Algebraic Foundations for Specification Refinements

Pablo F. Castro[1,2(✉)] and Nazareno Aguirre[1,2]

[1] Departamento de Computación, FCEFQyN, Universidad Nacional de Río Cuarto, Río Cuarto, Córdoba, Argentina
{pcastro,naguirre}@dc.exa.unrc.edu.ar

[2] Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Buenos Aires, Argentina

Abstract. In this paper we present a mathematical framework tailored for reasoning about specification/program refinements. The proposed framework uses formal concepts coming from Institution Theory and Category Theory, such as theories and morphisms, to capture the notion of specification/program refinement. The main benefits of the proposed mathematical theory are its generality and compositionality, that is, it is based on abstract concepts that can be used to reason about refinements in different formal settings (such as Z, B, VDM, Alloy, statecharts and others), as well as it heavily relies upon the notion of component, thus enabling modular reasoning over the process of specification/program refinement.

1 Introduction

Software Verification, i.e., the rigorous evaluation of a formal specification against a corresponding implementation, is perhaps the most widely acknowledged advantage of formal specification notations over their informal counterparts. Still, the task of formally verifying that a system correctly implements a specification is in general a complex task (although under certain restrictions, it can be algorithmically decided, e.g., via model checking), since systems and specifications are usually of a very different nature: the former are intrinsically operational and verbose, while the latter tend to be declarative and more concise.

An alternative to verification, strongly based on formal specification, consists of avoiding having to formally prove that an implementation complies with a specification, and instead *generate* a correct-by-construction implementation from a specification. Of course, this cannot be completely automated, but via a series of small, step by step, *sound* refinement steps, one may transform a declarative specification into an operational implementation complying with it. This has an obvious impact in scalability, since the "large" problem of system verification is modularised in a number of small sound steps, made by employing proved-correct *refinement rules*.

For most formal specification languages, such as Z [29], B [1] and VDM [19], the notion of refinement is usually a critical component. In effect, the B Method

L. Ribeiro and T. Lecomte (Eds.): SBMF 2016, LNCS 10090, pp. 112–128, 2016.
DOI: 10.1007/978-3-319-49815-7_7

provides the possibility of *refining* a specification, until an implementation is reached. Each refinement step generates a set of proof obligations, whose validity guarantees the correctness of the refinement. Z does not provide a language for refinement within the Z notation, but "external" notations can be systematically employed for refining Z specifications, as is described for instance in [6, 29]. In general, the approaches to refinement tend to be language/formalism dependent, since refinement rules depend on the specification language's constructs. In this work, we present an abstract categorical formulation of refinement, allowing us to capture the essentials of refinements in model based specification languages. Our approach, based on well known concepts from the theory of institutions, is defined at a level of abstraction that makes it formalism-independent, and enables us to capture what is the precise semantic relationship that must hold between a (structured) specification and its refinements, whatever the language these specifications and refinements correspond to. It also allows us to distinguish the more traditional refinement based on reducing nondeterminism, from an orthogonal kind of refinement, that of (abstract) state representation/implementation, and understand the relationship between them. We believe that an important aspect of the framework presented below is that its level of generality allows one to apply refinement over heterogeneous specifications, that is, specifications that are made using different formal languages; an example of this is CSP-Z [12] which uses Z for producing specification of states and operations and CSP for expressing the dynamic behavior of the systems; another important characteristic of our approach is that it enables compositional reasoning about refinements, that is, specifications that are structured in a collection of components can be refined by reasoning at the component level, simplifying in this way the task of refining.

The paper is structured as follows. In Sect. 2 we introduce the basic background assumed throughout this paper, we introduce the framework in Sects. 3 and 4, together with its properties. In Sect. 5 we present some conclusions and further work.

2 Preliminaries

In the following we use some basic notions of category theory. A *category* is a mathematical structure composed of two collections: a collection $a, b, c, \ldots$ of *objects*, and a collection $f, g, h, \ldots$ of arrows (or morphisms). Every arrow has two associated objects, its domain and codomain; we write $f : a \to b$ to indicate that a (resp. b) is the domain (resp. codomain) of arrow f. There are two basic operations involving arrows: the *identity*, that given an object a, it returns an arrow $id_a : a \to a$, and the *composition* which, given arrows $f : a \to b$ and $g : b \to c$, returns an arrow $f; g : a \to c$. Arrow composition is associative; identity arrows satisfy: $f; id_a = f$ and $id_b; f = f$, for every $f : a \to b$. A natural example of category is **Set**, made up of the collection of sets and the collection of functions between sets. A *functor* is essentially a homomorphism between categories. Given a category $\mathbf{C}$, we denote by $|\mathbf{C}|$ its collection of objects, and by $||\mathbf{C}||$ its collection of arrows.

Given a category $\mathbf{C}$, a bicategory [5] is composed of: *(i)* a collection of objects, called 0-cells; *(ii)* a category $\mathbf{C}(A, B)$, for each pair of 0-cells A, B, whose objects are called 1-cells and whose arrows are called 2-cells; and *(iii)* a (bi)functor: $; : \mathbf{C}(A, B) \times \mathbf{C}(B, C) \to \mathbf{C}(A, C)$, which satisfies some coherence properties: it must have an identity, and it must be associative. In bicategories, there are two kinds of arrows: the horizontal and the vertical ones. We refer the interested reader to [3], for an introduction to category theory. We will assume throughout the paper that the reader has some basic knowledge of category theory.

Since our main goal is to introduce the framework in an abstract language-independent manner, we do not use a particular logic to introduce the concepts; instead, we use the abstract setting of Institutions. It is useful to recall the definition of Institution:

Definition 1. *An institution* [13] *is given by:* (i) *a category* **Sign** *of signatures;* (ii) *a functor* $sen : \mathbf{Sign} \to \mathbf{Set}$*, that sends each signature to its set of formulas;* (iii) *a functor* $Mod : \mathbf{Sign}^{op} \to \mathbf{Cat}$*, that sends each signature to the category of its models*[1]*; and* (iv) *a collection of relations* $\models_\Sigma$ *(satisfaction relations relating models of a signature to formulas of the signature), that satisfies the following requirement:* $Mod(\sigma)(M') \models_\Sigma \phi \Leftrightarrow M' \models_\Sigma sen(\sigma)(\phi)$ *for any formula* $\phi \in sen(\Sigma)$ *and* $\sigma : \Sigma \to \Sigma'$.

Institutions are an abstract formulation of Model Theory. The last requirement in Definition 1 captures the fact that truth does not depend on notation.

