TOWARDS A GeoZ TOOLKIT

MARK DAWSON and STEVEN VICKERS
Department of Computing, Imperial College
180 Queen's Gate, London SW7 2BZ, United Kingdom
E-mail: {md,sjv}@doc.ic.ac.uk

ABSTRACT

The use of Geometric Logic as the foundation of a specification language called GeoZ is proposed elsewhere [4]. In this note we explore GeoZ from the perspective of practitioners, who are familiar with the existing Z notation, by explaining the issues that arise and the essential role of schema entailment in the GeoZ reformulation of Z's mathematical toolkit.

1 Introduction

In a companion paper [4], geometric logic is proposed as a logic suited to specification. Various aspects of this are described, including a notion of "schema entailment" that asserts unique (up to isomorphism) existence and hence can be used for definitions.

The aim of this paper is to show how such schema entailments can be used in describing a mathematical toolkit similar to that in Z [3]. The purpose of the toolkit is to provide an interface between mathematical "experts", who can prove unique existence and thereby establish the validity of schema entailments, and practising "engineers" who use the entailments in building up their specifications. Each schema entailment is therefore intended, like a good theorem, to package up some difficult mathematics in a useful and usable form.

It is hoped also that expert-verified schema entailments can provide the basis for extending software support tools, geometric logic environments, regardless of whether the verification has been carried out formally within the logic environment or not.

In GeoZ "schema" is a presentation of a (geometric) theory. As such it contains three kinds of definition:

1. A declaration of sorts (X, Y, Z).

2. The names and sorts of predicates, functions, propositions, and constants (P, f, c) appearing in the schema. Together with variables and the logical connectives, these are used to form formulas in the usual way.

3. A collection of axioms ($\phi \Longrightarrow_X \psi$). These are entailments between the geometric formulas ϕ and ψ.

A schema is presented with a typographic layout[1] that should be familiar to readers acquainted with the Z notation –

$$\begin{array}{|l}
\textit{Name}[X, Y, Z] \\
\hline
P : \mathbb{P}\, X \\
f : X \to Y \\
c : X \\
\hline
\phi \Longrightarrow_{x:X} \psi \\
\hline
\end{array}$$

The "*Name*" is optional and is omissible in many contexts. It gives a global name by which to refer to the schema.

Although this is described in detail in [4] we should remind ourselves that geometric logic is based on a positive, observational account of meaning which restricts its formulas to: (finite) conjunctions, (infinite) disjunctions, existential quantification and equality. Despite being prohibited from formulas, universal quantification, implication, negation and infinite conjunctions are can be found in the axioms of a theory in the following sense –

$\phi \Longrightarrow_{x:X} \psi$	implication ($\Longrightarrow$); universal quantification ($x : X$)
$\phi \Longrightarrow$ false	negation ($\neg\ \phi =_{def} \phi \Longrightarrow$ false)
$\phi_n \Longrightarrow \psi_n$	axiom schemas – infinite conjunctions

The language of a schema consists of an assignment of *arities* to the symbols in the presentation. At this level, and not elsewhere, function space ($\to$) and powerset ($\mathbb{P}$) operations are admissible and allow us to introduce functions and predicates –

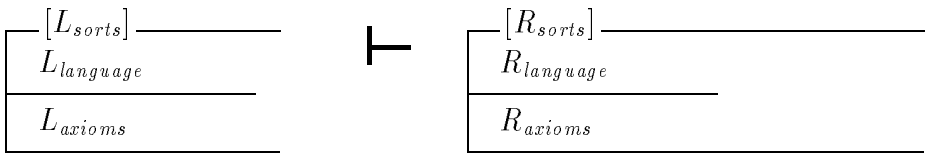
$$\begin{array}{|l}
f : X \to Y \\
P : \mathbb{P}\, X
\end{array}$$

The sorts X and Y must be geometric – in fact any geometric formula gives rise to a subsort through a formula-as-types principle and we take advantage of this in the toolkit.

[1] The syntax of many of the constructions presented in this paper is fluid – we are still experimenting and it is likely to evolve, and perhaps improve, as syntactic issues become clearer.

Schema Entailment

Of course there are many ways of *presenting* essentially the same theory and it is useful to be able to state when two schemas present the same theory. The *schema entailment* allows us to do this in GeoZ –



The schema "L" on the lefthand side of the entailment ($\vdash$) is used as the basis of a *superschema* "$L+R$" which contains the additional material presented in the schema "R". Clearly the combination of L with R will not make sense in general – indeed it need not be consistent. However, when R *definitionally extends* L a model of L will also be a model of $L + R$ and there will be an essentially unique way of interpreting the additional material provided by R in L – for further details of schema entailment see [4]. This means that a schema entailment is a powerful definitional tool because it asserts that R can be uniquely defined in terms of L – often in rather subtle ways.

