Fundamental Study
From static to dynamic abstract data-types: 
an institution transformation

Elena Zucca *

DISI – Dipartimento di Informatica e Scienze dell'Informazione, Università di Genova, Via Dodecaneso, 
35, 16146 Genova, Italy

Communicated by M. Wirsing

Abstract

We show how to extend in a canonical way a given formalism for specifying (static) data types (like usual algebraic specification frameworks) with dynamic features. What we obtain in this way is a corresponding formalism for specifying dynamic data-types based on the "state-as-algebra" approach: a dynamic data-type models a dynamically evolving system in which any state can be viewed as a static data type in the underlying formalism, and the dynamic evolution is given by operations handling configurations. Formally, our construction is a functor between two appropriate categories of (specialized) institutions. © 1999—Elsevier Science B.V. All rights reserved

Keywords: Institutions; Dynamic abstract data-types; Dynamic systems; Algebraic approach

Contents

1. Introduction................................................................. 110
2. An introductory example............................................. 113
3. Static and dynamic frameworks................................... 122
   3.1. Static frameworks.................................................. 122
   3.2. State sets............................................................ 126
   3.3. Dynamic frameworks............................................... 127
4. A parameterized dynamic framework............................. 130
   4.1. D-oids............................................................... 131
   4.2. Dynamic terms and their evaluation........................ 133
   4.3. Sentences.......................................................... 137
5. From static into dynamic data-types............................. 140
   5.1. Morphisms of static frameworks............................ 140

---

1 This work has been partially supported by Murst 40% – Modelli della computazione e dei linguaggi di programmazione and CNR – Formalismi per la specifica e la descrizione di sistemi ad oggetti.

* E-mail: zucca@disi.unige.it.

0304-3975/99/$ – see front matter © 1999—Elsevier Science B.V. All rights reserved
PII: S0304-3975(98)00125-X
1. Introduction

This paper deals with formal foundations for the specification of dynamic systems, in particular with the extension to the dynamic case of algebraic techniques.

The classical theory of abstract data-types deals originally with static systems, like data structures. A data structure is modelled by some variant of the notion of algebra over a signature, i.e. a (sorted) set of values with some additional mathematical structure (e.g. functions handling values). An abstract data-type is a class of algebras satisfying some properties defined by logical sentences. Many algebraic formalisms have been proposed, different w.r.t. the choice of algebras (e.g. total, partial, with predicates) and/or sentences (e.g. equations, positive conditional axioms, first-order formulas).

Each one of these formalisms is an institution in the now well-known sense of [14], i.e. a logical system defining models (semantic entities) over signatures giving their syntactic interface and sentences with an associated satisfaction relation stating properties which models either satisfy or not.

Many ways have been considered in the literature for extending the theory of abstract data-types to the dynamic case. For instance, a possibility is to represent the states of a dynamic system as terms (semantically, elements of a special sort) and state transitions as predicates. This approach ("state-as-term") is at the root of a variety of formalisms, like process algebras and algebraic transition systems (see e.g. [3] for a survey).

Although fruitful, this idea is partly unsatisfactory since in some sense static and dynamic aspects are mixed in a representation of this kind: for instance, "pure" functions and operations which modify the state are represented in the same way.

Recently, several extensions of algebraic specification techniques have been proposed, which are based on a different approach, which we call "state-as-algebra" (see e.g. [5, 7, 11, 12, 15, 16]).

As suggested by the name, the basic idea is to model states in the life of a dynamic system as algebras. The signature of these algebras (static signature) represents the static interface of the system, intuitively the observations that one may perform, getting answers which are depending on the current state. Dynamic evolution is modelled consequently by transformation between algebras. In particular, we are interested in "labelled" models, which distinguish between different kinds of possible transformations of the system (like procedures in imperative languages or methods in object-oriented languages and databases). With this choice, the dynamic interface of a system can be formalized in turn by some kind of signature, intuitively the modifications one may perform on the system.
We also use the name "dynamic data-types" for the state-as-algebra approaches, stressing the fact that in this case it is the overall data-type, modelling the system, which becomes dynamic; we call "dynamic-data types" the algebraic models of dynamic systems which indeed remain, essentially, within the usual algebraic framework, modelling states as (possibly special) values (as in the state-as-term approach), stressing the fact that in this case data, i.e. values, may represent dynamic entities.

The state-as-algebra view of a dynamic system has in our opinion many advantages. First of all, it is very natural to model a state as an algebra especially when thinking of "real large systems", like information systems, where a single snapshot is something very complicated and structured. The static and the dynamic features of a system are kept distinct in a clean way, and modelled by semantic entities of different nature. More importantly, this separation allows in principle to combine in a modular way different specification techniques for static and dynamic requirements, in the spirit of the integration of different formalisms which is now emerging as a fundamental topic (see e.g. [10]). Static requirements are conditions whose satisfaction can be stated w.r.t. a given state of the system, like integrity constraints in the case of databases. Since each state is an algebra, they can be expressed in one of the many well-established algebraic formalisms (i.e., an institution $SF$ for the static level), choosing the most adequate for any particular situation. Thus it is important to be able to specify dynamic aspects in a way that can be actually integrated with any of these formalisms. In this paper, we present a formalism $DF$ supporting this capability; formally, for each institution $SF$ chosen for the static level, we get an institution $DF(SF)$ enriching $SF$ by dynamic aspects.$^2$

With respect to the general notion of dynamic system presented above, this formalism models dynamic features by a family of (dynamic) operation symbols, like in a standard algebraic signature. The interpretation of a dynamic operation symbol is, roughly speaking, a function from states into states; moreover, for any pair of source and target states (algebras), say $A$ and $B$, there is a mapping from elements of $A$ into elements of $B$ (tracking map), intuitively keeping trace of the "identity" of entities during the evolution of the system. Indeed, we are interested in modelling systems with some notion of persistent identity of individuals, which can be recovered from a state to another. The definition of the algebras used as states is not fixed once at all, but provided by the underlying institution $SF$ for the static level (e.g. total algebras, partial algebras, first-order structures, ...).

Correspondingly, the sentences in $DF(SF)$, expressing the requirements that a dynamic system should satisfy, are constructed on top of the sentences expressing static requirements, provided by $SF$ (e.g. equations, positive conditional axioms, first-order formulas, ...). We consider three kinds of sentences in $DF(SF)$.

- **Invariants**, which are just static sentences (i.e. the sentences of $SF$). A dynamic system $\mathcal{A}$ satisfies $\phi$ in a state $A$ iff $A$ satisfies $\phi$ in the sense defined in $SF$.

---

$^2$ Actually, both $SF$ and $DF(SF)$ have a richer structure than an institution, as clarified in the sequel.
- **Pre-post conditions**, which are sentences of the form \( \{ \phi_1 \} dt \{ \phi_2 \} \), generalizing classical Hoare's triples, where \( \phi_1 \) and \( \phi_2 \) are static sentences and \( dt \) is a dynamic term, i.e. a term constructed by composing dynamic operations in sequence, hence denoting, roughly speaking, a transformation of states. A dynamic system \( \mathcal{A} \) satisfies \( \{ \phi_1 \} dt \{ \phi_2 \} \) in a state \( A \) iff, whenever \( A \) satisfies \( \phi_1 \) in \( SF \) and the state transformation denoted by \( dt \), applied to \( A \), gives a defined state, this state satisfies \( \phi_2 \) in \( SF \). The tracking map allows to relate valuations of variables before and after executing \( dt \), generalizing what is usually achieved in temporal logic by rigid variables (see e.g. [18]; see also [7] for some other work concerning relating sentences before and after a state transformation).

- **Dynamic equations**, i.e. sentences of the form \( dt_1 = dt_2 \), with \( dt_1, dt_2 \) dynamic terms. A dynamic system \( \mathcal{A} \) satisfies \( dt_1 = dt_2 \) in a state \( A \) iff the state transformations denoted by \( dt_1 \) and \( dt_2 \), applied to \( A \), give the same state (or are both undefined).

  We have defined above the validity of a dynamic sentence w.r.t. a dynamic system \( \mathcal{A} \) and a current state \( A \); the validity w.r.t. \( \mathcal{A} \) can be defined as usual by quantification over all the possible states. Thus, for instance, \( dt_1 = dt_2 \) is valid in \( \mathcal{A} \) iff \( dt_1 \) and \( dt_2 \) denote exactly the same state transformation in \( \mathcal{A} \). Hence, if \( dt_1 = dt_2 \), then \( \{ \phi_1 \} dt_1 \{ \phi_2 \} \) iff \( \{ \phi_1 \} dt_2 \{ \phi_2 \} \) for any pair of static sentences \( \phi_1, \phi_2 \).

  Invariants, pre-post conditions and dynamic equations are presented mainly for illustrating the expressive power of our formalism, showing examples of formulas which can actually be defined on top of a given formalism for static aspects: the definition of an "ideal specification language" for dynamic data-types is outside of the scope of this paper. For instance, a feature very useful for practical application would be some compact way of stating that the execution of a dynamic term should not affect some part of the state, unless specified otherwise (what is sometimes called frame assumption).

  We can now summarize the work done in the paper from a more technical point of view.

- We give a formal definition of **static framework** intended to be an abstraction of concrete formalisms suitable for the description of static aspects of a system. Indeed, the parameter \( SF \) of our construction must have some additional structure w.r.t. a generic institution: we require that models have carriers which are (sorted) sets (that is technically needed for the tracking map) and, correspondingly, that there is an explicit notion of variables in sentences with related valuations of these variables into (carriers of) models (that is technically needed for introducing pre-post sentences). These requirements are very general and met by existing algebraic formalisms, and correspond to a notion of "institution with variables" which we believe is of independent interest (see [20] for a similar notion).

- We give a formal notion of **dynamic framework** over a given static framework, intended to be an abstraction of possible formalisms defined "on top" of a given formalism for defining static aspects, adding dynamics. Formally, a dynamic framework is again a special case of institution: we require that models have carriers which are classes of states (which can be viewed as models in the underlying static framework). The above notion of valuation of variables is lifted in a natural way: a
(dynamic) valuation is given now by a state, together with a (static) valuation into the static model corresponding to this state.

- We present a concrete example $\mathcal{DF}$ of dynamic framework which can be defined over any static framework: i.e., a logical formalism specifying dynamic systems which is parametric over the specification formalism chosen for specifying the static aspects. Formally, we give a functor between two appropriate categories. This is very important since it means, first, that the designer of the system can choose the specification language he/she prefers for describing the static aspects; second, since the transformation is functorial, that a translation between two static formalisms can be naturally extended to a translation between the two corresponding dynamic formalisms.

This paper is the continuation of some preceding work on dynamic data-types. In particular, the models of our parameterized institution are based on $d$-oids presented in [4, 5], and actually we keep the same name. Anyway, some new ideas are needed here for obtaining a formalism which actually supports change of syntax (formally, for defining a reduct). The key problem is that dynamic operations may depend on a "richer" structure than the one specified in the interface (static signature): intuitively, that corresponds to the fact that in concrete cases states are usually partly or completely hidden to the outside.

Other related work is in [6], where it is shown that dynamic terms presented in this paper are a free structure, in [22], where a notion of implementation for dynamic data-types is presented, and in [2], where dynamic data-types are used as semantic framework for a kernel language of modules with state.

The paper is structured as follows. In Section 2 we introduce our ideas by means of a simple example of dynamic system. In Section 3 we give the definitions of static and dynamic framework. In Section 4 we define our parameterized institution. In Section 5 we show that our transformation is actually a functor between the appropriate categories. In the conclusion, we briefly discuss the relevance of our results and give some comparison with related work and hints towards further development.

A preliminary shorter version of this paper is [23].

2. An introductory example

In this section, we informally present the formalism $\mathcal{DF}$ for describing and specifying dynamic systems (parameterized by a formalism $\mathcal{SF}$ for static aspects) which will be formally defined in Section 4. In order to present our ideas, we use throughout the paper a toy example of dynamic system. We consider a graphical system for drawing and moving circles in the cartesian plane, informally described as follows. At a certain instant, there exists a finite number of circles, each one having a position, a size and a colour (red or green). Circles may partly or completely overlap. The system can be modified by either moving or resizing circles, by changing their colour (from green to
red and conversely) and by creating copies of existing circles. Moreover it is possible
to delete simultaneously all the, say, green circles.

We assume that the system starts from an initial state in which only one circle exists,
e.g. with center in the origin, radius one and green colour.

State sets. A natural way of representing a state of the system is as an algebra \( A \)
over the following signature \( \Sigma_{\mathcal{G}} \):

\[
\text{sig } \Sigma_{\mathcal{G}} = \\
\text{sorts } \text{int}, \text{posReal}, \text{colour}, \text{circle} \\
\text{opns } X, Y : \text{circle} \rightarrow \text{int} \\
\text{radius} : \text{circle} \rightarrow \text{posReal} \\
\text{col} : \text{circle} \rightarrow \text{colour}
\]

s.t. \( A_{\text{int}} \), \( A_{\text{posReal}} \) and \( A_{\text{colour}} \) are the sets of integer, positive real and colour values,
respectively, and \( A_{\text{circle}} \) is a (finite) set representing the existing circles. Let \( \mathcal{G} \) be the
class of such algebras: \( \mathcal{G} \) models all the possible states of the dynamic system we want
to describe. We will call \( \mathcal{G} \) a state set over \( \Sigma_{\mathcal{G}} \).

In this example, signature means algebraic many-sorted signature and algebra means
standard many-sorted total algebra; anyway, other choices could be taken, like partial
algebras, algebras with predicates, first-order structures; the concrete definitions of sig-
natures and algebras (or, adopting a more general word, models) over a signature are
part of the parameter \( \text{SF} \).

Dynamic operations and their interpretation. The signature \( \Sigma_{\mathcal{G}} \) above intuitively
represents the structure of each state of the system; in other words, the observations
one can perform on the system at a certain instant. Anyway, that does not describe
completely the system; what is missing is a description of the dynamic aspects of the
system, i.e. how the system can change (pass from one state to another).

In our formalism, dynamic evolution is defined by giving a family of dynamic
operations \( DOP_{\mathcal{G}} \), each one modelling one possible basic way of modifying the system.

\[
\text{DOP}_{\mathcal{G}} = \\
moves: \text{circle} \text{int} \text{int} \Rightarrow \\
movesAll: \text{int} \text{int} \Rightarrow \\
\text{resize}: \text{circle} \text{posReal} \Rightarrow \\
\text{changeCol}: \text{circle} \Rightarrow \\
\text{copy}: \text{circle} \Rightarrow \text{circle} \\
\text{delGreen}: \Rightarrow \text{circle}
\]

Note that dynamic operations have functionality of the form \( s_1 \ldots s_n \Rightarrow [s] \), with \([-] \)
denoting optionality, since they may return a final result or just have a side-effect on
the state (like methods in many object-oriented languages). In the dynamic operation
delGreen the final result is the number of the green circles which have been deleted.

The interpretation of a dynamic operation, say \( dop: s_1 \ldots s_n \Rightarrow [s] \), is, accordingly
with our view of a state as an algebra, a map associating with each pair \( (A, \{a_1, \ldots, a_n\}) \),
where $A \in \mathcal{C}$ is a state (algebra) and $\langle a_1, \ldots, a_n \rangle \in A_{s_1 \ldots s_n}$, a new state modelled by an algebra $B \in \mathcal{C}$ and, if $[s]$ not null, a value $b \in B_s$. In addition, we want to be able to recover how the elements of $A$ are transformed when passing to $B$; formally, we assume a partial map $f$, called tracking map, $f : A \rightarrow B$. The tracking map is significant only for elements which intuitively model entities which may change (like circles in the example); basic values (like integer, positive reals and colours in the example) are viewed as special elements which always exist and never change (i.e. the tracking map is always the identity over them). Intuitively, if $f(a) = b$, $a$ and $b$ represent the same entity; if $f(a)$ is not defined, then the entity represented by $a$ has been deleted passing from $A$ to $B$; conversely, if for some $b \in |B|$, $\exists a \in |A|$ s.t. $f(a) = b$, then $b$ represents a new entity in $B$, not existing before. Finally, if $f(a_1) = f(a_2)$ for some $a_1 \neq a_2$, then the two entities represented by $a_1$ and $a_2$ respectively “collapse” to the same entity passing from $A$ to $B$.

As should be clear from the case analysis above, the tracking map allows us to have a very general and abstract notion of persistent identity, different from the standard way identity is modeled in object systems, i.e. by unique names. This standard way would correspond indeed to assume that each entity $a$ has a name $Name(a)$ in some fixed set of unique names $N$, and we know “a priori” that $a$ and $b$ represent the same entity iff $Name(a) = Name(b)$. In our model instead, instantiations of the same entity in different states are recognized “a posteriori” by the tracking map (i.e., as explained below, if $f(a) = b$, then $a$ and $b$ represent the same entity). It should be clear that the former solution is in some sense “an implementation” of the latter; indeed, the choice of the set of unique names is arbitrary, and it is also not relevant the fact that each entity has exactly one name; the same effect would be achieved assuming e.g. to have an equivalence relation $\sim$ on $N$ and stating that $a$ and $b$ represent the same entity iff $Name(a) \sim Name(b)$. Actually, in [5], it has been proved that, under an assumption formalizing the intuition that in the system there is no “collapsing” of identities, it is always possible to construct an isomorphic system with a canonical choice of unique names (roughly speaking, the unique name of an entity $a$ is the equivalence class of all the elements $b$ in some state s.t. $b$ is transitively reachable from $a$ via the tracking map). On the contrary, the case in which the system allows two entities which were different to become the same is expressible in our model and not expressible by using unique names.

Consider, for example, the dynamic operation move. The expected interpretation associates, with each pair $\langle A, \langle \gamma, z_1, z_2 \rangle \rangle$, where $A \in \mathcal{C}$ represents a state of the system, $\gamma \in A_{circle}$ an existing circle and $z_1, z_2$ two integer values, a new state $B \in \mathcal{C}$ defined as follows:

- $|B| = |A|$, 
- $X^B(\gamma) = X^A(\gamma) + z_1$, $Y^B(\gamma) = Y^A(\gamma) + z_2$, 
- $X^B(\gamma') = X^A(\gamma')$, $Y^B(\gamma') = Y^A(\gamma')$, $\forall \gamma' \in B_{circle}, \gamma' \neq \gamma$, 
- $radius^B = radius^A$, $col^B = col^A$. 

In this case the tracking map from $|A|$ to $|B|$ is just the identity: no new elements are created. Note anyway that, as explained above, an equivalent definition could be to take $B_{\text{circle}}$ any set in bijection with $A_{\text{circle}}$, with the tracking map being this bijection, say $f$, and operations in $B$ defined by homomorphism:

$$X^B(f(\gamma)) = X^A(\gamma) + z_1, \ Y^B(f(\gamma)) = Y^A(\gamma) + z_2,$$

$$X^B(f(\gamma')) = X^A(\gamma'), \ Y^B(f(\gamma')) = Y^A(\gamma'), \ \forall \gamma' \in A_{\text{circle}}, \gamma' \neq \gamma,$$

$$\text{radius}^B(f(\gamma')) = \text{radius}^A(\gamma'), \ \text{and col}^B(f(\gamma')) = \text{col}^A(\gamma'), \ \forall \gamma' \in A_{\text{circle}}.$$

This equivalence can be formally expressed as an isomorphism in the category of d-oids over a signature (see Def/Prop. 13 in the sequel).