Example 1 (Higher Order Logic). Let us give a standard example of Institution, Higher Order Logic (or simply **HOL**) is one of the basic institutions used in computer science. Here we follow the definition given in [9]. Given a set of sorts S, the set of *types* of S (denoted $\overline{S}$) is the least set such that: $S \subseteq \overline{S}$, if $s_1, s_2 \in S$ then $s_1 \to s_2$. A **HOL** signature is a tuple (S, F) where S is a set of sorts and F is a set of typed constants $\{F_s \mid s \in \overline{S}\}$. A morphism between signatures $\sigma : (S, F) \to (S', F')$ is a function $\sigma : S \to S'$, and a family of functions $\{\sigma_s : F_s \to F'_{\sigma^*(s)} \mid s \in \overline{S}\}$, where σ^* is the inductive extension of σ to $\overline{S}$. On the other hand, models in **HOL** are given by interpreting each type as a set. A model M of a signature (S, F) maps each type s to a set M_s (mapping types of the form $s \to s'$ to functions). A morphism between (S, F) models is a collection of functions $m_s : M_s \to N_s$, such that for any $f \in M_{s \to s'}$ (for $s, s' \in \overline{S}$) we have: $m_{s'} \circ f = m_{s \to s'}(f) \circ m_s$. Terms of HOL are defined as usual, any $f \in F_s$ is said to be a term of type s; and $t(t')$ is a term of type s_2, when t is of type $s_1 \to s_2$ and t' is of type s. Sentences of signature (S, F) are built up from equations by using the usual boolean connectives and quantifiers, the functor $sen : \mathbf{Sign} \to \mathbf{Set}$, sends each signature to the sets of its sentences. It is direct to define the relation $\models$. The institution **HOL** is the tuple $(\mathbf{Sign}_{HOL}, sen_{HOL}, Mod_{HOL}, \models)$ as defined above.

[1] $\mathbf{Sign}^{op}$ denotes the dual category of **Sign**, obtained by reversing arrows. This is so since reducts and translations go in different directions.

A restricted version of the institution **HOL** is obtained by requiring signature morphisms to preserve types. We have used this institution to capture constructions coming from the Z notation [7].

Example 2 (Z Notation). We describe briefly this institution, the technical details can be found in [7]. The institution $\mathbf{Z} = (\mathbf{Zign}, \mathit{sen}, \mathit{Mod}, \models)$ is as follows. Signatures in **Zign** are tuples (V, T) where V is a collection of typed variables, and T the basic types. A morphism $\sigma : \Sigma \to \Sigma'$ between signatures is defined as in Example 1, but we require that, for any variable v, the translated variable $\sigma(v)$ has the same type as v. The functor *sen* is defined as in **HOL**, we consider the standard mathematical operators usual in **Z** (see [29]), which can be defined in **HOL**. The models and the $\models$ relation are the same as Example 1.

We assume the reader is familiar with the **Z** notation, standard references are [26, 29]. Another interesting example is the institution of communicating sequential processes [23] (named **CSP**), let us introduce the basics of this formal construction which will be useful in the rest of the paper.

Example 3 (Communicating Sequential Processes). The category of CSP signatures (denoted $\mathbf{Sign}_{CSP}$) has as objects tuples (A, N), where A is an alphabet of communications, and $N = (\overline{N}, \mathit{sort}, \mathit{param})$ contains the basic descriptions of processes: N is a collection of process names, *sort* is a function indicating the collection of possible communication in a given process (i.e., $\mathit{sort}(p) \subseteq A$); and *param*, for each process, returns the collection of its parameters. A morphism $\sigma : (A, N) \to (A', N')$ is given by functions $\alpha : A \to A'$ (translating alphabets) and $\nu : N \to N'$ (translating processes), respectively. Obviously, some coherence conditions are imposed over α and ν (e.g., preservation of parameters types, etc.) the interested reader is referred to [23]. There are different ways of giving semantics to CSP, one of them is to consider the set of possible traces of each process, this is called the *trace model*, model reducts can be defined directly over models, and model morphisms are captured as set inclusions; this gives rise to the category Mod_{CSP} of CSP models. On the other hand, sentences are given by standard CSP definitions by means of equations (see [15] for examples). The relation $M \models p(x_0, \ldots, x_k) = P$ holds when the interpretation of process p refines the set of traces defined by P[2]. For the sake of clarity we omit the technical definitions here, but them can be consulted in [23].

3 A Category of Refinements

Before describing our formalization of refinement, let us we introduce the formal vehicle we use to express system specifications. The basic notion we employ to specify the states of a system is that of *theory presentation* [10].

[2] In [23] this definition is stronger and the authors require that the sets of traces of both terms have to be the same, here we focus on refinement, and since that we only require an inclusion between the corresponding set of traces.

Definition 2. *Given an Institution $\mathbf{I} = \langle \mathbf{Sign}, sen, Mod, \models \rangle$, a theory presentation $S = \langle \Sigma, \Phi \rangle$ is made up of a signature $\Sigma \in |\mathbf{Sign}|$ and a set $\Phi \subseteq sen(\Sigma)$ of formulas (the axioms of the theory).*

Intuitively, a theory presentation is used to formally describe the states of the system. We have used the concept of theory presentation in [7] to capture the notion of *schema* employed in the Z notation; the generality of this concept allows us to give semantics to schema calculus through categorical constructions. Note also, that the given definition is independent of the logic used to described the state of the system, other logics can be used, some examples are show below. Let us give a simple example of how we can use theory presentation to express state specifications:

Example 4. Let us give a first example of theory and morphism in an Institution. Consider a simple specification of a memory, it can be written in the Institution of Z specifications ($\mathbf{Z}$), as follows:

$$Mem = ((\{Data, Nat\}, \{mem : Nat \twoheadrightarrow Data\}), \{\{true\}\})$$

which contains two types $Data$ and Nat and a term of type $Nat \twoheadrightarrow Data$, representing a function that maps naturals to data; there is no axioms. From now on, we write Z specifications using Z notation, that is:

$$Mem \mathrel{\widehat{=}} [Data : \mathbb{N};\ mem : \mathbb{N} \twoheadrightarrow \mathbb{N} \mid True]$$

On the other hand, a morphism between theory presentations is a translation of symbols that preserves properties:

Definition 3. *A theory morphism $\tau : \langle \Sigma, \Phi \rangle \to \langle \Sigma', \Phi' \rangle$ is a signature morphism $\sigma : \Sigma \to \Sigma'$ that satisfies the following condition: $\forall\, \phi \in \Phi \bullet \Phi' \models sen(\sigma)(\phi)$.*

Intuitively, a morphism between two specifications corresponds to two important concepts: specification embedding, that is, putting a specification into a wider system; and specification strengthening. Let us give an example in the Institution **CSP**, as presented in Sect. 2.

