Under consideration for publication in Formal Aspects of Computing

Categorical Foundations for Structured Specifications in Z

Pablo F. Castro[1,3] and Nazareno Aguirre[1,3] and Carlos L. Pombo[2,3] and T. S. E. Maibaum[4]

[1]Departamento de Computación, FCEFQyN, Universidad Nacional de Río Cuarto, Río Cuarto, Córdoba, Argentina.
[2]Departamento de Computación, FCEyN, Universidad de Buenos Aires, Buenos Aires, Argentina.
[3]Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Argentina
[4]Department of Computing & Software, McMaster University, Hamilton (ON), Canada.

Abstract. In this paper we present a formalization of the Z notation and its structuring mechanisms. One of the main features of our formal framework, based on category theory and the theory of institutions, is that it enables us to provide an abstract view of Z and its related concepts. We show that the main structuring mechanisms of Z are captured smoothly by categorical constructions. In particular, we provide a straightforward and clear semantics for promotion, a powerful structuring technique that is often not presented as part of the schema calculus. Here we show that promotion is already an operation over schemas (and more generally over specifications), that allows one to promote schemas that operate on a local notion of state to operate on a subsuming global state, and in particular can be used to conveniently define large specifications from collections of simpler ones. Moreover, our proposed formalization facilitates the combination of Z with other notations in order to produce heterogeneous specifications, i.e., specifications that are obtained by using various different mathematical formalisms. Thus, our abstract and precise formulation of Z is useful for relating this notation with other formal languages used by the formal methods community. We illustrate this by means of a known combination of formal languages, namely the combination of Z with CSP.

Keywords: Z Notation; System Specification; System Verification; Category Theory; Heterogeneous Specifications

1. Introduction

Formal specifications are descriptions of software systems given in terms of mathematical notations, that enable one to rigorously reason about system properties. Given the complexity both of software systems and the problems they solve, and how crucial it is in many contexts to guarantee that software performs its functionalities correctly, formal specification can be an important aid in requirements analysis, software design and implementation.

Correspondence and offprint requests to: Pablo F. Castro, Universidad Nacional de Río Cuarto, Ruta Nac. No. 36 Km. 601, Río Cuarto (5800), Córdoba, Argentina. e-mail: pcastro@dc.exa.unrc.edu.ar

Usually, software tends to be large and complex. So, an important concern in software development is *scalability*. Formal specifications of such software are typically also large, due to the level of detail that formality often requires, technically involved and hard to understand. A mechanism to achieve scalability, considered crucial by software engineers since the seminal work of Parnas [Par72, Par85], is software modularization. In the context of formal specification, modularization is realized via diverse structuring mechanisms, provided by formal notations with the aim of simplifying the task of describing and reasoning about specifications: large system descriptions are decomposed into several interacting modular specifications, enabling in this way reasoning compositionally over complex problems.

One of the formal languages that heavily relies on the concept of module is the Z notation [Nic95], a well-known formal specification language used for describing and reasoning about systems. Indeed, one of the main characteristics of Z is that it provides syntactical constructions to describe modules, called *schemas*, together with several structuring mechanisms for building large systems from schemas, usually called *schema operators*. These mechanisms, or operators, allow for the combination (and generalization) of schemas, to build complex specifications out of simpler parts. Some standard schema operators used in Z specifications are *(i) schema inclusion*, which facilitates the reuse of specifications by allowing the inclusion of one schema into another; *(ii) schema conjunction and disjunction*, which enable the combination of schemas by combining their constituent constraints using logical connectives; *(iii) schema quantification*, which enables one to generalize schemas over a set of values; and *(iv) schema promotion*, an operation that allows one to promote schemas that operate on "local" state to operate on a subsuming "global" state; in particular, promotion can be used to conveniently define large specifications from collections of simpler ones.

Promotion is one of the most powerful structuring mechanisms associated with Z. This technique allows one to map a local state into a global one. A particularly recurring use of promotion is in the definition of the global state as being composed of collections of local states; when these local states represent the states of module instances, one can achieve a construction similar to the notion of objects in object oriented programming [Mey00]. One of the contributions of this paper is the provision of a mathematical foundation for Z that provides a rigorous semantics for schema operators and its structuring mechanisms, including promotion. As we discuss in section 6, most of the proposed approaches to provide semantics to the Z notation do not reflect, or appropriately treat, the structure of Z specifications in the semantics. Our proposed characterization provides appropriate formalizations for schema operators in terms of categorical constructions, and in particular captures promotion as a straightforward construction. As we will show, this has important benefits, in particular because it allows us to formally deal, semantically, with promoted specifications.

Another important element employed for dealing with software complexity is *separation of concerns*. In fact, a common practice in software engineering is to model different aspects of a software system using different notations, each of which is better suited for a particular aspect. For instance, the static structure of a system may be captured using a given notation (e.g., class diagrams), while the run time behaviour of the system may be described in a different setting (e.g., using state machines). As argued in [HJ98, MML07, BSM04, FKNG92, Lan09], a complete description of a complex artifact of software might only be achieved by resorting to several orthogonal software notations, where each aspect of the system is modeled by choosing a suitable formalism. This philosophy is followed, for example, by the *Unified Modeling Language* [BSM04] (and related languages) where class diagrams are used for describing the architecture of the system, sequence diagrams and state charts are used for modeling the dynamic interaction of the system's components and object diagrams are employed to capture run time scenarios of the system execution.

Formal specification, as all forms of software specification, can greatly profit from this "multiple views" approach, and therefore heterogeneous formal notations for specifying, verifying and validating software specifications are receiving increasing attention. Intuitively, a heterogeneous specification is a mathematical model of a system that is built by resorting to several (and perhaps highly different) logical or formal languages. Each of these formal notations is used for describing some important aspect, or view, of the system. The combination of various formal languages allows software designs to gain expressivity and clarity. From a mathematical point of view, the combination of different notations introduces some technical issues that are hard to deal with. In particular, finding a mathematical framework expressive enough to harmonize the conflicts or inconsistencies that arise when different "views" are combined is not always a trivial task. The categorical framework that we propose in this work enjoys the benefit of providing such an abstract, and general setting, that can be used to combine Z with other languages and notations, thus making possible the use of the Z notation in a heterogeneous setting. This is the other main contribution of our proposal.

This work extends that presented in [CAPM12], where a basic categorical framework to capture schema conjunction and promotion is introduced. We now rework that framework to obtain a more expressive setting,

Player
$owns : \mathbb{P}\,\mathbb{N}$

Leave
$\Delta Player$
$t : \mathbb{N}$

$t \in owns$
$owns' = owns \setminus \{t\}$

Settle
$\Delta Player$
$t : \mathbb{N}$

$t \notin owns$
$owns' = owns \cup \{t\}$

Fig. 1. Examples of Z Schemas

by extending some definitions to be able to capture other schema operators such as schema disjunction and schema quantification. In addition, we study in detail the properties of promotion, and demonstrate that the introduced framework naturally supports heterogeneous specifications. We illustrate this with an example that involves process algebras.

The paper is structured as follows. Section 2 provides a short introduction to Z and the basic categorical concepts used throughout this text. Section 3 describes the categorical formalization of Z and its structuring mechanisms; in section 4, we show how promotion can be captured by resorting to standard constructions from the theory of institutions. In section 5 we show that the categorical framework provided in this paper can be used for combining Z with other formalisms to obtain heterogeneous specifications. We illustrate this by providing a formal semantics for a language that combines Z and CSP, that we refer to as CZP, and that is similar to Z-CSP [Fis97]. Finally, we discuss related work and present some final remarks. For the sake of readibility some technical details about the definitions given in Section 2 are gathered in the appendix.

2. Preliminaries

In this section, we introduce some basic concepts that are necessary throughout the paper. These include a description of the Z notation and its main features, and basic definitions regarding category theory and the theory of institutions.

2.1. The Z Notation

Z is a formal notation based on mathematical logic and set theory. It is often regarded as being *model based*, since specifications in the language describe systems behaviour via *models*, typically involving the description of data domains and operations on these domains [Woo96]. Such models are expressed in terms of well defined types, including a rich set of provided types, such as the typical numerical domains (technically, all defined in terms of the built-in type $\mathbb{Z}$), sets, sequences, tuples, relations and functions, etc. Z specifications are structured via the notion of *schema* [Woo96]. Essentially, a schema defines a set of typed variables, whose values might be constrained; so a schema has a *declaration* section, and a *constraint* (or predicate) section. This extremely simple notion is powerful and convenient for defining data domains and operations on these, as formal models of systems. As a first example, suppose that we need to specify a game similar to Risk [Ris63], consisting of players whose goal is to conquer territories on a map. For simplicity, let us suppose that territories are labelled by natural numbers, identifying each territory. We might start by defining players, indicating the territories they own. In Z, this is achieved by the schemas shown in Fig. 1. These are very simple schemas that have an empty predicate part (no special constraints on the variables). Basic operations for a player are settling in a territory, and leaving an occupied territory. In Z, operations are also captured by schemas; schemas characterising the settle and leave operations are shown in the same figure. In these schemas, $\Delta Player$ indicates that two copies of the schema *Player* are incorporated into *Settle* and *Leave*, one exact copy of *Player* and the other with its variables renamed by priming. This is done in order to capture the effect of settling (resp. leaving) as a relation between "pre" states of the player (the unprimed variables) and the "post" states of the player, resulting from settling on (resp. leaving from) a territory. Additional variables, in this case representing parameters of the operations, are incorporated and constrained in the

$$\begin{array}{|l}\multicolumn{1}{l}{\mathit{Game}}\\ \hline ps : \mathbb{P}\,\mathit{Player}\\ ts : \mathbb{P}\,\mathbb{N}\\ \hline ts \neq \emptyset\\ \forall\, p : ps \bullet p.owns \subseteq ts\\ \forall\, p_1, p_2 : ps \bullet p_1 \neq p_2 \Rightarrow p_1.owns \cap p_2.owns = \emptyset\\ \hline\end{array}$$

$$\begin{array}{|l}\multicolumn{1}{l}{\mathit{PromotePlayer}}\\ \hline \Delta \mathit{Game}\\ \Delta \mathit{Player}\\ p : \mathit{Player}\\ \hline p = \theta\mathit{Player} \wedge p \in ps\\ ts = ts'\\ ps' = ps \setminus \{p\} \cup \{\theta\mathit{Player}'\}\\ \hline\end{array}$$

Fig. 2. Examples of Promotion

predicate part of the schemas. When defining a schema in terms of another one, constraints from the used schema are made part of the constraints of the using schema; for instance, constraints from *Player* and *Player'* (coming from Δ*Player*) are part of the *Settle* schema (although in this case no actual constraints are incorporated, because the used schemas had no constraints). According to the denotational semantics of Z, a model for a schema is an assignment, that provides values in the corresponding types for the variables in the schema, and satisfies the predicate part of the schema [Spi88]. That is, a model provides actual values for the variables in a schema. Notice for instance that, for the case of *Player*, all possible models of the schema capture the "state space" for the player.

Z also features schema structuring operations, that is, operations that enable one to define schemas based on other existing schemas. A rather simple one is *schema composition*. Suppose that we would like to define an operation to capture the situation in which a player exchanges one territory for another one, i.e., it leaves a territory and settles in another one. Such an operation can be defined using schemas *Leave* and *Settle*, via a simple composition:

$$\mathit{Exchange} \mathrel{\widehat=} \mathit{Leave}[t_1/t] \fatsemi \mathit{Settle}[t_2/t]$$

This composition (slightly complicated with the renamings necessary for the composition to distinguish the t variables in the two schemas) captures the state change produced by applying the second operation to the state resulting from the application of the first operation.

Promotion is another structuring mechanism of Z. It enables one to *promote* definitions given in terms of "local states", to definitions of a "global state", often composed of various instances of the local state [Woo96]. As an example, suppose that we define the state of game, using our previously defined *Player* schema, as shown in Fig. 2. This schema explicitly indicates who are the players of the game (*ps*), and the territories composing the map (*ts*); it also constrains the valid states of the game to nonempty sets of territories, and prevents players from sharing the occupation of a territory. We have already defined game related operations *Settle* and *Leave*, but we have done so for *Player*. We would like to be able to *promote* those "local" operations to the "global" state characterised by *Game*, instead of having to redevelop them as operations on *Game*. In order to do so, one needs to define a *promotion schema*, i.e., a schema relating the local and global states. Notice that this schema indicates how a state change of a single player is embedded into a state change for the global state of the game. Note also the Θ notation, in this case Θ*Player* is used for referring to the state of *Player* before the execution of the action, in a similar way one might use $\Theta \mathit{Player}'$ to denote the state after. Now, one can promote the *Settle* operation to the system level, as follows:

$$\mathit{GameSettle} \mathrel{\widehat=} \exists\, \Delta \mathit{Player} \bullet \mathit{Settle} \wedge \mathit{PromotePlayer}$$

The existential quantification in this definition has the purpose of hiding the "local state", which by the restrictions in the *PromotePlayer* schema is already embedded into the state of the game. That makes *GameSettle* an operation exclusively on the state of the game.

2.2. Logic, Institutions and Category Theory

In this section we recall some useful definitions of logic, category theory and institution theory. For a detailed introduction to these topics the reader is referred to [End01, Dia08, Mac98, Fia04].

First, let us start providing the basic definitions of propositional logic. A *propositional vocabulary*, or *propositional signature*, is an enumerable collection of symbols acting as propositional variables: $\Sigma = \{p_0, p_1, p_2, \dots\}$; propositional formulas are defined as usual; propositional variables are formulae and boolean operators ($\neg$, $\vee$, $\wedge$ and $\Rightarrow$) combining formulae are formulae. Semantics for propositional formulae is given by functions $v : \Sigma \to \{\mathit{true}, \mathit{false}\}$ called *assignments*, which map propositional symbols to truth values, boolean operators are interpreted in a logical way as usual. A *translation* between two propositional signatures $\Sigma = \{p_0, p_1, p_2, \dots\}$ and $\Sigma' = \{p'_0, p'_1, p'_2, \dots\}$ is a function $\tau : \Sigma \to \Sigma'$. Given a translation $\tau : \Sigma \to \Sigma'$ and an assignment $v : \Sigma' \to \{\mathit{true}, \mathit{false}\}$, we can define an assignment $v|_\tau : \Sigma \to \{\mathit{true}, \mathit{false}\}$ as follows: $v|_\tau(p_i) = v(\tau(p_i))$; this is called the *$\tau$-reduct* of v.

First-order logic enriches propositional logic with quantifiers, variables ranging over individuals of certain sorts, function symbols and relation symbols. A first-order signature is a tuple $\Sigma = \langle F, R, \iota \rangle$ where F is a set of function symbols and R is a collection of relation symbols, and $\iota : F \cup R \to \mathbb{N}$ is a function returning the arities of the function and relation symbols. Constants will be represented by function symbols whose arity is 0. First-order terms are obtained as the smallest set S of syntactic objects such that 0-ary functions are in S and for every n-ary function symbol f and n-tuple $t_1, \dots, t_n$ in S^n, $f(t_1, \dots, t_n)$ is in S; first-order formulae is obtained as the smallest set S' such that for every n-ary relation symbol r and n-tuple $t_1, \dots, t_n$ of terms, $r(t_1, \dots, t_n)$ is in S', the boolean combination of formulae in S' is in S' and for any variable symbol x and formula α in S', $\exists\, x \bullet \alpha$ is in S'. An *interpretation* (or *structure*) for a first-order signature Σ is a structure: $M = \langle S, I \rangle$ where S is a non-empty set, and I is a function mapping each n-ary function symbol f to a n-ary function $I(f) : S \times \dots \times S \to S$ and each n-ary relation symbol r to a n-ary relation $I(r) : S \times \dots \times S$. Given a first-order logic formula α, a model of it is an interpretation M together with an assignment of elements of the set to variable symbols v such that $\langle M, v \rangle$ satisfies the formula, denoted as $\langle M, v \rangle \models \alpha$. This is naturally extended to sets of formulae by requiring the pair to be a model of all the formulae in the set. A *translation* between two first-order signatures $\Sigma = \langle F, R, \iota \rangle$ and $\Sigma' = \langle F', R', \iota' \rangle$ is a function $\tau : R \cup F \to R' \cup F'$ such that n-ary functions (resp. relations) of Σ are mapped to n-ary function (resp. relations) of Σ'. Given a translation $\tau : \Sigma \to \Sigma'$ and a model $M' = \langle \langle S', I' \rangle, v' \rangle$ of Σ' we can define a model $M'|_\tau = \langle \langle S'|_\tau, I'|_\tau \rangle, v'|_\tau \rangle$ of Σ as follows: $S|_\tau = S$, $I'|_\tau(f) = I'(\tau(f))$ for $f \in R \cup F$, $v'|_\tau = v'$; $M'|_\tau$ is called a τ-reduct of M', or simply a reduct of M' when τ is clear from the context.

A category is a mathematical structure composed of two collections: the collection of objects: $a, b, c, \dots$ and the collection of arrows (or morphisms): $f, g, h, \dots$ between them. An arrow has a domain and a codomain, and we write $f : a \to b$ to indicate that a is the domain of f and b is the codomain of f. We have two basic operations involving arrows: the *identity*, that given an object a produces 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 $f\,;\, g : a \to c$. Identity arrows satisfy: $f\,;\, id_b = f$ and $id_a\,;\, f = f$, for every $f : a \to b$. Also, for any arrows $f : a \to b$, $g : b \to c$, and $h : c \to d$ $f\,;(g\,;\, h) = (f\,;\, g)\,;\, h$. Sometimes, when the arrows are functions, instead of using the operator $;$ we use the symbol $\circ$ to write the composition of arrows in applicative order, e.g., $g \circ f : a \to c$. Given a category $\mathbf{C}$, its collection of objects is denoted by $|\mathbf{C}|$, and its collection of arrows by $||\mathbf{C}||$. The most natural example of a category is **Set**, made up of the collection of sets and the collection of functions between sets. The *dual category* of $\mathbf{C}$, denoted $\mathbf{C}^{op}$, has the same objects as $\mathbf{C}$, and its arrows are obtained by inverting the domain and codomain of the arrows of $\mathbf{C}$, formally: $||\mathbf{C}^{op}|| = \{f^{op} : b \to a \mid f : a \to b \in ||\mathbf{C}||\}$. We say that an arrow $f : a \to b$ is iso if there exists an arrow $g : b \to a$ such that $f \circ g = id_b$ and $g \circ f = id_a$; in this case we write $a \cong b$, or we simply say that a and b are isomorphic.

A *functor* is essentially a homomorphism between categories that maps objects to objects and arrows to arrows preserving identity and composition. A simple example of functor is the powerset functor of set theory (written $\mathbb{P} : \mathbf{Set} \to \mathbf{Set}$), which maps any set S to its powerset $\mathbb{P}(S)$ and maps any function $f : S \to S'$ to a function $\mathbb{P}(f) : \mathbb{P}(S) \to \mathbb{P}(S')$ defined as: $\mathbb{P}(f)(A) = \{f(x) \mid x \in A\}$. Another useful example of functor is the following, given any category $\mathbf{A}$ we denote by $Hom_{\mathbf{A}}(x, y)$ the set of arrows between x and y in $\mathbf{A}$; furthermore $Hom_{\mathbf{A}}(x, -) : \mathbf{A} \to \mathbf{Set}$ is a functor, that maps each object x to its sets of arrows, and each arrow $f : y \to z$ to a function $Hom(x,f) : Hom(x, y) \to Hom(x, z)$ mapping arrows to arrows, defined as $Hom(x,f)(g) = f \circ g$. Similarly, a *bifunctor* $F : \mathbf{A} \times \mathbf{B} \to \mathbf{C}$ is a functor from the product category $\mathbf{A} \times \mathbf{B}$ (with objects pairs of objects and arrows pair of arrows) to another category $\mathbf{C}$.