We consider now a dynamic operation creating new objects, like e.g. $\text{copy}$. The expected interpretation associates, with each pair $\langle A, \gamma \rangle$, where $A \in \mathcal{G}$ represents a state and $\gamma \in A_{\text{circle}}$ an existing circle, a pair $\langle B, \gamma' \rangle$ defined as follows:

$$B_{\text{circle}} = A_{\text{circle}} \cup \{ \gamma' \}, \ \text{with} \ \gamma' \notin A_{\text{circle}},$$

$$X^B(\gamma') = X^A(\gamma), \ Y^B(\gamma') = Y^A(\gamma), \ \text{radius}^B(\gamma') = \text{radius}^A(\gamma), \ \text{col}^B(\gamma') = \text{col}^A(\gamma),$$

$$X^B(\gamma'') = X^A(\gamma''), \ Y^B(\gamma'') = Y^A(\gamma''), \ \text{radius}^B(\gamma'') = \text{radius}^A(\gamma''), \ \text{col}^B(\gamma'') = \text{col}^A(\gamma''), \ \forall \gamma'' \in A_{\text{circle}}.$$

(Moreover, because $A, B \in \mathcal{G}$, $B_{\text{colour}} = A_{\text{colour}} = \{ \text{red, green} \}, B_{\text{int}} = A_{\text{int}} = \mathbb{Z}, B_{\text{posReal}} = A_{\text{posReal}} = \mathbb{R}$.) In this case the tracking map from $A_{\text{circle}}$ to $B_{\text{circle}}$ is the inclusion. Analogously to the case above, an equivalent definition can be obtained replacing the inclusion by any injective function from $A_{\text{circle}}$ into a set having one more element, and defining operations over images of old elements by homomorphism.

Note that a dynamic operation is expected to give in general the semantic counterpart of a complex behaviour. For instance, one dynamic operation typically corresponds to the execution of a method, even possibly involving calls of other methods inside; indeed what is modeled is the resulting semantics of the method.

**Constant dynamic operations and their interpretation.** In our approach it is convenient, for a complete analogy with the static case, to have constant dynamic operations, with functionality written $[s]$. Constant operations just define a particular state of the system (possibly together with a value), to be intended as one possible starting state of the evolution.

In our example, we add to $DOP_\mathcal{G}$ one constant dynamic operation

$$\textit{start: circle}$$

whose interpretation is a pair $\langle A, \gamma \rangle$ where

$$A_{\text{circle}} = \{ \gamma \},$$

$$X^A(\gamma) = 0, \ Y^A(\gamma) = 0, \ \text{radius}^A(\gamma) = 1, \ \text{col}^A(\gamma) = \text{green}.$$
Note that constant dynamic operations are different from parameterless non-constant operations (with functionality ⇒ [s], like `delGreen` in our example) which always have the state as implicit argument. This difference has no correspondence in usual algebraic frameworks, and is propagated to the definition of dynamic terms and sentences (see later on).

**D-oid signatures and D-oids.** The overall structure obtained enriching $\mathcal{E}$ by dynamic operations is called a d-oid over the d-oid signature $D\Sigma_{\mathcal{E}} = \langle \Sigma_{\mathcal{E}}, DOP_{\mathcal{E}} \rangle$ (Def/Prop. 11 and 12).

Note that, comparing the formal description of a dynamic system given by a d-oid with a formal description given in the traditional style of transition systems, the states of the transition system are given by the class $\mathcal{E}$ of $\Sigma_{\mathcal{E}}$-algebras, and there is a transition from $A$ into $B$ whenever, for some dynamic operation $dop$ and tuple of values $\vec{a}$ in $A$, $dop^{\mathcal{E}}(\langle A, \vec{a} \rangle) = \langle B[\vec{b}] \rangle$. Hence, in a d-oid there is implicitly the usual notion of "reachability" of states (from one or more initial states). Anyway, in a d-oid there is some more information which can be hardly expressed by the simple notion of transition system: first, that each dynamic operation is a function (hence the non-determinism only originates from the possibility of applying different dynamic operations, or the same with different arguments, to the same state), second the tracking map which "connects" elements of different states.

As usual, given a d-oid signature $D\Sigma = \langle \Sigma, DOP \rangle$, the class of the d-oids over $D\Sigma$ describes all the possible "implementations" of the interface defined by $\langle \Sigma, DOP \rangle$, which differ for their possible states and for the interpretation of the dynamic operations.

In the example considered until now, the states in $\mathcal{E}$ are exactly algebras over $\Sigma_{\mathcal{E}}$. In general this is a too strong requirement; in the definition of a d-oid over $D\Sigma_{\mathcal{E}}$, we only ask that the states are structures which "can be viewed" as $\Sigma_{\mathcal{E}}$-algebras (i.e. that, for any state, there is an associated $\Sigma_{\mathcal{E}}$-algebra). This models the fact that, in dynamic systems, states are partly or completely "hidden" to the outside, and is technically needed for defining the reduct of a d-oid. To see this, let us consider the following d-oid signature $\overline{D\Sigma} = \langle \overline{\Sigma}, \overline{DOP} \rangle$:

```plaintext
sig \overline{\Sigma} =
sorts int, posReal, circle
opns
   X, Y : circle → int
   radius : circle → posReal
\overline{DOP} =
   move : circle int int ⇒
   resize : circle posReal ⇒
   delGreen : int int ⇒
```

This d-oid signature is obtained from $D\Sigma_{\mathcal{E}}$ just forgetting some sorts (``colour''), static operations (``col'') and dynamic operations (``moveAll, changeCol, copy''); formally, there is a morphism of d-oid signatures from $\overline{D\Sigma}$ to $D\Sigma_{\mathcal{E}}$, see Definition 11, whose components are set inclusions.
In institutions [14], whenever there is a morphism between two signatures, say \( \sigma : \Sigma_1 \to \Sigma_2 \), there exists a functor \( -_\sigma \) from the category of \( \Sigma_2 \)-models to the category of \( \Sigma_1 \)-models, called the reduct functor; intuitively, this functor expresses the fact that any \( \Sigma_2 \)-model “can be viewed” as a \( \Sigma_1 \)-model. In the example above, that means that we expect to be able to be able to see \( \mathcal{C} \) as a \( \mathcal{C}_1 \)-model. In this reduct, it is not possible to take as states algebras over the restricted static signature \( \Sigma \). Indeed, the interpretation of \( \text{delGreen} \) in \( \mathcal{C} \) strictly depends on the \( \text{col} \) operation (since the value \( n \) returned by \( \text{delGreen} \) is the number of existing green circles), hence, if we forget \( \text{col} \), there is no canonical way of deriving an interpretation of \( \text{delGreen} \) over \( \Sigma \)-algebras. This is a simple example of a very general situation, i.e. the fact that the definition of dynamic operations in a system (e.g. procedures in a software module) may depend on some components of the state which are hidden, i.e. not declared in the interface.

The solution we take is to “relax” the definition of state set over \( \Sigma \); we do not require that the states are \( \Sigma \)-algebras, but only that there is a mapping from the class of the states into \( \Sigma \)-algebras, intuitively giving, for any state, its “view” as \( \Sigma \)-algebra (i.e. its observable view). With this definition, we can take as reduct of \( \mathcal{C} \) w.r.t. \( \overline{D\Sigma} \) the d-oid which keeps the same states of \( \mathcal{C} \), but where the view of a state, say \( A \) (which was \( A \) itself in \( \mathcal{C} \)) is the reduct \( A_{\overline{\Sigma}} \). See Def/Prop. 6 for the formalization of this idea.

**Dynamic terms.** In order to define the logical part of our formalism (i.e. sentences expressing properties of the dynamic system), we introduce first of all dynamic terms, which are the analogous of terms in usual algebraic frameworks. We distinguish constant and non-constant dynamic terms.

A non-constant dynamic term over a d-oid signature \( D\Sigma \) and an \( S \)-sorted family of variables \( X \), with \( S \) the sorts of (the static part of) \( D\Sigma \), is a sequence of the form

\[
[x_1 \leftarrow]dop_1(\bar{x}_1); \ldots; [x_n \leftarrow]dop_n(\bar{x}_n),
\]

where
- \( dop_1, \ldots, dop_n \) are non-constant dynamic operations in \( D\Sigma \);
- for each \( i = 1, \ldots, n \), \( \bar{x}_i \) is a tuple of variables whose number and types correspond to the arity of \( dop_i \); each one of these variables is either in \( X \) (i.e. is a free variable) or is \( x_j \), for some \( j < i \);
- square brackets denote optionality; for each \( i = 1, \ldots, n \), \( x_i \leftarrow \) is present only if \( dop_i \) has a result sort.

An example is

\[
y \leftarrow \text{copy}(x); \text{changeCol}(y); \text{changeCol}(x)
\]

A non-constant dynamic term denotes a state transformation. For instance the dynamic term above intuitively denotes the state transformation which consists in, first, creating a new circle \( y \) as a copy of an existing circle \( x \), then changing the colour of \( y \), finally changing the colour of \( x \). Note that the variable \( x \) is free (hence in order
to evaluate this dynamic term we need an initial state, say $A$, and a valuation of $x$ in $A_{\text{circle}}$, while the first occurrence of the variable $y$ is used to bind further occurrences.

Constant dynamic terms are defined in the same way, except that the first element of the sequence is a constant operation. For instance,

$$x \leftarrow \text{start}; y \leftarrow \text{copy}(x); \text{changeCol}(y); \text{changeCol}(x)$$

denotes the state where two circles exist, both with center in the origin, radius one and red colour, obtained as follows: first, taking as initial state $A$ and as valuation of $x$ in $A_{\text{circle}}$ the circle $y$ where $\langle A, y \rangle$ is the result of $\text{copy}$; then, applying the state transformation described above. Note that constant dynamic terms have no free variables.

As shown by the examples, constant dynamic terms denote particular states of the system (they can be seen as derived constant dynamic operations) while non-constant dynamic terms denote state transformations (roughly speaking, functions from states into states; they can be seen as derived non-constant dynamic operations). This is perfectly analogous to what happens for usual terms. Anyway, note that in the dynamic case, due to the fact that there is the implicit state parameter which plays a special role, constant dynamic terms are different from non-constant dynamic terms with no free variables, like for instance

$$y \leftarrow \text{delGreen}; \text{moveAll}(y, y)$$

which has no free variables, but still needs to be evaluated w.r.t. to an initial state. This dynamic term denotes a state transformation which consists in deleting all the existing green circles, keeping in $y$ their number, and then in moving all the remaining (hence red) circles by $y$ both horizontally and vertically.

Dynamic sentences. We consider now which should be intuitively a specification of a dynamic system with interface $\langle \Sigma_\chi, \text{DOP}_\chi \rangle$. In other words, we want to specify in an abstract way, by means of logical sentences, which we will call dynamic sentences, some requirements over the behaviour of the system.

At the model level, the definition of $d$-oid is parameterized by the framework $\text{SF}$ chosen for the static level, which defines what are (static) signatures and algebras modeling states. Analogously, at the specification level, we assume that some language of sentences is provided for specifying requirements over single states of the system (static sentences), and construct the dynamic sentences "on top" of those. In our running example, we take as static sentences first order sentences constructed taking as basis equalities between $\Sigma_\chi$-terms.

Analogously to dynamic terms, we distinguish dynamic sentences into constant and non-constant. The intuition is that constant dynamic sentences express properties which may hold or not in a dynamic system (formally, their validity is defined w.r.t. a $d$-oid $\mathcal{A}$), while non-constant dynamic sentences express properties depending on the current state of the system (formally, their validity is defined w.r.t. a $d$-oid $\mathcal{A}$ and a state $A$). Of course, the validity of a non-constant dynamic sentence w.r.t. a $d$-oid can be defined in a derived way by considering the sentence as implicitly quantified over all the possible states.
As first example of non-constant dynamic sentences, one can just consider static sentences. For instance, let us call $I$ the static sentence below

$$X(c_1) = X(c_2) \land Y(c_1) = Y(c_2) \supset \text{col}(c_1) = \text{col}(c_2)$$

requiring that two circles with the same center have the same colour; $I$ can be seen as a dynamic sentence which holds in a d-oid $\mathcal{A}$ and a current state $A$ iff it holds in (the observable view of) $A$ in the sense defined in SF.

If they are considered as implicitly universally quantified over the possible states, static sentences express invariants i.e. requirements which must hold in all the states of the system, like integrity constraints in the case of databases (another similar notion is that of class invariants in Eiffel [19]). For instance $I$ above can be seen as a dynamic sentence which holds in a d-oid $\mathcal{A}$ iff it holds in (the observable view of) any state of $\mathcal{A}$; the d-oid $\mathcal{G}$ defined above, whose states are all the $\Sigma_\mathcal{G}$-algebras, obviously does not satisfy this property. An example of d-oid satisfying this property is one whose states are all the $\Sigma_\mathcal{G}$-algebras where $I$ holds and where each method is defined like in $\mathcal{G}$, except that whenever the resulting algebra would not satisfy $I$, in which case a different result is given (for instance the state is not changed).

Invariants express “general” properties of the dynamic system, which do not refer to single dynamic operations. Anyway, in order to describe the behaviour of a dynamic system, one also needs to specify the expected behaviour of dynamic operations. A standard way of doing that is by pre-post sentences (generalizing Hoare triples). For instance, we show below two pre-post sentences expressing an expected property of move and changeCol, respectively.

$$\{X(c) = x \land Y(c) = y\} \text{move}(c,x',y') \{X(c) = x + x' \land Y(c) = y - y'\},$$

$$\{\text{col}(c) = k\} \text{changeCol}(c); \text{changeCol}(c) \{\text{col}(c) = k\}.$$

The first sentence specify how the coordinates of the center of the argument are modified by move; the second states that executing two times changeCol on a circle has as effect that its colour remains the same. Formally, these sentences are triples $\{\phi_1\} \text{dt} \{\phi_2\}$ where $\phi_1, \phi_2$ are static sentences and $\text{dt}$ is a dynamic term.

Note that using pre-post conditions turns out to be very verbose if one wants to give a “complete” specification of a dynamic operation. For instance, the first sentence above describes the effect of move on the position of the center of the argument $c$, but says nothing on the effect on the radius and colour of $c$ and, more in general, on possible side-effects on other existing circles. In the case of move what we intuitively expect is that changing the position of the center of $c$ is the unique effect of the call move$(c,x',y')$, while “the rest” remains unchanged. Anyway, in order to formally specify this fact we would need a quite long sentence explicitly listing all the properties which remain unchanged. This problem is known as the frame problem and usually solved by introducing some convention. Here we do not face this problem since our aim is not to design a good specification language, but to show that it is possible to enrich a formalism suitable for static aspects by dynamic features in a canonical way. Anyway, as a partial solution, we introduce another kind of dynamic sentences, which
we call *dynamic equations* and allow to state some properties in a more compact way. An example is the sentence below,

\[
move(c,x,y); move(c,x',y') = move(c,x + x',y + y'),
\]

stating that the effect of a sequence of two move operations on the same circle is the same of a unique move operation having as coordinate arguments the (componentwise) sum of the two coordinate arguments. In general, a dynamic equation is of the form \(dt_1 = dt_2\) where \(dt_1, dt_2\) are dynamic terms, and the intended meaning is that \(dt_1\) and \(dt_2\) have exactly the same effect. Thus if \(dt_1 = dt_2\) holds then, for each pair \(\phi_1, \phi_2\) of static sentences the validity of \(\{\phi_1\} dt_1 \{\phi_2\}\) must coincide with the validity of \(\{\phi_1\} dt_2 \{\phi_2\}\).

The sentences shown until now do not take into account the fact that a dynamic operation may have, together with an effect on the state, a result value, hence we want to be able to specify properties which involve this result. An example of sentence which illustrates this feature is the following, (partly) specifying the expected behaviour of *copy*:

\[
\{true\} c' \leftarrow copy(c) \{X(c') = X(c) \land Y(c') = Y(c) \land radius(c') = radius(c)\}.
\]

stating that the circle returned by a *copy* operation has the same center and radius of the argument circle.

Finally we illustrate constant sentences. The analogous of pre-post conditions in the constant case are sentences of the form \(\{dt\}\phi\) stating that \(\phi\) must hold in the state obtained as result of \(dt\). An example is

\[
\{c \leftarrow start\} X(c) = 0 \land Y(c) = 0 \land radius(c) = 1.
\]

(partially) specifying the expected behaviour of the constant dynamic operation *copy*.

Dynamic equations in the constant case have the form \(dt_1 = dt_2\) where \(dt_1, dt_2\) are constant dynamic terms and the intended meaning is that the two terms denote the same state (and possibly result). Invariants make no sense in the constant case.

In the following sections, we formally present the logic formalism \(DF\) for specifying dynamic data-types based on the ideas informally introduced so far.

As explained above, this formalism is parameterized by the underlying formalism \(SF\). Technically, this fact is expressed by defining \(DF\) as a functor from the category \(StFram\) of the static frameworks into the category \(DynFram\) of the dynamic frameworks.

The technical presentation is organized as follows:
- in Section 3 we define static and dynamic frameworks, i.e. the objects in \(StFram\) and \(DynFram\);
- in Section 4 we define the object part of the functor \(DF\), i.e. we show how to define, on top of a given static framework \(SF\), an associated dynamic framework \(DF(SF)\);
- in Section 5 we complete the construction on morphisms, i.e. we define appropriate morphisms between static (resp. dynamic) frameworks, thus defining the two cate-
categories $\text{StFram}$ and $\text{DynFram}$, and define the morphism part of $DF$, showing that actually the construction is functorial.

3. Static and dynamic frameworks

In this section, we give a formal definition of static and dynamic framework. A static framework is a logical formalism which has all the components required for being an institution (signatures, models, sentences and satisfaction relation – see Definition A.1 in the Appendix A) with some additional features which are, from one side, technically needed to be able to enrich this formalism by dynamic operations, from the other side typical of concrete algebraic formalisms. These additional features are summarized below.

- A signature $\Sigma$ has an underlying set of sorts $S$, and a model over $\Sigma$ has an underlying carrier which is an $S$-sorted set. (Actually, the essential point is to have a carrier which is a set: the choice of many-sorted frameworks is mainly for following the tradition in algebraic specifications.) That corresponds to consider concrete institutions in the sense of [8].
- Instead of considering sentences over a signature, we consider sentences over a signature with variables, i.e. a pair $\langle \Sigma, X \rangle$ where $\Sigma$ is a signature over $S$ and $X$ is an $S$-sorted set; correspondingly, a sentence is evaluated not just w.r.t. a $\Sigma$-model, but w.r.t. a valuation, which is a pair $\langle A, r: X \to |A| \rangle$ where $A$ is a $\Sigma$-model and $r$ is a map associating values with variables. Sentences as usually defined ("constant" sentences in the sense that their evaluation does not depend on a valuation but only on a $\Sigma$-model) are recovered as sentences without free variables (i.e. when $X = \emptyset$).

A similar notion is introduced in [20] under the name of context institution; in the conclusion we outline the differences.

3.1. Static frameworks

Notations. We denote by $id_A$ the identity of $A$, for any $A$ object in a category. If $C$ is a category, then $|C|$ is the class of its objects. The functor $SSet$ gives, for any set $S$, the category of sorted set over $S$ ($S$-sorted sets), with partial maps as morphisms (since we want to include algebraic frameworks with partial homomorphisms). If $\sigma: S \to S'$ is a map, $X$ is an $S'$-sorted set, then $X|_\sigma$ denotes the $S$-sorted set whose $s$-component is the $\sigma(s)$-component of $X$, for all $s \in S$; conversely, for any $S$-sorted set $X$, $\sigma(X)$ is the $S'$-sorted set whose $s'$-component is the disjoint union of all the $s$-components of $X$ s.t. $\sigma(s) = s'$ (see Def/Prop. A.2 and A.3 in the Appendix).

We first define the model part of a static framework, then we add sentences.