Example 5. Consider the following specification of a process:

$$\Gamma_0 = \{\, VDM1 = coin \to (choc \to VDM1 \mid coffee \to VDM1)\}$$

a vending machine that, after receiving a coin, serves chocolate or coffee. The signature of this process is given by $A = \{coin, choc\}$, we have a unique process name: $VDM1$, where $sort(VDM1) = \{choc, coin\}$ and $param(VDM1) = ()$. Indeed, we can devise a more restrictive version of the vending machine:

$$\Gamma_1 = \{\, VDM2 = coin \to choc \to VDM2\}$$

where the functions *sort* and *param* are defined as above. As can be verified, The identity translation $\sigma : VDM1 \to VDM2$ is a morphism between these two specifications, it represents the refinement of $VDM1$ achieved by removing some internal non-determinism.

For any institution $\mathbf{I}$, it is direct to prove that specifications and morphisms are a category (see [10] for the technical details).

Definition 4. *Given an institution* $\mathbf{I} = \langle \mathbf{Sign}, sen, Mod, \models \rangle$*,* $\boldsymbol{Pres}_{\mathbf{I}}$*, is the category composed of: 1. Theory presentations (see Definition 2) as objects, 2. Theory morphisms (see Definition 3) as morphisms.*

We just write $\mathbf{Pres}$ instead $\mathbf{Pres_I}$, when $\mathbf{I}$ is clear by context. Given any presentation s, we denote by $Ax(s)$ its sets of axioms, and $Sign(s)$ its signature. Note that, for any institution, $Sign : \mathbf{Pres} \rightarrow \mathbf{Sign}$ is a functor.

Another important concept when constructing software specifications is that of operation, usually operations are specified by stating their pre and post conditions. In our setting, operations are also logical theories, capturing their corresponding pre-post relations via formulas. Consider the following diagram in $\mathbf{Pres_I}$ (for any institution $\mathbf{I}$):

$$\begin{tikzcd} & Op & \\ S \arrow[ur, "i"] & & S' \arrow[ul, "j"'] \end{tikzcd}$$

In this diagram, S is an state specification, $i : S \rightarrow Op$ is an inclusion (the embedding of S into the operation), while S' (denoting the states after the operation execution) is a theory obtained by priming the symbols in S, and $j : S' \rightarrow Op$ is the embedding of S' into the operation specification.

Let us give an example of operation for the specification of a memory given above.

Example 6. Given the state specification Mem the following is an operation over it:

$$Write \mathrel{\widehat{=}} [\Delta Mem;\ a? : \mathbb{N};\ i? : Data \mid Mem' = Mem \oplus \{a \mapsto d\}]$$

Here note that ΔMem means that the signature of Mem and its axioms are included as part of $Write$ and similarly for Mem', i.e., the inclusions $i : Mem \rightarrow Write$ and $j : Mem' \rightarrow Write$ are just the identity mappings.

In order to put together data domain and operation specifications, the latter understood as the above diagrams, the concept of *bicategory* [5] can be used. In effect, domain specifications (theories) correspond to 0-cells, whereas operations are diagrams of the form $S \rightarrow Op \leftarrow S'$, called cospans. The morphisms between cospans, that make the corresponding diagram commute, are the 2-cells. Cospans are in fact one of the typical examples of bicategories, where the two classes of arrows are the operations (horizontal arrows) and the morphisms between these operations (vertical arrows).

Let us see how we build this construction. Given any institution $\mathbf{I}$, we define the bicategory of states and operations over $\mathbf{I}$ as the bicategory of cospans over $\mathbf{Pres_I}$ (a proof that it is already a bicategory can be found in [5]).

Definition 5. ***Spec*** *is the* bicategory of ***I***-specifications*, defined as the structure composed of:* (i) *the set* $|\mathbf{Pres}|$ *as its set of objects;* (ii) *for each pair of theory presentations* S, S'*, the category* $OP(S, S')$ *of cospans between* S *and* S' *(called 1-cells), and morphisms between cospans (called 2-cells); and* (iii) *the composition between 2-cells is defined as usual by using the composition (i.e., pushouts) of cospans (denoted by* $;$*).*

A specification is a subcategory of **Spec**, the subcategory generated by the corresponding schemas and operations. We denote by $Op : S \Rightarrow S'$ the existence of operation Op from S to S', i.e., the cospan $S \rightarrow Op \leftarrow S'$. From now on, we assume that **Sign** is an *adhesive* category [20]. Roughly speaking, this means that pushouts (generalized unions) are *well-behaved*; this, for example, ensures us that **Sign** has nice properties that allow us to put together different parts of a specification. Examples of adhesive categories are the categories of sets, graphs, labelled graphs, trees, amongst others. We also assume that **Sign** has a strict initial object (that is, any arrow $\Sigma \rightarrow \emptyset$ is an isomorphism). This holds for most logics; for instance, in propositional logic, the empty set is the initial signature (which is strict). These basic assumptions imply, among other things, that **Sign** has finite colimits; this is important since the colimit is the standard construction to put together specifications [13].

Now, let us start dealing with the problem of refinement. Operation refinement is typically understood as a kind of *strengthening*. As we already mentioned, arrows in $\mathbf{Pres_I}$ capture the concept of specification strengthening. However, these morphisms are not adequate for formalizing the notion of operation refinement, since the strengthening associated with operations make a distinction between preconditions and postconditions: they correspond to weakening preconditions and strengthening postconditions. First, we need to distinguish preconditions from postconditions, thus we require that any operation Op has to be an *extension* of the coproduct $S + S'$ of S and S', that is, we assume the following situation regarding any operation Op:

$$\begin{tikzcd}
 & Op & \\
 & S + S' \arrow[u, "u"] & \\
S \arrow[ur, "i"] \arrow[uur, bend left] & & S' \arrow[ul, "j"'] \arrow[uul, bend right]
\end{tikzcd}$$

and we require that the arrow u be monic, i.e., symbols from S and S' are not mixed in Op. This will be useful for calculating pre and postconditions. An essential property that we must guarantee is that, under this characterization of operation specification, operations can be composed[3]. This is guaranteed by the following Theorem.

