Consistency and refinement for partial specification in Z

Eerke Boiten, John Derrick, Howard Bowman and Maarten Steen

Computing Laboratory, University of Kent, Canterbury, CT2 7NF, U.K.
(Phone: +44 1227 827553, Email: {eab2,jd1,hb5,mwas}@ukc.ac.uk) *

Abstract. This paper discusses theoretical background for the use of Z as a language for partial specification, in particular techniques for checking consistency between viewpoint specifications. The main technique used is unification, i.e. finding a (candidate) least common refinement. The corresponding notion of consistency between specifications turns out to be different from the known notions of consistency for single Z specifications. A key role is played by correspondence relations between the data types used in the various viewpoints.

1 Partial specification

It is generally agreed that systems of a realistic size cannot be specified in single linear specifications, but rather should be decomposed into manageable chunks which can be specified separately. The traditional method for doing this is by hierarchical and functional decomposition. Nowadays, it is often claimed [11] that this is not the most natural or convenient (in relation to "perceived complexity") method – rather systems should be decomposed into different *aspects*. For each such *viewpoint* a specification of the system restricted to that particular aspect should be produced. Such *partial specifications* may omit certain parts of the system, because they are irrelevant to the particular aspect, and need not describe certain behaviours because they do not concern that specific viewpoint. Descriptions of this nature seem particularly appropriate for systems with various kinds of "users", each with their own view of the system. (Imagine, for example, the views of a library system that library managers, loan officers, clients, system operators, and programmers of the system would have.) Another reason for decomposing problems into aspects rather than subproblems is that different types of aspects have different specification languages that are best suited for them, for example dataflow diagrams for control flow, process algebras for "behaviour", data definition languages, et cetera.

There is one very serious problem in partial specification. Multiple viewpoints will describe what is intended to be the same system, and their descriptions will not in general be identical. Different viewpoints have different perspectives of

* This work was partially funded by the U.K. Engineering and Physical Sciences Research Council under grant number GR/K13035 and by British Telecom Labs., Martlesham, Ipswich, U.K.

the system, and they may even describe the system in different specification languages. This gives rise to an obligation to ensure that the partial specifications do not pose *contradictory* requirements: we need to check for *consistency*, potentially between descriptions in different languages and at different levels of abstraction.

One particular area in which viewpoint specification plays an important role is in *Open Distributed Processing* (ODP), an ISO standardisation initiative. The ODP reference model [9] defines five viewpoints for the specification of open distributed systems, with so-called *correspondence rules* as the links between the viewpoints (thus pinpointing the "is intended to be the same object" relationships). Our project 'Cross Viewpoint Consistency in Open Distributed Processing' aims to develop tools and techniques that enable the consistency of ODP specifications to be maintained. In previous papers [3, 5, 6] we have investigated techniques for consistency checking through unification in two of the main ODP specification languages: LOTOS [4] and Z [15]. The results obtained so far have convinced us that we need to explore further the nature of consistency checking and composition of partial specifications, and the role played by correspondence rules. This paper provides partial answers to these questions, mainly concentrating on the Z technique. Many of the issues raised will (thus) also be important to viewpoint specification in Z in general. The reader will be assumed to have a basic understanding of Z.

The next section presents a framework for consistency checking through unification, where the unification of specifications is a common refinement according to some refinement relation. Section 3 presents a number of general notions of consistency in Z. A concrete unification method for Z is then shown in section 4. We study mutual refinement in section 5, and conditions for a unified Z specification to be called consistent in section 6. In the final section we will draw some general conclusions and mention issues that need further research.

2 Consistency, unification, and refinement

We need to define what it means for a collection of viewpoint specifications to be consistent. Viewing the specifications as predicates over some universe, the logical definition of consistency is that it is impossible to derive both some proposition and its negation from the combined viewpoints.

In a context of specification and development of a concrete system, however, this abstract logical approach does not seem too useful. What *is* the universe we are quantifying over, and how do we map our specification language(s) to predicates over that universe? Would not a common semantic basis for possibly multiple languages necessarily be at such a low level that performing any kind of consistency proof becomes extremely complex [16]? What do we mean by "the combined viewpoints", will it always just be the logical conjunction of their formal interpretations, or do we need a more complex operator for combining viewpoints?

A more constructive view of consistency is one that is oriented towards system development. Instead of providing semantics for the specification languages, we encode our view of what specifications mean in *development relations*. Two specifications are in such a development relation if we consider one to be a correct development of the other on the way to an eventual implementation. A development relation may cross a language boundary, examples of such relations are semantics and translations, or it may not, in which case *refinement* relations form the main example, and equivalences another.

In such a framework, the consistency checking problem for a collection of viewpoint specifications is to find a specification which is a development of each of the viewpoint specifications according to the relevant development relations. Such common developments could also be called *unifications*. We are particularly interested in *least* unifications. In the special case where the development relation is a partial order (for example with refinement), a least unification is a *minimal* element in the set of unifications (where "minimal" is understood in the sense of fewest development steps done, least detail added, etc.) Such a least unification will be a most abstract specification that represents the viewpoints, which makes it a good choice to continue the unification process with. Suppose we wish to find a common development with yet another viewpoint. If the unification we chose is too concrete, we may have added details that make unification with the new viewpoint impossible. On the other hand, if we chose the most abstract one, we can be sure[2] that a unification with the new viewpoint, if it exists, can be found by unifying our previous unification with the new viewpoint. This guarantees that unifications of larger sets can be obtained by sequences of binary unifications.

In the examples we have looked at, it turned out that a lot of clarity was gained by being explicit about the overlap between the viewpoints. In many cases we can get away with assuming that equal names in different viewpoints refer to the same system object or function, but in some cases we cannot. A clear example of that is two objects with the same name but with different types: how do their types relate? Such relations between viewpoint specifications we will include explicitly as *correspondence relations* (as the name suggests, we think these may make up an important part of the correspondence rules in the ODP model). Unification and (existence of a) least common refinement will be with regard to a given correspondence relation.

In this way we have defined consistency as the existence of a least unification, with no additional tests. In practice, however, it is often convenient to generate a candidate least common development, i.e. some specification that *is* the least unification *if* one exists, and then to perform some consistency tests on it to determine whether it actually *is* a least unification. We will call such candidates "unifications" as well (using the term in a slightly sloppy sense). Finally note that it is strictly speaking incorrect to talk about *the* least unification of a

[2] Provided there are not multiple least unifications that are incomparable in the refinement ordering. Fortunately this will never occur with most known refinement relations, in particular Z refinement.

collection of viewpoint specifications, since for most specification languages and development relations there will be many equivalent ones.