Definition 1. A static model part is a 4-tuple $\langle \text{Sig, Sorts, Mod, } \models \rangle$ where
- $\text{Sig}$ is a category whose objects are called signatures;
- $\text{Sorts}$ is a functor, $\text{Sorts}: \text{Sig} \to \text{Set}$, called a sort functor for $\text{Sig}$; for any signature $\Sigma$, the elements of $S = \text{Sorts}(\Sigma)$ are called the sorts of $\Sigma$; we also say that $\Sigma$ is
over \( S \); for any signature morphism \( \sigma \), \( \text{Sorts}(\sigma) \) is denoted by \( \sigma \) when there is no ambiguity;

- \( \text{Mod} \) is a functor, \( \text{Mod} : \text{Sig}^{\text{op}} \rightarrow \text{Cat} \); for any signature \( \Sigma \), objects in \( \text{Mod}(\Sigma) \) are called \textit{models} over \( \Sigma \) or \( \Sigma \)-\textit{models}, and morphisms are called \( \Sigma \)-\textit{morphisms}; for any signature morphism \( \sigma : \Sigma_1 \rightarrow \Sigma_2 \), the functor

\[
\text{Mod}(\sigma) : \text{Mod}(\Sigma_2) \rightarrow \text{Mod}(\Sigma_1)
\]

is called the \textit{reduct} functor and denoted by \(-|\sigma|\);

- \( |\cdot| \) is a natural transformation, \( |\cdot| : \text{Mod} \rightarrow \text{SSet} \circ (\text{Sorts})^{\text{op}} \), s.t., for any signature \( \Sigma \), the functor \( |\cdot|_\Sigma \) is faithful; for any \( \Sigma \)-model \( A \), \( |A|_\Sigma \) is called the \textit{carrier} of \( A \), and denoted by \( |A| \) or even \( A \) when there is no ambiguity, and analogously for morphisms.

Accordingly to the terminology of \cite{1}, for any signature \( \Sigma \) over \( S \), \( \langle \text{Mod}(\Sigma), |\cdot|_\Sigma \rangle \) is a concrete category over \( \text{SSet}(S) \). The assumption that \( |\cdot|_\Sigma \) is faithful (i.e. for all parallel morphisms \( f, g : A \rightarrow B \), if \( |f| = |g| \) then \( f = g \)) models the fact that morphisms of static models are maps which respect some condition. Typical examples of static model parts are total or partial algebras over a many-sorted algebraic signature. Indeed, algebras are pairs consisting of a sorted set together with the interpretation of the operation symbols, hence the carrier is simply obtained taking the first component; moreover, an homomorphism between algebras is a map between their carriers compatible with operations.

Since a signature has an underlying set of sorts, it is possible to extend in a canonical way signatures to \textit{signatures with variables}, as shown below.

\textbf{Def/Prop. 2.} If \( \text{Sig} \) is a category of signatures with a sort functor, then \( \text{Sig}^{\text{Var}} \) denotes the category where an object is a pair \( \langle \Sigma, X \rangle \) with \( \Sigma \) signature over \( S \) and \( X \) sorted set over \( S \) whose elements are called \textit{variables}, and a morphism from \( \langle \Sigma_1, X_1 \rangle \) into \( \langle \Sigma_2, X_2 \rangle \) is a pair \( \langle \sigma, h \rangle \) with \( \sigma : \Sigma_1 \rightarrow \Sigma_2 \) and \( h : X_1 \rightarrow (X_2)_\sigma \). We call an object in \( \text{Sig}^{\text{Var}} \) a signature with variables. For any \( \langle \Sigma, X \rangle \in \text{Sig}^{\text{Var}} \), \( \text{id}_{\langle \Sigma, X \rangle} = \langle \text{id}_\Sigma, \text{id}_X \rangle \); for any \( \langle \sigma_1, h_1 \rangle, \langle \sigma_2, h_2 \rangle \) pair of morphisms in \( \text{Sig}^{\text{Var}} \), \( \langle \sigma_2, h_2 \rangle \circ \langle \sigma_1, h_1 \rangle = \langle \sigma_2 \circ \sigma_1, h_2 \circ h_1 \rangle \).

Since models have a carrier which is a sorted set, it is possible to extend in a canonical way models to \textit{valuations}, as shown below.

\textbf{Def/Prop. 3.} Let \( \langle \text{Sig}, \text{Sorts}, \text{Mod}, |\cdot| \rangle \) be a static model part. Then \( \text{Val} : (\text{Sig}^{\text{Var}})^{\text{op}} \rightarrow \text{Cat} \) is the functor defined by

\[
\forall \langle \Sigma, X \rangle \in \text{Sig}^{\text{Var}}, \text{Val}(\langle \Sigma, X \rangle) \text{ is the category where an object is a pair } \langle A, r \rangle, \text{ with } A \text{ model over } \Sigma \text{ and } r : X \rightarrow |A|, \text{ called a valuation of } X \text{ (into } A \text{), and } f : \langle A, r_A \rangle \rightarrow \langle B, r_B \rangle \text{ is a morphism iff } f : A \rightarrow B \text{ is a } \Sigma \text{-morphism and } r_B = |f| \circ r_A; \]

\footnote{In other words, \( \text{Val}(\langle \Sigma, X \rangle) \) is the comma category \( X \downarrow |\cdot|_\Sigma \).}
Definition 4. A static framework is a tuple $\mathbf{SF} = \langle \text{Sig}, \text{Sorts}, \text{Mod}, \text{-}, \text{Sen}, \models \rangle$ where
- $\langle \text{Sig}, \text{Sorts}, \text{Mod}, \text{-} \rangle$ is a static model part;
- $\text{Sen}$ is a functor, $\text{Sen}: \text{Sig}^{\text{Var}} \rightarrow \text{Set}$; for any signature with variables $\langle \Sigma, \mathcal{X} \rangle$, the elements of $\text{Sen}(\langle \Sigma, \mathcal{X} \rangle)$ are called sentences over $\Sigma$ and $\mathcal{X}$; for any morphism of signatures with variables $(\alpha, \beta)$, $\text{Sen}(\langle \alpha, \beta \rangle)$ is denoted by $\langle \alpha, \beta \rangle$ when there is no ambiguity;
- $\forall \langle \Sigma, \mathcal{X} \rangle \in |\text{Sig}^{\text{Var}}|$, $\models_{\langle \Sigma, \mathcal{X} \rangle}$ is a relation over $|\text{Val}(\langle \Sigma, \mathcal{X} \rangle)| \times \text{Sen}(\langle \Sigma, \mathcal{X} \rangle)$ s.t., for any morphism $\langle \alpha, \beta \rangle: \langle \Sigma_{1}, \mathcal{X}_{1} \rangle \rightarrow \langle \Sigma_{2}, \mathcal{X}_{2} \rangle$, the satisfaction condition
  $$\langle A, r \rangle \models_{\Sigma_{2}, \mathcal{X}_{2}} \models_{\langle \Sigma_{2}, \mathcal{X}_{2} \rangle}(\langle \alpha, \beta \rangle)(\phi) \text{ iff } \langle A, r \rangle \models_{\Sigma_{1}, \mathcal{X}_{1}|_{\langle \alpha, \beta \rangle}} \models_{\langle \Sigma_{1}, \mathcal{X}_{1} \rangle}(\phi)$$
holds for any valuation $\langle A, r \rangle$ in $\text{Val}(\langle \Sigma_{2}, \mathcal{X}_{2} \rangle)$ and for any sentence $\phi \in \text{Sen}(\langle \Sigma_{2}, \mathcal{X}_{2} \rangle)$.

For any signature $\Sigma$ over $S$, sentences over $\langle \Sigma, \emptyset \rangle$, where $\emptyset$ denotes the empty $S$-sorted set of variables, are called constant sentences over $\Sigma$ (and sentences over $\langle \Sigma, \mathcal{X} \rangle$ with $\mathcal{X} \neq \emptyset$ are called non-constant). Note that, for any $\Sigma$-model $A$, the only valuation of $\emptyset$ into $A$ is $\langle A, \emptyset \rangle$ where $\emptyset$ is the empty $S$-map. In this case we directly write $A \models_{\Sigma, \emptyset} \phi$ instead of $\langle A, \emptyset \rangle \models_{\langle \Sigma, \emptyset \rangle}(\phi)$.

It is immediate to see that if $\mathbf{SF} = \langle \text{Sig}, \text{Sorts}, \text{Mod}, \text{-}, \text{Sen}, \models \rangle$ is a static framework, then $\langle \text{Sig}^{\text{Var}}, \text{Val}, \text{Sen}, \models \rangle$ is an institution (see Definition A.1 in the Appendix A).

Moreover, we can derive from $\mathbf{SF}$ an institution having the same signatures and models by considering sentences as universally quantified, as stated in the proposition below.

In order to illustrate the meaning of this fact, let us consider the static framework of total algebras with equations, defined as follows:
- signatures are algebraic many-sorted signatures;
- the sort functor gives the first component of a signature;
- models are total algebras;
- the carrier gives, for any algebra $A$ over $\Sigma$ with sorts $S$, the underlying $S$-family of sets;
- for any signature with variables $\langle \Sigma, \mathcal{X} \rangle$, $\text{Sen}(\langle \Sigma, \mathcal{X} \rangle)$ gives the equations between $\Sigma$-terms with variables in $\mathcal{X}$;
- the validity of an equation $\phi$ over $\langle \Sigma, \mathcal{X} \rangle$ w.r.t. a valuation $\langle A, r \rangle$ is defined as usual.

From this static framework we can derive two different institutions:
- one where the signatures are the signatures with variables and the models are the valuations;
one where the signatures are the same and sentences are considered as universally quantified over their free variables, as formally shown in the proposition below; in this way what we get is the usual institution of many-sorted algebras with equations. Analogous considerations hold for mostly common institutions.

**Proposition 5.** If $\langle \text{Sig}, \text{Sorts}, \text{Mod}, - | -, \text{Sen}, \vdash \rangle$ is a static framework, then $\langle \text{Sig}, \text{Mod}, \text{Sen}'^\Sigma, \vdash^\Sigma \rangle$ is an institution, where

- $\forall \Sigma \in [\text{Sig}], \Sigma$ over $S$, $\text{Sen}'^\Sigma(\Sigma) = \{ \text{Sen}(\langle \Sigma, X \rangle) | X \in \text{SSet}(S) \}$;
- $\forall \sigma : \Sigma_1 \rightarrow \Sigma_2$ morphism in $\text{Sig}$, the map $\text{Sen}'^\Sigma(\sigma) : \text{Sen}'^\Sigma(\Sigma_1) \rightarrow \text{Sen}'^\Sigma(\Sigma_2)$, denoted by $\sigma$ when there is no ambiguity, is defined as follows:

$$\forall \phi \in \text{Sen}'^\Sigma(\Sigma_1),$$

if $\phi \in \text{Sen}(\langle \Sigma_1, X \rangle)$, then $\sigma(\phi) = \text{Sen}(\langle \sigma, i_X^\Sigma \rangle)(\phi)$

where $i_X^\Sigma$ denotes the injection from $X$ into $\sigma(X)|_\sigma$ (see Definition A.2 in the Appendix A);
- $\forall \Sigma \in [\text{Sig}], \vdash^\Sigma$ is defined by

if $\phi \in \text{Sen}(\langle \Sigma, X \rangle)$, then $A, r \vdash^\Sigma \phi$ iff $\forall (A, r) \in \text{Val}(\langle \Sigma, X \rangle), (A, r) \vdash^\Sigma (\Sigma, X, \phi)$

for any $\Sigma$-model $A$ and for any sentence $\phi$ over $\Sigma$.

**Proof.** Let $\sigma : \Sigma_1 \rightarrow \Sigma_2$ be a signature morphism, $A$ a $\Sigma_2$-model, $\phi$ a sentence over $\Sigma_1$, say $\phi \in \text{Sen}(\langle \Sigma_1, X \rangle)$. We have to prove that the two following conditions are equivalent:

(i) $\forall (A|_{\sigma}, r) \in |\text{Val}(\langle \Sigma_1, X \rangle)|$, $(A|_{\sigma}, r) \vdash^\Sigma (\Sigma_1, X, \phi)$,

(ii) $\forall (A, r) \subset |\text{Val}(\langle \Sigma_2, \sigma(X) \rangle)|$, $(A, r) \vdash^\Sigma (\Sigma_2, \sigma(X), \sigma(\phi))$.

(i) $\Rightarrow$ (ii) For any $(A, r) \in |\text{Val}(\langle \Sigma_2, \sigma(X) \rangle)|$, consider the reduct $(A, r)_{\langle \sigma, i_X^\Sigma \rangle} \vdash^\Sigma \phi$. Then, from the satisfaction condition for $\text{SF}$, $(A, r) \vdash^\Sigma (\langle \Sigma, X \rangle, \phi)$.

(ii) $\Rightarrow$ (i) Conversely, for any $(A|_{\sigma}, r) \in |\text{Val}(\langle \Sigma_1, X \rangle)|$, consider the evaluation $(A, r')$ of $\sigma(X)$ defined by

$$\forall s' \in \text{Sorts}(\Sigma_2),$$

$$\forall x \in (\sigma(X))_{s'}, \text{ if } x \in X, \text{ with } \sigma(s) = s', \text{ then } r'_{s'}(x) = \downarrow (A, r)_s(x).$$

(in categorical terms, $r' = \varepsilon|_{\sigma} \circ \sigma(r)$ where $\varepsilon|_{\sigma}$ is the counit of the adjoint situation $(i^\sigma, e^\sigma) : \sigma_{\downarrow 1_{\sigma}} : \text{SSet}(S_1) \rightarrow \text{SSet}(S_2)$, see the Appendix, Definition A.3).

Then, from (ii), $(A, r') \vdash^\Sigma (\langle \Sigma, X \rangle, \phi)$. Then, from the satisfaction condition for $\text{SF}$, $(A, r')_{\langle \sigma, i_X^\Sigma \rangle} \vdash^\Sigma (\Sigma_1, X, \phi)$, but $(A, r')_{\langle \sigma, i_X^\Sigma \rangle} = (A|_{\sigma}, r)$; indeed, $\forall s \in \text{Sorts}(\Sigma_1), \forall x \in X$, $(r'_{\sigma} \circ i_X^\Sigma)s(x) = \downarrow \sigma(r)_s(x) = \downarrow r_s(x).$ \footnote{Above and in what follows $=_{\downarrow}$ denotes strong equality, i.e. $e_1 =_{\downarrow} e_2$ is true iff either $e_1, e_2$ are both defined and equal, or both undefined.}
3.2. State sets

In this subsection and the following we give a formal definition of dynamic framework based on a given static framework SF. As in the static case models have a carrier which is a sorted set, and taking the carrier means forgetting the static structure (e.g. operations), in the dynamic case models have a carrier which is the class of the possible states, and taking the carrier means forgetting the dynamic structure (e.g. dynamic operations). Hence in this subsection we first formally define state sets. According to informal discussion in Section 2, the state set of a dynamic system with static interface C is a class of arbitrary structures which can be viewed as C-models in the underlying static framework: formally, it is a map from this class of structures into $|Mod(\Sigma)|$.

In what follows, we assume a fixed static framework $SF = \langle \text{Sig}, \text{Sorts}, \text{Mod}, \|\cdot\|, \text{Sen}, \|\cdot\| \rangle$.

The following definition shows how to canonically derive from the model functor $Mod$ in SF a functor $\overline{Mod}$ which gives, for any signature $\Sigma$, the category of the state sets over $\Sigma$.

**Def/Prop. 6.** The state set functor over SF, $\overline{Mod} : \text{Sig}^{op} \rightarrow \text{Cat}$, is defined as follows:

- $\forall \Sigma \in \text{Sig}$
  - an object $\mathcal{A}$ in $\overline{Mod}(\Sigma)$ is a pair $(\text{Dom}(\mathcal{A}), \text{View}_\mathcal{A})$, called a state set (over $\Sigma$), with $\text{Dom}(\mathcal{A})$ class of structures called states and $\text{View}_\mathcal{A} : \text{Dom}(\mathcal{A}) \rightarrow |\text{Mod}(\Sigma)|$;
  - we write $A \in \mathcal{A}$ instead of $A \in \text{Dom}(\mathcal{A})$;
  - a morphism $\varphi : \mathcal{A} \rightarrow \mathcal{B}$ in $\overline{Mod}(\Sigma)$ is a map which associates with any $A \in \mathcal{A}$ a pair $(B, f)$ with $B \in \mathcal{B}$ and $f : \text{View}_\mathcal{A}(A) \rightarrow \text{View}_\mathcal{B}(B)$ a $\Sigma$-morphism;
  - identity and composition are defined componentwise;
- $\forall \sigma : \Sigma_1 \rightarrow \Sigma_2$ morphism in Sig, the reduct functor $\|\cdot\|_\sigma : \overline{Mod}(\Sigma_2) \rightarrow \overline{Mod}(\Sigma_1)$ is defined as follows:
  - $\forall \sigma \in |\text{Mod}(\Sigma_2)|$, $\mathcal{A}|_\sigma = (\text{Dom}(\mathcal{A})|_\sigma, \|\cdot\|_\sigma \circ \text{View}_\mathcal{A})$;
  - $\forall \varphi : \mathcal{A} \rightarrow \mathcal{B}$ morphism in $\overline{Mod}(\Sigma_2)$, $\forall A \in \mathcal{A}|_\sigma$, $\varphi|_\sigma(A) = (B, f|_\sigma)$ if $\varphi(A) = (B, f)$.

The above definition is very general, since states over $\Sigma$ are allowed to be arbitrary structures, with the only requirement that they "can be viewed" as $\Sigma$-models. That allows for instance to use as states even models in a different static framework $SF'$, having some way of mapping models in $SF'$ into models in SF: this possibility will be used in Section 5 in order to canonically extend morphisms of static frameworks to morphisms of dynamic frameworks. Anyway, when working within a fixed static framework, we can take a more concrete definition of $\overline{Mod}$, which we will consider in the examples below. For any signature $\Sigma$, an object $\mathcal{A}$ in $\overline{Mod}(\Sigma)$ is a pair $(\text{Dom}(\mathcal{A}), \text{View}_\mathcal{A})$, where $\text{Dom}(\mathcal{A})$ is a class of $\Sigma^\mathcal{A}$-models, for some signature $\Sigma^\mathcal{A}$, and $\text{View}_\mathcal{A}$ is the $\sigma^\mathcal{A}$-reduct, for some signature morphism $\sigma^\mathcal{A} : \Sigma \rightarrow \Sigma^\mathcal{A}$. In this case, $\mathcal{A}$ can be equivalently represented by the pair $(\text{Dom}(\mathcal{A}), \sigma^\mathcal{A})$, since the reduct
is uniquely determined by the signature morphism; we will use this representation in the examples.

Intuitively, that corresponds to assume that, in a dynamic system $\mathcal{A}$ having static interface $\Sigma$, states are models over an internal signature $\Sigma^d$, and the view of a state as a $\Sigma$-model can be obtained via the reduct functor.

We illustrate now the intuitive meaning of state set morphisms and reducts.

A morphism $\varphi: \mathcal{A} \rightarrow \mathcal{B}$ of state sets over $\Sigma$ maps any state of $\mathcal{A}$, say $A$, into a state of $\mathcal{B}$, say $B$, and moreover maps the $\Sigma$-model corresponding to $A$ into the $\Sigma$-model corresponding to $B$ in a homomorphic way.

As an example, consider the class $\mathcal{C}$ of $\Sigma_6$-algebras of Section 2. This class can be seen as a state set over $\Sigma_6$ taking as signature morphism the identity. Let $\mathcal{C}'$ denote the class of $\Sigma_6$-algebras defined analogously to $\mathcal{C}$, but with the carrier of sort $\text{colour}$ be a singleton set. Then there is a morphism $\varphi$ from $\langle \mathcal{C}, \text{id}_{\Sigma_6} \rangle$ to $\langle \mathcal{C}', \text{id}_{\Sigma_6} \rangle$ defined as follows:

$$\forall A \in \mathcal{C}, \varphi(A) = \langle A', f \rangle,$$

where

$A'$ equal to $A$ except that $A'_{\text{colour}} = \{\bullet\}$, $\text{col}A'(c) = \bullet, \forall c \in A'_{\text{circle}}$;

$f$ is the $\Sigma_6$-morphism which maps $\text{red, green}$ into $\bullet$ and is the identity elsewhere.

Note that the morphism $\varphi$ makes identifications of elements at two levels; first, all the states in the source that differ only for the colour of some circle are mapped in the same state in the target; second, for each state in the source, the elements $\text{red and green}$ are mapped in the same element $\bullet$ in the corresponding state in the target.

Viewing a class of states $\mathcal{A}$ as a discrete category, a morphism $\varphi: \mathcal{A} \rightarrow \mathcal{B}$ of state sets can be defined in categorical terms as a pair $(F\varphi, N\varphi)$, where

- $F\varphi$ is a functor, $F\varphi: \mathcal{A} \rightarrow \mathcal{B}$;
- $N\varphi$ is a natural transformation, $N\varphi: \text{View}_{\mathcal{A}} \rightarrow \text{View}_{\mathcal{B}} \circ F\varphi$.

