Integrating Functional-Logic and State-Oriented Specifications

Peter Padawitz
padawitz@cs.uni-dortmund.de
http://ls5.cs.uni-dortmund.de/~peter
University of Dortmund

Abstract

The paper summarizes basic issues about the syntax, semantics and proof theory of *Swinging types* that were developed for specifying and verifying "static" data types and dynamic systems within a uniform logic. More details and all proofs can be found in [45]. More applications are given in [47].

1 Introduction

In contrast to modal logic and most approaches for specifying dynamic systems (see, e.g., [1, 20]) we propose a one-tiered framework that admits the specification of "static" data types and "dynamic" transition systems within a uniform logic. In modal logic, state transitions are interpreted on a higher level that does not interfere with the structure of individual states. Swinging types regard states as hidden objects, transition labels as visible data and transition relations as *dynamic predicates*. The behavioral identity of a hidden object depends on functional or relational *observers*, in other approaches also called selectors, accessors, attributes, inquiry operations, methods, mutators, destructors, etc. In functional approaches, behavioral equality usually comes as a sort of contextual equivalence, in modal logic as bisimilarity. In both cases behavioral equality is dual to structural equality insofar as the former is the least and the latter the greatest relations satisfying certain compatibility axioms. Swinging types admit all ways of specifying a behavioral equality, be they functional, relational or "transitional", i.e., determined by dynamic predicates. The latter case motivates the introduction of *weak congruences* that are compatible with static predicates, but only *zigzag compatible* with dynamic ones.

A swinging specification starts out from constructors for building up both visible and hidden data domains. Visible domains precede the hidden ones. A visible domain is characterized by the coincidence of its structural with its behavioral equality. Constructors of visible data are not allowed to have hidden arguments. This ensures that the theory of a hidden type is consistent w.r.t. its visible subtype. A swinging specification need not have hidden sorts, but if there are hidden sorts, there must also be visible ones. Otherwise hidden objects cannot be distinguished from each other. Formally, each hidden domain must be equipped with at least one functional observer that maps to a visible sort or one relational observer, regarded as a function that maps to the visible domain of truth values. Otherwise the hidden domain collapses because its behavioral equality identifies all its elements.

On top of the constructors, a swinging specification defines functions and *μ-predicates* in terms of Horn axioms that represent functional-logic programs or transition system specifications. μ-predicates are interpreted as the least solutions of their axioms. μ-predicates are often existential properties such as liveness or reachability. Roughly said, all inductively definable properties are μ-predicates. Hence structural equalities are μ-predicates. μ-predicates dealing with "infinite" objects are often "limits" of conditions on the objects' finite approximations. *ν-predicates* are usually complements of μ-predicates. They represent universal properties, and, if they cannot be turned into equivalent μ-predicates, they often express aspects of behavior "in the infinity", such as safety and invariance

conditions on state sequences. Formally, ν-predicates are specified in terms of *co-Horn clauses* and interpreted as the greatest solutions of their axioms. Behavioral equalities are ν-predicates. Above the ν-predicates, a swinging type may have further μ-predicates whose axioms are *generalized Horn clauses*. As co-Horn clauses may involve existential quantifiers in the conclusion, generalized Horn clauses may involve universal quantifiers in the premise. Since these quantifiers may violate the continuity of the consequence operators induced by the axioms, we provide a continuity criterion that generalizes the notion of *image finiteness* from transition systems to arbitrary goals.

The notions "μ-predicate" and "ν-predicate" stem from modal logic's μ-calculus (cf., e.g., [36, 54]) and relational fixpoint semantics (cf., e.g., [28]). The least and greatest fixpoints of state set operators used to define alternation-free μ-formulas can be translated directly into swinging specifications of μ- resp. ν-predicates (see Section 2).

Besides modal logic, swinging types integrate concepts, methods and results from many other formal approaches to system specification and verification. First of all, there is final-semantics approach to data types that was introduced for modelling *permutative types* such as finite sets, multisets and arrays with a finite domain (cf., e.g., [21, 55, 35]). [23, 25] extended it to the hidden-type approach that also covers object-oriented—though purely functional—specifications. From *dynamic data types* we adopt the specification of labelled transition systems (LTS) as ternary predicates (cf. [8, 6, 3]). *Stratified* logic programs with *stable* or *perfect* models provide ideas for constructing swinging types hierarchically (cf. [2]).

Coinductive function definitions in category theory [51, 34] and formats of transition system specifications [29] led us to the criterion of *coinductivity* for the behavioral consistency of a swinging type (see Section 6). Given a suitable functor F, category theorists call a function to be defined by coinduction if the definition is derived from the unique morphism that maps an F-coalgebra to the final F-coalgebra. This dualizes the category-theoretic notion of a definition by induction that is derived from the unique morphism that maps the initial F-algebra to an F-algebra. Initial F-algebras and final F-coalgebras are isomorphisms that are composed of the constructors and destructors of the type described by F. The connections between swinging types and the category-theoretic approach to data types is treated in detail in [47]. Besides the notion of coinductivity the category-theoretic approach yields the insight that not only visible, but also hidden types have constructors. On the other hand, it is purely functional and thus does not contribute to the axiomatic specification of predicates, in particular dynamic ones. Here modal logic and process algebra provide more inspirations.

The rewriting-oriented criteria developed in [42, 43, 44] are fully applicable to swinging types. This provides the basis for ensuring that a swinging type is *functional*, i.e. each of its ground terms is structurally equivalent to a unique normal form (= term consisting of constructors). This seems to exclude the specification of partial functions. However, partiality can always be simulated by introducing sum sorts that comprise "defined" and "undefined" values such as exceptions, error messages, etc. (see, e.g., [22] and the *exception monad* of [39] as used in [47]). Moreover, strong equality turns out to be a behavioral equality induced by a destructor that identifies exceptions, and even arbitrary partial-recursive functions can be specified by axioms of a swinging type (cf. [47], Sect. 6). Each functional specification can be transformed into an equivalent relational one whose only functions are constructors, while defined functions are transformed into corresponding input-output relations. This fact is crucial for the correctness of applying one of the main proof rules for swinging types, namely fixpoint induction, not only to (μ-)predicates, but also to defined functions. A functional specification can also be extended systematically by axioms for the complements of its structural equalities, in

other words, axioms for inequalities. This entails the correctness of practically indispensable proof rules such as *term splitting* and *clash*. The complements of non-equality predicates are accomplished by simply negating axioms (see Section 4).

The axioms for behavioral equalities are determined by those defined functions, static or dynamic predicates that are declared as *destructors*, *separators* and *transition predicates*, respectively, altogether called *observers*. Observational specifications in the sense of [14, 33, 13] and behavioral or hidden ones in the sense of [25, 52] deal exclusively with destructors (called attributes/methods in [25]). λ-calculi, process logics and dynamic data types are based on labelled transition systems (LTS), i.e. transition predicates. On the one hand, only [8] and [17] regard an LTS as a predicate of a many-sorted specification. On the other hand, the dynamic-type approach lacks specification and proof methods that are as powerful as those invented in process algebra [9] and modal logics for proving properties of LTS ("model checking"). But the dynamic-type approach keeps to first-order logic, while the modal-logic and process-algebra reasoning about processes and LTS leaves the structure of individual states out of its discourse.

Similarly to functional-logic programs and transition system specifications, the axioms of a swinging type represent more or less inductive definitions of (defined) functions or (μ-)predicates on constructors. This is necessary for ensuring that the specification is functional. *Coinductive axioms*, on the other hand, guarantee that the specification is *behaviorally consistent* (see below). For instance, both the visible type of (finite) lists and the hidden type of (infinite) streams have a constructor append-to-the-left, denoted by $_ :: _ : entry \times list \to list$ and $_ \& _ : entry \times stream \to stream$, respectively. In both cases, there are defined functions $head$ and $tail$, specified by the same axioms:

$$\begin{array}{ll} head(x :: L) \equiv x & tail(x :: L) \equiv L \\ head(x \& s) \equiv x & tail(x \& s) \equiv s \end{array}$$

Obviously, these equations are part of inductive definitions of $head$ and $tail$ and thus part of a functional list resp. stream specification. *Co*inductivity, however, is a requirement to axioms for hidden symbols only and thus may or not hold only for the last two equations. Indeed, they are coinductive because we have declared $head$ and $tail$ as observers. In this simple case it is quite easy to conclude that behavioral stream equality, say $\sim$, is compatible with all involved functions, i.e. $head$, $tail$ and $\&$. Declaring $head$ and $tail$ as observers means to axiomatize $\sim$ as follows:

$$s \sim s' \quad \Rightarrow \quad head(s) \equiv head(s') \wedge tail(s) \sim tail(s'). \tag{1}$$

(1) *is* the compatibility of $\sim$ with $head$ and $tail$. But $\sim$ is also compatible with $\&$ because $\sim$ denotes the greatest solution of (1) and thus $s \sim s'$ holds true *if and only if* the conclusion of (1) holds true. The argument would fail if $head$ and $tail$ were not declared as observers. Indeed, the coinductivity requirement to axioms for "non-observing" symbols are more restrictive (see Section 6). For instance, suppose that the list specification is extended by a hidden sort bag for finite multisets, the defined function $card : bag \times entry \to nat$ returning the number of occurrences of an element in a bag is declared as an observer and the embedding $mkbag : list \to bag$ of lists into bags is the only bag constructor. Then there is a defined function $chooselist : list \to bag$ specified inductively by the axiom

$$chooselist(bag(L)) \equiv L. \tag{2}$$

Declaring $chooselist$ as an observer would lead to the axiom

$$b \sim b' \quad \Rightarrow \quad chooselist(b) \equiv chooselist(b')$$

for behavioral bag equality, which does not comply with our intuition about this equality. Hence *chooselist* cannot be an observer and thus—as the reader of Section 6 will confirm—(2) is not coinductive. Indeed, if behavioral bag equality were compatible with *chooselist*, it would coincide with list equality!

An "LTS-inspired" stream specification replaces the functional observers $head$ and $tail$ by a transition predicate $_ \stackrel{_}{\longrightarrow} _ : stream \times entry \times stream$, the axioms for $head$ and $tail$ by $x\&s \stackrel{x}{\longrightarrow} s$ and (1) by:

$$\begin{aligned} s \sim s' \quad &\Rightarrow \quad (s \stackrel{x}{\longrightarrow} t \quad \Rightarrow \quad \exists t' : (s' \stackrel{x}{\longrightarrow} t' \wedge t \sim t')), \\ s \sim s' \quad &\Rightarrow \quad (s' \stackrel{x}{\longrightarrow} t' \quad \Rightarrow \quad \exists t : (s \stackrel{x}{\longrightarrow} t \wedge t \sim t')). \end{aligned}$$

The syntax of a swinging type leads directly to its *Herbrand model*, $Her(SP)$, which is a pure term model and thus interprets both structural equalities and behavioral ones as term relations, called *structural* and *behavioral SP-equivalence*, respectively, denoted by $\equiv_{SP}$ resp. $\sim_{SP}$. The *initial SP-model*, $Ini(SP)$, is the quotient of $Her(SP)$ by structural SP-equivalence, the *final SP-model*, $Fin(SP)$, is the quotient of $Her(SP)$ by behavioral SP-equivalence. The latter deviates from other final-semantics approaches where the final model comes as a quotient of the initial one. In fact, $\equiv_{SP}$ is included in $\sim_{SP}$ and thus some quotient of $Ini(SP)$ is isomorphic to $Fin(SP)$. However, the theory of the final model is easier to handle if one constructs it as a quotient of the Herbarnd model.

The standard axioms for structural equalities render $\equiv_{SP}$ a congruence relation. The algebraist likes congruences because they admit the construction of quotient models. The theorem prover is less keen on new models, but on the correctness of term replacement w.r.t. an equivalence relation, and this is guaranteed *for all first-order formulas* only if the relation is a congruence, in other words, first-order formulas are *congruence invariant*:

$$t \equiv_{SP} t' \quad \Rightarrow \quad \text{for all first-order formulas } \varphi : Her(SP) \models \varphi(t) \Leftrightarrow \varphi(t'). \tag{3}$$

Behavioral SP-equivalence is not a congruence, but a *weak congruence*. Weak congruences are compatible with functions and *static* predicates, but only *zigzag compatible* with *dynamic* predicates. Roughly said, they are bisimulations, generalized to arbitrary dynamic predicates. Modal logic provided us the idea of possible classes of first-order formulas whose elements are *weak-congruence invariant*:

$$t \sim_{SP} t' \quad \Rightarrow \quad \text{for all } \textit{poly-modal} \text{ formulas } \varphi : Her(SP) \models \varphi(t) \Leftrightarrow \varphi(t'). \tag{4}$$

We introduce three classes of modal first-order formulas. Those called **modal** formulas are the results of direct translations of modal-logic sentences into predicate logic. Such translations—whose images are also called *modal* or *guarded fragments* of first-order logic—have been studied by, e.g., Ohlbach [40], Bergstra and van Benthem [11, 12]. The main idea is to "internalize" the "frame" or LTS, which determines the interpretation of modal-logic sentences, as a binary, or if the LTS is labelled, ternary predicate (see Section 2). Modal formulas have a single ("state") variable and can be shown to be *bisimulation invariant* (see below). The greater class of **poly-modal** formulas admits several variables, but restricts (analogously to modal formulas) the "target term" of each dynamic-predicate occurrence to an existentially quantified variable. A **weakly modal** formula may also have *free* variables as target terms, which comprise the *output* of the formula (see Section 2). Weakly modal formulas with empty output are poly-modal. For guaranteeing that the final model of a swinging specification satisfies its axioms, the premises of the Horn axioms must be weakly modal, while the conclusions of the co-Horn axioms must be poly-modal (see Section 3).