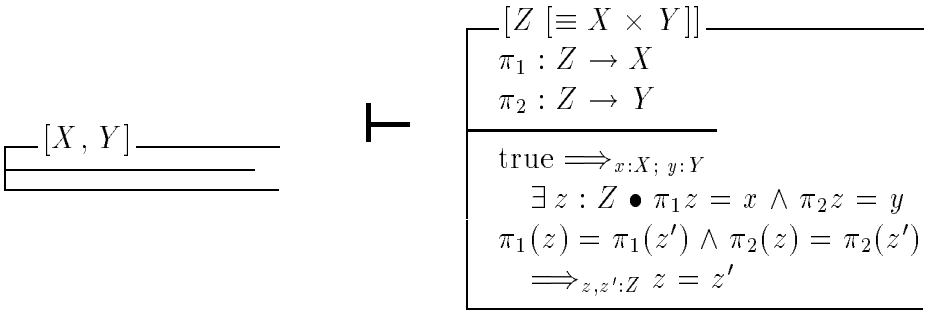
The GeoZ *Toolkit*

It is intended that (software) engineers who write specifications in GeoZ will be able to consult a library (or toolkit) of mathematical contructions, methods and notations each of which has been expressed as an appropriate schema extension and whose validity has been established by an "expert". The purpose of this paper is to examine the issues that an engineer, and to a lesser extent an expert, has to face when using (and extending) the toolkit.

2 Generic Constructions

The Z notation has a notion of parameterised construction (the generic schema) which allows families of concepts to be captured in a single definition [3], p. 38. Schema entailments provide this function in GeoZ.

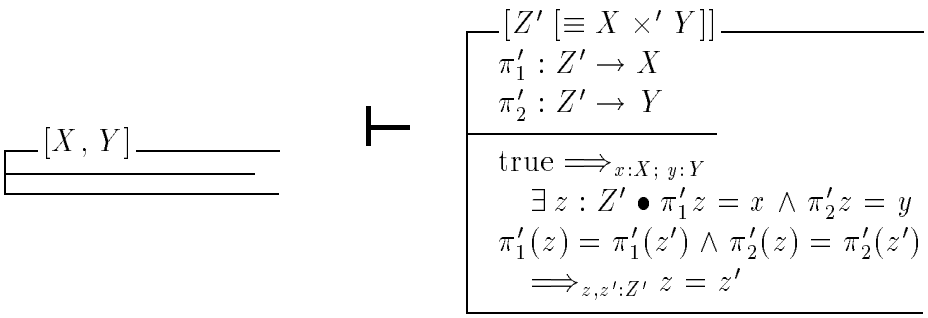
The definition of Cartesian products provides a suitable starting point. In the GeoZ toolkit products are expressed by definitional extension of the theory containing two sorts X and Y.



Notice that the extension introduces some new syntax: the projection functions π_1 and π_2 and, more importantly, a derived sort Z that we have chosen to call "$X \times Y$". These ingredients are governed by the two axioms in the "predicate" part. The syntactic name "$X \times Y$" for Z allows us to refer to the construction so that when we write "$X \times Y$" or "$A \times B$" we implicitly include an appropriate instance of the construction in the context in which the reference is made. We can always do this safely (the two sorts must be present for us to write down the name) because the effect of the new material is *definitional*.

The first axiom states that given a pair x of X and y of Y of observations we can observe some z of Z from which it is possible to extract both x and y. Which is the expected observational content of an element of the product. The second axiom expresses equality of objects in Z in terms of their constituents.

Before the schema entailment can be used by engineers an expert must show that there is an essentially unique way in which the extension schema can be interpreted in a model of the premiss schema. In this case suppose that there was some other extension having the same presentation but renamed –



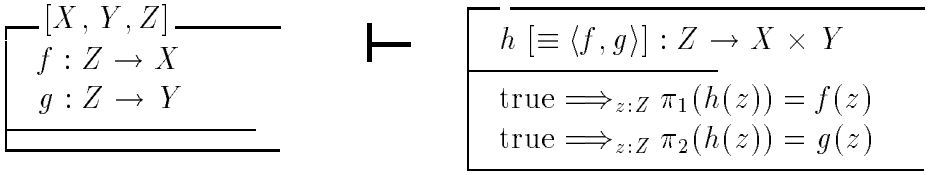
then both the derived sorts, Z and Z', must be isomorphic. Define a relation between the two sorts $\_ \equiv \_ : \mathbb{P}(Z \times Z')$ as –

$$z \equiv z' \Longleftrightarrow_{z:Z;\, z':Z'} \quad \exists\, x : X \bullet (\pi_1(z) = x \wedge \pi'_1(z') = x) \\ \wedge\, \exists\, y : Y \bullet (\pi_2(z) = y \wedge \pi'_2(z') = y)$$