If $\sigma: \Sigma_1 \rightarrow \Sigma_2$ is a signature morphism, then the reduct w.r.t. $\sigma$ of a state set $\mathcal{A}$ over $\Sigma_2$ keeps the same states of $\mathcal{A}$, but viewed now as $\Sigma_1$-models. For example, if $i$ denotes the inclusion morphism from $\Sigma$ into $\Sigma_6$ of Section 2, then $\langle \mathcal{C}, i \rangle$ is a state set over $\Sigma$.

3.3. Dynamic frameworks

Analogously to the static case, a dynamic framework (over a given static framework) is a logical formalism which has all the components required for being an institution with some additional features, which are

- a signature $D\Sigma$, called a dynamic signature, has a static part $\Sigma$ which is a signature in the underlying static framework; intuitively, that models the fact that the interface of a dynamic system consists of two parts giving the observations and the modifications, respectively, one can perform on the system;
- a model over $D\Sigma$, called a dynamic model, has a carrier which is a state set over $\Sigma$;
sentences (called *dynamic sentences*) are of two kinds, constant and non-constant. Analogously to the static case, constant dynamic sentences over \( D\Sigma \) are evaluated just w.r.t. a \( D\Sigma \)-model, while non-constant dynamic sentences over \( \langle D\Sigma, X \rangle \) are evaluated w.r.t. a valuation. In this case, accordingly with the intuition, a valuation is a triple \( \langle \mathcal{A}, A, r \rangle \) where \( \mathcal{A} \) is a \( D\Sigma \)-model, \( A \) is a state of \( \mathcal{A} \) (intuitively the current state in which to evaluate the sentence) and \( \langle \text{View}_{\mathcal{A}}(A), r \rangle \) is a (static) valuation of \( X \) into the \( \Sigma \)-model corresponding to \( A \).

The formal definitions follow.

**Definition 7.** A *dynamic model part* based on \( SF \) is a 4-tuple \( \langle \text{DSig}, St, DMod, |\cdot| \rangle \) where

1. \( \text{DSig} \) and \( DMod \) are like \( \text{Sig} \) and \( \text{Mod} \) in Definition 1;
2. \( St: \text{DSig} \to \text{Sig} \) is a functor, called a *static part* functor; for any dynamic signature \( D\Sigma \), \( St(D\Sigma) \) is called the *static part* of \( D\Sigma \), and analogously for a morphism;
3. \( |\cdot| \) is a natural transformation, \( |\cdot|: DMod \to \text{Mod} \circ (St)^{op} \), s.t., for any dynamic signature \( D\Sigma \), the functor \( |\cdot|_{D\Sigma} \) is faithful; for any \( D\Sigma \)-model \( \mathcal{A} \), \( |\mathcal{A}|_{D\Sigma} \) is called the *carrier* of \( \mathcal{A} \), and denoted by \( |\mathcal{A}| \) or even \( \mathcal{A} \) when there is no ambiguity, and analogously for morphisms.

Accordingly to the terminology of [1], for any dynamic signature \( D\Sigma \) with static part \( \Sigma \), \( \langle DMod(D\Sigma), |\cdot|_{D\Sigma} \rangle \) is a concrete category over \( \text{Mod}(\Sigma) \), i.e. the natural transformation \( |\cdot|: DMod \to \text{Mod} \circ (St)^{op} \) plays the same role of \( |\cdot|: \text{Mod} \to \text{SSet} \circ (\text{Sorts})^{op} \) in the static case. The assumption that \( |\cdot|_{D\Sigma} \) is faithful models the fact that morphisms of dynamic models are morphisms of state sets which respect some condition.

We introduce now, analogously to what we have done for a static model part, dynamic valuations. Note that, since \( \text{Sorts} \circ St \) is a sort functor for \( \text{DSig} \), the category of *dynamic signatures with variables* \( \text{DSig}^{\text{Var}} \) is defined by Definition 2. Taking \( \text{Sorts} \circ St \) as sort functor for \( \text{DSig} \) corresponds to assume that the sorts of a dynamic signature are the sorts of its static part. Indeed, the intuition is that the static part of a dynamic signature describes the structure of the possible states of the system (hence also which are the sorts of the elements which can exist in each state), while the dynamic signature itself enriches this information by describing the possible transformations from a state into another.

**Notations.** For any dynamic signature with variables \( \langle D\Sigma, X \rangle \), \( D\Sigma \) with static part \( \Sigma \), \( \mathcal{A} \) model over \( D\Sigma \), set \( \text{Val}_{\mathcal{A}}^{\Sigma}(X) = \{ (A, r) | A \in \mathcal{A}, r: X \to |\text{View}_{\mathcal{A}}(A)| \} = \{ (A, r) | A \in \mathcal{A}, \langle \text{View}_{\mathcal{A}}(A), r \rangle \in |\text{Val}(\langle \Sigma, X \rangle)| \} \).

**Def Prop. 8.** Let \( \langle \text{DSig}, St, DMod, |\cdot| \rangle \) be a dynamic model part based on \( SF \). Then the functor

\[ DVal: (\text{DSig}^{\text{Var}})^{op} \to \text{Cat} \]

is defined as follows:
$\forall (D\Sigma, X) \in |D\Sigma^{\text{Var}}|$, $D\Sigma$ with static part $\Sigma$, $D\text{Val}((D\Sigma, X))$ is the category where an object, called a dynamic valuation of $X$ (into $\mathcal{A}$), is a triple $\rho = \langle \mathcal{A}, A, r \rangle$, with $\mathcal{A}$ model over $D\Sigma$ and $(A, r) \in \text{Val}^{X}(\Sigma, X)$, and $\varphi : \langle \mathcal{A}, A, r \rangle \to \langle B, B, r_B \rangle$ is a morphism iff $\varphi : \mathcal{A} \to \mathcal{B}$ is a $D\Sigma$-morphism, $\varphi(A) = (B, f)$ and $f$ is a morphism from $\langle \text{View}, \mathcal{A}, r \rangle$ into $\langle \text{View}, \mathcal{B}, r_B \rangle$ in $\text{Val}(\Sigma, X)$ (i.e. $r_B = r_A \circ f$);

$\forall (d\sigma, h) : (D\Sigma_1, X_1) \to (D\Sigma_2, X_2)$ morphism in $D\Sigma^{\text{Var}}$, $d\sigma$ with static part $\sigma : \Sigma_1 \to \Sigma_2$,

- $\forall \langle \mathcal{A}, A, r \rangle \in |D\text{Val}((D\Sigma_2, X_2))|$, $\langle \mathcal{A}, A, r \rangle_{(d\sigma, h)} = \langle \mathcal{A}_{|d\sigma}, A, r_{|d\sigma} \circ h \rangle$;
- $\forall \varphi : \langle \mathcal{A}, A, r_A \rangle \to \langle \mathcal{B}, B, r_B \rangle$ morphism in $D\text{Val}((D\Sigma_2, X_2))$, $\varphi_{(d\sigma, h)}$ is the $D\Sigma_1$-morphism $\varphi_{|d\sigma} : \mathcal{A}_{|d\sigma} \to \mathcal{B}_{|d\sigma}$.

\textbf{Definition 9.} A dynamic framework based on SF is a 6-tuple $(D\Sigma, St, D\text{Mod}, \vdash, D\text{Sen}, \vdash)$ where

- $(D\Sigma, St, D\text{Mod}, \vdash)$ is a dynamic model part based on SF;
- $D\text{Sen}$ is a functor, $D\text{Sen} : D\Sigma^{\text{Var}} \to \text{Set}$; for any $D\Sigma \in |D\Sigma|$ (resp. $(D\Sigma, X) \in |D\Sigma^{\text{Var}}|$), the elements of $D\text{Sen}(D\Sigma)$ (resp. $D\text{Sen}((D\Sigma, X))$) are called constant dynamic sentences over $D\Sigma$ (resp. non-constant dynamic sentences over $(D\Sigma, X)$); for any morphism of dynamic signatures $d\sigma : D\Sigma_1 \to D\Sigma_2$, the satisfaction condition

$$\mathcal{A} \vdash_{D\Sigma_2} d\sigma(d\phi) \text{ iff } \mathcal{A}_{|d\sigma} \vdash_{D\Sigma_1} d\phi$$

holds for any dynamic model $\mathcal{A}$ in $D\text{Mod}(D\Sigma_2)$ and for any sentence $d\phi \in D\text{Sen}(D\Sigma_1)$;

- $\forall (D\Sigma, X) \in |D\Sigma^{\text{Var}}|$, $\vdash_{(D\Sigma, X)}$ is a relation over $|D\text{Mod}(D\Sigma)| \times D\text{Sen}(D\Sigma)$ s.t., for any morphism $d\sigma : D\Sigma_1 \to D\Sigma_2$, the satisfaction condition

$$d\sigma \vdash_{(D\Sigma, X_1)} (d\sigma, h)(d\phi) \text{ iff } d\sigma \vdash_{(D\Sigma, X_1)} (d\sigma, h)(d\phi)$$

holds for any dynamic valuation $\rho$ in $D\text{Val}((D\Sigma_2, X_2))$ and for any sentence $d\phi \in D\text{Sen}((D\Sigma_1, X_1))$.

Here above $D\Sigma + D\Sigma^{\text{Var}}$ denotes the sum category (coproduct in $\text{Cat}$); analogously below $D\text{Mod} + D\text{Val}$ denotes the coproduct of functors.

Note that, differently from the static case, constant sentences do not coincide with sentences without free variables. For instance, referring to our running example,

$$\{c \leftarrow \text{start}\} X(c) = 0 \land Y(c) = 0 \land \text{radius}(c) = 1 \land \text{col}(c) = \text{green}$$

is a constant sentence, intuitively stating a property of the constant dynamic operation $\text{start}$, while

$$\langle \mathcal{A}c \rangle \text{col}(c) = \text{green}$$

has no free variables, but of course its truth value depends on the current state.
Hence we need to keep explicitly the two kinds of sentences in a dynamic framework. Note that if DF = \langle DSig, St, DMod, \vdash, DSen, \vdash \rangle is a dynamic framework, then
\langle DSig + DSig^{\scriptscriptstyle Var}, DMod + DVal, DSen, \vdash \rangle
is an institution. Moreover, we can derive from DF an institution having the same signatures and models by considering sentences as universally quantified, as stated below.

**Proposition 10.** If DF = \langle DSig, St, DMod, \vdash, DSen, \vdash \rangle is a dynamic framework, then \langle DSig, DMod, DSen^{\scriptscriptstyle \forall}, \vdash^{\scriptscriptstyle \forall} \rangle is an institution, where
- \forall DS \in |DSig|, DS over S, DSen^{\scriptscriptstyle \forall}(\Sigma) = \cup \{DSen((DS,X)) \mid X \in |SSet(S)|\} \cup DSen(DS);
- \forall d\sigma: \Sigma_1 \rightarrow \Sigma_2 morphism in DSig, d\sigma with static part \sigma, the map
  \[ DSen^{\scriptscriptstyle \forall}(d\sigma): DSen^{\scriptscriptstyle \forall}(\Sigma_1) \rightarrow DSen^{\scriptscriptstyle \forall}(\Sigma_2), \]
denoted by d\sigma when there is no ambiguity, is defined as follows:
  \[ \forall d\phi \in DSen^{\scriptscriptstyle \forall}(\Sigma_1), \]
  if d\phi \in DSen(\Sigma_1), then DSen^{\scriptscriptstyle \forall}(d\sigma)(d\phi) = DSen(d\sigma)(d\phi);
  if d\phi \in DSen((\Sigma_1,X)), then DSen^{\scriptscriptstyle \forall}(d\sigma)(d\phi) = DSen((\sigma,i^\sigma_X))(\phi);
where i^\sigma_X denotes the injection from X into \sigma(X) (see Definition A.2 in Appendix A);
- \forall DS \in |DSig|, DS with static part \Sigma, \vdash^{\scriptscriptstyle \forall} is defined by
  if d\phi \in DSen(\Sigma), then \mathcal{A} \vdash^{\scriptscriptstyle \forall}_{DS} d\phi iff \mathcal{A} \vdash_{DS} d\phi;
  if d\phi \in DSen(\Sigma,X), then \mathcal{A} \vdash^{\scriptscriptstyle \forall}_{DS} d\phi iff \forall (A,r) \in Val^{\mathcal{A}}((\Sigma,X)),
  \mathcal{A} \vdash_{(DS,X)} d\phi
for any \Sigma-model \mathcal{A} and sentence d\phi over DS.

4. A parameterized dynamic framework

We define now a particular dynamic framework DF(SF) = \langle DSig, St, DMod, \vdash, DSen, \vdash \rangle constructed in a canonical way on top of a given static framework SF. First we define the model part. The basic idea is that dynamic signatures are pairs consisting of a static signature and a family of so-called *dynamic operation symbols* (like usual operation symbols, but with an hidden parameter, the state); correspondingly, dynamic models are state sets enriched by dynamic operations, which are, roughly speaking, transformations of states. Dynamic models of this kind have been firstly introduced in [4, 5] under the name of d-oids; we keep here the same name, since the basic idea is the same, even if the version presented in this paper is modified in order to fill
in the institutional framework (states are not required to be static models, as in the
original definition, but are allowed to be arbitrary structures which can be viewed as
static models, as defined in Definition 6).

4.1. D-oids

If \( S \) is a set, then \([S]\) denotes \( S \cup \{A\} \); we use \([s]\) for ranging over \([S]\), i.e. \([s]\)
stands for either an element of \( S \) or for the empty string.

**Definition 11.** A *d-oid signature* is a pair \((\Sigma, DOP)\) where \( \Sigma \) is a static signature and
\( DOP \) is an \([S] \cup (S^* \times [S])\)-sorted set of *dynamic operation symbols*; if \( dop \in DOP_{[s]} \),
then we write \( dop: [s] \), and say that \( dop \) is a *constant* dynamic operation symbol; if \( dop \in DOP_{s_1 \ldots s_n [s]} \),
then we write \( dop: s_1 \ldots s_n \Rightarrow [s] \). We denote by \( \text{DSig} \) the category
of d-oid signatures, defined in the obvious way, and by \( \text{St} \) the functor giving the first
component.

Here and in what follows, \( \bar{x} \) denotes the tuple \( x_1, \ldots, x_n \).

**Definition 12.** Let \( D\Sigma = \{\Sigma, DOP\} \) be a d-oid signature. For any \( \mathcal{A} \) state set over \( \Sigma \),
\( w \in S^* \), set \( \mathcal{A}_w = \{\langle A, \bar{a} \rangle \mid A \in \mathcal{A}, \bar{a} \in (\text{View}_\mathcal{A}(A))_w \} \).

Then, a *d-oid* over \( D\Sigma \) is a pair \( \mathcal{A} = \{[\mathcal{A}], \{dop^\mathcal{A}\}_{dop \in DOP} \} \) where:

- \([\mathcal{A}]\) is a state set over \( \Sigma \) (denoted simply by \( \mathcal{A} \) when there is no ambiguity);
- \( \forall dop: [s], dop^\mathcal{A} \in [\mathcal{A}]; dop^\mathcal{A} \) is called a *constant dynamic operation*;
- \( \forall dop: w \Rightarrow [s], dop^\mathcal{A} \) is a map which associates with any \( \langle A, \bar{a} \rangle \in \mathcal{A}_w \) a transfor-
mation of \( A \), i.e. a triple \( \langle B, f, [b] \rangle \) where \( \langle B, [b] \rangle \in \mathcal{A}_w \) and \( f: |\text{View}_\mathcal{A}(A)| \rightarrow |\text{View}_\mathcal{A}(B)| \); we write \( dop^\mathcal{A}(\langle A, \bar{a} \rangle) = \langle f: A \Rightarrow B, [b] \rangle \); \( dop^\mathcal{A} \) is called a (non-
constant) dynamic operation, and \( f \) is called tracking map.

A definition of dynamic operation in categorical terms can be given viewing, for any
\( w \in S^* \), \( \mathcal{A}_w \) as a discrete category, and defining the functor \( \text{View}_\mathcal{A}^w: \mathcal{A}_w \rightarrow \text{Mod}(\Sigma) \)
by \( \text{View}_\mathcal{A}^w(\langle A, \bar{a} \rangle) = \text{View}_\mathcal{A}(A) \); then, for any \( dop: w \Rightarrow [s], dop^\mathcal{A} \) can be viewed as a pair \( \langle F_{dop}^\mathcal{A}, N_{dop}^\mathcal{A} \rangle \), where

- \( F_{dop}^\mathcal{A} \) is a functor, \( F_{dop}^\mathcal{A}: \mathcal{A}_w \rightarrow [\mathcal{A}]; \)
- \( N_{dop}^\mathcal{A} \) is a natural transformation, \( N_{dop}^\mathcal{A}: |\Sigma| \circ \text{View}_\mathcal{A}^w \rightarrow |\Sigma| \circ \text{View}_\mathcal{A} \circ F_{dop}^\mathcal{A} \).

As illustrated by the example in Section 2, in concrete applications it is useful
to assume basic values like integers, booleans and so on, which can be viewed as
"constant" entities, that exist in each state and never change. If one wants to stress the
difference, then a specialized version of d-oids can be adopted, with an explicit "value
part" (see [5]). The corresponding theory is straightforward and does not introduce any
novelty; hence here, in a paper more devoted to a basic mathematical presentation, we
prefer to give a unified treatment which does not distinguish between pure values and
proper objects.

The notion of morphism for d-oids is perfectly analogous with the static classical
case: it is a morphism of state sets compatible with the (dynamic) operations. Roughly
speaking, if \( \varphi : \mathcal{A} \rightarrow \mathcal{B} \), then, for any \( \langle A, \bar{a} \rangle \in \mathcal{A}_w \), applying first \( \varphi \) and then \( \text{dop}^\# \) must give the same result of first applying \( \text{dop}^\# \) and then \( \varphi \), for any dynamic operation symbol \( \text{dop} \).

In this case, however, this commutativity is at two levels: first, we get the same state in \( \mathcal{B} \), say \( B' \) (possibly with a result \( b' \)) and, second, the elements of \( \text{View}_{\varphi}(A) \) are transformed into the same elements of \( \text{View}_{\text{dop}}(B') \).

**Def/Prop. 13.** The \( d \)-oids over a \( d \)-oid signature \( D\Sigma \) form a category \( \overline{D\text{Mod}}(D\Sigma) \) taking as morphisms the morphisms \( \varphi : \mathcal{A} \rightarrow \mathcal{B} \) of underlying state sets s.t. the following conditions hold:

(i) \( \forall \text{dop} : s \),

if \( \text{dop}^\# = \langle A, a \rangle \) and \( \varphi(A) = \langle B, f_A \rangle \),

then \( \text{dop}^\# - \langle B, b \rangle \), [and, if \( f_A(a) \) is defined, then \( f_A(a) = b \)];

(ii) \( \forall \text{dop} : w \Rightarrow [s] \), \( \langle A, \bar{a} \rangle \in \mathcal{A}_w \),

if \( \text{dop}^\#(\langle A, \bar{a} \rangle) = \langle f : A \Rightarrow A', a' \rangle \), \( \varphi(A) = \langle B, f_A \rangle \) and \( f_A(\bar{a}) = \bar{b} \),

then

\[
\text{dop}^\#(\langle B, \bar{b} \rangle) = \langle g : B \Rightarrow B', b' \rangle \\
\varphi(A') = \langle B', f_A' \rangle , g \circ |f_A| = |f_A'| \circ f;
\]

[and, if \( f_A'(a') \) is defined, then \( f_A'(a') = b' \)].

