SOFTWARE VERIFICATION RESEARCH CENTRE

DEPARTMENT OF COMPUTER SCIENCE

THE UNIVERSITY OF QUEENSLAND

Queensland 4072
Australia

TECHNICAL REPORT

No. 95-26

A Logic for Object-Z
(Additional Rules)

Graeme Smith

September 1995

Phone: +61 7 3365 1003
Fax: +61 7 3365 1533

Note: Most SVRC technical reports are available via anonymous ftp, from `ftp.cs.uq.edu.au` in the directory `/pub/SVRC/techreports`.

A Logic for Object-Z (Additional Rules)

Graeme Smith

Software Verification Research Centre
Department of Computer Science
University of Queensland, Australia

Abstract

This paper extends the logic for Object-Z presented in [5] by adding rules defining the priming of schemas, the schema operator 'pre', the sequential composition operator, class union, containment and the distributed forms of the operation operators. Corrections to the operation definition rule and the class renaming rule presented in [5] are included in the Appendix.

1 Introduction

Object-Z[2, 3, 4] is an extension to the formal specification language Z[1, 6] which facilitates specification in an object-oriented style. A logic for reasoning about Object-Z specifications is presented in [5][1]. The logic is an extension of $\mathcal{W}$[7], a logic advocated for Z and used as the basis for the deductive system in the Z Base Standard[1].

The purpose of this paper is to extend the rules of the logic to cover the following constructs: priming of schemas, the schema operator 'pre', the sequential composition operator (including its use in defining operations recursively), class union, containment and the distributed forms of the operation operators. The first three of these are complicated by the fact that Object-Z has a reference semantics, i.e. instances of classes are references to objects rather than objects themselves. An operation which changes the variable x of an object referenced by the object reference a generally refers to $a.x$ (the value of x before the operation) and $a.x'$ (the value of x after the operation). If a is unchanged by the operation (i.e. its continues to refer to the same object) then $a'.x = a.x$ and $a'.x' = a.x'$. Otherwise, a and a' refer to different objects whose x and x' values are not, in general, related.

Reference semantics also lead to the possibility of aliasing. That is, more than one object reference may refer to the same object. It is important to account for aliasing when defining sequential composition. For example, the operation $a.Op \mathbin{;} b.Op$ either applies the operation Op to the same object twice if $a = b$ or applies the operation to two different objects if $a \neq b$.

Throughout the paper, the following naming conventions are adopted.

d	declarations
p, q	predicates

[1] A familiarity with both Object-Z and the logic is assumed in this paper. Readers not so familiar may wish to refer to [2] and [5] respectively.

s, t	expressions
w, x, y, z	variables
a, b, c, u, v	object references
S	schemas
A, B	classes or class unions
Op	operation names
OP	operations
T	types
λ	rename lists

2 Priming

In Z, priming of schemas can be defined in terms of renaming. A suitable rule would be

$$\frac{S' = S[x_1'/x_1, \ldots, x_n'/x_n] \vdash}{\vdash} \quad (\ \mathit{Prime}\ )\ [\ \alpha(S) = \{x_1, \ldots, x_n\}\ ]$$

In Object-Z, however, this is not sufficient. Consider the following (state) schema where A is a class which has a variable $x : \mathbb{N}$.

$$\begin{array}{|l} a : A \\ \hline a.x = 10 \end{array}$$

Using the rule *Prime* above, priming this schema would result in the following.

$$\begin{array}{|l} a' : A \\ \hline a'.x = 10 \end{array}$$

This is not what we intend; the predicate refers to the x variable of the object referenced by a' *before* the next operation. The x variable of the object referenced by a' *after* the next operation is $a'.x'$.

To define priming in Object-Z, we introduce a meta-function π which transforms expressions of the form $a.x$ into $a.x'$ when x is a primary or secondary variable (as opposed to a constant or type) of a's class. The definition is recursive so that expressions such as $a_1.a_2.x$ are replaced by $a_1.a_2'.x'$ when x and a_2 are variables of the classes of a_2 and a_1 respectively.

$\pi([d \mid p]) \equiv [\pi(d) \mid \pi(p)]$

Figure 1: π applied to schemas.

$\pi(x : T) \equiv x : \pi(T)$
$\pi(d_1;\ d_2) \equiv \pi(d1);\ \pi(d2)$

Figure 2: π applied to declarations.