A *natural transformation* is a morphism between functors; given two functors $F, G : \mathbf{A} \to \mathbf{B}$, a natural transformation $\eta : F \stackrel{\cdot}{\to} G$ from F to G is a mapping that assigns to each object x of A an arrow $\eta_x : F(x) \to G(x)$ (called its components), such that for every arrow $f : x \to y$ in A we have $\eta_y \circ F(f) = G(f) \circ \eta_x$ (this is called the *natural* condition). Natural transformations will be denoted by the use of the symbols $\stackrel{\cdot}{\to}$, and, when useful, we identify a natural transformation with the collection of its components.

The concept of *adjoint* arises almost everywhere in category theory (and in mathematics in general). It can be thought of as a generalization of the notion of Galois connection. The simplest definition of the concept of adjoint is as follows. Given functors $F : \mathbf{A} \to \mathbf{B}$ and $G : \mathbf{B} \to \mathbf{A}$, G is said to be *right adjoint of* F when there is a bijection between $Hom_{\mathbf{A}}(F(x), y) \cong Hom_{\mathbf{B}}(x, G(y))$ which is natural in x and y. In this case, we simply say that F and G are adjoints.

A monoidal category is a category $\mathbf{C}$ plus a functor $\otimes : \mathbf{C} \times \mathbf{C} \to \mathbf{C}$ and an object 1 such that they satisfy, up to isomorphism, the diagrams corresponding to associativity and identity along with some coherence conditions (the so-called triangle and pentagon diagrams, see the appendix).

The notion of bicategory [Ber67] will be important when defining schemas and operations of Z . Bicategories are a generalization of categories, where we have an additional notion of arrow (between arrows) that admits two kinds of compositions; formally, a *bicategory* $\mathbf{V}$ consists of:

- a class of objects $|\mathbf{V}|$ (also called 0-cells),
- for every $x, y \in |\mathbf{V}|$, $\mathbf{V}(x, y)$ is a category whose objects are called arrows (or 1-cells) and whose morphisms are called 2-cells, composition in $\mathbf{V}(x, y)$ is called **vertical** composition.
- for every object $a \in |\mathbf{V}|$, a 1-cell $\mathbf{1}_a : a \to a \in |\mathbf{V}(a, a)|$, which is called the unit of a, and
- a bifunctor $;_{x,y,z} : \mathbf{V}(x, y) \times \mathbf{V}(y, z) \to \mathbf{V}(x, z)$, for every $x, y, z \in |\mathbf{V}|$, this is called the horizontal composition operator.

The bifunctor ; (we omit the subindexes when they are clear from context) must be associative up to isomorphism, see the appendix for the coherence conditions. We will make use the well-known constructions of products, coproducts, limits and colimits. Here, we only introduce the notion of coproduct for the other ones the reader is referred to [Mac98]. Consider a pair of arrows $f : a \to a + b$ and $f : b \to a + b$, if for every other pair of arrows $f' : a \to c$ and $g' : a \to c$ we have that there is a unique arrow $u : a + b \to c$ such that $f' = f \,;\, u$ and $g' = g \,;\, u$, then we say that $\langle f, g\rangle$ is a coproduct and $a + b$ is the coproduct object; the existence (and uniqueness) of u is often referred as the *universal property* of coproducts, and similarly for the other constructions.

Another useful concept we will use throughout this text is that of institutions. An institution is an abstract formulation of the model theory of a logical system.

Definition 2.1. ([GB92]) An *institution* is a structure of the form $\langle \mathsf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \{\models^{\Sigma}\}_{\Sigma \in |\mathsf{Sign}|}\rangle$ satisfying the following conditions:

- Sign is a category of signatures,
- $\mathbf{Sen} : \mathsf{Sign} \to \mathsf{Set}$ is a functor. Let $\Sigma \in |\ \mathsf{Sign}\ |$, then $\mathbf{Sen}(\Sigma)$ returns the set of Σ-sentences,
- $\mathbf{Mod} : \mathsf{Sign}^{\mathsf{op}} \to \mathsf{Cat}$ is a functor. Let $\Sigma \in |\ \mathsf{Sign}\ |$, then $\mathbf{Mod}(\Sigma)$ returns the category of Σ-models,
- $\{\models^{\Sigma}\}_{\Sigma \in |\mathsf{Sign}|}$, where $\models^{\Sigma} \subseteq |\ \mathbf{Mod}(\Sigma)\ | \times \mathbf{Sen}(\Sigma)$, is a family of binary relations,

and for any signature morphism $\sigma : \Sigma \to \Sigma'$, Σ-sentence $\phi \in \mathbf{Sen}(\Sigma)$ and Σ'-model $\mathcal{M}' \in |\ \mathbf{Mod}(\Sigma)\ |$, the following $\models$-invariance (or satisfaction) condition holds:

$$\mathcal{M}' \models^{\Sigma'} \mathbf{Sen}(\sigma)(\phi) \quad \text{iff} \quad \mathbf{Mod}(\sigma^{\mathsf{op}})(\mathcal{M}') \models^{\Sigma} \phi \ .$$

Note that in the $\models$-invariance condition, the expression $\mathbf{Sen}(\sigma)(\phi)$ must be associated to the left (i.e., $(\mathbf{Sen}(\sigma))(\phi)$), also note that $\mathbf{Sen}$ is a functor (which maps objects to objects and arrows to arrows) and $\sigma : \Sigma \to \Sigma'$ is a translation between signature, thus $\mathbf{Sen}(\sigma) : \mathbf{Sen}(\Sigma) \to \mathbf{Sen}(\Sigma')$, defined as the homomorphic extension σ to the structure of formulae is a function translating sentences of Σ to sentences of Σ'.

Let us give a simple example to clarify this definition. Consider propositional logic (PL from now on), we can define it as an institution as follows. The syntax of PL is given by a category Sign_{PL} which has as objects sets of propositional variables, and translations between them are its morphisms. On the other hand, the functor $\mathbf{Sen}_{PL}$, maps each sets of propositional variables to the collection of propositional formulae that can be constructed from them using the standard boolean operators, also $\mathbf{Sen}$ maps each translation σ of propositional variables to a translation $\mathbf{Sen}(\sigma)$ between formulae on the signature in the domain of σ to formulae on the signature in the codomain of σ by replacing propositional symbols in formulas in the ways prescribed by σ. The semantics of the logic is given by a functor $\mathbf{Mod}$ which maps each signature to its class of models, assignments of truth values to propositional variable symbols; here it is important to note that translations of propositional signatures are mapped to reducts (in the traditional sense of model theory [CK90]), thus a translation $\sigma : \Sigma \to \Sigma'$ is mapped to a reduct operation $\mathbf{Mod}(\sigma) : \mathbf{Mod}(\Sigma') \to \mathbf{Mod}(\Sigma)$, turning assignments of truth values to variable symbols in Σ' to an assignments of truth values to variable symbols of Σ, by simply forgetting the variables that are not in the codomain of σ, and mapping the other in the opposite direction that the one prescribed by σ. One must note that reducts go in the opposite direction w.r.t. signature translation, for this reason the functor **Mod** goes from Sign^{op} (the dual of Sign) to **Cat**.

Furthermore, given two different institutions we can define different kinds of morphisms between them. The following definition is taken from [Tar95], and formalizes the notion of institution representation (also called comorphisms).

Definition 2.2. ([Tar95]) Let $\mathbb{I} = \langle \mathsf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \{\models_\Sigma\}_{\Sigma \in |\mathsf{Sign}|} \rangle$ and $\mathbb{I}' = \langle \mathsf{Sign}', \mathbf{Sen}', \mathbf{Mod}', \{\models'_\Sigma\}_{\Sigma \in |\mathsf{Sign}'|} \rangle$ be institutions. Then, $\langle \gamma^{Sign}, \gamma^{Sen}, \gamma^{Mod} \rangle : I \to I'$ is an *institution representation* if and only if:

- $\gamma^{Sign} : \mathsf{Sign} \to \mathsf{Sign}'$ is a functor,
- $\gamma^{Sen} : \mathbf{Sen} \dot{\to} \gamma^{Sign} \,;\, \mathbf{Sen}'$, is a natural transformation,
- $\gamma^{Mod} : (\gamma^{Sign})^{\mathsf{op}} \,;\, \mathbf{Mod}' \dot{\to} \mathbf{Mod}$, is a natural transformation,

such that for any $\Sigma \in | \mathsf{Sign} |$, the function $\gamma^{Sen}_\Sigma : \mathbf{Sen}(\Sigma) \to \mathbf{Sen}'(\gamma^{Sign}(\Sigma))$ and the functor $\gamma^{Mod}_\Sigma : \mathbf{Mod}'(\gamma^{Sign}(\Sigma)) \to \mathbf{Mod}(\Sigma)$ preserve the following *satisfaction condition*: for any $\alpha \in \mathbf{Sen}(\Sigma)$ and $\mathcal{M}' \in | \mathbf{Mod}(\gamma^{Sign}(\Sigma)) |$,

$$\mathcal{M}' \models^{\gamma^{Sign}(\Sigma)} \gamma^{Sen}_\Sigma(\alpha) \;\text{ iff }\; \gamma^{Mod}_\Sigma(\mathcal{M}') \models^\Sigma \alpha \ .$$

An institution representation $\gamma : I \to I'$ expresses how a “poorer” logical system is encoded in another “richer” one. This is done by:

- constructing, for a given I-signature Σ, an I'-signature into which Σ can be interpreted,
- translating, for a given I-signature Σ, the set of Σ-sentences into the corresponding I'-sentences,
- obtaining, for a given I-signature Σ, the category of Σ-models from the corresponding category of Σ'-models.

The direction of the arrows shows how the whole of I is represented by some parts of I'. Institution representations enjoy some interesting properties. For instance, logical consequence is preserved, and, under some conditions, logical consequence is preserved in a conservative way. Roughly speaking, an institution representation embeds one logical system into another.

As above, a simple example is useful to clarify the concepts, consider the institution of first-order logic (named FOL), where Sign_{FOL} is the category whose objects are first-order signatures (i.e., collection of constant names, function symbols and relation symbols), and whose arrows are translations of first-order symbols. The functor **Sen** is defined in the usual way, by defining the set of first-order formulae over a given signature; the functor **Mod** maps FOL signatures to first-order models (as described in previous paragraphs), **Mod** maps symbol translation to first-order reducts. Now, we can define an institution representation between PL and FOL, as follows: $\langle \gamma^{Sign}, \gamma^{Sen}, \gamma^{mod} \rangle$ where γ^{Sign} maps a set of propositional variables (a propositional signature) to a set of nulary relations (first-order signature), γ^{Sen} maps propositional formulas to first-order formulas (a direct conversion) and γ^{Mod} extracts from a first-order structure a propositional assignment, by setting a proposition true iff the corresponding nulary relation holds. Summing up, this institutions representation shows how propositional logics can be embedded into first-order logic. A large collection of examples can be found in [Tar95].

3. A Categorical View of Z

In this section we provide a basic characterization of Z constructs. As we show below, this categorical framework allows us to give a clear semantics for Z , including the schema calculus, which is one of the most important aspects of this formal notation. All the operations over schemas: conjunction, disjunction, negation, quantification, and promotion, together with related constructions (schema inclusion, Δ and Ξ notation) have their categorical counterpart in this setting in a general and straightforward way. Moreover, the concept of action and its associated notions are also formally captured in the framework proposed below. One of the main benefits of this categorical foundation for Z is that it enables the combination of Z with other formal settings; an example of this is shown in Section 5. This is based on the fact that most formal languages used in computer science can be captured as institutions, this enables the combination of these formalisms with the categorical presentation of Z given below. The combination of different institutions is a topic that have been extensively studied over the past few years within the context of heterogeneous specifications [MTP97].

First, we introduce a category of signatures (named **Zign**); this basic category is used as a mathematical base to build other, more structured, categories. In this mathematical framework, Z schemas are captured as axiomatic theories, as discussed above. This agrees with the original semantics of Z given in [Spi84]. As described below, the plain category of theories is not powerful enough to capture the schema calculus used in Z , we show that a more expressive framework can be obtained by using monoidal categories and related constructions.

3.1. The Basic Categories

The first notions we need to introduce are those of type and typed variable.

Definition 3.1. Given a set $T = \{t_0, t_1, \dots\}$ of basic types, the set of types over T is defined as follows:

- Any $t_i \in T$ is a type.
- $\mathbb{Z}$ is a type.
- If t is a type, then $\mathbb{P}(t)$ is a type.
- If t, t' are types, then $t \times t'$ is a type.

A typed variable is a pair $\langle v, t\rangle$, where v is variable name and t is a type name. This is usually denoted by $v{:}t$. We use strings over the alphabet $\{a, b, c, \dots\}$ for both variables and variable types, we also consider any additional symbols that may be useful in these sets, e.g., symbols with superscripts.

It is worth noting that we will be able to introduce other useful types (e.g. $\rightarrow$ and $\nrightarrow$) by using these basic types (as done in Z). First, we start defining the concept of signature, that is, a collection of variables and types.

Definition 3.2. A signature is a tuple $\langle N, T, V\rangle$, where: N is the name of the signature, T is a set of basic types and $V = \{v_0{:}t_0, v_1{:}t_1, \dots\}$ is a collection of typed variables, where $t_i \in T$ for all $v_i{:}t_i \in V$.

Given a signature $\Sigma = \langle N, T, V\rangle$, we denote by $Types(\Sigma)$ the second projection of Σ, and we use $Var(\Sigma)$ to name the third projection of Σ. It is worth noting that we allow for repetition of variable names (with different types) in signatures. This is not the case in Z , where schema signatures cannot include repetition of variables (with different types). Thus the collection of signatures, as defined above, includes Z signatures but also provides a generalisation of them. This flexibility allows us to provide a simple formalisation of the so-called schema calculus.

On the other hand, morphisms between signatures are mappings between variables that preserve types. Signature names only serve to decorate, and make clearer, specifications; for the sake of simplicity, and when no confusions arise, we avoid writing the name of signatures when they are presented as tuples.

Definition 3.3. Given two signatures $\Sigma = \langle T, V\rangle$ and $\Sigma' = \langle T', V'\rangle$, a morphism $\sigma : \Sigma \rightarrow \Sigma'$, is a function $\sigma : V \rightarrow V'$ such that:

- $T \subseteq T'$,
- If $v{:}t \in V$, then $\sigma(v){:}t \in V'$.

When useful, we write $v{:}t \in \Sigma$ if $v{:}t$ is a variable in Σ. We also write $\sigma : \Sigma \hookrightarrow \Sigma'$ to indicate that σ is an inclusion, i.e., a function that maps each symbol to itself. Examples of signature morphisms are symbol substitutions (renaming variables in a signature), and embeddings of a signature into another one. It is worth pointing out that types are preserved by morphisms. Type renaming will not be needed in the framework described in this paper. It is direct to see that the set of signatures as objects and their morphisms as arrows constitute the category **Zign**.

Theorem 3.1. **Zign**, with the set of signatures as objects and the set of signature morphisms as arrows, is a category.

Theorem 3.2. **Zign** is finitely cocomplete.

Proof. This is equivalent to proving that it has coproducts, coequalizers and an initial object. For proving the existence of coproducts, consider the signatures $\Sigma_0 = \langle T_0, V_0\rangle$ and $\Sigma_1 = \langle T_1, V_1\rangle$. Then we define:

$$\Sigma_0 + \Sigma_1 = \langle T_0 \cup T_1, \{v0{:}t \mid v{:}t \in V_0\} \cup \{v1{:}t \mid v{:}t \in V_1\}\rangle$$

It is simple to observe that we can define injections $i : \Sigma_0 \to \Sigma_0 + \Sigma_1$ and $j : \Sigma_1 \to \Sigma_0 + \Sigma_1$ such that the corresponding diagram commutes, and the universal property holds. Thus, coproduces of arbitrary pair of signatures exist. The existence of coequalizers is proved by considering morphisms $f, g : \Sigma_0 \to \Sigma_1$, where $\Sigma_i = \langle T_i, V_i\rangle$ and R the smallest equivalence relation over V_1 such that $f(v{:}t)Rg(v{:}t)$ (note that $f(v{:}t)$ and $g(v{:}t)$ have the same type). Let us use V_1/R to refer to the quotient set. There is a selection function $s : V_1/R \to V_1$ that chooses one representative for each equivalence class. Therefore, let us define the following signature:

$$\Sigma = \langle T_1, \{s([x{:}t]) \mid [x{:}t] \in V_1/R\}\rangle$$

where $[x{:}t]$ is the equivalence class of $x{:}t$, and then we have a morphism $\sigma : \Sigma_1 \to \Sigma$ that maps each element to the representative of its equivalence class. Finally it is easy to prove that $\langle 0, \emptyset, \emptyset\rangle$ is the initial object of **Zign**. Thus **Zign** is finitely cocomplete. □

This result implies that we can use the colimit construction on arbitrary finite diagrams to put specifications together; this is a common technique used in categorical specifications [GB92, Fia04].

Furthermore, we can provide a generalised version of intersection for signatures.

Definition 3.4. Given signatures $\Sigma_i = \langle T_i, V_i\rangle$ (for $i = 0, 1$), their intersection is defined as follows:

$$\Sigma_0 \cap \Sigma_1 \overset{def}{=} \langle T_0 \cap T_1, V_0 \cap V_1\}\rangle$$

Let us remark the importance of this operation for the framework delineated here. It allows us to identify common parts of signatures, which will be important for putting schemas together.

We have defined the basic machinery that captures the syntactical aspects of Z and now we need to provide a formalisation of its semantics. Given a signature, an *interpretation* is an assignment of values to variables preserving their typing (i.e. every variable is mapped to a value in the set interpreting its type).

Definition 3.5. Given a signature $\Sigma = \langle T, V\rangle$, an interpretation is a (Σ-)structure $\mathbf{I} = \langle \{D_t\}_{t\in T}, \mathcal{I}\rangle$ such that $\mathcal{I}$ is a function defined as follows:

- $\mathcal{I}(D_{\mathbb{Z}}) = \mathbb{Z}$,
- $\mathcal{I}(t) = D_t$,
- $\mathcal{I}(t \times t') = \mathcal{I}(t) \times \mathcal{I}(t')$,
- $\mathcal{I}(\mathbb{P}(t)) = 2^{\mathcal{I}(t)}$,
- For every $v{:}t$, $\mathcal{I}(v) \in \mathcal{I}(t)$.

That is, an interpretation provides a suitable set for each type and an element (in these sets) for each variable. Note that structured types are interpreted using the denotation of basic types and the operations of standard set theory. The notion of morphisms between interpretations can be obtained is a straightforward way.

Definition 3.6. Given a signature $\Sigma = \langle N, T, V\rangle$ and two Σ-Interpretations $\mathbf{I}^i = \langle \{D^i_t\}_{t\in T}, \mathcal{I}^i\rangle$ for $i = 0, 1$, a morphism $f : \mathbf{I}^0 \to \mathbf{I}^1$ is a collection of functions $f : D^0_t \to D^1_t$ for each $t \in T$, such that the following commutation property holds:

$$f^*(\mathcal{I}^0(v)) = \mathcal{I}^1(v), \text{ for every } v : t \in T$$

where f^* is the homomorphic extension of f to structured types, defined as follows:

- $f^*(a) = f(a)$, if $a \in D^0_t$,
- $f^*(\langle a_0, a_1\rangle) = \langle f^*(a_0), f^*(a_1)\rangle$,
- $f^*(\{a_0 \mapsto b_0, a_1 \mapsto b_1, \dots\}) = \{f^*(a_0) \mapsto f^*(b_0), f^*(a_1) \mapsto f^*(b_1), \dots\}$,
- $f^*(\{a_0, a_1, \dots\}) = \{f^*(a_0), f^*(a_1), \dots\}$.

