad\qquad\qquad\quad \texttt{[]}$$

□

Since efficient implementations require deterministic reduction strategies, we define a generalization of the po reduction semantics.

Definition 6.12
The *po* ς*-reduction semantics*, $[\![\cdot]\!]^{\mathrm{po}}_{P,\varsigma} : \mathrm{T}_\Sigma \to \mathrm{T}_{\mathcal{C},\varsigma}$, is defined by

$$[\![t]\!]^{\mathrm{po}}_{P,\varsigma} := \bigsqcup \{ [\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma} \mid t \xrightarrow[P,\varsigma,\mathrm{po}]{*} t' \}.$$

□

6.4 Connection with the Previously Defined Cbv and Cbn Semantics

By a simple trick of notation we can make fully explicit that the cbv and cbn semantics defined in the previous section are special instances of the ς-semantics.

Definition 6.13
The forced strictnesses $\mathrm{cbv}, \mathrm{cbn} \in \bigcup_{n\in\mathbb{N}} \Sigma_n \to \mathbb{B}^n$ are defined by

$$\mathrm{cbv}(g)_i := \mathsf{tt} \quad \text{and} \quad \mathrm{cbn}(g)_i := \mathsf{ff}$$

for all $g^{(n)} \in \Sigma$ and $i \in [n]$. □

For the fixed-point semantics it is quite clear that the definitions of Section 5 are special instances of those given for the ς-semantics, for instance the definition of the ς-base data-type gives $\mathrm{BDT}_{\mathcal{C},\mathrm{cbv}} = \mathcal{T}_{\mathcal{C}}^{\perp}$ and $\mathrm{BDT}_{\mathcal{C},\mathrm{cbn}} = \mathcal{T}_{\mathcal{C},\perp}^{\infty}$. Just semantic cbv and cbn matching was not defined in Section 5. Nonetheless it is simple to prove that in the special cases of $\varsigma = \mathrm{cbv}$ and $\varsigma = \mathrm{cbn}$ the ς-matching conditions (i) and (iii) imply the order condition (ii), and that therefore semantic cbv/cbn matching is equivalent to the conditions given in the definition of the cbv/cbn transformation in Section 5.

The case of the reduction semantics is slightly more complicated. Obviously all redexes are cbn redexes ($\mathrm{Red}_{P,\mathrm{cbn}} = \mathrm{Red}_P$) and therefore the po reduction semantics and the po cbn reduction semantics are identical.

There is no similar correspondence for the cbv semantics. Cbv redexes, outermost cbv redexes, and innermost* redexes are the same. Also, using normal-forms for defining the li* reduction semantics does not matter since

$$\bigsqcup\{[\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma} \mid t \xrightarrow[P,\varsigma,\mathrm{po}]{*} t'\} = \begin{cases} t{\downarrow_{P,\varsigma,\mathrm{po}}} & \text{, if } t{\downarrow_{P,\varsigma,\mathrm{po}}} \text{ exists and } t{\downarrow_{P,\varsigma,\mathrm{po}}} \in \mathrm{T}_{\mathcal{C},\varsigma}; \\ \bot & \text{, otherwise;} \end{cases}$$

for all flatly ordered ς-base data-types $\mathrm{BDT}_{\mathcal{C},\varsigma}$. However, the li* reduction semantics reduces only the leftmost of the generally several redexes which are reduced in parallel by the po cbv reduction semantics. Consequently, we call the li* reduction also lo cbv reduction. A lo ς-reduction semantics is not defined, because in general it is not complete as was shown for $\varsigma = \mathrm{cbn}$ in Example 5.5. However, for $\varsigma = \mathrm{cbv}$ it is, and we prove this in Subsection 7.7.

From the perspective of the ς-semantics we see that the difference between the reduction semantics of the cbv and of the cbn semantics is only seemingly based on the positions of redexes. Instead, the fundamental difference is the kind of redexes which are reduced.

6.5 Well-Definedness of the ς-Reduction Semantics

We start with a property needed in the proof of the subsequent lemma.

Lemma 6.12 *Commutativity of semantic approximation and replacement of subterms*

$$[\![t[u \leftarrow t']]\!]^{\mathrm{alg}}_{\bot_\varsigma} = [\![t]\!]^{\mathrm{alg}}_{\bot_\varsigma}[u \leftarrow [\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma}]$$

for all $t, t' \in \mathrm{T}_\Sigma$ and $u \in \mathrm{Pos}(t)$. Note that $[\![t]\!]^{\mathrm{alg}}_{\bot_\varsigma}[u \leftarrow [\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma}] = [\![t]\!]^{\mathrm{alg}}_{\bot_\varsigma}$, if $u \notin \mathrm{Pos}([\![t]\!]^{\mathrm{alg}}_{\bot_\varsigma})$.
Proof idea: Structural induction on u. □

Besides proving the well-definedness of the ς-reduction semantics, the next two lemmas are of major importance for the ς-semantics.

Already in Subsection 5.4 we employed repeated reduction to approximate the semantic value of a term. The next lemma proves that reduction can only lead to a gain of information (w.r.t the cpo $\langle \mathrm{T}_{\mathcal{C},\varsigma}, \trianglelefteq \rangle$) and never to a loss of it.

Lemma 6.13 *Gain of information by reduction*

$$\begin{array}{rcl} t \xrightarrow[P]{} t' & \Longrightarrow & [\![t]\!]^{\mathrm{alg}}_{\bot_\varsigma} \trianglelefteq [\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma} \\ t \xrightarrow[P,\mathrm{no}]{} t' & \Longrightarrow & [\![t]\!]^{\mathrm{alg}}_{\bot_\varsigma} = [\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma} \end{array}$$

Proof: The existence of a reduction $t \xrightarrow[f(\vec{p})\longrightarrow r]{u} t'$ implies $t/u = f(\vec{p})\sigma$ and $t'/u = r\sigma$ for some substitution σ. We have $[\![f(\vec{p})\sigma]\!]^{\mathrm{alg}}_{\bot_\varsigma} = \bot_\varsigma \trianglelefteq [\![r\sigma]\!]^{\mathrm{alg}}_{\bot_\varsigma}$. Using the previous lemma we obtain

$$\begin{array}{rcccl} [\![t]\!]^{\mathrm{alg}}_{\bot_\varsigma} & = & [\![t[u \leftarrow f(\vec{p})\sigma]]\!]^{\mathrm{alg}}_{\bot_\varsigma} & = & [\![t]\!]^{\mathrm{alg}}_{\bot_\varsigma}[u \leftarrow [\![f(\vec{p})\sigma]\!]^{\mathrm{alg}}_{\bot_\varsigma}] \\ & \trianglelefteq & [\![t]\!]^{\mathrm{alg}}_{\bot_\varsigma}[u \leftarrow [\![r\sigma]\!]^{\mathrm{alg}}_{\bot_\varsigma}] & = & [\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma}. \end{array}$$

If $u \notin \mathrm{Outer}_P(t)$, then there is a position $v < u$ with $t(v) \in \mathcal{F}$. Hence $[\![t/v]\!]^{\mathrm{alg}}_{\perp_\varsigma} = \perp_\varsigma$ and therefore $u \notin \mathrm{Pos}([\![t]\!]^{\mathrm{alg}}_{\perp_\varsigma})$. This implies

$$[\![t]\!]^{\mathrm{alg}}_{\perp_\varsigma} = [\![t]\!]^{\mathrm{alg}}_{\perp_\varsigma}[u \leftarrow [\![f(\vec{p})\sigma]\!]^{\mathrm{alg}}_{\perp_\varsigma}] = [\![t]\!]^{\mathrm{alg}}_{\perp_\varsigma}[u \leftarrow [\![r\sigma]\!]^{\mathrm{alg}}_{\perp_\varsigma}] = [\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma}.$$

□

This lemma is also valid for simple and parallel ς-reduction, since for example $t \xrightarrow[P,\varsigma,\mathrm{no}]{} t'$ implies $t \xrightarrow[P,\mathrm{no}]{} t'$.

Lemma 6.14
The set of ς-redexes $\mathrm{Red}_{P,\varsigma}$ is residually closed.

Proof: Let $t \xrightarrow{u} t'$ be a reduction and $v \in \mathrm{RedPos}_{P,\varsigma}(t)$. Due to Lemma 2.4 we have $v \setminus t \xrightarrow{u} t' \subseteq \mathrm{RedPos}_P(t')$.

$\underline{v \parallel u, u \leq v}$: $t/v = t/\hat{v}$ for all $\hat{v} \in v \setminus t \xrightarrow{u} t'$ and hence $v \setminus t \xrightarrow{u} t' \subseteq \mathrm{RedPos}_{P,\varsigma}(t')$.

$\underline{v < u}$: Then $u = v.k.u'$ for some $k \in \mathbb{N}_+$ and $u' \in \mathbb{N}^*_+$. Since $v \in \mathrm{RedPos}_{P,\varsigma}(t)$, we have $t/v = f(\vec{t})$ and $t'/v = f(\vec{t'})$ with $t_k \xrightarrow{u'} t'_k$ and $t_i = t'_i$ for all other $i \in [n]$. Hence we can deduce from $\vec{t}$ being syntactically ς-matchable with some $f(\vec{p}) \in \mathrm{RedS}_P$ that $\vec{t'}$ is syntactically ς-matchable with $f(\vec{p})$ as well. Therefore $v \setminus t \xrightarrow{u} t' = \{v\} \subseteq \mathrm{RedPos}_{P,\varsigma}(t')$.

□

According to Section 2 this residual closure implies the confluence of the ς-reduction relations $\xrightarrow[P,\varsigma]{}$ and $\xrightarrow[P,\varsigma]{}\!\!\!\!\!\rightarrow$.

Lemma 6.15 *Well-definedness of the ς-reduction semantics*
The sets $T_{\mathrm{red}} := \{[\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma} \mid t \xrightarrow[P,\varsigma]{*} t'\}$ and $T_{\mathrm{po}} := \{[\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma} \mid t \xrightarrow[P,\varsigma,\mathrm{po}]{*} t'\}$ have respective least upper bounds for all $t \in \mathrm{T}_\Sigma$.

Proof: Since T_Σ is countable, $T_{\mathrm{red}}, T_{\mathrm{po}} \subseteq \{[\![t]\!]^{\mathrm{alg}}_{\perp_\varsigma} \mid t \in \mathrm{T}_\Sigma\}$ are countable as well.