$\pi(x = y) \equiv \pi(x) = \pi(y)$
$\pi(x \in y) \equiv \pi(x) \in \pi(y)$
$\pi(true) \equiv true$
$\pi(false) \equiv false$
$\pi(\neg\, p) \equiv \neg\, \pi(p)$
$\pi(p \vee q) \equiv \pi(p) \vee \pi(q)$
$\pi(p \wedge q) \equiv \pi(p) \wedge \pi(q)$
$\pi(p \Rightarrow q) \equiv \pi(p) \Rightarrow \pi(q)$
$\pi(p \Leftrightarrow q) \equiv \pi(p) \Leftrightarrow \pi(q)$
$\pi(\forall\, d \mid p \bullet q) \equiv \forall\, \pi(d) \mid \pi(p) \bullet \pi(q)$
$\pi(\exists\, d \mid p \bullet q) \equiv \exists\, \pi(d) \mid \pi(p) \bullet \pi(q)$
$\pi(\exists_1\, d \mid p \bullet q) \equiv \exists_1\, \pi(d) \mid \pi(p) \bullet \pi(q)$

Figure 3: π applied to predicates.

$\pi(x) \equiv x$
$\pi(\{s_1, \ldots, s_n\}) \equiv \{\pi(s_1), \ldots, \pi(s_n)\}$
$\pi(\{d \mid p \bullet q\} \equiv \{\pi(d) \mid \pi(p) \bullet \pi(q)\}$
$\pi(\mathbb{P}\, s) \equiv \mathbb{P}\, \pi(s)$
$\pi(s_1 \times \ldots \times s_n) \equiv \pi(s_1) \times \ldots \times \pi(s_n)$
$\pi((s_1, \ldots, s_n)) \equiv (\pi(s_1), \ldots, \pi(s_n))$
$\pi(\mu\, d \mid p \bullet q) \equiv \mu\, \pi(d) \mid \pi(p) \bullet \pi(q)$
$\pi(s.x) \equiv \pi(s).x$ $\quad$ s is a binding
$\pi(a.x) \equiv \pi(a).x'$ $\quad$ a is a reference to an object of class A and $x \in \alpha_1(A) \cup \alpha_2(A)$
$\pi(a.x) \equiv \pi(a).x$ $\quad$ a is a reference to an object of class A and $x \notin \alpha_1(A) \cup \alpha_2(A)$
$\pi(s\ t) \equiv \pi(s)\ \pi(t)$

Figure 4: π applied to expressions.

The definition of priming in Object-Z is

$$\frac{S' = \pi(S[x_1'/x_1, \ldots, x_n'/x_n]) \vdash}{\vdash} \quad (\ \mathit{Prime}\ )\ [\ \alpha(S) = \{x_1, \ldots, x_n\}\ ]$$

Since π only changes expressions involving object references, this definition can be used for priming Z schemas as well.

3 Precondition

In Z, the schema operator 'pre' can be defined in terms of hiding. A suitable rule would be

$$\frac{\text{pre}\, S = S \setminus (x_1, \ldots, x_n) \vdash}{\vdash} \quad (\ \mathit{Pre}\ )\ [\ \alpha(S) \cap (\mathit{WORD}' \cup \mathit{WORD}!) = \{x_1, \ldots, x_n\}\ ]$$

where WORD' and $\mathit{WORD}!$ are the sets of all words decorated with $'$ and $!$ respectively.

As with priming, the Z rule is not sufficient for Object-Z. Consider the following operation where A is a class which has a variable $x : \mathbb{N}$.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{Op}} \\ \hline
a, a' : A \\ \hline
a.x' = a.x + 1 \\ \hline
\end{array}$$

Using the rule *Pre* above, pre Op would be

$$\begin{array}{|l}
\multicolumn{1}{l}{\text{pre}\, \mathit{Op}} \\ \hline
a : A \\ \hline
\exists\, a' : A \bullet a.x' = a.x + 1 \\ \hline
\end{array}$$

The predicate is equivalent to $a.x' = a.x + 1$. The required predicate is, however, $\exists\, b : A \bullet b.x = a.x + 1$. That is, there exists an object of class A whose x value is one greater than that of a.

To define the pre operator for Object-Z, we introduce a meta-function ρ which returns a set of expression/identifier pairs (a, x) corresponding to the expressions $a.x$ occurring in a schema, declaration, predicate or expression.

$\rho([d \mid p]) = \rho(d) \cup \rho(p)$

Figure 5: ρ applied to schemas.

$\rho(x : T) = \rho(T)$
$\rho(d_1;\ d_2) = \rho(d1) \cup \rho(d2)$

Figure 6: ρ applied to declarations.