3 General forms of consistency in Z

The language Z [15] is often used for viewpoint specifications, see for example [1, 10, 11], so there is a clear need for the investigation of consistency between partial specifications in Z, as most of the cited papers observe. What makes it particularly important for our project is that the ODP reference model [9] has adopted Z as one of the formal description techniques to be used as a viewpoint language, in particular for the *information* viewpoint.

Before going into the consistency *between* Z specifications in later sections, we briefly look at some ways in which a Z specification *on its own* can be considered inconsistent.

First, there are the direct contradictions, which all allow us to prove both P and $\neg P$ for some predicate P, or in other words (removing the quantification) which allow us to derive "false" from the specification. This is the simplest and most obvious definition of inconsistency in Z. The strong typing system of Z prevents quite a few classes of errors, but some kinds of contradictions can still be written, for example:

- Postulating that an empty set has an element:

  $$x : \varnothing$$

- Abusing the fact that a function is a set of pairs:

  $$\begin{array}{|l} f : \mathbb{N} \rightarrow \mathbb{N} \\ \hline f = \{(1,2),(1,3)\} \end{array}$$

  (of course similar examples exist for all the different types of functions, including sequences).
- Inconsistent free types (a lot has been written on this, see [15, 2, 14]), for example $T ::= atom\langle\langle \mathbb{N} \rangle\rangle \mid fun\langle\langle T \rightarrow T \rangle\rangle$.

It is clear that inconsistencies of this type will also be inconsistencies if they occur in partial specifications.

A different type of possible inconsistency occurs when schemas have empty sets of bindings, for example (trivially)$D \mathrel{\widehat=} [\, x : S \mid \mathrm{false} \,]$. As long as we do not assert that we have a value from D, this is not an actual inconsistency. In the states-with-operations interpretation of Z, described for example in [15, chapter 5], a schema with an empty set of bindings is probably a specification error. A special case of this condition is known as the *Initialisation Theorem*: the schema describing the initial state of an abstract data type should not be empty. Even though we will be adopting the states-with-operations approach to specification in Z, it is not clear at this point if we should mark empty schemas as inconsistencies in partial specifications, and whether we should distinguish between state schemas and operation schemas in that respect.

In the sequel we will see that (candidate) unifications of Z viewpoint specifications may satisfy all of the above notions of consistency, and still not retain the interpretations of the viewpoints. Before we can observe this, we have to show how to construct unifications.

4 A unification method in Z

In this (long) section we construct candidate least unifications for pairs of Z specifications, with the normal Z refinement relation as the development relation for both partial specifications. The unification will turn out to be a least common refinement provided two conditions hold of the viewpoint specifications.

We will concentrate on the unification of state and operation schemas, since we envisage the viewpoint specification style for Z to have those as its major elements. Other Z constructs are degenerate cases of these, or not expected to occur in more than one viewpoint. Operations change states, so we need to unify state schemas first. After that we can adapt and unify operation schemas.

4.1 Unification of state schemas informally

Let us (for now) take as the underlying interpretation of a state schema $D_1 \mathrel{\widehat=} [\, x{:}Apple \mid NotWormEaten\; x \,]$ that it allows us to choose from all apples, but we have to discard the worm-eaten ones. To unify this with $D_2 \mathrel{\widehat=} [\, x{:}Fruit \mid NotRotten\; x \,]$ we have to take the union of the base sets (which is *Fruit* in this case) and the intersection of the conditions (*NotRotten* and, if it is an *Apple*, also *NotWormEaten*). Thus, we interpret the type declaration as giving a "set we choose from", and unification extends the range of choice. The predicates on the other hand are interpreted as restrictions, which need to be combined in unification. Formally predicates and subtypes are of course equivalent, which suggests we should have disjunctions or conjunctions in *both* cases. For the examples we have dealt with so far [6, 7], however, this default interpretation seemed to capture the intuition much better.

So, suppose we have the following two state schemas (which, given the above interpretation, will *not* be normalised[3]). They have the same name (with a distinguishing index), which means that they are linked by an implicit correspondence rule.

D_1	D_2
$x{:}S$	$x{:}T$
$pred_S$	$pred_T$

According to our intuive view of state schemas, their unification should be [6]:

[3] The normalisation of a state schema changes the variables to be of their maximal type, and puts all other typing information in the predicate.

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{D} \\
x : S \cup T \\
\hline
x \in S \Rightarrow pred_S \\
x \in T \Rightarrow pred_T \\
\hline
\end{array}
$$

However, this is not type correct in general: $S \cup T$ is an error unless S and T have the same maximal type. A *disjoint* union of S and T would not be right either, since then values that S and T have in common would be considered different.

4.2 Totalised correspondence relations provide unions

So how do we resolve the situation that S and T may have values in common and may also be type-incompatible? The answer is to modify the second-best known implementation of disjoint union as a product[4] and to use correspondence relations.

(In order to keep this explanation simple, we venture outside the Z typing system for a moment.) If $\mathbb{1}$ is a type with a single element not in S or T, let us call it $\perp$, then we could define the disjoint union of S and T by

$$S + T = S \times \mathbb{1} \cup \mathbb{1} \times T$$

i.e. $S + T = \{(s,t) \mid (s \in S \wedge t = \perp) \vee (s = \perp \wedge t \in T)\}$. The smallest product set containing this set is $S_\perp \times T_\perp$, where $Q_\perp$ is the union of Q and $\mathbb{1}$. (Still a disjoint union, but of an appreciably simpler kind.) Now what we will do is construct particular subsets within $S_\perp \times T_\perp$, starting from $S + T$. The rules are simple: we can add tuples (s,t) for which $s \in S$ and $t \in T$, provided we then remove tuples $(s,\perp)$ and $(\perp,t)$ from the set. (The interpretation of this is that we no longer consider s and t different. Compare the interpretation of disjoint union: no element from the one set is equal to any element from the other.)

Let us call such sets *totalised correspondence relations*. Totalised correspondence relations over S and T are characterised by the fact that for each $s \in S$ there is a tuple (s,x) for some $x \in T_\perp$, and exactly one such tuple if $x = \perp$, and similarly for all $t \in T$. Totalised correspondence relations are linked in a one-to-one way with *correspondence relations* between S and T: for $tot\ R$ the totalised correspondence of R, we have $tot\ R = R \cup (S \setminus \mathrm{dom}\ R) \times \mathbb{1} \cup \mathbb{1} \times (T \setminus \mathrm{ran}\ R)$, and $R = tot\ R \cap S \times T$. In particular, the empty relation corresponds to the disjoint union. For any correspondence relation (given by the specifier), the totalised correspondence relation will provide the desired union of state spaces.