$\underline{T_{\mathrm{red}}}$: Let $[\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma}, [\![t'']\!]^{\mathrm{alg}}_{\perp_\varsigma} \in T_{\mathrm{red}}$. Consequently $t \xrightarrow[P,\varsigma]{*} t'$ and $t \xrightarrow[P,\varsigma]{*} t''$. Due to the confluence of the ς-reduction relation there is a $\hat{t} \in \mathrm{T}_\Sigma$ with $t' \xrightarrow[P,\varsigma]{*} \hat{t}$ and $t'' \xrightarrow[P,\varsigma]{*} \hat{t}$. Together with Lemma 6.13 about the gain of information by reduction we get $[\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma} \trianglelefteq [\![\hat{t}]\!]^{\mathrm{alg}}_{\perp_\varsigma}$ and $[\![t'']\!]^{\mathrm{alg}}_{\perp_\varsigma} \trianglelefteq [\![\hat{t}]\!]^{\mathrm{alg}}_{\perp_\varsigma}$. Hence T_{red} is directed.

$\underline{T_{\mathrm{po}}}$: From the lemma about gain of information we deduce immediately that T_{po} is even a chain.

Since T_{red} and T_{po} are countable directed subsets of $\mathrm{T}_{\mathcal{C},\varsigma}$, their respective least upper bounds exist. □

7 Equivalence of the Definitions of the ς-Semantics

We prove in this section that the three definitions of the ς-semantics, that is those of the ς-fixed-point semantics, the global ς-reduction semantics, and the po ς-reduction semantics, are equivalent. We do this by proving separately soundness and completeness.

Definition 7.1
A term semantics $[\![\cdot]\!] : \mathrm{T}_\Sigma \to \mathrm{T}_{\mathcal{C},\varsigma}$ is *sound* w.r.t. another term semantics $[\![\cdot]\!]' : \mathrm{T}_\Sigma \to \mathrm{T}_{\mathcal{C},\varsigma}$ iff $[\![\cdot]\!] \preceq [\![\cdot]\!]'$ and *complete* iff $[\![\cdot]\!]' \preceq [\![\cdot]\!]$. □

Subsections 7.1 and 7.2 establish basic properties which are used subsequently. Afterwards, the soundness of the two ς-reduction semantics w.r.t. the ς-fixed-point semantics and the soundness of the po w.r.t. the global ς-reduction semantics are shown in Subsection 7.3. Subsection 7.4 contains the proof of completeness of the global ς-reduction semantics w.r.t. the ς-fixed-point semantics. In Subsection 7.5 we give a comprehensive method for proving the completeness of any reduction semantics based on a so called Π-fair reduction strategy w.r.t. a global reduction semantics. We apply this in Subsections 7.6 and 7.7 to show respectively the completeness of the po w.r.t. the global ς-reduction semantics and that of the li* w.r.t. the global cbv reduction semantics. The overall result of this section is:

Proposition 7.1 *Equivalence of the definitions of the ς-semantics*

$$[\![\cdot]\!]^{\mathrm{fix}}_{P,\varsigma} = [\![\cdot]\!]^{\mathrm{red}}_{P,\varsigma} = [\![\cdot]\!]^{\mathrm{po}}_{P,\varsigma}.$$

Proof: $[\![\cdot]\!]^{\mathrm{red}}_{P,\varsigma} \preceq [\![\cdot]\!]^{\mathrm{fix}}_{P,\varsigma}$ and $[\![\cdot]\!]^{\mathrm{po}}_{P,\varsigma} \preceq [\![\cdot]\!]^{\mathrm{fix}}_{P,\varsigma}$ proved by Lemma 7.8.
$[\![\cdot]\!]^{\mathrm{fix}}_{P,\varsigma} \preceq [\![\cdot]\!]^{\mathrm{red}}_{P,\varsigma}$ proved by Lemma 7.10.
$[\![\cdot]\!]^{\mathrm{red}}_{P,\varsigma} \preceq [\![\cdot]\!]^{\mathrm{po}}_{P,\varsigma}$ proved by Lemma 7.19. □

Hence we can define:

Definition 7.2
The ς-interpretation $\mathrm{DT}_\varsigma(P) := \mathrm{DT}^{\mathrm{fix}}_\varsigma(P)$ is called ς-*data-type* and the mapping $[\![\cdot]\!]_{P,\varsigma} := [\![\cdot]\!]^{\mathrm{fix}}_{P,\varsigma} = [\![\cdot]\!]^{\mathrm{red}}_{P,\varsigma} = [\![\cdot]\!]^{\mathrm{po}}_{P,\varsigma}$ is called ς-*(term) semantics*. □

7.1 Syntactic and Semantic ς-Matching

First, we need to prove some basic properties.

Lemma 7.2 *Characterization of syntactic ς-matching* (cf. Lem. 6.6)
The term tuple $\vec{t} \in (\mathrm{T}_\Sigma)^n$ is syntactically ς-matchable with $f(\vec{p}) \in \mathrm{RedS}_P$ by a substitution $\sigma : \mathrm{Var}(\vec{p}) \to \mathrm{T}_\Sigma$ iff conditions (i) and (ii) of semantic ς-matchability (Def. 6.4) are fulfilled for the semantically approximated term tuple $([\![t_1]\!]^{\mathrm{alg}}_{\perp_\varsigma}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\perp_\varsigma})$ and $x\sigma = \vec{t}/u_x$ with u_x given by $\{u_x\} := \mathrm{Pos}(x, \vec{p})$ for all $x \in \mathrm{Var}(\vec{p})$.
Proof idea: Structural induction on $\vec{p}$. □

The reader should notice again that the linearity of patterns is crucial for the well-definedness of u_x.

Lemma 7.3 *Commutativity of term semantics and forming of subterms*
Let $c \in \mathrm{T}_\mathcal{C}(X)$, $\beta : \mathrm{Var}(c) \to \mathrm{T}_{\mathcal{C},\varsigma}$, and $u \in \mathrm{Pos}(c) \cap \mathrm{Pos}([\![c]\!]^{\mathrm{alg}}_{(\mathrm{BDT}_{\mathcal{C},\varsigma}),\beta})$. Then

$$[\![c]\!]^{\mathrm{alg}}_{(\mathrm{BDT}_{\mathcal{C},\varsigma}),\beta}/u = [\![c/u]\!]^{\mathrm{alg}}_{(\mathrm{BDT}_{\mathcal{C},\varsigma}),\beta}.$$

Proof idea: Structural induction on u. □

The next proposition is of fundamental importance for the equality of the ς-fixed-point and the ς-reduction semantics.

Proposition 7.4 *Syntactic and semantic ς-matching*
Let $f^{(n)}(\vec{p}) \in \mathrm{RedS}_P$ and $\vec{t} \in (\mathrm{T}_\Sigma)^n$.

1. The following three statements are equivalent.
   a) $\vec{t}$ is syntactically ς-matchable with $f(\vec{p})$ (by a substitution $\sigma : X \to \mathrm{T}_\Sigma$).
   b) $([\![t_1]\!]^{\mathrm{alg}}_{\perp_\varsigma}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\perp_\varsigma})$ is semantically ς-matchable with $f(\vec{p})$.
   c) For all interpretations $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$ the tuple $([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A}}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A}})$ is semantically ς-matchable with $f(\vec{p})$ (by a respective valuation $\beta_\mathfrak{A} : X \to \mathrm{T}_{\mathcal{C},\varsigma}$).
2. If the statements above are fulfilled, then $\beta_\mathfrak{A}(x) = [\![x\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}$ for all $x \in \mathrm{Var}(\vec{p})$ and even $[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta_\mathfrak{A}} = [\![t\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}$ for all $t \in \mathrm{T}_\Sigma(\mathrm{Var}(\vec{p}))$.

Proof idea:

1. a) $\Longleftrightarrow$ b) follows from the characterization of syntactic and semantic ς-matching (Lemmas 7.2 and 6.6). b) $\Longrightarrow$ c) employs the monotonicity of semantics ς-matching which is valid according to Lemma 6.7 and c) $\Longrightarrow$ b) is trivial.
2. Follows from Lemmas 7.2 and 6.6 as well, using the previous lemma about commutativity.

□

The reader should note that $\vec{t}$ is not necessarily syntactically ς-matchable with $f(\vec{p})$, if $([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A}}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A}})$ is only semantically ς-matchable with $f(\vec{p})$ for some $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$.

7.2 Soundness of ς-Reduction

Here we prove in several steps the soundness of ς-reduction w.r.t. the ς-fixed-point semantics which was already claimed in Subsection 6.3.

Lemma 7.5
In fixed-points of the ς-transformation simple ς-reduction is sound, that is if $f(\vec{p}) \to r \in P$, $\sigma : X \to \mathrm{T}_\Sigma$, $\mathfrak{A} = \Phi_{P,\varsigma}(\mathfrak{A}) \in \mathrm{Int}_{\Sigma,\varsigma}$, and $f(\vec{p})\sigma$ is a ς-redex, then:

$$[\![f(\vec{p})\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}} = [\![r\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}.$$

Proof: Since $f(\vec{p})\sigma$ is a ς-redex, the term tuple $(p_1\sigma, \ldots, p_n\sigma)$ is syntactically ς-matchable with $f(\vec{p})$ by σ.