$\rho(x = y) = \rho(x \in y) = \rho(x) \cup \rho(y)$
$\rho(true) = \rho(false) = \varnothing$
$\rho(\neg\, p) = \rho(p)$
$\rho(p \vee q) = \rho(p \wedge q) = \rho(p \Rightarrow q) = \rho(p \Leftrightarrow q) = \rho(p) \cup \rho(q)$
$\rho(\forall\, d \mid p \bullet q) = \rho(\exists\, d \mid p \bullet q) = \rho(\exists_1\, d \mid p \bullet q) = \rho(d) \cup \rho(p) \cup \rho(q)$

Figure 7: ρ applied to predicates.

$\rho(x) = \varnothing$
$\rho(\{s_1, \ldots, s_n\}) = \rho(s_1) \cup \ldots \cup \rho(s_n)$
$\rho(\{d \mid p \bullet q\} = \rho(d) \cup \rho(p) \cup \rho(q)$
$\rho(\mathbb{P}\, s) = \rho(s)$
$\rho(s_1 \times \ldots \times s_n) = \rho((s_1, \ldots, s_n)) = \rho(s_1) \cup \ldots \cup \rho(s_n)$
$\rho(\mu\, d \mid p \bullet q) = \rho(d) \cup \rho(p) \cup \rho(q)$
$\rho(s.x) = \rho(s)$ $\quad$ s is a binding
$\rho(a.x) = \{(a, x)\}$
$\rho(s\ t) = \rho(s) \cup \rho(t)$

Figure 8: ρ applied to expressions.

The result of applying pre to an Object-Z operation OP can be formed by replacing all expressions of the form $a.x'$ in the predicate of OP by $u.x$ (where u is a new variable not declared in OP) and hiding the new variables as well as any primed or output variables of OP.

$$\frac{\begin{array}{l} \vdash OP = [d \mid p] \\ OP \vdash a_1 \in A_1 \\ \ldots \\ OP \vdash a_n \in A_n \\ d;\ u_1 : A_1;\ \ldots;\ u_n : A_n \mid u_1.x_1 = a_1.x_1' \wedge \ldots \wedge u_n.x_n = a_n.x_n' \vdash p \Leftrightarrow p_1 \\ \mathrm{pre}\, OP = [d;\ u_1 : A_1;\ \ldots;\ u_n : A_n \mid p_1] \setminus (u_1, \ldots, u_n, y_1, \ldots, y_m) \vdash \end{array}}{\vdash} \quad (\ \mathit{Pre}\ )\ [\ q\ ]$$

$$\begin{array}{ll} \text{where } q \equiv & \rho(p) \rhd WORD' = \{(a_1, x_1'), \ldots, (a_n, x_n')\} \\ & \rho(p_1) \rhd WORD' = \varnothing \\ & \alpha(d) \cap (WORD' \cup WORD!) = \{y_1, \ldots, y_m\} \\ & \{u_1, \ldots, u_n\} \cap \alpha(d) = \varnothing \end{array}$$

The proviso $\rho(p_1) \rhd WORD' = \varnothing$ ensures that p_1 has no expressions $a_i.x_i'$. That is, all occurrences of these expressions in p have been replaced in p_1 by $u_i.x_i$.

If OP has been formed by hiding an object reference a then p will be of the form $\exists\, a : A \bullet p_2$. Since any expression of the form $a.x'$ cannot be replaced in p_2, the above rule cannot be used ($\rho(p_1) \rhd WORD'$ will not be empty). Therefore, the following rule must be used first.

$$\frac{\mathrm{pre}(OP \setminus (a_1, \ldots, a_n)) = (\mathrm{pre}\, OP) \setminus (a_1, \ldots, a_n) \vdash}{\vdash} \quad (\ \mathit{PreHiding}\ )$$

4 Sequential Composition

The sequential composition operator enables the definition of an atomic operation whose effect is the same as performing two (or more) operations in sequence. In addition, outputs of the first operation in a sequential composition are identified with inputs with the same basenames of the second. These identified inputs and outputs are hidden in the resulting operation.

The schema of the operation $OP_1 \mathbin{⨾} OP_2$ can be formed by

- adding to the schema of OP_1 the predicate which states that all primary variables not in OP_1's delta-list are unchanged,
- noting any output variables of OP_1 which have the same basename as an input variable of OP_2,
- renaming each such input variable and the corresponding output variable to a common new variable (not declared in either OP_1 or OP_2),
- replacing all expressions of the form $a.x'$ in OP_1 by $u.x$ where u is a new variable,
- replacing all expressions of the form $b.y$ in OP_2 by $v.y$ where v is a new variable,
- sequentially composing the resulting schemas using Z's sequential composition operator,
- and hiding all the new variables after adding to the resulting schema the predicates detailed below.

