# Recursive Schema Definitions in Object-Z

Graeme Smith

Software Verification Research Centre
University of Queensland, Australia
smith@svrc.uq.edu.au

**Abstract.** Unlike Z, Object-Z allows schemas to be defined recursively. This enables mutual and self recursive structures, commonly occurring in object-oriented programs, to be readily specified. In this paper, we provide a fixed point interpretation of such definitions. In addition, we provide simple guidelines for producing non-recursive schema definitions which are semantically identical to recursive ones.

## 1 Introduction

Object-Z [7, 2] is an extension of Z [8] to facilitate specification in an object-oriented style. It is a conservative extension in the sense that the existing syntax and semantics of Z are retained and new constructs are added. The major new construct is the *class schema* which captures the object-oriented notion of a class by encapsulating a single state schema, and its associated initial state schema, with all the operation schemas which may change its variables. The class schema is not simply a syntactic extension but also defines a type whose instances are *object references*, i.e., identifiers which reference objects of the class.

The notion of object references in Object-Z is a major departure from the semantics of Z. It allows variables to be declared which, rather than directly representing a value, refer to a value in much the same way as pointers in a programming language. Their introduction facilitates the refinement of specifications to code in object-oriented programming languages.

Object references also have a profound influence on the structuring of specifications. When an object is merely referenced by another object, it is not encapsulated in any way by the referencing object. This enables the possibility of self and mutually recursive structures. To facilitate the specification of such structures, which occur commonly in object-oriented programs, Object-Z does not insist on the notion of "declaration before use" of Z. Specifically, it allows a class schema to include references to its own objects, and initial state and operations schemas to be defined recursively.

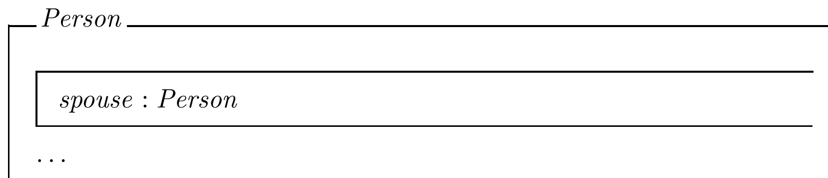
Griffiths [4] has provided a definition for recursive operations in Object-Z. This definition is given in terms of the denotational semantics of Object-Z [3, 5]. Therefore, it relies on some knowledge of this semantics and is not readily accessible to the average Object-Z user. In particular, it does not suggest a straightforward way to represent recursive operation definitions non-recursively to enable them to be more easily understood.

In this paper, we provide definitions of recursion for both initial state schemas and operations. Our approach is a simple application of fixed point theory which does not require an understanding of Object-Z's denotational semantics. Furthermore, our definitions provide an interpretation of recursive schemas in terms of semantically equivalent non-recursive definitions. In Section 2, we discuss and provide examples of recursion in Object-Z. In particular, we specify an ordered binary tree. In Section 3, we provide an overview of fixed point theory and show how it can be used to provide definitions for recursive initial state schema and operation definitions. In Section 4, we use the fixed point definitions to provide guidelines for representing recursive schemas in terms of non-recursive ones. We apply these guidelines to the ordered binary tree specification in Section 5.

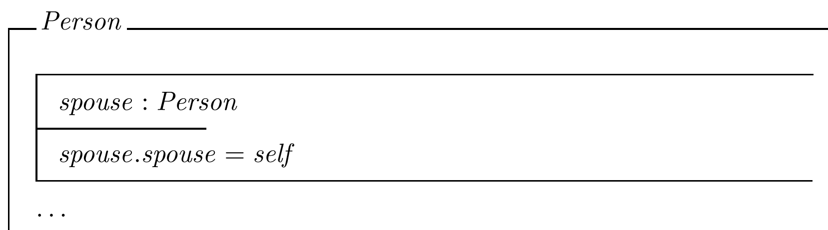
## 2 Recursion in Object-Z

The specification of recursive structures in Object-Z is facilitated by its reference semantics. The type corresponding to a class schema is a set of identifiers which reference objects of the class. That is, the objects they reference behave according to the definitions in the class schema. The identifiers themselves, however, are independent of these definitions.

This independence allows us to relax the notion of "definition before use" of Z. This notion prevents, for example, a schema $S$ including a variable whose value is an instance of $S$. In Object-Z, however, a class may contain a state variable that is an instance of the type defined by that class. For example, the following specification is allowed. (An ellipsis is used to denote elided initial state and operation definitions.)

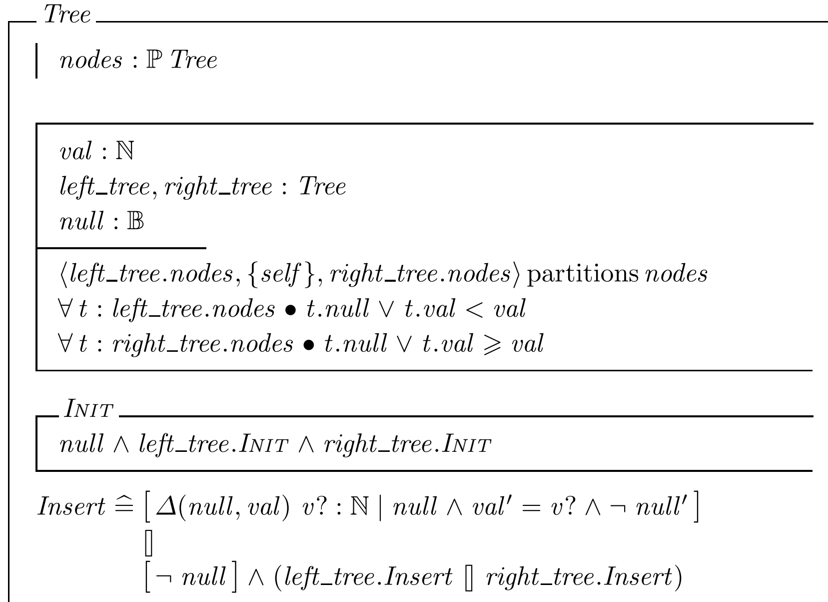



Each class in Object-Z has an implicitly declared constant *self* which for a given object of the class denotes that object's identifier. This enables mutually and self recursive structures such as the following to be specified.




To take full advantage of such recursive structuring, however, we need to be able to refer to the initial state schema and operations of the (recursively)

referenced objects. Object-Z therefore also relaxes the notion of "definition before use" for schemas within classes. For example, consider the following specification of an ordered binary tree based on that in Smith [7].




The class *Tree* describes the functionality of an ordered binary tree abstractly by defining an infinite tree structure, a subset of the nodes of which denote the actual tree. This subset necessarily includes the root node of the infinite structure. An example instance of the class is shown in Figure 1.