Due to Proposition 7.4 about syntactic and semantic ς-matching we know that $([\![p_1\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}, \ldots, [\![p_n\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}})$ is semantically ς-matchable with $f(\vec{p})$ by $\beta_\mathfrak{A} : X \to \mathrm{T}_{\mathcal{C},\varsigma}$, which is defined by $\beta_\mathfrak{A}(x) := [\![x\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}$ for all $x \in X$; hence

$$f^{\Phi_{P,\varsigma}(\mathfrak{A})}([\![p_1\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}, \ldots, [\![p_n\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}) = [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta_\mathfrak{A}} \tag{1}$$

and also (Prop. 7.4, item 2):

$$[\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta_\mathfrak{A}} = [\![r\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}. \tag{2}$$

Since $\mathfrak{A}$ is a fixed-point of $\Phi_{P,\varsigma}$, we have

$$[\![f(\vec{p})\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}} = f^{\mathfrak{A}}([\![p_1\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}, \ldots, [\![p_n\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}) = f^{\Phi_{P,\varsigma}(\mathfrak{A})}([\![p_1\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}, \ldots, [\![p_n\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}). \qquad (3)$$

Altogether:

$$[\![f(\vec{p})\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}} \overset{(3)}{=} f^{\Phi_{P,\varsigma}(\mathfrak{A})}([\![p_1\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}, \ldots, [\![p_n\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}) \overset{(1)}{=} [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta_{\mathfrak{A}}} \overset{(2)}{=} [\![r\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}.$$

□

Corollary 7.6
In fixed-points of the ς-transformation ς-reduction is sound, that is if $\mathfrak{A} = \Phi_{P,\varsigma}(\mathfrak{A}) \in \mathrm{Int}_{\Sigma,\varsigma}$, then for all $t, t' \in \mathrm{T}_\Sigma$:

$$t \xrightarrow[P,\varsigma]{*} t' \implies [\![t]\!]^{\mathrm{alg}}_{\mathfrak{A}} = [\![t']\!]^{\mathrm{alg}}_{\mathfrak{A}}.$$

Proof: Follows from the preceding lemma by the invariance of algebraic term semantics.
□

Corollary 7.7
ς-reduction is sound w.r.t. the ς-fixed-point semantics, that is

$$t \xrightarrow[P,\varsigma]{*} t' \implies [\![t]\!]^{\mathrm{fix}}_{P,\varsigma} = [\![t']\!]^{\mathrm{fix}}_{P,\varsigma}.$$

□

7.3 Soundness of the ς-Reduction Semantics

Lemma 7.8
The ς-reduction semantics are sound w.r.t. the ς-fixed-point semantics:

$$[\![\cdot]\!]^{\mathrm{red}}_{P,\varsigma} \preceq [\![\cdot]\!]^{\mathrm{fix}}_{P,\varsigma} \quad \text{and} \quad [\![\cdot]\!]^{\mathrm{po}}_{P,\varsigma} \preceq [\![\cdot]\!]^{\mathrm{fix}}_{P,\varsigma}.$$

Proof: Let $t, t' \in \mathrm{T}_\Sigma$ with $t \xrightarrow[P,\varsigma]{*} t'$. According to the soundness of ς-reduction sequences (Cor. 7.7) $[\![t]\!]^{\mathrm{fix}}_{P,\varsigma} = [\![t']\!]^{\mathrm{fix}}_{P,\varsigma}$. Since $\bot_\varsigma \sqsubseteq \mathrm{DT}^{\mathrm{fix}}_\varsigma(P)$ and since the algebraic term semantics is monotonic (Lemma 6.9), we have $[\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma} \trianglelefteq [\![t']\!]^{\mathrm{alg}}_{\mathrm{DT}^{\mathrm{fix}}_\varsigma(P)} = [\![t']\!]^{\mathrm{fix}}_{P,\varsigma}$. Together $[\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma} \trianglelefteq [\![t]\!]^{\mathrm{fix}}_{P,\varsigma}$ and therefore

$$[\![t]\!]^{\mathrm{red}}_{P,\varsigma} = \bigsqcup \{ [\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma} \mid t \xrightarrow[P,\varsigma]{*} t' \} \trianglelefteq [\![t]\!]^{\mathrm{fix}}_{P,\varsigma}$$

and

$$[\![t]\!]^{\mathrm{po}}_{P,\varsigma} = \bigsqcup \{ [\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma} \mid t \xrightarrow[P,\varsigma,\mathrm{po}]{*} t' \} \trianglelefteq [\![t]\!]^{\mathrm{fix}}_{P,\varsigma}.$$

□

Lemma 7.9
The po ς-reduction semantics is sound w.r.t. the global ς-reduction semantics:

$$[\![\cdot]\!]^{\mathrm{po}}_{P,\varsigma} \preceq [\![\cdot]\!]^{\mathrm{red}}_{P,\varsigma}.$$

Proof:

$$\begin{array}{rcccl} & \{t' | t \xrightarrow[P,\varsigma,\mathrm{po}]{*} t'\} & \subseteq & \{t' | t \xrightarrow[P,\varsigma]{*} t'\} & \\ \implies & \{[\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma} | t \xrightarrow[P,\varsigma,\mathrm{po}]{*} t'\} & \subseteq & \{[\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma} | t \xrightarrow[P,\varsigma]{*} t'\} & \implies \quad [\![t]\!]^{\mathrm{po}}_{P,\varsigma} \trianglelefteq [\![t]\!]^{\mathrm{red}}_{P,\varsigma}. \end{array}$$

□

Lemma 7.10
The global ς-reduction semantics is complete w.r.t. the ς-fixed-point semantics:

$$[\![\cdot]\!]^{\mathrm{fix}}_{P,\varsigma} \preceq [\![\cdot]\!]^{\mathrm{red}}_{P,\varsigma}$$

Proof: Let $t \in \mathrm{T}_\Sigma$ and $\mathfrak{A}_i := (\Phi_{P,\varsigma})^i(\bot_\varsigma) \in \mathrm{Int}_{\Sigma,\varsigma}$ for all $i \in \mathbb{N}$. According to the next lemma $\{[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A}_i} \mid i \in \mathbb{N}\}$ is cofinal in $\{[\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma} \mid t \xrightarrow[P,\varsigma]{*} t'\}$ which implies $\bigsqcup_{i\in\mathbb{N}} [\![t]\!]^{\mathrm{alg}}_{\mathfrak{A}_i} \trianglelefteq \bigsqcup\{[\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma} \mid t \xrightarrow[P,\varsigma]{*} t'\}$. Hence we have

$$[\![t]\!]^{\mathrm{fix}}_{P,\varsigma} = [\![t]\!]^{\mathrm{alg}}_{\mathrm{DT}^{\mathrm{fix}}_\varsigma(P)} = [\![t]\!]^{\mathrm{alg}}_{\bigsqcup_{i\in\mathbb{N}}\mathfrak{A}_i} \overset{\text{L. 6.9}}{=} \bigsqcup_{i\in\mathbb{N}} [\![t]\!]^{\mathrm{alg}}_{\mathfrak{A}_i} \trianglelefteq \bigsqcup\{[\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma} \mid t \xrightarrow[P,\varsigma]{*} t'\} = [\![t]\!]^{\mathrm{red}}_{P,\varsigma}.$$

□

The following lemma contains the heart of the completeness proof.

Lemma 7.11
The approximations of the ς-fixed-point semantics are cofinal in those of the global ς-reduction semantics, that is if $t \in \mathrm{T}_\Sigma$ and $\mathfrak{A}_i := (\Phi_{P,\varsigma})^i(\bot_\varsigma) \in \mathrm{Int}_{\Sigma,\varsigma}$ for all $i \in \mathbb{N}$, then for every $i \in \mathbb{N}$ there is a $t' \in \mathrm{T}_\Sigma$ with $t \xrightarrow[P,\varsigma]{*} t'$ and $[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A}_i} \trianglelefteq [\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma}$.

Proof:

$\underline{i = 0:}$ With $t' := t$ we have $t \xrightarrow[P,\varsigma]{*} t'$ and $[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A}_i} = [\![t]\!]^{\mathrm{alg}}_{\bot_\varsigma} = [\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma}$.

$\underline{i \Rightarrow i+1:}$

$\underline{t = f(t_1, \ldots, t_n):}$ $(f^{(n)} \in \mathcal{F})$.

$\underline{\text{Case 1:}}$ $([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}})$ is semantically ς-matchable with the left-hand side of $f(\vec{p}) \to r \in P$ by $\beta : X \to \mathrm{T}_{\mathcal{C},\varsigma}$.
Then

$$[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}} = f^{\mathfrak{A}_{i+1}}([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}}) = [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A}_i,\beta}. \tag{1}$$

In accordance with the hypotheses of the structural induction there exist $t'_1, \ldots, t'_n \in \mathrm{T}_\Sigma$ with

$$t_l \xrightarrow[P,\varsigma]{*} t'_l \tag{2}$$

and

$$[\![t_l]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}} \trianglelefteq [\![t'_l]\!]^{\mathrm{alg}}_{\bot_\varsigma} \tag{3}$$

for all $l \in [n]$.
Equation (2) implies

$$t = f(t_1, \ldots, t_n) \xrightarrow[P,\varsigma]{*} f(t'_1, \ldots, t'_n) \tag{4}$$

The monotonicity of algebraic term semantics w.r.t. the algebra (Lemma 6.9) gives us

$$[\![t'_l]\!]^{\mathrm{alg}}_{\bot_\varsigma} \trianglelefteq [\![t'_l]\!]^{\mathrm{alg}}_{\mathfrak{A}} \tag{5}$$

for all $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$ and $l \in [n]$.

From (3) and (5) and the monotonicity of semantic ς-matching (Lemma 6.7) we conclude that for all ς-interpretations $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$, especially $\mathfrak{A}_i$, $([\![t'_1]\!]^{\mathrm{alg}}_{\mathfrak{A}}, \ldots, [\![t'_n]\!]^{\mathrm{alg}}_{\mathfrak{A}})$ is semantically ς-matchable with $f(\vec{p})$ by respective valuations $\beta_{\mathfrak{A}}$, and

$$\beta \preceq \beta_{\mathfrak{A}}. \tag{6}$$

With Proposition 7.4 about syntactic and semantic ς-matching follows, that $(t'_1, \ldots, t'_n)$ is syntactically ς-matchable with $f(\vec{p})$ by a substitution σ and

$$[\![\hat{t}\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}_i} = [\![\hat{t}]\!]^{\mathrm{alg}}_{\mathfrak{A}_i,\beta_{\mathfrak{A}_i}} \tag{7}$$

for all $\hat{t} \in \mathrm{T}_\Sigma(\mathrm{Var}(\vec{p}))$.
The syntactic ς-matchability implies

$$f(t'_1, \ldots, t'_n) \xrightarrow[P,\varsigma]{} r\sigma. \tag{8}$$

Equation (7) is especially valid for $\hat{t} = r$:

$$[\![r\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}_i} = [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A}_i,\beta_{\mathfrak{A}_i}}. \tag{9}$$

From (6) and the monotonicity of algebraic term semantics w.r.t. the valuation (Lemma 6.8) we deduce

$$[\![r]\!]^{\mathrm{alg}}_{\mathfrak{A}_i,\beta} \trianglelefteq [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A}_i,\beta_{\mathfrak{A}_i}}. \tag{10}$$

Finally the hypotheses of the induction over i assures the existence of a $t' \in \mathrm{T}_\Sigma$ with

$$r\sigma \xrightarrow[P,\varsigma]{*} t' \tag{11}$$

and

$$[\![r\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}_i} \trianglelefteq [\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma}. \tag{12}$$

Together (4), (8) and (11) give

$$t = f(t_1, \ldots, t_n) \xrightarrow[P,\varsigma]{*} f(t'_1, \ldots, t'_n) \xrightarrow[P,\varsigma]{} r\sigma \xrightarrow[P,\varsigma]{*} t' \tag{13}$$

and we have

$$[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}} \overset{(1)}{=} [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A}_i,\beta} \overset{(10)}{\trianglelefteq} [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A}_i,\beta_{\mathfrak{A}_i}} \overset{(9)}{=} [\![r\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}_i} \overset{(12)}{\trianglelefteq} [\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma}. \tag{14}$$

Case 2: Otherwise, that is there exists no matching reduction rule.
Let $t' := t$. Then

$$t \xrightarrow[P,\varsigma]{*} t'$$

and

$$[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}} = f^{\mathfrak{A}_{i+1}}([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}}) = \perp \trianglelefteq [\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma}.$$

$t = G(t_1, \ldots, t_n)$: $(G^{(n)} \in \mathcal{C})$.
According to the hypotheses of the structural induction there exist $t'_1, \ldots, t'_n \in \mathrm{T}_\Sigma$ with

$$t_l \xrightarrow[P,\varsigma]{*} t'_l \tag{1}$$

and

$$[\![t_l]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}} \trianglelefteq [\![t'_l]\!]^{\mathrm{alg}}_{\perp_\varsigma} \tag{2}$$

for all $l \in [n]$.