1. The first set of predicates equate the values $a.x'$ in the first operation with $b.x$ in the second operation when a and b refer to the same object. There are two cases to consider.

(i) The first case arises when from the predicate of Op_1 and the precondition of Op_2 we can deduce that a and b are equal.

For example, suppose an object reference c is assigned the value of a in the post-state of the first operation and b is equated to c in the pre-state of the second.

$$
\begin{array}{|l}
\multicolumn{1}{l}{Op_1} \\
\hline
\Delta(c) \\
\hline
a.x' = a.x + 1 \\
c' = a \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\multicolumn{1}{l}{Op_2} \\
\hline
\Delta() \\
\hline
b = c \\
b.x' = b.x + 1 \\
\hline
\end{array}
$$

The effect of the operation $Op_1 \mathbin{⨾} Op_2$ is to add two to the x variable of the object referenced by a. The value of b in the second operation is equated with the value of a in the first operation since the sequential composition equates c' in the first operation with c in the second.

Therefore, if p_1 is the predicate of Op_1 (with primary variables not in Op_1's delta-list explicitly unchanged) and p_2 is the predicate of Op_2, when $a.x'$ and $b.x$ have been replaced by $u.x$ and $v.x$ respectively, the predicate we need to add to the schema of $Op_1 \fatsemi Op_2$ is

$$(p_1 \land (\mathrm{pre}[\Delta \mathit{State} \mid p_2])' \Rightarrow a = b') \Rightarrow u = v$$

We use the precondition of the schema $[\Delta \mathit{State} \mid p_2]$ rather than Op_2 because we don't wish to hide the output variables. Since common-named output variables are equated (when that of the first operation is not identified with an input variable in the second), they too can be used to equate a in the first operation with b in the second.

In general, b may be an expression involving the dot notation, a class constant or an auxiliary variable (e.g. an input of output variable). It's post-state value is therefore given by $\pi(\llparenthesis\ \lambda\ \rrparenthesis \odot b)$ where λ renames the variables of the class in which the operation occurs to their primed counterparts. Therefore, the general predicate is of the form

$$(p_1 \land (\mathrm{pre}[\Delta \mathit{State} \mid p_2])' \Rightarrow a = \pi(\llparenthesis\ \lambda\ \rrparenthesis \odot b)) \Rightarrow u = v$$

(ii) The second case arises when a and b are equated in the environment of the sequential composition. For example, consider the sequential composition of the following operations.

Op_1	Op_2
$\Delta()$	$\Delta()$
$a.x' = a.x + 1$	$b.x' = b.x + 1$

Although a and b aren't related by the predicates of Op_1 and Op_2, we must allow for the possibility that $a = b$ in the environment of $Op_1 \fatsemi Op_2$. For example, the expression may be part of the larger expression $[a = b] \bullet (Op_1 \fatsemi Op_2)$. Therefore, given that $a.x'$ has been replaced by $u.x$ and $b.x$ by $v.x$ we need a predicate

$$a = b \Rightarrow u = v$$

In general, this will only be true if b is not changed by the first operation, so given p_1 and λ as above, we have

$$(p_1 \Rightarrow b = \pi(\llparenthesis\ \lambda\ \rrparenthesis \odot b)) \Rightarrow (a = b \Rightarrow u = v)$$

2. The second set of predicates give a value for $b.x$ when the object referenced by b is unchanged by the first operation. Given that the references to objects changed by the first operation are $a_1, \ldots, a_n$, that p_1 and λ are as above and that b references an object of class B, when $b.y$ has been replaced by $v.y$, we require

$$\forall\, c : B \bullet (p_1 \Rightarrow \pi(\llparenthesis\ \lambda\ \rrparenthesis \odot b) = c) \Rightarrow (a_1 \neq c \land \ldots \land a_n \neq c \Rightarrow v = c)$$

3. The final set of predicates give a value for $a.x'$ when the object referenced by a is unchanged by the second operation. Given that the references to objects changed

by the second operation are $c_1, \ldots, c_i$ and that p_1 and λ are as above, when $a.x'$ has been replaced by $u.x$, we require

$$(p_1 \Rightarrow a \neq \pi(\langle\!|\ \lambda\ |\!\rangle \odot c_1) \wedge \ldots \wedge a \neq \pi(\langle\!|\ \lambda\ |\!\rangle \odot c_i)) \Rightarrow u = a$$