The expert must show –

$$\forall z : Z \bullet \exists z' : Z' \bullet z \equiv z'$$
$$\forall z : Z \bullet \forall z'_1, z'_2 : Z' \bullet z \equiv z'_1 \wedge z \equiv z'_2 \rightarrow z'_1 \equiv z'_2$$

Once we have introduced the derived sort of products we are able to use it to make additional definitions of notation –



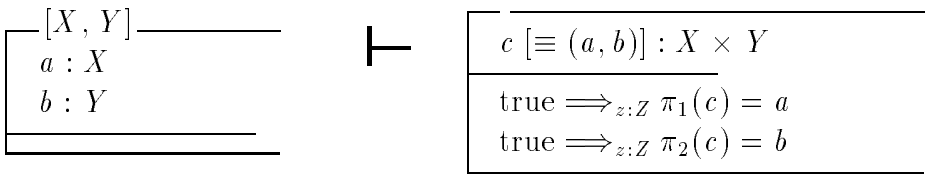
Here h is introduced as a definitional extension of the premiss schema containing two functions f and g of the specified types. Notice that we have given h the name "$\langle f, g \rangle$" which refers to f and g. When this syntax is used in a schema some additional type checking must be performed to ensure that the types X, Y and Z are assigned correctly. To show that the extension is unique suppose there is some other h' satisfying the axioms, then from the axioms we have –

$$\text{true} \Longrightarrow_{z:Z} \pi_1(h(z)) = f(z) = \pi_1(h'(z)) \wedge \pi_2(h(z)) = g(z) = \pi_2(h'(z))$$

Which, by the second axiom of products, gives us what we require –

$$\text{true} \Longrightarrow_{z:Z} h(z) = h'(z)$$

Another example is pair formation –



This supposes that a and b exist as objects of the sorts X and Y respectively and introduces a notation "(a, b)" for c. Again this schema entailment encapsulates the preconditions needed in the premiss schema before the construction can be used. In this case a simple form of type checking is involved, but as we will see, when axioms are present other "proof obligations" must be discharged.

3 Decidability and Complementation

As we said in the introduction neither inequality ($\neq$) nor negation ($\neg$) are built-in to geometric logic at the level of formulas. However, there are places both in the toolkit and in actual specifications where these properties are needed. The solution is straightforward: when we need to use inequality we must to have a *decidable* sort by which we mean –

Decidable$[X]$

$\_ \neq \_ : X \rightharpoonup X$

$x \neq x \Longrightarrow_{x:X}$ false

true $\Longrightarrow_{x,y:X}$ $x = y \lor x \neq y$

Once we have this we can define, for example, non-membership on finite sets of a sort X –

$[X]$

Decidable$[X]$

$\vdash$

$\_ \notin \_ : X \rightharpoonup \mathbb{F}\, X$

$x \notin S \Longleftrightarrow_{x:X;\ S:\mathbb{F}\, X} \forall\, y \in S \bullet x \neq y$

The schema inclusion "*Decidable*$[X]$" is a *precondition* for the applicability of the entailment – the operator "$\notin$" cannot be used if the sort X doesn't satisfy "*Decidable*$[X]$".

In a similar way when we need to know the complement of a predicate P on a sort X we can write –

Complements$[X]$

$P, P' : \mathbb{P}\, X$

$P(x) \land P'(x) \Longrightarrow_{x:X}$ false

true $\Longrightarrow_{x:X}$ $P(x) \lor P'(x)$

An example is the definition of the difference of two relations R and S on X which requires the complement S' of S –

$[X]$

$S, S', P : \mathbb{P}\, X$

Complements$[X, S, S']$

$\vdash$

$R \setminus S : \mathbb{P}\, X$

$(R \setminus S)(x) \Longleftrightarrow_{x:X} R(x) \land S'(x)$

4 Characterising the Natural Numbers

We give an approach to the definition of the natural numbers ($\mathbb{N}$) in this section, somewhat along the lines of the definition of $\mathbb{F}\, X$ in [4]. This illustrates a general technique of definition which can be applied to other toolkit types.