Here ends our brief excursion outside the Z typing system; we now give the formal definitions in Z. The main differences arise from the need to use explicit injection functions (into free types) where we used set unions above. The one-to-one correspondence also holds in Z, it just looks a bit more complicated.

[4] The best known one is $S + T = \{0\} \times S \cup \{1\} \times T$.

Definition 1 (Type with bottom) For any type S, we define the type $S_\perp$ by the following free type definition:

$$S_\perp \; ::= \; \perp_S \mid justS\langle\!\langle S \rangle\!\rangle$$

For all such types, a function $theS$ is defined as the inverse of the injection $justS$:

$$\begin{array}{|l}
theS : S_\perp \twoheadrightarrow\!\!\!\!\!\!\!\!\;+\;\; S \\
\hline
\mathrm{dom}\, theS = \mathrm{ran}\, justS \\
\forall\, x{:}S \bullet theS\ (justS\ x){=}x
\end{array}$$

□

Definition 2 (Totalisation of a relation) The totalisation $tot\ R$ of a relation R on two given types S and T is defined as follows:

$$\begin{array}{|l}
[S,T] \\
\hline
tot : (S \leftrightarrow T) \rightarrowtail\!\!\!\!\twoheadrightarrow (S_\perp \leftrightarrow T_\perp) \\
\hline
\forall\, R : S \leftrightarrow T \bullet \\
\quad tot\ R = theS \mathbin{;} R \mathbin{;} justT \\
\qquad\quad \cup \{x : S \setminus \mathrm{dom}\, R \bullet (justS\ x, \perp_T)\} \cup \{y : T \setminus \mathrm{ran}\, R \bullet (\perp_S, justT\ y)\}
\end{array}$$

□

Totalised correspondences provide the possibility to specify anything between disjoint union (take the correspondence to be the empty relation) and union (take the correspondence to be the identity relation on the intersection). Moreover, they provide the opportunity to relate elements of types that cannot be directly related in Z even if they appear to be identical.

Example 3 (Union of enumerated types) If we have $S = a \mid b \mid c$ and $T = a \mid d$, we can form the obvious union where both a's are identified by taking the correspondence to be $\{(a,a)\}$. The totalised correspondence relation is then the set $\{(justS\ b, \perp_T), (justS\ c, \perp_T), (justS\ a, justT\ a), (\perp_S, justT\ d)\}$ which can be seen as a renaming of the set $\{b,c,a,d\}$.

□

If we provide a correspondence relation between S and T, which points out exactly which values in S correspond to which values in T, the totalised correspondence provides the required union of S and T. In many cases the specifier need not explicitly state what the correspondence relation is, the default correspondence relation defined below may give the desired result.

Definition 4 (Default correspondence) The default correspondence relation on schemas $D_1 \mathrel{\widehat{=}} [\, x{:}S \mid pred_S\,]$ and $D_2 \mathrel{\widehat{=}} [\, x{:}T \mid pred_T\,]$ is $\{(x,x) \bullet x \in S \cap T\}$ if $S \cap T$ is a well-typed expression (i.e. S and T have a common supertype); otherwise it is the empty relation.

□

(In another paper [3] we have described how the correspondence relation can be used to unify viewpoints that use different representations of the same data, using the same unification rules. In the current paper we concentrate on correspondence relations that are partial identity relations or other injective functions; the paper [3] shows that allowing general relations extends viewpoint unification with datatype implementation.)

4.3 State unification using correspondences

Let us assume that the correspondence relation between the types S and T is given by the relation $R \subseteq S \times T$. The inhabitants of the unified state schema will be the tuples of $tot\,R$.

$$
\begin{array}{|l}
\hline
D \\
x_1 : S_\perp;\ x_2 : T_\perp \\
\hline
(x_1, x_2) \in tot\ R \\
\forall\, x : S \bullet x_1 = justS\ x \Rightarrow pred_S \\
\forall\, x : T \bullet x_2 = justT\ x \Rightarrow pred_T \\
\hline
\end{array}
$$

This looks like we are actually maintaining two values for the state variable x; however, due to (x_1, x_2) being in $tot\,R$ it is the case that either exactly one of the two values is $\perp$ and thus invalid, or the two values are "equal" (since they are in R, and R only contains tuples of things we consider equal).

In the examples that follow, we will often observe isomorphisms between schemas: the schemas that get constructed often have additional clutter of constructor functions and their inverses, and renamings of all inhabitants of the schema usually exist that yield the intuitively desirable schemas. Such (injective) renamings of all inhabitants of a schema we call *isomorphisms*, they form a special case of data refinement in both directions, see section 5.

Example 5 (Union of enumerated types, continued) Continuing from example 3, suppose we have schemas $D_1 \mathrel{\widehat{=}} [\,x : S\,]$ and $D_2 \mathrel{\widehat{=}} [\,x : T\,]$ where the types are given by $S\ ::=\ a \mid b \mid c$ and $T\ ::=\ a \mid d$, and the correspondence relation by $R = \{(a, a)\}$ (the default correspondence would be empty). Their unification is given by the schema

$$
\begin{array}{|l}
\hline
D \\
x_1 : S_\perp;\ x_2 : T_\perp \\
\hline
(x_1, x_2) \in tot\ R \\
\forall\, x : S \bullet x_1 = justS\ x \Rightarrow \mathrm{true} \\
\forall\, x : T \bullet x_2 = justT\ x \Rightarrow \mathrm{true} \\
\hline
\end{array}
$$

which (see example 3) is isomorphic to the schema $D \mathrel{\widehat{=}} [\,x : V\,]$ where $V\ ::=\ a \mid b \mid c \mid d$. Using the default would result in two different a's.

□

The next two examples do use the default correspondence.

Example 6 The schemas

$$
\begin{array}{|l}
\hline
D_1 \\
x : \mathbb{Z} \\
\hline
-1 \leq x \leq 3 \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\hline
D_2 \\
x : \mathbb{Z} \\
\hline
\exists\, z : \mathbb{N} \bullet x = z + z \\
\hline
\end{array}
$$

have the same type of component so their correspondence relation is the identity relation on that type. The schema that results from unification is

$$
\begin{array}{|l}
\multicolumn{1}{l}{D} \\
\hline
x_1 : \mathbb{Z}_\perp ;\ x_2 : \mathbb{Z}_\perp \\
\hline
(x_1, x_2) \in \mathit{tot}\,\{(x, x) \mid x \in \mathbb{Z}\} \\
\forall\, x : \mathbb{Z} \bullet x_1 = \mathit{just}\mathbb{Z}\, x \Rightarrow -1 \leq x \leq 3 \\
\forall\, x : \mathbb{Z} \bullet x_2 = \mathit{just}\mathbb{Z}\, x \Rightarrow \exists\, z : \mathbb{N} \bullet x = z + z \\
\hline
\end{array}
$$

The totalised identity relation on $\mathbb{Z}$ is the set $\{(\mathit{just}\mathbb{Z}\, x, \mathit{just}\mathbb{Z}\, x) \mid x \in \mathbb{Z}\}$, so this schema is isomorphic to

$$
\begin{array}{|l}
\multicolumn{1}{l}{D} \\
\hline
x : \mathbb{Z} \\
\hline
-1 \leq x \leq 3 \\
\exists\, z : \mathbb{N} \bullet x = z + z \\
\hline
\end{array}
$$

which, as it turns out, is the intersection (or rather: the *conjunction*) of the input schemas. This can be shown to hold in general: if the types of the components are identical, the union of the schemas is their conjunction for the default correspondence relation.