The rule for sequential composition is given below[2].

$$\frac{\begin{array}{l}
A :: \vdash [OP_1 \mid z_1' = z_1 \wedge \ldots \wedge z_j' = z_j][w_{h+1}/w_1!, \ldots, w_{2*h}/w_h!] = [d_1 \mid p_1] \\
A :: \vdash OP_2[w_{h+1}/w_1?, \ldots, w_{2*h}/w_h?] = [d_2 \mid p_2] \\
A :: \vdash a_1 \in A_1 \wedge \ldots \wedge a_n \in A_n \\
A :: \vdash b_1 \in B_1 \wedge \ldots \wedge b_m \in B_m \\
A :: d_1;\ u_1 : A_1;\ \ldots;\ u_n : A_n \mid u_1.x_1 = a_1.x_1' \wedge \ldots \wedge u_n.x_n = a_n.x_n' \vdash p_1 \Leftrightarrow p_3 \\
A :: d_2;\ v_1 : b_1;\ \ldots;\ v_m : B_m \mid v_1.y_1 = b_1.y_1 \wedge \ldots \wedge v_m.y_m = b_m.y_m \vdash p_2 \Leftrightarrow p_4 \\
A :: OP_1 \mathbin{⨾} OP_2 = \\
\qquad [u_1 : A_1;\ \ldots;\ u_n : A_n;\ d_1 \mid p_3] \mathbin{⨾} [v_1 : B_1;\ \ldots;\ v_m : B_m;\ d_2 \mid p_4] \bullet \\
\qquad [(p_1 \wedge (\mathrm{pre}[\Delta \mathit{State} \mid p_2])' \Rightarrow a_1 = \pi(\langle\!|\ \lambda\ |\!\rangle \odot b_1)) \Rightarrow u_1 = v_1 \\
\qquad \ldots \\
\qquad (p_1 \wedge (\mathrm{pre}[\Delta \mathit{State} \mid p_2])' \Rightarrow a_n = \pi(\langle\!|\ \lambda\ |\!\rangle \odot b_m)) \Rightarrow u_n = v_m \\
\qquad (p_1 \Rightarrow b_1 = \pi(\langle\!|\ \lambda\ |\!\rangle \odot b_1)) \Rightarrow \\
\qquad\qquad (a_1 = b_1 \Rightarrow u_1 = v_1) \wedge \ldots \wedge (a_n = b_1 \Rightarrow u_n = v_1) \\
\qquad \ldots \\
\qquad (p_1 \Rightarrow (b_m = \pi(\langle\!|\ \lambda\ |\!\rangle \odot b_m)) \Rightarrow \\
\qquad\qquad (a_1 = b_m \Rightarrow u_1 = v_m) \wedge \ldots \wedge (a_n = b_m \Rightarrow u_n = v_m) \\
\qquad \forall\, c : B_1 \bullet (p_1 \Rightarrow \pi(\langle\!|\ \lambda\ |\!\rangle \odot b_1) = c) \Rightarrow (a_1 \neq c \wedge \ldots \wedge a_n \neq c \Rightarrow v_1 = c) \\
\qquad \ldots \\
\qquad \forall\, c : B_m \bullet (p_1 \Rightarrow \pi(\langle\!|\ \lambda\ |\!\rangle \odot b_m) = c) \Rightarrow (a_1 \neq c \wedge \ldots \wedge a_n \neq c \Rightarrow v_m = c) \\
\qquad (p_1 \Rightarrow a_1 \neq \pi(\langle\!|\ \lambda\ |\!\rangle \odot c_1) \wedge \ldots \wedge a_1 \neq \pi(\langle\!|\ \lambda\ |\!\rangle \odot c_i)) \Rightarrow u_1 = a_1 \\
\qquad \ldots \\
\qquad (p_1 \Rightarrow a_n \neq \pi(\langle\!|\ \lambda\ |\!\rangle \odot c_1) \wedge \ldots \wedge a_n \neq \pi(\langle\!|\ \lambda\ |\!\rangle \odot c_i)) \Rightarrow u_n = a_n] \\
\qquad \setminus (u_1, \ldots, u_n, v_1, \ldots, v_m, w_{h+1}, \ldots, w_{2*h}) \vdash
\end{array}}{A ::\ \vdash} \ (\ \mathit{Seq}\ )\ [\ q\ ]$$