A precise categorical formulation of (ii) is the commutativity of the two diagrams in Figs. 1 and 2, where \( F^w_\varphi : \mathcal{A}_w \rightarrow \mathcal{B}_w \) is the functor defined by

\[
F^w_\varphi(\langle A, \bar{a} \rangle) = \langle B, \bar{b} \rangle \quad \text{iff} \quad F\varphi(A) = B, \ |(N\varphi)_A|\langle \bar{a} \rangle = \bar{b}
\]
and $N^w_\varphi : \text{View}^w_\varphi \to \text{View}^w_\varphi \circ F^w_\varphi$ is the natural transformation defined by $(N^w_\varphi)_{(A,\bar{a})} = (N_\varphi)_A$.

Note that the natural transformation in the low horizontal arrow of Fig. 2 is well-defined since $F_{\text{dop}} \circ F^w_\varphi = F^{[s]}_\varphi \circ F_{\text{dop}}$, by the commutativity of Fig. 1.

**Def/Prop. 14.** For any $d$-oid signature morphism $d\sigma : D\Sigma_1 \to D\Sigma_2$ with static part $\sigma$, the reduct functor $-|_{d\sigma} : \overline{D\text{Mod}}(D\Sigma_2) \to \overline{D\text{Mod}}(D\Sigma_1)$ is defined enriching the corresponding reduct of state sets by the interpretation of the dynamic operations:

$$
\forall \mathcal{A} \in \overline{D\text{Mod}}(D\Sigma_2),
\forall \text{dop} : s \text{ in } D\Sigma_1,
\text{dop}^{(\sigma|_{d\sigma})} \rightarrow d\sigma(\text{dop})^{\mathcal{A}},
\forall \text{dop} : w \Rightarrow [s] \text{ in } D\Sigma_1, \langle A, \bar{a} \rangle \in \mathcal{A}|_{d\sigma},
\text{dop}^{(\sigma|_{d\sigma})}(\langle A, \bar{a} \rangle) = \langle f|_{s} : A \Rightarrow B[s, b] \rangle \text{ iff } d\sigma(\text{dop})^{\mathcal{A}}(\langle A, \bar{a} \rangle) = \langle f : A \Rightarrow B[s, b] \rangle.
$$

Altogether, we have defined the functor $\overline{D\text{Mod}} : \text{DSig}^{\text{op}} \to \text{Cat}$.

Moreover, let $|-| : \overline{D\text{Mod}} \to \text{Mod} \circ (\text{St})^{\text{op}}$ be the natural transformation mapping $d$-oids in underlying state sets.

**Fact 15.** The $4$-tuple $(\text{DSig}, \text{St}, \overline{D\text{Mod}}, |-|)$ is a dynamic model part.

We define now dynamic terms and dynamic sentences.

### 4.2. Dynamic terms and their evaluation

Analogously to what is usually done in the static case, we can define dynamic terms built by variables and dynamic operation symbols, denoting intuitively derived state transformations. In the dynamic case, the basic way of composing terms is sequential composition (possibly with value passing). Hence a dynamic term is of the form

$$[x_1 \leftarrow \text{dop}_1(\bar{x}_1); \ldots; x_n \leftarrow \text{dop}_n(\bar{x}_n)],$$

and denotes intuitively the execution in sequence of $\text{dop}_1, \ldots, \text{dop}_n$; at each step, $x_i$ is present only if $\text{dop}_i$ has a result sort, is a variable of this sort, which may be used in the sequel for denoting the entity returned as final result of $\text{dop}_i$; in other words, $x_i$ is a binding variable with scope

$$[x_{i+1} \leftarrow \text{dop}_{i+1}(\bar{x}_{i+1}); \ldots; x_n \leftarrow \text{dop}_n(\bar{x}_n)].$$

An example of dynamic term is, referring to our toy graphical system,

$$c_2 \leftarrow \text{copy}(c_1); n \leftarrow \text{delGreen}; \text{move}(c_2, n, n),$$

denoting intuitively an execution sequence in which first a new circle $c_2$ is created as a copy of an existing circle denoted by $c_1$, then all the existing $n$ green circles are deleted, and finally $c_2$ is moved both horizontally and vertically by $n$. This term has one free variable $c_1$. Note that if $c_1$ is a green rectangle, hence $c_2$ is created green.
too, then we expect this execution sequence to produce a run-time error; formally, the evaluation of this term w.r.t. a valuation \( \langle \emptyset, A, r \rangle \) where \( \text{col}(r(c_1)) = \text{green} \) is undefined (see below).

We have described until now \textit{non-constant} dynamic terms, which can be seen as derived non-constant dynamic operations. Corresponding to constant dynamic operations, we define also \textit{constant dynamic terms}, of the form

\[
[x_0 \leftarrow \text{dop}_0; [x_1 \leftarrow \text{dop}_1(\bar{x}_1); \ldots; [x_n \leftarrow \text{dop}_n(\bar{x}_n)],
\]

where \( \text{dop}_0 \) is a constant dynamic operation and \( \text{dop}_1, \ldots, \text{dop}_n \) are non-constant dynamic operations, which denote intuitively the execution in sequence of \( \text{dop}_1, \ldots, \text{dop}_n \) starting from the initial state denoted by \( \text{dop}_0 \).

For instance, \( c \leftarrow \text{start}; c' \leftarrow \text{copy}(c) \) is a constant dynamic term denoting intuitively the state in which two circles exist, both with center in the origin, radius one and green colour.

Constant dynamic terms cannot contain free variables. Note, anyway, that, as for dynamic sentences, constant dynamic terms are different from non-constant dynamic terms without free variables. For instance, \( n \leftarrow \text{delGreen} \) has no free variables, but is a non-constant dynamic term.

Dynamic terms have been introduced in [6], where it is proved that, choosing a syntax with the property of being a unique canonical representation, they are a free structure. Here we prefer a more suggestive notation with explicit binding variables, as shown above; for simplifying the presentation avoiding problems related to name clashes, we assume in what follows that in a dynamic term all binding variables are distinct and that free variables and binding variables are disjoint sorted sets.

Moreover in [6] an extended version of dynamic terms is presented which is convenient for concrete applications, i.e., assuming that in the underlying static framework static terms are available, dynamic terms built on top of static terms, like e.g.

\[ \text{move}(c_2, X(c_1), Y(c_2)) \]

Anyway we skip here this extension, which is straightforward, for shortening the presentation.

\textbf{Notations.} By \( A \) we denote the empty string, with the convention that \( dt; A = dt \). If \( X \) is an \( S \)-sorted set and \( s \in S \) and \( x \not\in \text{Dom}(X) \), then \( X\{x \mapsto s\} \) denotes the \( S \)-sorted set defined by \( X\{x \mapsto s\}_s = X_s \cup \{x\}, \ X\{x \mapsto s\}_{s'} = X_{s'} \), for all \( s' \neq s \). We abbreviate \( \emptyset\{x \mapsto s\} \) by \( \{x \mapsto s\} \).

\textbf{Definition 16.} Let \( \langle D \Sigma, X \rangle \) be a d-oid signature with variables. Then, the sets of the \textit{constant dynamic terms} over \( D \Sigma \) and \textit{non-constant dynamic terms} over \( D \Sigma \) and \( X \), denoted \( DT(D \Sigma), DT(\langle D \Sigma, X \rangle) \), respectively, are inductively defined in Fig. 3, where we write

\[
X \vdash dt \quad \text{for} \ dt \in DT(\langle D \Sigma, X \rangle),
\]

\( \vdash dt \quad \text{for} \ dt \in DT(D \Sigma). \)
For any dynamic term $dt$, we denote by $\text{Var}(dt)$ the sorted set of the binding variables of $dt$, defined in the obvious way.

We define now the evaluation of dynamic terms. According to the intuition, the evaluation of a non-constant dynamic term over $D\Sigma$ and $X$, say

$$[x_1 \leftarrow \text{dop}_1(\overline{x}_1); \ldots; [x_n \leftarrow \text{dop}_n(\overline{x}_n)],$$

is performed w.r.t. a d-oid $\mathcal{A}$, a state $A$ of $\mathcal{A}$ as initial state and a mapping $r_A$ of the free variables $X$ into (the $\Sigma$-model corresponding to) $A$ (this triple is what we have called a dynamic valuation). The result of the evaluation is a final state $B$ and a valuation into (the $\Sigma$-model corresponding to) $B$ of the variables $X \cup \text{Var}(dt)$ where $\text{Var}(dt) = \{x_1, \ldots, x_n\}$ are the new variables introduced by the dynamic term.

The evaluation of a constant dynamic term w.r.t. a d-oid $\mathcal{A}$ gives a state $B$ and a mapping of $\text{Var}(dt)$ into (the $\Sigma$-model corresponding to) $B$.

The formal definition is given below. We denote by $\overline{\text{Val}}$ the functor obtained from $\overline{\text{Mod}}$ by the canonical extension of Def/Prop. 8.

Notations. If $r : X \to B$ is an $S$-sorted map, $b \in B_x$, then $r\{x \leftrightarrow b\} : X\{x \leftrightarrow s\} \to R$ is the $S$-sorted map defined by $r\{x \leftrightarrow b\}(x) = b$, $r\{x \leftrightarrow b\}(y) = r_s'(y)$, $\forall (y, s') \neq \langle x, s \rangle$.

**Definition 17.** Let $\langle D\Sigma, X \rangle$ be a d-oid signature with variables, $\rho = \langle \mathcal{A}, A, r_A \rangle \in \overline{\text{Val}}(\langle D\Sigma, X \rangle)$. Then the **evaluation** of non-constant dynamic terms over $\langle D\Sigma, X \rangle$ w.r.t. $\rho$, $\lll \lll_{(D\Sigma, X)}$, and of constant dynamic terms over $D\Sigma$ w.r.t. $\mathcal{A}, \lll \lll'_{D\Sigma}$, are inductively defined in Fig. 4, where we write

$$dt \xrightarrow{\langle A, r_A \rangle} (B, r_B) \text{ for } (\lll dt\rrl_{(D\Sigma, X)}) = (B, r_B),$$

$$dt \xrightarrow{\langle A, r_A \rangle} (B, r_B) \text{ for } (\lll dt\rrl'_{(D\Sigma, X)}) = (B, r_B).$$

In order to show how the evaluation works, let us evaluate the dynamic term

$$c_2 \leftarrow \text{copy}(c_1); \text{changeCol}(c_2).$$

w.r.t. the d-oid $\mathcal{C}$ described in Section 2, an initial state $A$ and a mapping $r_A$ s.t. $r_A(c_1) = \gamma \in A_{\text{circle}}$. 
Fig. 4. Evaluation of dynamic terms.

We have that

\[ \text{copy}^g(\langle A, \gamma \rangle) = \langle e_{A,B}: A \Rightarrow B, \gamma' \rangle, \]

where \( e_{A,B} \) denotes the embedding (family of set inclusions) from \(|A|\) into \(|B|\).

Hence, instantiating the first of rules (ii),

\[
\begin{align*}
\frac{\text{changeCol}(c_2) \quad \langle B(e_{A,B} \circ r_A)\{c_2 \mapsto \gamma'\} \rangle}{c_2 \leftarrow \text{copy}(c_1); \text{changeCol}(c_2) \quad \langle A, r_A \rangle \quad \langle C, r_C \rangle}
\end{align*}
\]

Now, instantiating the first of rules (iii) and recalling that \( \text{changeCol}(c_2) \) stands for \( \text{changeCol}(c_2); A \),

\[
\begin{align*}
\frac{\text{changeCol}(c_2) \quad \langle C, r_C \rangle}{\text{changeCol}(c_2) \quad \langle B(e_{A,B} \circ r_A)\{c_2 \mapsto \gamma'\} \rangle \quad \langle B, r_B \rangle \quad \langle C, r_C \rangle}
\end{align*}
\]

where \( r_C = \text{id}_{|B|} \circ (e_{A,B} \circ r_A)\{c_2 \mapsto \gamma'\} \), \( C \) is the same of \( B \) except for \( \text{col}(\gamma') \) (in particular \(|C| = |B|\)) and the premise can be immediately obtained by instantiating rule (i). Hence we get as final result the state \( C \) and the mapping \( r_C \) which maps \( c_1 \) into \( \gamma \) and \( c_2 \) into \( \gamma' \).
In order to see an example of undefined evaluation (modeling a run-time error), let us now evaluate the term

\[ n \leftarrow \text{delGreen; move}(c, n, n), \]

w.r.t. \( \mathcal{G} \), an initial state \( A \) and the mapping \( r_A(c) = \gamma \), with \( \text{col}^A(\gamma) = \text{green} \).

The effect of the dynamic operation \( \text{delGreen} \) is to delete the green circles. Formally,

\[ \text{delGreen}^A(A) = \langle p_{A,B} : A \Rightarrow B, z \rangle \]

where \( B \) is the same of \( A \) except that \( B_{\text{circle}} = A_{\text{circle}} \setminus \{ \gamma \mid \text{col}^A(\gamma) = \text{green} \} \), \( z \) is the cardinality of \( \{ \gamma \mid \text{col}^A(\gamma) = \text{green} \} \) and \( p_{A,B} \) is the partial map being the identity on the red circles and undefined on the green. Hence, applying the first of rules (iii), the term \( \text{move}(c, n, n) \) must be evaluated w.r.t. the mapping \( p_{A,B} \circ r_A \) which is undefined on \( c \), hence the evaluation is undefined.

It is straightforward (see Definition A.5 in the Appendix) to define the renaming of dynamic terms, which is, for any \( d\sigma : D\Sigma_1 \rightarrow D\Sigma_2 \) (resp., \( (d\sigma, h) : \langle D\Sigma_1, X_1 \rangle \rightarrow \langle D\Sigma_2, X_2 \rangle \)) morphism of d-oid signatures (resp. d-oid signatures with variables), a map

\[ DT(d\sigma) : DT(D\Sigma_1) \rightarrow DT(D\Sigma_2) \]

(resp. \( DT((d\sigma, h)) : DT(\langle D\Sigma_1, X_1 \rangle) \rightarrow DT(\langle D\Sigma_2, X_2 \rangle) \))

denoted by \( d\sigma \) (resp. \( (d\sigma, h) \)) when there is no ambiguity.

The following proposition will be needed later (Theorem 21) in order to prove that the satisfaction condition holds for the formalism we have constructed.

**Proposition 18.** Under the assumptions of Definition A.5, and denoting by \( \sigma \) the static part of \( d\sigma \),

\[ \forall dt \in DT(D\Sigma_1), \forall \mathcal{A} \in |\text{DMod}(D\Sigma_2)|, \| dt \|^\mathcal{A} = \langle B, (r_B)_{|\sigma} \rangle \iff \| d\sigma(dt) \|^\mathcal{A} = \langle B, r_B \rangle \]

\[ \forall dt \in DT(\langle D\Sigma_1, X_1 \rangle), \forall \rho \in |\text{DVal}(\langle D\Sigma_2, X_2 \rangle)|, \| dt \|^\rho = \langle B, (r_B)_{|\sigma} \circ (h + \text{id}_{\text{var}(dt)}) \rangle \iff \| (d\sigma, h)(dt) \|^\rho = \langle B, r_B \rangle. \]

**4.3. Sentences**

**Definition 19.** For any d-oid signature \( D\Sigma \) (resp., d-oid signature with variables \( \langle D\Sigma, X \rangle \)), \( D\Sigma \) with static part \( \Sigma \), the set \( \overline{\text{DSen}(D\Sigma)} \) (resp. \( \overline{\text{DSen}(\langle D\Sigma, X \rangle)} \)) of the constant sentences over \( D\Sigma \) (resp. non-constant sentences over \( \langle D\Sigma, X \rangle \)) is defined in Fig. 5, where we write

\[ \vdash d\phi \quad \text{for} \quad d\phi \in \overline{\text{DSen}(D\Sigma)} \]

\[ X \vdash d\phi \quad \text{for} \quad d\phi \in \overline{\text{DSen}(\langle D\Sigma, X \rangle)} \].

**Definition 20.** Under the assumptions of Definition 17, and denoting by \( \Sigma \) the static part of \( D\Sigma \), the satisfaction of constant dynamic sentences over \( D\Sigma \) w.r.t. \( \mathcal{A} \), \( \mathcal{A} \models_{D\Sigma} \)
Fig. 6. Satisfaction of dynamic sentences.

(i) \( X \vdash \phi \iff \phi \in \text{Sen}(\langle \Sigma, X \rangle) \)

(ii) \( X \vdash \{\phi_1\} dt \{\phi_2\} \iff \phi_1 \in \text{Sen}(\langle \Sigma, X \rangle), dt \in \text{DT}(\langle \Delta \Sigma, X \rangle), \phi_2 \in \text{Sen}(\langle \Sigma, X \cup \text{Var}(dt) \rangle) \)

\( \{dt\} \phi \quad \text{if } dt \in \text{DT}(\Delta \Sigma), \phi \in \text{Sen}(\langle \Sigma, \text{Var}(dt) \rangle) \)

(iii) \( X \vdash dt_1 = dt_2 \iff dt_1, dt_2 \in \text{DT}(\Delta \Sigma), \text{Var}(dt_1) = \text{Var}(dt_2) \)

Fig. 5. Dynamic sentences.

and of non-constant dynamic sentences over \( \langle \Delta \Sigma, X \rangle \) w.r.t. \( \rho \), \( \rho \models_{\langle \Delta \Sigma, X \rangle} \), are inductively defined in Fig. 6.

We recall now the examples of sentences given in Section 2 for our toy graphical system.

The sentence \( I = \)

\( X(c_1) = X(c_2) \land Y(c_1) = Y(c_2) \supset \text{col}(c_1) = \text{col}(c_2) \)

is obtained by rule (i) in Fig. 5 and its satisfaction is defined by rule (i) in Fig. 6 by stating that \( I \) is valid w.r.t.a d-oid \( \mathcal{A} \), a current state \( A \) and a mapping \( r_A : X \rightarrow \text{View}_\mathcal{A}(A) \) iff \( I \) is valid w.r.t. \( \text{View}_\mathcal{A}(A), r_A \) in the sense defined by \( \mathcal{S} \) (in this case, validity in usual first-order logic). Recall that, following Def/Prop. 6, a state \( A \) is not directly a \( \Sigma \)-algebra, but there is mapping \( \text{View}_\mathcal{A}(\cdot) \) which associates with each state a \( \Sigma \)-algebra.

If we consider \( I \) as implicitly universally quantified over the possible states (as formally stated in Proposition 10), this sentence expresses an invariant, i.e. a requirement which must hold in all the states of the system.

The sentences

\( \{X(c) = x \land Y(c) = y\} \text{move}(c, x', y') \{X(c) = x + x' \land Y(c) = y = y'\} \)

\( \{\text{col}(c) = k\} \text{changeCol}(c); \text{changeCol}(c) \{\text{col}(c) = k\} \)

\( \{\text{true}\} c' \leftarrow \text{copy}(c) \{X(c') = X(c) \land Y(c') = Y(c) \land \text{radius}(c') = \text{radius}(c)\} \)
can be obtained by the first of rules (ii) in Fig. 5, and
\[
moves(c,x,y); moves(c,x',y') = moves(c,x+x',y+y'),
\]
can be obtained by the first of rules (iii) in Fig. 5. Note in the third sentence that in a pre-post sentence \{ϕ₁\} \; dt \{ϕ₂\} the postcondition is a formula over a larger family of variables than the precondition, since it may contain also the variables introduced by \; dt (in this case \(c'\) can appear in the postcondition but not in the precondition). The motivation is to be able to refer to all the intermediate results of the dynamic operations applied in \; dt (the result of \; copy(c) in the example).

Finally,
\[
\{c \leftarrow start\} X(c) = 0 \land Y(c) = 0 \land radius(c) = 1
\]
is a constant dynamic sentence obtained by the second of rules (ii) in Fig. 5.

The validity of these rules is defined in Fig. 6 formalizing the intuition given in Section 2.

As mentioned in the introduction, our aim here is mainly to show some examples of dynamic sentences which may be defined on top of static sentences, without any attempt at fixing an “ideal” specification language for concrete applications. For instance, note that the given definition of satisfaction for pre-post sentences corresponds to what is usually called partial correctness, and a different definition of satisfaction corresponding to total correctness could be easily given. Also note that the given definition of satisfaction for dynamic equalities is very strong: we require that the two dynamic terms have exactly the same variables, and that we get the same state and the same valuation of these variables as result. For practical purposes, some weaker requirement would be more convenient, e.g. requiring to get two states which have the same view as \(Σ\)-models, and the same values only for the common variables.