[3] Note that this is straightforward to prove for standard cospans when we have a finitely cocomplete category.

Theorem 1. *Given an institution* $\mathbf{I}$*, if* $Sign : \mathbf{Pre} \to \mathbf{Sign}$ *is faithful, then, given operations* $Op_1 : S \Rightarrow S'$ *and* $Op_2 : S' \Rightarrow S''$*, the composition (denoted by* $Op_1 \fatsemi Op_2$*) exists, and is obtained by taking the colimit of the diagram composed by solid arrows below:*

$$\begin{tikzcd}
 & & Op_1 \fatsemi Op_2 & & \\
 & Op_1 \arrow[ur, dashed] & S + S' \arrow[u, dashed, "u"] & Op_2 \arrow[ul, dashed] & \\
S \arrow[ur] \arrow[urr, dashed] & & S' \arrow[ul] \arrow[ur] & & S'' \arrow[ul] \arrow[ull, dashed]
\end{tikzcd}$$

Proof. *Since the category* $\mathbf{Pre}$ *is finitely cocomplete (Sign reflects colimits* [13]*), we know that the colimit of the diagram exists. We have to prove that the arrow* $u : S + S'' \to Op_1 \fatsemi Op_2$ *is mono. Since* $\mathbf{Sign}$ *is adhesive and has strict initial elements, the injection morphisms of a coproduct are monos, i.e., the arrows* $i : Op_1 \to Op_1 + Op_2$ *and* $j : Op_2 \to Op_1 + Op_2$ *are monos. Now, since* $Op_1 \fatsemi Op_2$ *is a colimit, we have an arrow* $Op_1 \fatsemi Op_2 \to Op_1 + Op_2$*; therefore, by properties of monic arrows, the morphisms* $Op_1 \to Op_1 \fatsemi Op_2$ *and* $Op_2 \to Op_1 \fatsemi Op_2$ *are monos. That is, the arrows* $f : S \to Op_1 \fatsemi Op_2$ *and* $g : S'' \to Op_1 \fatsemi Op_2$ *are monos (since they are compositions of monic arrows). Therefore, the arrow* $[f, g] = u$ *is monic (since* $\mathbf{Sign}$ *is adhesive and Sign reflects monos).*

As we explained, we need to factor the precondition and postcondition from an operation specification, to describe what a refinement is. Let us first deal with preconditions. A precondition is a predicate prescribing for which states an operation is correctly defined. Categorically, and given a component S, this concept of precondition over S corresponds to an arrow $pre : S \to P$. That is, *pre* is an extension of S which characterizes the states where the precondition is true. We require that *pre* preserves language; that is: $Sign(pre)$ must be iso in $\mathbf{Sign}$. Now, preconditions can be weakened during the refinement of an operation. This corresponds to a construction called *coslice category*. The coslice category $S \downarrow \mathbf{Pres}^{op}$ has arrows $Pre : S \to P$ as objects; its morphisms are arrows $f : pre \to pre'$ that make the following diagram commute in $\mathbf{Pre}$:

$$\begin{tikzcd}
 & S \arrow[dl, "pre"'] \arrow[dr, "pre'"] & \\
P & & P' \arrow[ll, "f"]
\end{tikzcd}$$

Notice that we used $\mathbf{Pres}^{op}$, since arrows go "in the opposite direction" for preconditions. We will denote by $\mathbf{pre(S)}$ the subcategory $S \downarrow \mathbf{Pres}^{op}$, of preconditions of S. The same observations that we made for preconditions can be extrapolated to postconditions. More precisely, a postcondition is an arrow $post : S + S' \to Q$ describing the correct final states of a given operation. Notice that, for postconditions, we include the language of the initial state (i.e., S). The reason for this is that, in model based specification languages, it is customary to often describe the "post states" in relation to the "pre states", i.e., to describe the transition relation of the operation as the postcondition of the operation. Using the (inclusion) arrows $i : S \to S + S'$ and $j : S' \to S + S'$, we obtain the cospan:

$$\begin{tikzcd}
& Q & \\
S \arrow[ur, "i;Post"] & & S' \arrow[ul, "j;Post"']
\end{tikzcd}$$

We require these two arrows to be extensions; furthermore, i; *post* must be conservative (see [11] for the definition of these concepts); intuitively this means that a postcondition does not add any restrictions on initial states.

The category $S + S' \downarrow$ **Pres** gives us the base category to reason about postconditions of operations transforming S. We denote by $\mathbf{post}(S')$ the subcategory of $S + S' \downarrow$ **Pres** of postconditions. Notice that, as opposed to the case of preconditions, in this case the arrows go in the usual direction.

As we mentioned previously, in order to be able to refine operations we need to express them as composed by preconditions postconditions. Notice that, given a precondition $pre : S \to P$ and a postcondition: $post : S + S' \to Q$ of an operation Op, we can compose these as follows:

$$\begin{tikzcd}
& & Op & & \\
& & {[pre, post]} \arrow[u, "u"] & & \\
& P \arrow[ur] & & Q \arrow[ul] & \\
S \arrow[ur, "Pre"] \arrow[uuurr, bend left] & & S \arrow[ul, "Pre"'] \arrow[ur, "j;Post"] & & S' \arrow[ul, "i;Post"'] \arrow[uuull, bend right]
\end{tikzcd}$$

where $[pre, post]$ is the colimit of the above diagram. When the (unique) arrow $u : [pre, post] \to Op$ is conservative, we say that the operation $Op : S \Rightarrow S'$ can be factorized in $Pre : S \to P$ and $Post : S + S' \to S'$; in this case, we write Op as $[pre, post]$. The following theorem allows us to guarantee that every operation can be factorized, and therefore to treat operations as defined by pre and postconditions.