Together with the monotonicity of all operations of a ς-interpretation (2) implies

$$G^{\mathfrak{A}_{i+1}}([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}}) \trianglelefteq G^{\mathfrak{A}_{i+1}}([\![t'_1]\!]^{\mathrm{alg}}_{\perp_\varsigma}, \ldots, [\![t'_n]\!]^{\mathrm{alg}}_{\perp_\varsigma}). \tag{3}$$

Furthermore, the operation of a constructor symbol is the same in all ς-interpretations:

$$G^{\mathfrak{A}_{i+1}} = G^{\mathfrak{A}_0} = G^{\perp_\varsigma}. \tag{4}$$

Let $t' := G(t'_1, \ldots, t'_n)$. Then (1) gives us

$$t = G(t_1, \ldots, t_n) \xrightarrow[P,\varsigma]{*} G(t'_1, \ldots, t'_n) = t'$$

and (3) and (4) imply

$$\begin{aligned}
[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}} &= G^{\mathfrak{A}_{i+1}}([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}}) \\
&\overset{(3)}{\trianglelefteq} G^{\mathfrak{A}_{i+1}}([\![t'_1]\!]^{\mathrm{alg}}_{\perp_\varsigma}, \ldots, [\![t'_n]\!]^{\mathrm{alg}}_{\perp_\varsigma}) \\
&\overset{(4)}{=} G^{\perp_\varsigma}([\![t'_1]\!]^{\mathrm{alg}}_{\perp_\varsigma}, \ldots, [\![t'_n]\!]^{\mathrm{alg}}_{\perp_\varsigma}) \\
&= [\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma}.
\end{aligned}$$

□

Unfortunately it is not possible to use the same proof method for the po ς-reduction semantics, since the step from (2) to (4) is not valid for po ς-reduction.

7.5 Completeness of Π*-Fair Reduction Semantics*

We do not prove the completeness of the po ς-reduction semantics directly, as has been done in detail in [5]. The proof there is based on O'Donnell's proof that eventually outermost sequences terminate whenever possible (Lemma 10 and Theorem 17 in [24]). Bergstra and Klop extend O'Donnell's proof to so called Π-fair reduction sequences ([2]). By using that extended version we obtain a more general result, showing that any semantics based on Π-fair reduction is complete. This is subsequently applied in Subsections 7.6 and 7.7 to the po ς- and the li* semantics, respectively.

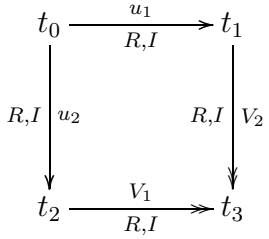
In this and the following subsections I is an arbitrary instance of an almost orthogonal TRS R over a signature Σ, so that $\mathrm{Red}_{R,I}$ is residually closed. $\langle A, \trianglelefteq \rangle$ is a cpo (with $A \neq \emptyset$) and $[\![\cdot]\!]_\perp : \mathrm{T}_\Sigma \to A$ a mapping such that $t \xrightarrow[R,I]{} t' \Longrightarrow [\![t]\!]_\perp \trianglelefteq [\![t']\!]_\perp$.

For having a concrete example, the reader may keep in mind that later R will be our program P, I an instance ς, and $[\![\cdot]\!]_\perp$ a semantic ς-approximation $[\![\cdot]\!]_{\perp_\varsigma}$.

Due to its complexity, Bergstra's and Klop's proof is not reproduced here but only outlined and cited as far as necessary for our extension. Unfortunately it considers only the smaller class of orthogonal TRSs. Nonetheless, the generalization to instances of almost orthogonal TRSs with residually closed redex sets is straightforward, because the proof is based on a variation of the parallel moves lemma (2.5), as already remarked in [2].

Definition 7.3 (Cf. Definition 7.2 in [2])
A predicate $\Pi \subseteq \mathbb{N}^* \times \mathrm{T}_\Sigma$ with $(u,t) \in \Pi \Longrightarrow u \in \mathrm{RedPos}_{R,I}(t)$ is *gaining* iff it has the following three properties:

I) *Preservation of a Π-redex-position.*
For all I-reductions



with $V_1 = u_1 \setminus t_0 \xrightarrow[R,I]{u_2} t_2$, $V_2 = u_2 \setminus t_0 \xrightarrow[R,I]{u_1} t_1$, and all $w_i \in \mathrm{RedPos}_{R,I}(t_i)$ $(i \in [4])$ with

$$w_1 \in w_0 \setminus t_0 \longrightarrow t_1, \quad w_2 \in w_0 \setminus t_0 \longrightarrow t_2,$$
$$w_3 \in w_1 \setminus t_1 \longrightarrow t_3, \quad w_3 \in w_2 \setminus t_2 \longrightarrow t_3,$$

we have

$$\Pi(w_0,t_0), \Pi(w_2,t_2), \Pi(w_3,t_3) \Longrightarrow \Pi(w_1,t_1).$$

II) *$\neg\Pi$-reductions do not create new Π-redex-positions.*
For every reduction $t \xrightarrow[R,I]{u} t'$ such that $\neg\Pi(u,t)$, every v' such that $\Pi(v',t')$ has a respective predecessor, that is a v with $v' \in v \setminus t \xrightarrow[R,I]{u} t'$ such that $\Pi(v,t)$.

III) *$\neg\Pi$-reductions do not change the semantic approximation.*
For every $t \xrightarrow[R,I]{u} t'$ such that $\neg\Pi(u,t)$, we have $[\![t]\!]_\perp = [\![t']\!]_\perp$.

We write $t \xrightarrow[\Pi]{u} t'$ and $t \xrightarrow[\neg\Pi]{u} t'$ for a reduction $t \xrightarrow[R,I]{u} t'$ with $\Pi(u,t)$ and $\neg\Pi(u,t)$, respectively. □

The definition of the first property is slightly different from that given by Bergstra and Klop who use the transitive-reflexive closure $\xrightarrow{*}$ instead of the parallel $\twoheadrightarrow$. Since that would require introducing the supplementary concepts of development and diagram we use our equivalent definition. Property III is added for our extension towards semantics.

Definition 7.4 (Def. 7.5 in [2])

1. Let $A = t_1 \xrightarrow[R,I]{v_1} t_2 \xrightarrow[R,I]{v_2} \ldots$ be a (finite or infinite) reduction sequence such that $u_j \in \mathrm{RedPos}_{R,I}(t_j)$ and $u_{i+1} \in u_i \setminus t_i \xrightarrow[R,I]{v_i} t_{i+1}$ for all $i \geq j$ as far as u_i is defined. Then the sequence $u_j, u_{j+1}, u_{j+2}, \ldots$ is called a *trace* in A.

2. Let A be as in 1. and let $\Pi \subseteq \mathbb{N}^* \times \mathrm{T}_\Sigma$ be a predicate. Then a trace $u_j, u_{j+1}, \ldots$ in A is a *Π-trace* iff for all $i \geq j$ we have $\Pi(u_i, t_i)$.

3. Let A be a reduction and $\Pi \subseteq \mathbb{N}^* \times \mathrm{T}_\Sigma$ be a predicate. Then A is *Π-fair* iff A contains no infinite Π-traces.

□

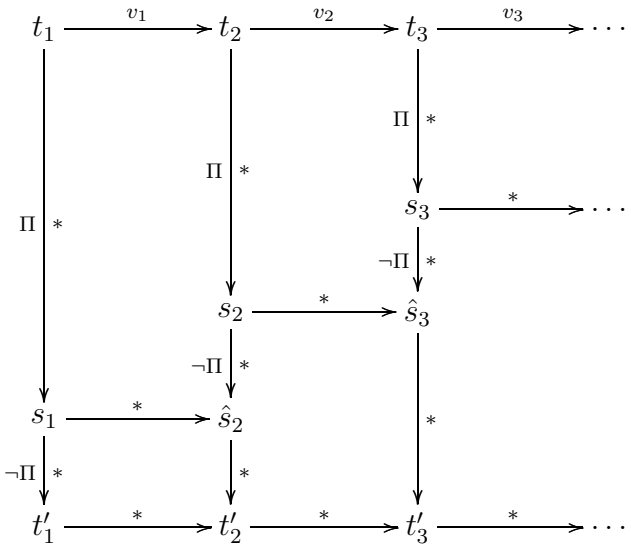


Figure 1: The Π-fair reduction construction.

Lemma 7.12
If Π is a gaining predicate, $A = t_1 \xrightarrow[R,I]{v_1} t_2 \xrightarrow[R,I]{v_2} \ldots$ a Π-fair reduction sequence, and $t_1 \xrightarrow[R,I]{} t'_1$ a reduction, then there exists a Π-*fair reduction construction* as shown in figure 1 such that

1. the sequence $s_1, \hat{s}_2, s_2, \hat{s}_2, s_3, \ldots$ converges to A, that is there exists a $j \in \mathbb{N}$ with $t_i = s_i$ for all $i \geq j$, and
2. the residual reduction sequence $t'_1 \xrightarrow[R,I]{*} t'_2 \xrightarrow[R,I]{*} \ldots$ of the Π-fair reduction construction is Π-fair.