$$\begin{array}{ll}
\text{where } q \equiv & \rho(d_1) \cup \rho(d_2) = \varnothing \\
& \beta_!(OP_1) \cap \beta_?(OP_2) = \{w_1, \ldots, w_h\} \\
& \rho(p_1) \rhd \mathit{WORD}' = \{(a_1, x_1), \ldots, (a_n, x_n)\} \\
& \rho(p_2) \rhd \mathit{WORD} = \{(b_1, y_1), \ldots, (b_m, y_m)\} \\
& \mathrm{dom}\, \rho(p_2) \rhd \mathit{WORD}' = \{c_1, \ldots, c_i\} \\
& \rho(p_3) \rhd \mathit{WORD}' = \varnothing \\
& \rho(p_4) \rhd \mathit{WORD} = \{(v_1, y_1), \ldots, (v_m, y_m)\} \\
& \alpha_1(A) \setminus \delta_{OP}(OP_1) = \{z_1, \ldots, z_j\} \\
& \alpha_1(A) \cup \alpha_2(A) = \{z_1, \ldots, z_k\} \\
& \lambda = z_1 \rightsquigarrow z_1', \ldots, z_k \rightsquigarrow z_k' \\
& \{u_1, \ldots, u_n\} \cap \{v_1, \ldots, v_m\} = \varnothing \\
& \{u_1, \ldots, u_n, v_1, \ldots, v_m\} \cap \{w_{h+1}, \ldots, w_{2*h}\} = \varnothing \\
& \{u_1, \ldots, u_n, v_1, \ldots, v_m, w_{h+1}, \ldots, w_{2*h}\} \cap (\alpha(d_1) \cup \alpha(d_2)) = \varnothing
\end{array}$$

[2]In the rule, values of possibly different types are compared (e.g. a_1 and b_1) with the understanding that, if they are of different types, then the comparison evaluates to *false*.

As with the precondition rule, this rule is only applicable when neither OP_1 nor OP_2 have been formed by hiding object references. In the case where they have, one or both of the following rules must be used.

$$\frac{(OP_1 \setminus (a_1, \ldots, a_n)) \fatsemi OP_2 = (OP_1 \fatsemi OP_2) \setminus (a_1, \ldots, a_n) \vdash}{\vdash} \quad (\ \mathit{SeqHiding1}\ )\ [\ q\ ]$$

where $q \equiv \{a_1, \ldots, a_n\} \cap \alpha(OP_2) = \varnothing$

$$\frac{OP_1 \fatsemi (OP_2 \setminus (a_1, \ldots, a_n)) = (OP_1 \fatsemi OP_2) \setminus (a_1, \ldots a_n) \vdash}{\vdash} \quad (\ \mathit{SeqHiding2}\ )\ [\ q\ ]$$

where $q \equiv \{a_1, \ldots, a_n\} \cap \alpha(OP_1) = \varnothing$

5 Class Union

Class union is used to declare an object which belongs to one of an arbitrary set of classes.

If a is a member of a class union then a's class is one of the classes in the class union.

$$\frac{\vdash a \in A_1 \cup \ldots \cup A_n}{\vdash a \in A_1 \vee \ldots \vee a \in A_n} \quad (\ \mathit{ClassUnion}\ )$$

6 Containment

Containment is used to capture the structure of a specified system. See [2] for examples.

An object of type $A_{©}$ is a member of class A.

$$\frac{\vdash a \in A_{©}}{\vdash a \in A} \quad (\ \mathit{ContainedObjectType}\ )$$

The properties of containment are captured by two rules:

1. An object cannot directly or indirectly contain itself.

$$\frac{\begin{array}{l} \vdash a_1 \in A_1 \\ A_1 :: \ \ \vdash a_2 \in A_{2©} \\ \ldots \\ A_{n-1} :: \ \ \vdash a_n \in A_{n©} \end{array}}{\vdash a_1. \ldots .a_n \neq a_1} \quad (\ \mathit{NoSelfContainment}\ )$$

2. An object cannot be directly contained within two distinct objects.

$$\frac{\begin{array}{l} \vdash a_1 \in A_1 \\ \vdash a_2 \in A_2 \\ A_1 :: \ \vdash x \in B_{©} \\ A_2 :: \ \vdash y \in B_{©} \\ \vdash a_1 \neq a_2 \end{array}}{\vdash a_1.x \neq a_2.y} \quad (\ \mathit{UniqueContainment}\ )$$