□

Example 7 Any similarity between this example and the previous one will be discussed in later sections. Schemas $D_1 \mathrel{\widehat{=}} [\, x : -1 \,..\, 3 \,]$ and $D_2 \mathrel{\widehat{=}} [\, x : \{z : \mathbb{N} \bullet z + z\} \,]$ have the identity relation on the intersection of their component types as the correspondence relation, i.e. $\{(0,0),(2,2)\}$. The schema resulting from their unification is isomorphic to $D \mathrel{\widehat{=}} [\, x : (-1 \,..\, 3) \cup \{z : \mathbb{N} \bullet z + z\} \,]$.

□

The final example shows that a schema with an empty set of bindings might fulfill a very useful role when we apply this state unification rule: it is the unit of state unification, modulo a trivial renaming.

Example 8 (The empty state) For the states $D_1 \mathrel{\widehat{=}} [\, x : \varnothing \,]$ and $D_2 \mathrel{\widehat{=}} [\, x : T \mid \mathit{pred}_T \,]$ there is only one correspondence relation possible, viz. the empty relation, the only subset of $\varnothing \times T = \varnothing$[5]. Totalising yields the set $\{x : T \bullet (\perp_\varnothing, \mathit{just}T\, x)\}$. (Note that the type $\varnothing_\perp$ has only one element, viz. $\perp_\varnothing$.) Thus, the unified schema is

[5] We are aware that for strict Z typing we have to state the types of the elements that the various empty sets do not contain.

$$
\begin{array}{|l}
\hline
\textit{D} \\
x_1 : \varnothing_\perp ;\; x_2 : T_\perp \\
\hline
(x_1, x_2) \in \{ x : T \bullet (\perp_\varnothing, \mathit{justT}\ x) \} \\
\forall\, x : \varnothing \bullet x_1 = \mathit{just}\varnothing\, x \Rightarrow \mathrm{true} \\
\forall\, x : T \bullet x_2 = \mathit{justT}\ x \Rightarrow \mathit{pred}_T \\
\hline
\end{array}
$$

which is obviously isomorphic to $D \mathrel{\widehat{=}} [\, x : T \mid \mathit{pred}_T \,]$.

□

The significance of this is that we have a uniform way of treating state schemas across viewpoints: if a certain state schema is not defined at all in one viewpoint, we may regard it as defined to be an empty state space.

4.4 Unification of operation schemas

The unification of operation schemas proceeds in two steps. In the first step, all schemas get adapted to the unified state schemas. In the second step, operations that are defined in both viewpoints are unified using their pre- and postconditions.

In the presentation of these rules, we assume that the state has changed exactly according to the rule for state unification given above, i.e. that no renamings of the inhabitants have taken place. Note, however, that because these renamings are injective functions, we can freely translate back and forth between the isomorphic state spaces. In other words, in most concrete cases the expressions with lots of constructor functions etc., as we give them here, can be translated into something more intuitive, just as we did for state schemas in the examples.

An operation that was originally defined on the state D_1 by

$$
\begin{array}{|l}
\hline
\mathit{Op}_1 \\
\Delta D_1 ;\; \mathit{Decl}_1 \\
\hline
\mathit{pred}_1 \\
\hline
\end{array}
$$

gets adapted to the new state schema by changing it to

$$
\begin{array}{|l}
\hline
\mathit{AdOp}_1 \\
\Delta D ;\; \mathit{Decl}_1 \\
\hline
x_1 \in \operatorname{ran} \mathit{justS} \\
x_1{}' \in \operatorname{ran} \mathit{justS} \\
\textbf{let}\ x == \mathit{theS}\ x_1 \;;\; x' == \mathit{theS}\ x_1{}' \bullet \mathit{pred}_1 \\
\hline
\end{array}
$$

(and of course an almost identical rule is used for operations from the second viewpoint.) A very similar rule can be given for operations that do not change the state (i.e. that have ΞD in their declarations). The situation is only slightly more complicated if operations operate on multiple states – the rule above can then be applied repeatedly, and the only complication is the bookkeeping of which references to states have been updated to refer to changed states.

The unification of two viewpoint operations should exhibit possible behaviour of each of the viewpoint operations in each situation where the viewpoint operation was applicable. This requirement can be formalised using pre- and postconditions.[6] The unified operation should be applicable whenever one of the viewpoint operations is, i.e. its precondition should be the disjunction of the viewpoint operation preconditions. Moreover, when the unified operation is applied to a state satisfying one particular precondition, a state should result that satisfies the corresponding postcondition. Such an operation unification is also described by Ainsworth et al. [1], there called *union*, but they fail to mention that the union may not exist. The candidate least unification of operation schemas $AdOp_1$ and $AdOp_2$, both operating on the same state, is given by[7]

$$
\begin{array}{|l}
\hline
\textit{UnOp} \\
\textit{Decls};\ \Delta D \\
\hline
\text{pre}\ AdOp_1 \vee \text{pre}\ AdOp_2 \\
\text{pre}\ AdOp_1 \Rightarrow \text{post}\ AdOp_1 \\
\text{pre}\ AdOp_2 \Rightarrow \text{post}\ AdOp_2 \\
\hline
\end{array}
$$

where *Decls* is obtained by textually unifying the declarations of $AdOp_1$ and $AdOp_2$. That this schema only defines the desired unification under additional restrictions is clear from a little calculation. Note that the precondition of an operation Op with no input or output, operating on $State$, is given by

$$\text{pre}\ Op = \exists\, State' \bullet Op$$

We write pre_1 for pre $AdOp_1$ etc for clarity in the following calculation:

$$
\begin{array}{ll}
 & \text{pre}\ UnOp \\
\equiv & \qquad \{\ \text{definition pre}\ \} \\
 & \exists\, State' \bullet (\text{pre}_1 \vee \text{pre}_2) \wedge (\text{pre}_1 \Rightarrow \text{post}_1) \wedge (\text{pre}_2 \Rightarrow \text{post}_2) \\
\equiv & \qquad \{\ \text{pre}_1 \text{ and } \text{pre}_2 \text{ do not refer to } State'\ \} \\
 & (\text{pre}_1 \vee \text{pre}_2) \wedge \exists\, State' \bullet (\text{pre}_1 \Rightarrow \text{post}_1) \wedge (\text{pre}_2 \Rightarrow \text{post}_2)
\end{array}
$$