The notion of *reduct* introduced in Section 2 can be straightforwardly extended to this new setting.

$$\{\langle ts \Rrightarrow \{0,1\}, ps \Rrightarrow \{\langle owns \Rrightarrow \{0\}\rangle\}\rangle\} \models \begin{array}{|l}\hline \textit{Game} \\ ps : \mathbb{P}\,\textit{Player} \\ ts : \mathbb{P}\,\mathbb{N} \\ \hline\end{array}$$

Fig. 3. An interpretation of schema Game.

Definition 3.7. Given two signatures $\Sigma_i = \langle T_i, V_i\rangle$ (for $i = 0, 1$), a morphism $\sigma : \Sigma_0 \to \Sigma_1$ and a Σ_1-structure $\mathbf{I} = \langle \{D_t\}_{t\in T_1}, \mathcal{I}\rangle$ we define the σ-reduct of $\mathbf{I}$ (denoted $\mathbf{I}|_\sigma$) as follows:

$$\mathbf{I}|_\sigma = \langle \{D_t\}_{t\in T_0}, \mathcal{I} \circ \sigma^*\rangle$$

where σ^* is an extension of σ mapping types to types as follows:

- $\sigma^*(v{:}t) = \sigma(v){:}t$,
- $\sigma^*(t \times t') = t \times t'$,
- $\sigma^*(\mathbb{P}(t)) = \mathbb{P}(t)$

In Z literature, interpretations are called *bindings*, as remarked in [Spi92, pp. 28]. Furthermore, we can generalise the notion of interpretation to obtain a variation of this concept. The main idea is that we now admit variables to be interpreted in many ways, leading to a *loose interpretation* described by a collection of interpretations. This enables a more general semantics for schemas: each variable in a schema is now interpreted as a set of possible values, in contrast to the more usual *tight* semantics, where a given interpretation assigns a unique value to a variable. This semantics will be useful, in particular, for formalising promotion (see Section 4).

Definition 3.8. A loose interpretation of a signature Σ is a collection $\{\mathbf{I}_j\}_{j\in J}$, where each $\mathbf{I}_j$ is an interpretation of Σ.

Similarly, we introduce a generalisation of the concept of morphism between interpretations.

Definition 3.9. A morphism $m : \{\mathbf{I}_i\}_{i\in I} \to \{\mathbf{I}'_j\}_{j\in J}$ between two loose interpretations is given by: a function $\iota : I \to J$ and a collection of morphisms $\{m_i : \mathbf{I}_i \to \mathbf{I}'_{\iota(i)}\}_{i\in I}$.

Now, it is possible to define a functor sending each signature to the category of its interpretations, as follows.

Definition 3.10. We define a functor $\mathbf{Mod} : \mathbf{Zign}^{op} \to \mathbf{Cat}$ as follows:

- For each signature Σ, $\mathbf{Mod}(\Sigma)$ is the category whose objects are the collection of interpretations of Σ, and the arrows are the homomorphisms between these structures.
- For each signature morphism $\sigma : \Sigma \to \Sigma'$, $\mathbf{Mod}(\sigma) : \mathbf{Mod}(\Sigma') \to \mathbf{Mod}(\Sigma)$ is the following functor:
  - For each interpretation $\{\mathbf{I}_i\}_{i\in I}$, $\mathbf{Mod}(\Sigma)(\{\mathbf{I}_i\}_{i\in I}) = \{(\mathbf{I}_i)|_\sigma\}_{i\in I}$, that is, it applies pointwise reduct,
  - For each interpretation morphism $m = \{m_i : \mathbf{I}_i \to \mathbf{I'}_{\iota(i)}\}_{i\in I}$, $\mathbf{Mod}(\Sigma)(m) = \{(m_i)|_\sigma : (\mathbf{I}_i)|_\sigma \to (\mathbf{I'}_{\iota(i)})|_\sigma\}_{i\in I}$, are the morphisms obtained by restricting the m_i's to the reduct $(\mathbf{I}_i)|_\sigma$.

Some additional words regarding the interpretation of schemas are useful. Note that a schema can be used to describe the state space of a system; thus, in many cases, a given interpretation can be thought of as describing a state of the system (assigning values to variables); but it should be noted that there could be many interpretations of this schema, each one describing a different feasible state of the system; in the case of loose interpretations, they identify a schema with a collection of feasible states.

We use the standard Z notation to describe signatures, that is, we list the variables with their types inside a box. We provide some examples to illustrate the ideas presented above. An example of a model for *Game* is shown in Figure 3, using a notation borrowed from [Woo96]. This interpretation maps ts to the set $\{0, 1\}$ and ps to a corresponding set. Also note that we use a triple arrow $\Rrightarrow$ to describe an interpretation (or binding). Note that, in the aforementioned figure, each variable is mapped to a corresponding value, the symbol $\models$ is used to state that this structure is an interpretation of the schema *Game*. We will use the same notation when the schema contains some restrictions, as described below.

Now, we need to incorporate in our framework the Z formulas; they are the syntactic ingredient that enables Z designers to write software specifications. Given a signature Σ, we denote by $\mathbf{Sen}(\Sigma)$ the set of Z formulas that we can obtain by using the symbols of Σ. Z formulas include propositional operators, first-order quantifiers, relational operators and lambda expressions; the interested reader can find a detailed definition in [Woo96].

Definition 3.11. We define the functor $\mathbf{Sen} : \mathbf{Zign} \to \mathbf{Set}$ as follows:

- For each signature Σ, we define $\mathbf{Sen}(\Sigma)$ as the set of Z formulas built from the symbols in Σ.
- For each translation $\sigma : \Sigma_0 \to \Sigma_1$, $\mathbf{Sen}(\sigma) : \mathbf{Sen}(\Sigma_0) \to \mathbf{Sen}(\Sigma_1)$ is the translation of formulas obtained based on morphism σ.

The notion of satisfiability can be straightforwardly defined for Z formulas and (loose) interpretations. This relation is denoted by $\vDash_\Sigma \subseteq \mathbf{Mod}(\Sigma) \times \mathbf{Sen}(\Sigma)$. Using these definitions we can prove that Z is an institution.

Theorem 3.3. The structure $\mathbf{Z}$, formed by: *(i)* the category $\mathbf{Zign}$, *(ii)* the functor $\mathbf{Sen} : \mathbf{Zign} \to \mathbf{Set}$, that sends each signature to its set of formulas, *(iii)* the functor $\mathbf{Mod} : \mathbf{Zign}^{op} \to \mathbf{Cat}$, that sends each signature to the category of its models, and *(iv)* the collection of relations $\vDash_\Sigma$ (satisfaction relations relating models of a signature to formulas of the signature), is an *Institution.*

Proof. The main point to prove is that the $\models$-invariance condition for satisfaction holds. This can be proven by structural induction on formulas; since Z formulas are built using standard logical operators and higher-order constructions the interested reader is referred to [GB92, Bor99]. $\square$

We have characterised the basic logical machinery of Z using institutions, we can start providing the definitions concerning the structuring mechanism of Z .

The notion of schema is the most important structuring mechanisms of Z . A schema defines a set of typed variables, and provides constraints on these variables. At first sight, the notion of axiomatic theory captures naturally the notion of schema, as illustrated in [Spi84, CAPM12]; however, schemas support several operations over them (conjunction, disjunction, promotion, etc) that do not have a corresponding formalisation in the category of theories over signatures. In this context, a theory is a signature together with a set of axioms over that signature closed under logical consequence. Pairs formed by a signature and an arbitrary set of axioms are usually called theory presentations and denote the theory obtained by closing the set of axioms by logical consequence. When no confusion arise we will use the term "theory" to denote the latter too. For this reason, in the following definition, we use a generalisation of the notion of theory that allows us to obtain an abstract formalisation of the schema calculus. Intuitively, we define a schema as a "disjunction" of several axiomatic theories. This reflects the fact that schema disjunction is one of the basic operators when combining schemas. Let us give the formal definition of schema.

Definition 3.12. A schema is a a tuple $\langle N, \Sigma, \{\Gamma_i\}_{i \in I}\rangle$ composed of:

- A name N,
- A signature $\Sigma \in \mid \mathbf{Zign} \mid$, containing the set of typed variables declared in the schema,
- A collection $\{\Gamma_i\}_{i\in I}$ of sets of formulas.

As before, we omit the name of schemas and/or signatures when possible; and we assume that the name of a schema and its signature coincide.

Given a schema $S = \langle \Sigma, \{\Gamma_i\}_{i\in I}\rangle$, we denote by $\mathit{Sign}(S)$ its signature, and by $\mathit{Ax}(S)$ its sets of axioms. Note that the relation $\vDash$ can be straightforwardly extended to deal with schemas.

Definition 3.13. If $\mathit{Zchm}(\Sigma)$ denotes the collection of all the schemas with signature Σ, then the relation $\vDash \subseteq \mathit{Mod}(\Sigma) \times \mathit{Zchm}(\Sigma)$ is defined as follows:

$$M \vDash S \text{ iff } \exists\, \Gamma \in \mathit{Ax}(S) \bullet M \vDash \Gamma$$

Let us give some examples to illustrate how these extended notions of axiomatic theories can be employed to capture the concept of schema. Consider the schemas in Figure 4 taken from [Jac97]. Let us write down each of these Z schemas as tuples following Definition 3.12. The box named *Division* in the figure denotes the following schema:

$$\langle \mathit{Division}, \langle \{\mathbb{N}\}, \{n : \mathbb{N}, d : \mathbb{N}, q : \mathbb{N}, r : \mathbb{N}\}\rangle, \{\{d \neq 0, r < d, n = q * d + r\}\}\rangle,$$

while the schema *T_Division* has the following formal counterpart:

$$\langle \mathit{T\_Division}, \langle \{\mathbb{N}\}, \{n : \mathbb{N}, d : \mathbb{N}, q : \mathbb{N}, r : \mathbb{N}\}\rangle, \{\{d \neq 0, r < d, n = q * d + r\}, \{r = 0, q = 0, d = 0\}\}\rangle.$$

Division

$n, d, q, r : \mathbb{N}$

$d \neq 0$
$r < d$
$n = q * d + r$

DivideByZero

$d, q, r : \mathbb{N}$

$d = 0$
$q = 0$
$r = 0$

$$T\_Division \mathrel{\widehat=} Division \vee DivideByZero$$

Fig. 4. Schemas: *Division*, *DivisionByZero*, and *T_Division*.

Note that the last schema is obtained as a disjunction of two schemas. As we will show below, the operations over schemas such as conjunction and disjunction can be formalized as categorical operations. It is worth remarking that the colimit construction is the standard mechanism to conjoin theory presentations (see [GB92] for a detailed presentation); however, in the category of plain theories there is no obvious way to obtain schema disjunction as a categorical construction. We show that the generalisation of theories presented here allows for a categorical formalisation of schema disjunction by using the notion of symmetric monoidal category; we will return to this topic later on.

Let us now define a notion of morphism between schemas; signature morphisms can be straightforwardly extended to schema morphisms:

Definition 3.14. Given schemas $S_0 = \langle \Sigma^0, \{\Gamma^0_i\}_{i \in I} \rangle$ and $S_1 = \langle \Sigma^1, \{\Gamma^1_j\}_{j \in J} \rangle$, a schema morphism $\sigma : S_0 \to S_1$ is a signature morphism $\sigma : Sign(S_0) \to Sign(S_1)$ that satisfies the following condition:

$$\forall\, \Gamma^1 \in Ax(S_1) \bullet \exists\, \Gamma^0 \in Ax(S_0) \bullet \Gamma^1 \models Sen(\sigma)(\Gamma^0) \qquad (1)$$

Essentially, a schema morphism is a mapping between logical theories [End01] in a general sense: each collection of formulas of the target theory logically implies at least one collection of formulas of the origin specification. Roughly speaking, in a schema morphism, the target schema is stronger than the source one, in the sense that the former chooses some theories of the latter and refines them; that is, the implementation (the target schema) selects what part of the source schema (that can be thought as a specification) will implement. Furthermore, note that, if we only consider schemas with singleton sets of sentences, then we obtain the standard concept of interpretation between axiomatic theories.

Schemas and schema morphisms constitute a category. This is the basic structure upon which specifications are built. From now on, when no confusion is likely to arise, instead of writing $\mathbf{Sen}(\sigma)(\Gamma)$ we write $\sigma(\Gamma)$, where Γ is a collection of formulas.

Theorem 3.4. The structure **Zchm** formed by the set of Z schemas and the set of Z schema morphisms, is a category.

Proof. Given a schema S, the identity morphism $id_S : S \to S$ is defined by mapping each symbol to itself. This obviously satisfies Condition 1. Given schemas: $S_j = \langle N^j, \Sigma^j, \Gamma^j_i \rangle$ with $j = 0, 1, 2$ and morphisms $\sigma_1 : S_0 \to S_1$ and $\sigma_2 : S_1 \to S_2$, then $\sigma_2 \circ \sigma_1 : S_0 \to S_2$ is obtained by the composition of the signature translations. The only point to prove is that this translation satisfies Condition 1 of Definition 3.14; we have that:

$$\forall\, \Gamma^2_i \in Ax(S_2) \bullet \exists\, \Gamma^1_i \in Ax(S_1) \bullet \Gamma^2_i \models \sigma_2(\Gamma^1_i)$$

and,

$$\forall\, \Gamma^1 \in Ax(S_1) \bullet \exists\, \Gamma^0 \in Ax(S_0) \bullet \Gamma^1 \models \sigma_1(\Gamma^0)$$

Considering that $\models$ is transitive, and using properties of FOL, we get:

$$\forall\, \Gamma^2 \in Ax(S_2) \bullet \exists\, \Gamma^0 \in Ax(S_0) \bullet \Gamma^2 \models (\sigma_2 \circ \sigma_1)(\Gamma^0)$$

and this ends the proof. $\square$

Note that we can extend the notation $\mathit{Sign}(S)$ (returning the signature of schema S) to a (forgetful) functor $\mathit{Sign} : \mathbf{Zchm} \to \mathbf{Zign}$ which, given a schema returns its signature and given a morphism between schemas returns the signature translation associated with it.

The category **Zchm** has some interesting properties. The first property we prove is that **Zchm** has initial objects.

Theorem 3.5. **Zchm** has an initial element.

Proof. The initial element (up to isomorphism) is $\langle \mathbf{0}, \langle \emptyset, \emptyset \rangle, \{\{\mathbf{true}\}\} \rangle$. Note that $\langle \emptyset, \emptyset \rangle$ is an initial element of **Zign**. Also note that, for any translation from this signature to any other, we have that Condition 1 is trivially true. □

Moreover, we can prove that this category is finitely cocomplete.

Theorem 3.6. **Zchm** is finitely cocomplete.

Proof. Let $D : \mathbf{I} \to \mathbf{Zchm}$ be a finite diagram in **Zchm**; we can obtain a finite diagram $\mathit{Sign} \circ D : \mathbf{I} \to \mathbf{Zign}$ in **Zign**, since **Zign** is finitely cocomplete. This diagram has a colimit, so let $f_i : D(i) \to \Sigma$ be the colimit cocone and Σ its tip. Consider the schema:

$$\langle \Sigma, \{\textstyle\bigcup_{i \in I} f_i(\mathit{sel}(i)) \mid \mathit{sel} : I \to \bigcup_{i \in I} \mathit{Ax}(D(i)) \wedge \forall\, i \in I : \mathit{sel}(i) \in \mathit{Ax}(D(i))\}\rangle$$

where, given a schema S, $\mathit{Ax}(S)$ denotes its collection of axioms, and function $\mathit{sel} : I \to \bigcup_{i \in I} \mathit{Ax}(D(i))$ can be thought of as a selection function. It chooses one set of axioms for each schema in diagram D. We can consider that these collections of axioms are indexed by selection functions. Now, take any arrow $f_i : D(i) \to S$. Given a Γ_f in S, we have that for some Γ^i_k in $D(i)$, $f_i(\Gamma^i_k) \subseteq \Gamma_f$ by definition; this means that $f_i : D(i) \to S$ is a morphism in **Zchm** for any i, thus conforming a cocone. The universality of this cocone follows from the universality of the corresponding cocone in **Zign**. □

In the following, it will be useful to consider schemas with a finite number of axioms and a finite number of variables. The category of finite schemas is defined as follows.

Theorem 3.7. The structure $\mathbf{Zchm}_{\mathit{fin}}$ formed by:

- The objects are schemas $\langle \Sigma, \{\Gamma_i\}_{i \in I} \rangle$ such that I is finite, each Γ_i is finite, and Σ has a finite number of variables and types,
- The arrows are the arrows of **Zchm** restricted to the objects of $\mathbf{Zchm}_{\mathit{fin}}$.

is a category.

Proof. The proof follows as a corollary of of Theorem 3.4. □

Note that Z specifications use finite schemas, that is, in practice we work in $\mathbf{Zchm}_{\mathit{fin}}$. The following properties of $\mathbf{Zchm}_{\mathit{fin}}$ can be proved analogously to the proofs of Theorems 3.5 and 3.6.

Theorem 3.8. $\mathbf{Zchm}_{\mathit{fin}}$ has an initial object.

Theorem 3.9. $\mathbf{Zchm}_{\mathit{fin}}$ is finitely cocomplete.

3.2. Schema Operators

One of the basic operations over schemas is the so-called schema conjunction, an informal description of which can be obtained from Figure 5. In this figure the schema *Division* is defined by means of conjoining schemas *Quotient* and *Remainder*. Basically, a conjunction of two schemas constructs a new signature, retaining the symbols shared by the signatures and conjoining their axioms. This operation can be captured using the notion of pushout, as illustrated in Figure 6.

In this figure schemas S and T are put together preserving their common part W; it is worth noting that W could be any subset of the common part of S and T. Usually, we consider W to be the entire part shared by the two schemas, that is, $\wedge$ is the pushout along the intersection of its signatures. This is formalised as follows:

Quotient

$n, d, q, r : \mathbb{N}$

$d \neq 0$
$n = q * d + r$

Remainder

$r, d : \mathbb{N}$

$r < d$

$$\mathit{Division} \mathrel{\widehat{=}} \mathit{Quotient} \wedge \mathit{Remainder}$$

Fig. 5. Schemas: *Quotient*, *Remainder*, and *Division*.

$$\begin{tikzcd}
& S \wedge T & \\
S \arrow[ur, "i"] & & T \arrow[ul, "j"'] \\
& W \arrow[ul] \arrow[ur] &
\end{tikzcd}
\qquad
\begin{tikzcd}
& U & \\
& S \wedge T \arrow[u, dashed, "u"] & \\
S \arrow[ur, "i"] \arrow[uur, dashed, bend left] & & T \arrow[ul, "j"'] \arrow[uul, dashed, bend right] \\
& W \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

Fig. 6. Schema conjunction as a pushout.

Definition 3.15. Given $S = \langle \Sigma, \{\Gamma_i\}_{i \in I} \rangle$ and $S' = \langle \Sigma', \{\Gamma_j\}_{j \in J} \rangle$, the operation $\wedge$ defines the schema:

$$S \wedge S' \stackrel{def}{=} \langle \Sigma \cup \Sigma', \{\Gamma_{ij}\}_{ij \in I \times J} \rangle$$

where:

- $\Sigma \cup \Sigma' = \langle \mathit{Types}(\Sigma) \cup \mathit{Types}(\Sigma'), \mathit{Var}(\Sigma) \cup \mathit{Var}(\Sigma') \rangle$,
- $\Gamma_{ij} = \Gamma_i \cup \Gamma_j$.