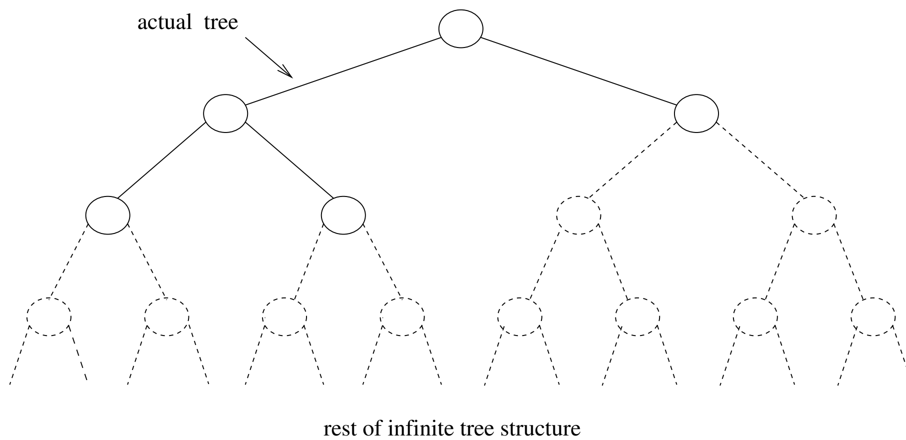



**Fig. 1.** Abstract representation of a binary tree

The state of class *Tree* is defined recursively in terms of two subtrees *left_tree* and *right_tree*. It also has a variable *val* denoting the value in the root node of the tree (or subtree) and a Boolean-valued variable *null* denoting whether or not the root node is part of the actual tree. To facilitate specifying the properties of an ordered binary tree in the state schema's predicate, a constant *nodes* denoting the set of all nodes in the infinite structure, is declared.

Initially, the tree is empty (i.e., for all nodes, *null* is true). In class *Tree*, this is specified recursively. The predicate of the initial state schema states that the root node is not part of the tree (*null* is true) and that its left and right subtrees are also in their initial states.

A node is added to the tree when a value is inserted into that node. The operation *Insert* is also specified recursively. If the root node is not part of the tree then the input value $v?$ is inserted into it and it is added to the tree. If, on the other hand, the root node is already part of the tree then the *Insert* operation is applied to either the left or right subtree ( $[\!]$ denotes angelic choice). The final two predicates of the state schema must be true after the operation and, hence, determine which of the subtrees *Insert* is applied to.

The recursive schema definitions greatly simplify the specification of the ordered binary tree. However, for each definition, we need to be certain that a solution, in the form of a schema, satisfying the definition exists, and that we can uniquely choose a solution if more than one exists. Furthermore, we would like to be able to reexpress the definitions in a non-recursive fashion in order to facilitate reasoning about the schemas and their class.

## 3 Fixed Point Definitions

To guarantee that a recursive schema definition has a solution, we prove, in this section, the existence of a least fixed point. We do this according to the fixed point theory presented in Back and von Wright which allows unbounded nondeterminism within the constructs being defined recursively [1].

Let the domain of the recursively defined construct be $D$. To prove the existence of a least fixed point, we need to find a *complete lattice* $\sqsubseteq_D$ on $D$. That is, a partial order for which there exists, for any subset $s$ of $D$, a greatest lower bound $glb_D(s)$ and least upper bound $lub_D(s)$. This is formalised as follows.

$$
\begin{array}{l}
\sqsubseteq_D : D \leftrightarrow D \\
glb_D, lub_D : \mathbb{P}\, D \rightarrow D \\
\hline
\forall\, d : D \bullet d \sqsubseteq_D d \\
\forall\, d_1, d_2 : D \bullet d_1 \sqsubseteq_D d_2 \wedge d_2 \sqsubseteq_D d_1 \Rightarrow d_1 = d_2 \\
\forall\, d_1, d_2, d_3 : D \bullet d_1 \sqsubseteq_D d_2 \wedge d_2 \sqsubseteq_D d_3 \Rightarrow d_1 \sqsubseteq_D d_3 \\
\forall\, s : \mathbb{P}\, D \bullet \\
\quad (\forall\, d : s \bullet glb_D(s) \sqsubseteq_D d \wedge d \sqsubseteq_D lub_D(s)) \wedge \\
\quad (\forall\, lb : D \mid (\forall\, d : s \bullet lb \sqsubseteq_D d) \bullet lb \sqsubseteq_D glb_D(s)) \wedge \\
\quad (\forall\, ub : D \mid (\forall\, d : s \bullet d \sqsubseteq_D ub) \bullet lub_D(s) \sqsubseteq_D ub)
\end{array}
$$

We then need to reformulate the recursive schema definition as a (non-recursive) function $f_D$ which is monotone. Monotonicity of $f_D$ is formalised as follows.

$$
\begin{array}{|l}
f_D : D \rightarrow D \\
\hline
\forall\, d_1, d_2 : D \bullet d_1 \sqsubseteq_D d_2 \Rightarrow f_D(d_1) \sqsubseteq_D f_D(d_2)
\end{array}
$$

Given these definitions, according to the Knaster-Tarski theorem, $f_D$ has a least fixed point $\mu_D$. This least fixed point is chosen as the unique solution of the recursive definition.

To calculate the value of $\mu_D$, we need to find the "limit" of applying $f_D$ to the bottom element of the lattice, i.e., to $glb_D(D)$. In other words, we need to apply $f_D$ to $glb_D(D)$, then apply $f_D$ to the result of this application of $f_D$, and continue in this fashion until we reach a value $d$ such that $f_D(d) = d$. This value $d$ is the least fixed point $\mu_D$.

If there is a possibility of unbounded nondeterminism in elements of $D$, it is not possible to prove $f_D$ is continuous. Hence, it is not possible to use the theory of Scott which defines this limit over the natural numbers [6]. Instead, the limit is defined over the *ordinals* [1]. The ordinals $\mathbb{O}$ extend the natural numbers, 0 to $\omega$, with the additional elements $\omega + 1, \omega + 2, \ldots, 2 * \omega, 2 * \omega + 1, \ldots$. The least fixed point $\mu_D$ is defined below.

$$
\begin{array}{|l}
\mu_D : D \\
\hline
\exists\, \gamma : \mathbb{O} \bullet \\
\qquad f_D^{\gamma}(glb_D(D)) = f_D^{\gamma+1}(glb_D(D)) \wedge \\
\qquad \mu_D = f_D^{\gamma}(glb_D(D))
\end{array}
$$

### 3.1 Initial State Schemas

An initial state schema in Object-Z comprises a predicate part only [7]. Not all recursive initial state schema definitions have a solution. For example, the definition $\mathit{INIT} \mathrel{\widehat=} [\neg\, \mathit{INIT}]$ requires $\neg\, \mathit{INIT}$ when $\mathit{INIT}$. Similarly, the semantically identical definitions $\mathit{INIT} \mathrel{\widehat=} [\mathit{INIT} \Rightarrow \text{false}]$ and $\mathit{INIT} \mathrel{\widehat=} [\mathit{INIT} \Leftrightarrow \text{false}]$ have no solutions. This suggests the need for a proof obligation to show that initial state schemas have a solution.