There are differences between van Benthem's modal fragment and poly-modal formulas that forbid the direct application of his results to swinging types. [12] translates *propositional* modal logic and thus formulas of the modal fragment are built over a one-sorted signature with only unary (static) and binary (dynamic) predicates, while we start out from a many-sorted signature with predicates of various arities. Nevertheless, even poly-modal formulas enjoy a *Hennessy-Milner theorem*[1], i.e. (4) together with the converse:

$$t \sim_{SP} t' \quad \Leftarrow \quad \text{for all } \textit{poly-modal} \text{ formulas } \varphi : Her(SP) \models \varphi(t) \Leftrightarrow \varphi(t'). \tag{5}$$

This is trivially valid because $\sim_{SP}$ is reflexive and $t \sim x$ is a poly-modal formula.

In the dynamic-data-type approach of [4], the Hennessy-Milner Theorem has been generalized to a class of *observational formulas* with *patterns of experiments*. They somewhat resemble poly-modal formulas, but are built upon a fixed, rather unstructured interpretation of their visible components.

A structure A interpreting the signature of a swinging specification SP is *behaviorally SP-consistent* if A interprets behavioral equality as a weak congruence and if the quotient of A by that weak congruence satisfies the axioms of SP. SP itself is *behaviorally consistent* if the Herbrand model of SP interprets behavioral equality as a weak congruence. The interpretation is denoted by $\sim_{SP}$, the quotient is called the final SP-model (see above). The modality assumptions on the axioms of SP (see above) imply that each model of SP with a weakly congruent interpretation of behavioral equality is behaviorally SP-consistent (Thm. 3.3(b)). Hence the final model is really a model if $\sim_{SP}$ is a weak congruence. This justifies the notion of behavioral consistency and shows the significance of syntactic criteria for this property.

Swinging types "swing" between many poles: between visible and hidden domains, between several states (= individual hidden objects), between functions declared as constructors and those used as defined functions, between structural and behavioral equalities, between functional-logic programs for functions and static predicates and "transitional programs" for dynamic predicates, between μ- and ν-predicates.

Section 2 and 3 sum up syntactical resp. semantical definitions and results associated with swinging specifications. Section 4 presents proof rules and the main criteria for behavioral consistency. Three non-trivial specifications are presented: bank accounts (2.4), infinite streams (2.5) and arbitrary streams (4.9). More applications can be found in [47].

2 Syntax

We assume familiarity with the basic notions of many-sorted logic with equality (cf., e.g., [27, 19, 56]). As has been shown by, e.g., [24, 38, 41, 42, 44], this logic admits presenting and verifying not only primitive data types with first-order functions, but also generic types with almost all features of current functional-logic specification or programming languages.

For any expression (term, formula, etc.) e, **var(e)** denotes the set of all variables occurring in e. e is **ground** if $var(e)$ is empty. **e(t)** denotes an expression that includes the (tuple of) subexpression(s) t, while **e[t/u]** stands for e with t substituted for u.

Swinging signatures mainly distinguish between visible and hidden sorts, constructors and defined functions, μ- and ν-predicates and static and dynamic predicates. These sets of symbols cover struc-

[1] For the original modal-logic version, see [32], Thm. 2.2, or [54], Thms. 5.3.2 and 5.3.3.

tural as well as behavioral equalities and the observers that determine the latter. The distinctions were motivated intuitively in Section 1. Further more technical reasons can only be given after the signatures are equipped with axioms (see Def. 2.3).

Definition 2.1 (signatures, terms and atoms) A **signature** $\Sigma = (S, F, P)$ consists of a set S of **sorts**, an S^+-sorted set F of **function symbols** and an S^*-sorted set P of **predicates** that splits into a set μP of **μ-predicates** and a set νP of **ν-predicates**. s, s', etc. stand for single sorts, w for sort sequences. A function symbol $f \in F_{\Sigma,ws}$ is written as $f : w \to s$ and a predicate $r \in P_{\Sigma,w}$ as $r : w$. For all $s \in S$, μP implicitly includes the **(structural) equality (predicate)** $\equiv_s : ss$.[2] Σ is **swinging** if the following conditions hold true:

- S splits into a set $visS$ of **visible sorts** and a set $hidS$ of **hidden sorts**.
- F splits into a set of **constructors** and a set DF of **defined functions**.
- For all $s \in hidS$, DF includes a set of **destructors** $f : sw \to s'$. If $s' \in visS$, then f is an **attribute**. Otherwise f is a **method**.
- For all $s \in hidS$, μP includes a set of **separators** $r : sw$ and a set of **transition predicates** $\delta : sws'$.
- For all $s \in S$, νP implicitly contains the **behavioral equality (predicate)** $\sim_s : ss$.

A function symbol $f : w \to s$ is **visible** if $ws \in visS^+$. f is **hidden** if f is not visible. For all constructors $c : w \to s$, $s \in visS$ implies $w \in visS^*$. A predicate $r : w$ is **logical** if r is not an equality predicate. r is **visible** if $w \in visS^+$. r is **hidden** if r is not visible. Structural equalities are μ-predicates. Destructors, separators and transition predicates are called **observers**.

- Visible equality predicates, separators and ν-predicates belong to the set $statP$ of **static predicates**.
- Transition and hidden equality predicates belong to the set $dynP$ of **dynamic predicates**, which are always μ-predicates.

Each predicate is static or dynamic. Only visible equality predicates are static *and* dynamic.

Let X be a set of S-sorted variables. $T_\Sigma(X)$ and T_Σ denote the S-sorted sets of Σ-terms and ground Σ-terms, respectively, which are defined as usual. A **Σ-normal form** is a Σ-term that consists of constructors and variables. $NF_\Sigma(X)$ and NF_Σ denote the S-sorted sets of Σ-normal forms and ground Σ-normal forms, respectively. $t \in T_\Sigma(X)_s$ is **visible** resp. **hidden** if s is visible resp. hidden. t is **unary** if $var(t)$ is a singleton.

Given $r : w \in P$ and $t \in T_\Sigma(X)_w$, $r(t)$ is a **Σ-atom**. If r is a μ-predicate, then $r(t)$ is a **μ-atom**. Otherwise $r(t)$ is a **ν-atom**. $r(t)$ is an **equation** if r is an equality predicate. An equation between term tuples t and t' stands for the conjunction of the equations between corresponding components of t resp. t'. An atom $r(t)$ is **logical**, **visible**, **hidden**, **static** or **dynamic** if r is visible, hidden, static or dynamic, respectively. ❑

Equality and behavioral equalities will not be listed explicitly in signatures examples. Behavioral equalities are specified via observers (see Def. 2.3). Each function symbol $f : s \to s'$ is also regarded as a constructor constant of the functional sort $s \to s'$. Functional sorts are hidden. $s \to s'$ has the (implicit) observer $apply : (s \to s') \times s \to s'$. Non-constant functional-sort constructors such as

[2] We use "$\equiv$" for distinguishing the *symbol* for structural equality from semantical identity, which is denoted by "=".

function composition are specified in terms of *apply*:

$$apply(f \circ g, x) \equiv apply(f, apply(g, x))$$

or, in more readable notation,

$$(f \circ g)(x) \equiv f(g(x)).$$

Hence, semantically, the behavioral equality for a functional sort coincides with *extensional equality*.

An S-sorted function $\sigma : X \to T_\Sigma(X)$ is called a **substitution**. The **domain of** σ, $dom(\sigma)$, is the set of all variables x with $x\sigma \neq x$. σ_Y denotes the restriction of σ that is defined by $x\sigma_Y = x\sigma$ for all $x \in Y$ and $x\sigma_Y = x$ for all $x \in X \setminus Y$. If σ maps each variable of $dom(\sigma)$ to a term in some given set T of terms, we write $\sigma : X \to T$ in order to indicate that σ satisfies $\sigma(dom(X)) \subseteq T$. The **instance** $t\sigma$ of a term or atom t **by** σ is obtained from t by replacing each variable x by $x\sigma$.

Given a parameterized specification SP with parameter $PAR = (\Sigma, AX)$ with signature Σ and a signature morphism $\sigma : \Sigma \to \Sigma'$, let $domain(\sigma) =_{def} \{s \in \Sigma \mid \sigma(s) \neq s\} = \{s_1, \ldots, s_n\}$. The specification $SP[\sigma]$, usually written as

$$SP[s_1 \mapsto s'_1, \ldots, s_n \mapsto s'_n],$$

is called the **actualization of** SP **along** σ and obtained from SP by replacing all (!) occurrences in SP of $s \in domain(\sigma)$ by $\sigma(s)$ and by deleting the axioms of PAR.

Definition 2.2 (Σ-formulas) A **Σ-goal** is a finite conjunction of Σ-atoms. Given a finite subset Y of X and goals G and H, the formula $\exists YG$ is an **existential goal** and the formula $\forall Y(G \Rightarrow H)$ is a **universal goal**. The empty conjunction is called the **empty goal** and denoted by $\emptyset$ or $TRUE$. The empty disjunction is denoted by $FALSE$.

Let G be a goal, r be a logical predicate and f be a defined function. A formula of the form $r(t) \Leftarrow G$ resp. $f(t) \equiv u \Leftarrow G$ is a **Horn clause for** r resp. f. Given a finite disjunction φ of existential goals, $r(t) \Rightarrow (G \Rightarrow \varphi)$ is a **co-Horn clause for** r. Given a finite conjunction φ of universal goals, $r(t) \Leftarrow \varphi$ is a **generalized Horn clause for** r. The formulas $\varphi \Leftarrow TRUE$ and $TRUE \Rightarrow \varphi$ are identified with φ.

Suppose that Σ is swinging. A (first-order) formula φ is **poly-modal** if φ is equivalent[3] to a formula built up by the following rules:

- A static atom is poly-modal.
- If φ and ψ are poly-modal, then $\neg\varphi$, $\varphi \wedge \psi$ and for all $x \in X$, $\exists x\varphi$ are poly-modal.
- If $\delta(t, x)$ is a dynamic atom and φ is poly-modal, then for all $x \in X \setminus var(t)$, $\exists x(\delta(t, x) \wedge \varphi)$ is poly-modal.

A formula φ is **weakly modal with output out**$(\varphi) \subseteq X$ if φ is equivalent to a formula built up by the following rules:

- A poly-modal formula is weakly modal with output $\emptyset$.
- A dynamic atom $\delta(t, x)$ with $x \in X \setminus var(t)$ is weakly modal with output $\{x\}$.
- If φ and ψ are weakly modal with disjoint outputs Y resp. Z, then $\varphi \wedge \psi$ is weakly modal with output $Y \cup Z$.

[3] See Def. 3.1.

- If φ is weakly modal with output Y, then for all $x \in X$, $\exists x\varphi$ is weakly modal with output $Y \setminus \{x\}$. ❑

Poly-modal formulas are weakly modal. The output of a weakly modal formula consists of *free* variables.

Modal formulas arise from the translation of modal into predicate logic. Given a transition relation $\rightarrow$ and propositions p representing state sets, assertions of the form "the state x satisfies the modal-logic formula φ" can be compiled into modal formulas in the sense of Def. 2.2 as follows:

$$
\begin{array}{lcl}
compile(x \models p) & = & r_p(x) \quad \text{for all propositions } p \\
compile(x \models \varphi \wedge \psi) & = & compile(x \models \varphi) \wedge compile(x \models \psi) \\
compile(x \models \varphi \vee \psi) & = & compile(x \models \varphi) \vee compile(x \models \psi) \\
compile(x \models \langle . \rangle \varphi) & = & \exists y(x \rightarrow y \wedge compile(y \models \varphi)) \\
compile(x \models [.]\varphi) & = & \forall y(x \rightarrow y \Rightarrow compile(y \models \varphi)) \\
compile(x \models \mu p.(\varphi_1 \vee \cdots \vee \varphi_n)) & = & r_p(x) \text{ where } r_p \text{ is specified by the axioms} \\
 & & \quad r_p(x) \Leftarrow compile(x \models \varphi_1),\ \ldots,\ r_p(x) \Leftarrow compile(x \models \varphi_n) \\
compile(x \models \nu p.(\varphi_1 \wedge \cdots \wedge \varphi_n)) & = & r_p(x) \text{ where } r_p \text{ is specified by the axioms} \\
 & & \quad r_p(x) \Rightarrow compile(x \models \varphi_1),\ \ldots,\ r_p(x) \Rightarrow compile(x \models \varphi_n)
\end{array}
$$

A swinging signature Σ is implicitly associated with the set $\mathbf{EQ}_\Sigma$ of **congruence axioms for** Σ, given by Horn clauses expressing reflexivity, symmetry and compatibility with the function symbols and μ-predicates of Σ, together with a co-Horn axiom:

$$r(x_1, \ldots, x_n) \quad \Rightarrow \quad ((x_1 \equiv y_1 \wedge \cdots \wedge x_n \equiv y_n) \ \Rightarrow \ r(y_1, \ldots, y_n))$$

for each ν-predicate $r \in \Sigma$.