Let us call an *induction algebra* a set equipped with a constant and a unary operator. The $\mathbb{N}$ is the initial induction algebra, with constant 0 and unary operation s, the successor opertion. We can specify it using infinitary disjunctions as in the following schema entailment. The premiss is the empty theory, which means that $\mathbb{N}$ comes from the world of pure abstract mathematics.

$$\vdash \quad [Z\ [\equiv \mathbb{N}]]$$
$$0 : Z$$
$$\mathsf{s}\_ : Z \to Z$$
$$0 = \mathsf{s}(x) \Longrightarrow_{x:Z} \text{false}$$
$$\mathsf{s}(x) = \mathsf{s}(y) \Longrightarrow_{x,y:Z} x = y$$
$$\text{true} \Longrightarrow_{x:Z} \bigvee_{n=0}^{\infty} \exists\, x_1, \ldots, x_n \bullet x = x_n \land x_0 = 0 \land \bigwedge_{i=1}^{n} x_i = \mathsf{s}(x_{i-1})$$

The extension introduces the necessary language and then defines how terms are constructed with an infinite disjunction of the possibilities. It also gives equations which define equality among the terms of the type. In this case we need to state that 0 is not the successor of any number and that two numbers are equal when they are the successors of equal numbers.

The freeness property, after which we do not need to know the infinitary axioms given above, is as follows:

$$[X]$$
$$x_0 : X$$
$$t : X \to X$$
$$\vdash \quad f\ [\equiv \text{iter}(t, x_0)] : \mathbb{N} \to X$$
$$\text{true} \Longrightarrow f(0) = x_0$$
$$\text{true} \Longrightarrow_{n:\mathbb{N}} f(\mathsf{s}(n)) = t(f(n))$$

The function "iter" maps $n \mapsto t^n(x_0)$.

Actually, it's well-known that $\mathbb{N}$ *cannot* be characterized up to isomorphism by axioms of first-order logic, but the trick here is the infinitary disjunction. Of course, it looks rather like cheating – in defining $\mathbb{N}$ we have presupposed its existence in the set of disjuncts. Really, the schema entailments used here to

characterize $\mathbb{N}$ are postulating its existence rather than provably asserting it, but once we have it then the existence of other free algebras can be proved.

However, these mathematical soul-searchings of the experts can be ignored by the practising engineers. What the schema entailment assures them is that when they write down a recursive definition such as

$$f(0) = x_0$$
$$f(n+1) = t(f(n))$$

then it really does define a function.

Mathematical Induction

Mathematical induction principles can also be expressed using schema entailments. Here is simple induction:

$$\frac{P : \mathbb{P}\,\mathbb{N}}{\text{true} \Longrightarrow P(0) \qquad P(n) \Longrightarrow_{n:\mathbb{N}} P(s(n))} \vdash \frac{}{\text{true} \Longrightarrow_{n:\mathbb{N}} P(n)}$$

Similarly one can describe course of values induction, using the induction step $\forall\, m \in [0 \ldots n) \bullet P(m) \Longrightarrow_{n:\mathbb{N}} P(n)$, once $[0 \ldots \_) : \mathbb{N} \to \mathbb{F}\,\mathbb{N}$ has been defined so that $[0 \ldots n) = \{m : \mathbb{N} \bullet 0 \leq m < n\}$. Here is a somewhat more unusual induction principle that appears to be a geometric form of well-founded induction:

$$\frac{P : \mathbb{P}\,\mathbb{N} \qquad \phi : \mathbb{P}(\mathbb{N} \times \mathbb{N})}{\phi(m,n) \Longrightarrow_{m,n:\mathbb{N}} m < n \qquad \text{true} \Longrightarrow_{n:\mathbb{N}} P(n) \vee \exists\, m : \mathbb{N} \bullet \phi(m,n) \qquad P(m) \wedge \phi(m,n) \Longrightarrow_{m,n:\mathbb{N}} P(n)} \vdash \frac{}{\text{true} \Longrightarrow_{n:\mathbb{N}} P(n)}$$

Proof $<$ is well-founded, so we can use well-founded induction. We prove $P(n)$ under the induction hypothesis $\forall\, m : \mathbb{N} \bullet (m < n \Rightarrow P(m))$. By the second axiom, either we have $P(n)$ already, or there is some m with $\phi(m,n)$ and hence $m < n$. Hence by induction $P(m)$ and so $P(n)$ by the third axiom. $\square$