In this section, we show that all recursive initial state schema definitions whose predicates are expressed, or can be reexpressed, with certain restrictions on the occurrences of $\mathit{INIT}$ have a least fixed point. Specifically, these restrictions prevent $\mathit{INIT}$ occurring in a predicate $p$ where, for any predicate $q$ and declaration $d$, the predicate $\neg\, p$, $p \Leftrightarrow q$, $p \Rightarrow q$, $\exists_1\, d \bullet p$ or $\forall\, d \mid p \bullet q$ appears as part of the predicate of the initial state schema. (Note that $\forall\, d \mid p \bullet q$ is equivalent to $\forall\, d \bullet p \Rightarrow q$, and hence causes the same problem as $p \Rightarrow q$, and that $\exists_1\, d \bullet p$ is defined in terms of $\forall\, d \mid p \bullet q$ [8].)

Given predicates $p$ and $q$ which respect this restriction on occurrences of $\mathit{INIT}$, we define a relation $\sqsubseteq_{\mathit{INIT}}$ on initial state schemas such that

$$[\, p \,] \sqsubseteq_{\mathit{INIT}} [\, q \,]$$
$$\Leftrightarrow$$
$$(\forall\, d \bullet q \Rightarrow p)$$

where $d$ declares all variables occurring free in $p$ and $q$.

Since implication is reflexive, anti-symmetric and transitive, the relation $\sqsubseteq_{\mathit{INIT}}$ is a partial order. Furthermore, it is a complete lattice [1]. For a finite, non-empty sets of predicates $s = \{p_1, p_2, \ldots, p_n\}$ the greatest lower bound $glb_{\mathit{INIT}}(s)$ is $p_1 \vee p_2 \vee \ldots \vee p_n$ and the least upper bound $lub_{\mathit{INIT}}(s)$ is $p_1 \wedge p_2 \wedge \ldots \wedge p_n$.

Given an initial state schema definition $\mathit{INIT} \mathrel{\widehat=} [\, p \,]$, the function representing this definition maps a given parameter $x$ to the right-hand side of the definition with all occurrences of the left-hand side, i.e., $\mathit{INIT}$, replaced by $x$. (We let the meta-notation $s(a/b)$ denote $s$ with all occurrences of $b$ replaced by $a$.)

$$f_{\mathit{INIT}} = \lambda\, x \bullet [\, p \,](x/\mathit{INIT})$$

The function $f_{\mathit{INIT}}$ is monotone as shown below. (Predicates $q$, $r$, $s$ and $t$ respect the restriction on occurrences of $\mathit{INIT}$. $d$ declares all variables occuring free in $q$, $r$, $s$ and $t$.)

**Theorem**
If $\forall\, d \bullet q \Rightarrow r$ then $\forall\, d \bullet f_{\mathit{INIT}}(q) \Rightarrow f_{\mathit{INIT}}(r)$.

**Proof**
The proof is by induction on the structure of $f_{\mathit{INIT}}$.

1. If $\forall\, d \bullet q \Rightarrow r$ then $\forall\, d \bullet q \wedge s \Rightarrow r \wedge s$ and $\forall\, d \bullet q \vee s \Rightarrow r \vee s$. Therefore, the theorem holds for any $f_{\mathit{INIT}}$ which returns predicates constructed from just $\wedge$ and $\vee$.
   For example, if $f_{\mathit{INIT}} = \lambda\, x \bullet x \wedge (q \vee (x \wedge r))$, where $x$ does not occur in $q$ or $r$, then for any predicates $a$ and $b$, given that all free variables of $a$, $b$, $q$ and $r$ are declared by $d$ and $\forall\, d \bullet a \Rightarrow b$, $\forall\, d \bullet a \wedge r \Rightarrow b \wedge r$ and, therefore, $\forall\, d \bullet q \vee (a \wedge r) \Rightarrow q \vee (b \wedge r)$ and, therefore, $\forall\, d \bullet a \wedge (q \vee (a \wedge r)) \Rightarrow b \wedge (q \vee (b \wedge r))$. That is, $\forall\, d \bullet f_{\mathit{INIT}}(a) \Rightarrow f_{\mathit{INIT}}(b)$.
2. If $\forall\, d \bullet q \Rightarrow r$ then $\forall\, d \bullet (s \Rightarrow q) \Rightarrow (s \Rightarrow r)$. Therefore, the theorem holds for any $f_{\mathit{INIT}}$ which returns predicates constructed from $\wedge$, $\vee$ and $\Rightarrow$ provided that $\mathit{INIT}$ does not occur on the left-hand side of an implication.
3. If $\forall\, d \bullet q \Rightarrow r$ then, for any declaration $d_1$, $\forall\, d \bullet (\forall\, d_1 \bullet q) \Rightarrow (\forall\, d_1 \bullet r)$ and $\forall\, d \bullet (\exists\, d_1 \bullet q) \Rightarrow (\exists\, d_1 \bullet r)$. Therefore, the theorem holds for any $f_{\mathit{INIT}}$ which returns predicates constructed from $\forall$, $\exists$, $\wedge$, $\vee$ and $\Rightarrow$ provided that $\mathit{INIT}$ does not occur on the left-hand side of an implication. (Note that this step includes predicates $\forall\, d \mid p \bullet q$, which is equivalent to $\forall\, d \bullet p \Rightarrow q$, and $\exists\, d \mid p \bullet q$, which is equivalent to $\exists\, d \bullet p \wedge q$.)

Since parts of predicates not including $\mathit{INIT}$ can be constructed in terms of any operators (cf., predicates $q$ and $r$ in the example in step 1 above), it follows

that if $f_{\textit{INIT}}$ returns predicates that respect the restriction on occurrences of $\textit{INIT}$ then the theorem holds.
□

Hence, the least fixed point of a recursive schema definition exists when the occurrences of $\textit{INIT}$ in its predicate are appropriately restricted. Since for all predicates $p$ and declarations of $p$'s free variables $d$, $\forall\, d \bullet p \Rightarrow \text{true}$, the bottom element of $\sqsubseteq_{\textit{INIT}}$ is $[\,\text{true}\,]$. Given the initial state schema definition $\textit{INIT} \mathrel{\widehat{=}} [\,p\,]$, we therefore have the following unique value for $\textit{INIT}$.

$$\textit{INIT} = f^{\gamma}_{\textit{INIT}}([\,\text{true}\,])$$

where $\gamma \in \mathbb{O}$ such that $f^{\gamma}_{\textit{INIT}}([\,\text{true}\,]) = f^{\gamma+1}_{\textit{INIT}}([\,\text{true}\,])$.