[6] Note that, unlike the precondition, the postcondition of a Z operation schema cannot be uniquely determined. For a schema $Op \mathrel{\widehat=} [\Delta D \mid pred]$ which (to avoid some semantic problems) satisfies the condition $pred \Rightarrow \text{pre}\ Op$, any condition P such that $\text{pre}\ Op \wedge P \Leftrightarrow pred$ will do as "the" postcondition, in particular *pred* itself. Thus any occurrence of post Op in the sequel should be taken to refer to *some* possible postcondition of Op.

[7] Wim Feijen pointed out the similarity between the conditions in this schema and those in the w(eakest)p(recondition)-calculus for the guarded command $P_1 \rightarrow Op_1 \ \square\ P_2 \rightarrow Op_2$ where pre_i has the role of the guard.

$$\equiv \qquad \{ \text{ case analysis, } \exists\, State' \bullet \text{pre}_i \Rightarrow \text{post}_i \text{ holds } \}$$

$$(\text{pre}_1 \lor \text{pre}_2) \land \exists\, State' \bullet \text{pre}_1 \land \text{pre}_2 \Rightarrow \text{post}_1 \land \text{post}_2$$

$$\equiv \qquad \{ \text{ pre}_1 \text{ and pre}_2 \text{ do not refer to } State' \}$$

$$(\text{pre}_1 \lor \text{pre}_2) \land (\text{pre}_1 \land \text{pre}_2 \Rightarrow \exists\, State' \bullet \text{post}_1 \land \text{post}_2)$$

In other words, the precondition of the union is *only* the disjunction of the preconditions if both postconditions can be satisfied when both preconditions are. This is an essential condition which will form part of our consistency check. In fact, it is already a condition for the union to be a common refinement of the operations, and it is useful to give it a name.

Definition 9 Operations A and B, operating on the same state space $State$, are said to be *operation consistent* iff

$$\forall\, State \bullet \text{pre}\, A \land \text{pre}\, B \Rightarrow \exists\, State' \bullet \text{post}\, A \land \text{post}\, B.$$

□

4.5 Unification is least common refinement

Here we present what amounts to a correctness proof for the unification rules given above. The proof will be in three steps: showing that the adapted operations with the unified state form data refinements of the viewpoints; showing that unified operations are (operation) refinements of the adapted operations; and finally a proof that the unification is a *least* common refinement. The proof given below imposes extra conditions on the viewpoint specifications in two places: one is operation consistency as defined above, the other is *state consistency* which follows from analysis of the preconditions of the adapted operations.

First we show that the unified state with the adapted operations form data refinements of the viewpoints with operations. For that purpose we have to formally link the state schemas using a *retrieve relation*. For the unified state schema D and the state schema of the first viewpoint $D_1 \mathrel{\widehat=} [\, x{:}S \mid pred_S \,]$ the retrieve relation is given by the schema

$Retr1$
$D_1;\ D$
$x_1 = justS\ x$

There are two conditions to prove that this is a valid data refinement [15], making any universal quantifications implicit:

1. pre $Op_1 \land Retr1 \Rightarrow$ pre $AdOp_1$
2. pre $Op_1 \land Retr1 \land AdOp_1 \Rightarrow \exists\, x' \bullet Retr1' \land Op_1$

The proof of the first property has a big hurdle in the middle of it. For simplicity we ignore the contribution of $Decl_1$ to the predicate $AdOp_1$ since it makes the same contribution to Op_1.

$\quad$ pre $AdOp_1$

$\equiv \quad$ { definition of pre }

$\quad \exists\, x_1{}' ;\ x_2{}' \bullet AdOp_1$

$\equiv \quad$ { definition $AdOp_1$ }

$\quad \exists\, x_1{}' ;\ x_2{}' \bullet D \wedge D' \wedge x_1 \in \operatorname{ran} justS$

$\qquad \wedge\ x_1{}' \in \operatorname{ran} justS \wedge pred_1[theS\ x_1/x][theS\ x_1{}'/x']$

$\equiv \quad$ { conjuncts independent of new state }

$\qquad D \wedge x_1 \in \operatorname{ran} justS$

$\wedge \quad \exists\, x_1{}' ;\ x_2{}' \bullet D' \wedge x_1{}' \in \operatorname{ran} justS \wedge pred_1[theS\ x_1/x][theS\ x_1{}'/x']$

$\equiv \quad$ { **WISH:** $x_2{}'$ always exists here; translation $x' := theS\ x_1{}'$ }

$\quad D \wedge x_1 \in \operatorname{ran} justS \wedge \exists\, x' \bullet D_1[theS\ x_1{}'/x] \wedge pred_1[theS\ x_1/x]$

$\equiv \quad$ { definition of pre }

$\quad D \wedge x_1 \in \operatorname{ran} justS \wedge \text{pre } Op_1[theS\ x_1/x]$

$\Leftarrow \quad$ { definition $Retr1$, substitution }

$\quad Retr1 \wedge \text{pre } Op_1$

Of course the crux of this proof is the step marked with **WISH**. It is clear that we need an extra condition here, the predicate really depends on $x_2{}'$ through the conjunct D'. A correct $x_2{}'$ may not exist in exactly one type of situation: $(x_1{}', x_2{}') = (justS\ x, justT\ y)$ and $(x,y) \in R$, $pred_S$ holds but $pred_T[y/x]$ does not hold. That is to say, the output value of the operation is linked by the correspondence relation to an "illegal" value, whereas the input value is linked to a legal one (and thus not excluded from the translated precondition $Retr1 \wedge$ pre Op_1). At this point we will assume that the viewpoints are *state consistent* to prevent this problem:

Definition 10 The two state schemas $D_1 \mathrel{\widehat=} [\, x{:}S \mid pred_S \,]$ and $D_2 \mathrel{\widehat=} [\, x{:}T \mid pred_T \,]$ are *state consistent* with respect to the correspondence relation $R \subseteq S \times T$ iff

$$(x,y) \in R \Leftrightarrow (pred_S \Leftrightarrow pred_T[y/x])$$

$\square$

This is a sufficient, but not a necessary condition; for a further discussion of related properties, see section 6. The second property is more easily proved:

$\quad \exists\, x' \bullet Retr1' \wedge Op_1$

$\equiv \quad$ { definitions }

$\quad \exists\, x' \bullet D_1{}' \wedge D' \wedge x_1{}' = justS\ x' \wedge D \wedge D' \wedge pred_1$

$\equiv \quad$ { D and D' independent of x'; $theS$ is inverse of $justS$ }

$\quad (\exists\, x' \bullet D_1{}' \wedge theS\ x_1{}' = x' \wedge pred_1) \wedge D \wedge D'$

$\equiv \quad$ { one point rule for existential quantifier }

$$pred_S[theS\, x_1{}'/x] \wedge pred_1[theS\, x_1{}'/x'] \wedge D \wedge D'$$

$\Leftarrow$ $\{$ first conjunct follows from D'; property of substitution $\}$

$$pred_1[theS\, x_1/x][theS\, x_1{}'/x'] \wedge x_1 = justS\; x \wedge D \wedge D'$$

$\Leftarrow$ $\{$ definitions $AdOp_1$ and $Retr1$, add conjunct $\}$

$$\mathrm{pre}\; Op_1 \wedge AdOp_1 \wedge Retr1$$

Of course the proof for the second viewpoint is completely analogous.

The second step is to show that $UnOp$ is a common refinement of $AdOp_1$ and $AdOp_2$. In this case, too, it suffices to give only one half of the proof. Because this step involves no change of state space, we only need to prove the two conditions for operation refinement [15], again omitting universal quantifications:

1. pre $AdOp_1 \Rightarrow$ pre $UnOp$
2. pre $AdOp_1 \wedge UnOp \;\Rightarrow\; AdOp_1$

The first is only true if the *operation consistency* condition holds, see the calculation of pre $UnOp$ above (and then it is a one line proof). The second is easily proved using the fact that the predicate part of an operation schema A can be given as pre $A \wedge$ post A.

The final step of the least common refinement proof is showing that the unification is a *least* common refinement. This will be done by showing that an *arbitrary* refinement of both viewpoints is necessarily a refinement of the unification.

Suppose that state schema E with operation schema Opp also form a (data) refinement of both viewpoint specifications (D_1, Op_1) and (D_2, Op_2), and that the state of E is given by the (fresh) variable y. This means that two retrieve relations exists, let us assume they are given by $(i{=}1,2)$

$Retr_i$
$D_i;\; E$
$retr_i$

The assumption that these are data refinements translates into assumptions we can use in proofs:

1. pre $Op_i \wedge Retr_i \;\Rightarrow$ pre Opp
2. pre $Op_i \wedge Retr_i \wedge Opp \;\Rightarrow\; \exists\, x' \bullet Retr_i{}' \wedge Op_i$

We now prove that, under these assumptions, (E, Opp) is a data refinement of $(D, UnOp)$. Thus we have to find some retrieve relation $RetrED$ such that

1. pre $UnOp \wedge RetrED \;\Rightarrow$ pre Opp
2. pre $UnOp \wedge RetrED \wedge Opp \;\Rightarrow\; \exists\, x_1{}';\, x_2{}' \bullet RetrED' \wedge UnOp$

Our choice for that retrieve relation is the following schema.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\textit{RetrED}\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}} \\
D;\ E \\
\hline
retr_1[theS\ x_1/x] \vee retr_2[theT\ x_2/x] \\
\hline
\end{array}
$$

(The main motivation for this particular choice is that it works.)

Now we prove the two properties. For the first we leave out universal quantification over y, the "concrete state".

$$
\begin{array}{ll}
 & \forall\, x_1;\ x_2 \bullet \text{pre}\ Opp \Leftarrow \text{pre}\ UnOp \wedge RetrED \\
\equiv & \qquad \{\ \text{assuming operation consistency}\ \} \\
 & \forall\, x_1;\ x_2 \bullet \text{pre}\ Opp \Leftarrow (\text{pre}\ AdOp_1 \vee \text{pre}\ AdOp_2) \wedge RetrED \\
\equiv & \qquad \{\ \text{definition}\ RetrED\ \} \\
 & \forall\, x_1;\ x_2 \bullet \text{pre}\ Opp \Leftarrow (\text{pre}\ AdOp_1 \vee \text{pre}\ AdOp_2) \wedge D \wedge E \\
 & \qquad\qquad\qquad \wedge retr_1[theS\ x_1/x] \vee retr_2[theT\ x_2/x] \\
\Leftarrow & \qquad \{\ \text{calculus}\ \} \\
 & \forall\, x_1;\ x_2 \bullet \text{pre}\ Opp \Leftarrow (\text{pre}\ AdOp_1 \wedge D \wedge E \wedge retr_1[theS\ x_1/x]) \\
 & \qquad\qquad\qquad \vee(\text{pre}\ AdOp_2 \wedge D \wedge E \wedge retr_2[theT\ x_2/x]) \\
\equiv & \qquad \{\ \text{definition pre}\ AdOp_i\ \text{(state consistency);} \\
 & \qquad\ \ \text{translation}\ (x_1, x_2)\ :=\ (justS\ x, justT\ y)\ \} \\
 & \forall\, x;\ y \bullet \text{pre}\ Opp \Leftarrow (\text{pre}\ Op_1 \wedge D_1 \wedge E \wedge retr_1) \\
 & \qquad\qquad\qquad \vee((\text{pre}\ Op_2)[y/x] \wedge D_2 \wedge E \wedge retr_2[y/x]) \\
\equiv & \qquad \{\ \text{definition}\ Retr_i;\ \text{assumptions}\ \} \\
 & \text{true}
\end{array}
$$

The second proof is a quite complicated one. We are asked to prove that $\forall\, x_1;\ x_2;\ y \bullet P \Rightarrow (\exists\, x_1{}';\ x_2{}' \bullet Q)$ for certain predicates P and Q. The proof proceeds by first showing how $\exists\, x_1{}';\ x_2{}' \bullet Q$ can be rewritten as $\exists\, x' \bullet Q_1 \vee \exists\, x' \bullet Q_2$. Then we do a case introduction on P such that $P = (P_1 \vee P_2)$ and we show that $\forall\, x_1;\ x_2;\ y \bullet P_i \Rightarrow (\exists\, x' \bullet Q_i)$ follows from the assumption that E is a refinement of the i-th viewpoint, which then completes the proof.