The same argument applies for other well-founded orders, such as "lexicographic order" on $\mathbb{N} \times \mathbb{N}$ – $(m_1, m_2) < (n_1, n_2)$ iff $m_1 < n_1$ or $m_1 = n_1$ and $m_2 < n_2$. It seems that constructively it is a weaker principle than that of well-founded induction, but nonetheless it suffices in practical examples. (Classically, they are equivalent. For suppose $<$ is a relation on a set X, satisfying the above induction principle. Let $S \subseteq X$, and define $P(x)$ to mean either $x \notin S$ or there is a descending chain $x_n > \ldots > x_0$, all in S, with $x = x_n$ and x_0 minimal (with respect to $<$) in S. Define $\phi(x, y)$ to mean that $x < y$ and x and y are both in S. Given x, if $\neg\ \exists\, y : X \bullet \phi(y, x)$ then either $x \notin S$ or x is minimal in S, and either way we have $P(x)$. If $P(y)$ and $\phi(y, x)$ then $y \in S$ so we have a descending chain $y = y_n > \ldots > y_0$ all in S, with y_0 minimal in S, and $x \in S$, so there is a similar chain $x > y_n > \ldots$ from x, so $P(x)$. It follows from the induction principle that $\forall\, x : X \bullet P(x)$. If S is non-empty then choose $x \in S$. $P(x)$, so we have $x = x_n > \ldots > x_0$ with x_0 minimal in S. It follows that every non-empty subset S has a minimal element, and this condition is equivalent to well-foundedness of $<$.)

5 Finite Sets

The flexibility with which one can handle sets in classical mathematics (and in Z) is severely constrained in geometric logic – it is restricted to *finite* sets.[2] Only finite sets can be elements of types – this is enforced by the fact that whereas $\mathbb{F}\, X$ is a type, $\mathbb{P}\, X$ is only an arity – or universally quantified over, and observationally, only finite sets can be successfully apprehended. The GeoZ toolkit has much to say about finite sets because it is only with them that we have the flexibility that Z attributes to *all* sets, and in this paper we shall only be able to scratch the surface.

Let us first remark that even for finite sets, the geometric logic causes some odd phenomena, generally connected with decidability issues. Most notoriously, *a subset of a finite set need not be finite.* Specifically, consider a finite set $S : \mathbb{F}\, X$ and a predicate $P : \mathbb{P}\, X$. You might expect there to be a finite set corresponding to $\{x \in S \bullet P(x)\}$, but to apprehend that subset by knowing all its elements we have to know not only which elements of S satisfy P, and hence are in the subset, but also which do *not* satisfy P and so are to be excluded. This can usually only be done if P has a complement. Similarly, the intersection of two finite sets need not be a finite set. This is because, unlike the operation of union which can combine two sets without concern for what they contain (repetitions don't matter), the operation of intersection must compare the elements of the two sets to decide which are in common and belong in the intersection and

[2] Constructively, there are more than one notion of finite sets. The ones we refer to here are the *Kuratowski finite* sets.

which are not and should be excluded. The result of this process is a finite set when the underlying sort has a decidable equality.

The finite powerset $\mathbb{F}\,X$ was defined in [4]; let us from that derive some schema entailments that illustrate well the distinction between expert mode and engineer mode. First, we give a *Principle of Simple* $\mathbb{F}$*-induction.*

$$\begin{array}{|l}\hline [X] \\ P : \mathbb{P}\,\mathbb{F}\,X \\ \hline \text{true} \Longrightarrow P(\varnothing) \\ P(S) \Longrightarrow_{x:X;\ S:\mathbb{F}\,X} P(\{x\} \cup S) \\ \hline \end{array} \quad \vdash \quad \begin{array}{|l}\hline \\ \hline \text{true} \Longrightarrow_{S:\mathbb{F}\,X} P(S) \\ \hline \end{array}$$

Proof Let $U = \{T \in \mathbb{F}\,X \bullet \forall\, S \in \mathbb{F}\,X \bullet (P(S) \to P(T \cup S))\}$. This is a submonoid of $\mathbb{F}\,X$, and the second axiom tells us that it contains the generators $\{x\}$ and hence is the whole of $\mathbb{F}\,X$. But from $P(\varnothing)$ we now deduce that $P(S)$ for all S. □

