Appendix A

Glossary of Z Notation

This appendix presents a glossary of the Z notation used in this thesis. The glossary is based on the glossary of Z notation presented in Hayes [39] with modifications to reflect more closely the more recent Z notation of Spivey [89].

Mathematical Notation

A.1 Definitions and Declarations

Let x, x_k be identifiers and let T, T_k be non-empty, set-valued expressions.

$\mathit{LHS} == \mathit{RHS}$	Definition of LHS as syntactically equivalent to RHS.
$\mathit{LHS}[X_1, X_2, \ldots, X_n] == \mathit{RHS}$	Generic definition of LHS, where $X_1, X_2, \ldots, X_n$ are variables denoting formal parameter sets.
$x : T$	A declaration, $x : T$, introduces a new variable x of type T.
$x_1 : T_1;\ x_2 : T_2;\ \ldots;\ x_n : T_n$	List of declarations.
$x_1, x_2, \ldots, x_n : T$	$\mathrel{\widehat=} x_1 : T;\ x_2 : T;\ \ldots;\ x_n : T$

$[X_1, X_2, \ldots, X_n]$	Introduction of free types named $X_1, X_2, \ldots, X_n$.

A.2 Logic

Let P, Q be predicates and let D be a declaration or a list of declarations.

$\mathit{true}, \mathit{false}$	Logical constants.
$\neg\, P$	Negation: "not P".
$P \land Q$	Conjunction: "P and Q".
$P \lor Q$	Disjunction: "P or Q or both".
$P \Rightarrow Q$	$\mathrel{\widehat=} (\neg\, P) \lor Q$ Implication: "P implies Q" or "if P then Q".
$P \Leftrightarrow Q$	$\mathrel{\widehat=} (P \Rightarrow Q) \land (Q \Rightarrow P)$ Equivalence: "P is logically equivalent to Q".
$\forall\, x : T \bullet P$	Universal quantification: "for all x of type T, P holds".
$\exists\, x : T \bullet P$	Existential quantification: "there exists an x of type T such that P holds".
$\exists_1\, x : T \bullet P$	Unique existence: "there exists a unique x of type T such that P holds".
$\forall\, x_1 : T_1;\ x_2 : T_2;\ \ldots;\ x_n : T_n \bullet P$	"For all x_1 of type T_1, x_2 of type T_2, …, and x_n of type T_n, P holds."
$\exists\, x_1 : T_1;\ x_2 : T_2;\ \ldots;\ x_n : T_n \bullet P$	Similar to $\forall$.
$\exists_1\, x_1 : T_1;\ x_2 : T_2;\ \ldots;\ x_n : T_n \bullet P$	Similar to $\forall$.
$\forall\, D \mid P \bullet Q$	$\Leftrightarrow \forall\, D \bullet P \Rightarrow Q$
$\exists\, D \mid P \bullet Q$	$\Leftrightarrow \exists\, D \bullet P \land Q$
$t_1 = t_2$	Equality between terms.
$t_1 \neq t_2$	$\Leftrightarrow \neg\, (t_1 = t_2)$

A.3 Sets

Let X be a set; S and T be subsets of X; t, t_k terms; P a predicate; and D declarations.