### 3.2 Operation Schemas

An operation schema comprises a declaration and a predicate part and a $\Delta$-list (listing the state variables which may change). Given declarations $d_1$ and $d_2$, predicates $p$ and $q$ and lists of variables $u$ and $v$, we define a relation $\sqsubseteq_{Op}$ on operations such that[1]

$$\begin{array}{l} [\,\Delta(u)\ d_1 \mid p\,] \sqsubseteq_{Op} [\,\Delta(v)\ d_2 \mid q\,] \\ \Leftrightarrow \\ \{u\} \subseteq \{v\} \wedge d_1 \subseteq d_2 \wedge (\forall\, d \bullet p \Rightarrow q) \end{array}$$

where $d$ declares all variables occurring free in $p$ and $q$.

Note that, for convenience, we treat declarations as sets of basic declarations of the form $x : T$. (Hence, the $\subseteq$ symbol between $d_1$ and $d_2$ above is a meta-logical operator as are the $\wedge$ symbols.)

It is easy to show that the relation $\sqsubseteq_{Op}$ is reflexive, anti-symmetric and transitive. Hence, it is a partial order. Furthermore, it is a complete lattice [1]. The greatest lower bound of a set $s = \{[\,\Delta(u_1)\ d_1 \mid p_1\,], \ldots, [\,\Delta(u_n)\ d_n \mid p_n\,]\}$ is $[\,\Delta(u_1 \cap \ldots \cap u_n)\ d_1 \cap \ldots \cap d_n \mid p_1 \wedge \ldots \wedge p_n\,]$. The least upper bound of $s$ is $[\,\Delta(u_1 \cup \ldots \cup u_n)\ d_1 \cup \ldots \cup d_n \mid p_1 \vee \ldots \vee p_n\,]$.

Object-Z does not permit operations to appear in declarations and predicates [7]. Therefore, all recursive operations are defined as operation expressions. Such expressions may be constructed using the operation operators $\wedge$, $\parallel$, $\parallel_!$, $[\!]$, $\fatsemi$ and $\bullet$ (scope enrichment) and may involve hiding and renaming [7].

Smith [7] shows how arbitrary operation expressions can be expressed in terms of operation schemas. For each of the operators, the $\Delta$-list of the resulting operation schema is the union of the $\Delta$-lists of the argument operations. Also, the schema part of the resulting operation schema can be defined in terms of the Z schema operators $\forall$, $\exists$, $\wedge$, $\vee$ and renaming. For example, the schema part of an operation formed using the sequential composition operator $\fatsemi$ is simply the

[1] This ordering is not the refinement ordering on operations.

conjunction of the argument schemas with the intermediate state and communicated[2] variables renamed (so that they are identified) and hidden. Hiding is defined in terms of $\exists$.

Given an operation definition $Op \mathrel{\widehat=} OP$ (where $OP$ is an operation expression), the result of the function representing the definition is formed by replacing all occurrences of $Op$ in $OP$ by the parameter $x$. That is,

$$f_{Op} = \lambda\, x \bullet OP(x/Op)$$

The function $f_{Op}$ is monotone as shown below.

**Theorem**
If $\{u\} \subseteq \{v\}$ and $d_1 \subseteq d_2$ and $\forall\, d \bullet p \Rightarrow q$ and $f_{Op}([\,\Delta(u)\ d_1 \mid p\,]) = [\,\Delta(u')\ d_1' \mid p'\,]$ and $f_{Op}([\,\Delta(v)\ d_2 \mid q\,]) = [\,\Delta(v')\ d_2' \mid q'\,]$ then $\{u'\} \subseteq \{v'\}$ and $d_1' \subseteq d_2'$ and $\forall\, d \bullet p' \Rightarrow q'$.

**Proof**

1. Since all operation operators form the union of the $\Delta$-lists of their argument operations, if $\{u\} \subseteq \{v\}$ then $\{u'\} \subseteq \{v'\}$.
   For example, if $f_{Op} = \lambda\, x \bullet x \wedge [\,\Delta(w)\ d_3 \mid p_3\,]$ then $\{u'\} = \{u\} \cup \{w\}$ and $\{v'\} = \{v\} \cup \{w\}$.
2. Since all operation operators are defined in terms of Z schema operators and these latter operators merge declarations, if $d_1 \subseteq d_2$ then $d_1' \subseteq d_2'$.
3. From steps 1, 2 and 3 of the proof of Section 3.1, it follows that if $\forall\, d \bullet p \Rightarrow q$ then $\forall\, d \bullet p' \Rightarrow q'$ provided that $f_{Op}$ constructs $p'$ and $q'$ using only $\forall$, $\exists$, $\wedge$ and $\vee$. Also, since given that $\forall\, d \bullet p \Rightarrow q$ then $\forall\, d \bullet p(x/y) \Rightarrow q(x/y)$, it follows that $\forall\, d \bullet p' \Rightarrow q'$ provided that $f_{Op}$ constructs $p'$ and $q'$ using $\forall$, $\exists$, $\wedge$, $\vee$ and renaming.

Since all operation operators are defined in terms of the Z schema operators $\forall$, $\exists$, $\wedge$, $\vee$ and renaming (and these in turn are defined in terms of the predicate operators $\forall$, $\exists$, $\wedge$ and $\vee$ and renaming), it follows that if $f_{Op}$ returns a valid Object-Z operation expression then the theorem holds.
$\square$

Hence, the least fixed point exist for all recursive operation definitions. Since for all sets $s$, $\varnothing \subseteq s$, and for all predicates $p$ and declarations of $p$'s free variables $d$, $\forall\, d \bullet \text{false} \Rightarrow p$, the bottom element of $\sqsubseteq_{Op}$ is $[\,\text{false}\,]$. Given the operation definition $Op \mathrel{\widehat=} OP$, we therefore have the following unique value for $Op$.

$$Op = f_{Op}^{\gamma}([\,\text{false}\,])$$

where $\gamma \in \mathbb{O}$ such that $f_{Op}^{\gamma}([\,\text{false}\,]) = f_{Op}^{\gamma+1}([\,\text{false}\,])$.

[2] In Object-Z, sequential composition combines the notions of piping and sequential composition of Z [7].

## 4 Interpreting Recursive Definitions

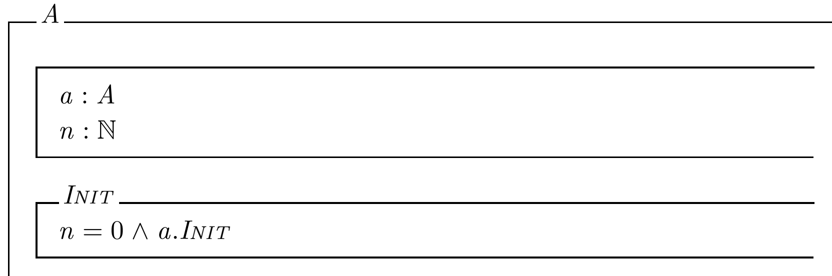
In this section, we show how, from the fixed point definitions, we can derive non-recursive schema definitions which are semantically equivalent to the recursive ones. This is done by calculating the results of successive applications of the function $f_D$ to the bottom of the associated lattice $\sqsubseteq_D$.