Next, we give a means for defining functions on $\mathbb{F}\,X$ by recursion:

$$\begin{array}{|l}\hline [X, Y] \\ y_0 : Y \\ f : X \times Y \to Y \\ \hline \text{true} \Longrightarrow_{x:X;\ y:Y} \\ \quad f(x, f(x, y)) = f(x, y) \\ \text{true} \Longrightarrow_{x,x':X;\ y:Y} \\ \quad f(x, f(x', y)) = f(x', f(x, y)) \\ \hline \end{array} \quad \vdash \quad \begin{array}{|l}\hline h : \mathbb{F}\,X \to Y \\ \hline \text{true} \Longrightarrow h(\varnothing) = y_0 \\ \text{true} \Longrightarrow_{x:X;\ S:\mathbb{F}\,X} \\ \quad h(\{x\} \cup S) \\ \qquad = f(x, h(S)) \\ \hline \end{array}$$

Proof Let X, Y, y_0 and f be a model for the premiss schema. Our models are allowed to be in any G-frames, and these are elementary toposes. This allows us to use the full strength of intuitionistic logic, including higher-order types and the subobject classifier. We shall use Y^Y, which is a monoid under composition. First, we show that the premiss entails the extension

$$\begin{array}{|l}\hline g : \mathbb{F}\,X \times Y \to Y \\ \hline \text{true} \Longrightarrow_{y:Y} g(\varnothing, y) = y \\ \text{true} \Longrightarrow_{y:Y;\ S,T:\mathbb{F}\,X} \\ \qquad g(S \cup T, y) = g(S, g(T, y)) \\ \text{true} \Longrightarrow_{x:X;\ y:Y} g(\{x\}, y) = f(x, y) \\ \hline \end{array}$$

If g can be found satisfying the given axioms, then its curried form $cur\ g : \mathbb{F}\,X \to Y^Y$ is a monoid homomorphism and so is determined uniquely by its action on the generators $\{x\}$; and the axioms define this by $cur\ g(\{x\})(y) = f(x,y)$. Hence we just need to prove existence.

f curries to give a function $cur\ f : X \to Y^Y$. Let M be the submonoid generated by the image X' of $cur\ f$. The first axiom tells us that the generators $cur\ f(x)$ commute, and we deduce from that that M is commutative. For let C be the centralizer in Y^Y of X', *i.e.* the set of functions f that commute with all the elements of X'. C is a submonoid of Y^Y that contains X', and hence contains M – all elements of M commute with all elements of X'. Now let C' be the centralizer in Y^Y of M, a submonoid of Y^Y that contains X' and hence M. Hence M is commutative. Now let M' the set of idempotents in M, those elements ϕ such that $\phi \circ \phi = \phi$. Given that M is commutative, it follows that M' is a submonoid, and it contains X'. Hence $M' = M$, so M is a semilattice. It follows that there is a unique monoid homomorphism $f' : \mathbb{F}\,X \to M$ such that $f'(\{x\}) = cur\ f(x)$. g is defined by $g(S,y) = f'(S)(y)$.

Now we define $h(S) = g(S, y_0)$, and uniqueness follows by $\mathbb{F}$-induction. □

The justification for this schema entailment was somewhat tricky and is done in expert mode. But once proved, it gives what is in effect a programming construct in engineer's mode: functions $h : \mathbb{F}\,X \to Y$ can be defined recursively in the form

$$h(0) = y_0$$
$$h(\{x\} \cup S) = f(x, h(S))$$

provided that two proof obligations are met,

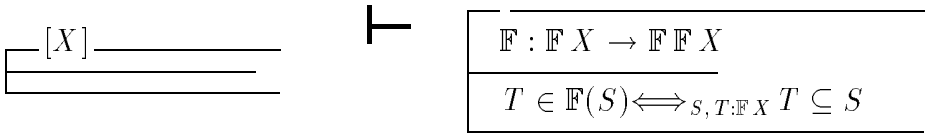
$$f(x, f(x', y)) = f(x', f(x, y))$$
$$f(x, f(x, y)) = f(x, y)$$

The recursive style is similar to that of [2], where functions like h are defined using a non-deterministic "choose" function that chooses an element of a set: something like

$$\begin{array}{lcll} h(S) & = & y_0, & \text{if } S = 0 \\ & = & f(x, h(remove(x,S))), & \text{otherwise} \\ & & \quad \text{where } x = choose(S) & \end{array}$$

However, our proof obligations remove the non-determinism.