$t \in S$	Set membership: "t is a member of S".
$t \notin S$	$\Leftrightarrow \neg (t \in S)$
$S \subseteq T$	$\Leftrightarrow (\forall x : S \bullet x \in T)$ Set inclusion.
$S \subset T$	$\Leftrightarrow S \subseteq T \wedge S \neq T$ Strict set inclusion.
$\emptyset$	The empty set.
$\{t_1, t_2, \ldots, t_n\}$	The set containing the values of terms $t_1, t_2, \ldots, t_n$.
$\{x : T \mid P\}$	The set containing exactly those x of type T for which P holds.
$(t_1, t_2, \ldots, t_n)$	Ordered n-tuple of $t_1, t_2, \ldots, t_n$.
$T_1 \times T_2 \times \ldots \times T_n$	Cartesian product: the set of all n-tuples such that the kth component is of type T_k.
$\mathit{first}(t_1, t_2, \ldots, t_n)$	$\mathrel{\widehat=} t_1$ Similarly, $\mathit{second}(t_1, t_2, \ldots, t_n) \mathrel{\widehat=} t_2$, etc.
$\{x_1 : T_1;\ x_2 : T_2;\ \ldots;\ x_n : T_n \mid P\}$	The set of all n-tuples $(x_1, x_2, \ldots, x_n)$ with each x_k of type T_k such that P holds.
$\{D \mid P \bullet t\}$	The set of values of the term t for the variables declared in D ranging over all values for which P holds.
$\{D \bullet t\}$	$\mathrel{\widehat=} \{D \mid \mathit{true} \bullet t\}$
$\mathbb{P}\, S$	Powerset: the set of all subsets of S.
$\mathbb{P}_1\, S$	$\mathrel{\widehat=} \mathbb{P}\, S \setminus \{\emptyset\}$ The set of all non-empty subsets of S.

$\mathbb{F}\,S$	$\mathrel{\widehat=} \{\, T : \mathbb{P}\,S \mid T \text{ is finite} \,\}$ Set of finite subsets of S.
$\mathbb{F}_1\,S$	$\mathrel{\widehat=} \mathbb{F}\,S \setminus \{\emptyset\}$ Set of finite non-empty subsets of S.
$S \cap T$	$\mathrel{\widehat=} \{x : X \mid x \in S \land x \in T\}$ Set intersection.
$S \cup T$	$\mathrel{\widehat=} \{x : X \mid x \in S \lor x \in T\}$ Set union.
$S \setminus T$	$\mathrel{\widehat=} \{x : X \mid x \in S \land x \notin T\}$ Set difference.
$\bigcap SS$	$\mathrel{\widehat=} \{x : X \mid (\forall S : SS \bullet x \in S)\}$ Intersection of a set of sets; SS is a set containing as its members subsets of X, i.e. $SS : \mathbb{P}(\mathbb{P}\,X)$.
$\bigcup SS$	$\mathrel{\widehat=} \{x : X \mid (\exists S : SS \bullet x \in S)\}$ Union of a set of sets; $SS : \mathbb{P}(\mathbb{P}\,X)$.
$\#S$	Size (number of distinct members) of a finite set.

A.4 Numbers

$\mathbb{R}$	The set of real numbers.
$\mathbb{Z}$	The set of integers (positive, zero and negative).
$\mathbb{N}$	$\mathrel{\widehat=} \{n : \mathbb{Z} \mid n \geq 0\}$ The set of natural numbers (non-negative integers).
$\mathbb{N}_1$	$\mathrel{\widehat=} \mathbb{N} \setminus \{0\}$ The set of strictly positive natural numbers.
$m \mathinner{..} n$	$\mathrel{\widehat=} \{k : \mathbb{Z} \mid m \leq k \land k \leq n\}$ The set of integers between m and n inclusive.
$\min S$	Minimum of a set; for $S : \mathbb{P}_1\,\mathbb{Z}$, $\min S \in S \land (\forall x : S \bullet x \geq \min S)$.
$\max S$	Maximum of a set; for $S : \mathbb{P}_1\,\mathbb{Z}$, $\max S \in S \land (\forall x : S \bullet x \leq \max S)$.

A.5 Relations

A binary relation is modelled by a set of ordered pairs hence operators defined for sets can be used on relations. Let X, Y, and Z be sets; $x : X$; $y : Y$; S be a subset of X; T be a subset of Y; and R a relation between X and Y.