The ordinals $0, \omega, 2 * \omega, \ldots$ are referred to as limit ordinals [1]. The successive application of the function $f_D$ to an element $d : D$ is calculated differently for limit ordinals and arbitrary ordinals $\alpha$ as shown below [1].

$f_D^0(d) = d$
$f_D^{\alpha+1}(d) = f_D(f_D^{\alpha}(d))$ for arbitrary ordinals $\alpha$
$f_D^{\alpha}(d) = lub_D(\{\beta \mid \beta < \alpha \bullet f_D^{\beta}(d)\})$ for nonzero limit ordinals $\alpha$

### 4.1 Initial State Schemas

In this section, we present, via an example, a general method for deriving a non-recursive initial state schema from a recursive definition. We use the fixed point theory of Section 3.1, and hence require that the recursive definition respects the restrictions on the occurrences of *INIT*. Consider the following recursively defined class.




Initially, an object of class $A$ has $n$ equal to zero. Furthermore, the object referenced by $a$ also has $n$ equal to zero, and so on. To represent the initial state schema non-recursively, we need to extend the class by introducing a secondary variable (one whose value can be derived from the values of the other variables [7]) in order to be able to refer to all objects referenced by the class. This variable $s$ models the infinite sequence of objects $self, a, a.a, a.a.a, \ldots$. That is, class $A$ is replaced by the semantically equivalent class $A1$. (By including a visibility list $\upharpoonright(\ldots)$ which does not include $s$, $s$ is effectively removed from the class's interface.)

$A1$

$\upharpoonright(a, n, \mathit{INIT})$

$a : A$
$n : \mathbb{N}$
$\Delta$
$s : \mathbb{N} \to A$

---

$s(0) = \mathit{self}$
$\forall\, i : \mathbb{N} \bullet s(i+1) = s(i).a$

$\mathit{INIT}$

$n = 0 \wedge a.\mathit{INIT}$

For this class, $f_{\mathit{INIT}}$ is equal to $\lambda\, x \bullet [\, n = 0 \wedge a.x \,]$. Applying this function to the bottom element of the initial state schema lattice, $[\, \text{true} \,]$, we have

$$\begin{aligned}
f_{\mathit{INIT}}([\,\text{true}\,]) &= [\, n = 0 \wedge a.[\,\text{true}\,] \,] \\
&= [\, n = 0 \,] \\
f^2_{\mathit{INIT}}([\,\text{true}\,]) &= [\, n = 0 \wedge a.[\, n = 0 \,] \,] \\
&= [\, n = 0 \wedge a.n = 0 \,] \\
f^3_{\mathit{INIT}}([\,\text{true}\,]) &= [\, n = 0 \wedge a.[\, n = 0 \wedge a.n = 0 \,] \,] \\
&= [\, n = 0 \wedge a.n = 0 \wedge a.a.n = 0 \,] \\
&= [\, s(0).n = 0 \wedge s(1).n = 0 \wedge s(2).n = 0 \,]
\end{aligned}$$

and so on. It is easy to see that for an arbitrary natural number $\nu$ that

$$f^{\nu}_{\mathit{INIT}}([\,\text{true}\,]) = [\, \forall\, i : 0 \mathrel{..} \nu - 1 \bullet s(i).n = 0 \,]$$

Note that the expression $\mathit{self}.n$ is equivalent to $n$.

Hence, at the first non-zero limit ordinal $\omega$, we have

$$\begin{aligned}
f^{\omega}_{\mathit{INIT}}([\,\text{true}\,]) &= \mathit{lub}_{\mathit{INIT}}(\{\nu : \mathbb{N} \bullet f^{\nu}_{\mathit{INIT}}([\,\text{true}\,])\}) \\
&= \bigwedge \nu : \mathbb{N} \bullet f^{\nu}_{\mathit{INIT}}([\,\text{true}\,]) \\
&= [\, \forall\, i : \mathbb{N} \bullet s(i).n = 0 \,]
\end{aligned}$$

Applying the function again, we have

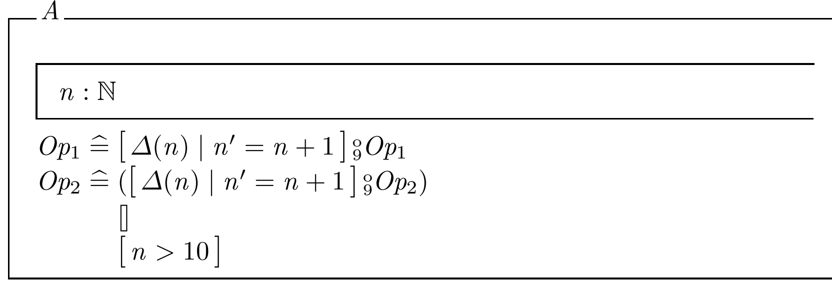
$$\begin{aligned}
f^{\omega+1}_{\mathit{INIT}}([\,\text{true}\,]) &= [\, n = 0 \wedge a.[\, \forall\, i : \mathbb{N} \bullet s(i).n = 0 \,] \,] \\
&= [\, n = 0 \wedge \forall\, i : \mathbb{N} \bullet a.s(i).n = 0 \,] \\
&= [\, \forall\, i : \mathbb{N} \bullet s(i).n = 0 \,]
\end{aligned}$$

Therefore, $f^{\omega}_{\mathit{INIT}}([\,\text{true}\,]) = f^{\omega+1}_{\mathit{INIT}}([\,\text{true}\,])$ and so, according to the theory in Section 3.1,

$$\mathit{INIT} = f^{\omega}_{\mathit{INIT}}([\,\text{true}\,]) = [\, \forall\, i : \mathbb{N} \bullet s(i).n = 0 \,]$$

### 4.2 Operation Schemas

We now look at using the fixed point theory of Section 3.2, to derive non-recursive operation schemas from recursive definitions. As in the previous section, we will illustrate the general approach via examples. Consider the following Object-Z class from Smith [7].



It is not immediately clear what the meaning of $Op_1$ is (the recursion never terminates[3]). However, our theory is valid for all operation expressions and so we should be able to find a least fixed point. The function corresponding to this operation definition is

$$f_{Op1} = \lambda\, x \bullet [\, \Delta(n) \mid n' = n + 1\,] \mathbin{\text{;}} x$$

Applying this function to the bottom of the operation lattice, i.e., to $[\,\text{false}\,]$, we get

$$\begin{aligned} f_{Op1}([\,\text{false}\,]) &= [\, \Delta(n) \mid n' = n + 1\,] \mathbin{\text{;}} [\,\text{false}\,] \\ &= [\,\text{false}\,] \end{aligned}$$

Since $f^0_{Op1}([\,\text{false}\,])$ is also equal to $[\,\text{false}\,]$ by definition, we have $f^0_{Op1}([\,\text{false}\,]) = f_{Op1}([\,\text{false}\,])$. Therefore,

$$Op_1 = [\,\text{false}\,]$$

That is, the operation has a false precondition and can, therefore, never occur [7]. All operations in which recursion cannot terminate will similarly be equivalent to the operation $[\,\text{false}\,]$.