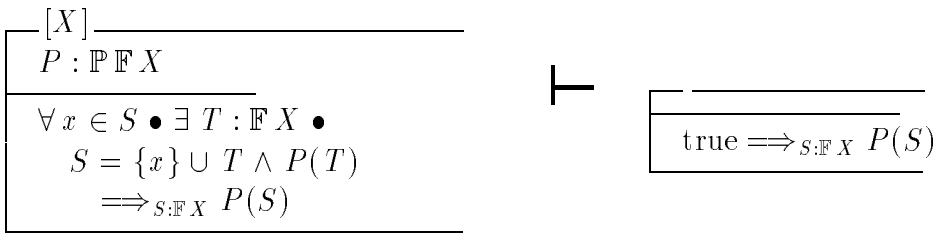
Such principles of recursion and induction are powerful tools for handling finite sets. As an illustration, consider the (classically obvious) fact that if S is finite then so is $\mathbb{F}\,S$. Observationally, if $S = \{x_1, \ldots, x_n\}$, then the elements of $\mathbb{F}\,S$ can be written out as 2^n expressions. The result can be expressed as a schema entailment:



(Of course, $T \subseteq S$ means $\forall\, x \in T \bullet x \in S$.)

Now this specifies what is wanted, including the finiteness property. However, the schema entailment still requires proof and this can be supplied through a recursive definition of the function $\mathbb{F}$.

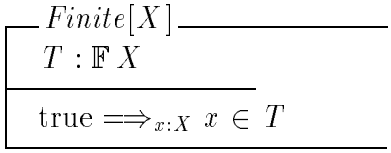
Let us finish with another novel induction principle, the *Principle of Strong* $\mathbb{F}$*-induction.* What makes it stronger is that the induction hypothesis, $\forall\, x \in S \bullet \exists\, T : \mathbb{F}\, X \bullet (S = \{x\} \cup T = P(T))$, implies $S = \varnothing \vee \exists\, x \in S \bullet \exists\, T : \mathbb{F}\, X \bullet (S = \{x\} \cup T = P(T))$, which is a bundled up form of the base case and the simple induction hypothesis. Since the induction hypothesis is stronger, induction proofs are easier to find.



Proof Exercise in expert mode! □

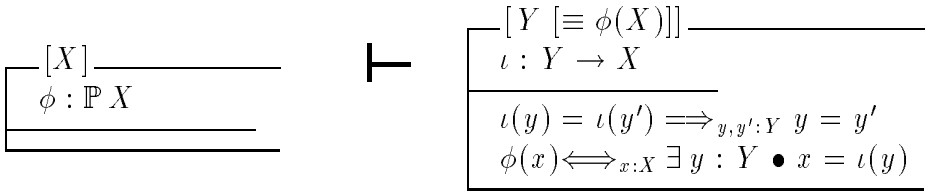
Finiteness

A sort is finite if it is contained in its own finite powerset.



Subtypes

We mentioned in the introduction that a geometric formula ϕ defines a subtype. Here the derived type $\phi(X)$ is the sort consisting of the elements of X that satisfy ϕ.



The function ι is the injection that ϕ induces from the subtype to the supertype.

6 Subarities and Functions

Functions are generally considered to be a particular kind of relation, namely the total, single-valued ones, so a function arity $X \to Y$ is in some sense a *subarity* of the relation arity $\mathbb{P}(X \times Y)$ (usually written $X \rightarrowtail Y$ in Z). Logically, a subarity corresponds to extra axioms (rather in the same way as a subtype corresponds to a formula to be satisfied): so a relation $R : X \rightarrowtail Y$ is a function if it satisfies the axioms

$$\begin{array}{ll} \text{true} \Longrightarrow_{x:X} \exists\, y : Y \bullet xRy & \text{(R is total)} \\ xRy \wedge xRy' \Longrightarrow_{x:X;\, y,y':Y} y = y' & \text{(R is single-valued)} \end{array}$$

(Some syntactic issues are glossed over by this account. Syntactically, Z allows general relations to be applied to terms, $R(x)$ being "undefined" unless there is a unique y such that xRy. We believe that the benefits of this syntactic flexibility are outweighed by the semantic problems, and prefer to admit $R(x)$ as well-formed only when R is declared to be of function type. Thus for us, use of the function type subarity has syntactic implications not carried by using a relation type with totality and single-valuedness axioms. But let us stress that this is purely a matter of syntax – all uses of function application can be expressed equivalently using the logic of relations.)

Z has various notations for different kinds of relations, based on different combinations of four axioms: totality and single-valuedness, as above, and also –

$$\begin{array}{ll} \text{true} \Longrightarrow_{y:Y} \exists\, x : X \bullet xRy & \text{(R is surjective)} \\ xRy \wedge x'Ry \Longrightarrow_{x,x':X;\, y:Y} x = x' & \text{(R is injective)} \end{array}$$

Since all four axioms are geometric, the combinations give subarities in our sense. For instance, $f : X \nrightarrow Y$ declares f to be a *partial function* from X to Y, *i.e.* a single-valued relation.