$X \leftrightarrow Y$	$\widehat{=} \mathbb{P}(X \times Y)$ The set of relations between X and Y.
$x \underline{R} y$	$\widehat{=} (x, y) \in R$ x is related by R to y.
$x \mapsto y$	$\widehat{=} (x, y)$
$\{x_1 \mapsto y_1, x_2 \mapsto y_2, \ldots, x_n \mapsto y_n\}$	$\widehat{=} \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ The relation relating x_1 to y_1, x_2 to y_2, ..., and x_n to y_n.
$\operatorname{dom} R$	$\widehat{=} \{x : X \mid (\exists y : Y \bullet x \underline{R} y)\}$ The domain of a relation: the set of x components that are related to some y.
$\operatorname{ran} R$	$\widehat{=} \{y : Y \mid (\exists x : X \bullet x \underline{R} y)\}$ The range of a relation: the set of y components that some x is related to.
$R_1 \fatsemi R_2$	$\widehat{=} \{x : X;\ z : Z \mid (\exists y : Y \bullet x\, R_1\, y \wedge y\, R_2\, z)\}$ Forward relational composition; $R_1 : X \leftrightarrow Y$; $R_2 : Y \leftrightarrow Z$.
$R_1 \circ R_2$	$\widehat{=} R_2 \fatsemi R_1$ Relational composition. This form is primarily used when R_1 and R_2 are functions.
$R^{\sim}$	$\widehat{=} \{y : Y;\ x : X \mid x \underline{R} y\}$ Transpose of a relation R.
$\operatorname{id} S$	$\widehat{=} \{x : S \bullet x \mapsto x\}$ Identity function on the set S.
R^k	The homogeneous relation R composed with itself k times: given $R : X \leftrightarrow X$, $R^0 = \operatorname{id} X$ and $R^{k+1} = R^k \fatsemi R$.

R^+	$\mathrel{\widehat=} \bigcup\{n : \mathbb{N}_1 \bullet R^n\}$ $= \bigcap\{Q : X \leftrightarrow X \mid R \subseteq Q \land Q \fatsemi Q \subseteq Q\}$ Transitive closure.
R^*	$\mathrel{\widehat=} \bigcup\{n : \mathbb{N} \bullet R^n\}$ $= \bigcap\{Q : X \leftrightarrow X \mid \mathrm{id}\,X \subseteq Q \land R \subseteq Q \land Q \fatsemi Q \subseteq Q\}$ Reflexive transitive closure.
$R\llparenthesis\, S \,\rrparenthesis$	$\mathrel{\widehat=} \{y : Y \mid (\exists\, x : S \bullet x \mathrel{\underline{R}} y)\}$ Image of the set S through the relation R.
$S \lhd R$	$\mathrel{\widehat=} \{x : X;\ y : Y \mid x \in S \land x \mathrel{\underline{R}} y\}$ Domain restriction: the relation R with its domain restricted to the set S.
$S \ndres R$	$\mathrel{\widehat=} (X \setminus S) \lhd R$ Domain subtraction: the relation R with the elements of S removed from its domain.
$R \rhd T$	$\mathrel{\widehat=} \{x : X;\ y : Y \mid x \mathrel{\underline{R}} y \land y \in T\}$ Range restriction to T.
$R \nrres T$	$\mathrel{\widehat=} R \rhd (Y \setminus T)$ Range subtraction of T.
$R_1 \oplus R_2$	$\mathrel{\widehat=} (\mathrm{dom}\, R_2 \ndres R_1) \cup R_2$ Overriding; $R_1, R_2 : X \leftrightarrow Y$.

A.6 Functions

A function is a relation with the property that each member of its domain is associated with a unique member of its range. As functions are relations, all the operators defined above for relations also apply to functions. Let X and Y be sets, and T be a subset of X (i.e. $T : \mathbb{P}\, X$).

$f\ t$	The function f applied to t.
$X \pfun Y$	$\mathrel{\widehat=} \{f : X \leftrightarrow Y \mid (\forall\, x : \mathrm{dom}\, f \bullet (\exists_1\, y : Y \bullet x \mathrel{\underline{f}} y))\}$ The set of partial functions from X to Y.