$$
\begin{array}{ll}
 & \exists\, x_1{}';\ x_2{}' \bullet RetrED' \wedge UnOp \\
\equiv & \qquad \{\ \text{definition}\ UnOp,\ \text{assuming operation consistency}\ \} \\
 & (\exists\, x_1{}';\ x_2{}' \bullet RetrED' \wedge AdOp_1) \vee (\exists\, x_1{}';\ x_2{}' \bullet RetrED' \wedge AdOp_2)
\end{array}
$$

The simplifications of these disjuncts will be completely analogous so we show only one:

$$
\begin{array}{ll}
 & \exists\, x_1{}';\ x_2{}' \bullet RetrED' \wedge AdOp_1 \\
\Leftarrow & \qquad \{\ \text{definition of}\ RetrED'\ \text{and}\ AdOp_1\ \} \\
 & \exists\, x_1{}';\ x_2{}' \bullet D' \wedge E' \wedge (retr_1[theS\ x_1/x])' \wedge D \wedge x_1 \in \text{ran}\ justS
\end{array}
$$

$$\wedge\; x_1{}' \in \operatorname{ran} justS \wedge pred_1[theS\; x_1/x][theS\; x_1{}'/x']$$

$$\equiv \qquad \{\ \text{assuming state consistency, translate } x_1{}' := justS\; x'\ \}$$

$$\exists\, x' \bullet D_1{}' \wedge E' \wedge retr_1{}' \wedge D \wedge x_1 \in \operatorname{ran} justS \wedge pred_1[theS\; x_1/x]$$

$$\equiv \qquad \{\ \text{definition of } Retr_1\ \}$$

$$\exists\, x' \bullet Retr_1{}' \wedge D \wedge x_1 \in \operatorname{ran} justS \wedge pred_1[theS\; x_1/x]$$

The antecedent (we called it P in the proof overview above) of the universal quantification can be rewritten in the form $P_1 \vee P_2$ as follows:

$$\text{pre}\; UnOp \wedge RetrED \wedge Opp$$

$$\equiv \qquad \{\ \text{assuming operation consistency}\ \}$$

$$(x_1 \in \operatorname{ran} justS \wedge \text{pre}\; AdOp_1 \wedge RetrED \wedge Opp)$$

$$\vee \quad (x_2 \in \operatorname{ran} justT \wedge \text{pre}\; AdOp_2 \wedge RetrED \wedge Opp)$$

Now we show that each of the disjuncts in the antecedent (P_i) proves one of the disjuncts in the consequent (Q_i). Again these two proofs are completely analogous, so only one is given.

$$\forall\, x_1; x_2; y \bullet x_1 \in \operatorname{ran} justS \wedge \text{pre}\; AdOp_1 \wedge RetrED \wedge Opp$$

$$\Rightarrow \exists\, x' \bullet Retr_1{}' \wedge D \wedge x_1 \in \operatorname{ran} justS \wedge pred_1[theS\; x_1/x]$$

$$\Leftarrow \qquad \{\ \text{assuming state consistency, translate } x_1 := justS\; x\ \}$$

$$\forall\, x; y \bullet \text{pre}\; Op_1 \wedge D_1 \wedge E \wedge retr_1 \wedge Opp$$

$$\Rightarrow \exists\, x' \bullet Retr_1{}' \wedge pred_1 \wedge D_1$$

$$\equiv \qquad \{\ \text{definition } Retr_1\text{, assumption}\ \}$$

$$\text{true}$$

This concludes our proof that every common refinement of the viewpoints is a refinement of the unification, and thus the unification is indeed the least common refinement.

5 On mutual refinement

The previous section has shown how the unification is (with a few conditions) "the" least common refinement of the viewpoints, by proving that all common refinements are refinements of the unification. Obviously, in general multiple least common refinements exist – for example other unifications with *different* correspondence relations that fulfill the state and operation consistency conditions. One might think that the equivalence classes induced by mutual refinement contain only specifications that are equal modulo an injective renaming of the inhabitants of the schemas ("isomorphism"). This, however, is not the case.

Have another look at examples 6 and 7. The viewpoint specifications given there are actually semantically identical, their only difference is that some information has shifted from the type of x to the schema predicate. The source of difference in the examples is in the correspondence relation that was used. (Why we would choose different default correspondence relations for "identical" specifications will be discussed in section 7.) The question of whether these unified state schemas are refinements fully depends on what operations are defined in the viewpoints. In general the unification of example 6 will not be a refinement, because the state consistency condition is violated (x=4 is excluded by the first viewpoint predicate, for example). On the other hand, state consistency holds in example 7, so that is a correct refinement. However, if the only operation defined on both viewpoints is

$$\begin{array}{|l}
\multicolumn{1}{l}{Op_i} \\
\Delta D_i \\
\hline
x' = 2 - x
\end{array}$$

the unification is a refinement in example 6 as well. (If state consistency holds for the inhabitants of all operation schemas, i.e. in this case just 0 and 2, the unification is a refinement. See section 6 for a further discussion of this.) The unification of these two operations will (modulo renaming) be Op_1 in the situation of example 6, and Op_2 in example 7 – quite different operations, but *both* least common refinements. This may seem strange at first.

However, in general in Z any state schema with operations can be data-refined by either *embedding* the state space in a superset (unconditionally), or by *restricting* the state space to a subset *which is closed under all operations*. (In the example, $\{0,2\}$ is indeed closed under $\lambda\, x \bullet 2 - x$.) The operations will be unchanged in the first case, and restricted to the new state space in the latter case. This may result in severely restricted operations; to see this, consider that the rules for data refinement (if there is no initial state) are already satisfied if the retrieve relation is empty (if Abs is false, in the terms of [15]).

So, classes of specifications that are mutual refinements will be (perhaps unexpectedly) large. In the next section we will argue that *not* all of the least common refinements reflect our interpretation of viewpoints, and we will look for criteria for choosing among them.

6 Consistency for partial specification in Z

At this point we are able to assess what consistency means for partial specification in Z. First, we have to observe that our unification method does not generate internal inconsistencies in the sense of section 3. We only produce state and operation schemas, which do not lead to inconsistencies when contradictions occur (rather to uninhabited schemas). The free types we introduce are non-recursive. So we are confident that specifications unified with our method will be consistent, considered on their own, whenever the viewpoint specifications are.

Of course, as became clear in the proofs, a different consistency issue turns up in the case of partial specification: not within a specification, but between specifications. The unification may not always be a refinement of the viewpoints involved, and if it is not, no common refinement satisfying the given correspondence relation exists[8], so an inconsistency between the viewpoints has been found. The condition of operation consistency is clearly a necessary and sufficient one for consistency between viewpoints – however, it can only be checked for operations that operate on the same state, i.e. only when a state unification has been decided on. The choice of a correspondence relation is critical for finding a correct state unification, considering the role that the correspondence plays in determining state consistency.