Note that $S \wedge S'$ is (up to isomorphism) the tip of the pushout depicted in the following diagram:

$$\begin{tikzcd}
& S \wedge S' & \\
S \arrow[ur] & & S' \arrow[ul] \\
& \langle \Sigma \cap \Sigma', \{\{\mathbf{true}\}\} \rangle \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

Where $\langle \Sigma \cap \Sigma', \{\{\mathbf{true}\}\} \rangle$ is the schema consisting of the intersection of the signatures (see Definition 3.4) and the axiom **true**. Observe that in **Zchm**, $\wedge$ is not a bifunctor, that is, we cannot generalise it to morphisms. However, restricting **Zchm** to language inclusion, we obtain the functoriality of $\wedge$. This is proved in the theorem below. The interesting point here is that this coincides with the view of schema inclusion given in Z . Let us introduce the subcategory $\mathbf{Zchm}_{\subseteq}$, defined as follows:

Definition 3.16. The category $\mathbf{Zchm}_{\subseteq}$ is the subcategory of **Zchm** such that:

- The objects of $\mathbf{Zchm}_{\subseteq}$ are the objects of **Zchm**,
- The morphisms of $\mathbf{Zchm}_{\subseteq}$ are the signature inclusions in **Zchm** that satisfy Definition 3.14.

Since the relation of inclusion is reflexive and transitive, it is trivial to prove that $\mathbf{Zchm}_{\subseteq}$ is indeed a category; also note that this subcategory preserves colimits and pullbacks (intersections). In this new category we can capture the operator $\wedge$ as a bifunctor.

Definition 3.17. The bifunctor $\wedge : \mathbf{Zchm}_{\subseteq} \times \mathbf{Zchm}_{\subseteq} \to \mathbf{Zchm}_{\subseteq}$ is defined as follows:

- For objects S and S', $S \wedge S'$ is the schema of Definition 3.15.
- For arrows $m : S \to S_0$ and $m' : S' \to S_0'$ it returns the mediating arrow $(m \wedge m')$ of the diagram:

$$\begin{tikzcd}
 & & S_0 \wedge S_0' & & \\
S_0 \arrow[urr, hook] & & S \wedge S' \arrow[u, hook, "m \wedge m'"'] & & S_0' \arrow[ull, hook'] \\
S \arrow[u, hook, "m"'] \arrow[urr, hook] & & \langle \Sigma_0 \cap \Sigma_0', \{\{\mathbf{true}\}\}\rangle \arrow[ull, hook'] \arrow[urr, hook] & & S' \arrow[u, hook', "m'"] \arrow[ull, hook'] \\
 & & \langle \Sigma \cap \Sigma', \{\{\mathbf{true}\}\}\rangle \arrow[ull, hook'] \arrow[u, hook] \arrow[urr, hook] & &
\end{tikzcd}$$

Proving that $\wedge$ is indeed a bifunctor is straightforward because it is associative and it has an identity (up to isomorphism). Thus we have a symmetric monoidal category.

Theorem 3.10. The structure $\langle \mathbf{Zchm}_{\subseteq}, \wedge, \mathbf{1}\rangle$, where: $\mathbf{1} \overset{def}{=} \langle\langle\emptyset,\emptyset\rangle,\{\emptyset\}\rangle$, is a symmetric monoidal category.

Proof. First, consider the following natural isomorphisms:

$$\alpha_{S_0,S_1,S_2} : (S_0 \wedge S_1) \wedge S_2 \to S_0 \wedge (S_1 \wedge S_2).$$

Their existence is given by the associativity (up to isomorphism) of pushouts; then α is formed by iso arrows and therefore it is a natural isomorphism; furthermore, we have that the corresponding associativity diagram commutes (see Section A.1 further details). The unit of $\wedge$ is given by the schema $\mathbf{1} \overset{def}{=} \langle\langle\emptyset,\emptyset\rangle,\{\emptyset\}\rangle$ thus, by simple calculation $\mathbf{1} \wedge \langle \Sigma', \{\Gamma_i\}_{i\in I}\rangle = \langle \Sigma', \{\Gamma_i\}_{i\in I}\rangle$. That is, the identities give us natural isomorphisms: $\lambda_S : \mathbf{1} \wedge S \sim S$ and $\rho_S : S \wedge \mathbf{1} \sim S$, and the coherence properties hold. That finishes the proof that $\langle \mathbf{Zchm}_{\subseteq}, \wedge, \mathbf{1}\rangle$ is a monoidal category; it is straightforward to see that it is symmetric by definition of $\wedge$. □

Note that in Z we can only combine schemas with *compatible* signatures [Spi92], that is, they must give the same type to their common variables. Note that, if we combine two incompatible schemas in **Zchm**, by mean of $\wedge$, we obtain an object of this category which is not a Z schema; however, this is not a problem; restricting ourselves to manipulate compatible schemas will guarantee that we always construct valid Z specifications. In other words, **Zchm** provides not only Z schemas but also a more general version of them, allowing us to deal with the problems related with schema compatibility in a simple and direct way.

We can define an implication between schemas (denoted by $\Rightarrow$); this operator allows us to introduce the negation and the disjunction operators over schemas as shown below.

Definition 3.18. Given schemas $S^i = \langle \Sigma^i, \{\Gamma^i_j\}_{j\in J_i}\rangle$ for $i = 0, 1$, we define the schema:

$$S^0 \Rightarrow S^1 = \langle \Sigma^0 \cup \Sigma^1, \Gamma^*\rangle$$

where:

$$\Gamma^* = \{\textstyle\bigcup_{j_0\in J_0}\{\sigma_2(\neg f(j_0))\} \mid f \in \mathit{Sel}(\{\Gamma^0_{j_0}\}_{j_0\in J_0})\} \cup \{\sigma_1(\Gamma^1_{j_1})\}_{j_1\in J_1}$$

with $\sigma_1 : \Sigma \to \Sigma^0 \cup \Sigma^1$ and $\sigma_2 : \Sigma \to \Sigma^0 \cup \Sigma^1$ being the inclusions of the pushout, and:

$$\mathit{Sel}(\{\Gamma_j\}_{j\in J}) = \{f : J \to \textstyle\bigcup_{j\in J}\Gamma_j \mid \forall j : f(j) \in \Gamma_j\}$$

being the space of functions that selects members of the collection Γ. From now on, we denote it by *Sel* when no confusion arise. We consider Γ^* as indexed by pairs formed by selection functions and $j \in J$.

The interesting point here is that, under certain conditions, $\Rightarrow$ is a right adjoint of $\wedge$. To prove this, we need to consider the category $[\mathbf{Zchm}]_{\Sigma}$; this category is the subcategory of **Zchm** containing the set of schemas over the signature Σ, and restricting morphisms to the identity on Σ. That is, in this category we have all the schemas with the same language, and the arrows can be identified with schema strengthening.

Definition 3.19. Given Σ, the category $[\mathbf{Zchm}]_\Sigma$ is formed by:

- Schemas S such that $Sign(S) = \Sigma$ as objects.
- Morphisms $m : S \to S'$ in **Zchm** such that $Sign(m) = id_\Sigma$ as arrows.

Le us first that $\Rightarrow$ is a bifunctor (contravariant in the first parameter) in $[\mathbf{Zchm}]_\Sigma$.

Theorem 3.11. The mapping $\Rightarrow$: $[\mathbf{Zchm}]_\Sigma^{op} \times [\mathbf{Zchm}]_\Sigma \to [\mathbf{Zchm}]_\Sigma$, defined as in Definition 3.18 over objects can be extended to a bifunctor.

Proof. First, note that the mapping can be straightforwardly extended to morphisms in $[\mathbf{Zchm}]_\Sigma$ mapping identities to identities. We only need to prove that, if $m : S_0 \to S$ and $m' : S' \to S'_0$ are arrows in $[\mathbf{Zchm}]_\Sigma$, then $m \Rightarrow m' : (S \Rightarrow S') \to (S_0 \Rightarrow S'_0)$ is an arrow in $[\mathbf{Zchm}]_\Sigma$. Let us assume there is a set of formulae $\Gamma_i^0 \in Ax(S_0 \Rightarrow S'_0)$. By definition we have that either: $\Gamma_i^0 \in Ax(S'_0)$, in which case the proof is straightforward, or $\Gamma_i^0 \in \{\bigcup_{j\in J}\{\neg f(j)\} \mid f \in Sel(\{\Gamma'_j\}_{j\in J})\}$ where $\{\Gamma'_j\}_{j\in J} = Ax(S'_0)$. Suppose that $\neg\,\exists\,\Gamma \in Ax(\neg S) \bullet \Gamma_0^i \vDash \Gamma$; where $Ax(\neg S) = \{\bigcup_{j\in J}\{\neg f(j_0)\} \mid f \in Sel(\{\Gamma_j\}_{j\in J})\}$ and $\{\Gamma_j\}_{j\in J} = Ax(S)$ (as in Definition 3.18). That is, we have a structure M such that $M \vDash \Gamma_i^0$ and $M \nvDash \Gamma_i^\neg$ for every $\Gamma_i^\neg \in Ax(\neg S)$. Let us pick one $\neg\varphi_i$ for each $\Gamma_i^\neg \in Ax(\neg S)$ such that $M \vDash \neg\varphi_i$, we know that $\{\varphi_i \mid \neg\varphi_i \in \Gamma_i^\neg\} = \Gamma_j$ for some $\Gamma_j \in Ax(S)$; thus $M \vDash \Gamma_j$. Then there is a set of formulae $\Gamma_0 \in Ax(S_0)$ such that $M \vDash \Gamma_0$ (since we have an arrow $m : S_0 \to S$), but note that for some $\psi \in \Gamma_0$ we have that $\neg\psi \in \Gamma_i^\neg$ which is a contradiction. $\square$

Then we can prove the following theorem:

Theorem 3.12. There is a natural isomorphism $hom(S_0 \Rightarrow S_1, S_2) \cong hom(S_1, S_0 \wedge S_2)$ in $[\mathbf{Zchm}]_\Sigma$.

Proof. Let us define an one-to-one mapping from $hom(S_0 \Rightarrow S_1, S_2)$ to $hom(S_1, S_0 \wedge S_2)$. Since the morphisms in $[\mathbf{Zchm}]_\Sigma$ are the identities in **Zign**, we only need to prove that, if we have a morphism $id : S_0 \Rightarrow S_1 \to S_2$, meaning that condition:

$$\forall\,\Gamma^2 \in Ax(S_2) \bullet \exists\,\Gamma^\Rightarrow \in Ax(S_0 \Rightarrow S_1) \bullet \Gamma^2 \vDash \Gamma^\Rightarrow,$$

holds, then the corresponding condition holds for $id : S_1 \to S_0 \wedge S_1$.

Now, given $\Gamma^\wedge \in Ax(S_0 \wedge S_1)$, by definition of $\wedge$, $\Gamma^\wedge = \Gamma^0 \cup \Gamma^2$ where $\Gamma^0 \in Ax(S_0)$ and $\Gamma^2 \in Ax(S_2)$. Now, observe that for any $\Gamma^\Rightarrow \in Ax(S_0 \Rightarrow S_1)$, by definition, we have either

- $\Gamma^\Rightarrow = \Gamma^1$, where $\Gamma^1 \in Ax(S_1)$, or
- $\Gamma^\Rightarrow = \bigcup_{i\in I}\{\neg f(i)\}$, where $f : I \to Ax(S_0)$ is a selection function.

In the first case we have that $\Gamma^2 \vDash \Gamma^1$, for some $\Gamma^1 \in Ax(S_1)$ and then $\Gamma^2 \cup \Gamma^0 \vDash \Gamma^1$, for some $\Gamma^1 \in Ax(S_1)$. Otherwise, $\Gamma^\Rightarrow = \bigcup_{i\in I}\{\neg f(i)\}$, and then $\Gamma^0 \cup \bigcup\{\neg f(i)\} \vDash \mathbf{false}$ holds, since $\bigcup\{\neg f(i)\}$ is composed of the negations of formulas of $Ax(S_0)$; and therefore $\Gamma^0 \cup \Gamma^2 \vDash \mathbf{false}$ must also hold. Thus, for any $\Gamma^1 \in Ax(S_1)$ we have $\Gamma^0 \cup \Gamma^2 \vDash \Gamma^1$ holds, and consequently in either case the theorem follows. $\square$

As a direct consequence of the previous theorem, we can prove the following result.

Theorem 3.13. For any signature Σ, we have that $\langle [\mathbf{Zchm}]_\Sigma, \wedge, \langle \Sigma, \{\emptyset\}\rangle\rangle$ is a monoidal closed category.

Summarising, we have bifunctors $\wedge$ and $\Rightarrow$; the first captures (in a general way) the concept of schema conjunction, while the bifunctor $\Rightarrow$ can be associated with an implication between schemas. When we restrict the languages of the schemas to a particular language (defining the states of our system), the two functors are adjoints. Furthermore, we can use these two operators to define the remaining Z schema operators.

Definition 3.20. We define the operators $\vee, \neg$ as follows:

- $\neg S \overset{def}{=} S \Rightarrow \mathbf{False}_{Sign(S)}$, where $\mathbf{False}_\Sigma \overset{def}{=} \langle \Sigma, \{\{\mathbf{false}\}\}\rangle$,
- $S \vee S' \overset{def}{=} \neg S \Rightarrow S'$.

Following the proof of Theorem 3.11, it is straightforward to prove that $\vee$ is a bifunctor in $[\mathbf{Schm}]_\Sigma$, and $\neg$ is a contravariant functor in $[\mathbf{Schm}]_\Sigma$.

Other interesting schema operators are the schema quantifiers; these operators allow one to introduce universal or existential quantification over schemas. They can be straightforwardly formalised considering finite schemas. A more powerful logical framework might be obtained by generalising this to schemas with an infinite number of axioms; this is not studied in this paper since we restrict ourselves to formalising Z .

We can capture existential quantification as a functor: $\exists\, x \in t \bullet - : \mathbf{Zchm}_{\mathit{fin}} \to \mathbf{Zchm}_{\mathit{fin}}$, as described in the following definition.

Definition 3.21. We define $\exists\, x \in t \bullet - : \mathbf{Zchm}_{\mathit{fin}} \to \mathbf{Zchm}_{\mathit{fin}}$ as follows:

- Given a schema $S = \langle \Sigma, \{\Gamma_i\}_{i \in I}\rangle$ where $\Sigma = \langle T, V\rangle$, then:

  $$\exists\, x \in t \bullet S \overset{def}{=} \langle \Sigma - \{x : t\}, \{\Gamma^*_i\}_{i \in I}\rangle$$

  where: $\Sigma - \{x : T\} \overset{def}{=} \langle T \cup \{t\}, V - \{x : t\}\rangle$, and: $\Gamma^*_i \overset{def}{=} \{\exists\, x \in T \bullet \bigwedge \Gamma_i\}$, where $\bigwedge \Gamma_i$ is the conjunction of the formulas in Γ_i
- Given a translation $\sigma : S \to S'$, we define an arrow $\exists\, x \in t \bullet \sigma : \exists\, x \in t \bullet S \to \exists\, x \in t \bullet S$ as:

  $$\exists\, x \in t \bullet \sigma(v : t) \overset{def}{=} v : t$$

  that is, it is the restriction of σ to $\langle T, V - \{x : T\}\rangle$.

  This mapping is a functor as proven in the following theorem.

Theorem 3.14. $\exists\, x \in t \bullet - : \mathbf{Zchm}_{\mathit{fin}} \to \mathbf{Zchm}_{\mathit{fin}}$ is a functor.

Proof. $\exists\, x \in t \bullet - : \mathbf{Zchm}_{\mathit{fin}} \to \mathbf{Zchm}_{\mathit{fin}}$ is a mapping between schemas. Given a morphism $m : S \to S'$, where $S = \langle s, \{\Gamma_i\}_{i \in I}\rangle$ and $S' = \langle s', \{\Gamma'_{i'}\}_{i' \in I'}\rangle$, then we have that:

$$\forall\, \Gamma' \in Ax(S') \bullet \exists\, \Gamma \in Ax(S) \bullet \Gamma' \vDash m(\Gamma)$$

But note that using properties of first-order logic we get the formula:

$$\forall\, \Gamma' \in Ax(S') \bullet \exists\, \Gamma \in Ax(S) \bullet \exists\, x \in t \bullet \bigwedge \Gamma' \vDash \exists\, x \in t \bullet \bigwedge m(\Gamma)$$

and then $\exists\, x \in T \bullet m : \exists\, x \in t \bullet S \to \exists\, x \in t \bullet S'$ is a morphism in $\mathbf{Zchm}_{\mathit{fin}}$.

We now have to prove that it preserves identity and composition of morphisms. For any identity $id : S \to S$, it is trivial to see that it maps each variable to itself and therefore we obtain an identity: $\exists\, x \in t \bullet id : \exists\, x \in t \bullet S \to \exists\, x \in t \bullet S$. For composition, take two arrows in $\mathbf{Zchm}$, $m_0 : S_0 \to S_1$ and $m_1 : S_1 \to S_2$, where $S_i = \langle s_i, \{\Gamma^i_{j_i}\}_{j_i \in J_i}\rangle$, for $i = 0, 1, 2$. Proving that $\mathit{Sign}(m_1 \circ m_0) : \mathit{Sign}(S_0) \to \mathit{Sign}(S_2)$ is a translation is straightforward. Then, we have:

$$\forall\, \Gamma^2 \in Ax(S_2) \bullet \exists\, \Gamma^1 \in Ax(S_1) \bullet \Gamma^2 \vDash m_1(\Gamma^1)$$

and:

$$\forall\, \Gamma^1 \in Ax(S_1) \bullet \exists\, \Gamma^0 \in Ax(S_0) \bullet \Gamma^1 \vDash m_0(\Gamma^0)$$

Using properties of first-order quantifiers, this means that:

$$\forall\, \Gamma^2 \in Ax(S_2) \bullet \exists\, \Gamma^1 \in Ax(S_1) \bullet \exists\, x \in t \bullet \bigwedge \Gamma^2 \vDash m_1(\exists\, x \in t \bullet \bigwedge \Gamma^1)$$

and

$$\forall\, \Gamma^1 \in Ax(S_2) \bullet \exists\, \Gamma^0 \in Ax(S_1) \bullet \exists\, x \in t \bullet \bigwedge \Gamma^1 \vDash m_0(\exists\, x \in t \bullet \bigwedge \Gamma^0)$$

and therefore by transitivity we obtain:

$$\forall\, \Gamma^2 \in Ax(S_2) \bullet \exists\, \Gamma^0 \in Ax(S_2) \bullet \exists\, x \in t \bullet \bigwedge \Gamma^2 \vDash m_1 \circ m_0(\exists\, x \in t \bullet \bigwedge \Gamma^0)$$

This implies that $m_1 \circ m_0$ is a morphism between schemas. $\square$

Universal quantification can be defined as usual, as the dual of existential quantification as follows.

Definition 3.22. Given a schema S we define: $\forall\, x \in t \bullet S \overset{def}{=} \neg(\exists\, x \in t \bullet \neg S)$.

Using the same reasoning as before we can prove that it is a functor:

Theorem 3.15. The mapping $\forall\, x \in t \bullet - : \mathbf{Zchm}_{\mathit{fin}} \to \mathbf{Zchm}_{\mathit{fin}}$ is a functor.