Definition 2.3 A **specification** $SP = (\Sigma, AX)$ consists of a signature Σ and a set AX of first-order Σ-formulas, called the **axioms** of SP. SP is **swinging** if Σ is swinging and has three subspecifications

$$visSP = (vis\Sigma, visAX) \subseteq hidSP = (hid\Sigma, hidAX) \subseteq \nu SP = (\nu\Sigma, \nu AX) \subseteq SP$$

such that $hidAX$ and νAX implicitly include the Horn resp. co-Horn clauses among the congruence axioms for Σ and the following conditions hold true:

(1)	**The visible level** $visSP$
	consists of visible sorts and visible constructors, a set DF of defined functions, a set P of static μ-predicates, Horn axioms $f(t) \equiv u \Leftarrow \varphi$ for DF and $r(t) \Leftarrow \varphi$ for P such that (a) r is logical, t is a tuple of normal forms and $var(u) \subseteq var(t, \varphi)$.
(2)	**The hidden level** $hidSP \setminus visSP$
	consists of hidden sorts and hidden constructors, a set DF of defined functions, a set P of static μ-predicates, a set DP of dynamic predicates and Horn axioms $f(t) \equiv u \Leftarrow \varphi$ for DF, $r(t) \Leftarrow \varphi$ for P and $\delta(t, u) \Leftarrow \varphi$ for DP such that (a) holds true and (b) φ is weakly modal such that $var(t) \cap out(\varphi) = \emptyset$.
(3)	**The ν-level** $\nu SP \setminus hidSP$
	consists of a set P of ν-predicates (including the behavioral equalities) and co-Horn axioms $r(t) \Rightarrow (G \Rightarrow \varphi)$ for P such that (a) holds true, $G \Rightarrow \varphi$ is poly-modal and G is a goal over $hidSP$. The axioms for behavioral equalities are called **behavior axioms** and read as follows:
	$x \sim_s y \;\Rightarrow\; x \equiv y$ — for all visible sorts $s \in \Sigma$
	$x \sim_s y \;\Rightarrow\; f(x, z) \sim f(y, z)$ — for all destructors $f : sw \to s' \in \Sigma$
	$x \sim_s y \;\Rightarrow\; (r(x, z) \;\Rightarrow\; r(y, z))$ — for all separators $r : sw \in \Sigma$
	$x \sim_s y \;\Rightarrow\; (r(y, z) \;\Rightarrow\; r(x, z))$ — for all separators $r : sw \in \Sigma$
	$x \sim_s y \;\Rightarrow\; (\delta(x, z, x') \;\Rightarrow\; \exists y'(\delta(y, z, y') \wedge x' \sim y'))$ — for all transition predicates $\delta : sws' \in \Sigma$
	$x \sim_s y \;\Rightarrow\; (\delta(y, z, y') \;\Rightarrow\; \exists x'(\delta(x, z, x') \wedge x' \sim y'))$ — for all transition predicates $\delta : sws' \in \Sigma$
(4)	**The μ-level** $SP \setminus \nu SP$
	consists of a set P of static μ-predicates, a set DP of dynamic predicates and generalized Horn axioms $r(t) \Leftarrow \varphi$ for P and $\delta(t, u) \Leftarrow \varphi$ for DP such that (a) and (b) hold true and for all universal goals $\forall Y(G \Rightarrow H)$ of φ, G is a goal over νSP.

If $SP = visSP$, then SP is **visible**. ❑

Together with the condition of Def. 2.1 that all hidden constructors have hidden ranges the levels of a swinging specification entail a hierarchy of their Herbrand models. Excluding hidden constructors with visible ranges is also motivated intuitively by the viewpoint that objects with hidden components cannot be visible. Hidden constructors with visible ranges represent certain "contexts" and thus are better modelled by observers that make contexts visible. Visible normal forms t of a functional specification are uniquely decomposable: all ground normal forms that are equivalent to a ground instance of t are themselves ground instances of t. Hidden normal forms enjoying the same property with respect to *behavioral* equivalence are *strongly normal* (cf. Def. 4.7). For instance, the stream term $x\&s$ (cf. Ex. 2.5) is strongly normal. A hidden constructor $c : w \to s$ with visible range s can be replaced by a constructor $c' : w \to s'$ such that s' is hidden and $c(t)$ is uniquely decomposable w.r.t. structural s-equivalence iff $c'(t)$ is uniquely decomposable w.r.t. behavioral s'-equivalence.

Condition 2.3(a) does not only appeal to the syntax of functional-logic programs, but also has a number of consequences, which facilitate the reasoning about data types significantly. At first, the condition admits a simple proof that SP is complete (cf. Def. 3.4): for all defined functions f of SP, $Her(SP)$ (cf. Def. 2.3) must satisfy $Def(x) \Rightarrow Def(f(x))$ where Def is the standard μ-predicate specified by a Horn axiom

$$Def(c(x_1, \ldots, x_n)) \quad \Leftarrow \quad Def(x_1) \wedge \cdots \wedge Def(x_n)$$

for each constructor c of SP. Secondly, the condition is an essential part of simple confluence, consistency and thus functionality criteria such as [44], 10.46 und 10.48. Thirdly, the condition ensures the soundness of the most frequently used deduction rules *atom unfolding* and *term unfolding* (see Section 4).

The modality assumptions on the axioms of the hidden, μ- and ν-level of SP are essential for the behavioral consistency of SP-models (cf. Def. 3.1). They restrict the occurrences of dynamic predicates in the axioms, but this restriction is much weaker than previous similarly motivated conditions such as the non-existence of hidden equations in Horn axiom premises (cf., e.g., [15], Cor. 4; [56], Thm. 5.4.5; [14], Ex. 3.24). Condition 2.3(b) also reveals the technical reason for the distinction between static and dynamic predicates. While static predicates can often be transformed easily into Boolean functions because all their arguments have a sort of "input mode", a dynamic predicate has at least one argument (usually the last), which takes up output that is produced when an axiom for the predicate is "called". In fact, a static predicate r may also have output arguments, provided that these are not produced by a dynamic predicate δ in the premise of an axiom for r. For instance, an axiom of the form $r(t, u(x)) \Leftarrow \delta(v, x)$ satisfies 2.3(b) only if r is dynamic.

Since the behavior axioms are completely determined by the observers, they are omitted in examples. The separation of the ν-level from the μ-level prevents a μ-predicate and a ν-predicate from being specified in a mutually-recursive way. Such *alternating fixpoints* were difficult to handle and are actually not needed in practice, even for specifying modal operators (cf. [45], Ex. 2.7). The hierarchy assumption in 2.3(3): G is a goal over $hidSP$, and the corresponding one in 2.3(4): G is a goal over νP, are essential for the monotonicity of the consequence operators that build up the Herbrand model.

A **parameterized specification** SP contains **parameter specifications** that consist of **empty sorts**, defined functions, predicates and first-order axioms. A sort s of SP is empty if SP does not contain constructors of type $w \to s$. Empty sorts correspond to the type variables of polymorphic type expressions. Consequently, structured sort symbols such as $list(entry)$ denote polymorphic types. The equation $list = list(entry)$ declares $list$ as a short notation for $list(entry)$. We use CASL notations for structuring specifications (cf. [16]): `then` denotes the **extension** operator that combines a specification SP with additional signature symbols and axioms, `and` builds the **union** of specifications and identifies synonymous (and equally-typed) symbols of the argument specifications.

Example 2.4 The following specification is a swinging version of a favorite example for demonstrating formal approaches to the *object-oriented* or *state-based* specification of data types (cf., e.g., [24, 37]). We presuppose a visible specification NAT of natural number arithmetic.

ENTRY

`sorts`	$entry$	
`preds`	$_ \not\equiv _ : entry \times entry$	
`vars`	$x, y : entry$	
`axioms`	$x \equiv y \vee x \not\equiv y$	$x \not\equiv y \Leftrightarrow \neg(x \equiv y)$

LIST = ENTRY `then`

`vissorts`	$list = list(entry)$
`constructs`	$nil :\to list$
	$_ :: _ : entry \times list \to list$

TRANSACTION = ENTRY `and` NAT `then`

vissorts	$transaction$
constructs	$from, to : nat \times entry \to transaction$
static μ**-preds**	$_ \not\equiv _ : transaction \times transaction$
vars	$x, y : entry \quad m, n : nat$
Horn axioms	standard inequality axioms (see Section 4)

ACC = TRANSACTION **and** LIST[$entry \mapsto transaction$] **then**

hidsorts	acc
constructs	$new : entry \to acc$
	$credit, debit : nat \times acc \to acc$
	$send : acc \times nat \times entry \to acc$
	$receive : entry \times nat \times acc \to acc$
destructs	$name : acc \to entry$
	$bal : acc \to nat$
	$record : acc \to list(transaction)$
static μ**-preds**	$_ \not\equiv _ : acc \times acc$
vars	$x : entry \quad n : nat \quad a, b : acc$
Horn axioms	$name(new(x)) \equiv x$
	$bal(new(x)) \equiv 0$
	$record(new(x)) \equiv nil$
	$name(credit(n, a)) \equiv name(a)$
	$bal(credit(n, a)) \equiv bal(a) + n$
	$record(credit(n, a)) \equiv record(a)$
	$name(debit(n, a)) \equiv name(a)$
	$bal(debit(n, a)) \equiv bal(a) - n$
	$record(debit(n, a)) \equiv record(a)$
	$name(send(a, n, x)) \equiv name(a)$
	$bal(send(a, n, x)) \equiv bal(a) - n$
	$record(send(a, n, x)) \equiv to(n, x) :: record(a)$
	$name(receive(x, n, a)) \equiv name(a)$
	$bal(receive(x, n, a)) \equiv bal(a) + n$
	$record(receive(x, n, a)) \equiv from(n, x) :: record(a)$
	standard inequality axioms (see Section 4)

ACCOUNT = ACC **and** LIST[$entry \mapsto transaction$] **then**

vissorts	$com \quad message$
hidsorts	set
constructs	$\emptyset :\to set$
	$insert : acc \times set \to set$
	$_ \cup _ : set \times set \to set$
	$credit, debit, bal? : nat \to com$
	$send : nat \times entry \to com$
	$receive : entry \times nat \to com$
	$_._ : entry \times com \to message$
	$create : entry \to message$
	$transfer : entry \times nat \times entry \to message$

	$_;_ : message \times message \to message$
separators	$_ \in _ : acc \times set$
	$_ \notin _ : acc \times set$
dynamic preds	$_ \stackrel{-}{\longrightarrow} _ : acc \times com \times acc$
transpreds	$_ \stackrel{-}{\Longrightarrow} _ : set \times message \times set$
static μ-preds	$_ \notin _ : entry \times set$
ν-preds	$_ \approx _ : message \times message$
vars	$x, y : entry \quad m, n : nat \quad a, b : acc \quad as, bs, as', bs' : set \quad c : com \quad m, m' : message$
Horn axioms	$x \in insert(x, s)$ $\qquad$ $x \in s \cup s' \Leftarrow x \in s$
	$x \in insert(y, s) \Leftarrow x \in s$ $\qquad$ $x \in s \cup s' \Leftarrow x \in s'$
	$x \notin \emptyset$ $\qquad$ $x \notin s \cup s' \Leftarrow x \notin s \wedge x \notin s'$
	$x \notin insert(y, s) \Leftarrow x \not\equiv y \wedge x \notin s$
	$a \stackrel{credit(n)}{\longrightarrow} credit(n, a)$
	$a \stackrel{debit(n)}{\longrightarrow} debit(n, a) \Leftarrow n \leq bal(a)$
	$a \stackrel{send(n,y)}{\longrightarrow} send(a, n, y) \Leftarrow n \leq bal(a)$
	$a \stackrel{receive(x,n)}{\longrightarrow} receive(x, n, a)$
	$a \stackrel{bal?(n)}{\longrightarrow} a \Leftarrow n \equiv bal(a)$
	$as \stackrel{create(x)}{\Longrightarrow} insert(new(x), as) \Leftarrow x \notin as$
	$insert(a, as) \stackrel{name(a).c}{\Longrightarrow} insert(b, as) \Leftarrow a \stackrel{c}{\longrightarrow} b$
	$as \cup as' \stackrel{transfer(x,n,y)}{\Longrightarrow} bs \cup bs' \Leftarrow as \stackrel{x.send(n,y)}{\Longrightarrow} bs \wedge as' \stackrel{y.receive(x,n)}{\Longrightarrow} bs'$
	$as \cup as' \stackrel{transfer(x,n,y)}{\Longrightarrow} bs \cup bs' \Leftarrow as \stackrel{y.receive(x,n)}{\Longrightarrow} bs \wedge as' \stackrel{x.send(n,y)}{\Longrightarrow} bs'$
	$as \stackrel{m;m'}{\Longrightarrow} bs \Leftarrow as \stackrel{m}{\Longrightarrow} as' \wedge as' \stackrel{m'}{\Longrightarrow} bs$
	$x \notin as \Leftarrow x \equiv name(a) \wedge a \notin as$
co-Horn axioms	$m \approx m' \Rightarrow \exists\, bs, bs' : (as \stackrel{m}{\Longrightarrow} bs \wedge as \stackrel{m'}{\Longrightarrow} bs' \wedge bs \sim bs')$