For $Op_2$, the function representing the recursive definition is

$$f_{Op2} = \lambda\, x \bullet ([\, \Delta(n) \mid n' = n + 1\,] \mathbin{\text{;}} x) \mathbin{[\!]} [\, n > 10\,]$$

Applying this to the bottom element $[\,\text{false}\,]$ we have

[3] We use the word "terminates" loosely here. We are dealing with recursive definitions involving sets and predicates, not programs.

$$
\begin{aligned}
f_{Op2}([\,\text{false}\,]) &= ([\,\Delta(n) \mid n' = n+1\,] \mathbin{⨾} [\,\text{false}\,]) \;[]\; [\,n > 10\,] \\
&= [\,n > 10\,] \\
f^2_{Op2}([\,\text{false}\,]) &= ([\,\Delta(n) \mid n' = n+1\,] \mathbin{⨾} [\,n > 10\,]) \;[]\; [\,n > 10\,] \\
&= [\,\Delta(n) \mid n' = n+1 \wedge n' > 10\,] \;[]\; [\,n > 10\,] \\
&= [\,\Delta(n) \mid n' = n+1 \wedge n' > 10\,] \;[]\; f_{Op2}([\,\text{false}\,]) \\
f^3_{Op2}([\,\text{false}\,]) &= ([\,\Delta(n) \mid n' = n+1\,] \\
&\qquad \mathbin{⨾} ([\,\Delta(n) \mid n' = n+1 \wedge n' > 10\,] \;[]\; [\,n > 10\,])) \\
&\quad [] \; [\,n > 10\,] \\
&= [\,\Delta(n) \mid n' = n+2 \wedge n' > 10\,] \\
&\quad [] \; [\,\Delta(n) \mid n' = n+1 \wedge n' > 10\,] \;[]\; [\,n > 10\,] \\
&= [\,\Delta(n) \mid n' = n+2 \wedge n' > 10\,] \;[]\; f^2_{Op2}([\,\text{false}\,])
\end{aligned}
$$

Continuing in this fashion, we see that for any natural number $\nu$,

$$
f^{\nu+1}_{Op2}([\,\text{false}\,]) = [\,\Delta(n) \mid n' = n+\nu \wedge n' > 10\,] \;[]\; f^{\nu}_{Op2}([\,\text{false}\,])
$$

Note that $[\,n > 10\,]$ resulting from $f_{Op2}([\,\text{false}\,])$ is semantically identical to $[\,\Delta(n) \mid n' = n+0 \wedge n' > 10\,]$.

Therefore, for the limit ordinal $\omega$ (noting that, for a given list of variables $x$, the schema part of $[\,\Delta(x)\ a\,] \;[]\; [\,\Delta(x)\ b\,]$ is equivalent to $[\,a\,] \vee [\,b\,]$ [7]) we have

$$
\begin{aligned}
f^{\omega}_{Op2}([\,\text{false}\,]) &= [] \;\nu : \mathbb{N} \bullet [\,\Delta(n) \mid n' = n+\nu \wedge n' > 10\,] \\
&= \exists\, \nu : \mathbb{N} \bullet [\,\Delta(n) \mid n' = n+\nu \wedge n' > 10\,] \\
&= [\,\Delta(n) \mid \exists\, \nu : \mathbb{N} \bullet n' = n+\nu \wedge n' > 10\,] \\
&= [\,\Delta(n) \mid n' \geqslant n \wedge n' > 10\,]
\end{aligned}
$$

Applying the function again, we have

$$
\begin{aligned}
f^{\omega+1}_{Op2}([\,\text{false}\,]) &= ([\,\Delta(n) \mid n' = n+1\,] \mathbin{⨾} [\,\Delta(n) \mid n' \geqslant n \wedge n' > 10\,]) \\
&\quad [] \; [\,n > 10\,] \\
&= [\,\Delta(n) \mid n' > n \wedge n' > 10\,] \;[]\; [\,n > 10\,] \\
&= [\,\Delta(n) \mid n' \geqslant n \wedge n' > 10\,]
\end{aligned}
$$

Therefore, $f^{\omega}_{Op2}([\,\text{false}\,]) = f^{\omega+1}_{Op2}([\,\text{false}\,])$ and so, according to the fixed point theory of Section 3.2,

$$
Op_2 = [\,\Delta(n) \mid n' \geqslant n \wedge n' > 10\,]
$$

For classes with recursive object references, we need to add a secondary variable to provide access to all referenced objects as was done in the example of Section 4.1. As a simple example, consider the following class (also from Smith [7]).

$$
\begin{array}{|l}
\hline
B \\
\begin{array}{|l}
\hline
b : B \\
n : \mathbb{N} \\
\hline
\end{array} \\
Op \mathrel{\widehat=} b.Op \mathbin{[\!]} \left[\, \Delta(n) \mid n' = n + 1 \,\right] \\
\hline
\end{array}
$$

It can be extended to the semantically equivalent class $B1$ below.

$$
\begin{array}{|l}
\hline
B1 \\
\upharpoonright(b, n, Op) \\
\begin{array}{|l}
\hline
b : B \\
n : \mathbb{N} \\
\Delta \\
s : \mathbb{N} \to B \\
\hline
s(0) = \mathit{self} \\
\forall\, i : \mathbb{N} \bullet s(i+1) = s(i).b \\
\hline
\end{array} \\
Op \mathrel{\widehat=} b.Op \mathbin{[\!]} \left[\, \Delta(n) \mid n' = n + 1 \,\right] \\
\hline
\end{array}
$$

The function representing the recursive definition of $Op$ is

$$f_{Op} = \lambda\, x \bullet b.x \mathbin{[\!]} \left[\, \Delta(n) \mid n' = n + 1 \,\right]$$

Applying this to $\left[\,\text{false}\,\right]$, we have

$$
\begin{aligned}
f_{Op}(\left[\,\text{false}\,\right]) &= b.\left[\,\text{false}\,\right] \mathbin{[\!]} \left[\, \Delta(n) \mid n' = n + 1 \,\right] \\
&= \left[\, \Delta(n) \mid n' = n + 1 \,\right] \\
f^2_{Op}(\left[\,\text{false}\,\right]) &= b.\left[\, \Delta(n) \mid n' = n + 1 \,\right] \mathbin{[\!]} \left[\, \Delta(n) \mid n' = n + 1 \,\right] \\
&= b.\left[\, \Delta(n) \mid n' = n + 1 \,\right] \mathbin{[\!]} f_{Op}(\left[\,\text{false}\,\right]) \\
f^3_{Op}(\left[\,\text{false}\,\right]) &= b.b.\left[\, \Delta(n) \mid n' = n + 1 \,\right] \mathbin{[\!]} b.\left[\, \Delta(n) \mid n' = n + 1 \,\right] \\
&\quad \mathbin{[\!]} \left[\, \Delta(n) \mid n' = n + 1 \,\right] \\
&= b.b.\left[\, \Delta(n) \mid n' = n + 1 \,\right] \mathbin{[\!]} f^2_{Op}(\left[\,\text{false}\,\right])
\end{aligned}
$$