7 Distributed Operators

Distributed operators for conjunction, parallel and choice are defined in terms of the basic operators as follows. These definitions rely on the schemas $[d \mid p]$ having a finite number of bindings $s_1, \ldots, s_n$.

$$\frac{\begin{array}{l} \vdash \{s_1, \ldots, s_n\} = [d \mid p] \\ \bigwedge\ d \mid p \bullet OP = s_1 \odot OP \wedge \ldots \wedge s_n \odot OP \vdash \end{array}}{\vdash} \quad (\ \mathit{DistConjunction}\ )$$

$$\frac{\begin{array}{l} \vdash \{s_1, \ldots, s_n\} = [d \mid p] \\ \big\|\ d \mid p \bullet OP = s_1 \odot OP \parallel \ldots \parallel s_n \odot OP \vdash \end{array}}{\vdash} \quad (\ \mathit{DistParallel}\ )$$

$$\frac{\begin{array}{l} \vdash \{s_1, \ldots, s_n\} = [d \mid p] \\ \Box\ d \mid p \bullet OP = s_1 \odot OP \ \Box\ \ldots\ \Box\ s_n \odot OP \vdash \end{array}}{\vdash} \quad (\ \mathit{DistChoice}\ )$$

The distributed sequential composition of a set of operations is formed by sequentially composing the operations in any possible order. This is defined using the distributed choice operator as follows.

$$\frac{\begin{array}{l} \vdash \{s_1, \ldots, s_n\} = [d \mid p] \\ ⨟\ d \mid p \bullet OP = \Box\ t : 1 \mathinner{..} n \rightarrowtail \{s_1, \ldots, s_n\} \bullet \\ \qquad\qquad\qquad t(1) \odot OP ⨟ \ldots ⨟ t(n) \odot OP \vdash \end{array}}{\vdash} \quad (\ \mathit{DistSeq}\ )$$

8 Recursion

The sequential composition operator can also be used recursively. This is only meaningful, however, when the recursion can terminate. A recursively defined operation is, therefore, necessarily of the form

$$OP_1 = (OP_2 ⨟ OP_1)\ \Box\ OP_3$$

where it is guaranteed that after a finite number of occurrences of OP_2, OP_2 will not be enabled, i.e. $\neg(\mathbin{\S}\ j : 1 \mathinner{\ldotp\ldotp} n \bullet OP_2)$.

OP_1 is then a choice of 0 up to $n-1$ occurrences of OP_2 followed by OP_3.

$$\frac{\begin{array}{l}\vdash OP_1 = (OP_2 \mathbin{\S} OP_1) \mathbin{[]} OP_3 \\ \vdash \neg(\mathbin{\S}\ j : 1 \mathinner{\ldotp\ldotp} n \bullet OP_2)\end{array}}{\vdash OP_1 = OP_3 \mathbin{[]} ([]\ i : 1 \mathinner{\ldotp\ldotp} n-1 \bullet (\mathbin{\S}\ j : 1 \mathinner{\ldotp\ldotp} i \bullet OP_2) \mathbin{\S} OP_3)} \quad (\ \mathit{SeqRecursion}\ )\ [\ q\ ]$$

$$\text{where } q \equiv \begin{array}{l} i \notin \alpha(OP_2) \cup \alpha(OP_3) \\ j \notin \alpha(OP_2) \end{array}$$

9 Conclusion

This paper has extended the logic for Object-Z in [5] to cover all constructs in the language apart from history invariants. While the rules presented provide an axiomatic semantics of the language constructs they define, their use in reasoning is limited, in some cases, by their complexity. It is envisaged, however, that less complex rules, for deriving particular properties or for specific uses of the constructs, could be derived from the rules presented. It is these rules which would form the basis of a practical proof system.

References

[1] S.M. Brien and J.E. Nicholls. Z Base Standard: Version 1. Technical Report PRG-107, Oxford University Computing Laboratory, 1992.

[2] R. Duke, G. Rose, and G. Smith. Object-Z: A specification language advocated for the description of standards. Technical Report 94-45, Software Verification Research Centre, Department of Computer Science, University of Queensland, Australia, 1994.

[3] G. Rose. Object-Z. In S. Stepney, R. Barden, and D. Cooper, editors, *Object Orientation in Z*, Workshops in Computing, pages 59–77. Springer-Verlag, 1992.