ACCOUNT provides two logical dynamic predicates, one defined on single accounts, the other on sets of accounts. Commands become messages sent to accounts. Equations of the form $as : m \equiv bs$ become state transitions $as \stackrel{m}{\Longrightarrow} bs$. Behavioral equivalence of single accounts is determined by the destructors *name*, *bal* and *record* and the transition predicate $\longrightarrow : acc \times com \times acc$. Roughly said, two objects are behaviorally equivalent if they have the same *name*-, *bal*- and *record*-values and if they may send or receive the same messages. The transition predicate $\Longrightarrow : set \times message \times set$ "lifts" local account transitions to global changes of sets of accounts. Behavioral equivalence of account sets is determined by the separator $\in$ and the transition predicates $\Longrightarrow$. The set constructors *insert* and $\cup$ model concurrency, similarly to the parallelization operator | of CCS.

The command constructur *bal*? is inspired by [31], Ex. A.6, where a functional-logic program *make_account* is given that solves *constraints* (= goals) such as

$$as \stackrel{create(x);\ x.credit(100);\ x.debit(50);\ x.bal?(n)}{\Longrightarrow} bs. \tag{1}$$

For instance, if the program is called with the message sequence of (1), it actually generates an expansion of (1) into a goal that represents a solution of (1) and contains the *solved* subgoal $n \equiv 50$ (cf. [47], Section 2.1).

Explicit state structures should be distinguished from the *agents* occurring in stream and process calculi. While ACCOUNT separates states from agents (= commands), process calculi identify them: an agent is both an abstract program and the initial state the program starts out from. ❑

Example 2.5 A swinging specification of infinite sequences of *entry*-elements reads as follows:

INFSEQ = LIST `and` NAT `then`

`hidsorts`	$stream = stream(entry)$	
`constructs`	$_\&_ : entry \times stream \to stream$	
	$blink :\to stream(nat)$	
	$nats : nat \to stream$	
	$odds : stream \to stream$	
	$zip : stream \times stream \to stream$	
	$map : (entry \to entry) \times stream \to stream$	
`destructs`	$head : stream \to entry$	
	$tail : stream \to stream$	
`defuncts`	$_\#_ : list \times stream \to stream$	
	$evens : stream \to stream$	
	$firstn : nat \times stream \to list$	
	$nthtail : nat \times stream \to stream$	
`static` μ`-preds`	$exists : (entry \to bool) \times stream$	
ν`-preds`	$forall : (entry \to bool) \times stream$	
	$fair : (entry \to bool) \times stream$	
`vars`	$n : nat \quad x, y : entry \quad L : list \quad s, s' : stream$	
	$f : entry \to entry \quad g : entry \to bool$	
`Horn axioms`	$head(x\&s) \equiv x$	$tail(x\&s) \equiv s$
	$head(blink) \equiv 0$	$tail(blink) \equiv 1\&blink$
	$head(nats(n)) \equiv n$	$tail(nats(n)) \equiv nats(n+1)$
	$head(zip(s, s')) \equiv head(s)$	$tail(zip(s, s')) \equiv zip(s', tail(s))$
	$head(odds(s)) \equiv head(s)$	$tail(odds(s)) \equiv odds(tail(tail(s))$
	$head(map(f, s)) \equiv f(s)$	$tail(map(f, s)) \equiv map(f, tail(s))$
	$nil\#s \equiv s$	$(x :: L)\#s \equiv x\&(L\#s)$
	$evens(s) \equiv odds(tail(s))$	
	$firstn(0, s) \equiv nil$	
	$firstn(n+1, s) \equiv head(s) :: firstn(n, tail(s))$	
	$nthtail(0, s) \equiv s$	
	$nthtail(n+1, s) \equiv nthtail(n, tail(s))$	
	$exists(g, s) \Leftarrow g(head(s)) \equiv true$	
	$exists(g, s) \Leftarrow exists(g, tail(s))$	
`co-Horn axioms`	$forall(g, s) \Rightarrow g(head(s)) \equiv true \wedge forall(g, tail(s))$	
	$fair(g, s) \Rightarrow exists(g, s) \wedge fair(g, tail(s))$	

The following should hold in a standard model of INFSEQ. & appends an entry to a stream. *blink* denotes a stream whose elements alternate between zeros and ones. $nats(n)$ generates the stream of all numbers starting from n. $odds(s)$ returns the stream of all elements of s that have odd-numbered positions in s. *zip* merges two streams into a single stream by alternatively appending an element of one stream to an element of the other stream. # concatenates a list and a stream into a stream. *head*, *tail*, *firstn*, *nthtail*, *map*, *exists* and *forall* have the same meaning as stream functions as they have as list functions. $fair(g, s)$ holds true iff g holds true for infinitely many elements of s. ❑

3 Semantics

Definition 3.1 (semantical notions) Let $\Sigma = (S, F, P)$ be a signature. A **Σ-structure** A consists of an S-sorted set, the **carrier** of A, also denoted by A, for all $f : w \to s \in F$, a function $f^A : A_w \to A_s$, and for all $r : w \in P$, a relation $r^A \subseteq A_w$. $\overline{r} : w \in P$ is called the **complement of r w.r.t.** A if $\overline{r}^A = A_w \setminus r^A$. If P is empty, A is called a **Σ-algebra**. A is a **Herbrand structure** if for all $s \in S$, $A_s = T_{\Sigma,s}$, and for all $f : w \to s \in F$ and $t \in T_\Sigma(X)_w$, $f^A(t) = f(t)$.

Given $\approx_s : ss \in P$ for all $s \in S$, A is a **structure with $\approx$-equality** if for all $s \in S$, $\approx_s^A = \{(a, a) \mid a \in A_s\}$.

Let A and B be Σ-structures. B is **monotone w.r.t.** A if for all ground μ-atoms p, $A \models p$ implies $B \models p$, and for all ground ν-atoms q, $B \models q$ implies $A \models q$.

A $\mu\nu\Sigma$- resp. $\nu\mu\Sigma$**-homomorphism** $h : A \to B$ is an S-sorted function such that for all $f : w \to s \in F$, $h_s \circ f^A = f^B \circ h_w$, for all $r \in \mu\Sigma$ resp. $r \in \nu P$, $h(r^A) \subseteq r^B$, and for all $r \in \nu P$ resp. $r \in \mu\Sigma$, $r^B \subseteq h(r^A)$. h is a **Σ-isomorphism** if there is a $\mu\nu\Sigma$- resp. $\nu\mu\Sigma$-homomorphism $g : B \to A$ such that $g \circ h = id_A$ and $h \circ g = id_B$. A and B are **Σ-isomorphic** iff there is a Σ-isomorphism $h : A \to B$.

The interpretation of Σ-terms in a Σ-structure A depends on a (first-order) **valuation** of variables **in** A, i.e. an S-sorted function $b : X \to A$. Given a further valuation $c : X \to A$ and $Y \subseteq X$, we write $b =_Y c$ if $b(x) = c(x)$ for all $x \in X \setminus Y$. b extends to a function $b^* : T_\Sigma(X) \to A$ defined by $b^*(x) = b(x)$ for all $x \in X$ and $b^*(t) = f^A(b^*(t_1), \ldots, b^*(t_n))$ for all $t = f(t_1, \ldots, t_n) \in T_\Sigma(X)$. Given a term t with $var(t) = \{x_1, \ldots, x_n\}$, we sometimes use the function $t^A : A^n \to A$ defined by $t^A(b(x_1), \ldots, b(x_n)) = b^*(t)$. A is **reachable** if for all $a \in A$ there is $t \in T_\Sigma$ with $t^A = a$.[4]

A valuation $b : X \to A$ **solves** an atom $r(t)$ in A if $b^*(t) \in r^A$. This notion extends to first-order formulas as usual. If b solves φ in A, we write $A \models_b \varphi$. A **satisfies** or is a **model of** φ or φ is **valid** in A, written $A \models \varphi$, if all valuations in A solve φ in A. Two Σ-formulas φ and ψ are **equivalent** if all Σ-structures satisfy $\varphi \Leftrightarrow \psi$.

Let Σ be swinging. An S-sorted binary relation $\approx \subseteq A \times B$ is **compatible** with $f : w \to s \in F$ if for all $a \in A_w$ and $b \in B_w$, $a \approx b$ implies $f^A(a) \approx f^B(b)$. $\approx$ is **compatible** with $r : w \in P$ if for all $a \in A_w$ and $b \in B_w$, $a \approx b$ implies $a \in r^A$ iff $b \in r^B$. $\approx$ is **zigzag compatible** with $\delta : ws \in P$ if for all $(a, a') \in \delta^A$, $a \approx b$ implies $(b, b') \in \delta^B$ for some $b' \in B_s$ with $a' \approx b'$ and for all $(b, b') \in \delta^B$, $a \approx b$ implies $(a, a') \in \delta^A$ for some $a' \in A_s$ with $a' \approx b'$. An S-sorted equivalence relation $\approx \subseteq A \times A$ is a **Σ-congruence on** A if $\approx$ is compatible with $F \cup P$. $\approx$ is a **weak Σ-congruence on** A if $\approx$ is compatible with $F \cup statP$ and zigzag compatible with $dynP$.

Let $\approx$ be a (behavioral) Σ-congruence on A. Then the **quotient** $B = A/\!\approx$ **of** A **by** $\approx$ is the Σ-structure that interprets $s \in S$ as the quotient set $A_s/\!\approx$ and $f : w \to s \in F$ as the function $f^B : B_w \to B$ defined by $f^B([a]) = [f^A(a)]$ where $[a]$ denotes the **equivalence class** of a consisting of all $b \in A$ with $a \approx b$.[5] **nat**: $A \to B$ always denotes the **natural mapping** that sends an element a to its equivalence class $[a]$. If $\approx$ is a congruence, then B interprets $r : w \in P$ as the set of $[a] \in B_w$ with $a \in r^A$. If $\approx$ is a weak congruence, this definition is restricted to static predicates, while for all dynamic predicates $\delta : ws \in P$,

$$([a], [b]) \in \delta^B \iff_{def} \exists\, b' \approx b : (a, b') \in \delta^A.$$

[4] Each Σ-structure has a least reachable Σ-substructure (with respect to the inclusion of carriers).

[5] If $a = (a_1, \ldots, a_n)$, then $[a]$ stands for $([a_1], \ldots, [a_n])$.

Let $SP = (\Sigma, AX)$ be a specification. A Σ-structure A is an SP**-model** if A satisfies AX and EQ_Σ. **Mod(SP)** denotes the class of all SP-models. $\mathbf{Mod}_{\equiv}$**(SP)** is the class of SP-models with $\equiv$-equality.

Let Σ be swinging. A Σ-structure A is **behaviorally** SP**-consistent** if $\sim^A$ is a weak Σ-congruence and $A/\sim^A$ is an SP-model.[6] $\mathbf{Mod}_{be}$**(SP)** is the class of SP-models A such that $\sim^A$ is an equivalence relation that includes $\equiv^A$. $\mathbf{Mod}_{bc}$**(SP)** is the class of SP-models A such that $\sim^A$ is a weak Σ-congruence. $\mathbf{Mod}_{\mu\nu}$**(SP)** is the class of SP-models A that interpret all μ-predicates as the least relations and all ν-predicates as the greatest relations satisfying AX. ❑

Σ-congruences are weak Σ-congruences because the latter are reflexive.

Theorem 3.2 ([45], Thm. 3.8) *Let Σ be a swinging signature and A be a Σ-structure.*

(1) Let $\approx$ be a weak congruence on A, φ be a weakly modal formula with output Y and $b, c : X \to A$. Then $b \approx c$ and $A \models_b \varphi$ imply $A \models_{c'} \varphi$ for some c' with $b \approx c' =_Y c$.