That is, for an arbitrary natural number $\nu$, we have

$$f^{\nu+1}_{Op}(\left[\,\text{false}\,\right]) = s(\nu).\left[\, \Delta(n) \mid n' = n + 1 \,\right] \mathbin{[\!]} f^{\nu}_{Op}(\left[\,\text{false}\,\right])$$

and, hence,

$$f^{\omega}_{Op}(\left[\,\text{false}\,\right]) = \mathop{[\!]}\, \nu : \mathbb{N} \bullet s(\nu).\left[\, \Delta(n) \mid n' = n + 1 \,\right]$$

Also,

$$
\begin{aligned}
f_{Op}^{\omega+1}([\,\text{false}\,]) = {} & b.(\mathop{[\!]}\ \nu : \mathbb{N} \bullet s(\nu).\,[\,\Delta(n) \mid n' = n+1\,]) \\
& \mathop{[\!]} \\
& [\,\Delta(n) \mid n' = n+1\,] \\
= {} & (\mathop{[\!]}\ \nu : \mathbb{N}_1 \bullet s(\nu).\,[\,\Delta(n) \mid n' = n+1\,]) \\
& \mathop{[\!]} \\
& s(0).\,[\,\Delta(n) \mid n' = n+1\,] \\
= {} & \mathop{[\!]}\ \nu : \mathbb{N} \bullet s(\nu).\,[\,\Delta(n) \mid n' = n+1\,]
\end{aligned}
$$

Therefore,

$$
Op = \mathop{[\!]}\ \nu : \mathbb{N} \bullet s(\nu).\,[\,\Delta(n) \mid n' = n+1\,]
$$

## 5 Tree Example Revisited

In this section, we use the approach developed in Section 4 to provide a non-recursive interpretation of the schemas of the ordered binary tree of Section 2. This example presents us with a slightly more complex structure of referenced objects. To allow access to these objects, we extend the class with a type $\mathit{LeftRight} ::= \mathit{left} \mid \mathit{right}$, as well as a secondary variable $s$ which maps sequences of elements of the type *LeftRight* to the corresponding tree objects as shown in Figure 2.

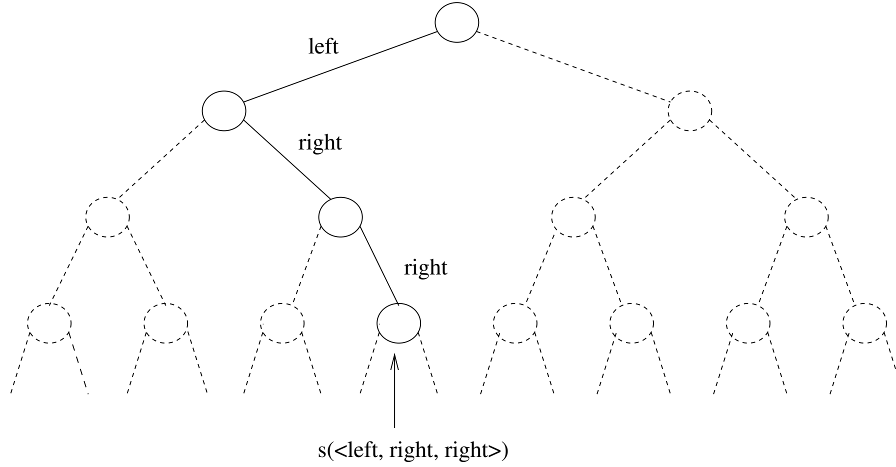



**Fig. 2.** Root node of subtree $s(\langle \mathit{left}, \mathit{right}, \mathit{right} \rangle)$

The state schema of the extended class has the additional secondary variable declaration

$$s : (\mathrm{seq}\, \mathit{LeftRight}) \to \mathit{Tree}$$

and predicate

$$\begin{array}{l} s(\langle\,\rangle) = \mathit{self} \\ \forall\, i : \mathrm{seq}\, \mathit{LeftRight} \bullet \\ \qquad s(i \frown \langle \mathit{left} \rangle) = s(i).\mathit{left_tree} \land \\ \qquad s(i \frown \langle \mathit{right} \rangle) = s(i).\mathit{right_tree} \end{array}$$

### 5.1 Initial State Schema

The function representing the initial state schema of class *Tree* is

$$f_{\mathit{INIT}} = \lambda\, x \bullet \left[\, \mathit{null} \land \mathit{left_tree}.x \land \mathit{right_tree}.x \,\right]$$

Applying this to the bottom element $[\,\mathrm{true}\,]$, we have

$$\begin{array}{ll} f_{\mathit{INIT}}([\,\mathrm{true}\,]) & = [\, \mathit{null} \land \mathit{left_tree}.[\,\mathrm{true}\,] \land \mathit{right_tree}.[\,\mathrm{true}\,]\, ] \\ & = [\, \mathit{null} \,] \\ f^2_{\mathit{INIT}}([\,\mathit{true}\,]) & = [\, \mathit{null} \land \mathit{left_tree}.[\,\mathit{null}\,] \land \mathit{right_tree}.[\,\mathit{null}\,]\, ] \\ & = [\, \mathit{null} \land \mathit{left_tree}.\mathit{null} \land \mathit{right_tree}.\mathit{null} \,] \end{array}$$

$$\begin{array}{ll} f^3_{\mathit{INIT}}([\,\mathrm{true}\,]) & = [\, \mathit{null} \land \mathit{left_tree}.[\, \mathit{null} \land \mathit{left_tree}.\mathit{null} \land \mathit{right_tree}.\mathit{null} \,] \\ & \quad \land \mathit{right_tree}.[\, \mathit{null} \land \mathit{left_tree}.\mathit{null} \land \mathit{right_tree}.\mathit{null} \,]\, ] \\ & = [\, \mathit{null} \land \mathit{left_tree}.\mathit{null} \land \mathit{right_tree}.\mathit{null} \\ & \quad \land \mathit{left_tree}.\mathit{left_tree}.\mathit{null} \land \mathit{left_tree}.\mathit{right_tree}.\mathit{null} \\ & \quad \land \mathit{right_tree}.\mathit{left_tree}.\mathit{null} \land \mathit{right_tree}.\mathit{right_tree}.\mathit{null} \,] \end{array}$$

It is easy to see that

$$f^{\omega}_{\mathit{INIT}}([\,\mathrm{true}\,]) = [\, \forall\, i : \mathrm{seq}\, \mathit{LeftRight} \bullet s(i).\mathit{null} \,]$$

Furthermore, $f^{\omega+1}_{\mathit{INIT}}([\,\mathrm{true}\,]) = f^{\omega}_{\mathit{INIT}}([\,\mathrm{true}\,])$. Therefore,

$$\mathit{INIT} = [\, \forall\, i : \mathrm{seq}\, \mathit{LeftRight} \bullet s(i).\mathit{null} \,]$$

That is, all nodes are initially null. This is what we intuitively expected from the recursive definition.