$X \fun Y$	$\defs \{f : X \pfun Y \mid \dom f = X\}$ The set of total functions from X to Y.
$X \pinj Y$	$\defs \{f : X \pfun Y \mid (\forall y : \ran f \bullet (\exists_1 x : X \bullet x \underline{f} y))\}$ The set of partial one-to-one functions (partial injections) from X to Y.
$X \inj Y$	$\defs \{f : X \pinj Y \mid \dom f = X\}$ The set of total one-to-one functions (total injections) from X to Y.
$X \psurj Y$	$\defs \{f : X \pfun Y \mid \ran f = Y\}$ The set of partial onto functions (partial surjections) from X to Y.
$X \surj Y$	$\defs (X \psurj Y) \cap (X \fun Y)$ The set of total onto functions (total surjections) from X to Y.
$X \bij Y$	$\defs (X \surj Y) \cap (X \inj Y)$ The set of total one-to-one onto functions (total bijections) from X to Y.
$X \ffun Y$	$\defs \{f : X \pfun Y \mid f \in \mathbb{F}(X \times Y)\}$ The set of finite partial functions from X to Y.
$X \finj Y$	$\defs \{f : X \inj Y \mid f \in \mathbb{F}(X \times Y)\}$ The set of finite partial one-to-one functions from X to Y.
$(\lambda x : X \mid P \bullet t)$	$\defs \{x : X \mid P \bullet x \mapsto t\}$ Lambda-abstraction: the function that, given an argument x of type X such that P holds, gives a result which is the value of the term t.
$(\lambda x_1 : T_1; \ldots; x_n : T_n \mid P \bullet t)$	$\defs \{x_1 : T_1; \ldots; x_n : T_n \mid P \bullet (x_1, \ldots, x_n) \mapsto t\}$
$\mathsf{disjoint}\ [I, X]$	$\defs \{S : I \pfun \mathbb{P} X \mid \forall i, j : \dom S \bullet i \neq j \Rightarrow S(i) \cap S(j) = \emptyset\}$ Pairwise disjoint; where I is a set and S an indexed family of subsets of X (i.e. $S : I \pfun \mathbb{P} X$).
$S\ \underline{\mathsf{partition}}\ T$	$\defs S \in \mathsf{disjoint} \land \bigcup \ran S = T$

A.7 Sequences

Let X be a set; A and B be sequences with elements taken from X; and $a_1, \ldots, a_n$ terms of type X.

$\mathrm{seq}\, X$	$\widehat{=} \{A : \mathbb{N}_1 \pfun X \mid (\exists\, n : \mathbb{N} \bullet \mathrm{dom}\, A = 1..n)\}$ The set of finite sequences whose elements are drawn from X.
$\mathrm{seq}_\infty\, X$	$\widehat{=} \{A : \mathbb{N}_1 \pfun X \mid A \in \mathrm{seq}\, X \lor \mathrm{dom}\, A = \mathbb{N}_1\}$ The set of finite and infinite sequences whose elements are drawn from X.
$\#A$	The length of a finite sequence A. (This is just '#' on the set representing the sequence.)
$\langle\rangle$	$\widehat{=} \{\}$ The empty sequence.
$\mathrm{seq}_1\, X$	$\widehat{=} \{s : \mathrm{seq}\, X \mid s \neq \langle\rangle\}$ The set of non-empty finite sequences.
$\langle a_1, \ldots, a_n\rangle$	$= \{1 \mapsto a_1, \ldots, n \mapsto a_n\}$
$\langle a_1, \ldots, a_n\rangle \frown \langle b_1, \ldots, b_m\rangle$	$= \langle a_1, \ldots, a_n, b_1, \ldots, b_m\rangle$ Concatenation. $\langle\rangle \frown A = A \frown \langle\rangle = A$.
$\mathit{head}\ A$	The first element of a non-empty sequence: $A \neq \langle\rangle \Rightarrow \mathit{head}\ A = A(1)$.
$\mathit{tail}\ A$	All but the head of a non-empty sequence: $\mathit{tail}\,(\langle x\rangle \frown A) = A$.
$\mathit{last}\ A$	The final element of a non-empty finite sequence: $A \neq \langle\rangle \Rightarrow \mathit{last}\ A = A(\#A)$.
$\mathit{front}\ A$	All but the last of a non-empty finite sequence: $\mathit{front}\,(A \frown \langle x\rangle) = A$.
$\mathit{rev}\,\langle a_1, a_2, \ldots, a_n\rangle$	$= \langle a_n, \ldots, a_2, a_1\rangle$ Reverse of a finite sequence; $\mathit{rev}\,\langle\rangle = \langle\rangle$.