(2) **Hennessy-Milner Theorem.** *Suppose that $\sim^A$ is a weak congruence. Then for all $b, c : X \to A$, $b \sim^A c$ iff for all poly-modal formulas φ, $A \models_b \varphi$ iff $A \models_c \varphi$.* ❑

Theorem 3.3 ([45], Thm. 3.9) *Let SP be a swinging specification and A be a Σ-structure such that $\sim^A$ is a weak Σ-congruence. Let φ be a weakly modal formula with output Y and $B = A/\sim^A$.*

(a) For all $c : X \to A$, $B \models_{nat\circ c} \varphi$ iff $A \models_{c'} \varphi$ for some c' with $c \sim^A c' =_Y c$.

(b) A is behaviorally SP-consistent. ❑

Definition 3.4 Let SP be a swinging specification and $hidSP = (\Sigma, AX)$ (cf. Def. 2.3). The **cut calculus for** SP consists of the following inference rules for deriving Horn clauses.[7]

axiom rule	$\dfrac{TRUE}{\varphi} \Downarrow$	where $\varphi \in AX \cup EQ_\Sigma$
instantiation	$\dfrac{\varphi}{\varphi\sigma} \Downarrow$	where $\sigma : X \to T_\Sigma(X)$
modus ponens	$\dfrac{p \Leftarrow \varphi, \quad \varphi}{p} \Downarrow$	
$\wedge$-introduction	$\dfrac{\varphi, \quad \psi}{\varphi \wedge \psi} \Downarrow$	

Given a formula φ, we write $SP \vdash_{cut} \varphi$ if φ is derivable with the cut calculus for SP. **(Structural)** SP**-equivalence** is the binary relation on T_Σ that is defined as follows:

$$t \equiv_{SP} t' \iff_{def} SP \vdash_{cut} t \equiv t'.$$

Given $t \in T_\Sigma$ and $u \in NF_\Sigma$, u is a **normal form of** t if t and u are SP-equivalent. SP is **complete** if each ground Σ-term has a normal form. SP is **consistent** if each two SP-equivalent ground normal forms are equal. SP is **relational** if it does not contain defined functions. SP is **functional** if it is complete and consistent. In this case **nf(t)** denotes the unique normal form of a ground term (tuple) t. ❑

Relational specifications are functional.[8]

[6] In [18], a function that is compatible with $\sim^A$ is called *behaviorally coherent*.
[7] Arrows attached to a rule indicate the direction of consequence, here with respect to all Σ-structures.
[8] No joke!

Both completeness and consistency are essential for ensuring the soundness of proof rules dealing with constructors and defined functions. Consistency calls for syntactic criteria some of which are already involved in the definition of a swinging specification. For sufficient ones, see [44, 46]. Completeness is a much simpler proof obligation: SP is complete iff for all defined functions $f : w \to s$ and $t \in NF_{\Sigma,w}$, $f(t) \equiv_{SP} u$ for some $u \in NF_\Sigma$. This can often be shown by induction on the size of t.

A term model of $hidSP$ (cf. 2.3) could be defined directly in terms of the cut calculus. Since this does not work for the ν- and μ-levels of SP, we prefer an equivalent definition that uses *consequence operators* on substructures. Roughly said, a consequence operator Φ stepwise adds valid atoms to models of a subspecification. If Φ is *monotonic*, a model of the entire specification is obtained from a fixpoint of Φ. The existence of a suitable fixpoint is ensured by the fixpoint theorem of Knaster and Tarski. Morever, if Φ is *continuous*, then Kleene's fixpoint theorem provides a stepwise construction of the fixpoint (see [45], Thm. 4.2).

Definition 3.5 Let $SP = (\Sigma, AX)$ be a specification, $SP' = (\Sigma', AX')$ be a subspecification of SP, A be a Σ'-structure and $\mathcal{C}$ be the class of Σ-structures whose Σ'-reduct agrees with A. The **$(AX \setminus AX')$-consequence operator on A**, $\Phi : \mathcal{C} \to \mathcal{C}$, is defined as follows: For all μ-predicates $r \in \Sigma \setminus \Sigma'$ and $B \in \mathcal{C}$,

$$a \in r^{\Phi(B)} \iff_{def} \exists\ (r(t) \Leftarrow \varphi) \in AX \setminus AX'\ \exists\ b : X \to A : a = b^*(t) \wedge B \models_b \varphi.$$

For all ν-predicates $r \in \Sigma \setminus \Sigma'$ and $B \in \mathcal{C}$,

$$a \in r^{\Phi(B)} \iff_{def} \forall\ (r(t) \Rightarrow \varphi) \in AX \setminus AX'\ \forall\ b : X \to A : a = b^*(t) \Rightarrow B \models_b \varphi.\ \square$$

Definition 3.6 Let Σ be a swinging signature and $\mathcal{C}$ be a class of Σ-structures, $I \in \mathcal{C}$ is **initial in** $\mathcal{C}$ if for all $A \in \mathcal{C}$ there is a unique $\mu\nu\Sigma$-homomorphism $h : I \to A$. $T \in \mathcal{C}$ is **final in** $\mathcal{C}$ if for all $A \in \mathcal{C}$ there is a unique $\nu\mu\Sigma$-homomorphism $h : A \to T$ (cf. Def. 3.1). $\square$

Each two initial (resp. final) Σ-structures are Σ-isomorphic.

Definition 3.7 Let $SP = (\Sigma, AX)$ be a swinging specification. The **Herbrand SP-model**, **Her(SP)**, is the Herbrand Σ-structure that is defined as follows (cf. Def. 3.5):

- For all predicates $r \in hid\Sigma$, $r^{Her(SP)} = r^{lfp(\Phi)}$ where Φ is the $hidAX$-consequence operator on T_Σ.
- For all $r \in \nu\Sigma \setminus hid\Sigma$, $r^{Her(SP)} = r^{gfp(\Psi)}$ where Ψ is the $(\nu AX \setminus hidAX)$-consequence operator on $Her(SP)|_{hid\Sigma}$.
- For all $r \in \Sigma \setminus \nu\Sigma$, $r^{Her(SP)} = r^{lfp(\Theta)}$ where Θ is the $(AX \setminus \nu AX)$-consequence operator on $Her(SP)|_{\nu\Sigma}$.

The interpretation of $\sim$ in $Her(SP)$ is called **behavioral SP-equivalence** and denoted by $\sim_{SP}$. SP is **behaviorally consistent** if $\sim_{SP}$ is a weak Σ-congruence.

The **initial SP-model**, **Ini(SP)**, is the quotient of $Her(SP)$ by SP-equivalence (Def. 3.4). Provided that SP is behaviorally consistent, the **final SP-model**, **Fin(SP)**, is the quotient of $Her(SP)$ by $\sim_{SP}$. **$\mathbf{Mod}_{bcr}$(SP)** denotes the class of reachable $A \in Mod_{bc}(SP)$ such that for all dynamic predicates $\delta : ws$ and $(a, t^A) \in \delta^A$, $t \sim_{SP} u$ implies $(a, u^A) \in \delta^A$.

SP is **continuous** if the above consequence operators Ψ and Θ are downward resp. upward continuous. A first-order formula satisfied by $Her(SP)$ is called an **inductive theorem of** SP.

Let $SP = (\Sigma, AX)$ be a parameterized specification with parameter $PAR = (\Sigma, AX)$ (cf. Section 2). An **actualization** $SP[\sigma]$ **of** SP **along** σ is **correct w.r.t.** a specification $SP' \supseteq SP[\sigma]$ if $Her(SP')$ satisfies $\sigma(AX)$. The **Herbrand** SP**-model** is the class of Herbrand models of actualizations of SP. An **inductive theorem of** SP is an inductive theorem of all actualizations of SP. SP is **functional**, **behaviorally consistent** or **continuous** if all actualizations of SP are functional, behaviorally consistent or continuous, respectively. ❑

The hierarchical syntax of SP is reflected by the Herbrand SP-model, which therefore resembles the *stable* model of stratified logic program (cf. [2]). Herbrand and initial models always exist. Final models, however, presuppose behavioral consistency.

Theorem 3.8 ([45], Thm. 4.8) *Let $SP = (\Sigma, AX)$ be a swinging specification.*

(1) $Her(SP) \in Mod_{be}(SP) \cap Mod_{\mu\nu}(SP)$ (cf. Def. 3.1).

(2) For all first-order formulas φ and $\sigma : X \to T_\Sigma$, $Her(SP) \models_\sigma \varphi$ iff $Ini(SP) \models_{nat o \sigma} \varphi$.

(3) $Ini(SP) \in Mod_{\equiv}(SP) \cap Mod_{\mu\nu}(SP)$.

(4) $Her(SP)$ is initial in $Mod(SP)$.

(5) $Ini(SP)$ is initial in $Mod_{\equiv}(SP)$.

(6) For all ground hidSP-atoms p, $Her(SP) \models p$ iff $SP \vdash_{cut} p$, in particular, for all equality predicates $\equiv$ of SP, $\equiv^{Her(SP)} = \equiv_{SP}$ (cf. 3.4). ❑

Theorem 3.9 ([45], Thm. 5.1) *Let $SP = (\Sigma, AX)$ be a behaviorally consistent specification (cf. Defs. 3.1, 3.7).*

(1) $Her(SP) \in Mod_{bc}(SP)$ and thus by Thm. 3.3(b), $Fin(SP) \in Mod(SP)$.

(2) For all poly-modal formulas φ and $\sigma : X \to T_\Sigma$, $Her(SP) \models_\sigma \varphi$ iff $Fin(SP) \models_{nat o \sigma} \varphi$.

(3) $Fin(SP) \in Mod_{bcr}(SP) \cap Mod_{\mu\nu}(SP)$.

(4) If SP is visible, then $Fin(SP)$ coincides with $Ini(SP)$.

(5) $Fin(SP)$ is final in $Mod_{bcr}(SP)$. ❑

4 Proof rules and criteria for behavioral consistency

The fact that $Her(SP)$ interprets μ- and ν-predicates as least resp. greatest relations on T_Σ that satisfy AX is crucial for the soundness of the following proof rules. Let AX_r be the set of axioms for a logical predicate r.

fixpoint induction on r

$$\frac{r(x) \Rightarrow \psi}{\exists q : q(x) \Rightarrow \psi \ \wedge \ \bigwedge_{\varphi \in AX_r} \varphi[q/r]} \Updownarrow \quad \text{if } r \in \mu P$$

coinduction on r

$$\frac{\psi \Rightarrow r(x)}{\exists q : \psi \Rightarrow q(x) \ \wedge \ \bigwedge_{\varphi \in AX_r} \varphi[q/r]} \Updownarrow \quad \text{if } r \in \nu P$$

The case $r = \sim$ provides a rule for proving behavioral equivalences:

coinduction on $\sim$

$$\frac{\psi \Rightarrow t \sim u}{\exists q : \psi \Rightarrow q(t,u) \ \wedge \ \bigwedge_{\varphi \in AX_\sim} \varphi[q/\sim]} \Updownarrow$$

Note that $AX_\sim$ is the set of behavior axioms of SP (cf. Def. 2.3(3)).

Fixpoint induction is due to Park (cf. [49]). Both rules can be generalized easily from a single predicate r to several predicates $r_1, \ldots, r_n$ such that it admits proving n conjectures $\psi_1 \Leftarrow r_1(x), \ldots, \psi_n \Leftarrow r_n(x)$ resp. $\psi_1 \Rightarrow r_1(x), \ldots, \psi_n \Rightarrow r_n(x)$ simultaneously.

Fixpoint induction deals with conjectures $\psi \Leftarrow r(x)$ stating that ψ holds true for data related to each other by r. r is often the graph of a defined function (see below). Coinduction deals with inverse conjectures $\psi \Rightarrow r(x)$ stating that r holds true for a set of data specified by ψ. In both cases, the conjecture must be given as an implication $\psi \Leftarrow \varphi$. Fixpoint induction is applicable if the premise φ can be specified as a μ-predicate. Coinduction is applicable if the conclusion ψ can be specified as a ν-predicate. Applying the rule eliminates this predicate from the conjecture so that the rules in some way reduce a proof obligation.

Fixpoint induction and coinduction are equivalence transformations. The downward implication $\Downarrow$ holds true because $Her(SP)$ satisfies AX and thus we may define q as r. The upward implication $\Uparrow$ is valid because all solutions of AX_r in r are supersets resp. subsets of the least resp. greatest solution of AX_r that provides the interpretation of r in the Herbrand model. In other words, if $Her(SP)$ satisfies $\varphi[q/r]$ for all $\varphi \in AX_r$, then $q(x) \Leftarrow r(x)$ resp. $q(x) \Rightarrow r(x)$ are also satisfied. Hence the antecedent of fixpoint induction resp. coinduction follows from the succedent $\psi \Leftarrow q(x)$ resp. $\psi \Rightarrow q(x)$.

q is an existentially quantified predicate variable whose value ranges between ψ and r. Choosing q as a proper subset (in the case of induction) resp. superset (in the case of coinduction) of ψ means to **generalize** resp. **co-generalize** ψ. This complies with the intuition that a smaller relation expresses a stronger condition.