Proof: Theorem 7.8 in [2] (see also Lemma 17 in [24]). □

Lemma 7.13 *Semantic cofinality of* Π*-fair reduction sequences I*
Let $A = t_1 \xrightarrow[R,I]{} t_2 \xrightarrow[R,I]{} \ldots$ be a Π-fair reduction sequence and $B = t_1 \xrightarrow[R,I]{} t'_1$ a reduction. Then exists an $l \in \mathbb{N}_+$ with $[\![t'_1]\!]_\perp \trianglelefteq [\![t_l]\!]_\perp$.
Proof: We consider the Π-fair reduction construction of A and B with the identifiers used in Lemma 7.12 and prove by induction that $[\![t'_1]\!]_\perp \trianglelefteq [\![s_i]\!]_\perp$ for all $i \in \mathbb{N}_+$.

$\underline{i = 1:}$ By property III the reduction $s_1 \xrightarrow[\neg\Pi]{*} t'_1$ implies $[\![t'_1]\!]_\perp = [\![s_1]\!]_\perp$.

$\underline{i \Rightarrow i+1:}$ We have $s_i \xrightarrow{*} \hat{s}_{i+1}$ and $s_{i+1} \xrightarrow[\neg\Pi]{*} \hat{s}_{i+1}$. Our prerequisite concerning $[\![\cdot]\!]_\perp$ and property III imply $[\![s_i]\!]_\perp \trianglelefteq [\![\hat{s}_{i+1}]\!]_\perp = [\![s_{i+1}]\!]_\perp$. Together with the induction hypotheses we get $[\![t'_1]\!]_\perp \trianglelefteq [\![s_{i+1}]\!]_\perp$.

According to Lemma 7.12 there exists an $l \in \mathbb{N}_+$ with $t_l = s_l$. Hence we have $[\![t'_1]\!]_\perp \trianglelefteq [\![s_l]\!]_\perp = [\![t_l]\!]_\perp$. □

Lemma 7.14 *Semantic cofinality of* Π*-fair reduction sequences II*
Let $A = t_1 \xrightarrow[R,I]{} t_2 \xrightarrow[R,I]{} \ldots$ be a Π-fair reduction sequence and $B = t_1 \xrightarrow[R,I]{*} t'$ a reduction. Then exists a $k \in \mathbb{N}_+$ with $[\![t']\!]_\perp \trianglelefteq [\![t_k]\!]_\perp$.

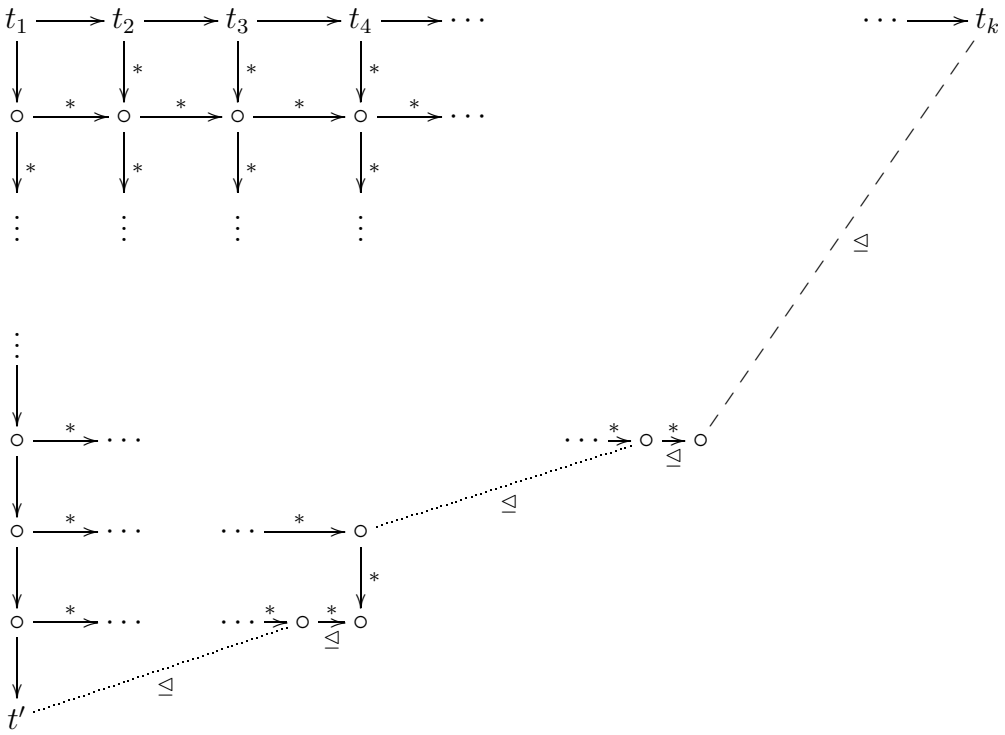


Figure 2: Proof idea of Lemma 7.14

Proof idea: Induction over the length of B, employing the preceding lemma for the induction step; see Figure 2. □

Thus we finally obtain:

Proposition 7.15 *Completeness of* Π*-fair reduction semantics*
Let a *global reduction semantics* $[\![\cdot]\!]^{\mathrm{red}}_{R,I} : \mathrm{T}_\Sigma \to A$ be defined by

$$[\![t]\!]^{\mathrm{red}}_{R,I} := \bigsqcup \{ [\![t']\!]_\perp \mid t \xrightarrow[R,I]{}{}^{*} t' \}$$

and let a 'successor'-function $F : \mathrm{T}_\Sigma \to \mathrm{T}_\Sigma$, fixing a unique Π-fair reduction sequence $t \xrightarrow[R,I]{}{}^{*} F(t) \xrightarrow[R,I]{}{}^{*} F^2(t) \xrightarrow[R,I]{}{}^{*} \ldots$ for every term t, define a Π*-fair reduction semantics* $[\![\cdot]\!]^{\Pi}_{R,I} : \mathrm{T}_\Sigma \to A$ by

$$[\![t]\!]^{\Pi}_{R,I} := \bigsqcup_{i \in \mathbb{N}} [\![F^n(t)]\!]_\perp$$

Then the Π-fair reduction semantics is complete w.r.t. the global reduction semantics, that is

$$[\![\cdot]\!]^{\mathrm{red}}_{R,I} \preceq [\![\cdot]\!]^{\Pi}_{R,I}.$$

□

7.6 Completeness of the Po ς-Reduction Semantics

To apply the result of the last subsection we just have to prove that $\Pi^{\mathrm{o}}_{P,\varsigma}$ is a gaining predicate, where $\Pi^{\mathrm{o}}_{R,I}(u,t)$ holds iff u is an outermost I-redex position of t. Since this proof is only tedious it was moved to the appendix.

Lemma 7.16
The predicate $\Pi^{\mathrm{o}}_{P,\varsigma}$ is gaining.

Proof: Property I: Lemma A.1. Property II: Lemma A.4 and Lemma A.2. Property III: Lemma 6.13 (gain of information). □

Corollary 7.17
The po ς-reduction semantics is complete w.r.t. the global ς-reduction semantics, that is $[\![\cdot]\!]^{\mathrm{red}}_{P,\varsigma} \preceq [\![\cdot]\!]^{\mathrm{po}}_{P,\varsigma}$.

Proof: The po ς-reduction semantics is a $\Pi^{\mathrm{o}}_{P,\varsigma}$-fair reduction semantics. □

7.7 Completeness of the Lo Cbv Reduction Semantics

Analogously to the previous subsection we define a predicate $\Pi^{\mathrm{lo}}_{R,I}$ by $\Pi^{\mathrm{lo}}_{R,I}(u,t) :\Longleftrightarrow$ u is the leftmost-outermost I-redex position of t. Proving $\Pi^{\mathrm{lo}}_{P,\mathrm{cbv}}$ to be gaining was moved to the appendix as well. We finally obtain:

Lemma 7.18
The predicate $\Pi^{\mathrm{lo}}_{P,\mathrm{cbv}}$ is gaining.

Proof: Property I: Lemma A.5. Property II: Lemma A.6. Property III: Lemma A.7. □

Corollary 7.19
The lo cbv reduction semantics (li* reduction semantics) is complete w.r.t. the global cbv reduction semantics, that is $[\![\cdot]\!]^{\mathrm{red}}_{P,\mathrm{cbv}} \preceq [\![\cdot]\!]^{\mathrm{lo}}_{P,\mathrm{cbv}}$. □

8 Properties of the ς-Semantics

In Section 4 we discussed some properties which we expect our semantics to have. We record that since $[\![\cdot]\!]_{P,\varsigma} = [\![\cdot]\!]^{\mathrm{alg}}_{\mathrm{DT}_\varsigma(P)}$ is an algebraic semantics, the ς-semantics is compositional. The ς-semantics is modular as well; the broad reasoning of Section 4 can easily be transformed into a formal proof for the ς-semantics.

We mentioned in Section 4 that the ς-term semantics uniquely determines the ς-data type although the equation

$$f^{\mathrm{DT}_\varsigma(P)}(\vec{\underline{t}}) = [\![f(\vec{t})]\!]_{P,\varsigma}$$

(for all $f^{(n)} \in \Sigma$, $\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$, $\vec{t} \in (\mathrm{T}_\Sigma)^n$ with $[\![t_1]\!]_{P,\varsigma} = \underline{t}_1, \ldots, [\![t_n]\!]_{P,\varsigma} = \underline{t}_n$) does not give a complete definition of the ς-data type $\mathrm{DT}_\varsigma(P)$ by $[\![\cdot]\!]_{P,\varsigma}$, because there may not be for every $\underline{t} \in \mathrm{T}_{\mathcal{C},\varsigma} \setminus \mathrm{T}_\mathcal{C}$ a $t \in \mathrm{T}_\Sigma$ with $[\![t]\!]_{P,\varsigma} = \underline{t}$.