$\frown/\, AA$	$= AA(1) \frown \ldots \frown AA(\#AA)$ Distributed concatenation; where $AA : \mathrm{seq}(\mathrm{seq}(X))$. $\frown/\langle\rangle = \langle\rangle$.
$A \subseteq B$	$\Leftrightarrow \exists\, C : \mathrm{seq}_\infty X \bullet A \frown C = B$ A is a prefix of B. (This is just '$\subseteq$' on the sets representing the sequences.)
$\mathit{squash}\ f$	Convert a finite function, $f : \mathbb{N} \twoheadrightarrow X$, into a sequence by squashing its domain. That is, $\mathit{squash}\ \{\} = \langle\rangle$, and if $f \neq \{\}$ then $\mathit{squash}\ f = \langle f(i)\rangle \frown \mathit{squash}\ (\{i\} \lhd f)$, where $i = \min(\mathrm{dom}\, f)$. For example, $\mathit{squash}\ \{2 \mapsto A, 27 \mapsto C, 4 \mapsto B\} = \langle A, B, C\rangle$.
$A \upharpoonright T$	$\widehat{=}\ \mathit{squash}\ (A \rhd T)$ Restrict the range of the sequence A to the set T.

A.8 Bags

$\mathrm{bag}\, X$	$\widehat{=}\ X \nrightarrow \mathbb{N}_1$ The set of bags whose elements are drawn from X. A bag is represented by a function that maps each element in the bag onto its frequency of occurrence in the bag.
$[\![\,]\!]$	The empty bag $\emptyset$.
$[\![x_1, x_2, \ldots, x_n]\!]$	The bag containing $x_1, x_2, \ldots, x_n$, each with the frequency that it occurs in the list.
$\mathit{items}\ s$	$\widehat{=}\ \{x : \mathrm{ran}\, s \bullet x \mapsto \#\{i : \mathrm{dom}\, s \mid s(i) = x\}\}$ The bag of items contained in the sequence s.

A.9 Axiomatic Definitions

Let D be a list of declarations and P a predicate.

The following axiomatic definition introduces the variables in D with the types as declared in D. These variables must satisfy the predicate P. The scope of the variables is the whole specification.

$$\begin{array}{|l} D \\ \hline P \end{array}$$

A.10 Generic Definitions

Let D be a list of declarations, P a predicate and $X_1, X_2, \ldots X_n$ variables.

The following generic definition is similar to an axiomatic definition, except that the variables introduced are generic over the sets $X_1, X_2, \ldots X_n$.

$$\begin{array}{|l} [X_1, X_2, \ldots X_n] \\ \hline\hline D \\ \hline P \\ \hline \end{array}$$

The declared variables must be uniquely defined by the predicate P.

Schema Notation

A.11 Schema Definition

A schema groups together a set of declarations of variables and a predicate relating the variables. If the predicate is omitted it is taken to be true, i.e. the variables are not further restricted. There are two ways of writing schemas: vertically, for example,

$$
\begin{array}{|l}
\multicolumn{1}{l}{S} \\ \hline
x : \mathbb{N} \\
y : \operatorname{seq} \mathbb{N} \\ \hline
x \leq \# y \\ \hline
\end{array}
$$

and horizontally, for the same example,

$$S \mathrel{\widehat=} [x : \mathbb{N};\ y : \operatorname{seq} \mathbb{N} \mid x \leq \# y]$$

Schemas can be used in signatures after $\forall$, λ, $\{...\}$, etc.:

$$(\forall S \bullet y \neq \langle\rangle) \Leftrightarrow (\forall x : \mathbb{N};\ y : \operatorname{seq} \mathbb{N} \mid x \leq \# y \bullet y \neq \langle\rangle)$$

$\{S\}$	Stands for the set of objects described by schema S. In declarations $w : S$ is usually written as an abbreviation for $w : \{S\}$.