If all predicates of SP are static, then the restriction of coinduction on $\sim$ to unconditional behavioral equivalences essentially agrees with the proof technique of **hidden coinduction** introduced in [18, 26]. In this case the behavior axioms of SP are congruence axioms except for the first one that expresses the coincidence of $\sim$ and $\equiv$ an visible terms. Moreover, the succedent of coinduction on $\sim$ reduces to:

$$\exists q : q(t,u) \ \wedge \bigwedge_{\varphi \in AX_\sim} \varphi[q/\sim].$$

But this is also the proof obligation of hidden coinduction: choose a relation q whose restriction to visible sorts coincides with "equality" (that is the first behavior axiom), and show that q satisfies certain congruence axioms.

While the above rules are correct w.r.t. $Her(SP)$ because the Herbrand model interprets predicates as *least/greatest* relations satisfying their axioms, the following rules are sound because $Her(SP)$ is a fixpoint of the consequence operators constructed from SP. Unfolding an atom $r(t)$ means to apply all axioms for r to $r(t)$ and thus to split the conjecture surrounding $r(t)$ into as many subgoals as there are axioms for r. Since $Her(SP)$ is a fixpoint of the consequence operators Φ, Ψ and Θ (cf. Def. 3.7), complete unfolding rules are *equivalence* transformations:

μ-atom unfolding
$$\frac{r(t)}{\bigvee_{i=1}^{n} \exists Z_i : (t \equiv t_i \wedge \varphi_i)} \Updownarrow \quad \text{if } r \in \mu P,$$
$$\{r(t_1) \Leftarrow \varphi_1, \ldots, r(t_n) \Leftarrow \varphi_n\} = AX_r \text{ and } Z_i = var(r(t_i) \Leftarrow \varphi_i)$$

ν-atom unfolding
$$\frac{r(t)}{\bigwedge_{i=1}^{n} \forall Z_i : (t \equiv t_i \Rightarrow \varphi_i)} \Updownarrow \quad \text{if } r \in \nu P,$$
$$\{r(t_1) \Rightarrow \varphi_1, \ldots, r(t_n) \Rightarrow \varphi_n\} = AX_r \text{ and } Z_i = var(r(t_i) \Rightarrow \varphi_i)$$

Theorem provers usually combine unfolding with term splitting and clash (see below), applied to the equations $t \equiv t_i$. For all $1 \leq i \leq n$ such that term splitting and clash lead to $FALSE$, the summand $t \equiv t_i \wedge \varphi_i$) resp. factor $t \equiv t_i \Rightarrow \varphi_i$) can be omitted when the unfolding rule's succedent is constructed without violating the rule's correctness.

Given a predicate r of SP, the fixpoint properties of $Her(SP)$ often lead to (co-)Horn axioms for the complement $\overline{r}$ of r w.r.t. $Her(SP)$ if one simply negates the premises resp. conclusions of the axioms for r. For instance, if $AX_r = \{r(t_1) \Leftarrow \varphi_1, \ldots, r(t_n) \Leftarrow \varphi_n\}$, then the fixpoint property implies that $Her(SP)$ satisfies

$$r(x) \;\Leftrightarrow\; \bigvee_{i=1}^{n} \exists Z_i : (x \equiv t_i \wedge \varphi_i) \quad \text{and thus} \quad \neg r(x) \;\Leftrightarrow\; \bigwedge_{i=1}^{n} \forall Z_i : (x \not\equiv t_i \vee \neg\varphi_i).$$

If this conjunction is equivalent to a formula $\bigvee_{i=1}^{n} \exists X_i \psi_i$ consisting of atoms and negative literals $\neg q(t)$ such that $q = r$ or SP contains the complement $\overline{q}$ of q, then the Horn clauses

$$\overline{r}(x) \Leftarrow \psi_1[\overline{q}(t)/\neg q(t)], \;\ldots,\; \overline{r}(x) \Leftarrow \psi_n[\overline{q}(t)/\neg q(t)]$$

axiomatize $\overline{r}$ as the complement of r—provided that the extended specification *terminates* ([46], Satz 6.1.9).

If r is a predicate of the visible or hidden level of SP, then $\overline{r}$ can be specified as a ν-predicate, either in terms of r:

$$\overline{r}(x) \;\Rightarrow\; (r(x) \;\Rightarrow\; FALSE),$$

or by grouping the axioms for r as follows:

$$r(t_1), \;\ldots,\; r(t_k), \;\; r(u_1) \Leftarrow (\varphi_1 \wedge q(v_1)), \;\ldots,\; r(u_n) \Leftarrow (\varphi_1 \wedge q(v_n)),$$

and "dualizing" them:

$$\overline{r}(t_1) \Rightarrow FALSE, \;\ldots,\; \overline{r}(t_k) \Rightarrow FALSE, \;\; \overline{r}(u_1) \Rightarrow (\varphi_1 \Rightarrow \overline{q}(v_1)), \;\ldots,\; \overline{r}(u_n) \Rightarrow (\varphi_n \Rightarrow \overline{q}(v_n))$$

(see [46], Satz 8.3.4). The negation of a ν-predicate r specified by arbitrary co-Horn axioms such as:

$$r(t) \;\Rightarrow\; (G \;\Rightarrow\; (\exists X_1(G_1 \wedge r_1(t_1)) \vee \cdots \vee \exists X_n(G_n \wedge r_n(t_n)))),$$

leads to generalized Horn axioms for $\overline{r}$:

$$\overline{r}(t) \;\Leftarrow\; (G \;\wedge\; \forall X_1(G_1 \Rightarrow \overline{r_1}(t_1)) \wedge \;\cdots \wedge \forall X_n(G_n \Rightarrow \overline{r_n}(t_n))).$$

Axioms for complements is all one needs for refuting conjectures in the Herbrand model. The main inference rule used in a refutation proof is the unfolding of complement atoms $\overline{r}(t)$.

Given a functional specification SP, ground goals over $hidSP$ can be proved in a rewriting-oriented way, by applying Horn axioms as logic programs and reducing goals to $TRUE$. Moreover, instead of applying congruence axioms goal reductions rewrite terms analogously to the way they *resolve* logical atoms.

A **fresh variable** of a Horn clause $\varphi = (r(t) \Leftarrow H)$ (resp. $\varphi = (t \equiv u \Leftarrow H)$) is a variable that occurs in u or H, but not in t. **fresh(φ)** denotes the set of fresh variables of φ. Note that Condition 2.3(a) implies $fresh(\varphi) \subseteq H$ if φ is a Horn axiom of a swinging specification.

Definition 4.1 Let SP be a swinging specification and $hidSP = (\Sigma, AX)$. The **reduction calculus for** SP consists of the following rules for reducing goals. Let G be a Σ-goal and $\sigma : X \to T_\Sigma(X)$.

rewriting $\dfrac{G(t\sigma)}{G(u\sigma) \wedge H\sigma} \Uparrow$ if $\varphi = (t \equiv u \Leftarrow H) \in AX$ and $fresh(\varphi)\sigma \subseteq NF_\Sigma(X)$

resolution $\dfrac{r(t\sigma) \wedge G}{H\sigma \wedge G} \Uparrow$ if $r \not\equiv \equiv$, $\varphi = (r(t) \Leftarrow H) \in AX$ and $fresh(\varphi)\sigma \subseteq NF_\Sigma(X)$

reflection $\dfrac{t \equiv t \wedge G}{G} \Uparrow$

A sequence $G_1, \ldots, G_n$ of goals such that for all $1 \leq i < n$, G_{i+1} is obtained from G_i by applying one of the above rules, is called an SP**-reduction of** G_1 **into** G_n and we write $G_1 \vdash_{SP} G_n$. ❑

Definition and Theorem 4.2 (Church-Rosser Theorem) [44] *Let SP be a swinging specification and $hidSP = (\Sigma, AX)$. For all ground Σ-goals G, $G \vdash_{SP} \emptyset$ implies $SP \vdash_{cut} G$. A complete specification SP is functional iff SP is* **confluent**, *i.e. for all ground goals G, $SP \vdash_{cut} G$ implies $G \vdash_{SP} \emptyset$.* ❑

Theorem 4.2 also implies that a functional specification can be transformed into an equivalent relational one by turning each defined function into its graph or input-output relation:

Definition 4.3 (flat formula) Let Σ be a swinging signature and Σ' be Σ without defined functions. A first-order Σ-formula φ is **flat** if all logical atoms of φ are Σ'-atoms and for all equations $t \equiv u$ of φ, either $t, u \in T_{\Sigma'}(X)$ or there are a defined function $f : w \to s$ and $t' \in T_{\Sigma'}(X)_w$ such that $t = f(t')$ and $u \in T_{\Sigma'}(X)$.

Any first-order formula φ can be transformed into an equivalent flat formula $flat(\varphi)$. For a set F of formulas, $flat(F) =_{def} \{flat(\varphi) \mid \varphi \in F\}$. Moreover, $rel(\Sigma)$ is obtained from Σ by replacing each defined function $f : w \to s \in \Sigma$ by the **graph** $r_f : ws$ of f. $rel(F)$ is obtained from F by replacing each equation $f(t) \equiv u$ of F with defined function f by the atom $r_f(t, u)$. ❑

For all modal, poly-modal and weakly modal formulas φ, $flat(\varphi)$ is modal, poly-modal or weakly modal, respectively (cf. 2.2).

Definition 4.4 Let $SP = (\Sigma, AX)$ be a swinging specification. The swinging specifications $flat(SP) = (\Sigma, flat(AX))$ and $rel(SP) = (rel(\Sigma), rel(flat(AX)))$ are called the **flat** and **relational versions of** SP, respectively. ❑

The only function symbols of $rel(\Sigma)$ are the constructors of Σ. A visible predicate of $rel(\Sigma)$ is a visible predicate of Σ or the graph of a visible defined function of Σ. A transition predicate of $rel(\Sigma)$ is a transition predicate of Σ or the graph of a destructor of Σ. A dynamic predicate of $rel(\Sigma)$ is a transition predicate of Σ or the graph of a defined function of Σ.

Theorem 4.5 ([45], 4.13-4.15) *Let SP be a functional and continuous specification. Then $rel(SP)$ is functional and continuous and for all first-order formulas formulas φ,*

$$Her(SP) \models \varphi \iff Her(rel(SP)) \models rel(flat(\varphi)).$$

Given a defined function $f \in \Sigma$, $\sim_{SP}$ is compatible with f iff $\sim_{rel(SP)}$ is zigzag compatible with the graph r_f of f. ❑

One of the most useful consequences of Thm. 4.5 is the soundness of fixpoint induction and unfolding for proving inductive theorems about defined functions: if SP is functional, then by Thm. 4.5(1), the following functional counterparts of fixpoint induction on μ-predicates and μ-atom unfolding, respectively, are correct. Let f be a defined function and AX_f be the set of axioms for f (cf. Def. 4.3).

fixpoint induction on f
$$\frac{f(x) \equiv y \Rightarrow \psi}{\exists q : q(x,y) \Rightarrow \psi \ \wedge \ \bigwedge_{\varphi \in flat(AX_f)} \varphi[q(t,u)/(f(t) \equiv u)]} \Updownarrow$$

term unfolding
$$\frac{\varphi(f(t))}{\bigvee_{i=1}^{n} \exists Z_i : (t \equiv t_i \wedge \varphi(u_i) \wedge \psi_i)} \Updownarrow$$
where $\{f(t_1) \equiv u_1 \Leftarrow \psi_1, \ldots, f(t_n) \equiv u_n \Leftarrow \psi_n\} = AX_f$
and $Z_i = var(f(t_i) \equiv u_i \Leftarrow \psi_i)$

A further consequence of functionality is the soundness of rules for removing constructors:

term splitting $\dfrac{c(t_1,\ldots,t_n) \equiv c(u_1,\ldots,u_n)}{t_1 \equiv u_1 \wedge \cdots \wedge t_n \equiv u_n} \Updownarrow$ where c is a constructor

clash $\dfrac{c(t) \equiv d(u)}{FALSE} \Updownarrow$ where c and d are different constructors

If SP is functional, these equivalences imply that the **standard inequality axioms for** SP specify the complement of $\equiv$ (cf. Def. 3.1):

$c(x_1,\ldots,x_n) \not\equiv c(y_1,\ldots,y_n) \ \Leftarrow \ x_i \not\equiv y_i$ for all constructors $c : s_1 \ldots s_n \to s$ and $1 \leq i \leq n$,
$c(x) \not\equiv d(y)$ for all different constructors c and d.