Subarities for other Z function symbols are constructed in the same fashion according to the table below:

Subarity	*Special name*	*Single − valued*	*Total*	*Injective*	*Surjective*
$\leftrightarrow$	*relation*	$\times$	$\times$	$\times$	$\times$
$\pfun$	*partial function*	✓	$\times$	$\times$	$\times$
$\fun$	*function*	✓	✓	$\times$	$\times$
$\pinj$		✓	$\times$	✓	$\times$
$\inj$	*injection*	✓	✓	✓	$\times$
$\psurj$		✓	$\times$	$\times$	✓
$\surj$	*surjection*	✓	✓	$\times$	✓
$\bij$	*bijection*	✓	✓	✓	✓

Finite functions

If we want to apprehend functions for ourselves (as opposed to being given them in a signature) then they need to be finite relations and to be elements of a sub*type* of $\mathbb{F}(X \times Y)$. Certain conditions must hold before the subarity axioms correspond to subtype formulae.

For instance, if X is decidable, then single-valuedness of a finite relation R can be expressed by the formula

$$\forall (x, y) \in R \bullet \forall (x', y') \in R \bullet x \neq x' \lor y = y'$$

By formulae-as-types, this can be considered a type $X \pfun Y$, of partial finite functions, a subtype of $\mathbb{F}(X \times Y)$.

Similarly, if X is finite then totality of $R : \mathbb{F}(X \times Y)$ is expressed by

$$\forall\, x \in X \bullet \exists\, y : Y \bullet (x, y) \in R$$

In general, different conditions are needed for different axioms to give subtypes of $\mathbb{F}(X \times Y)$:

- for single-valuedness – need X decidable
- for injectivity – need Y decidable
- for totality – need X finite
- for surjectivity – need Y finite

These can be combined, so for instance if both X and Y are decidable then we have Z's type $X \pinj_f Y$ of finite partial injections.

Once we have these types of finite relations, we can ask in addition when they are finite (decidability is easy: if X and Y are both decidable then so are $\mathbb{F}(X \times Y)$ and its subtypes).

The easiest answer is that if X and Y are both finite and both decidable, then the properties mentioned above are all complementable. For instance, for single-valuedness the complement is

$$\exists (x, y) \in R \bullet \exists (x', y') \in R \bullet x = x' \land y \neq y'$$

Hence under these strong conditions, all the subarities of $\mathbb{P}(X \times Y)$ mentioned correspond to finite decidable subsets of $\mathbb{F}(X \times Y)$.

Under more general conditions, answers are less clear-cut, though there are specific results that are useful sometimes. For instance, it can be proved that if X and Y are merely finite then the type of finite total relations from X to Y is also finite.

7 Conclusions

The GeoZ toolkit is still being developed, and like Z it is intended to be extensible. We hope that this paper has demonstrated the essential role of schema entailments in its formulation. The final result should be a useful library of mathematical contructions and techniques presented in a coherent way for engineers to refer to when they write GeoZ specifications.

We can ensure that a specification is well formed by checking that the contexts in which schema entailments are used are correct with respect to the premisses of the entailments involved. In many cases, *e.g.* operations on natural numbers, pairing, and so on, this will amount to little more than type checking. In other cases when the premiss of the entailment makes use of a theory through schema inclusion we can check that a similar inclusion is present in the context. However, in general we will need to prove that the preconditions as set out in the premiss of the entailment are satisfied – *e.g.* when the entailment is used to express a method such as induction this checking amounts to proving that the induction holds. There is some hope that this can be mechanised and in particular cases automated to provide tools to support GeoZ.

We have only had space to touch upon many of the important issues here. In particular we have not been able to discuss the application of schema morphisms to the specification of operations, implementations and refinement. We hope to make a more detailed exposition that addresses these points shortly.

Acknowledgements The work was assisted financially by the British Science and Engineering Research Council under the "Foundational Structures in Computing Science" research project at Imperial College.

References

[1] C.L. Hankin, I.C. Mackie, and R. Nagarajan, editors. *Theory and Formal Methods 1994: Proceedings of the Second Imperial College, Department of Computing, Workshop on Theory and Formal Methods*, Møller Center, Cambridge, UK, 11–14 September 1995. IC-Press.

[2] C.B. Jones. *Systematic software development using VDM (2e).* international series in computer science. Prentice-Hall, 1990.

[3] J.M. Spivey. *The Z Notation: A Reference Manual.* Series in Computer Science. Prentice Hall International, 2nd edition, 1992.

[4] S.J. Vickers. Geometric logic as a specification language. In Hankin et al. [1].