Theorem 2. *For any given institution* $\mathbf{I}$*, every operation in* $\mathbf{Spec_I}$ *can be factorized in a unique way (up to isomorphism).*

Proof. *Let us first prove that there is at least one factorization. Given* $Op : S \Rightarrow S'$*, suppose that* $S = \langle \Sigma_S, \Phi_S \rangle$*. Let us define* $pre : S \to P$*, where* $P = \langle \Sigma_P, i^{-1}(\Phi_{Op} \cap i(\Phi_{\Sigma_S})) \rangle$*, where* $i : S \to Op$*,* i^{-1} *is the usual pre image over sets and* Φ_{Σ_S} *denotes the set of all formulas generated from* Σ*. Note that* $pre : S \to P$ *is mono, since Sign reflects monos. Let us prove that it is a morphism between presentations. If we have that* $\phi \in i^{-1}(\Phi_{Op} \cap i(\Phi_{\Sigma_S}))$*, then* $i(\phi) \in \Phi_{Op} \cap i(\Phi_{\Sigma_S})$ *but therefore* $\Phi_{Op} \models i(\Phi)$*. Now, let us define the postcondition* $post : \Sigma + \Sigma' \to Q$*; we define* Q *as* $\langle \Sigma_S + \Sigma_{S'}, j^{-1}(\Phi_{Op} \setminus i(\Phi_{Pre})) \rangle$*. By using the identity* $\Sigma_S +$

$\Sigma_{S'} \to \Sigma_S + \Sigma_{S'}$, the proof that $post : \Sigma + \Sigma' \to Q$ is an arrow between theory presentations is similar to that of pre. Now, we need to prove that $[pre, post]$ is the unique (up to isomorphism) factorization. Consider the following diagram:

$$\begin{tikzcd}
 & & & & Op & & & & \\
 & & {[Pre, Post]} \arrow[urr, "f"] & & & & {[Pre', Post']} \arrow[ull, "g"'] & & \\
 & P \arrow[ur] \arrow[rr, dashed, bend left, "\overline{x}"] & & P' \arrow[urrr] \arrow[ll, dashed, bend left, "\overline{x}^{-1}"] & & Q \arrow[ulll] \arrow[rr, dashed, bend left, "\overline{y}"] & & Q' \arrow[ul, leftrightarrow] \arrow[ll, dashed, bend left, "\overline{y}^{-1}"] & \\
S \arrow[ur] \arrow[urrr] & & & & S \arrow[ulll] \arrow[ul] \arrow[ur] \arrow[urrr] & & & & S'' \arrow[ulll] \arrow[ul]
\end{tikzcd}$$

Let us show that there exist arrows x and y as shown above. Note that, since $Sign(pre) : Sign(S) \to Sign(P)$, and **Sign** *and $Sign(pre') : Sign(S) \to Sign(P')$ are iso in* **Sign***, we can define the following arrow $x = Sign(pre)^{-1}; Sign(pre') : P \to P'$. Note that we have the following diagram which commutes, since P and P' are pre-conditions:*

$$\begin{tikzcd}
 & Op & \\
P \arrow[ur, "f"] & & P' \arrow[ul, "g"'] \\
 & S \arrow[ul, "pre"] \arrow[ur, "pre'"'] &
\end{tikzcd}$$

The arrows $(f; pre) : P \to [P, Q]$ and $(g; pre')P' \to [P', Q']$ are conservative. Furthermore, since $Sign(pre)$ and (pre') are iso, we have an arrow in **Sign** *$x : Sign(P) \to Sign(P')$ which is iso. Let us prove that this arrow is a morphism between theory presentations: if $\phi \in P$, then $(f;pre)(\phi) \in Op$. Then, since these arrows commute (see the diagram above), we have that $(g; pre')(\phi) \in Op$, and therefore $(g;pre')^{-1}(\phi) \in P'$. So, by the commutativity of the diagram above, we get $x(\phi) \in P'$. Similarly, we can find a iso morphism $y : Q \to Q'$. Finally, by properties of colimit we get that $[P, Q]$ and $[P', Q']$ are isomorphic.*

We are now ready to define operation refinement. Given two operations Op and Op', factorized as $[pre, post]$ and $[pre', post']$, respectively, a *refinement* is composed of two arrows f and g, in the situation, involving the cospans of the two operations, captured in the following diagram:

$$\begin{tikzcd}
 & {[pre, post]} \arrow[d, "{[f,g]}"] & \\
 & {[pre', post']} & \\
S \arrow[uur, bend left] \arrow[ur] & & S' \arrow[uul, bend right] \arrow[ul]
\end{tikzcd}$$

Arrow f is in $\mathbf{pre}(S)$, while arrow g is in $\mathbf{post}(S')$. According to the definitions of these subcategories, an operation refinement is composed of a precondition weakening and a postcondition strengthening, precisely as we expected. The following result, stating that operations and operation refinements constitute a category, is an important one: it implies that operation refinements can be composed, an essential property for step by step refinement.

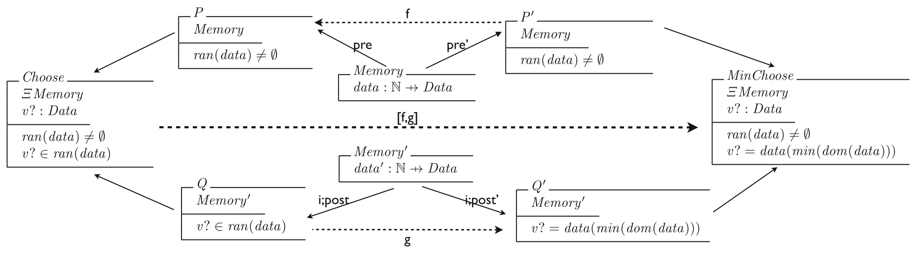
Theorem 3. *Given two theories S and S', the collection of operations $Op(S, S')$ between S and S', and refinement arrows between the corresponding factorizations, is a category, denoted by* $\mathbf{Ref}(S, S')$.

Furthermore, given factorizations $[pre, post] : S \to S'$ and $[pre', post'] : S' \to S''$, we can consider the following diagram:

$$\begin{tikzcd}
& & & & C & & & & \\
& P \arrow[urrr, dashed] & & Q \arrow[ur, dashed] & & P' \arrow[ul, dashed] & & Q' \arrow[ulll, dashed] & \\
S \arrow[ur, "pre"] & & S \arrow[ul, "pre"'] \arrow[ur, "i;post"] & & S' \arrow[ul, "j;post"'] \arrow[ur, "pre'"] & & S' \arrow[ul, "pre'"'] \arrow[ur, "i';post'"] & & S'' \arrow[ul, "j';post'"']
\end{tikzcd}$$

where C is the colimit of the base of the diagram (called cocone). Taking the factorizations of arrows $S \to C$ and $S'' \to C$, we obtain an object of $\mathbf{Ref}(S, S'')$. We can then define a bifunctor (composition) of factorizations $\fatsemi : \mathbf{Ref}(S, S') \times \mathbf{Ref}(S', S'') \to \mathbf{Ref}(S, S'')$. It is not hard to see that this bifunctor satisfies the coherence properties required for composition in bicategories (it is defined by using colimits in a similar way that it is done in cospan categories).