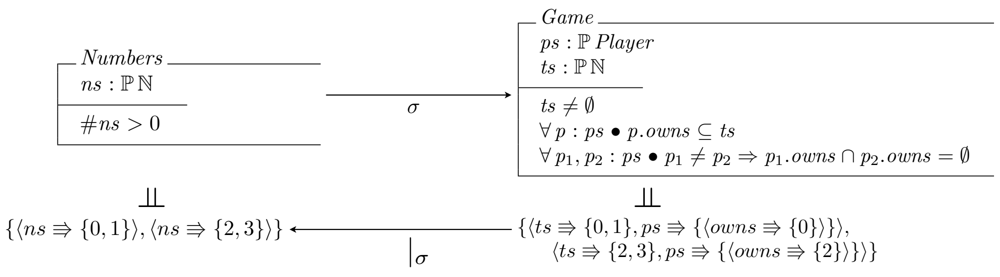
Numbers
$ns : \mathbb{P}\,\mathbb{N}$
$\#ns > 0$
σ
Game
$ps : \mathbb{P}\, Player$
$ts : \mathbb{P}\,\mathbb{N}$
$ts \neq \emptyset$
$\forall\, p : ps \bullet p.owns \subseteq ts$
$\forall\, p_1, p_2 : ps \bullet p_1 \neq p_2 \Rightarrow p_1.owns \cap p_2.owns = \emptyset$
$\models$
$\models$
$\{\langle ns \Rrightarrow \{0,1\}\rangle, \langle ns \Rrightarrow \{2,3\}\rangle\}$
$|_\sigma$
$\{\langle ts \Rrightarrow \{0,1\}, ps \Rrightarrow \{\langle owns \Rrightarrow \{0\}\rangle\}\rangle,$
$\langle ts \Rrightarrow \{2,3\}, ps \Rrightarrow \{\langle owns \Rrightarrow \{2\}\rangle\}\rangle\}$

Fig. 7. An example involving schemas, schema models, a schema morphism and the corresponding model reduct.

Furthermore, we can introduce quantification over schemas (as done in Z) as follows.

Definition 3.23. Given schemas S, W in $\mathbf{Zchm}_{\mathit{fin}}$ such that $\mathit{Sign}(S) = \langle T, \{v_0{:}t_0, \ldots, v_n{:}t_n\}\rangle$, we define:

$$\exists\, S \bullet W \overset{def}{=} \exists\, v_0 \in t_0 \bullet (\exists\, v_1 \in t_1 \bullet \ldots (\exists\, t_n \in t_n \bullet W)\ldots),\ \ \textit{and}$$
$$\forall\, S \bullet W \overset{def}{=} \forall\, v_0 \in t_0 \bullet (\forall\, v_1 \in t_1 \bullet \ldots (\forall\, t_n \in t_n \bullet W)\ldots).$$

Let us note some facts about schemas and schemas operators. As we said, an arrow $S \to S'$ represents schema strengthening (modulo translation), as understood in Z. For instance, we have the following properties:

Theorem 3.16. We have the following arrows in **Zchm**:

- $S \to S \wedge S'$,
- $S \vee S' \to S$, when $\mathit{Sign}(S \vee S') \cong \mathit{Sign}(S)$,
- $S \to \mathbf{True}_{\mathit{Sign}(S)}$, where $\mathbf{True}_\Sigma \overset{def}{=} \langle \Sigma, \{\{\mathbf{true}\}\}\rangle$,
- $S \wedge \mathbf{True}_{\mathit{Sign}(S)} \cong S$,
- $S \vee \mathbf{False}_{\mathit{Sign}(S)} \cong S$, where $\mathbf{False}_\Sigma \overset{def}{=} \langle \Sigma, \{\{\mathbf{false}\}\}\rangle$,
- $\forall\, x \in t \bullet S \wedge S' \cong (\forall\, x \in t \bullet S) \wedge (\forall\, x \in t \bullet S')$, in $\mathbf{Zchm}_{\mathit{fin}}$.

Proof. The properties follow directly from the definitions of $\wedge, \exists$ and $\mathbf{T}_\Sigma$. $\square$

Since in practice we only use finite schemas, in what follows we assume, unless otherwise stated, that we are working in the subcategory $\mathbf{Zchm}_{\mathit{fin}}$. Furthermore, let us note that, given finite schemas S and S', the schema $S \wedge S'$ is also finite. That is, considering the category of finite schemas named $\mathbf{Zchm}^{\subseteq}_{\mathit{fin}}$, we get the following result:

Theorem 3.17. $\langle \mathbf{Zchm}^{\subseteq}_{\mathit{fin}}, \wedge, \langle\langle \emptyset, \emptyset\rangle, \{\emptyset\}\rangle\rangle$ is a symmetric monoidal category.

In order to clarify the above view of signatures and schemas as objects in a category, consider the diagram in Figure 7. This diagram involves two simple schemas, one of them being our previous *Game* schema, and the other being a simple schema defining a nonempty set of natural numbers. A schema morphism in this diagram shows that the simpler schema is embedded (via morphism σ) into the schema *Game*, where the variable *ns* is translated as variable *ts*. Notice that, for this morphism to be correct, one must be able to prove that the axiom $\#ns > 0$ is a consequence of the constraints in the *Game* schema, which is trivial. After this simple example, the reader familiar with Z may notice that schema morphisms subsume the notion of schema strengthening. Models complement the picture of schemas and schema morphisms.

An example of interpretation for $Numbers$ is also shown in Figure 7, using the notation of [Woo96]. This model maps *ns* to the sets $\{0,1\}$ and $\{2,3\}$; note that morphism σ induces a mapping $()_{|\sigma} : \mathbf{Mod}(\mathit{Game}) \to \mathbf{Mod}(\mathit{Numbers})$ between models of *Game* and models of $Number$ [GB92]. This mapping builds *reducts* as defined above, i.e., given a model of *Game*, it removes from the model all the parts that are unnecessary to interpret symbols originating in $Numbers$, obtaining a model of the smaller schema; this scenario is also shown in Figure 7.

(a)
$$\begin{tikzcd}
 & C & \\
A \arrow[ur, "f"] & & B \arrow[ul, "g"']
\end{tikzcd}$$

(b)
$$\begin{tikzcd}
 & Op & \\
S \arrow[ur, "i"] \arrow[rr, dashed, "(-)'"] & & S' \arrow[ul, "j"']
\end{tikzcd}$$

Fig. 8. Cospans, and Z operations as cospans.

3.3. Z Operations

In Z specifications, one usually defines operations via schemas that relate the description of the pre and post states of the operation. In our categorical view of Z , operations correspond to a particular class of diagrams, of the form shown in Figure 8 (a), where A and B are the related "domain" schemas, and C is the operation schema. Such a diagram, called cospan, is a categorical diagram in the category **Zchm**; a cospan is usually described by a pair of morphisms with common codomain, e.g., $\langle f : A \to C, g : B \to C \rangle$.

An operation for a system S (captured as a schema) is typically specified as a schema extending ΔS, i.e., over the conjunction of S and S', where S' represents the "post" state of S, i.e., the state after the operation has been executed. Such an operation is also a cospan, and has the form shown in Figure 8 (b). Meanwhile, the priming operation can be characterised as a functor in **Zign** as we do in the following definition.

Definition 3.24. For any signature $\Sigma = \langle T, V \rangle$ we define the functor $(-)' : \mathbf{Zign} \to \mathbf{Zign}$ as follows:

- $\Sigma' \overset{def}{=} \langle T, \{v':t \mid v:t \in V\} \rangle$
- Given $\sigma : \Sigma \to \Sigma_0$ where $\Sigma_0 = \langle T_0, V_0 \rangle$, we define $\sigma'(v':t) \overset{def}{=} w' : t$ if $\sigma(v:t) = w:t$.

The proof that this is a functor is straightforward. This functor can be extended to the category **Zchm** as follows it is done in the following definition.

Definition 3.25. We define $(-)' : \mathbf{Zchm} \to \mathbf{Zchm}$ as follows:

- For objects: $\langle \Sigma, \{\Gamma_i\}_{i \in I} \rangle' \overset{def}{=} \langle \Sigma', \{Sen((-)')(\Gamma_i)\}_{i \in I} \rangle$
- For arrows: let $\sigma : s_0 \to s_1$ be a translation, then $\sigma' : s_0' \to s_1'$ is obtained by applying $(-)'$.

Next theorem proves that the functor $(-)'$ is an equivalence of categories.

Theorem 3.18. The functor $(-)' : \mathbf{Zchm} \to \mathbf{Zchm}$ is an equivalence of categories.

An operation is a specification that relates initial states with final states, in a way similar to the way program specifications are written in Hoare Logic; that is, operations are cospans whose domain and codomain are related in a certain way. Let us formalise the concept of operations more precisely.

Definition 3.26. Let $T, S \in \mid \mathbf{Zchm}_{fin} \mid$, then an operation Op is a cospan with shape $f : S \to Op \leftarrow T : g$ in $\mathbf{Zchm}_{fin}$ such that $T \cong S$ (that is, there is an arrow between T and S).

We use the notation $Op : S \Rightarrow T$ to express that $\langle f : S \to Op, g : T \to Op \rangle$ is an operation.

Operations modifying the state S of a system (captured as a schema) are usually defined over ΔS. ΔS can also be captured categorically:

Definition 3.27. Given a schema S, we denote by ΔS the coproduct $S + S'$.

Note that, for any other schema combining S_1 and S_2 (meaning that we have schema morphisms from S_1 and S_2 to the combined schema), there exists a unique schema morphism u from the coproduct to this combined schema that makes the diagram (involving these schemas and the schema morphisms corresponding to the combinations) commute. This situation is described in Figure 9, for the case of ΔS, the coproduct of S and S'.

We have used an arrow notation for cospans, in our characterisation of Z operations. In fact, cospans can be thought of as arrows (or morphisms), which are composed by applying pushouts [Ber67]. This is the basic way in which schema composition is categorically captured.

Definition 3.28. Given two cospans $\langle f : S \to Op_1, g : T \to Op_1 \rangle$ and $\langle f' : W \to Op_2, g' : V \to Op_2 \rangle$ such that $T \cong W$, we define the schema $Op_1 \fatsemi Op_2$ as follows:

$$Op_1 \fatsemi Op_2 \overset{def}{=} \exists\, u(T) \bullet Op_1 \rhd Op_2$$

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

Fig. 9. Categorical definition of ΔS as a coproduct.

where $Op_1 \rhd Op_2$ is the tip of the following colimit diagram:

$$\begin{tikzcd}
 & & & Op_1 \rhd Op_2 & & & \\
 & Op_1 \arrow[urr, "i"] & & & & Op_2' \arrow[ull, "j"'] & \\
S \arrow[ur, "f"] & & T \arrow[ul, "g"'] & \cong & W \arrow[ur, "f'"] & & V \arrow[ul, "g'"']
\end{tikzcd}$$

$u : T \to Op_1 \rhd Op_2$ is the arrow from T to the colimit, and $u(T)$ denotes the schema obtained by translating T by using u. Note that $\langle i \circ f : S \to Op_1 \rhd Op_2, j \circ g' : V \to Op_1 \rhd Op_2 \rangle$ is a cospan.

Note that in Z we usually want to compose two operations of the form $Op_i : S \to S'$ (for $i = 1, 2$); the reader should note that the composition of such schemas can be obtained using the constructions presented above. This generalizes the usual composition of cospans in category theory [Ber67].

Another useful construction in Z is the ΞS operation. This operator on schemas denotes a *skip* operation. That is, it is a special case of ΔS, in which S and S' are identical. This schema operator can be also defined (up to isomorphism) as follows:

Definition 3.29. Given $S = \langle \Sigma, \{\Gamma_i\}_{i \in I} \rangle$, $\Xi S : S \Rightarrow S'$ is defined as follows:

$$\Xi S = \langle \Sigma + \Sigma', \{v = v' \mid v : t \in \Sigma\} \rangle$$

This schema has some interesting properties: it is an identity with respect to the composition of operations:

Theorem 3.19. Given $Op : S \Rightarrow S'$, we have the following properties of ΞS:

- $Op \fatsemi \Xi S \cong Op$
- $\Xi S \fatsemi Op \cong Op$

where the symbol $\cong$ indicates that there exists an isomorphism between the two corresponding objects in $\mathbf{Zchm}_{\mathit{fin}}$.

Proof. Note that Op is a tip of a cocone with the shape of Definition 3.28, thus we have a mediating arrow $Op \rhd \Xi S$ to Op. It is straightforward using Def.inition 3.21 to check that the language of $\exists\, u(S') \bullet Op \rhd \Xi S$ is isomorphic to Op, thus we have an iso $Op \fatsemi \Xi S \to Op$. The other property is similar. $\square$

Given schemas S and S', we have a category $\mathsf{Cospans}(S, S')$ where the objects are the cospans between S and S' and the morphisms are the schema morphisms between the corresponding cospans. Interestingly, the category $\mathsf{Op}(S, S')$ of operations between S and S' is a subcategory of $\mathsf{Cospan}(S, S')$.

Definition 3.30. Given two (isomorphic) schemas S and S', the category of operations between S and S' denoted as $\mathsf{Op}(S, S')$ is formed by:

- Operations as defined in Definition 3.26 $\langle i : S \to Op, j : S' \to Op \rangle$ as objects,
- Given two operations $\langle i : S \to Op, j : S' \to Op \rangle$ and $\langle i : S \to Op', j : S' \to Op \rangle$, an arrow is a morphism $m : Op \to Op'$ (in $\mathbf{Zchm}_{\mathit{fin}}$), such that the following diagram commutes.

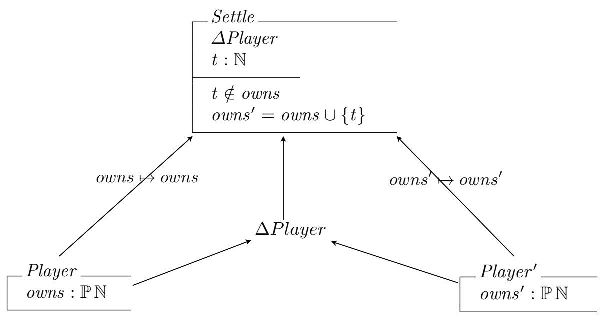

Fig. 10. A Z specification as a categorical diagram in **Zpec**.

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

These kinds of constructions form a *bicategory* [Ber67]. An important remark is that we can think of our category of schemas as having two different kinds of arrows, one representing schema morphisms (schema embeddings after translation), and another one capturing Z operations (as cospans), with $\fatsemi$ acting as the composition for the latter.

Definition 3.31. Zpec is formed by:

- The set of finite schemas as its set of objects (called 0-cells).
- For each pair of schemas S, S', the category $Op(S, S')$ of operations between S and S' (called 1-cells), and morphisms between operations (called 2-cells).
- The composition between 2-cells is defined as in Definition 3.28.

It is well-known that a category together with the collection of cospans satisfies the definition of a bicategory, see for instance [Mac98]; in the same way we can prove that **Zpec** is a bicategory.

Theorem 3.20. Zpec is a bicategory.

Proof. The horizontal composition of the bicategory is $\fatsemi$, and the identity is given by $1_S = \Xi S$ for any S. We must prove that the laws of bicategories hold. First, note that, by properties of colimits, we have that $(Op_0 \rhd Op_1) \rhd Op_2 \cong Op_0 \rhd (Op_1 \rhd Op_2)$. Since $\exists\, x \in t \bullet -$ is a functor, and functors preserve isomorphisms, we have that $(Op_0 \fatsemi Op_1) \fatsemi Op_2 \cong Op_0 \fatsemi (Op_1 \fatsemi Op_2)$. Thus we can straightforwardly define a natural isomorphism between $(Op_0 \fatsemi Op_1) \fatsemi Op_2$ and $Op_0 \fatsemi (Op_1 \fatsemi Op_2)$. Note that, since Theorem 3.19, ΞS behaves as an identity. Thus the pentagon and triangle diagrams commute. □

Summarizing, a Z specification is a collection of schemas $S_0, \ldots, S_n$ together with a set of cospans $Op_i : S_i \Rightarrow S'_i$, all elements in the bicategory of Z specifications. An example illustrating schemas and operations, and their relationships, is shown in Figure 10, as a diagram in **Zpec**.

4. Schemas as Types and Promotion

In this section we focus on formalizing *promotion*. A key feature of Z that facilitates promotion, and software specification in general, is the use of schemas as types. In order to capture this in our mathematical formalism, we need to spice up our categorical framework with some additional machinery. Towards this goal, we

```
── S ─────────────────
│ v₀ : T₀
│ . . .
│ vₙ : Tₙ
├──────────
│ φ
└─────────────────────
```

```
── S^P ───────────────
│ v₀ : Index ⇸ T₀
│ . . .
│ vₙ : Index ⇸ Tₙ
│ S : ℙ Index
├──────────
│ S ⊆ dom(v₀)
│ ⋮
│ S ⊆ dom(vₙ)
│ ∀ s ∈ S • φ^P
└─────────────────────
```

Fig. 11. A schema and its manager construction.

introduce the concept of schema manager, which supports the idea of managing schema instances. Roughly speaking, a manager of a component C is a component that provides the behaviour of various instances of C, and usually enables the manipulation of these instances. This technique makes it possible to interpret schemas as types in a way that differs from the established mechanism for doing this as presented in [Woo96]. Our approach consists of building a manager specification. Consider the schemas in Figure 11; the one on the left represents an arbitrary schema, involving $v_0 : T_0, \dots, v_n : T_n$ as its typed variables. The schema on the right represents the *manager* for the previous schema, where ϕ^P is obtained from ϕ by adding a parameter of type *Index* to each variable. For the schema on the right, *Index* is simply a fresh *given type.*

First, we define the notion of promoted signature. Since our construction of promotion introduces a new type, this definition of promoted signature uses the name of the signature to introduce the new type.

Definition 4.1. We define a functor $(-)^P : \mathbf{Zign} \to \mathbf{Zign}$ as follows:

- For signatures $\Sigma = \langle N, T, V\rangle$:

$$\Sigma^P \overset{def}{=} \langle N^P, T \cup \{\mathit{Index}\}, \{v{:}\mathit{Index} \nrightarrow t \mid v{:}t \in V\} \cup \{N{:}\,\mathbb{P}\,\mathit{Index}\}\rangle.$$

- For translations $\sigma : \Sigma_0 \to \Sigma_1$ (where $\Sigma_i = \langle N_i, T_i, V_i\rangle$ for $i = 0, 1$):

$$\sigma^P(v{:}t') \overset{def}{=} \begin{cases} \sigma(v){:}\mathit{Index} \nrightarrow t & \text{if } t' = \mathit{Index} \nrightarrow t \\ N_1{:}\,\mathbb{P}\,\mathit{Index} & \text{otherwise} \end{cases}$$

Here *Index* is a fixed given type.

Proving that $(-)^P$ is indeed a functor is direct. Using this definition, we introduce a mapping $(-)^P : \mathbf{Zchm} \to \mathbf{Zchm}$ that promotes schemas.

Definition 4.2. We define the mapping $(-)^P : \mathbf{Zchm}_{\mathit{fin}} \to \mathbf{Zchm}_{\mathit{fin}}$ in the following way:

- For schemas:

$$\langle N, \langle T, V\rangle, \{\Gamma_i\}_{i\in I}\rangle^P \overset{def}{=} \langle N^P, \langle T, V\rangle^P, \{\{\forall x \in N \bullet \textstyle\bigvee_{i\in I} \bigwedge \Gamma_i(x)\} \cup \{N \subseteq \mathit{dom}\ v_i \mid v_i{:}t_i \in V\}\}\rangle,$$

  where $\Gamma(x) \overset{def}{=} \{\varphi(x) \mid \varphi \in \Gamma\}$, for any set of formulas Γ.
- For arrows $\sigma : S_0 \to S_1$ (where $S_i = \langle N_i, \langle T_i, V_i\rangle, \{\Gamma^i_j\}_{j\in J}\rangle$, for $i = 0, 1$, are schemas) we proceed as follows. First, note that a morphism between schemas is basically a translation between their signatures that preserves axioms; thus we define $\sigma^P : \Sigma_0^P \to \Sigma_1^P$ as in Definition 4.1 and then show that condition of Definition 3.14 holds.