Another feature frequently allowed in concrete specification languages (e.g. the Eiffel’s assertion language [19]) is the possibility of comparing the state after executing a dynamic term in terms of the state before execution by means of “primed” variables, e.g. writing an equality like \(x = x'\). The problem in allowing this feature is that one needs to be able to interpret this equality symbol somewhere, hence to assume that \(x\) and \(x'\) range over the same set of values. Now, this is not guaranteed in our model, since algebras representing different states are not required to have the same carrier (set of elements). There are two possible solutions: either to define, for each sort \(s\), a universe \(V_s\) of all the possible elements of sort \(s\) in all the possible states, or directly to assume that all the sorts for which one is interested in writing equalities of the form \(x = x'\) have a fixed carrier (this is a very reasonable specialization of the model suitable for concrete applications).

Finally, some compact way of stating that the execution of a dynamic term should not affect some part of the state, unless specified otherwise (what is sometimes called frame assumption), would be very useful. This analysis will be matter of further work.

It is straightforward (see Def/Prop. A.6 in the Appendix A) to define the renaming of dynamic sentences, which is, for any \(dσ: DΣ₁ → DΣ₂\) (resp., \(⟨dσ, h⟩: ⟨DΣ₁, X₁⟩ → \)
\( \langle D\Sigma_2, X_2 \rangle \) morphism of d-oid signatures (resp. d-oid signatures with variables), a map

\[
\overline{D\text{Sen}}(d\sigma) : \overline{D\text{Sen}}(D\Sigma_1) \rightarrow \overline{D\text{Sen}}(D\Sigma_2)
\]

(resp. \( \overline{D\text{Sen}}(\langle d\sigma, h \rangle) : \overline{D\text{Sen}}(\langle D\Sigma_1, X_1 \rangle) \rightarrow \overline{D\text{Sen}}(\langle D\Sigma_2, X_2 \rangle) \))
denoted by \( d\sigma \) (resp. \( \langle d\sigma, h \rangle \)) when there is no ambiguity.

The following theorem states the main technical result of this section, i.e. that the canonical construction described until now actually gives a logical formalism with the property that “truth is invariant under change of syntax”.

**Theorem 21.** The 6-tuple \( \overline{DF} = \langle \overline{DSig}, \overline{St}, \overline{DMod}, |-|, \overline{D\text{Sen}}, \models \rangle \) is a dynamic framework.

**Proof.** We have to show that \( \overline{DF} \) respects the satisfaction condition. That can be proved inductively on the structure of dynamic sentences, using as basis the fact that the satisfaction condition holds for the underlying static framework \( SF \) and that the evaluation of dynamic terms is also invariant w.r.t. change of syntax, as stated in Proposition 18. \( \Box \)

5. From static into dynamic data-types

In this section, we show that the transformation from a static framework \( SF \) into a dynamic framework \( \overline{DF} \) described until now is actually a well-behaved institution transformation, i.e. it is a functor between two appropriate categories of institutions.

To this end, we first define a suitable notion of morphisms over static (resp. dynamic) frameworks. This should be of course a specialization of morphisms between institutions, taking into account the additional structure present in a static (resp. dynamic) framework. In the literature there exists a variety of different definitions of arrows between institutions (see [9] for a survey); here, we consider institution morphisms (see [13]), which correspond to the notion of enriching an institution by new features. This choice is due to the fact that less technical machinery is needed for illustrating this case; anyway, it is possible to show that other arrows between institutions are preserved by our construction, following a pattern analogous to the one shown below.

5.1. Morphisms of static frameworks

If \( I, I' \) are institutions, then an institution morphism from \( I \) into \( I' \) is a triple \( \langle \Phi, \alpha, \beta \rangle \) where \( \Phi \) maps any signature \( \Sigma \) of \( I \) into a signature \( \Sigma' \) of \( I' \), \( \alpha \) maps any \( \Sigma' \)-sentence into a \( \Sigma \)-sentence and \( \beta \) maps any \( \Sigma \)-model into a \( \Sigma' \)-model, in such a way to preserve satisfaction. Note that signatures and models are translated together, while sentences are translated in the opposite direction (see the formal definition in the Appendix, Definition A.7). The intuition is that there is an institution morphism from \( I \) into \( I' \) if \( I \) is an “enrichment” of \( I' \); hence \( \Phi \) and \( \beta \) give the signatures and models of
I' obtained from signatures and models of I, respectively, forgetting the additional structure, while sentences in I' are "embedded" into sentences in I by \( \alpha \). A typical example is the mapping from first order logic into equational logic (signatures are enriched by predicate symbols, models are enriched consequently and equations are recovered as first-order sentences interpreting equality as a particular predicate symbol).

Hence a morphism of static frameworks from SF into SF' will have at least the three components \((\Phi, \alpha, \beta)\) described above. Anyway, in a static framework sentences are over signatures with variables, defined as a canonical extension of signatures, and their satisfaction is defined w.r.t. valuations, defined as a canonical extension of models. Correspondingly, we must be able to map any signature with variables, say \( (\Sigma, X) \), of SF, into a signature with variables \( (\Sigma', X') \) of SF', by means of a canonical extension \( \Phi^{\text{Var}} \) of \( \Phi \). This canonical extension can be defined assuming to have one more component \( \delta \) which maps any sort of \( \Sigma \) into a sort of \( \Phi(\Sigma) \), as will be shown below.

Moreover, we must be able to map any valuation \((A, r)\) over \( (\Sigma, X) \) into a valuation \((\beta(A), r')\) of \( (\Sigma', X') \), by means of a canonical extension \( \beta^{\text{Val}} \) of \( \beta \). This canonical extension can be defined assuming to have one more component \( \gamma \) which maps any element of \( A \) into an element of \( \beta(\Sigma) \), as will be shown below. Hence in summary a morphism of static frameworks will be \((\Phi, \delta, \alpha, \beta, \gamma)\).

We introduce now the first canonical extension, from a functor between two categories of signatures to a functor between the corresponding categories of signatures with variables.

We recall that, for any map \( \delta : S_1 \to S_2 \), there is a functor \( \delta : SSet(S_1) \to SSet(S_2) \) (see Appendix, Def/Proposition A.3) s.t., for any \( S_1 \)-sorted set \( X \), \( \delta(X)_{s_1} \) is the disjoint union of all \( X_{s_1} \) s.t. \( \delta(s_1) = s_2 \); \( i^\delta_X \) is the injection from \( X \) into \( \delta(X)_{s_2} \), i.e. \( \forall s_1 \in S_1 \) with \( \delta(s_1) = s_2 \), \( (i^\delta_X)_{s_1} \) is the injection from \( X_{s_1} \) into \( \delta(X)_{s_2} \).

The following proposition states that, given a functor \( \Phi \) between two categories of signatures, it is possible to extend \( \Phi \) to signatures with variables if we have a natural transformation \( \delta \) telling us, for any signature \( \Sigma \), how the sorts of \( \Sigma \) are mapped into the sorts of \( \Phi(\Sigma) \). Indeed in this case a signature with variables, say \( (\Sigma, X) \), is mapped in \( (\Phi(\Sigma), \delta_\Sigma(X)) \) (intuitively, variables of a sort, say \( s \), become variables of sort \( \delta_\Sigma(s) \)).

**Def/Prop. 22.** Let \( \Phi : \text{Sig} \to \text{Sig}' \) be a functor between two categories of signatures with sort functors \( \text{Sorts, Sorts}' \) respectively, \( \delta : \text{Sorts} \to \text{Sorts}' \circ \Phi \) be a natural transformation.

Then \( \Phi^{\text{Var}} : \text{Sig}^{\text{Var}} \to (\text{Sig}')^{\text{Var}} \) is the functor defined as follows:

- \( \forall (\Sigma, X) \in |\text{Sig}^{\text{Var}}|, \Phi^{\text{Var}}((\Sigma, X)) = (\Phi(\Sigma), \delta_\Sigma(X)); \)
- \( \forall (\sigma, h) : (\Sigma_1, X_1) \to (\Sigma_2, X_2) \) morphism in \( \text{Sig}^{\text{Var}} \),
  \( \Phi^{\text{Var}}((\sigma, h)) = (\Phi(\sigma), \eta_\Sigma^\sigma \circ \delta_\Sigma_1(h)) : (\Phi(\Sigma_1), \delta_\Sigma_1(X_1)) \to (\Phi(\Sigma_2), \delta_\Sigma_2(X_2)) \)
where \( \eta_\Sigma^\sigma : \delta_\Sigma_1 \circ -_{|\sigma} \to -_{|\Phi(\sigma) \circ \delta_\Sigma} \) is the natural transformation defined by:

\( \forall X \in |SSet(\text{Sorts}(\Sigma_2))|, \)

\( \eta_X(i^\delta_{\Sigma_2}^\sigma(x)) = i^\delta_{\Sigma_1}(x), \forall x \in X_{|\sigma}. \)
Proof.
- $\forall (\Sigma, X) \in |\text{Sig}^{\text{Var}}|$, $(\Phi(\Sigma), \delta_\Sigma(X)) \in |(\text{Sig}^{\prime})^{\text{Var}}|$; indeed, since $\delta_\Sigma : \rightarrow \text{Sorts}(\Sigma) \rightarrow \text{Sorts}'(\Phi(\Sigma))$, we have that $\delta_\Sigma(X) \in |\text{SSet}(\text{Sorts}'(\Phi(\Sigma)))|$.
- $\forall (\sigma, h) : (\Sigma_1, X_1) \rightarrow (\Sigma_2, X_2)$ morphism in $\text{Sig}^{\text{Var}}$, $(\Phi(\sigma), \eta^\sigma_{X_1} \circ \delta_\Sigma(h))$ is a morphism in $(\text{Sig}^{\prime})^{\text{Var}}$; indeed, first of all, note that the diagram below commutes since $\delta$ is a natural transformation, hence $\eta^\sigma$ is well-defined by Proposition A.4 in the Appendix A.

$$\begin{array}{ccc}
\text{Sorts}(\Sigma_1) & \xrightarrow{\delta_\Sigma} & \text{Sorts}'(\Phi(\Sigma_1)) \\
\downarrow \sigma & & \downarrow \Phi(\sigma) \\
\text{Sorts}(\Sigma_2) & \xrightarrow{\delta_\Sigma} & \text{Sorts}'(\Phi(\Sigma_2)) \\
\end{array}$$

Now, since $h : X_1 \rightarrow (X_2)_\sigma$, $\delta_\Sigma(h) : \delta_\Sigma(X_1) \rightarrow \delta_\Sigma((X_2)_\sigma)$; moreover,

$$\eta^\sigma_{X_2} : \delta_\Sigma((X_2)_\sigma) \rightarrow (\delta_\Sigma(X_2))_{\Phi(\sigma)}$$

by definition.
- $\Phi^{\text{Var}}$ is actually a functor. Indeed
  - $\Phi^{\text{Var}}(id_{(\Sigma, X)}) = \Phi^{\text{Var}}((id_\Sigma, id_X)) = (\Phi(id_\Sigma), \eta^\Sigma_X \circ \delta(id_X))$.
    - But $\delta(id_X) = id(\delta_X(X))$ and $\eta^\Sigma_X$ is again the identity of $\delta_\Sigma(X)$, indeed by definition $\eta^\Sigma_X(i^\Sigma_X(x)) = i^\Sigma_X(x), \forall x \in X|_{id_X} = X$.
  - Consider $(\sigma_1, h_1) : (\Sigma_1, X_1) \rightarrow (\Sigma_2, X_2)$, $(\sigma_2, h_2) : (\Sigma_2, X_2) \rightarrow (\Sigma_3, X_3)$. We have the following diagram.

$$\begin{array}{ccc}
\text{Sorts}(\Sigma_1) & \xrightarrow{\delta_\Sigma} & \text{Sorts}'(\Phi(\Sigma_1)) \\
\downarrow \sigma_1 & & \downarrow \Phi(\sigma_1) \\
\text{Sorts}(\Sigma_2) & \xrightarrow{\delta_\Sigma} & \text{Sorts}'(\Phi(\Sigma_2)) \\
\downarrow \sigma_2 & & \downarrow \Phi(\sigma_2) \\
\text{Sorts}(\Sigma_3) & \xrightarrow{\delta_\Sigma} & \text{Sorts}'(\Phi(\Sigma_3)) \\
\end{array}$$

Then

$$\Phi^{\text{Var}}((\sigma_2, h_2) \circ (\sigma_1, h_1)) = \Phi^{\text{Var}}((\sigma_2 \circ \sigma_1, (h_2)_{\sigma_1} \circ h_1))$$

$$= (\Phi(\sigma_2 \circ \sigma_1), \eta^\Sigma_{X_3} \circ \delta_{\Sigma_1}(((h_2)_{\sigma_1} \circ h_1)))$$

$$= (\Phi(\sigma_2 \circ \sigma_1), \eta^\Sigma_{X_3} \circ \delta_{\Sigma_1}(((h_2)_{\sigma_1} \circ h_1)).$$
On the other side,
\[ \Phi^{\vartriangleleft}(\langle \sigma_2, h_2 \rangle) \circ \Phi^{\vartriangleleft}(\langle \sigma_1, h_1 \rangle) = \langle \Phi(\sigma_2), \eta_{X_3}^{\vartriangleleft} \circ \delta_{X_2}(h_2) \circ \Phi(\sigma_1), \eta_{X_2}^{\vartriangleleft} \circ \delta_{X_1}(h_1) \rangle \]
\[ = \langle \Phi(\sigma_2) \circ \Phi(\sigma_1), (\eta_{X_3}^{\vartriangleleft} \circ \delta_{X_2}(h_2))_{\Phi(\sigma_1)} \circ \eta_{X_2}^{\vartriangleleft} \circ \delta_{X_1}(h_1) \rangle \]
\[ = \langle \Phi(\sigma_2 \circ \sigma_1), (\eta_{X_3}^{\vartriangleleft})_{\Phi(\sigma_1)} \circ \delta_{X_2}(h_2) \rangle_{\Phi(\sigma_1)} \circ \eta_{X_2}^{\vartriangleleft} \circ \delta_{X_1}(h_1) \rangle. \]

By Proposition A.4 in the Appendix A, \( \eta_{X_3}^{\vartriangleleft} = \eta_{X_3}^{\vartriangleleft}_{\Phi(\sigma_1)} \circ \eta_{X_2}^{\vartriangleleft} \). Moreover, \( \eta_{X_2}^{\vartriangleleft} \circ \delta_{X}(h_2) = \eta_{X_1}^{\vartriangleleft} \) since \( \eta^{\vartriangleleft} \) is a natural transformation and \( h_2 : X_2 \rightarrow X_3_{\sigma_2} \).

We consider now the second canonical extension mentioned in the beginning of this section, i.e. from a natural transformation \( \beta: \text{Mod} \rightarrow \text{Mod}' \circ \Phi \) between two model functors into a natural transformation \( \beta^{Val}: \text{Val} \rightarrow \text{Val}' \circ \Phi^{\vartriangleleft} \) between the corresponding valuation functors. This extension can be defined assuming to have, for any signature \( \Sigma \), for any \( \Sigma \)-model, say \( A \), a mapping \( \gamma^{\Sigma}_A \) of the elements of \( A \) into the elements of \( A' = \beta_\Sigma(A) \). In this way, any valuation \( \langle A, r \rangle \) over \( \langle \Sigma, X \rangle \) can be transformed into a valuation \( \langle A', r' \rangle \), with \( r': \delta_\Sigma(X) \rightarrow |A'| \) defined as follows: if \( r_\Sigma(x) = a \), then \( r'_{\beta_\Sigma}(x) = \gamma^{\Sigma}_A(a) \). Formally, \( \gamma^\Sigma \) is a natural transformation, for any \( \Sigma \in |\Sigma| \), and moreover all the \( \gamma^\Sigma \) form a family which is well-behaved w.r.t. signature morphisms (see condition (2) in the definition below).

**Def/Prop. 23.** Let \( \langle \Sigma, \text{Sorts}, \text{Mod}, \cdot| \cdot \rangle \) and \( \langle \Sigma', \text{Sorts}', \text{Mod}', \cdot| \cdot \rangle \) be two static model parts, \( \Phi: \Sigma \rightarrow \Sigma' \) a functor and \( \beta: \text{Sorts} \rightarrow \text{Sorts}' \circ \Phi \) a natural transformation. Let moreover \( \beta: \text{Mod} \rightarrow \text{Mod}' \circ \Phi^{\vartriangleleft} \) be a natural transformation, and \( \gamma \) be an \( |\Sigma| \)-family of natural transformations s.t.

(i) \( \forall \Sigma \in |\Sigma|, \gamma^{\Sigma}: \delta_\Sigma \circ |\Sigma| \rightarrow |\beta_\Sigma| \circ \Phi_\Sigma \) is well-defined.
(ii) \( \forall \sigma: \Sigma_1 \rightarrow \Sigma_2 \text{ morphism in } \Sigma, \gamma^{\Sigma_2}_1 \circ |\sigma| = (\langle \Phi(\sigma) \circ \gamma^{\Sigma_2} \rangle \circ (\eta^{\vartriangleleft} \circ |\Sigma|). \)

Then, \( \beta^{Val}: \text{Val} \rightarrow \text{Val}' \circ \Phi^{\vartriangleleft} \) is the natural transformation defined as follows:
\[ \forall \langle \Sigma, X \rangle \in |\Sigma| \circ \Phi, \beta^{Val}(\langle \Sigma, X \rangle) \in |\Sigma| \circ \Phi \text{ is the functor defined by:} \]
\[ \forall \langle A, r \rangle \in |\text{Val}(\langle \Sigma, X \rangle)|, \]
\[ \beta^{Val}(\langle \Sigma, X \rangle)(\langle A, r \rangle) = \langle \beta_\Sigma(A), \gamma^{\Sigma}_A \circ \delta_{X}(r) \rangle; \]
\[ \forall f: \langle A, r_A \rangle \rightarrow \langle B, r_B \rangle \text{ morphism in } \text{Val}(\langle \Sigma, X \rangle), \]
\[ \beta^{Val}(\langle \Sigma, X \rangle)(f) = \beta_\Sigma(f). \]

**Proof.**
- First, we show that, \( \forall \langle \Sigma, X \rangle, \beta^{Val}(\langle \Sigma, X \rangle) \) is well-defined.
  - \( \forall \langle A, r \rangle \in |\text{Val}(\langle \Sigma, X \rangle)|, \) since \( r: X \rightarrow |A|_\Sigma \), we have that \( \delta_{X}(r): \delta_{X}(X) \rightarrow \delta_{X}(|A|_\Sigma) \); moreover \( \gamma^{\Sigma}_A: \delta_{X}(|A|_\Sigma) \rightarrow |\beta_\Sigma(A)|_{\Phi_\Sigma} \), hence \( \gamma^{\Sigma}_A \circ \delta_{X}(r) \) is actually a map from \( \delta_{X}(X) \) into \( |\beta_\Sigma(A)| \).
\[ \forall f : \langle A, r_A \rangle \rightarrow \langle B, r_B \rangle \text{ morphism in } Val(\langle \Sigma, X \rangle), \text{ we have to show that } \beta_{\langle \Sigma, X \rangle}^V(f) = \beta_\Sigma(f) \text{ is a morphism in } Val'(\langle \Phi(\Sigma), \delta_\Sigma(X) \rangle), \text{ i.e. the following diagram commutes.} \]

\[ \begin{array}{c}
\delta_\Sigma(\langle A \rangle) \\
\downarrow \delta_\Sigma(r_A) \\
\delta_\Sigma(X) \\
\downarrow \delta_\Sigma(f) \\
\delta_\Sigma(\langle B \rangle) \\
\end{array} \xrightarrow{\eta_\Sigma} \begin{array}{c}
|\beta_\Sigma(A)|' \\
\downarrow |\beta_\Sigma(f)|' \\
|\beta_\Sigma(B)|' \\
\end{array} \]

Indeed, the right part of the diagram commutes since \( \eta_\Sigma \) is a natural transformation, and the left part commutes since \( r_B = |f| \circ r_A \) from the fact that \( f \) is a morphism in \( Val(\langle \Sigma, X \rangle) \), and \( \delta_\Sigma \) is a functor.