A.12 Schema Operators

Let S be defined as above and $w : S$.

$w.x$	$\mathrel{\widehat=} (\lambda S \bullet x)(w)$ Projection functions: the component names of a schema may be used as projection (or selector) functions, e.g. $w.x$ is w's x component and $w.y$ is its y component; of course, the predicate '$w.x \leq \# w.y$' holds.

θS	The (unordered) tuple formed from a schema's variables, e.g. θS contains the named components x and y.
Compatibility	Two schemas are compatible if the declared sets of each variable common to the declaration parts of the two schemas are equal. In addition, any global variables referenced in predicate part of one of the schemas must not have the same name as a variable declared in the other schema; this restriction is to avoid global variables being *captured* by the declarations.
Inclusion	A schema S may be included within the declarations of a schema T, in which case the declarations of S are merged with the other declarations of T (variables declared in both S and T must have the same declared sets) and the predicates of S and T are conjoined. For example, $\begin{array}{\|l} \hline T \\ S \\ z : \mathbb{N} \\ \hline z < x \\ \hline \end{array}$ is equivalent to $\begin{array}{\|l} \hline T \\ x, z : \mathbb{N} \\ y : \operatorname{seq} \mathbb{N} \\ \hline x \leq \#y \land z < x \\ \hline \end{array}$ The included schema (S) may not refer to global variables that have the same name as one of the declared variables of the including schema (T).
Decoration	Decoration with subscript, superscript, prime, etc: systematic renaming of the variables declared in the schema. For example, S' is $[x' : \mathbb{N};\ y' : \operatorname{seq} \mathbb{N} \mid x' \leq \#y']$.
$\neg\, S$	The schema S with its predicate part negated. For example, $\neg\, S$ is $[x : \mathbb{N};\ y : \operatorname{seq} \mathbb{N} \mid \neg\, (x \leq \#y)]$.
$S \land T$	The schema formed from schemas S and T by merging their declarations and conjoining (and-ing) their predicates. The

two schemas must be compatible (see above).
Given $T \mathrel{\widehat=} [x : \mathbb{N};\ z : \mathbb{P}\,\mathbb{N} \mid x \in z]$, $S \land T$ is

$$\begin{array}{|l}
\hline
S \land T \\
x : \mathbb{N} \\
y : \operatorname{seq} \mathbb{N} \\
z : \mathbb{P}\,\mathbb{N} \\
\hline
x \leq \# y \land x \in z \\
\hline
\end{array}$$

$S \lor T$

The schema formed from schemas S and T by merging their declarations and disjoining (or-ing) their predicates. The two schemas must be compatible (see above). For example, $S \lor T$ is

$$\begin{array}{|l}
\hline
S \lor T \\
x : \mathbb{N} \\
y : \operatorname{seq} \mathbb{N} \\
z : \mathbb{P}\,\mathbb{N} \\
\hline
x \leq \# y \lor x \in z \\
\hline
\end{array}$$

$S \Rightarrow T$

The schema formed from schemas S and T by merging their declarations and taking ‘ pred $S \Rightarrow$ pred T’ as the predicate. The two schemas must be compatible (see above). For example, $S \Rightarrow T$ is

$$\begin{array}{|l}
\hline
S \Rightarrow T \\
x : \mathbb{N} \\
y : \operatorname{seq} \mathbb{N} \\
z : \mathbb{P}\,\mathbb{N} \\
\hline
x \leq \# y \Rightarrow x \in z \\
\hline
\end{array}$$