Let us prove that the mapping $(-)^P : \mathbf{Zchm} \to \mathbf{Zchm}$ is a functor.

Theorem 4.1. The mapping $(-)^P : \mathbf{Zchm} \to \mathbf{Zchm}$ is a functor.

Proof. First, we need to prove that, given $\sigma : S_0 \to S_1$, for $S_i = \langle \Sigma_i, \{\Gamma^i_j\}_{j\in J_i}\rangle$, $\sigma^P : S_0^P \to S_1^P$ is a morphism between schemas, that is, we must prove that:

$$\forall\, \Gamma^1 \in Ax(S_1^P) \bullet \exists\, \Gamma^0 \in Ax(S_0^P) \bullet \Gamma^1 \models \sigma^P(\Gamma^0).$$

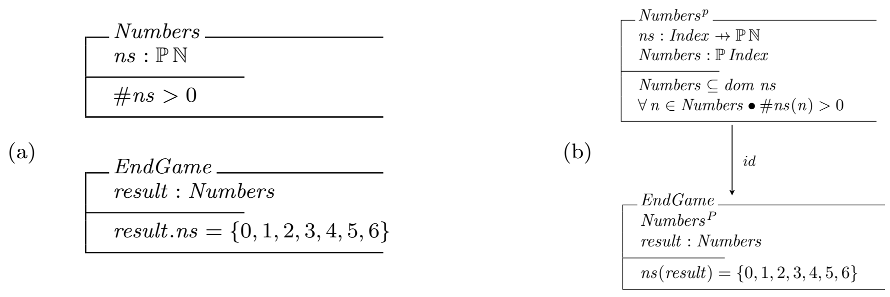
(a)
Numbers
$ns : \mathbb{P}\,\mathbb{N}$
$\# ns > 0$
EndGame
$result : Numbers$
$result.ns = \{0, 1, 2, 3, 4, 5, 6\}$
(b)
$Numbers^p$
$ns : Index \rightarrow\!\!\!\!\!\!+ \mathbb{P}\,\mathbb{N}$
$Numbers : \mathbb{P}\, Index$
$Numbers \subseteq dom\ ns$
$\forall\, n \in Numbers \bullet \# ns(n) > 0$
id
EndGame
$Numbers^P$
$result : Numbers$
$ns(result) = \{0, 1, 2, 3, 4, 5, 6\}$

Fig. 12. Using managers as types

First, note that, since $\sigma : S_0 \to S_1$ is a morphism in **Zchm**, we have that:

$$\forall\, \Gamma^1 \in Ax(S_1) \bullet \exists\, \Gamma^0 \in Ax(S_0) \bullet \Gamma^1 \vDash \sigma(\Gamma^0).$$

and then by properties of FOL we get:

$$\forall\, x \in N \bullet \bigvee\nolimits_{j \in J_1} \bigwedge \Gamma^1_i(x) \vDash \sigma^P(\forall\, x \in N \bullet \bigvee\nolimits_{j \in J_0} \bigwedge \Gamma^0_i(x))$$

thus implying that $\sigma^P : S_0^P \to S_1^P$ is a morphism between schemas.

Now, we need to prove that promoting schemas preserves identities and composition. The preservation of composition is straightforward since the translation of a composition is defined basically as the composition of signature translations. For identity, if we have $id : S \to S$, then $id^P : S^P \to S^P$ maps each variable to itself and the new introduced variable to itself too, that is, it is the identity translation over $Sign(S)$, also it satisfies Condition 1 of Definition 3.14. □

Using a schema as a type can be achieved by including the manager of the schema. Let us illustrate this with an example.

Consider the schema given in Figure 12 (a); it defines the end state of a game where the player needs to have conquered territories 0 to 6. Notice that the actual semantics of this schema can be defined using the schema manager $Numbers^P$ introduced above, simply by including $Numbers^P$ in the schema. This has a self evident categorical interpretation, and the existence of an arrow between $Numbers$ and $EndGameEndGame$ relates them both syntactically and semantically. The resulting diagram is shown in Figure 12 (b), where $result.ns$ is just syntactic sugar for $ns(result)$.

We can define a similar transformation over operations. Consider the schemas in Figure 13; the schema on the left is the definition of an operation, where $v_0, \dots, v_n, v_0', \dots, v_n'$ are the variables of S and S', respectively. We introduce the schema on the right. This situation is graphically depicted as a categorical diagram in Fig. 14. Therein, the dashed arrows denote the application of the transformation described above. Thus, the definition of this mapping is as follows.

Definition 4.3. Given an operation in $Op(S, S') = \langle f : S \to Op, g : S' \to Op \rangle$ where $i : S \to S'$ is the required isomorphism, we define an operation in $Op^p(S^p, (S')^p)$ as follows.

- S^p and $(S')^p$ are defined as in Definition 4.2.
- Let $Op = \langle\langle T, V\rangle, \{\Gamma_i\}_{i \in I}\rangle$ be the tip of the cospan, then we define $Op^p : S^p \Rightarrow (S')^p$ to be $Op^P = \langle\langle T \cup \{Index\}, \{v{:}Index \rightarrow\!\!\!\!\!\!+ t \mid v{:}t \in V\} \cup \{S{:}\,\mathbb{P}\, Index, S'{:}\,\mathbb{P}\, Index, this{:}S, this'{:}S'\}\rangle, \{\Gamma_i^p\}_{i \in I}\rangle$.

where:

$$\Gamma_i^p = \{\varphi[v_0(this)/v_0, \dots v_n(this)/v_n, i(v_0)(this')/i(v_0), \dots i(v_n)(this')/i(v_n)] \mid \varphi \in \Gamma_i\} \cup \bigcup_{i \leq n} \{\{this'\} \lhd i(v_i) = \{this\} \lhd v_i\} \cup \{S = S'\}$$

That is, the promoted operation takes into account the new variables representing the instance to which the operation is applied, and the rest of the operation is reformulated accordingly. Let us note that, in the promoted operations, we add an axiom stating that $S = S'$, thus the indexes in S are exactly the same as the indexes belonging to S'. That is, we assume that operations do not change the number of instances

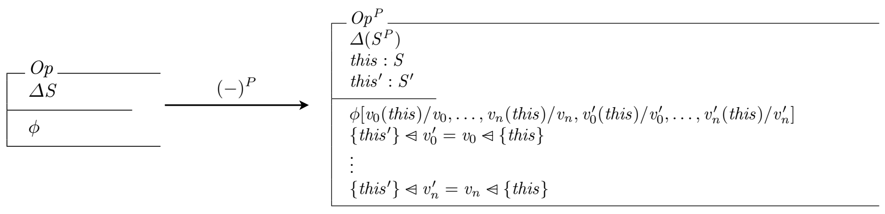
Op
ΔS
ϕ
$(-)^P$
Op^P
$\Delta(S^P)$
$this : S$
$this' : S'$
$\phi[v_0(this)/v_0, \ldots, v_n(this)/v_n, v_0'(this)/v_0', \ldots, v_n'(this)/v_n']$
$\{this'\} \lhd v_0' = v_0 \lhd \{this\}$
$\vdots$
$\{this'\} \lhd v_n' = v_n \lhd \{this\}$

Fig. 13. Operation promotion using managers.

$$\begin{tikzcd}
{} & {} & {} & Op^P & {} \\
{} & Op \arrow[urr, dashed] & S^P \arrow[ur] & {} & (S^P)' \arrow[ul] \\
S \arrow[ur] \arrow[urr, dashed] & {} & S' \arrow[ul] \arrow[urr, dashed] & {} & {}
\end{tikzcd}$$

Fig. 14. Categorical diagram depicting operation promotion.

of a promoted type. Since operations that add or delete instances could be needed (as in object oriented programming), the definition of promotion for operations can be straightforwardly modified to support this; we do not tackle this issue in this paper.

In Figure 13 this translation is illustrated by means of the usual notation of Z .

The important point is that the translation $(-)^P$: **Zpec** $\to$ **Zpec** is a mapping between structures, which maps schemas to promoted schemas, and operations to promoted operations. We can define it in three parts:

- A functor $(-)^P$: **Zchm** $\to$ **Zchm**, which translates schemas in the way described above.
- A functor $(-)^P : \mathsf{Op}(S, T) \to \mathsf{Op}(S^P, T^P)$, that translates operations to promoted operations. (For the sake of simplicity we use $(-)^P$ for naming both these functors.)
- The canonical extension of $(-)^P$ to formulas, as explained above.

The following lemma will be important to prove some properties about promotion.

Lemma 4.1. In $\mathbf{Zchm}_{fin}$ we have an arrow: $(\exists\, v \in \mathit{Index} \nrightarrow t \bullet S^P) \to (\exists\, v \in t \bullet S)^P$.

Proof. Let $\langle\langle V, T\rangle, \{\Gamma_i\}_{i\in I}\rangle$ be a schema. To prove that there is such a morphism, first note that the language of both schemas (domain and codomain) are exactly the same except for the new type introduced that may have different names. We should prove that:

$$\forall\, \Gamma \in Ax((\exists\, v \in t \bullet S)^P) \bullet \exists\, \Gamma' \in Ax(\exists\, v \in \mathit{Index} \nrightarrow t \bullet S^P) \bullet \Gamma \vDash \Gamma'$$

But note that $Ax((\exists\, v \in t \bullet S)^P)$ has a unique axiom which is:

$$\forall\, x \in N \bullet \bigvee_{i\in I} \exists\, v(x) \in t \bullet \Gamma_i$$

and similarly for $Ax(\exists\, v \in \mathit{Index} \nrightarrow t \bullet S^P)$ whose axiom is:

$$\exists\, v \in \mathit{Index} \nrightarrow t \bullet \forall\, x \in N' \bullet \bigvee_{i\in I} \bigwedge \Gamma_i(x)$$

That both axioms are equivalent (renaming N' by N) follows from the rule of skolemization of second order logic, thus the translation that maps each symbol to itself and N (the new variable introduced in the codomain) is mapped to N' (the new variable introduced by promotion in the domain) is an arrow between $(\exists\, v \in \mathit{Index} \nrightarrow t \bullet S^P)$ and $(\exists\, v \in t \bullet S)^P$. $\square$

The following theorem can be proven by resorting to the definition of $(-)^P$.

Theorem 4.2. $(-)^P$: **Zpec** $\to$ **Zpec** is a lax functor.

$$\begin{tikzcd}
 & & Op_0'' \fatsemi Op_1'' & & \\
 & Op_0'' \arrow[ur] & Op_0' \fatsemi Op_1' \arrow[u] & Op_1'' \arrow[ul] & \\
 & Op_0' \arrow[u, "h"] \arrow[ur] & Op_0 \fatsemi Op_1 \arrow[u] & Op_1' \arrow[u, "k"'] \arrow[ul] & \\
 & Op_0 \arrow[u, "h'"] \arrow[ur] & & Op_1 \arrow[u, "k'"'] \arrow[ul] & \\
S \arrow[ur] \arrow[uuur, bend left] \arrow[uur, bend left] & & S' \arrow[ul] \arrow[ur] \arrow[uuul, bend right] \arrow[uul, bend right] \arrow[uuur, bend left] \arrow[uur, bend left] & & S'' \arrow[ul] \arrow[uuul, bend right] \arrow[uul, bend right]
\end{tikzcd}$$

Fig. 15. Distributivity of $\fatsemi$ and $\circ$

Proof. First, let us recall the components of a lax functor. A lax functor $F : \mathbf{A} \to \mathbf{B}$ between bicategories $\mathbf{A}$ and $\mathbf{B}$ is composed of: (i) for every object A of $\mathbf{A}$ an object $F(A)$ in $\mathbf{B}$, (ii) for every pair of objects A, B of $\mathbf{A}$, a functor $F_{A,B} : \mathbf{A}(A,B) \to \mathbf{B}(F(A), F(B))$, (iii) for every triple of objects A, B, C in $\mathbf{A}$, a natural transformation: $\gamma_{A,B,C} : ;_{F(A),F(B),F(C)} \circ (F_{A,B} \times F_{B,C}) \stackrel{\cdot}{\to} F_{A,C} \circ ;_{A,B,C}$, and (iv) for every object A in $\mathbf{A}$ a natural transformation $\delta_A : 1_{F(A)} \stackrel{\cdot}{\to} F_{A,A} \circ 1_A$, where 1_A is the unit isomorphism of the corresponding bicategory. Subject to the coherence laws (see Section A.2 for further details).

First, let us define a natural transformation $\delta_A : 1_{A^P} \stackrel{\cdot}{\to} (-)^P \circ 1_A$. Note that in both cases (the domain and codomain of the morphism) for any schema A, we have Ξ^P_A. Thus an iso arrow exists (the identity).

Now, let us define a natural transformation: $\gamma : \fatsemi \circ (-)^P \times (-)^P \to (-)^P \circ \fatsemi$. First note that by the properties of colimit we have an arrow: $m : Op_1^P \rhd Op_2^P \to (Op_1 \rhd Op_2)^P$. But now since $(\exists\, v \in t \bullet -)$ is a functor we get an arrow: $\exists\, u(T^P) \bullet Op_1^P \rhd Op_2^P \to: \exists\, u(T^P) \bullet (Op_1 \rhd Op_2)^P$, thus by Lemma 4.1, we get an arrow $\exists\, u(T) \bullet Op_1^P \rhd Op_2^P \to: (\exists\, u(T) \bullet Op_1 \rhd Op_2)^P$. By Definition 3.28 this is an arrow $Op_1^P \fatsemi Op_2^P \to (Op_1 \fatsemi Op_2)^P$. This collection of arrows form the natural transformation γ. As before the coherence properties follow from the definition of $\fatsemi$ and the properties of colimits. $\square$

Lax functors are morphisms between bicategories; this means that promotion is coherent with respect to identities and composition of operations. One property that we obtain as a corollary of this is that promotion distributes w.r.t. composition of schema strengthening:

Theorem 4.3. For schemas S, S', S'' where $h', h \in \| \mathsf{Op}(S, S') \|$ and $k', k \in \| \mathsf{Op}(S', S'') \|$, we have that $(k \circ k') \fatsemi (h \circ h') = (k \fatsemi h) \circ (k' \fatsemi h')$ holds.

An illustration of the meaning of this theorem can be found in Figure 15.

First note that this equation is well-typed since $\fatsemi$ is a bifunctor, that is, it also maps arrows in $\mathsf{Op}(S, S') \times \mathsf{Op}(S', S'')$ to arrows in $\mathsf{Op}(S, S'')$. Therefore, the equation of Theorem 4.3 states that composing operations and then strengthening them (for instance, by strengthening preconditions in the usual way) is the same as strengthening and then composing.

Furthermore, we can prove the following theorem.

Theorem 4.4. Given two schemas S_0 and S_1 we have an arrow $a : S_0^P \wedge S_1^P \to (S_0 \wedge S_1)^P$.

Proof. For the first property, let $S_0 = \langle\langle T_0, V_0\rangle, \{\Gamma^0_i\}_{i \in I}\rangle$ and $S_1 = \langle\langle T_1, V_1\rangle, \{\Gamma^1_j\}_{j \in J}\rangle$ be schemas. Recall that a morphism between two schemas is a translation between their signatures that preserves axioms (Definition 3.14). Let us first define this translation. The signature of $S_0^P \wedge S_1^P$ is:

$$Sign(S_0^P \wedge S_1^P) = \langle T_0 \cup T_1 \cup \{Index\}, \{v : Index \pfun t \mid v \in V_0 \cup V_1\} \cup \{s_0 : \mathbb{P}\, Index\} \cup \{s_1 : \mathbb{P}\, Index\}\rangle$$

File
$f : \mathit{Key} \pfun \mathit{Record}$
$\mathit{blocked} : \mathit{Bool}$

$\mathit{FileBlocked}$
File
$\mathit{blocked}$

$\mathit{FileUnblocked}$
File
$\neg \mathit{blocked}$

$\mathit{FileBlocked}^P$
$f : \mathit{Index} \pfun (\mathit{Key} \pfun \mathit{Record})$
$\mathit{blocked} : \mathit{Index} \pfun \mathit{Bool}$
$\mathit{FileB} : \mathbb{P}\,\mathit{Index}$
$\mathit{FileB} \subseteq \mathit{dom}\ f$
$\mathit{FileB} \subseteq \mathit{dom}\ \mathit{blocked}$
$\forall x \in \mathit{FileB} \bullet x.\mathit{blocked}$

$\mathit{FileUnblocked}^P$
$f : \mathit{Index} \pfun (\mathit{Key} \pfun \mathit{Record})$
$\mathit{blocked} : \mathit{Index} \pfun \mathit{Bool}$
$\mathit{FileU} : \mathbb{P}\,\mathit{Index}$
$\mathit{FileU} \subseteq \mathit{dom}\ f$
$\mathit{FileU} \subseteq \mathit{dom}\ \mathit{blocked}$
$\forall x \in \mathit{FileU} \bullet \neg x.\mathit{blocked}$

$\mathit{FileBlocked}^p \vee \mathit{FileUnblocked}^p$
$f : \mathit{Index} \pfun (\mathit{Key} \pfun \mathit{Record})$
$\mathit{blocked} : \mathit{Index} \pfun \mathit{Bool}$
$\mathit{FileB} : \mathbb{P}\,\mathit{Index}$
$\mathit{FileU} : \mathbb{P}\,\mathit{Index}$
$\mathit{FileB} \subseteq \mathit{dom}\ f$
$\mathit{FileU} \subseteq \mathit{dom}\ f$
$\mathit{FileB} \subseteq \mathit{dom}\ \mathit{blocked}$
$\mathit{FileU} \subseteq \mathit{dom}\ \mathit{blocked}$
$(\forall fx \in \mathit{FileB} \bullet x.\mathit{blocked}) \vee (\forall f \in \mathit{FileU} \bullet \neg x.\mathit{blocked})$

$(\mathit{FileBlocked} \vee \mathit{FileUnblocked})^p$
$f : \mathit{Index} \pfun (\mathit{Key} \pfun \mathit{Record})$
$\mathit{blocked} : \mathit{Index} \pfun \mathit{Bool}$
$\mathit{File} : \mathbb{P}\,\mathit{Index}$
$\mathit{File} \subseteq \mathit{dom}\ f$
$\mathit{File} \subseteq \mathit{dom}\ \mathit{blocked}$
$(\forall x \in \mathit{File} \bullet (x.\mathit{blocked} \vee \neg x.\mathit{blocked})$

Fig. 16. Counterexample for promotion

where $s_0 : \mathbb{P}\,\mathit{Index}$ and $s_1 : \mathbb{P}\,\mathit{Index}$ are the variables added by promoting the corresponding schemas. On the other hand, we have:

$$\mathit{Sign}((S_0 \wedge S_1)^P) = \langle T_0 \cup T_1 \cup \{\mathit{Index}\}, \{v{:}\mathit{Index} \pfun t \mid v \in V_0 \cup V_1\} \cup \{s_0 \wedge s_1{:}\mathbb{P}\,\mathit{Index}\}\rangle$$

Thus, we can define a translation $\sigma : \mathit{Sign}(S_0^P \wedge S_1^P) \to \mathit{Sign}((S_0 \wedge S_1)^P)$, mapping $S_0{:}\mathit{Index}$ and $S_1{:}\mathit{Index}$ to $S_0 \wedge S_1{:}\mathit{Index}$, and any other variable to itself. To finish the proof we need to show that this translation satisfies the condition of Def. 3.14. Note that we have:

$$\mathit{Ax}(S_0 \wedge S_1)^P = \{\{\forall x \in S_0 \wedge S_1 \bullet \textstyle\bigvee_{(i,j) \in I \times J} \bigwedge \Gamma^0_i(x) \wedge \bigwedge \Gamma^1_j(x)\}\}$$

and:

$$\mathit{Ax}(S_0^P \wedge S_1^P) = \{\{\forall x \in S_0 \bullet \textstyle\bigvee_{i \in I} \bigwedge \Gamma^0_i(x), \forall x \in S_1 \bullet \bigvee_{j \in J} \bigwedge \Gamma^1_j(x)\}\}$$

which is equivalent by considering properties of FOL and distributivity of $\vee$ w.r.t. $\wedge$ in propositional logic to:

$$\mathit{Ax}(S_0^P \wedge S_1^P) = \{\{\forall x \in S_0 \bullet \forall x \in S_1 \bullet \textstyle\bigvee_{(i,j) \in I \times J} \bigwedge \Gamma^0_i(x) \wedge \bigwedge \Gamma^1_j(x)\}\}$$

Now taking into account that variables $S_0{:}\mathit{Index}$ and $S_1{:}\mathit{Index}$ are mapped to $S_0 \wedge S_1{:}\mathit{Index}$ by σ, we obtain that:

$$\forall x \in S_0 \bullet \textstyle\bigvee_{i \in I} \bigwedge \Gamma^0_i(x), \forall x \in S_1 \bullet \bigvee_{j \in J} \bigwedge \Gamma^1_j(x) \vDash \sigma(\forall x \in S_0 \bullet \forall x \in S_1 \bullet \bigvee_{(i,j) \in I \times J} \bigwedge \Gamma^0_i(x) \wedge \bigwedge \Gamma^1_j(x))$$

Thus, $\sigma : S_0^P \wedge S_1^P \to (S_0 \wedge S_1)^P$ is a morphism in $\mathbf{Zchm}_{\mathit{fin}}$ $\square$

Roughly speaking, this property states a (semi-)distributive property of promotion w.r.t. schema conjunction. One may expect an equivalence (an isomorphism in this case) in this theorem since universal quantification distributes over conjunction in first-order logic (and promotion has an universal character). The main problem here is that promotion introduces new variables, and in this case the arrow a does not have an inverse.