### 5.2 Insert Operation

Let $\mathit{NodeInsert} \mathrel{\widehat{=}} [\, \Delta(\mathit{null}, \mathit{val})\ v? : \mathbb{N} \mid \mathit{null} \land \mathit{val}' = v? \land \mathit{null}' \,]$. The function representing the operation *Insert* of class *Tree* is

$$\begin{array}{ll} f_{\mathit{Insert}} = \lambda\, x \bullet & \mathit{NodeInsert} \\ & [] \\ & [\, \neg\, \mathit{null} \,] \land (\mathit{left_tree}.x \ [] \ \mathit{right_tree}.x) \end{array}$$

Applying this to the bottom element $[\,\text{false}\,]$, we have

$$
\begin{aligned}
f_{\mathit{Insert}}([\,\text{false}\,]) = {} & \mathit{NodeInsert} \\
& \sqcup \\
& [\,\neg\, \mathit{null}\,] \wedge (\mathit{left_tree}.[\,\text{false}\,] \ \sqcup\ \mathit{right_tree}.[\,\text{false}\,]) \\
= {} & \mathit{NodeInsert}
\end{aligned}
$$

$$
\begin{aligned}
f^2_{\mathit{Insert}}([\,\text{false}\,]) = {} & \mathit{NodeInsert} \\
& \sqcup \\
& [\,\neg\, \mathit{null}\,] \wedge (\mathit{left_tree}.\mathit{NodeInsert} \ \sqcup\ \mathit{right_tree}.\mathit{NodeInsert})
\end{aligned}
$$

$$
\begin{aligned}
f^3_{\mathit{Insert}}([\,\text{false}\,]) = {} & \mathit{NodeInsert} \\
& \sqcup \\
& [\,\neg\, \mathit{null}\,] \\
& \wedge \\
& ((\mathit{left_tree}.\mathit{NodeInsert} \\
& \quad \sqcup \\
& \quad (\neg\, \mathit{left_tree}.\mathit{null} \wedge (\mathit{left_tree}.\mathit{left_tree}.\mathit{NodeInsert} \\
& \qquad\qquad \sqcup \\
& \qquad\qquad \mathit{left_tree}.\mathit{right_tree}.\mathit{NodeInsert}))) \\
& \ \sqcup \\
& \ (\mathit{right_tree}.\mathit{NodeInsert} \\
& \quad \sqcup \\
& \quad (\neg\, \mathit{right_tree}.\mathit{null} \wedge (\mathit{right_tree}.\mathit{left_tree}.\mathit{NodeInsert} \\
& \qquad\qquad \sqcup \\
& \qquad\qquad \mathit{right_tree}.\mathit{right_tree}.\mathit{NodeInsert}))))
\end{aligned}
$$

From this we can see that the *NodeInsert* operation is applied to a node $N$ such that all nodes between the root node and $N$ are not null. The choice of the actual node out of those fulfilling this condition is made angelically. Therefore, we can deduce that[4]

$$
\begin{aligned}
& f^{\omega}_{\mathit{Insert}}([\,\text{false}\,]) = \\
& \quad \bigsqcup\ i : \operatorname{seq} \mathit{LeftRight} \mid (\forall\, j : \operatorname{seq} \mathit{LeftRight} \mid j \subset i \bullet \neg\, s(j).\mathit{null}) \bullet \\
& \qquad\qquad s(i).\mathit{NodeInsert}
\end{aligned}
$$

It also follows that $f^{\omega+1}_{\mathit{Insert}} = f^{\omega}_{\mathit{Insert}}$ and hence

$$
\begin{aligned}
& \mathit{Insert} = \\
& \quad \bigsqcup\ i : \operatorname{seq} \mathit{LeftRight} \mid (\forall\, j : \operatorname{seq} \mathit{LeftRight} \mid j \subset i \bullet \neg\, s(j).\mathit{null}) \bullet \\
& \qquad\qquad s(i).\mathit{NodeInsert}
\end{aligned}
$$

Once again, this is intuitively what we expected. The choice of the node to which *Insert* is applied is further restricted by the state schema's predicate which ensures the tree is ordered.

[4] Note that since $j$ and $i$ are sequences, $j \subset i$ means that $j$ is a prefix of $i$.

## 6 Conclusion

This paper has presented fixed point definitions for recursive initial state and operation schemas in Object-Z. In particular, it has provided a set of conditions under which recursive initial state schemas are consistent. These conditions amount to restrictions on the occurrences of $\mathit{INIT}$ in the predicate of an initial state schema. Also, the paper has shown that all recursive operation schema definitions are consistent.

The primary advantage of fixed point definitions is that they provide a straightforward way of representing recursive schema definitions by semantically equivalent non-recursive ones. The paper illustrates guidelines for doing this via simple examples and shows how to apply these guidelines to a recursive specification of an ordered binary tree.

### Acknowledgements

The author would like to thank Ian Hayes for fruitful discussions on aspects of this work, and for comments on an earlier draft of this paper. This work is funded by Australian Research Council grant number A49801500.

## References

1. R.-J. Back and J. von Wright. *Refinement Calculus: A Systematic Introduction*. Graduate Texts in Computer Science. Springer-Verlag, 1998.
2. R. Duke, G. Rose, and G. Smith. Object-Z: A specification language advocated for the description of standards. *Computer Standards and Interfaces*, 17:511–533, 1995.
3. A. Griffiths. An extended semantic foundation for Object-Z. In *1996 Asia-Pacific Software Engineering Conference (APSEC'96)*, pages 194–207. IEEE Computer Society Press, 1996.
4. A. Griffiths. A semantics for recursive operations in Object-Z. In L. Groves and S. Reeves, editors, *Formal Methods Pacific'97 (FMP'97)*, pages 81–102. Springer-Verlag, 1997.
5. A. Griffiths. *A Formal Semantics to Support Modular Reasoning in Object-Z*. PhD thesis, Software Verification Research Centre, University of Queensland, 1998.
6. D. Scott and C. Gunther. Semantic domains and denotational semantics. In *Handbook of Theoretical Computer Science*, chapter 12, pages 633–674. Elsevier Science Publisher, 1990.
7. G. Smith. *The Object-Z Specification Language*. Advances in Formal Methods. Kluwer Academic Publishers, 2000.
8. J.M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall, 2nd edition, 1992.