Repeated applications of term splitting and clash remove an equation $t \equiv t'$ iff t and t' are ground normal forms. For eliminating equations with variables one also needs *term replacement*:

term replacement
$$\frac{t \equiv t' \wedge \varphi(t)}{\varphi(t')} \Downarrow$$
$$\frac{(t \equiv t' \wedge \varphi(t)) \ \Rightarrow \ \psi(t)}{\varphi(t') \ \Rightarrow \ \psi(t')} \Uparrow$$

Behavioral consistency ensures that—due to our Hennessy-Milner Theorem 3.2(2)—term replacement modulo behavioral equality leads to sound proof rules such as:

behavioral term placement
$$\frac{t \sim t' \wedge \varphi(t)}{\varphi(t')} \Downarrow \quad \text{if } \varphi \text{ is poly-modal}$$
$$\frac{(t \sim t' \wedge \varphi(t)) \ \Rightarrow \ \psi(t)}{\varphi(t') \ \Rightarrow \ \psi(t')} \Uparrow \quad \text{if } \varphi \text{ and } \psi \text{ are poly-modal}$$

Existential quantifiers in the conclusions of co-Horn axioms and universal quantifiers in the premises of generalized Horn axioms may violate the continuity of a swinging specification (cf. [45], Section 5). In modal logic, a *finitely branching* or *image finite* transition system ensures continuity. The generalization of image finiteness to arbitrary existential or universal goals in the sense of Def. 2.2 leads to the following definition:

Definition 4.6 (image finiteness) Let $SP = (\Sigma, AX)$ be a swinging specification. Given a Σ-goal G,

$$\mathcal{S}(G) \quad =_{def} \quad \{\tau : var(G) \to NF_\Sigma \mid Her(SP) \models G\tau\}$$

is the set of **normal form solutions of** G. Given $Y \subseteq X$, G is **Y-image finite** if for all $\sigma : X \to NF_\Sigma$, $\mathcal{S}(G\sigma_{X \setminus Y})$ is finite. An existential $\nu\Sigma$-goal $\exists Y \varphi$ is **image finite** if φ is a $hid\Sigma$-goal or splits into goals

G and H such that G is a Y-image finite $hid\Sigma$-goal. A universal Σ-goal $\forall Y(G \Rightarrow H)$ is **image finite** if G and H are $\nu\Sigma$-goals or G is a Y-image finite $\nu\Sigma$-goal.

A disjunction (resp. conjunction) of image finite existential (resp. universal) goals is **image finite**. SP is **image finite** if for all co-Horn axioms $p \Rightarrow (G \Rightarrow \varphi)$ of SP, φ consists of image finite existential goals and for all generalized Horn axioms $p \Leftarrow \varphi$ of SP, φ consists of image finite universal goals. ❑

For instance, the specification INFSEQ of Ex. 2.5 is image finite. However, the conclusion of the following axiom for $fair$ is not image finite:

$$fair(g,s) \;\Rightarrow\; \exists n, s' : (nthtail(n,s) \equiv s' \wedge g(head(s')) \equiv true \wedge fair(tail(s'))).$$

In terms of Def. 4.6, $G = (nthtail(n,s) \equiv s' \wedge g(head(s')) \equiv true)$ and $H = fair(tail(s'))$. The existential goal $\exists n, s' : (G \wedge H)$ is not image finite because there are streams t such that $G[t/s]$ has infinitely many normal form solutions. However, let $G' = (G \wedge forall(not \circ g, firstn(n,s)))$. Then $\exists n, s' : (G' \wedge H)$ is image finite because for all streams t, $G'[t/s]$ has at most one normal form solution.

A visible specification SP is behaviorally consistent. If SP has nonempty hidden, μ- or ν-levels, additional conditions are needed to ensure that SP is behaviorally consistent. We first group the symbols and atoms specified above the visible level of SP (cf. Def. 2.3). A symbol is **non-observing** if it is not an observer.

Given a term t and a term tuple (t,a,u), an atom $\delta(t,a,u)$ is **observing** if either δ is a transition predicate and u is a term or δ is the graph[9] of a destructor and u is a term or δ is a separator and u is empty. $\delta(t,u)$ is **non-observing** if either δ is a non-observing dynamic predicate and u is a term or δ is the graph of a non-observing defined function and u is a term or δ is a non-observing static predicate and u is empty. A goal is **non-observing** if it consists of non-observing atoms.

Given a term tuple t, **hidvar(t)** denotes the set of hidden variables of t.

Definition 4.7 (coinductivity) Let $SP = (\Sigma, AX)$ be a swinging specification. A Σ-normal form t is **strongly normal** if for all $\sigma : X \to NF_\Sigma$ and $u \in NF_\Sigma$, $t\sigma \sim_{SP} u$ implies $t\tau = u$ and $\sigma \sim_{SP} \tau$ for some $\tau : X \to NF_\Sigma$. A co-Horn clause $r(t) \Rightarrow \varphi$ is **coinductive** if t is strongly normal.

A Horn clause $p \Leftarrow \varphi$ is **coinductive** if either $p = \delta(t,u)$ is non-observing and t is strongly normal or $p = \delta(t,a,u)$ is observing,

$$\varphi \;=\; G \wedge \delta_1(t_1,a_1,u_1) \wedge G_1 \wedge \cdots \wedge \delta_n(t_n,a_n,u_n) \wedge G_n$$

and the following conditions hold true: Let $V_0 = var(t,a,G)$ and for all $1 \le i \le n$, $V_i = V_{i-1} \cup var(a_i,u_i,G_i)$.

(1) t is strongly normal or $t = c(t')$ for a constructor c and a strong normal form t', a is strongly normal, G is weakly modal and non-observing, $var(u) \subseteq V_n$ and $out(G) \cap var(t,a) = \emptyset$.

(2) For all $1 \le i \le n$, $\delta_i(t_i,a_i,u_i)$ is observing, (t_i,a_i) is normal, u_i is strongly normal, G_i is a weakly modal non-observing goal, $var(t_i) \subseteq V_{i-1}$, $(var(u_i) \cup out(G_i)) \cap (V_{i-1} \cup var(a_i,u_i)) = \emptyset$ and $hidvar(a_i) \subseteq var(a)$.

SP is **coinductive** if

(3) for all axioms φ of $SP \setminus visSP$, φ is coinductive or an axiom for a non-observing symbol f such that $\sim_{SP}$ is (zigzag) compatible with f,

[9] Cf. Def. 4.3.

(4) for all axioms $p \Leftarrow \varphi$ for observers and all non-observing symbols f occurring in φ, the axioms for f are coinductive and do not contain observers.[10] ❑

Note the different rôles the variables of t, a resp. u play in an observing atom $\delta(t, a, u)$: those of t are *consumed*, those of u are *produced*, $var(a)$ may contain both "input" and "output" variables. Intuitively, conditions 4.7(1) and (2) entail a data flow through a conductive axiom $p \Leftarrow \varphi$ that starts out from t and the "input part" of a, proceeds to the "output part" of a, t_i and the "input part" of a_i, $i > 0$, then propagates from u_i and the "output part" of a_i to t_j and the "input part" of a_j, $j > i$, and finally returns to the "output part" of a and u.

At least the observers must have coinductive axioms if the whole specification shall be coinductive. The conditions on observer axioms are less restrictive than those on axioms for non-observing symbols. This may lead one to declare more symbols as observers. However, more observers increase the number of behavior axioms and thus the number of "cases" generated by unfolding a behavioral equivalence $t \sim t'$ or by applying conduction to a clause of the form $\psi \Rightarrow t \sim t'$ (see above).

Functional visible specifications are coinductive because then all ground normal forms are strongly normal and thus all axioms are coinductive. Other coinductive specifications cover usual formats of transition system specifications [29], SOS (= structural operational semantics) rules [50], codatatypes [30] as well as Δ/Γ-*complete* equations[11] [26] or *observer complete* function definitions [13]. Δ/Γ-completeness and observer completeness are simple subcases of coinductivity. They deal with purely functional specifications whose behavioral equality is determined by destructors only and whose axioms are mostly unconditional equations.

ACCOUNT (Ex. 2.4) and INFSEQ (Ex. 2.5) are coinductive. Even the following specification of stream comprehension is coinductive:

$$head(filter(g, s)) \equiv x \quad \Leftarrow \quad head(s) \equiv x \wedge g(x) \equiv true \tag{1}$$

$$head(filter(g, s)) \equiv head(filter(g, tail(s))) \quad \Leftarrow \quad head(s) \equiv x \wedge g(x) \equiv false. \tag{2}$$

However, only coinductivity *and* functionality imply behavioral consistency (see Thm. 4.8 below), but (1) and (2) are not complete and thus not functional for streams s none of whose elements satisfies g. Correct axioms for $filter$ can only be part of a specification of finite *and* infinite streams such as STREAM (Ex. 4.9).

Coinductive axioms yield a schema for SOS definitions as well as for algebraic nets, SDL systems (see [47], Section 4), labelled transition logic (LTL; [7]) and rewriting logic (*Maude* specifications; [37]). These formalisms mainly differ with respect to the structure of a state. SOS and LTL regard states as (abstract) stores of values, while net states are tuples of multisets, usually called markings. SDL states are unstructured, but implicitly associated with valuations of program variables. States in Maude are sets of objects as in the above example (cf. [37], Sect. 12.4.1).

Coinductive *definition schemas* should not be confused with coinductive *proof rules* such as fixpoint or hidden coinduction (see Section 4). To emphasize the difference some authors call the former *corecursion schemas* (cf., e.g., [10]).

Theorem 4.8 ([45], Thms. 5.15 and 6.4) *A coinductive, functional and image finite specification is continuous and behaviorally consistent.* ❑

[10] This excludes mutually-recursive axiomatizations of observers and non-observers.

[11] Δ and Γ are sets of destructors and constructors, respectively. Δ/Γ-completeness is a special case of the congruence criterion of [52], Thm. 16.

Example 4.9 In the following stream specification, the destructors *head* and *tail* of INFSEQ (cf. Ex. 2.5) are replaced by a transition predicate $\longrightarrow$: $stream \times entry \times stream$. This allows us to include finite sequences into the stream domain.

STREAM = LIST `and` NAT `then`

`hidsorts`	$stream = stream(entry)$
`constructs`	$empty :\rightarrow stream$
	$_\&_ : entry \times stream \rightarrow stream$
	$blink :\rightarrow stream(nat)$
	$nats : nat \rightarrow stream$
	$odds, evens : stream \rightarrow stream$
	$zip : stream \times stream \rightarrow stream$
	$map : (entry \rightarrow entry) \times stream \rightarrow stream$
	$filter : (entry \rightarrow bool) \times stream \rightarrow stream$
`separators`	$disabled : stream$
`transpreds`	$_ \stackrel{_}{\longrightarrow} _ : stream \times entry \times stream$
`static` μ`-preds`	$enabled, finite : stream$
	$exists : (entry \rightarrow bool) \times stream$
ν`-preds`	$fair : (entry \rightarrow bool) \times stream$
	$infinite : stream$
	$forall : (entry \rightarrow bool) \times stream$
`vars`	$n : nat \quad x, y : entry \quad L : list \quad s, s', t, t' : stream$
	$f : entry \rightarrow entry \quad g : entry \rightarrow bool$
`Horn axioms`	$x\&s \stackrel{x}{\longrightarrow} s$
	$blink \stackrel{0}{\longrightarrow} 1\&blink$
	$nats(n) \stackrel{n}{\longrightarrow} nats(n+1)$
	$odds(s) \stackrel{x}{\longrightarrow} odds(t) \Leftarrow s \stackrel{x}{\longrightarrow} s' \wedge s' \stackrel{y}{\longrightarrow} t$
	$evens(s) \stackrel{x}{\longrightarrow} evens(t) \Leftarrow s \stackrel{y}{\longrightarrow} s' \wedge s' \stackrel{x}{\longrightarrow} t$
	$zip(s, s') \stackrel{x}{\longrightarrow} zip(s', t) \Leftarrow s \stackrel{x}{\longrightarrow} t$
	$zip(s, s') \stackrel{x}{\longrightarrow} zip(s, t) \Leftarrow disabled(s) \wedge s' \stackrel{x}{\longrightarrow} t$
	$map(f, s) \stackrel{f(x)}{\longrightarrow} map(f, t) \Leftarrow s \stackrel{x}{\longrightarrow} t$
	$filter(g, s) \stackrel{x}{\longrightarrow} filter(g, t) \Leftarrow s \stackrel{x}{\longrightarrow} t \wedge g(x) \equiv true$
	$filter(g, s) \stackrel{y}{\longrightarrow} t' \Leftarrow s \stackrel{x}{\longrightarrow} t \wedge g(x) \equiv false \wedge filter(g, t) \stackrel{y}{\longrightarrow} t'$
	$enabled(s) \Leftarrow s \stackrel{x}{\longrightarrow} t$
	$disabled(empty)$
	$disabled(odds(s)) \Leftarrow disabled(s)$
	$disabled(evens(s)) \Leftarrow disabled(s)$
	$disabled(evens(s)) \Leftarrow s \stackrel{x}{\longrightarrow} t \wedge disabled(t)$
	$disabled(zip(s, s')) \Leftarrow disabled(s) \wedge disabled(s')$
	$disabled(map(f, s)) \Leftarrow disabled(s)$
	$disabled(filter(g, s)) \Leftarrow disabled(s)$
	$disabled(filter(g, s)) \Leftarrow s \stackrel{x}{\longrightarrow} t \wedge g(x) \equiv false \wedge disabled(filter(g, t))$
	$finite(s) \Leftarrow disabled(s)$
	$finite(s) \Leftarrow s \stackrel{x}{\longrightarrow} t \wedge finite(t)$
	$exists(g, s) \Leftarrow s \stackrel{x}{\longrightarrow} t \wedge g(x) \equiv true$
	$exists(g, s) \Leftarrow s \stackrel{x}{\longrightarrow} t \wedge exists(g, t)$

co-Horn axioms
$$infinite(s) \Rightarrow \exists x,t : (s \xrightarrow{x} t \wedge infinite(t))$$
$$forall(g,s) \Rightarrow (s \xrightarrow{x} t \Rightarrow (g(x) \equiv true \wedge forall(g,t)))$$
$$fair(g,s) \Rightarrow exists(g,s)$$
$$fair(g,s) \Rightarrow (s \xrightarrow{x} t \Rightarrow fair(g,t))$$