One of the main points of our formalisation is that we characterise promotion as a mapping between specifications. To the authors' knowledge this is not achieved in any related work. For instance, in [Woo90] a promoted state is captured as a particular schema; this prevents an investigation of the properties of promotion, such as the one stated in Theorem 4.4.

Let us illustrate the fact that promotion does not distribute over disjunction consider the schemas in Figure 16.

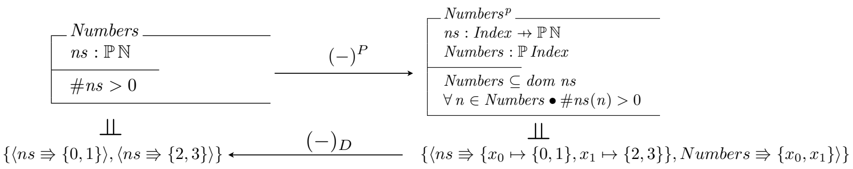
Numbers
$ns : \mathbb{P}\,\mathbb{N}$
$\#ns > 0$
$(-)^P$
$Numbers^p$
$ns : Index \nrightarrow \mathbb{P}\,\mathbb{N}$
$Numbers : \mathbb{P}\,Index$
$Numbers \subseteq dom\ ns$
$\forall\, n \in Numbers \bullet \#ns(n) > 0$
$\perp\!\!\!\perp$
$\{\langle ns \Rrightarrow \{0,1\}\rangle, \langle ns \Rrightarrow \{2,3\}\rangle\}$
$(-)_D$
$\perp\!\!\!\perp$
$\{\langle ns \Rrightarrow \{x_0 \mapsto \{0,1\}, x_1 \mapsto \{2,3\}\}, Numbers \Rrightarrow \{x_0, x_1\}\rangle\}$

Fig. 17. Example of mappings between schemas and models

We use the example introduced in [Jac97] adapted to our setting, where we have a file and we promote this schema to have a file system. In this case we use the technique introduced above, and therefore a schema $File^p$ is defined (this schema can be included in other ones to form the specification of a filesystem). The important point here is that the schema $(FileBlocked \lor FileUnblocked)^p$ is not equivalent to $FileBlocked^p \lor FileUnblocked^p$. It worth noting, that the latter schema introduces two subsets of *Index* while the former introduces only one (here named *File* for the sake of clarity).

4.1. Promotion as an Institution Representation

Let us note that, given a model M of a promoted schema S^P, we can define a corresponding model M_D (a degraded model), which forgets the new sort introduced. In Figure 17 a simple example of mapping between schemas and their models is shown to illustrate these ideas.

As explained in Section 2, these kinds of mappings are called *institution representations* [Tar95], which are mappings between logical systems. Intuitively, a collection of schemas and the relations between them define a logical system. An institution representation allows us to translate one logical system into another, while keeping the basic properties of theories, or schemas in this case.

Institution representations were introduced informally above, where we argued for their inclusion for capturing promotion. As institutions are an abstract characterisation of logical systems, institution representations capture the notion of embedding of a logical system into another one [Tar95]. The logical machinery of Z used for describing states and operations constitutes an institution, and the operation of promoting schemas corresponds to an institution representation from this institution to itself. The key elements involved in the promotion process are:

- The definition of a mapping (functor) $(-)^P$: **Zign** $\to$ **Zign**, mapping a signature to its promoted signature.
- The definition of a mapping (natural transformation) $(-)_D : Mod \circ (-)^P \to Mod$, mapping models of promoted signatures to models of the original signature.
- The definition of a mapping (natural transformation) $(-)^P$: **Sen** $\to$ **Sen** $\circ\, (-)^P$ mapping formulas of the original signature to formulas of the promoted signature.

These mappings satisfy the property $M \models \phi^P \Leftrightarrow M_D \models \phi$. That is, a model of a promoted signature satisfies a promoted property if and only if the degraded model satisfies the original property. A graphical representation of this situation is shown in Fig. 18. To clarify this diagram, suppose that we have a translation from one schema signature to another schema signature (named σ). Notice that reducts move in the opposite direction of translations (this explains the $(-)^{op}$ in the definition of institutions). Then, if we take a reduct of a promoted schema, and so we take the degraded model (the right path of the diagram), we obtain the same model as if we take the degraded model first and then take the reduct (the left path in the diagram). This ensures the coherence between the operations of strengthening and promotion in Z, which is guaranteed by the following theorem.

Theorem 4.5. $(-)^P$ and $(-)_D$ form an institution representation.
Proof. As shown above $(-)^P$: **Zign** $\to$ **Zign** is a functor. The functor $\Sigma_D : Mod(\Sigma^P) \to Mod(\Sigma)$ is defined as follows:

- Given a structure $\mathbf{I} = \{I_j\}_{j \in J}$ of Σ^P, we define

$$\mathbf{I}_D \overset{def}{=} U_{j \in J, x \in I_j(N)}\{D_N(x)(I_j)\}$$

$$\begin{tikzcd}
\Sigma_2 & & Mod(\Sigma_2) & Mod(\Sigma_2^P) \arrow[l, "(-)_D"'] \arrow[d, "Mod(\sigma^P)"] \\
\Sigma_1 \arrow[u, "\sigma"] & & Mod(\Sigma_1) \arrow[from=1-3, "Mod(\sigma)"'] & Mod(\Sigma_1^P) \arrow[l, "(-)_D"]
\end{tikzcd}$$

Fig. 18. Institution representations.

where $D_N(x)(I_j)(v:t) \overset{def}{=} I_j(v)(x)$.

It is straightforward to extend this mapping to reducts. Identities are mapped to identities, and composition is preserved, thus $(-)_D$ is a functor. For formulas, each formula φ is translated to $\varphi^P \overset{def}{=} \forall\, x \in N \bullet \varphi(x)$, which is obviously natural w.r.t. translation. We only need to prove that:

$M \models \forall\, x \in N \bullet \varphi(x) \Leftrightarrow M_D \models \varphi$

but this is straightforward by definition of M_D.

5. Heterogeneous Z Specifications and Structuring

Following the recent trend in software engineering that favours a "multiple views" approach to specification and design, the Z notation has been extended in various ways, in combination with other notations. Some of these extensions are Z-CSP [Fis97] (Z plus the process algebra CSP), and Z plus statecharts [Web96]; more recently, the language *Circus* was introduced in [Woo01]. This language combines Z and CSP, providing also a refinement calculus for this extension of Z. These *heterogeneous* specification languages pose new challenges, e.g., for defining appropriate formal semantics for the composite languages, and for providing effective mechanisms to reason about these specifications.

A consequence of the abstract nature of our formalisation of Z, and its structuring mechanisms, is that we can deal with these extensions in a systematic way. Most formalisms for specifying software systems can be viewed as institutions; well-known examples are: first-order logics [GB92], temporal logics [GB92], modal logics [GB92], Unity-like languages [FM92] and process algebras [MR06], all constitute institutions. Our formalisation of the basic construction of Z in an institutional setting, and the wide toolset available from the theory of institutions, enables us to flexibly combine Z with other formalisms, obtaining extensions of Z with appropriate, well structured semantics. It is important to remark that the combination of institutions is a well studied area; see for instance [MTP97]. In this section we use basic ways of combining two institutions but the interested reader is referred to the cited work to have a deeper understanding of this area.

In order to illustrate this important characteristic of our formalisation, we describe in this section the combination of Z with CSP (structured CSP, as introduced in [MR06]). The combination thus obtained is, in essence, similar to the framework Z-CSP, with a well defined structured semantics, that makes the semantic relationships between different (heterogeneous) components of a specification explicit. We make use of the CSP (structured CSP) institution. The interested reader can find the details of this formalism in [MR06]. Signatures in this institution are pairs $\langle A, P\rangle$, where A is an alphabet (used for the communication of processes), and P is a collection of process names. Elements of both A and P have an associated list of typed parameters. A morphism $\langle f, g\rangle : \langle A, P\rangle \to \langle A', P'\rangle$ between two CSP signatures consists of an injective function $f : A \to A'$, mapping members of A to members of A' preserving parameters and their types[1], and a function $g : P \to P'$, mapping process names to process names, preserving parameters and their types. The category of CSP signatures is called CSPSig [MR06]. A CSP theory is a tuple $\langle \Sigma, \pi\rangle$, where Σ is a CSP signature, and π is a set of processes in the CSP notation. A model of a theory is given by a set of traces corresponding to the processes of the theory. For the sake of simplicity, we employ a finite trace semantics (as introduced in [MR06]), although the failure-divergence semantics is also supported in this institution. We have a morphism between models $M_1 \to M_2$ iff $M_2 \sqsubseteq M_1$ (i.e., M_2 is a refinement of M_1). A simple

[1] The use of injective mappings introduces some subtle technical problems when combining specifications. A way of avoiding these problems is described in [MR06].

$$\begin{array}{lcl} \{\langle\rangle, \langle coin\rangle, & & \\ \langle coin, choc\rangle, & & A = \{coin, choc\} \\ \langle coin, choc, coin\rangle & \models & P = \{VM\} \\ \ldots\} & & \pi = \{VM = coin \rightarrow choc \rightarrow VM\} \end{array}$$

Fig. 19. A theory in Structured CSP, and a model of it.

example of a vending machine is described as a CSP theory in Figure 19. Neither communication letters nor processes have parameters in this example. A model of the theory accompanies the example as well.

A new institution **CZP** can be defined using the institutions **CSP** and **Z**. Essentially, we want specifications to have a data part, given in Z with its corresponding operations, and a process part, with each atomic process being associated with an operation as described in the Z part of the specification.

Definition 5.1. The category SignCZP of **CZP** signatures is composed of:

- tuples $\Sigma = \langle \Sigma_{CSP}, \Sigma_Z \rangle$ as signatures, where Σ_{CSP} and Σ_Z are CSP and Z signatures, respectively;
- a morphism $\sigma : \Sigma \rightarrow \Sigma'$ is a tuple of morphisms $\langle f : \Sigma_{CSP} \rightarrow \Sigma'_{CSP}, g : \Sigma_Z \rightarrow \Sigma'_Z \rangle$.

We can also introduce a functor **Sen**, that formalizes the syntactical constructions of this new formalism.

Definition 5.2. The functor $\mathbf{Sen}_{CZP}$ is defined as follows:

$\mathbf{Sen}_{CZP}(\langle \Sigma_{CSP}, \Sigma_Z \rangle) = \langle \mathbf{Sen}_{CSP}(\Sigma_{CSP}), \mathbf{Sen}_Z(\Sigma_Z) \rangle$.

The semantics of this logical systems is given by execution traces, characterised in the following definition:

Definition 5.3. The functor $\mathbf{Mod}_{CZP}$ is defined as follows:

- Given $\Sigma = \langle \Sigma_{CSP}, \Sigma_Z \rangle$, we define:

  $$\mathbf{Mod}(\Sigma) = \{\langle \langle a_1, \ldots, a_n \rangle, \langle \mathbf{I}_1, \ldots, \mathbf{I}_{n+1} \rangle \mid \exists\, M \in \mathbf{Mod}(\Sigma_{CSP}) : \langle a_1, \ldots, a_n \rangle \in M \wedge \mathbf{I}_i \in \mathbf{Mod}(\Sigma_Z)\}.$$

  That is, a model is a possible trace together with a sequence of states representing the state changes produced by this finite trace.
- Given a morphism $\sigma : \Sigma_0 \rightarrow \Sigma_1$ (where $\langle \Sigma_i = \langle A_i, N_i \rangle \rangle$) the morphism $\mathbf{Mod}(\sigma)$ is defined pointwise, using reducts of traces as defined in [MR06] and reducts of schema interpretations as defined in Section 3.

That is, models are execution traces, together with models of the corresponding operations. The relation $\models_{CZP}$ is also defined resorting to $\models_{CSP}$ and $\models_Z$ as follows:

Definition 5.4. $M \models \langle \pi, \phi \rangle$ 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 $\mathbf{I}_i \models \phi$, for every i. (Here note that each $\mathbf{I}_i$ is a collection of (loose) interpretations as defined in Section 3).

Specifications in **CZP** are theories in **Z**; together with processes in **CSP** and some elements that coordinate the two, as defined in the following definition.

Definition 5.5. A theory in **CZP** is a tuple $\langle \Sigma_{CSP}, \Sigma_Z, S, Ops, events, \pi \rangle$, where:

- $\Sigma_{CSP} = \langle A, N \rangle$ is a signature in **CSP**,
- Σ_Z is a signature in **Z**,
- S is a schema $\langle S, \Phi \rangle$,
- $OPS = \{op_0 : S \Rightarrow S', \ldots\, op_n : S \Rightarrow S'\}$ is a collection of operations over the state S[2],
- $event : A \rightarrow OPS$ is a function mapping events to operations, and mapping $event : Par(a) \rightarrow Par(event(a))$, for every $a \in A$, thus identifying parameters of each event with parameters of the corresponding operation.
- π is a set of **CSP** processes.

In the above, for a given event a, $Par(a)$ returns its (actual) parameters and for a Z operation schema $OP : S \Rightarrow S'$, $Par(OP)$ returns its list of parameters (input and output parameters).

[2] Here we use the notation $op : S \Rightarrow S'$ to indicate that op is a Z operation on state S, as defined in Section 3.

$$
\begin{aligned}
&(skip)^P \overset{def}{=} skip\\
&(stop)^P \overset{def}{=} stop\\
&n(x_1 : T_1, \ldots, x_n : T_n)^P = n(x : S, x_1 : T_1, \ldots, x_n : T_n)\\
&(a \to Proc)^P \overset{def}{=} a?x : X \to Proc^P\\
&(?y{:}T \to Proc)^P \overset{def}{=} ?x{:}X?y{:}T \to Proc^P\\
&(S \Box Q)^P \overset{def}{=} S^P \Box S^P\\
&(S \sqcap Q)^P \overset{def}{=} S^P \sqcap Q^P\\
&(S \parallel Q)^P \overset{def}{=} S^P \parallel Q^P\\
&(P \mid\mid\mid Q)^P \overset{def}{=} S^P \mid\mid\mid S^P
\end{aligned}
$$

$$M_D \overset{def}{=} \bigcup_{x \in S} \{\sigma_x \mid \sigma \in M\}$$

where σ_x is obtained by deleting the events in the trace where x is not present, similarly for the corresponding interpretation of schemas.

Fig. 20. Promoting basic CSP operators, and degrading traces.

Morphisms between **CZP** theories are straightforwardly defined pointwise. The relation $\models$ can be extended to theories:

$$M \models \langle \Sigma_{CSP}, \Sigma_Z, S, OPS, event, \pi \rangle$$

iff

$$\pi_1(M) \models \pi \text{ and } \langle \mathbf{I}_i, \mathbf{Mod}((-)')(\mathbf{I}_{i+1}), event \rangle \models event(a_i)$$

where $\langle \mathbf{I}_i, \mathbf{Mod}((-)')(\mathbf{I}_{i+1}), event \rangle$ is the interpretation obtained by using $\mathbf{I}_i$ to give values to the variables in Σ_Z (unprimed variables in OP), $\mathbf{I}_{i+1}$ to assign values to variables in Σ'_Z (primed variables in OP),and the values of the input and output variables in OP are assigned according the function $event$ that matches event (and its parameters) with operations (and its parameters). An example is shown in Figure 21.

Let us note that promotion can be easily extended to this new institution. We define functor $(-)^P$: $\mathsf{CZPSign} \to \mathsf{CZPSign}$, mapping signatures to signatures, as follows. Given a signature $\langle \Sigma_{CSP}, \Sigma_Z \rangle$, Σ_Z is translated to Σ_Z^P, and Σ_{CSP} is mapped to the following **CSP** signature:

- If $a \in A$, then $a^P = a.x$, where $x \in S$, being S the new type introduced in Σ_Z^P,
- If $n \in N$, then $n(x_1 : T_1, \ldots, x_n : T_n)^P = n(x : S, x_1 : T_1, \ldots, x_n : T_n)$.

This functor is extended to sentences in **CZP**: the translation of a process is defined inductively as in Figure 20, and the translation of **Z** formulas is defined as in Section 3. Furthermore, we define the mapping $(-)_D$ between models as in Figure 20. This extension of promotion is also an institution representation:

Theorem 5.1. Mappings $(-)^P$ and $(-)_D$ form an institution representation.

Fig. 21 shows, using a simple example, how promotion works in this new setting. In this case, we have a standard specification of a buffer with its corresponding process specification. The schemas and the CSP process on the left are promoted to the corresponding one the right. Via promotion, we obtain a specification with various buffers whose executions interleave.

6. Related Work

Several frameworks have been proposed to give a formal semantics for Z. The original semantics proposed in [Spi84] uses signatures and axiomatic theories, and the semantics of these axiomatic theories is given by means of varieties; this work can be thought of as a precursor of the framework introduced here. However, in that initial work, many important aspects of Z (such as schema calculus, schemas as types and promotion) cannot be captured.