The problem can be remedied, because the ς-semantics is modular and the data type $\langle \mathrm{T}_{\mathcal{C},\varsigma}, \trianglelefteq \rangle$ is ω-inductive. If there is a term $t_\perp$ with $[\![t_\perp]\!]_{P,\varsigma} = \perp$, then for any finite partial data term $\underline{t} \in (\mathrm{T}_{\mathcal{C},\varsigma} \cap \mathrm{T}_{\mathcal{C},\perp}) \setminus \mathrm{T}_\mathcal{C}$ a program term $t \in \mathrm{T}_\Sigma$ with $[\![t]\!]_{P,\varsigma} = \underline{t}$ is constructible by using $t_\perp$ and constructor symbols. If there is no such $t_\perp$, then we extend our program P by a new function symbol $c_\perp$ to a program P', without giving a rule for $c_\perp$. Consequently $[\![c_\perp]\!]_{P',\varsigma} = \perp$. If we can solve the problem for the infinite data terms as well, then we have $\mathrm{DT}_\varsigma(P) = \mathrm{DT}_\varsigma(P')|_\Sigma$ due to modularity. Since $\langle \mathrm{T}_{\mathcal{C},\varsigma}, \trianglelefteq \rangle$

is ω-inductive, there exist for all tuples of (possibly infinite) terms $\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$ chains $T_1, \ldots, T_n \subseteq \mathrm{T}_{\mathcal{C},\varsigma} \cap \mathrm{T}_{\mathcal{C},\perp}$ of finite terms with $\underline{t}_1 = \bigsqcup T_1, \ldots, \underline{t}_n = \bigsqcup T_n$ and these finite terms can be denoted by program terms as just observed. Since all operations are continuous we have

$$\begin{array}{rcl} f^{\mathrm{DT}_\varsigma(P)}(\vec{\underline{t}}) & = & f^{\mathrm{DT}_\varsigma(P)}(\bigsqcup T_1, \ldots, \bigsqcup T_n) \\ & = & \bigsqcup f^{\mathrm{DT}_\varsigma(P)}(T_1, \ldots, T_n) \quad = \quad \bigsqcup\{[\![f(\vec{t})]\!]_{P,\varsigma} \mid \vec{t} \in (T_1, \ldots, T_n)\} \end{array}$$

and the ς-data type is completely defined.

It should be noted that a similar reasoning using furthermore the fact that our programs are only of first order shows that the ς-semantics is *fully abstract* ([22, 16]).

As mentioned in Section 5.4, our language permits the definition of non-sequential operations — in contrast to common functional programming languages. By restricting the form of admissible patterns the definable operations can be restricted to the usual sequential one (cf. Section 10). The issue of sequentiality is discussed further in [5].

We noticed already w.r.t. the cbn semantics in Subsection 5.4 that since the reduction semantics is defined as least upper bound of semantic approximations, data terms are possibly only approximated to arbitrary precision, but never actually reached as final results. Obviously this is unavoidable for infinite data terms and many partial data terms, but we do expect an operational semantics to compute all finite, total data terms.

Lemma 8.1 *Computability of the ς-reduction semantics*
Let $t \in \mathrm{T}_\Sigma$.

1. If $[\![t]\!]^{\mathrm{red}}_{P,\varsigma} = \underline{t} \in \mathrm{T}_\mathcal{C}$, then $t{\downarrow_{P,\varsigma}}$ exists and $[\![t]\!]^{\mathrm{red}}_{P,\varsigma} = \underline{t} = t{\downarrow_{P,\varsigma}}$.
2. If $[\![t]\!]^{\mathrm{po}}_{P,\varsigma} = \underline{t} \in \mathrm{T}_\mathcal{C}$, then $t{\downarrow_{P,\mathrm{po},\varsigma}}$ exists and $[\![t]\!]^{\mathrm{po}}_{P,\varsigma} = \underline{t} = t{\downarrow_{P,\mathrm{po},\varsigma}}$.

Proof:

1. Let $\underline{t} = [\![t]\!]^{\mathrm{red}}_{P,\varsigma} = \bigsqcup\{[\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma} \mid t \xrightarrow[P,\varsigma]{*} t'\} \in \mathrm{T}_\mathcal{C}$. Since $\langle \mathrm{T}_{\mathcal{C},\varsigma}, \trianglelefteq \rangle$ is ω-inductive and $\underline{t}$ is ω-compact, there exists $[\![\hat{t}]\!]^{\mathrm{alg}}_{\perp_\varsigma} \in \{[\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma} \mid t \xrightarrow[P,\varsigma]{*} t'\}$ with $\underline{t} = [\![\hat{t}]\!]^{\mathrm{alg}}_{\perp_\varsigma}$. Then $\underline{t} \in \mathrm{T}_\mathcal{C}$ implies $\hat{t} = \underline{t}$. Moreover $t \xrightarrow[P,\varsigma]{*} \hat{t} = \underline{t}$ and hence $\underline{t}$ is the ς-normal-form of t.
2. Analogously.

□

As an aside we remark that together with the equality of the general and the po ς-reduction semantics the equality of the ς-normal-form and the po ς-normal-form of a term follows.

9 Applications of ς-Semantics

Not only do ς-semantics provide a uniform framework for cbv and cbn semantics, but also the mixed strictnesses permit a natural description of real functional programming languages which rarely have a pure cbv or cbn semantics.

The functional programming language HOPE with its strict functions and non-strict constructors has already been mentioned. All basically strict languages like (the functional subsets of) ML and SCHEME have a predefined set of non-strict functions: a form of conditional (`if`), various others like sequential logical `and` and `or`, and `delay` in SCHEME for simulating cbn evaluation.

For non-strict functional programming languages strictness is an issue, because cbv evaluation proves to be more efficient than cbn evaluation and also facilitates parallel implementations ([11], Chapter 13 in [27]). Therefore, many modern non-strict functional programming languages provide various kinds of strictness annotations which may be generated automatically by a strictness analysis or supplied by the user ([26, 25, 27]). These annotations may be for arguments and/or results of functions and/or constructors in type declarations and/or at individual application points.

Strictness annotations at argument positions in type declarations coincide with our forced strictness. Hence one may prove the equivalence of the cbn semantics and a particular ς-semantics for a particular program and then use a ς-reduction strategy for a correct implementation.

Strictness annotations at individual application points are useful when information about contexts is included. For example in the program

```
even(Zero)           →  Succ(Zero)
even(Succ(Zero))     →  Zero
even(Succ(Succ(x)))  →  even(x)
```

$\texttt{even}^{\mathrm{DT}_{\mathrm{cbn}}(P)}(\underline{t})$ is only unequal to $\bot$ for total terms $\underline{t} \in \mathrm{T}_{\mathcal{C}}$. Notions like structure strictness (our strictness), spine strictness, and normalization strictness describe this kind of dependency on the (non-)partiality of arguments. This information can be transferred to strictness annotations at individual applications of constructors, for instance in `even(Succ(a))` the position of `Succ` may be considered strict in its argument.

This kind of annotation is not expressible by ς-semantics. It is not the aim of this paper but would even contradict a uniform presentation of cbv and cbn semantics, since varying forced strictnesses of constructors would require $\mathrm{T}^{\infty}_{\mathcal{C},\bot}$ as carrier set of the data-type, excluding the true cbv semantics. However, continuing in this direction is straightforward.

10 Efficient ς-Reduction Strategies

The po ς-reduction strategy is too costly to be used in actual implementations. Sharing common subterms yields an improvement. Even in the case of cbv semantics it saves space and time of copying complex data structures. However, since this technique is orthogonal to and at a different level than the concept of reduction strategy, it is not discussed here. The step from ς-reduction semantics to a ς-graph reduction semantics in the spirit of Raoult and Kennaway ([17]) should be small.

Obviously a complete ς-reduction strategy does not need to reduce all ς-redexes which are reduced by the po ς-reduction strategy. Many approximations of a subset of needed ς-redexes will spring to the reader's mind. The concept of gaining predicates enables easy proofs of completeness of such ς-reduction strategies.

Restricting the set of admissible patterns of a program can lead to considerably simpler ς-reduction strategies. In Berry's example ([3])

$$\begin{array}{lcl} \texttt{h(True, False, x)} & \to & \texttt{True} \\ \texttt{h(False, x, True)} & \to & \texttt{True} \\ \texttt{h(x, True, False)} & \to & \texttt{True} \end{array}$$

all three arguments in the term $\texttt{h}(t_1,\ t_2,\ t_3)$ have to be reduced in parallel (alternately)[7], since the term's semantics may not be $\bot$ even if any one argument denotes $\bot$. Hence, one may only permit sequential sets of patterns (cf. strong sequentiality in [14]), that is those which are translatable into explicit tests; for example:

$$\texttt{add(x, y)} \ \to \ \texttt{case}_{\texttt{Zero,Succ}}\texttt{(y, x, Succ(add(x, sel}_{\texttt{Succ},1}\texttt{(y))))}$$

The semantics of $\texttt{case}_{C_1,\ldots,C_n}$ and $\texttt{sel}_{C_j,i}$ is given by

$$\begin{array}{lcl} \texttt{case}_{C_1,\ldots,C_n}(C_j(\vec{t}), t_1, \ldots, t_n) & \to & t_j \\ \texttt{sel}_{C_j,i}(C_j(\vec{t})) & \to & t_i \end{array}$$

with $\mathcal{C} = \{C_1, \ldots, C_n\}$, $C_j^{(m)} \in \mathcal{C}$, $i \in [m]$.

This transfers some complexity from the reduction strategy to the case expressions which replace the patterns. Locating a ς-redex in a term is still not trivial though, due to the possible forced strictness of functions and constructors. Moreover, while patterns form a natural part of the ς-semantics, `case` is a special function since it needs to be non-strict in all its arguments but the first, even in the cbv semantics.

As an aside should be noted that for programs with tests instead of patterns the lo cbn reduction strategy is normalizing ([24, 2]) but not complete: With `undef` $\to$ `undef` reducing `[A, B, undef, C]` does not yield its cbn semantics `[A, B,` $\bot$`, C]` but `A:B:`$\bot$. However, this is exactly the result any interpreter of a modern non-strict functional programming language gives. The lo reduction strategy becomes complete, when we define the list constructor `Cons` (`:`) of the output list to be strict in its first argument (Any output of a real-world functional program is either a list of characters (string) or a list of commands for the operating system).

11 Conclusion

In this paper we defined and studied the ς-semantics, a comprehensive semantics for functional constructor-based programs with patterns.

By introducing the forced strictness ς as a parameter we obtained a single definition for both cbv and cbn semantics. Thus the true common and distinguishing features of these two standard semantics were highlighted. We established that the usual operational definition by innermost, respectively outermost reduction is rather deceptive. Instead, a forced strictness ς fixes an instance of the program characterized by a set of ς-redexes. Reductions in this instance are sound w.r.t. the ς-semantics and therefore serve as basis for operational ς-semantics.

We defined a denotational ς-semantics which assigns a data-type to a program in a modular way and moreover ensures the compositionality of the ς-semantics. We defined two operational ς-semantics. The general ς-reduction semantics is elementary and the means by which the completeness of arbitrary Π-fair reduction semantics was proved. This result was applied to the po ς-semantics. Furthermore, it provides a tool for proving the soundness and completeness of other, future ς-reduction semantics. In

[7]Surprisingly, a complete sequential reduction strategy exists nonetheless ([1]). It is practically useless due to its inefficiency though.

general, we are able to validate a property for all instances of the ς-semantics by a single proof.