In the final STREAM-model, $s \xrightarrow{x} t$ holds true if x is the first entry and t is the rest of s. *disabled* and *enabled* separate empty from nonempty streams. $finite$ and $infinite$ distinguish finite from infinite streams. The other function symbols and predicates are interpreted as the synomymous symbols of INFSEQ (cf. Ex. 2.5). The Horn axioms were inspired by transition system specifications given in [53, 34]. CCS-like processes can be specified coinductively in a quite similar way (see [47]). ❑

5 Conclusion

We have introduced swinging types as a specification formalism that covers functional, relational and state-oriented "transitional" techniques. Swinging types combine the dominant algebraic touch of other data type presentations with concepts, results and methods obtained in relational semantics, modal logic, higher-order functional programming and Horn clause rewriting. The integration of functions and relations becomes particularly evident in the possibility to use defined functions *or* static *or* dynamic predicates as observers that determine the behavior axioms and thus the interpretation of behavioral equality.

Since the number of observers raises the number of behavior axioms and thus the number of cases produced by unfolding behavioral equivalences, only a few functions or predicates should be declared as observers. For most behavioral equalities, one or two observers turn out to be sufficient (cf. [47]). Behavioral consistency and behavioral term replacement require that $\sim$ is a weak congruence. For this purpose we have established coinductivity as a—mainly syntactic—property of a swinging specification that ensures weak congruence and covers most other congruence criteria to be found in the literature on hidden/observational or process types. From a practical point of view, more general weak-congruence criteria than coinductivity are not needed. However, special cases should be distinguished from each other and establish a classification of "coinductive program schemas" as part of a design methodology for swinging types. Different schemas may correspond to different application areas and lead to tailor-made verification and transformation rules.

Coinductivity is accompanied by the other indispensable requirement, namely functionality, which means intuitively that all data presented by the type have unique normal form representations (w.r.t. structural equivalence). By Thm. 4.2, criteria for functionality reduce to criteria for confluence (see, e.g., [44], Sect. 10.5). Functionality is indispensible for verifying defined functions of a swinging type: applications of fixpoint induction on defined functions, term unfolding, term splitting and clash may not be correct if the specification is not functional (see Section 4).

The third main condition besides functionality and coinductivity is image finiteness, which ensures that the consequence operators that build up the Herbrand model are continuous and thus admit inductive arguments on predicates such as, for instance, in the proof of Theorem 4.8.

The syntactic structure of swinging specifications is motivated by the intended applications as well as the goal to obtain simple Herbrand models that both reflect the specifier's intuitive models and can be reasoned about formally with the help of powerful proof rules. These provide the basis for test and

proof procedures that are still to be worked out and implemented. Their development should be guided by case studies along the lines of [47] and integrated into the development of design methods based on swinging types. Case studies are also needed for investigating the range of traditional methods and applications that could be covered by this approach.

[45] also defines and discusses hierarchical relationships between several swinging types. Structured types involving several swinging specifications, in particular specifications *with import* and refinements, are the topic of [48]. They admit, for instance, the specification of defined functions in terms of ν-predicates or to implement visible by hidden sorts and structural by behavioral equalities.

References

[1] *Abstract State Machines (aka Gurevich Machines or Evolving Algebras)*, http://www.eecs.umich.edu/gasm

[2] K.R. Apt, H.A. Blair, A. Walker, *Towards a Theory of Declarative Knowledge*, in: J. Minker, ed., Deductive Databases and Logic Programming, Morgan Kaufmann (1988) 89-148

[3] E. Astesiano, M. Broy, G. Reggio, *Algebraic Specification of Concurrent Systems*, in [5]

[4] E. Astesiano, A. Giovini, G. Reggio, *Observational Structures and their Logic*, Theoretical Computer Science 96 (1992) 249-283

[5] E. Astesiano, H.-J. Kreowski, B. Krieg-Brückner, eds., *Algebraic Foundations of Systems Specification*, IFIP State-of-the-Art Report, Springer 1999

[6] E. Astesiano, G. Reggio, *Algebraic Specification of Concurrency*, Proc. WADT'91, Springer LNCS 655 (1993) 1-39

[7] E. Astesiano and G. Reggio, *Labelled Transition Logic: An Outline*, Technical Report DISI-TR-96-20, University of Genova 1996

[8] E. Astesiano, M. Wirsing, *Bisimulation in Algebraic Specifications*, in: H. Ait-Kaci, M. Nivat, eds., Resolution of Equations in Algebraic Structures 1, Academic Press (1989) 1-31

[9] J.C.M. Baeten, W.P. Weijland, *Process Algebra*, Cambridge University Press 1990

[10] J. Barwise, L. Moss, *Vicious Circles: On the Mathematics of Non-Wellfounded Phenomena*, CSLI Publications, Stanford 1996

[11] J. van Benthem, J. Bergstra, *Logic of Transition Systems*, J. Logic, Language, and Information 3 (1995) 247-283

[12] J. van Benthem, *Exploring Logical Dynamics*, CSLI Publications, Stanford 1996

[13] M. Bidoit, R. Hennicker, *Observer Complete Definitions are Behaviourally Coherent*, Report, University of Munich (1999)

[14] M. Bidoit, R. Hennicker, M. Wirsing, *Behavioural and Abstractor Specifications*, Science of Computer Programming 25 (1995) 149-186

[15] M. Broy, M. Wirsing, *Partial Abstract Types*, Acta Informatica 18 (1982) 47-64

[16] The CoFI Task Group on Language Design, *CASL: The Common Algebraic Specification Language*, 1998, http://www.brics.dk/Projects/CoFI/Documents/CASL/Summary

[17] G. Costa, G. Reggio, *Specification of Abstract Dynamic Data Types: A Temporal Logic Approach*, Theoretical Computer Science 173 (1997) 513-554

[18] R. Diaconescu, K. Futatsugi, *CafeOBJ Report*, AMAST Series in Computing 6, World Scientific 1998

[19] H. Ehrig, B. Mahr, *Fundamentals of Algebraic Specification 1*, Springer 1985

[20] H. Ehrig, F. Orejas, *Dynamic Abstract Data Types: An Informal Proposal*, EATCS Bulletin 53 (June 1994) 162-169

[21] V. Giarratana, F. Gimona, U. Montanari, *Observability Concepts in Abstract Data Type Specifications*, Proc. MFCS '76, Springer LNCS 45 (1976) 576-587

[22] J.A. Goguen, *Stretching First Order Equational Logic: Proofs with Partiality, Subtypes and Retracts*, UCSD, 1997, www-cse.ucsd.edu/users/goguen/ps/ftp97.ps.gz

[23] J.A. Goguen, R. Diaconescu, *Towards an Algebraic Semantics for the Object Paradigm*, Proc. 9th ADT Workshop, Springer LNCS 785 (1994) 1-29

[24] J.A. Goguen, J. Meseguer, *Unifying Functional, Object-Oriented and Relational Programming with Logical Semantics*, in: B. Shriver, P. Wegner, eds., Research Directions in Object-Oriented Programming, MIT Press (1987) 417-477

[25] J.A. Goguen, G. Malcolm, *A Hidden Agenda*, UCSD Technical Report CS97-538, San Diego 1997

[26] J.A. Goguen, G. Malcolm, *Hidden Coinduction: Behavioral Correctness Proofs for Objects*, to appear in MSCS

[27] J.A. Goguen, J.W. Thatcher, E.G. Wagner, *An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types*, in: R. Yeh, ed., Current Trends in Programming Methodology 4, Prentice-Hall (1978) 80-149

[28] A.D. Gordon, *A Tutorial on Co-induction and Functional Programming*, Proc. Functional Programming Glasgow 1994, Springer (1995) 78-95

[29] J.F. Groote, F. Vaandrager, *Structured Operational Semantics and Bisimulation as a Congruence*, Information and Computation 100 (1992) 202-260

[30] T. Hagino, *Codatatypes in ML*, J. Symbolic Computation (1989) 629-650

[31] M. Hanus, *Curry: An Integrated Functional Logic Language*, RWTH Aachen, 1999

[32] M. Hennessy, R. Milner, *Algebraic Laws for Nondeterminism and Concurrency*, J. of the ACM 32 (1985) 137-161

[33] R. Hennicker, M. Bidoit, *Observational Logic*, Proc. AMAST '98, Springer LNCS 1548 (1998) 263-277

[34] B. Jacobs, J. Rutten, *A Tutorial on (Co)Algebras and (Co)Induction*, EATCS Bulletin 62 (June 1997) 222-259

[35] S. Kamin, *Final Data Type Specifications: A New Data Type Specification Method*, ACM TOPLAS 5 (1983) 97-123

[36] K.G. Larsen, *Proof Systems for Hennessy-Milner Logic with Recursion*, Proc. CAAP '88, Springer LNCS 299 (1988) 215-230

[37] J. Meseguer, *A Logical Theory of Concurrent Objects and its Realization in the Maude Language*, in: G. Agha, P. Wegner, A. Yonezawa, eds., Research Directions in Object-Based Concurrency, MIT Press (1993) 313-389

[38] B. Möller, A. Tarlecki, M. Wirsing, *Algebraic Specifications of Reachable Higher-Order Algebras*, Proc. 5th ADT Workshop, Springer LNCS 332 (1988) 154-169

[39] E. Moggi, *Notions of Computation and Monads*, Information and Computation 93 (1991) 55-92

[40] H.-J. Ohlbach, *Semantic-Based Translation Methods for Modal Logics*, J. Logic and Computation 1 (1991) 691-746

[41] P. Padawitz, *Computing in Horn Clause Theories*, Springer 1988

[42] P. Padawitz, *Deduction and Declarative Programming*, Cambridge University Press 1992

[43] P. Padawitz, *Inductive Theorem Proving for Design Specifications*, J. Symbolic Computation 21 (1996) 41-99

[44] P. Padawitz, *Proof in Flat Specifications*, in [5]

[45] P. Padawitz, *Swinging Types = Functions + Relations + Transition Systems*, University of Dortmund 1998, http://ls5.cs.uni-dortmund.de/~peter/Rome.ps.gz

[46] P. Padawitz, *Theorie der Programmierung*, Course Notes, University of Dortmund 1998, http://ls5.cs.uni-dortmund.de/~peter/TdP96.ps.gz

[47] P. Padawitz, *Sample Swinging Types*, Report, University of Dortmund 1999, http://ls5.cs.uni-dortmund.de/~peter/BehExa.ps.gz

[48] P. Padawitz, *Modular Swinging Types*, Report, University of Dortmund 1999, http://ls5.cs.uni-dortmund.de/~peter/MST.ps.gz

[49] D. Park, *Fixpoint Induction and Proofs of Program Properties*, in: B. Meltzer, D. Michie, eds., Machine Intelligence 5, Elsevier (1969) 59-78

[50] G.D. Plotkin, *An Operational Semantics for CSP*, in: D. Bjørner, ed., Proc. IFIP TC-2 Working Conf. Formal Description of Programming Concepts II, North-Holland (1983) 199-225

[51] H. Reichel, *An Approach to Object Semantics based on Terminal Coalgebras*, Math. Structures in Comp. Sci. 5 (1995) 129-152

[52] G. Roşu, J. Goguen, *Hidden Congruent Deduction*, Proc. FTP '98, to appear in Springer LNAI (1999)

[53] J.J.M.M. Rutten, *Universal Coalgebra: A Theory of Systems*, Report CS-R9652, CWI, SMC Amsterdam 1996

[54] C. Stirling, *Modal and Temporal Logics*, in: S. Abramsky et al., eds., Handbook of Logic in Computer Science, Clarendon Press (1992) 477-563

[55] M. Wand, *Final Algebra Semantics and Data Type Extensions*, J. Computer and System Sciences 19 (1979) 27-44

[56] M. Wirsing, *Algebraic Specification*, in: J. van Leeuwen, ed., Handbook of Theoretical Computer Science, Elsevier (1990) 675-788