In [Bau99], institutions are used for providing semantics to Z specifications; in this work, schemas are captured as logical sentences in an institution, and therefore a Z specification is viewed as an unstructured set of expressions. In contrast, our approach makes use of theories and morphisms between them in formalising Z designs, thus leading to a well structured categorical semantics of designs. In [Buj04], category theory is used in the definition of a relational semantic framework to interpret Z, as well as other specification languages. As in our case, the approach allows for heterogeneous specification; however, the work uses Z simply as an example of a language based on the “state & operations” viewpoint, but it does not show how to deal with

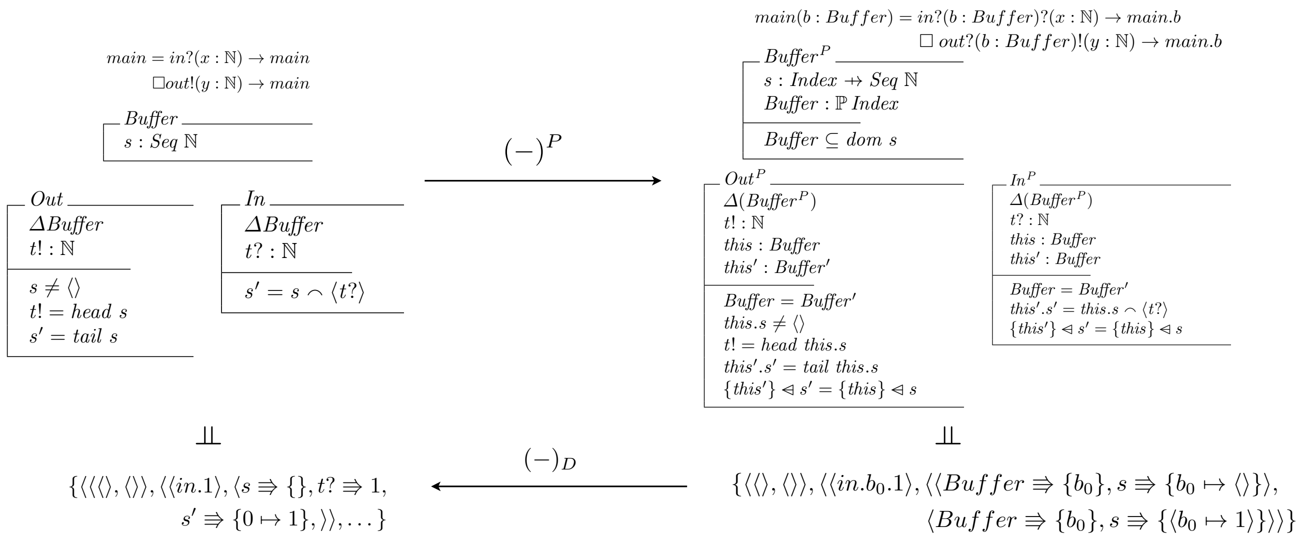

Fig. 21. Promoting CZP specifications.

Z 's structuring mechanisms. In [BM00], the authors propose a set of rules to manipulate Z schemas; as opposed to our work, these rules are motivated as a means for refactoring specifications.

Other work, in particular [HR99, HR99b], proposes the use of higher-order logic to interpret schemas: this has the clear benefit of interpreting schemas as types in a simple way; however, the semantics of specifications obtained in this way does not reflect the structuring of specifications, flattening a modular Z specification into a typed set theory. In addition, the direct interpretation of schemas as types in a higher-order logic introduces difficulties when dealing with operations; in particular, the authors need to change the Z notation to deal with the constructions ΔS and ΞS. Note that, in our framework, operations are schemas that extend ΔS, meaning that they relate pre and post states; furthermore, observing the semantics, and taking into account that in $\mathit{Mod}(\Sigma)$ the arrows go in the other direction, the semantics of an operation is provided by a relationship between states, which coincides with the basic intuition about operations in Z. This, in our opinion, reflects accurately the intuitions associated with the Z notation.

On the other hand, the schema calculus is presented in detail in the standard literature [Woo96, Spi88]; the properties of schema conjunction, disjunction and inclusion are described and illustrated with several examples, but promotion is often, as argued in [Woo90], introduced in an informal way. In [Woo90], the author captures some properties of promotion in a rigorous way; in our opinion the proposed mathematical formulation of promotion does not capture the real power of the technique, which is essentially a mapping between specifications, as proposed in Section 4.

Regarding heterogeneous specifications, special mention should be made of the *Unifying Theories of Programming* (or UTP) [HJ98], a formal framework that makes possible the unification of different programming paradigms and related formalisms, such as imperative programming and process algebra. A nice example of the use of UTP to unify different languages (and their semantics) is given in [OCW09], where UTP is employed to give the semantics of Circus [Woo01], a combination of Z with process algebras (in a CSP style). An interesting aspect of UTP is that it allows for the formalization of the refinement calculus as used in Z; note that we have not included in this paper a treatment of pre/postconditions and the Z refinement calculus in our categorical framework. We leave this as further work. In comparison with our work, UTP provides a technically simpler semantics to Z constructions, by only using notions coming from first and second order logic; the categorical framework described in this paper is technically more involved, using categorical constructions (bicategories, natural transformations, etc) that might not be familiar for most Z users. However, we believe that a key benefit of our framework w.r.t. to the approaches mentioned above is that it provides a transparent (and abstract) semantics to Z: schemas are captured as logical theories, and the structure of modular Z specifications are naturally reflected in their categorical semantics; furthermore, its high level of abstraction enables its combination with other frameworks in a more or less direct way, as shown in Section 5. Other authors have proposed the combination of Z with other formalisms, for instance [Web96, Fis97], but the frameworks proposed in these works are *ad-hoc* and cannot be used in more general settings.

7. Conclusions

We have proposed a mathematical foundation for Z and its structuring mechanisms; this formal framework makes use of well established abstract descriptions 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 [FM92, Fia04] usually found in formal methods. Several alternative approaches to provide a formal semantics for Z can be found in the literature, as explained in Section 6, but we believe that our approach fits better with the original motivations for Z's schema operators, where priming denotes a purely syntactical operation, a syntactic operation also extensively used in other logics for program specification (e.g., in TLA). The interpretation of priming (and related operators) as categorical operations over logical theories provides a simple understanding of Z constructions, with a good separation of concerns between the interpretation of schemas and schema operators, dealing even with promotion, a sophisticated, and widely used, specification structuring mechanism. Moreover, our approach maintains the structure of specifications when providing semantics for them, leading to explicit semantic relationships between component schemas and the composite schemas they are part of, which can be exploited to *promote* reasoning, and with potential benefits for automated reasoning. Finally, our formalisation is at a level of abstraction that allows for a view of logical systems as building blocks. This provides the rigour and flexibility needed to characterise not only Z but also its related languages and extensions, in particular the heterogeneous ones. We have illustrated this point via a formal, well structured, combination of Z with CSP, resulting in a formalism in essence equivalent to the Z-CSP formal method, and "inheriting" the structuring of the composed languages, in particular promotion.

References

[Ris63] *Risk! Rules of Play.* Parker Brothers, 1963.

[Abr96] Abrial, J. R.: *The B-Book, Assigning Programs to Meanings.* Cambridge University Press, 1996.

[BSM04] Baar, T., Strohmeier, A., Moreira, A. and Mellor, S.: *UML 2004.* Lecture Notes in Computer Science, volume 3273. Springer-Verlag, 2004.

[BW99] Barr, M. and Wells, C.: *Category Theory for Computer Science.* Centre de Recherches Mathématiques, Université de Montréal, 1999.

[Bau99] Baumeister, H.: Relating Abstract Datatypes and Z-Schemata. *In Proc. of WADT '99*, Lecture Notes in Computer Science, volume 1827, Springer-Verlag, 1999.

[Ber67] Bérnabou, J.: Introduction to Bicategories. *In Complementary Definitions of Programming Language Semantics*, LNM 42, Springer-Verlag, 1967.

[Bor94] Borceux, F.: *Handbook of Categorical Algebra: Volume 1: Basic Category Theory.* Enc. of Mathematics and its Applications, Cambridge University Press, 1994.

[Bor99] Borzyszkowski, T.: Higher-Order Logic and Theorem Proving for Structured Specifications, *In Proc. of WADT '99*, Lecture Notes in Computer Science, volume 1827, Springer-Verlag, 1999.

[BM00] Brien, S. M. and Martin, A. P.: A Calculus for Schemas in Z. *Journal of Symbolic Computation*, 30(1), Elsevier, 2000.

[Buj04] Bujorianu, M. C.: Integration of Specification Languages Using Viewpoints. *In Proc. of IFM '04*, Lecture Notes in Computer Science, volume 2999, Springer-Verlag, 2004.

[BG77] Burstall, R. and Goguen, J.: Putting Theories together to make Specifications. *In Proc. of Intl. Joint Conference on Artificial Intelligence*, 1977.

[CAPM10] Castro, P. F., Aguirre, N., Lopez Pombo, C. G. and Maibaum, T. S. E.: Towards Managing Dynamic Reconfiguration of Software Systems in a Categorical Setting *In Proc. of ICTAC '10*, Lecture Notes in Computer Science, volume 6255. Springer-Verlag, 2010.

[CAPM12] Castro, P. F., Aguirre, N., Lopez Pombo, C. G., and Maibaum, T. S. E.: A Categorical Approach to Structuring and Promoting Z Specifications, *In Proc. of FACS '12*, Lecture Notes in Computer Science, volume 7684. Springer-Verlag, 2012.

[CK90] Chang, C. C. and Keisler, H. J.: *Model Theory.* 3rd. Ed., North Holland, 1990.

[Dia08] Diaconescu, R.: *Institution-Independent Model Theory.* Birkhäuser Verlag, 2008.

[End01] Enderton, H.: *A Mathematical Introduction to Logic.* 2nd. Ed., Academic Press, 2001.

[Fia04] Fiadeiro, J.: *Categories for Software Engineering.* Springer-Verlag, 2004.

[FM92] Fiadeiro, J. and Maibaum, T. S. E.: Temporal Theories as Modularisation Units for Concurrent System Specification. *Formal Aspects of Computing*, 4(3), Springer-Verlag, 1992.

[FKNG92] Finkelstein, A., Kramer, J., Nuseibeh, B., Finkelstein, L. and Goedicke, M.: Viewpoints: A Framework for Integrating Multiple Perspectives in System Development. *International Journal of Software Engineering and Knowledge Engineering*, 2(1), 1992.

[Fis97] Fischer, C.: Combining CSP and Z. *Technical Report*, University of Oldenburg, 1997.

[GB92] Goguen, J. and Burstall, R.: Institutions: Abstract Model Theory for Specification and Programming. *Journal of the ACM*, 39(1), ACM Press, 1992.

[HR99] Henson, M. and Reeves, S.: Revising Z: Part I - Logic and Semantics. *Formal Aspects of Computing*, 11(4), Springer-Verlag, 1999.

[HR99b] Henson, M. and Reeves, S.: Revising Z: Part II - Logical Development. *Formal Aspects of Computing*, 11(4), Springer-Verlag, 1999.

[HJ98] Hoare, C. A. R. and Jifeng, H.: *Unifying Theories of Programming.* Prentice Hall College Division, 1998.

[Jac97] Jacky, J.: *The Way of Z, Practical Programming with Formal Methods.* Cambridge University Press, 1997.

[Lan09] Lano, K.: *Model-Driven Software Development With UML and Java.* Course Technology, 2009.

[Mac98] MacLane, S.: *Categories for the Working Mathematician.* (second ed.) Springer-Verlag, 1998.

[Mey00] Meyer, B.: *Object-Oriented Software Construction.* Prentice Hall, 2000.

[MML07] Mossakowski, T., Maeder, C. and Lüttich, K.: The Heterogeneous Tool Set (Hets), *In Proc. of 4th International Verification Workshop in connection with CADE-21*, CEUR-WS.org, 2007.

[Nic95] Nicholls, J.: *Z Notation: Version 1.2.* Z Standards Panel, 1995.

[MTP97] Mossakowski, T., Tarlecki, A. and Pawlowski, W.: Combining and Representing Logical Systems, *In Proc. of Category Theory and Computer Science '97*, Lecture Notes in Computer Science, volume 1290, Springer-Verlag, 1997.

[MR06] Mossakowski, T. and Roggenbach, M.: Structured CSP - A Process Algebra as an Institution, *In Proc. of WADT '06*, Lecture Notes in Computer Science, volume 4409, Springer-Verlag, 2006.

[OCW09] Oliveira, M., Cavalcanti, A. and Woodcock, J.: A UTP Semantics for Circus. *Formal Aspects of Computing*, 21(2), Springer-Verlag, 2009.

[Par72] Parnas, D.: In the Criteria to be Used in Decomposing Systems into Modules. *Commun. ACM*, 15(12), 1972.

[Par85] Parnas, D.: The Modular Structure of Complex System. *IEEE Trans. Software Eng.*, 11(3), 1985.

[Smi00] Smith, G.: *The Object Z Specification Language.* Advances in Formal Methods Series, Kluwer Academic Publishers, 2000.

[Spi84] Spivey, J. M.: *Towards a Formal Semantics for the Z Notation.* Oxford University Computing Laboratory, T.M. PRG-41, 1984.

[Spi88] Spivey, J. M.: *Understanding Z: A Specification Language and its Formal Semantics.* Cambridge Tracts in Theoretical Computer Science, 1988.

[Spi92] Spivey, J. M.: *The Z Notation: A Reference Manual.* Prentice Hall, 1992.

[Tar95] Tarlecki, A.: Moving Between Logical Systems, *In Proc. of ADT/COMPASS '95*, Lecture Notes in Computer Science, volume 1130, Springer-Verlag, 1995.

[Web96] Webber, M.: Combining Statecharts and Z for the Design of Safety-Critical Control Systems, *In Proc. of FME '96*, Lecture Notes in Computer Science, volume 1051, Springer-Verlag, 1996.

[Woo90] Woodcock, J.: Mathematics as a Management Tool: Proof Rules for Promotion, *In Software Engineering for Large Software Systems*, Springer-Verlag Netherlands, 1990.

[Woo96] Woodcock, J. and Davies J.: *Using Z: Specification, Refinement, and Proof.* Prentice Hall, 1996.

[Woo01] Woodcock, J. and Cavancanti A.: Circus: A Concurrent Refinement Language. *Technical Report*, Oxford University Computing Laboratory, Oxford, UK, 2001.

A. Further details on category theory definitions

In this section we introduce further details about some categorical notions introduced in Section 2.

A.1. Monoidal categories

A monodical category $\langle \mathbf{C}, \oplus, id \rangle$ has natural isomorphisms:

- $\alpha_{A,B,C} : A \otimes (B \otimes C) \to (A \otimes B) \oplus C$ for objects A, B, C,
- $\rho_A : A \otimes 1 \to A$,
- $\lambda_A : 1 \otimes A \to A$,

Such that the following diagrams commute, the triangle diagram:

$$\begin{tikzcd}
A \otimes (1 \otimes B) \arrow[rr, "\alpha(A,1,B)"] \arrow[dr, "A\otimes\lambda_B"'] & & (A \otimes 1) \otimes B \arrow[dl, "\rho_A \otimes B"] \\
& A \otimes B &
\end{tikzcd}$$

and the pentagon diagram:

$$\begin{tikzcd}
& A \otimes (B \otimes (C \otimes D)) \arrow[dl, "id_A\otimes\alpha_{B,C,D}"'] \arrow[dr, "\alpha_{A,B,C\otimes D}"] & \\
A \otimes ((B \otimes C) \otimes D) \arrow[d, "\alpha_{A,B\otimes C,D}"'] & & (A \otimes B) \otimes (C \otimes D) \arrow[d, "\alpha_{A\otimes B,C,D}"] \\
(A \otimes (B \otimes C)) \otimes D \arrow[rr, "\alpha_{A,B,C\otimes D}"'] & & ((A \otimes B) \otimes C) \otimes D
\end{tikzcd}$$

Roughly speaking, these diagrams state that $\otimes$ is associative (up to isomorphism) and it has a unit (up to isomorphism). An excellent introduction to monodical categories (and their application to computer science) is given in [BW99].

A.2. Bicategories

Let us give the coherence conditions for bicategories. Given a bicategory $\mathbf{V}$, the coherence conditions are given by natural transformations:

- $\alpha_{A,B,C,D} : ;_{A,B,C} \circ (Id \times ;_{B,C,D}) \dot{\to} ;_{A,C,D} \circ (;_{A,B,C} \times Id)$,
- $\lambda_{A,B} : ;_{A,B,B} \times (Id \times 1_B) \dot{\to} Id$ and
- $\rho :_{A,B} : ;_{A,A,B} \circ (1_A \times Id) \dot{\to} Id$.

Intuitively, α (called the associator) shows that ; is associative up to isomorphism, and ρ and λ show that 1_A is an identity up to isomorphism; these natural transformation are subject to the following coherence laws expressed by the commutativity of the following diagrams:

$$\begin{tikzcd}
p\,;(q\,;(r\,;\,t)) \arrow[rr, "id_p\,;\,\alpha"] \arrow[d, "\alpha"'] & & p\,;((q\,;\,r)\,;\,t) \arrow[d, "\alpha"] \\
(p\,;\,q)\,;(r\,;\,t) \arrow[dr, "\alpha"'] & & (p\,;(q\,;\,r))\,;\,t \arrow[dl, "\alpha\,;\,id_t"] \\
& ((p\,;\,q)\,;\,r)\,;\,t &
\end{tikzcd}$$

and:

$$\begin{tikzcd}
p\,;(1_B\,;\,q) \arrow[rr, "\alpha"] \arrow[dr, "id_p\,;\,\rho"'] & & (p\,;\,1_B)\,;\,q \arrow[dl, "\lambda\,;\,id_q"] \\
& p\,;\,q &
\end{tikzcd}$$

where $p : A \to B, q : B \to C, r : C \to D$ and $t : D \to E$ are 1-cells of the bicategory.

Now, we give the a detailed definition of *lax functor*. Given two bicategories $\mathbf{A}$ and $\mathbf{B}$, a *lax functor* $F : \mathbf{A} \to \mathbf{B}$ between them is composed of:

- For every object $A \in |\mathbf{A}|$, an object $F(A) \in |\mathbf{B}|$,
- For every pair of objects $A, B \in |\mathbf{A}|$, a functor $F_{A,B} : \mathbf{A}(A,B) \to \mathbf{B}(F(A), F(B))$,
- For every triple of objects $A, B, C \in |\mathbf{A}|$, a natural transformation $\gamma_{A,B,C} : ;_{F(A),F(B),F(C)} \circ (F_{A,B} \times F_{B,C}) \to F_{A,C} \circ ;_{A,B,C}$,
- For every object $A \in |\mathbf{A}|$, a natural transformation $\delta_A : 1_{F(A)} \to F_{A,A} \circ 1_A$. Where 1_A is the unit object of $\mathbf{A}$.

Subject to the coherence laws expressed by the commutativity of the following diagrams:

$$\begin{tikzcd}
F(f)\,;\,(F(g)\,;\,F(h)) \arrow[r, "id\,;\,\gamma"] \arrow[d, "\alpha"'] & F(f)\,;\,F(g\,;\,h) \arrow[r, "\gamma"] & F(f\,;\,(g\,;\,h)) \arrow[d, "F(\alpha)"] \\
(F(f)\,;\,F(g))\,;\,F(h) \arrow[r, "\gamma\,;\,Id"'] & F(f\,;\,g)\,;\,F(h) \arrow[r, "\gamma"'] & F((f\,;\,g)\,;\,h)
\end{tikzcd}$$

and:

$$\begin{tikzcd}
1_{F(A)}\,;\,F(p) \arrow[d, "\delta\,;\,Id"'] \arrow[dr, "\rho"] & & F(p)\,;\,1_{F(B)} \arrow[dl, "\lambda"'] \arrow[d, "id\,;\,\delta"] \\
F(1_A)\,;\,F(p) \arrow[d, "\gamma"'] & F(p) & F(p)\,;\,F(1_B) \arrow[d, "\gamma"] \\
F(1_A\,;\,p) \arrow[ur, "F(\rho)"'] & & F(p\,;\,1_B) \arrow[ul, "F(\lambda)"]
\end{tikzcd}$$