- Second, we have to show that \( \forall \langle \sigma, X \rangle, \beta_{\langle \sigma, X \rangle}^V \) is a functor. That is a trivial consequence of the definition.

- Finally, we have to show that \( \beta_{\langle \Sigma, X \rangle}^V \) is a natural transformation, i.e., \( \forall \langle \sigma, h \rangle : \langle \Sigma_1, X_1 \rangle \rightarrow \langle \Sigma_2, X_2 \rangle \), the following diagram commutes.

\[ \begin{array}{c}
Val(\langle \Sigma_1, X_1 \rangle) \\
\downarrow |\langle \sigma, h \rangle| \\
Val(\langle \Phi(\Sigma_1), \delta_{\Sigma_1}(X_1) \rangle) \\
\downarrow |\langle \Phi(\sigma), \delta_{\Sigma_1}(h) \rangle| \\
\end{array} \xrightarrow{\beta_{\langle \Sigma_1, X_1 \rangle}^V} \begin{array}{c}
Val(\langle \Phi(\Sigma_2), \delta_{\Sigma_2}(X_2) \rangle) \\
\downarrow |\langle \sigma, h \rangle| \\
Val(\langle \Sigma_2, X_2 \rangle) \\
\downarrow |\langle \sigma, h \rangle| \\
\end{array} \]

- Consider \( \langle A, r \rangle \in |Val(\langle \Sigma_2, X_2 \rangle)| \). We have:

\[ \beta_{\langle \Sigma_2, X_2 \rangle}^V(\langle A, r \rangle)|_{\langle \sigma, h \rangle} = \langle \beta_{\Sigma_2}(A)|_{\Phi(\Sigma)}, \gamma_{\Sigma_2}^\Sigma \circ \delta_{\Sigma_2}(r) \rangle|_{\Phi(\sigma)} \circ \eta_{X_2}^\Sigma \circ \delta_{\Sigma_2}(h) \]

\[ \beta_{\langle \Sigma_1, X_1 \rangle}^V(\langle A, r \rangle)|_{\langle \sigma, h \rangle} = \langle \beta_{\Sigma_1}(A)|_{\Phi(\sigma)}, \gamma_{\Sigma_1}^\Sigma \circ \delta_{\Sigma_1}(r) \rangle|_{\Phi(\sigma)} \circ \eta_{X_2}^\Sigma \circ \delta_{\Sigma_2}(h). \]

The first components are equal since \( \beta \) is a natural transformation. For the second components, we have to prove:

\[ (\gamma_{\Sigma_2}^\Sigma)|_{\Phi(\sigma)} \circ (\delta_{\Sigma_2}(r))|_{\Phi(\sigma)} \circ \eta_{X_2}^\Sigma = \gamma_{A|_{\sigma}}^\Sigma \circ \delta_{\Sigma_2}(r|_{\sigma}). \]

But \( (\delta_{\Sigma_2}(r))|_{\Phi(\sigma)} \circ \eta_{X_2}^\Sigma = \eta_{X_2}^\sigma \circ \delta_{\Sigma_2}(r|_{\sigma}) \) since \( \eta^\sigma \) is a natural transformation and \( r : X_2 \rightarrow \vert A \vert \). Hence the thesis follows by the hypothesis (2) on \( \gamma \).

- The commutativity holds for morphisms since \( \beta \) is a natural transformation. \( \Box \)
We can now give the formal definition of morphism of static frameworks.

**Definition 24.** Given two static frameworks

\[ SF = (\text{Sig}, \text{Sorts}, \text{Mod}, |-|, \text{Sen}, \vdash) \quad SF' = (\text{Sig}', \text{Sorts}', \text{Mod}', |-|', \text{Sen}', \vdash') \]

a morphism of static frameworks \( sm = (\Phi, \delta, \alpha, \gamma) : SF \to SF' \) consists of

- a functor \( \Phi : \text{Sig} \to \text{Sig}' \);
- a natural transformation \( \delta : \text{Sorts} \to \text{Sorts}' \circ \Phi ; \)
- a natural transformation \( \alpha : \text{Sen}' \circ \Phi_{\text{Var}} \to \text{Sen} ; \)
- a natural transformation \( \beta : \text{Mod} \to \text{Mod}' \circ \Phi \);
- an \( |\text{Sig}| \)-family of natural transformations \( \gamma \) s.t.

(i) \( \forall \Sigma \in |\text{Sig}|, \gamma_{\Sigma} : |\Sigma| \to |-|_{\Phi(\Sigma)} \circ |\Sigma| \)

(ii) \( \forall \sigma : \Sigma_1 \to \Sigma_2 \) morphism in \( \text{Sig} , \)

\[ \gamma_{\Sigma_1} \circ |\sigma| = (|-|_{\Phi(\sigma)} \circ \gamma_{\Sigma_2}) \circ (|\sigma| \circ |\Sigma_1|) . \]

s.t. \( \forall (\Sigma, X) \in |\text{Sig}_{\text{Var}}| , \) the condition

\[ \beta^\text{val}_{(\Sigma, X)}((A, r)) \vdash \phi' \iff (A, r) \vdash (\text{Val}((\Sigma, X))) \phi' \]

holds for any \( (A, r) \in |\text{Val}((\Sigma, X))| \) and \( \phi' \in \text{Sen}'(\Phi_{\text{Var}}((\Sigma, X))) \).

**Def/Prop. 25.** Static frameworks with morphisms of static frameworks and composition and identity defined componentwise form a category, which we denote by \( \text{St Fram} \).

### 5.2. Morphisms of dynamic frameworks

We give now a formal definition of morphism of dynamic frameworks. Recall that a dynamic framework \( DF \) is defined as based on a certain static framework \( SF \) (in other words, the components of the static framework are implicit components of the dynamic framework too). Hence, a morphism of dynamic frameworks \( dm : DF \to DF' \) will be based on a morphism between the corresponding static frameworks \( sm : SF \to SF' \), which will deal with mapping of static components. For what concerns the dynamic components, the pattern is the same already shown for static frameworks: the three standard components \( D\Phi, D\alpha, D\beta \) required for (generic) institution morphisms and related to (dynamic) signatures, models and sentences, respectively, will be needed. Then we have to consider the additional structure present in a dynamic framework w.r.t. a generic institution. Two canonical extensions need to be considered, analogously to the static case.

- Since in a dynamic framework (non-constant) sentences are over dynamic signatures with variables, we need to canonically extend the functor \( D\Phi \) to a functor between the two corresponding categories of dynamic signatures with variables. Anyway, since any category of dynamic signatures, say \( \text{DSig} \), has an associated sort functor, given by the composition \( \text{Sorts} \circ \text{St} \) where \( \text{St} : \text{DSig} \to \text{Sig} \) is the static part functor for \( \text{DSig} \) and \( \text{Sorts} \) is the sort functor for \( \text{Sig} \), the extension defined by
Def/Prop. 22 above applies as well to dynamic signatures, taking a natural transformation $D\delta: \text{Sorts} \circ St \rightarrow \text{Sorts}' \circ St' \circ D\Phi$ defined by:

$$\forall D\Sigma \in |DSig|, D\Sigma \text{ with static part } \Sigma, D\delta_{D\Sigma} = \delta_{\Sigma}.$$ 

The only obvious compatibility requirement is that $St \circ \Phi = St' \circ D\Phi$.

Moreover, we must be able to map any dynamic valuation $(\mathcal{A}, A, r)$ over $(D\Sigma, X)$ into a dynamic valuation $(D\beta(\mathcal{A}), A', r')$ of $(D\Sigma', X')$, by means of a canonical extension $D\beta^{Val}$ of $D\beta$. This canonical extension can be defined assuming to have one more component $D_D^D\Sigma$ s.t., for any $\mathcal{A}$ dynamic model over $D\Sigma$, $D\beta_{D\Sigma}^{D\Sigma}(\mathcal{A})$ is a morphism of state sets over $\Sigma' = \Phi(\Sigma)$. The source of $D\beta_{D\Sigma}^{D\Sigma}$ is the state set $\tilde{P}_\Sigma(\mathcal{A})$ obtained from the carrier of $\mathcal{A}$ keeping the same states and transforming their views as $\Sigma$-models into views as $\Sigma'$-models, by the $\beta_{\Sigma}$ component of $\text{sm}$ ($\tilde{P}$ is formally defined in Def/Prop. 26 below). Recall that a morphism of state sets over $\Sigma'$ maps any state in the source state set into a pair consisting of a state in the target state set and a $\Sigma'$-morphisms between the views of the two states. Hence, we can define $A'$ to be the state s.t. $D\beta_{D\Sigma}^{D\Sigma}(A) = (A', f)$, and $r' = |f| \circ \tilde{r}$ where $\tilde{r}$ is the mapping of $X'$ into the view of $A'$ obtained transforming $r$ by the extension $\beta^{Val}$ to valuations of $\beta$. The formal definition of $D\beta^{Val}$ is given in Def/Prop. 27 below.

Hence in summary a morphism of dynamic frameworks will be $(D\Phi, D\delta, D\beta, D\gamma)$.

Def/Prop. 26. Let $(\text{Sig}, \text{Sorts}, \text{Mod}, \to)$ and $(\text{Sig}', \text{Sorts}', \text{Mod}', \to')$ be two static model parts, $\Phi: \text{Sig} \rightarrow \text{Sig}'$ a functor and $\delta: \text{Sorts} \rightarrow \text{Sorts}' \circ \Phi$ a natural transformation. Let moreover $\beta: \text{Mod} \rightarrow \text{Mod}' \circ \Phi$ be a natural transformation. Then, $\tilde{\beta}_{\Sigma}: \text{Mod}(\Sigma) \rightarrow \text{Mod}'(\Phi(\Sigma))$ is the functor defined by:

$$\forall \mathcal{A} \in |\text{Mod}(\Sigma)|,$$

$$\tilde{\beta}_{\Sigma}(\mathcal{A}) = (\text{Dom}(\mathcal{A}), |\beta| \circ \text{View}_{\mathcal{A}})$$

$$\forall \phi: \mathcal{A} \rightarrow \mathcal{B} \text{ morphism in } \text{Mod}(\Sigma),$$

$$\forall A \in \tilde{\beta}_{\Sigma}(\mathcal{A}), \tilde{\beta}_{\Sigma}(\phi)(A) = (B, \beta_{\Sigma}(f)) \text{ if } \phi(A) = (B, f).$$

Def/Prop. 27. Let $(DSig, St, DMod, \to)$ and $(DSig', St', DMod', \to')$ be two dynamic model parts, $D\Phi: DSig \rightarrow DSig'$ a functor and $D\delta: \text{Sorts} \circ St \rightarrow \text{Sorts}' \circ St' \circ D\Phi$ a natural transformation. Let moreover $D\beta: DMod \rightarrow DMod' \circ D\Phi$ be a natural transformation and $\gamma$ be an $|DSig|$-family of natural transformations s.t.

$$\forall D\Sigma \in |DSig|, D\beta_{D\Sigma}^{D\Sigma}: \tilde{\gamma}_{\Sigma} \circ \to_{D\Sigma} \rightarrow \to'_{D\Phi(\Sigma)} \circ D\beta_{D\Sigma}.$$ 

$$\forall d\sigma: D\Sigma_1 \rightarrow D\Sigma_2 \text{ morphism in } DSig, d\sigma \text{ with static part } \sigma,$$

$$D\beta_{D\Sigma_1} \circ -|_{d\sigma} = -|_{d\sigma} \circ D\beta_{D\Sigma_2}.$$
Then,
\[ D\beta^{Val}_{(D\Sigma,X)} : DVal \to DVal' \circ D\Phi^{Var} \]
is the natural transformation defined as follows:
\[ \forall (D\Sigma,X) \in |D\Sigma^{Var}|, D\Sigma \text{ with static part } \Sigma, \]
\[ D\beta^{Val}_{(D\Sigma,X)} : \text{Val}((D\Sigma,X)) \to \text{Val}'((D\Sigma,X)) \text{ is the functor defined by} \]
\[ \forall (\mathcal{A}, A, r) \in |\text{Val}((D\Sigma,X))|, \]
\[ D\beta^{Val}_{(D\Sigma,X)}((\mathcal{A}, A, r)) = \langle D\beta_{D\Sigma}(\mathcal{A}), A', |f| \circ \overline{f} \rangle \]
\[ \text{if } D\gamma^{D\Sigma}_{D\Sigma}(A) = \langle A', f \rangle \text{ and } \beta^{Val}_{(D\Sigma,X)}((\text{View}_{D\Sigma}(A), r)) = \langle \beta_{\Sigma}(\text{View}_{D\Sigma}(A)), \overline{r} \rangle; \]
\[ \forall \phi : (\mathcal{A}, A, r_A) \to (\mathcal{B}, B, r_B) \text{ morphism in } D\text{Val}((D\Sigma,X)), \]
\[ D\beta^{Val}_{(D\Sigma,X)}(\phi) = D\beta_{D\Sigma}(\phi). \]

**Def/Prop. 28.** Given two dynamic frameworks

\[ DF = \langle \text{Dsig, St, D Mod, }|-, D\text{Sen}, \vdash \rangle, \ D\Sigma' = \langle \text{Dsig}', \text{St}', \text{D Mod}', |-, D\text{Sen}', \vdash \rangle, \]

based on the static frameworks SF, SF', respectively, a morphism of dynamic frameworks

\[ dm = \langle \Phi, D\beta, D\alpha, D\gamma \rangle : DF \to D\Sigma', \]

based on the morphism of static frameworks sm : SF \to SF', sm = \langle \Phi, \delta, \beta, \alpha, \gamma \rangle consists of

- a functor \[ D\Phi : \text{Dsig} \to \text{Dsig}', \text{s.t. } \text{St}' \circ D\Phi = \Phi \circ \text{St}; \]
- a natural transformation \[ D\alpha : \text{DSen}' \circ (D\Phi + D\Phi^{Var}) \to \text{DSen}; \]
- a natural transformation \[ D\beta : \text{D Mod} \to \text{D Mod}' \circ D\Phi; \]
- an \[ |\text{Sig}|\text{-indexed family of natural transformations } D\gamma \text{ s.t.} \]
  \[ \forall D\Sigma \in |\text{Dsig}|, D\gamma^{D\Sigma} : \beta_{\Sigma} \circ |-|_{D\Sigma} \to |-|_{D\Phi(D\Sigma)} \circ D\beta_{D\Sigma} \]
  \[ \forall \forall \sigma : D\Sigma_1 \to D\Sigma_2 \text{ morphism in } D\text{Sig}, d\sigma \text{ with static part } \sigma, \]
  \[ D\gamma^{D\Sigma_1} \circ |d\sigma| = |d\sigma| \circ D\gamma^{D\Sigma_2} . \]

s.t. the following two conditions hold:

(i) \[ \forall D\Sigma \in |\text{Dsig}|, \]
\[ D\beta_{D\Sigma}(\mathcal{A}) \vdash_{D\Phi(D\Sigma)}^D d\phi' \iff \mathcal{A} \vdash_{D\Sigma} D\alpha_{D\Sigma}(d\phi') \]

for any \[ \mathcal{A} \in |D\text{Mod}(D\Sigma)| \text{ and } d\phi' \in \text{DSen}'(D\Phi(D\Sigma)); \]

(ii) \[ \forall (D\Sigma, X) \in |D\Sigma^{Var}|, \]
\[ D\beta_{(D\Sigma,X)}^{Val} \vdash_{D\Phi^{Var}(D\Sigma,X)} d\phi' \iff \rho \vdash_{(D\Sigma,X)} D\alpha_{D\Sigma,X}(d\phi') \]

for any \[ \rho \in |D\text{Val}((D\Sigma,X))| \text{ and } d\phi' \in \text{DSen}'(D\Phi^{Var}((D\Sigma,X))). \]
Def/Prop. 29. Dynamic frameworks with morphisms of dynamic frameworks and composition and identity defined componentwise form a category, which we denote by DynFram.

5.3. A functor from static to dynamic frameworks

We can now show that the construction presented in Section 4 is actually a functor from the category of static frameworks into the category of dynamic frameworks. To this end, we have to extend the construction to morphisms.

First of all, recall that d-oid signatures (as defined by Definition 11) are pairs \( \langle \Sigma, DOP \rangle \) where \( \Sigma \) is a static signature, say over \( S \), and \( DOP \) is an \( [S] \cup (S^\ast \times [S]) \)-sorted set of symbols. It is straightforward to extend some definitions and results we have shown for \( S \)-sorted sets to \( [S] \cup (S^\ast \times [S]) \)-sorted sets, as outlined below.

- Let \( Dop \) be the functor giving, for any set of sorts \( S \), the category of the \( [S] \cup (S^\ast \times [S]) \)-sorted sets, defined analogously to Definition A.2.
- For any map \( \delta : S_1 \rightarrow S_2 \), we can define a functor \( \delta : Dop(S_1) \rightarrow Dop(S_2) \) analogously to Definition A.3.
- Hence, given a functor \( \Phi : Sig \rightarrow Sig' \) between two categories of signatures with sort functors \( Sorts, Sorts' \) respectively, \( \delta : Sorts \rightarrow Sorts' \circ \Phi \), a natural transformation, we can define the extension of \( \Phi \) to d-oid signatures,

\[
\phi^{Dop} : DSig \rightarrow DSig'
\]

analogously to Definition 22.

Intuitively, this extension maps any d-oid signature, say \( \langle \Sigma, DOP \rangle \), into a d-oid signature \( \langle \Phi(\Sigma), DOP' \rangle \) where the static symbols are translated by \( \Phi \) and the dynamic operation symbols are the same but have changed their functionality accordingly with the sort renaming \( \delta_\Sigma \).

Def/Prop. 30. For any static framework \( SF \), let \( DF(SF) \) denote the dynamic framework based on \( SF \) defined in Section 4 (Definitions 11, 13, and 14). Moreover, for any \( sm : SF \rightarrow SF' \) morphism of static frameworks, with \( SF = \langle Sig, Sorts, Mod, |\cdot|, Sen, \# \rangle \), \( SF' = \langle Sig', Sorts', Mod', |\cdot'|, Sen', \#' \rangle \), let \( DF(sm) \) denote the morphism of dynamic frameworks, \( DF(sm) : DF(SF) \rightarrow DF(SF') \), defined by \( DF(sm) = \langle D\Phi, D\beta, D\alpha, D\gamma \rangle \) where

- \( D\Phi = \phi^{Dop} \),

\[
\forall D\Sigma \in DSig, D\Sigma \text{ with static part } \Sigma, D\beta_{D\Sigma} : DMod(D\Sigma) \rightarrow DMod'(D\Phi(D\Sigma_1)) \text{ is defined enriching the functor } \beta_{\Sigma} : Mod(\Sigma) \rightarrow Mod(\Phi(\Sigma)) \text{ between the underlying categories of state sets by the interpretation of the dynamic operations:}
\]

\[
\forall \alpha \in [DMod(D\Sigma)], \\
\forall dop : s \in D\Sigma, \\
dop^{D\beta_{D\Sigma}(\alpha)} = dop'_{\alpha};
\]
Fig. 7. Translating dynamic sentences.

\[ dt_1 = dt_2 \rightarrow dt_1 = dt_2 \]

\( \forall \text{dop}: w \Rightarrow [s] \text{ in } D\Sigma, \langle A, \bar{a} \rangle \in D\beta_{D\Sigma}(\mathcal{A}) \),
\( \text{dop}^{D\beta_{D\Sigma}(\mathcal{A})}(\langle A, \bar{a} \rangle) - \langle \delta_{D\Sigma}(f): A \rightarrow B[1, b] \rangle \text{ iff } \text{dop}^{\mathcal{A}}(\langle A, \bar{a} \rangle) - \langle f: A \rightarrow B[1, b] \rangle \).