Let us return briefly to the points in the proofs where we needed "state consistency". It was already claimed there that weaker conditions would also suffice in particular cases, and there is an example supporting that claim in the previous section. The condition we are looking for is that *if* a before-state is linked to a unified state by the state unification's retrieve relation, a possible corresponding after-state should *also* be linked to the unified state by that retrieve relation. State consistency guarantees that by making sure the correspondence relation does not link legal with illegal values. Another option would be to demand that all operations "respect" the correspondence relation, but this would give a quantification over all present and future operations. Also, that would make state unification dependent on operations, which seems to introduce a circular dependency.

So, now we know that state consistency is formally too strong, is it a problem to impose it as a condition on state unification? We should probably let our interpretation come to the rescue here. In general, in Z data refinement it is not necessary for every abstract state to be represented by a concrete state. However, in the examples we have considered so far, the data types defined in the viewpoints included *only* meaningful values that would be just as meaningful in the unification. For a unified state space *not* to represent some values of a viewpoint state space just seems wrong in our interpretation. This is exactly what state consistency prevents. Thus, state consistency may be *formally* too strong for checking that a unification is a refinement, in our *interpretation* it is the right condition even when it is not formally necessary. A methodological advantage of using the state consistency condition is that it greatly simplifies the unification process: state unification can be done independently of operation unification. Thus new operations may be added at any later point without introducing the risk of an invalidated state unification.

A certain way of guaranteeing state consistency is to define R not on $S \times T$ but on its subset $\{x{:}S \mid pred_S\} \times \{x{:}T \mid pred_T\}$.

To summarize: unification of internally consistent viewpoint specifications will result in an internally consistent (candidate) unification. In order to check whether the unification is indeed a common refinement, two types of conditions need to be checked. The *state consistency* condition is formally too strong, but we cannot do better without looking at the operations that have been defined,

[8] Observe that any two specifications are consistent for the empty correspondence relation.

and it conforms with our intuition of state unification. On each pair of operations that is unified we will have to check for *operation consistency*: if both preconditions are satisfied, can both postconditions be satisfied too? The choice of a correspondence relation is crucial for state consistency, and it indirectly also influences operation consistency.

7 Concluding remarks

One might have expected that using Z for partial specification would require a different specification style, a different interpretation, or even a different refinement relation. This paper has shown that for the most part, the states-with-operations style with standard interpretation and refinement will do just fine. Particular interpretations for viewpoint specification occur at two points only:

- Our motivation for imposing state consistency is supported by our interpretation. However, the formal condition of state unification being independent of the operations would lead to the same requirement.

- The notion of a *default correspondence* is clearly dependent on an interpretation of viewpoint specifications. Examples 6 and 7 show that there can hardly be a formal motivation: semantically identical specifications lead to different default correspondence rules. The correspondence relation is *the* parameter in unifying viewpoint specifications; note, however, that we could completely have left out our intuitive ideas about it by not defining a "default" correspondence at all.

The unification method we presented covers only a restricted part of Z: state and operation schemas, in which we have mostly disregarded input and output. These could easily be added to the unification rules. Unification rules for many other Z specification constructs (for example *Init* operations) can be obtained as degenerate cases of the state and operation rules.

This method for unifying two viewpoints has to be embedded in a larger scale unification method. This addresses how to proceed if unifications do not satisfy the consistency criteria – whether and how to choose different correspondence relations, how to determine that no sensible correspondence relation exists and thus viewpoints are fundamentally inconsistent, and how to deal with that [8]. Also, the method will have to be extended from two to an arbitrary number of viewpoints. Fortunately, the binary method appears to be associative up to isomorphism under realistic restrictions on the correspondence relations involved.

As it is presented now, the results of unification contain many complicated expressions due to occurrences of injection functions and "bottoms". We will fix this by adding a "renaming" component to our unification method, which maps $tot\ R$ to some target data type, formalising the "isomorphisms" we appealed to in most of the examples in this paper. The "default" renaming will give the desired result immediately in most cases.

Acknowledgements

We would like to thank Ralph Miarka for his comments on a draft of this paper. The LaTeX code for this paper was generated using the MathSPad editing tool (http://www.win.tue.nl/win/cs/wp/mathspad/) with special stencils for oz.sty.

References

1. M. Ainsworth, A. H. Cruickshank, L. J. Groves, and P. J. L. Wallis. Viewpoint specification and Z. *Information and Software Technology*, 36(1):43–51, February 1994.
2. R. D. Arthan. On free type definitions in Z. In Nicholls [12], pages 40–58.
3. E. Boiten, J. Derrick, H. Bowman, and M.Steen. Unification and multiple views of data in Z. In J.C. van Vliet, editor, *Computing Science in the Netherlands*, pages 73–85, November 1995.
4. T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. *Computer Networks and ISDN Systems*, 14(1):25–29, 1988.
5. H. Bowman, J. Derrick, and M. Steen. Some results on cross viewpoint consistency checking. In Raymond and Armstrong [13], pages 399–412.
6. J. Derrick, H. Bowman, and M. Steen. Maintaining cross viewpoint consistency using Z. In Raymond and Armstrong [13], pages 413–424.
7. J. Derrick, H. Bowman, and M. Steen. Viewpoints and Objects. In J. P. Bowen and M. G. Hinchey, editors, *Ninth Annual Z User Workshop*, LNCS 967, pages 449–468, Limerick, September 1995. Springer-Verlag.
8. A.C.W. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. Inconsistency handling in multiperspective specifications. *IEEE Transactions on Software Engineering*, 20(8):569–578, August 1994.
9. ITU Recommendation X.901-904 — ISO/IEC 10746 1-4. *Open Distributed Processing - Reference Model - Parts 1-4*, July 1995.
10. D. Jackson. Structuring Z specifications with views. Technical Report CMU-CS-94-126, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, 1994.
11. D. Jackson and M. Jackson. Problem decomposition for reuse. *Software Engineering Journal*, 1995. To appear.
12. J. E. Nicholls, editor. *Z User Workshop, York 1991*, Workshops in Computing. Springer-Verlag, 1992.
13. K. Raymond and L. Armstrong, editors. *IFIP TC6 International Conference on Open Distributed Processing*. Chapman and Hall, Brisbane, Australia, February 1995.
14. A. Smith. On recursive free types in Z. In Nicholls [12], pages 3–39.
15. J. M. Spivey. *The Z notation: A reference manual*. Prentice Hall, 1989.
16. P. Zave and M. Jackson. Conjunction as composition. *ACM Transactions on Software Engineering and Methodology*, 2(4):379–411, October 1993.

To appear in Formal Methods Europe, Oxford, March 1996, in LNCS