[4] G. Rose and R. Duke. An Object-Z specification of a mobile phone system. In K. Lano and H. Haughton, editors, *Object-Oriented Specification Case Studies*, pages 110–129. Prentice-Hall International, 1993.

[5] G. Smith. A logic for Object-Z. Technical Report 94-48, Software Verification Research Centre, Department of Computer Science, University of Queensland, Australia, 1994.

[6] J.M. Spivey. *The Z Notation: A Reference Manual (2nd Ed.)*. Series in Computer Science. Prentice-Hall International, 1992.

[7] J.C.P. Woodcock and S.M. Brien. $\mathcal{W}$: A logic for Z. In J.E. Nicholls, editor, *Z User Workshop*, Workshops in Computing, pages 77–98. Springer-Verlag, 1992.

Appendix

I. In [5], the rule defining the meaning of an operation is as follows.

> The schema part of an operation consists of the inherited part of the operation conjoined with the schema part of the explicit operation definition interpreted in the local environment of the class enriched with $\Delta\textsc{State}$ (i.e. $\textsc{State} \wedge \textsc{State}'$).
>
> For each operation definition
>
> $Op \mathrel{\widehat=} OP$
>
> in $A[X_1, \ldots, X_n]$ we have
>
> $$\frac{A[t_1, \ldots, t_n] :: \quad Op = \Delta\textsc{State} \bullet (\uparrow Op \wedge b \odot OP) \vdash}{A[t_1, \ldots, t_n] :: \quad \vdash} \quad (\ \mathit{OpDef}\ )\ [\ p\ ]$$
>
> where $p \equiv b = \langle\!|\ X_1 \leadsto t_1, \ldots, X_n \leadsto t_n\ |\!\rangle$

This rule does not allow auxiliary operation variables (e.g. input and output variables) declared in an inherited class' operation to be used in the explicit operation definition in the inheriting class. However, we would like to be able to do this in Object-Z. Furthermore, it is not necessary that $\uparrow Op$ be interpreted in the environment enriched with $\Delta\textsc{State}$.

The rule should, therefore, be as follows.

> For each operation definition
>
> $Op \mathrel{\widehat=} OP$
>
> in $A[X_1, \ldots, X_n]$ we have
>
> $$\frac{A[t_1, \ldots, t_n] :: \quad Op = \Delta\textsc{State} \wedge \uparrow Op \bullet b \odot OP \vdash}{A[t_1, \ldots, t_n] :: \quad \vdash} \quad (\ \mathit{OpDef}\ )\ [\ p\ ]$$
>
> where $p \equiv b = \langle\!|\ X_1 \leadsto t_1, \ldots, X_n \leadsto t_n\ |\!\rangle$

II. In [5], the rule defining class renaming is as follows.

> An inherited class may have attributes and operations renamed. The properties of the resulting class are obtained by applying the renaming to the properties of the original class.
>
> $$\frac{A :: \quad \vdash p}{A[\lambda] :: \quad \vdash \langle\!|\ \lambda\ |\!\rangle \odot p} \quad (\ \mathit{ClassRenaming}\ )$$

This rule is unsound when p has a conjunct of the form $\mathit{STATE} = S$, $\mathit{INIT} = S$ or $Op = S$ (where Op is the name of an operation of A) since the variables declared in S will not be renamed, where appropriate, in its declaration or predicate.

The rule should, therefore, be replaced by the following rules. They allow the meaning of the state schema, initial state schema and operations of the renamed class to be derived. Properties of the local definitions of the renamed class can be derived from the definition of η.

$$\frac{A :: \ \vdash \mathit{STATE} = S}{A[\lambda] :: \ \vdash \mathit{STATE} = S[\lambda]} \quad (\ \mathit{StateRename}\ )$$

$$\frac{A :: \ \vdash \mathit{INIT} = S}{A[\lambda] :: \ \vdash \mathit{INIT} = S[\lambda]} \quad (\ \mathit{InitRename}\ )$$

$$\frac{A :: \ \vdash Op = S}{A[\lambda] :: \ \vdash \langle\!|\ \lambda\ |\!\rangle \odot Op = S[\lambda][\lambda']} \quad (\ \mathit{OpRename}\ )\ [\ p\ ]$$

where $p \equiv \alpha_1(A) \cup \alpha_2(A) \lhd \langle\!|\ \lambda\ |\!\rangle = \langle\!|\ x_1 \mapsto y_1, \ldots, x_n \mapsto y_n\ |\!\rangle$
$\lambda' = x_1' \mapsto y_1', \ldots, x_n' \mapsto y_n'$