\( \forall (D\Sigma, X) \in |D\mathbf{Sig}|_{\text{Var}}, D\Sigma \text{ with static part } \Sigma \), the maps
\( D\alpha_{D\Sigma} : \overline{D\text{Sen}}(D\Phi(D\Sigma)) \rightarrow (\overline{D\text{Sen}}(D\Sigma)), \) and
\( D\alpha_{(D\Sigma, X)} : \overline{D\text{Sen}}(D\Phi^{\text{Var}}((D\Sigma, X)) \rightarrow (\overline{D\text{Sen}}((D\Sigma, X))) \)
are defined in Fig. 7, where where we write
\( d\phi' \rightarrow d\phi \) for \( D\alpha_{D\Sigma}(d\phi') = d\phi \) (resp. \( D\alpha_{(D\Sigma, X)}(d\phi') = d\phi \)).

\( \forall D\Sigma \in |D\mathbf{Sig}|, \forall \mathcal{A} \in |D\mathbf{Mod}(D\Sigma)|, D\gamma_{\mathcal{A}, D\Sigma}^{D\Sigma} = \text{id}_{|\mathcal{A}|_{D\Sigma}}. \)

Intuitively, a morphism of static frameworks \( \text{sm} \) can be canonically extended to a morphism between the corresponding d-oid frameworks as follows.

- A d-oid signature \( \langle \Sigma, D\text{OP} \rangle \) is transformed into a d-oid signature \( \langle \Sigma', D\text{OP}' \rangle \) where only static symbols are changed as specified by \( \text{sm} \) (\( \Sigma' = \Phi(\Sigma) \)), while dynamic operation symbols in \( D\text{OP}' \) are the same as in \( D\text{OP} \), but their functionality changes accordingly with the sort renaming \( \delta_{D\Sigma} \) in \( \text{sm} \).

- A d-oid \( \mathcal{A} \) over \( \langle \Sigma, D\text{OP} \rangle \) is transformed into a d-oid \( \mathcal{A} \) over \( \langle \Sigma', D\text{OP}' \rangle \) which has the same states of \( \mathcal{A} \), viewed now as \( \Sigma' \)-models via the \( \beta_{D\Sigma} \)-component of \( \text{sm} \). Since the states remain the same, the interpretation of dynamic operation symbols can remain basically the same; only tracking maps must be changed, since they were before \( \text{Sorts}(\Sigma) \)-sorted maps, and they must become \( \text{Sorts}(\Sigma') \)-sorted maps; that can be achieved by the functor \( \delta_{D\Sigma} : \text{SSet}(\text{Sorts}(\Sigma)) \rightarrow \text{SSet}(\text{Sorts}(\Sigma')) \) associated with the sort renaming \( \delta_{D\Sigma} \).

- A dynamic sentence over \( \langle \Sigma', D\text{OP}' \rangle \) is transformed into a dynamic sentence over \( \langle \Sigma, D\text{OP} \rangle \) by translating its components which are static sentences as specified by the \( \alpha \) components of \( \text{sm} \), while dynamic operation symbols remain unchanged.

In summary, we can say that \( \text{sm} \) is extended to a morphism \( \text{dm} \) of dynamic frameworks which behaves, roughly speaking, as \( \text{sm} \) on the static aspects and "is the identity" on the dynamic aspects.
We can state now our main technical result of this section.

**Theorem 31.** The mapping $\overline{DF}$ defined in Def/Prop. 30 is a functor from $\text{StFram}$ into $\text{DynFram}$.

6. Conclusion

We have presented a canonical construction which associates with any static framework (an institution of static data-types) a corresponding dynamic framework (an institution of dynamic data-types). This construction is formally a functor between the appropriate categories.

The relevance of this work is twofold. First, we have shown that dynamic data-types, as already introduced in [4, 5] only as models over a fixed signature, actually define an institution, and moreover an institution which is parameterized on the underlying institution chosen for modelling static aspects.

More in general, our work is concerned with the important topic of integrating different formalisms, since we show here how to enrich an existing formalism for expressing static aspects with additional ingredients which allow to handle dynamics.

In this paper, the ingredients we choose are: d-oids as dynamic models (dynamics is modelled by dynamic operations) and pre–post sentences and dynamic equations as sentences. Of course different solutions can be adopted, like modelling dynamics by transitions between states (instead of using operations), and choosing sentences in some kind of modal or temporal logic. An interesting question for further work is whether it is still possible with these different choices to get a canonical construction. For instance, it seems quite straightforward to introduce some temporal operator in our sentences. A final result of this investigation could be a general notion of "sum" between a formalism for static aspects and a formalism for dynamic aspects.

Moreover, the definition of static frameworks (with corresponding morphisms) we have introduced in this paper corresponds (as a "side-benefit") to a notion of *institutions with variables* which is of independent interest. A similar notion has been introduced in [20] under the name of *context institution*. In a context institution, for any signature $Σ$ it is defined a category of contexts over $Σ$. The corresponding notion in our framework is the (sub)category of the signatures with variables with first component $Σ$. Hence, the main difference is that contexts are an abstract notion (from a context it is possible to extract a sorted set of variables by means of a forgetful functor), while here we adopt a concrete approach where signatures with variables are pairs. As obvious consequence, context institutions are more general; on the other side, our choice allows a simpler treatment.

Another paper strictly related to our work since it presents a canonical extension of a standard institution by dynamic aspects is [7]. Starting from an institution $I$, [7] constructs an institution where a model is a class of pairs of $Σ$-models (called an
algebraic relation), intuitively modelling initial and final states of a transformation, and a sentence is like a sentence in \( I \), but where some symbols may refer to the either initial or final state, e.g. \( x' \geq x + n \). The same style is adopted e.g. in the assertions of the Eiffel language [19]. These sentences are analogous to our pre-post sentences; the main difference is that in [7] there is no notion of dynamic signature (introducing names for different state transformations) and, correspondingly, no notion of dynamic model formalizing the overall behaviour of a system; what can be specified is the behaviour of one given transformation.

An important question that one could ask is whether the framework presented in this paper is suitable for dealing with concurrent systems. The answer is twofold. On one hand, for what concerns the particular construction given in Sect. 4, this dynamic framework does not directly deal with concurrent behaviour. It is possible to associate in a canonical way a labelled transition system to a \( d \)-oid: for any given state \( A \), the possible transitions starting from \( A \) are all the triples \( (A, dop(\bar{a}), B) \) s.t. the dynamic operation \( dop \) applied to \( (A, \bar{a}) \) gives \( (B[a], b) \). Anyway, there is no notion of control state like in automata (the only possible form of control is given by the fact that a dynamic operation \( dop \) can be undefined on some state \( A \), and this can models that \( dop \) is not allowed in \( A \)). On the other hand, frameworks based on transition systems/automata can be easily viewed as particular cases of the abstract notion of dynamic framework presented in Section 3.3. Hence, as already said above, it is worthwhile to analyze the possibility of defining such frameworks in a parameterized way as done for the \( d \)-oid framework in this paper. The construction could also provide some result of composition via categorical constructions (limits and colimits). Indeed a very interesting topic to be investigated is how composition operators typical of concurrent processes should be integrated with composition operators typical of data-types taken from the underlying static framework. Some very recent research in that direction is [17].

Finally, we mention another topic which deserves further investigation, i.e. the relation between the "state-as-algebra" approach (states are models in a static framework), which we attempt to formalize in this paper (see e.g. [7, 11, 15, 16] for other work following this approach), and the "state-as-term" approach (states are elements in a set), which is taken in the traditional algebraic modelling of dynamics, see e.g. [3] for a survey. We have some preliminary result about that, showing in some simple case a correspondence between the two approaches.

Acknowledgements

I would like to thank all the participants of the informal meeting on FLIRTS (Formalism Logic Institution Relating, Translating and Structuring), held in Genova, October 1995, for many helpful comments and suggestions on a preliminary presentation of this work. I also thank the anonymous referees for their help in improving the paper.
Appendix A

Definition A.1. An institution is a tuple \( \langle \text{Sig}, \text{Mod}, \text{Sen}, \vdash \rangle \) where
- \( \text{Sig} \) is a category whose objects are called signatures;
- \( \text{Mod} \) is a functor, \( \text{Mod} : \text{Sig}^{op} \to \text{Cat} \); for any signature \( \Sigma \), objects in \( \text{Mod}(\Sigma) \) are called models over \( \Sigma \) or \( \Sigma \)-models, and morphisms are called \( \Sigma \)-morphisms; for any signature morphism \( \sigma : \Sigma_1 \to \Sigma_2 \), the functor

\[
\text{Mod}(\sigma) : \text{Mod}(\Sigma_2) \to \text{Mod}(\Sigma_1)
\]

is called the reduct functor and denoted by \( -|_{\sigma} \);
- \( \text{Sen} \) is a functor, \( \text{Sen} : \text{Sig} \to \text{Set} \); for any signature \( \Sigma \), the elements of \( \text{Sen}(\Sigma) \) are called sentences over \( \Sigma \); for any morphism of signatures \( \sigma \), \( \text{Sen}(\sigma) \) is denoted by \( \sigma \) when there is no ambiguity;
- \( \forall \alpha \in |\text{Sig}| \), \( \vdash_\Sigma \) is a relation over \( |\text{Mod}(\Sigma)| \times |\text{Sen}(\Sigma)| \) s.t. for any morphism \( \sigma : \Sigma_1 \to \Sigma_2 \), the satisfaction condition

\[
A \vdash_\Sigma \sigma(\phi) \text{ iff } A|_{\sigma} \vdash_\Sigma, \phi
\]

holds for any \( \Sigma_2 \)-model \( A \) and sentence \( \phi \in \text{Sen}(\Sigma_1) \).

Def/Prop. A.2. The functor \( \text{SSet} : \text{Set}^{op} \to \text{Cat} \) is defined as follows:
- for any set \( S \) (whose elements are called sorts), \( \text{SSet}(S) \) is the category whose objects are \( S \)-families of sets, called sorted sets over \( S \) or \( S \)-sorted sets, and whose morphisms are \( S \)-families of (partial) maps;
- for any map \( \delta : S_1 \to S_2 \),
  - \( \forall X \in |\text{SSet}(S_2)| \), \( (X|_{\delta})_s = X_{\delta(s)} \), \( \forall s \in S_1 \);
  - \( \forall f : X \to Y \) morphism in \( \text{SSet}(S_2) \), \( (f|_{\delta})_s(a) = f_{\delta(s)}(a) \), \( \forall a \in X|_{\delta}, \forall s \in S_1 \).
If \( X \) is an \( S \)-sorted set, \( s \in S \), then \( X\{x \mapsto s\} \) denotes the \( S \)-sorted set defined by \( X\{x \mapsto s\}_s = X_s \cup \{x\} \), \( X\{x \mapsto s\}_s' = X_{s'} \), \( \forall s' \neq s \). We abbreviate the empty family \( \{\emptyset\}_s \in S \) by \( \emptyset \), \( \emptyset\{x \mapsto s\} \) by \( \{x \mapsto s\} \).

Def/Prop. A.3. For any map \( \delta : S_1 \to S_2 \), we denote by \( \delta : \text{SSet}(S_1) \to \text{SSet}(S_2) \) the functor defined by
- \( \forall X \in |\text{SSet}(S_1)| \), \( \delta(X)_s = \bigcup \{X_{s_1} \mid \delta(s_1) = s_2\} \), \( \forall s_2 \in S_2 \); let \( i_X^s \) denote the injection from \( X \) into \( \delta(X)_s \), i.e. \( \forall s_1 \in S_1 \) with \( \delta(s_1) = s_2 \), \( i_X^s \), is the injection from \( X_{s_1} \) into \( \delta(X)_s \);
- \( \forall f : X \to Y \) morphism in \( \text{SSet}(S_1) \),

\[
\delta(f)_s(i_X^s(x)) = f_{s_1}(x), \forall x \in X_{s_1}, s_2 = \delta(s_1).
\]

It is easy to prove that \( \delta \) is the left adjoint of \( -|_{\delta} \) : \( \text{SSet}(S_2) \to \text{SSet}(S_1) \), \( i_X^s \) being the unit of the adjunction (see [21]). We can generalize this result as shown below.

Proposition A.4. Let the diagram \( \Delta \) in Fig. 8 commute in Set.
We denote by $\eta^A : h_1 \circ \delta \rightarrow - \circ h_2$ the natural transformation defined by
\[
\forall X \in |\text{SSet}(S_2)|,
\eta^A_X(i^0_{X,0}(x)) = i^0_{X,1}(x), \forall x \in X_{[\delta]}.
\]

If $\Delta$ is a commuting diagram consisting of two diagrams $\Delta_1, \Delta_2$, as shown in Fig. 9, then, $\forall X \in |\text{SSet}(S_3)|, \eta^A_X = (\eta^A_{X_1})_{\delta'} \circ \eta^A_{X_0}$.

**Proof.** Let us assume for concreteness that the disjoint union is represented as follows:
\[
\forall s : S_1 \rightarrow S_2, X \in |\text{SSet}(S_1)|,
\delta(X)_s = \{ (x,s_1) \mid x \in X_{s_1}, \delta(s_1) = s \}, \forall s \in S_2.
\]
Hence $i^0_X(x) = (x,s_1), \forall x \in X_{s_1}, \forall s_1 \in S_1; \eta^A_X((x,s_1)) = (x,\delta(s_1)), \forall x \in (X_{[\delta]})_{s_1} = X_{(\delta_{s_1})}, \forall s_1 \in S_1.

First, we show that $\eta^A_X$ is well-defined, $\forall X \in |\text{SSet}(S_2)|$. Indeed, $\forall s' \in S_1$,
\[
\forall x' \in (h_1(X_{[\delta]}))_{s_1'}, x' = (x,s_1) \text{ for some } s_1 \in S_1, x \in (X_{[\delta]})_{s_1}, \text{ with } h_1(s_1) = s'.
\]
Then $x \in X_{(\delta_{s_1})}$ by definition of $-_{[\delta]}$, hence $\langle x, \delta(s_1) \rangle \in (h_2(X))_{h_2(\delta(s_1))=\delta'(h(s_1))} = (h_2(X)_{[\delta']})_{h_1(s_1)=s'}$. 

Fig. 8.

Fig. 9.
Second, we show that $\eta^4$ is a natural transformation, i.e. $\forall f : X \to Y$ morphism in $\mathbb{S}Set(S_2)$, the diagram in Fig. 10 commutes in $\mathbb{S}Set(S'_2)$. Indeed, $\forall s'_1 \in S'_1$,

\[
\forall x' \in (h_1(X_{\delta}))_{s'_1}, x' = \langle x, s_1 \rangle \text{ for some } s_1 \in S_1, x \in (X_{\delta})_{s_1}, \text{ with } h_1(s_1) = s'_1.
\]

Assume $f_{\delta(s_1)}(x) = y$ (the case $f_{\delta(s_1)}(x)$ undefined is analogous). Then we have

\[
\begin{align*}
(h_1(f_{\delta}))_{s'_1}(\langle x, s_1 \rangle) &= \langle y, s_1 \rangle, \\
\eta^4_X(\langle x, s_1 \rangle) &= \langle x, \delta(s_1) \rangle, \\
(h_2(f))_{s'_1}(\langle x, \delta(s_1) \rangle) &= (h_2(f))_{s'_1}(\langle x, \delta(s_1) \rangle)
\end{align*}
\]

Consider now the diagram in Fig. 9. We first verify the functionality.

Indeed, $\eta^4_X : h_1(X_{\delta_2 \circ \delta_1}) \to h_3(X)_{\delta_2 \circ \delta_1}$; on the other side,

\[
\begin{align*}
(h_1(f_{\delta}))_{s'_1}(\langle x, s_1 \rangle) &= \langle y, s_1 \rangle, \\
\eta^4_X(\langle x, s_1 \rangle) &= \langle x, \delta(s_1) \rangle, \\
(h_2(f))_{s'_1}(\langle x, \delta(s_1) \rangle) &= (h_2(f))_{s'_1}(\langle x, \delta(s_1) \rangle)
\end{align*}
\]

Moreover,

\[
\forall s'_1 \in S'_1, \forall x' \in (h_1(X_{\delta_2 \circ \delta_1}))_{s'_1}, x' = \langle x, s_1 \rangle \text{ with } s_1 \in S_1. \text{ Then}
\]

\[
\begin{align*}
\eta^4_X(\langle x, s_1 \rangle) &= \langle x, \delta(s_1) \rangle \\
\eta^4_X(\langle x, s_1 \rangle) &= \langle x, \delta(\delta_1(s_1)) \rangle \\
\eta^4_X(\langle x, s_1 \rangle) &= \langle x, \delta_2(\delta_1(s_1)) \rangle, \text{ and } \eta^4_X(\langle x, s_1 \rangle) = \langle x, \delta_2(\delta_1(s_1)) \rangle.
\end{align*}
\]

\[\square\]

**Definition A.5.** Let $d: D\Sigma_1 \to D\Sigma_2$ (resp., $\langle d, h \rangle : \langle D\Sigma_1, X_1 \rangle \to \langle D\Sigma_2, X_2 \rangle$) be a morphism of d-oid signatures (resp. d-oid signatures with variables). Then, the map

\[DT(d): DT(D\Sigma_1) \to DT(D\Sigma_2)\]

(resp. $DT(\langle d, h \rangle): DT(\langle D\Sigma_1, X_1 \rangle) \to DT(\langle D\Sigma_2, X_2 \rangle)$)

denoted by $d\sigma$ (resp. $\langle d\sigma, h \rangle$) when there is no ambiguity, is inductively defined in Fig. 11, where we write

\[dt \mapsto dt' \text{ for } d\sigma(dt) = dt' \text{ (resp. } \langle d\sigma, h \rangle(dt) = dt').\]
Definition A.6. Let $d\sigma: D\Sigma_1 \rightarrow D\Sigma_2$ (resp., $\langle d\sigma, h \rangle: \langle D\Sigma_1, X_1 \rangle \rightarrow \langle D\Sigma_2, X_2 \rangle$) be a morphism of d-oid signatures (resp. d-oid signatures with variables), $d\sigma$ with static part $\sigma$. Then, the map

$$D\text{Sen}(d\sigma): D\text{Sen}(D\Sigma_1) \rightarrow D\text{Sen}(D\Sigma_2)$$

(resp. $D\text{Sen}(\langle d\sigma, h \rangle): D\text{Sen}(\langle D\Sigma_1, X_1 \rangle) \rightarrow D\text{Sen}(\langle D\Sigma_2, X_2 \rangle)$)

denoted by $d\sigma$ (resp. $\langle d\sigma, h \rangle$) when there is no ambiguity, is inductively defined in Fig. 12, where we write

$$d\phi \rightarrow d\phi' \quad \text{for} \quad d\sigma(d\phi) = d\phi' \quad \text{(resp.} \quad \langle d\sigma, h \rangle(d\phi) = d\phi')$$

Altogether, we have defined a functor

$$D\text{Sen}: D\text{Sig} + D\text{Sig}_\text{Var} \rightarrow \text{Set}.$$

Definition A.7. Let $I = \langle \text{Sig}, \text{Mod}, \text{Sen}, \llbracket \cdot \rrbracket \rangle$ and $I' = \langle \text{Sig}', \text{Mod}', \text{Sen}', \llbracket \cdot \rrbracket' \rangle$ be two institutions. Then an institution morphism from $I$ into $I'$ is a triple $\langle \Phi, \alpha, \beta \rangle$ where

- $\Phi$ is a functor, $\Phi: \text{Sig} \rightarrow \text{Sig}'$;
- $\alpha$ is a natural transformation $\alpha: \text{Sen}' \circ \Phi \rightarrow \text{Sen}$;
- $\beta$ is a natural transformation, $\beta: \text{Mod} \rightarrow \text{Mod}' \circ \Phi$
The condition
\[
\beta_{\Sigma}(A) \models_{\Phi(\Sigma)} \phi' \iff A \models_{\Sigma} \phi.
\]
holds for any \( A \in \mathcal{M}_{\Sigma} \) and \( \phi' \in \text{Sen}'(\Phi(\Sigma)) \).

References