Let us present an example, illustrating our above construction. We have already introduced a specification of memories, with some operations. Consider an additional operation, called *Choose*, whose purpose is to nondeterministically choose an address, and returns the data stored in it. An implementation, or more concrete specification, of this operation may reduce nondeterminism, for instance by deterministically choosing a specific address to be read. A possible implementation would be to use the minimum of the addresses, and return the value read in it. This specification, called *MinChoose*, together with the more abstract *Choose*, their pre/postcondition factorizations and the refinement, are shown in the diagram below, together with the corresponding arrows. Notice that the arrows between schemas Q and Q' imply that any model of Q' would be a model of Q (semantic arrows go in the other direction).



Let us finally put together specifications, operations, and operation refinements. Note that bicategory **Pres** is unsuitable to subsume refinement, since arrows between cospans (the vertical arrows) capture the notion of specification strengthening, not refinement. In order to deal with this issue, we define a new class of arrows between cospans: given operations Op, Op', with factorizations $[Pre, Post]$ and $[Pre', Post']$, respectively, we define the bicategory **Spec** of specifications as follows.

Definition 6. *The structure of specifications (called* **Ref***) and refinements is defined as follows:*

- *The collection of 0-cells is given by the collection of theory presentations.*
- *For each pair of theories S and S', we have the category* ***Ref***(S, S') *as defined in Theorem 3, where cospans are the 1-cells, and refinements are the 2-cells.*
- *The composition* $⨟ : \mathbf{Ref}(S, S) \times \mathbf{Ref}(S', S'') \to \mathbf{Ref}(S, S'')$ *is as defined above.*

The following result shows the coherence of the above structure.

Theorem 4. ***Ref*** *is a bicategory.*

Proof. *That* $\mathbf{Ref}(S, S')$ *is a category follows from Theorem 4. The key of the proof is showing that* $⨟$ *behaves as a composition. In order to show this, it suffices to take the factorization of the colimit of the factorizations.*

3.1 Heterogeneous Refinements

Let us give a simple example of how the framework described in the section above can be used to combine notions of refinements coming from different formal systems. Consider the combination of Z with CSP, this formal system can be defined in diverse ways, we take the definition of the institution **CZP** (that combines CSP with Z) specifications given in [7].

Definition 7. *The institution* ***CZP*** *is defined as follows:*

- *Signatures are tuples (Σ_Z, Σ_{CSP}), where Σ_Z is a Z signature, and Σ_{CSP} is a CSP signature, and signature morphisms are pairs of signature morphisms.*
- *sen is defined pointwise: $sen(\Sigma_Z, \Sigma_{\mathsf{CSP}}) = (sen(\Sigma_Z), sen(\Sigma_{CSP}))$,*
- *Given a signature Σ_{CZP}, a model is this signature is a subset of the set:*

$$\{\langle a_0, \ldots, a_n \rangle, \langle \mathbf{I_0}, \ldots, \mathbf{I}_{n+1}, \rangle) \mid \langle a_0, \ldots, a_1 \rangle \in Mod(\sigma_Z) \wedge \mathbf{I}_j \in Mod(\Sigma_Z)\},$$

  that is collection of traces together with a set of interpretations in **Z** *representing the state changes of the system during the given execution.*
- *The satisfaction relation is defined as follows: $M \models \langle \pi, \phi \rangle$ iff $\pi_1(M) \models \pi$ and for every $\langle \mathbf{I}_1, \ldots, \mathbf{I}_{n+1} \rangle \in \pi_2(M)$ we have $\mathbf{I}_i \models \phi$,*

In this institution, a theory presentation is defined as follows [7]:

Definition 8. *A theory in **CZP** is a tuple $\langle \Sigma_{CSP}, \Sigma_Z, S, Ops, events, \pi\rangle$, where: 1. $\Sigma_{CSP} = \langle A, N\rangle$ is a signature in **CSP**, 2. Σ_Z is a signature in **Z**, 3. S is a collection of formulas, 4. $OPS = \{op_0 : S \Rightarrow S', \dots op_n : S \Rightarrow S'\}$ is a collection of operations over presentation $\langle \Sigma_Z, S\rangle$. 5. $event : A \to OPS$ is a function mapping events to operations, 6. π is a set of **CSP** processes.*

Now, we can define the notion of refinement of specification in **CZP**:

Definition 9. *Given theories presentations $P_i = \langle \Sigma^i_{CSP}, \Sigma^i_Z, S, Ops^i, events^i, \pi^i\rangle$, for $i \in \{0,1\}$ a **CZP** refinement $r : P_0 \to P_1$ is given by: 1. An arrow $z : \langle \Sigma^0_Z, S^0\rangle \to \langle \Sigma^1_Z, S^1\rangle$ in $\mathbf{Pres_Z}$, 2. An arrow $p : \langle \Sigma^0_{CSP}, \pi^0\rangle \to \langle \Sigma^1_{CSP}, \pi^1\rangle$ in $\mathbf{Pres_{CSP}}$, 3. A mapping $i : Ops^0 \to Ops^1$, such that, for each $o \in Ops$, there are arrows $r : o \to i(o)$ in $\mathbf{Ref_Z}$, and the following holds: $i \circ events^0 = events^1 \circ p$.*

Roughly speaking, a refinement in **CZP** is composed of refinements of processes and refinements of schemas and operations satisfying certain coherence properties, basic properties of category theory imply that specifications and refinements in **CZP** conform a category.

4 Data Refinement

We have described a category of refinements that allows us to reason about the process of refining operations and strengthening state descriptions. Another mechanism for refining specifications is the so-called *data refinement* [14]. This form of refinement is achieved by adding details to the datatypes used in the specifications. In this way, specifications get closer to the data structures available in programming languages. Categorically, data refinements can be characterized by the so-called *institution representations*. Intuitively, a data refinement is a mapping between specifications that preserve basic properties. First, let us introduce the notion of institution representation, as presented in [27].