We saw that, although implementations may translate patterns into explicit tests, the ς-semantics is naturally defined on the basis of patterns. Finally, the ς-semantics gives a simple formal foundation for the prevailing mixed strictnesses of modern functional programming languages.

We did not give a declarative ς-semantics, that is a semantics which views reduction rules as equations and uses models, in which all equations are valid, as data-types. The reason is that generally ς-data-types are not models, due to the forced strictness ς which does not appear in the program. Only the cbn data-type is a model. The cbn data-type can even be uniquely characterized as the minimal interpretation which is a model ($\mathrm{DT}_{P,\mathrm{cbn}}(P) = \mathrm{Least}_{\langle \mathrm{Int}_{\Sigma,\mathrm{cbn}},\sqsubseteq\rangle}(\mathrm{Int}_{\Sigma,\mathrm{cbn}} \cap \mathrm{Mod}(P))$). For the cbv data-type an analogous characterization exists, if partial algebras (with true partial operations) are used instead of $\mathrm{Alg}^{\infty}_{\Sigma,\perp}$. Nonetheless, a simple declarative ς-semantics seems not to be attainable ([5]).

We considered only first-order functional programs without sorts or types. Thus the essential points were well exposed. Higher-order functional programs require more complicated semantic domains than simple algebras, and the introduction of a type system, at least to distinguish base data elements formed by constructors and functions of different orders, would be unavoidable. The concept of ς-semantics should carry over to higher-order functional programs without principal difficulties. Due to the definability of non-sequential operations this semantics would be fully abstract as well ([28]). Nonetheless, it may be worthwhile to put this into effect, since new issues may arise and further insights may be gained. The work on unboxed values ([26, 19]) may be a good guide for the definition of the semantic domains and the type system.

Formal semantics are not only the prerequisite for the implementation of correct interpreters and compilers and the verification of programs, but also lead us to a continuously improving understanding of the nature of programming languages.

Acknowledgement. The author is especially indebted to Thomas Noll for many fruitful discussions.

References

[1] S. Antoy and A. Middeldorp. A sequential reduction strategy. In *Proceedings of the 4th International Conference on Algebraic and Logic Programming*, LNCS 850, pages 168–185, Madrid, September 1994.

[2] J.A. Bergstra and J.W. Klop. Conditional rewrite rules: Confluence and termination. *Journal of Computer and System Sciences*, 32:323–362, 1986.

[3] G. Berry. Stable models of typed λ-calculi. In *Proceedings of the 5th ICALP Conference*, LNCS 62, pages 73–89, Udine, Italy, 1978.

[4] G. Berry and J.-J. Lévy. Minimal and optimal computations of recursive programs. In *Fourth ACM Symposium on Principles of Programming Languages*, pages 215–226, 1977.

[5] O. Chitil. Denotationelle und operationelle Semantiken für konstruktorbasierte funktionale Programmiersprachen erster Ordnung. Master's thesis, Aachen University of Technology, 1995.

[6] W. Clinger and J. Rees (eds.). Revised^4 report on the algorithmic language Scheme, November 1991.

[7] B. Courcelle. Recursive applicative program schemes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 459–492. Elsevier, Amsterdam, 1990.

[8] O. Danvy and J. Hatcliff. Cps-transformation after strictness analysis. *ACM Letters on Programming Languages and Systems*, 1(3):195–212, September 1992.

[9] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–320. Elsevier, Amsterdam, 1990.

[10] P.J. Downey and R. Sethi. Correct computation rules for recursive languages. *SIAM J. Comput.*, 5(3):378–401, September 1976.

[11] A.F. Field and P.G. Harrison. *Functional Programming*. Addison-Wesley, 1988.

[12] J.A. Goguen, J.W. Thatcher, E.G. Wagner, and J.B. Wright. Initial algebra semantics and continuous algebras. *JACM*, 24(1):68–95, January 1977.

[13] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *JACM*, 27(4):797–821, 1980.

[14] G. Huet and J.-J. Lévy. Computations in orthogonal rewriting systems I + II. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic – Essays in Honor of Alan Robinson*, pages 395–443. MIT Press, 1991. Original version: Call by Need Computations in Non-Ambiguous Term Rewriting Systems, Report 359, INRIA, 1979.

[15] G. Huet and D.C. Oppen. Equations and rewrite rules: A survey. In R. Book, editor, *Formal Language Theory: Perspectives and Open Problems*, pages 349–405. Academic Press, 1980.

[16] A. Jung et al. Domains and denotational semantics: History, accomplishments and open problems. Technical Report CSR-96-2, University of Birmingham, 1996. http://www.cs.bham.ac.uk/∼axj/papers.html.

[17] R. Kennaway. On "On graph rewritings". *Theoretical Computer Science*, 52:37–58, 1987.

[18] J.W. Klop. Term rewriting systems. In S. Abramsky, Dov. M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 1–116. Oxford University Press, 1992.

[19] J. Launchbury and R. Paterson. Parametricity and unboxing with unpointed types. In *Programming Languages and Systems — ESOP'96*, LNCS 1058, pages 204–218, 1996.

[20] J. Loeckx and K. Sieber. *The Foundations of Program Verification, Part B.* Wiley–Teubner, 1987.

[21] Z. Manna. *Mathematical Theory of Computation*, chapter 5. MCGraw Hill, 1974.

[22] R. Milner. Fully abstract models of typed λ-calculi. *Theoretical Computer Science*, 4:1–22, 1977.

[23] M. Nivat. On the interpretation of recursive polyadic program schemes. In *Symposia Mathematica 15 – Convegni del Febbraio e dell' Aprile del 1973*, pages 255–281, Rom, 1975.

[24] M.J. O'Donnell. *Computing in Systems Described by Equations.* LNCS 58. Springer, 1977.

[25] J. Peterson, K. Hammond, et al. Report on the programming language Haskell, version 1.3. Technical Report YALEU/DCS/RR-1106, Yale University, 1996. http://haskell.cs.yale.edu/haskell-report/haskell-report.html.

[26] S.L. Peyton-Jones and John Launchbury. Unboxed values as first class citizens in a non-strict functional language. In *Conf. on Functional Programming Languages and Computer Architecture*, LNCS 523, pages 636–666, 1991.

[27] R. Plasmeijer and M. van Eekelen. Concurrent Clean language report, version 1.0, April 1995.

[28] G.D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.

[29] B.K. Rosen. Tree-manipulating systems and Church-Rosser theorems. *JACM*, 20:160–187, 1973.

[30] S.J. Thompson. A logic for Miranda. *Formal Aspects of Computing*, 1:339–365, 1989.

[31] S.J. Thompson. A logic for Miranda, revisited. *Formal Aspects of Computing*, 7:412–429, 1995.

[32] J. Vuillemin. Correct and optimal implementations of recursion in a simple programming language. *Journal of Computer and System Sciences*, 9:332–354, 1974.

[33] J. Vuillemin. *Syntax, sémantique et axiomatique d'un langage de programmation simple.* Thèse, Université Paris VI, 1974.

[34] W. Wechler. *Universal Algebra for Computer Scientists.* Springer, 1992.

[35] M. Wirsing. Algebraic specification. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics*, pages 675–788. Elsevier, Amsterdam, 1990.

A Predicates $\Pi^{\mathrm{o}}_{P,\varsigma}$ and $\Pi^{\mathrm{lo}}_{P,\mathrm{cbv}}$ Are Gaining

Here we give the proofs which were omitted in Subsections 7.6 and 7.7.

As there, I is an arbitrary instance of an almost orthogonal TRS R over a signature Σ, so that $\mathrm{Red}_{R,I}$ is residually closed, $\langle A, \trianglelefteq\rangle$ is a cpo with $A \neq \emptyset$, and $[\![\cdot]\!]_\perp : \mathrm{T}_\Sigma \to A$ a mapping with $t \xrightarrow[R,I]{} t' \Longrightarrow [\![t]\!]_\perp \trianglelefteq [\![t']\!]_\perp$.

A.1 $\Pi^{\mathrm{o}}_{P,\varsigma}$ *Is Gaining*

We show that $\Pi^{\mathrm{o}}_{R,I}$ has property I and $\Pi^{\mathrm{o}}_{P,\varsigma}$ has property II of gaining redexes. Property III is already proved for $\Pi^{\mathrm{o}}_{P,\varsigma}$ by Lemma 6.13 about gain of information.

Lemma A.1 *Preservation of an outermost redex-occurrence*
$\Pi^{\mathrm{o}}_{R,I}$ has property I of gaining redexes.

Proof: (Cf. Example 7.4 (ii) in [2])
Let $(v] := \{v' \in \mathbb{N}^*_+ \mid v' \leq v\}$ for arbitrary $v \in \mathbb{N}^*_+$. We use the same identifiers as in definition 7.3.

a) Show: $(w_1] \cap V_2 = \emptyset$.

Suppose $w_1' \in (w_1] \cap V_2$. From $w_1 \leq w_1$, $w_1' \in u_2 \setminus t_0 \xrightarrow{u_1} t_1$ and $w_1 \in w_0 \setminus t_0 \xrightarrow{u_1} t_1$ the definition of residuals gives us $u_2 \leq w_0$. However, $w_0 = u_2$ contradicts $w_1 \in w_0 \setminus t_0 \xrightarrow{u_2} t_2$ and $u_2 < w_0$ contradicts $\Pi^{\mathrm{o}}_{R,I}(w_0, t_0)$.

b) Show: $\Pi^{\mathrm{o}}_{R,I}(w_1, t_1)$.

Suppose $\neg\Pi^{\mathrm{o}}_{R,I}(w_1, t_1)$. Then exists w_1' with $\Pi^{\mathrm{o}}_{R,I}(w_1', t_1)$ and $w_1' < w_1$. Since $(w_1'] \subseteq (w_1]$ property a) gives us $(w_1'] \cap V_2 = \emptyset$. This implies $w_1' \setminus t_1 \xrightarrow{V_2\!\!\!\!\rightarrow} t_3 = \{w_1'\} \in \mathrm{RedPos}_{R,I}(t_3)$. Since $w_1' < w_1$ and by a) we know that $\{w_3\} = w_1 \setminus t_1 \xrightarrow{V_2\!\!\!\!\rightarrow} t_3 = \{w_1\}$, we get $\Pi^{\mathrm{o}}_{R,I}(w_3, t_3)$ in contradiction to the assumptions.