$S \Leftrightarrow T$

The schema formed from schemas S and T by merging their declarations and taking ‘ pred $S \Leftrightarrow$ pred T’ as the predicate. The two schemas must be compatible (see above). For example, $S \Leftrightarrow T$ is

$$\begin{array}{|l}
\hline
S \Leftrightarrow T \\
x : \mathbb{N} \\
y : \operatorname{seq} \mathbb{N} \\
z : \mathbb{P}\,\mathbb{N} \\
\hline
x \leq \# y \Leftrightarrow x \in z \\
\hline
\end{array}$$

$S \setminus (v_1, v_2, \ldots, v_n)$

Hiding: the schema S with variables $v_1, v_2, \ldots, v_n$ hidden – the variables listed are removed from the declarations and are existentially quantified in the predicate. The parantheses may be omitted when only one variable is hidden.

$S \upharpoonright (v_1, v_2, \ldots, v_n)$

Projection: The schema S with any variables that do not occur in the list $v_1, v_2, \ldots, v_n$ hidden – the variables are removed from the declarations and are existentially qualified in the predicate. For example, $(S \land T) \upharpoonright (x, y)$ is

$$\begin{array}{|l}
\hline
\textbf{}(S \land T) \upharpoonright (x, y) \\
x : \mathbb{N} \\
y : \operatorname{seq} \mathbb{N} \\
\hline
(\exists z : \mathbb{P}\,\mathbb{N} \bullet \\
\qquad x \leq \#y \land x \in z) \\
\hline
\end{array}$$

The list of variables may be replaced by a schema; the variables declared in the schema are used for projection.

$\exists D \bullet S$

Existential quantification of a schema.
The variables declared in the schema S that also appear in the declarations D are removed from the declarations of S. The predicate of S is existentially quantified over D. For example, $\exists x : \mathbb{N} \bullet S$ is the following schema.

$$\begin{array}{|l}
\hline
\exists x : \mathbb{N} \bullet S \\
y : \operatorname{seq} \mathbb{N} \\
\hline
\exists x : \mathbb{N} \bullet \\
\qquad x \leq \#y \\
\hline
\end{array}$$

The declarations may include schemas. For example,

$$\begin{array}{|l}
\hline
\exists S \bullet T \\
z : \mathbb{N} \\
\hline
\exists S \bullet \\
\qquad x \leq \#y \land z < x \\
\hline
\end{array}$$

$\forall D \bullet S$

Universal quantification of a schema.
The variables declared in the schema S that also appear in the declarations D are removed from the declarations of S. The predicate of S is universally quantified over D. For example, $\forall x : \mathbb{N} \bullet S$ is the following schema.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\forall x : \mathbb{N} \bullet S} \\ \hline
y : \operatorname{seq} \mathbb{N} \\ \hline
\forall x : \mathbb{N} \bullet \\
\qquad x \leq \# y \\ \hline
\end{array}
$$

The declarations may include schemas. For example,

$$
\begin{array}{|l}
\multicolumn{1}{l}{\forall S \bullet T} \\ \hline
z : \mathbb{N} \\ \hline
\forall S \bullet \\
\qquad x \leq \# y \land z < x \\ \hline
\end{array}
$$

A.13 Operation Schemas

The following conventions are used for variable names in those schemas which represent operations, that is, which are written as descriptions of operations on some state,

undashed state before the operation,

dashed state after the operation,

ending in "?" inputs to (arguments for) the operation, and

ending in "!" outputs from (results of) the operation.

The basename of a name is the name with all decorations removed.

ΔS

$\widehat{=} S \land S'$
Change of state schema: this is a default definition for ΔS. In some specifications it is useful to have additional constraints

on the change of state schema. In these cases ΔS can be explicitly defined.

ΞS

$\mathrel{\widehat=} [\Delta S \mid \theta S' = \theta S]$
No change of state schema.