Definition 10 *(Institution representation). Let $I = \langle \mathbf{Sign}, sen, Mod, \{\models_\Sigma\}_{\Sigma \in |\mathbf{Sign}|}\rangle$ and $I' = \langle \mathbf{Sign}', sen', Mod', \{\models'_\Sigma\}_{\Sigma \in |\mathbf{Sign}'|}\rangle$ be institutions. The structure $\langle \gamma^{Sign}, \gamma^{sen}, \gamma^{Mod}\rangle : I \to I'$ is an* institution representation *if and only if:*

1. *$\gamma^{Sign} : \mathbf{Sign} \to \mathbf{Sign}'$ is a functor,*
2. *$\gamma^{sen} : sen \to sen' \circ \gamma^{Sign}$, is a natural transformation,*
3. *$\gamma^{Mod} : Mod' \circ (\gamma^{Sign})^{\mathsf{op}} \to Mod$, is a natural transformation,*

Moreover, for any $\Sigma \in|\ \mathbf{Sign}\ |$, the function $\gamma^{Sen}_\Sigma : sen(\Sigma) \to sen'(\gamma^{Sign}(\Sigma))$ and the functor $\gamma^{Mod}_\Sigma : Mod'(\gamma^{Sign}(\Sigma)) \to Mod(\Sigma)$ preserve the following satisfaction condition: for any $\alpha \in sen(\Sigma)$ and $\mathcal{M}' \in|\ Mod(\gamma^{Sign}(\Sigma))\ |$, $\mathcal{M}' \models_{\gamma^{Sign}(\Sigma)} \gamma^{Sen}_\Sigma(\alpha)$ iff $\gamma^{Mod}_\Sigma(\mathcal{M}') \models_\Sigma \alpha$.

An institution representation captures an embedding of a given logic in a richer logic. Data abstractions correspond to *endo* institutions representations, that is, they describe how a specification can be mapped into another one *within* the same formalism.

First, let us note that given a (endo) representation map such that γ^{Mod} is epi can be extended to a endofunctor between the corresponding categories of theory presentations.

Theorem 5. *Let* $\mathbf{I}$ *be an institution, and an institution representation* $abs = \langle \gamma^{Sign}, \gamma^{Sen}, \gamma^{Mod} \rangle : \mathbf{I} \to \mathbf{I}$*, with* γ^{Mod} *epi, then the mapping* $abs : \mathbf{Pres} \to \mathbf{Pres}$*, defined as follows:*

- *For any theory presentation* $\langle \Sigma, Ax \rangle$*,* $abs(\langle \Sigma, Ax \rangle) = \langle \gamma^{Sign}(\Sigma), \gamma^{Sen}_{\Sigma}(Ax) \rangle$
- *For any theory morphism* $\sigma : \langle \Sigma, Ax \rangle \to \langle \Sigma', Ax' \rangle$*,*

$$abs(\sigma) = \langle \gamma^{Sign}(\sigma), Sen(\gamma^{Sign}(\sigma)) \rangle$$

is a functor.

Proof. *The proof is straightforward by resorting to properties of institution representations, and the fact that abs is an endofunctor and* γ^{Mod} *is epi.*

Finally, the concept of data refinement can be formally defined using the notion of lax functor (homomorphisms between bicategories).

Definition 11. *Given specifications* $\mathbf{C_0}, \mathbf{C_1}$ *(subcategories of the bicategory* **Spec***) a* data refinement *is a lax functor* $a : \mathbf{C}_0 \to \mathbf{C}_1$ *composed by:*

- *A mapping between the 0-cells (theory presentations), defined by an (endo) representation map* $\langle \gamma^{Sign}, \gamma^{Sen}, \gamma^{Mod} \rangle$ *such that* γ^{Mod} *is epi.*
- *Mappings between cospans (1-cells):* $f_{S,S'} : Op_{C_0}(S, S') \to Op_{C_1}(S, S')$*. For any operation Op we call* $abs(op)$ *its corresponding operation obtained by applying the data refinement.*

As in any lax functor, mappings $f_{S,S'}$ *are subject to some coherence laws, roughly speaking, identity and composition must be preserved* [5].

Intuitively, a data refinement is composed of a mapping between specifications (that preserves properties) and a mapping between operations. In this case the natural transformation γ^{Mod} can be thought of as the usual abstraction function [14]; the requirement that such mappings be surjective (epi) is standard for abstraction mappings.

Let us now present an example of data refinement. We use the Z notation to illustrate the above defined concepts, using an example of memories and memories with cache based on that described in [17]. A memory is, as we explained before, simply a mapping from addresses to data. A cache memory is composed of two memories: one smaller memory playing the role of the cache, and a main memory. The assumption is that the cache is faster, and thus can be used to speed up memory writing and reading. In Fig. 1 we have the specification of memories with cache, and their operations. In that figure, we can observe two specifications of a memory; the arrows between *Memory* (resp. *Memory'*) and *CacheMemory* (resp. *Memory'*) are obtained by mapping the function $data : \mathbb{N} \to Data$ to a

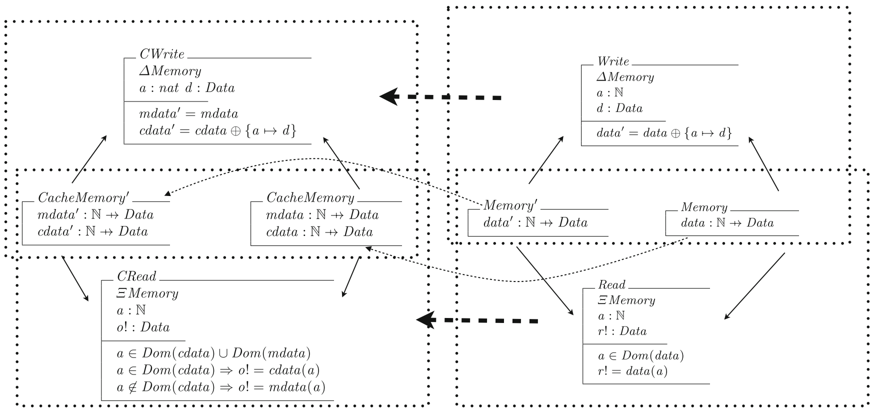


Fig. 1. An example of data refinement

pair of functions $mdata : \mathbb{N} \rightarrow Data$ and $cdata : \mathbb{N} \rightarrow Data$. The abstraction function in this case is obtained by the union of the two functions (see [17]). The mappings between the corresponding operations are represented by the big arrows between the squares.

5 Related Work and Conclusions

We have proposed an abstract, language independent, mathematical foundation for refinements. The abstract setting that we presented was developed using well established abstract notions of logical systems. Indeed, the notions that we used in this formalisation have been employed to structure concurrent system specification languages and algebraic specification languages, and other formalisms [10]; we think that one of the main benefits of this abstract framework is the possibility of combining different refinement calculi in a simple way by resorting to categorical constructions.