□

Verifying property II of gaining predicates takes more effort.

Definition A.1 (Cf. Def. 32 in [24])
$\mathrm{Red}_{R,I}$ is *outer* iff

$$\left.\begin{array}{l} t \xrightarrow[R,I]{w} t' \text{ is a reduction,} \\ u < v < w, v \in \mathrm{RedPos}_{R,I}(t), u \in \mathrm{RedPos}_{R,I}(t') \end{array}\right. \Longrightarrow \quad u \in \mathrm{RedPos}_{R,I}(t).$$

□

Lemma A.2 (Lemma 14 in [24])
If $\mathrm{Red}_{R,I}$ is outer, then $\Pi^{\mathrm{o}}_{R,I}$ satisfies property II of gaining redexes.

Proof: Let $t \xrightarrow[R,I]{u} t'$ be a reduction with $\neg\Pi^{\mathrm{o}}_{R,I}(u, t)$ and $\Pi^{\mathrm{o}}_{R,I}(v', t')$.

a) Show: There is no $w < v'$ with $w \in \mathrm{RedPos}_{R,I}(t)$.

Suppose such a w exists. Without loss of generality $\Pi^{\mathrm{o}}_{R,I}(w, t)$. Then $w \setminus t \xrightarrow{u} t' = \{w\} \subseteq \mathrm{RedPos}_{R,I}(t')$ in contradiction to $\Pi^{\mathrm{o}}_{R,I}(v', t')$.

b) Show: $v' \in \mathrm{RedPos}_{R,I}(t)$.

$u < v'$ contradicts a) and $u = v'$ together with a) contradicts $\neg\Pi^{\mathrm{o}}_{R,I}(u,t)$, leaving:

$v' \parallel u$: Then $t/v' = t'/v'$ and therefore $v' \in \mathrm{RedPos}_{R,I}(t)$.

$v' < u$: Since $\neg\Pi^{\mathrm{o}}_{R,I}(u,t)$ there is a $w \in \mathrm{RedPos}_{R,I}(t)$ with $w < u$, and even $v' \leq w < u$ according to a). The outer property finally gives $v' \in \mathrm{RedPos}_{R,I}(t)$.

Together a) and b) imply $\Pi^{\mathrm{o}}_{R,I}(v,t)$ and $v \setminus t \xrightarrow[R,I]{u} t' = \{v'\}$ for $v := v'$. □

It only remains to show that our redex sets are outer.

Lemma A.3 ([24])
The redex set Red_R is outer.

Proof: Let $t \xrightarrow[l\longrightarrow r]{w} t'$ be a reduction, $u < v < w$, $v \in \mathrm{RedPos}_R(t)$, and $u \in \mathrm{RedPos}_R(t')$. Hence exist $v', w' \in \mathbb{N}_+^*$ with $v = u.v'$ and $w = v.w' = u.v'.w'$. The reduction implies the existence of a substitution σ with

$$t/u = t'/u[v'.w' \leftarrow l\sigma] \text{ and } t'/u = t/u[v'.w' \leftarrow r\sigma] \tag{1}$$

Since $u \in \mathrm{RedPos}_R(t')$, there exists a redex scheme $\hat{l} \in \mathrm{RedS}_R$ and a substitution $[t_1/x_1, \ldots, t_n/x_n] : \mathrm{Var}(\hat{l}) \to \mathrm{T}_\Sigma$ with

$$t'/u = \hat{l}[t_1/x_1, \ldots, t_n/x_n] \tag{2}$$

Since Red_R is residually closed and $v \setminus t \xrightarrow[l\longrightarrow r]{w} t' = \{v\}$, we have $v \in \mathrm{RedPos}_R(t')$. According to Lemma 2.1 there exist $v_1, v_2 \in \mathbb{N}_+^*$ with $v = u.v_1.v_2$ and $\hat{l}/v_1 \in X$, that is $\hat{l}/v_1 = x_k$ for a $k \in [n]$.

Altogether we obtain

$$\begin{aligned} t/u &\overset{(1)}{=} t'/u[v'.w' \leftarrow l\sigma] \\ &\overset{(2)}{=} (\hat{l}[t_1/x_1, \ldots, t_n/x_n])[v'.w' \leftarrow l\sigma] \\ &= \hat{l}[t_1/x_1, \ldots, t_k[v_2.w' \leftarrow l\sigma]/x_k, \ldots, t_n/x_n]. \end{aligned}$$

Hence $t/u \in \mathrm{Red}_R$ and therefore $u \in \mathrm{RedPos}_R(t)$. □

Lemma A.4
The redex set $\mathrm{Red}_{P,\varsigma}$ is outer.

Proof: Let $t \xrightarrow[l\longrightarrow r,\varsigma]{w} t'$ be a ς-reduction, $u < v < w$, $v \in \mathrm{RedPos}_{P,\varsigma}(t)$, and $u \in \mathrm{RedPos}_{P,\varsigma}(t')$. Hence exist $v', w' \in \mathbb{N}_+^*$ and $k \in \mathbb{N}_+$ with $v = u.k.v'$ and $w = v.w' = u.k.v'.w'$. Since $\mathrm{Red}_{P,\varsigma} \subseteq \mathrm{Red}_P$, the previous lemma gives us $u \in \mathrm{RedPos}_P(t)$, that is

$$t/u = f(t_1, \ldots, t_n) \text{ and } t'/u = f(t'_1, \ldots, t'_n)$$

for an $f^{(n)} \in \mathcal{F}$ and some $t_1, \ldots, t_n, t'_1, \ldots, t'_n \in \mathrm{T}_\Sigma$ with

$$t_i = t'_i \tag{1}$$

for all $i \in [n]$ with $i \neq k$, and

$$t_k \xrightarrow[l\longrightarrow r,\varsigma]{v'.w'} t'_k.$$

Since $v' \in \mathrm{RedPos}_{P,\varsigma}(t)$, we have $t_k \xrightarrow[P,\varsigma,\mathrm{no}]{v'.w'} t'_k$. Then by Lemma 6.13 about gain of information

$$[\![t_k]\!]^{\mathrm{alg}}_{\perp_\varsigma} = [\![t'_k]\!]^{\mathrm{alg}}_{\perp_\varsigma}. \tag{2}$$

Since $u \in \mathrm{RedPos}_{P,\varsigma}(t')$, the function symbol f is not forcedly strict for $([\![t'_1]\!]^{\mathrm{alg}}_{\perp_\varsigma}, \ldots, [\![t'_n]\!]^{\mathrm{alg}}_{\perp_\varsigma})$, and $([\![t'_1]\!]^{\mathrm{alg}}_{\perp_\varsigma}, \ldots, [\![t'_n]\!]^{\mathrm{alg}}_{\perp_\varsigma})$ is semantically ς-matchable with the pattern $\vec{p}$ of a redex scheme $f(\vec{p}) \in \mathrm{RedS}_P$. Due to (1) and (2) this is valid for $([\![t_1]\!]^{\mathrm{alg}}_{\perp_\varsigma}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\perp_\varsigma})$ as well. Hence $u \in \mathrm{RedPos}_{P,\varsigma}(t)$. □

A.2 $\Pi^{\mathrm{lo}}_{P,\mathrm{cbv}}$ *Is Gaining*

We show that $\Pi^{\mathrm{lo}}_{R,I}$ has property I and $\Pi^{\mathrm{lo}}_{P,\mathrm{cbv}}$ has property II and property III of gaining redexes.

Lemma A.5 *Preservation of an lo redex occurrence*
$\Pi^{\mathrm{lo}}_{R,I}$ has property I of gaining redexes.

Proof: Using the identifiers of definition 7.3 we have $\Pi^{\mathrm{o}}_{R,I}(w_1, t_1)$ due to Lemma A.1. Let w'_1 be the lo I-redex position of t_1 and suppose that $w_1 \neq w'_1$. Analogously to the proof of Lemma A.1 we conclude $w'_1 \in \mathrm{RedPos}_{R,I}(t_3)$ with $w'_1 <_{\mathrm{lex}} w_1 = w_3$ in contradiction to the assumption $\Pi^{\mathrm{lo}}_{R,I}(w_3, t_3)$. □

Lemma A.6
$\Pi^{\mathrm{lo}}_{P,\mathrm{cbv}}$ has property II of gaining redexes.

Proof: Let $t \xrightarrow[P,\mathrm{cbv}]{u} t'$ be a reduction with $\neg\Pi^{\mathrm{lo}}_{P,\mathrm{cbv}}(u, t)$. Let v and v' be the lo cbv redex positions of t and t', respectively. Obviously $t \xrightarrow[P,\mathrm{cbv}]{u} t' = \{v\} \subseteq \mathrm{RedPos}_{P,\mathrm{cbv}}(t')$. To show that $v = v'$ suppose:

$\underline{v <_{\mathrm{lex}} v'}$: Contradicts v' being lo in t'.

$\underline{v' <_{\mathrm{lex}} v}$:

- $\underline{v' < v}$: Contradicts v' being a cbv redex, because cbv redexes never contain any redexes as proper subterms.
- $\underline{v' \parallel v}$: $\underline{v' > u}$: Together with $v' <_{\mathrm{lex}} v$ this implies $u <_{\mathrm{lex}} v$ in contradiction to v being lo in t.
  - $\underline{v' \parallel u}$: Because of $t/v' = t'/v'$ we have $v' \in \mathrm{RedPos}_{P,\mathrm{cbv}}(t)$. Together with $v' <_{\mathrm{lex}} v$ this contradicts v being lo in t.
  - $\underline{v' < u}$: All three ordering assumptions together imply $u <_{\mathrm{lex}} v$ in contradiction to v being lo in t.

□

Lemma A.7
$\Pi^{\mathrm{lo}}_{P,\mathrm{cbv}}$ has property III of gaining redexes.

Proof: Let o be the lo cbv redex position of t. $t \xrightarrow[P,\mathrm{cbv}]{u} t'$ with $\neg\Pi^{\mathrm{lo}}_{P,\mathrm{cbv}}(u, t)$ implies $o := \mathrm{LOuter}_{P,\mathrm{cbv}}(t) \neq u$ so that $o \setminus t \xrightarrow[P,\mathrm{cbv}]{u} t' = \{o\} \subseteq \mathrm{RedPos}_{P,\mathrm{cbv}}(t')$. Consequently t and t' still contain redexes and we have $[\![t]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbv}}} = \perp = [\![t']\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbv}}}$. □