With respect to related work, various formalizations of refinement calculi have been previously presented. Most of these are concrete, language or formalism specific (e.g., [2,24,29]). In [2], there is a categorical treatment of refinement, but is restricted to the use of categories to capture semantic domains. In [4], a categorical framework of allegories is used to deal with program calculation, in the functional programming sense (as opposed to our case, where we consider the notion of state to be inherent to model based specification). In [25], an abstract treatment of refinement is presented, using the theory of π-institutions. However, [25] does not deal with the notions of operation or component, in the sense of component based specification, as we do in this paper. In [21], refinement is studied in comparison with composition, in the context of action-based systems; the treatment is categorical, but the approach is different from ours: [21] employs a category where objects are software components, and different arrows capture superposition and refinement between components. This work concentrates on action refinement, and does not deal with data refinement.

Unifying Theories of Programming (UTP) [16] provides a common notion of refinement for different programming paradigms, and it is used for providing the semantics of heterogeneous specification languages such as Circus [28]. It is worth noting that UTP mainly uses first-order logic and fixpoint constructions, whereas the framework described in this paper does not depend on any particular logic, it is based on the abstract notion of logical theory; thus, it can be employed to capture the notion of refinement in other settings, examples of this are specification languages using higher-order logics, infinitary logics, etc.

Finally, it is worth mentioning that there exist a broad literature on structuring algebraic specifications that may be applied to refinement as a particular case. For instance, [18] describes a categorical formulation of data refinement using lax transformations, this approach focuses on the semantics of an imperative language, even though the authors propose extensions to cope with more expressive languages. On the other hand, Institution theory has been used to provide heterogeneous specification formalisms, for instance, those described in [8, 22], although none of them particularly deal with specification refinements.

References

1. Abrial, J.-R.: The B-Book. Cambridge University Press, Cambridge (1996)
2. Back, R.J., von Wright, J.: Refinement Calculus: A Systematic Introduction. Springer, New York (1998)
3. Barr, M., Wells, C.: Category Theory for Computer Science. Centre de Recherches Mathématiques, Université de Montréal, Montreal (1999)
4. Bird, R., de Moor, O.: Algebra of Programming. Prentice-Hall, Upper Saddle River (1997)
5. Borceux, F.: Handbook of Categorical Algebra. Basic Category Theory, Encyclopedia of Mathematics and its Applications, vol. 1. Cambridge University Press, Cambridge (1994)
6. Cavalcanti, A.L.C.: A Refinement calculus for Z. Ph.D. thesis, Oxford University Computing Laboratory, Oxford, UK (1997)
7. Castro, P., Aguirre, N., Lopez Pombo, C., Maibaum, T.: Categorical foundations for structured specifications in Z. Form. Asp. Comput. **27**(5–6), 831–865 (2015)
8. Diaconescu, R.: Grothendieck institutions. Appl. Categ. Struct. **10**(4), 383–402 (2002)
9. Diaconescu, R.: Institution-Independent Model Theory. Birkhäuser Verlag, Basel (2008)
10. Fiadeiro, J.: Categories for Software Engineering. Springer, Heidelberg (2004)
11. Fiadeiro, J., Sernadas, A.: Structuring theories on consequence. In: Sannella, D., Tarlecki, A. (eds.) ADT 1987. LNCS, vol. 332, pp. 44–72. Springer, Heidelberg (1988). doi:10.1007/3-540-50325-0_3
12. Fischer, C.: Combining CSP and Z. Technical report, University of Oldenburg (1996)
13. Goguen, J., Burstall, R.: Institutions: abstract model theory for specification and programming. J. ACM **39**(1), 95–146 (1992). ACM Press

14. He, J., Hoare, C.A.R., Sanders, J.W.: Data refinement refined resume. In: Robinet, B., Wilhelm, R. (eds.) ESOP 1986. LNCS, vol. 213, pp. 187–196. Springer, Heidelberg (1986). doi:10.1007/3-540-16442-1_14
15. Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall International, Upper Saddle River (1985)
16. Hoare, C.A.R., He, J.: Unifying Theories of Programming. Prentice Hall International Series in Computer Science. Prentice-Hall, Upper Saddle River (1998)
17. Jackson, D.: Data Abstractions. Logic, Language, and Analysis. MIT Press, Cambridge (2006)
18. Johnson, M., Naumann, D., Power, J.: Category theoretic models of data refinement. Electr. Notes Theor. Comput. Sci. **225**, 21–38 (2009)
19. Jones, C.B.: Systematic Software Development Using VDM, 2nd edn. Prentice Hall, New York (1990)
20. Lack, S., Sobociński, P.: Adhesive categories. In: Walukiewicz, I. (ed.) FoSSaCS 2004. LNCS, vol. 2987, pp. 273–288. Springer, Heidelberg (2004). doi:10.1007/978-3-540-24727-2_20
21. Lopes, A., Fiadeiro, J.: Superposition: composition vs refinement of non-deterministic, action-based systems. Form. Asp. Comput. **16**(1), 5–18 (2004). Springer
22. Mossakowski, T.: Heterogeneus specification and the heterogeneous tool set. Habilitation thesis (2005)
23. Mossakowski, T., Roggenbach, M.: Structured CSP – a process algebra as an institution. In: Fiadeiro, J.L., Schobbens, P.-Y. (eds.) WADT 2006. LNCS, vol. 4409, pp. 92–110. Springer, Heidelberg (2007). doi:10.1007/978-3-540-71998-4_6
24. Morgan, C.C.: Programming from Specifications. Prentice-Hall, Upper Saddle River (1990)
25. Rodrigues, C., Martins, M., Madeira, A., Barbosa, L.: Refinement by interpretation in π-institutions. In: Proceedings of the 15th International Refinement Workshop (2011)
26. Spivey, J.M.: The Z Notation: A Reference Manual. Prentice Hall, Upper Saddle River (1992)
27. Tarlecki, A.: Moving between logical systems. In: Haveraaen, M., Owe, O., Dahl, O.-J. (eds.) ADT/COMPASS -1995. LNCS, vol. 1130, pp. 478–502. Springer, Heidelberg (1996). doi:10.1007/3-540-61629-2_59
28. Woodcock, J., Cavalcanti, A.L.C.: The semantics of *Circus*. In: Bert, D., Bowen, J.P., Henson, M.C., Robinson, K. (eds.) ZB 2002. LNCS, vol. 2272, pp. 184–203. Springer, Heidelberg (2002). doi:10.1007/3-540-45648-1_10
29. Woodcock, J., Davies, J.: Using Z: Specification, Refinement, and Proof. Prentice Hall, Upper Saddle River (1996)