A.14 Operation Schema Operators

pre S

Precondition: the after-state components (dashed) and the outputs (ending in "!") are hidden, e.g. given,

$$
\begin{array}{|l}
\multicolumn{1}{l}{S} \\ \hline
x?, s, s', y! : \mathbb{N} \\ \hline
s' = s - x? \land y! = s' \\ \hline
\end{array}
$$

pre S is,

$$
\begin{array}{|l}
\multicolumn{1}{l}{\mathrm{pre}\ S} \\ \hline
x?, s : \mathbb{N} \\ \hline
\exists s', y! : \mathbb{N} \bullet \\
\qquad s' = s - x? \land y! = s' \\ \hline
\end{array}
$$

$S \fatsemi T$

Schema composition: if we consider an intermediate state that is both the final state of the operation S and the initial state of the operation T then the composition of S and T is the operation which relates the initial state of S to the final state of T through the intermediate state. To form the composition of S and T we take the pairs of after-state components of S and before-state components of T that have the same basename, rename each pair to a new variable, take the conjunction of the resulting schemas, and hide the new variables. For example, $S \fatsemi T$ is,

$$
\begin{array}{|l}
\multicolumn{1}{l}{S \fatsemi T} \\ \hline
x?, s, s', y! : \mathbb{N} \\ \hline
(\exists ss : \mathbb{N} \bullet \\
\qquad ss = s - x? \land y! = ss \\
\qquad \land\ ss \leq x? \land s' = ss + x?) \\ \hline
\end{array}
$$

Appendix B

Z Semantics for DAML+OIL

In this appendix, we present the complete Z semantics for the ontology language DAML+OIL. As DAML+OIL emphasizes on the description of abstract concepts, discussion of concrete (data type-related) properties are not considered in the Z semantics.

B.1 Basic Concepts

Everything in DAML+OIL (and RDF) is regarded a Web resource, hence, we make *Resource* a given type, which is not interpreted.

In DAML+OIL, resources are grouped under various *classes*, which are related to each other via *properties*. Hence, we model *Class* and *Property* as subsets of *Resource*. Moreover, these two sets are disjoint.

$$[\mathit{Resource}]$$

$$\begin{array}{|l}
\mathit{Class} : \mathbb{P}\,\mathit{Resource} \\
\mathit{Property} : \mathbb{P}\,\mathit{Resource} \\
\hline
\mathit{Class} \cap \mathit{Property} = \emptyset
\end{array}$$

To link the members of a class to itself and the pairs of resources a property maps back to this property, we define two important auxiliary functions: *instances* and *sub_val*.

$$instances : Class \rightarrow \mathbb{P}\, Resource \qquad sub\_val : Property \rightarrow (Resource \leftrightarrow Resource)$$

In DAML+OIL, there are two pre-defined special classes: *Thing* and *Nothing*, which is the super class/sub class of all classes, respectively. In other words, the instances of *Thing* is the whole set *Resource* whereas *Nothing* does not hold any instance.

$$\begin{array}{|l}
Thing, Nothing : Class \\
\hline
instances(Thing) = Resource \\
instances(Nothing) = \emptyset
\end{array}$$

B.2 Class Elements

DAML+OIL defines a number of properties to relate classes. In this section, we present the definitions of their Z counterparts. Note that these properties are translated as Z relations since they are *meta-level* properties.

The Z relations *subClassOf*, *disjointWith*, *sameClassAs* are all binary relations that apply to two classes.

$$\begin{array}{|l}
subClassOf, disjointWith, sameClassAs : Class \leftrightarrow Class \\
\hline
\forall c_1, c_2 : Class \bullet \\
\quad c_1 \ \underline{subClassOf}\ c_2 \Leftrightarrow instances(c_1) \subseteq instances(c_2) \\
\forall c_1, c_2 : Class \bullet \\
\quad c_1 \ \underline{disjointWith}\ c_2 \Leftrightarrow instances(c_1) \cap instances(c_2) = \emptyset \\
\forall c_1, c_2 : Class \bullet \\
\quad c_1 \ \underline{sameClassAs}\ c_2 \Leftrightarrow instances(c_1) = instances(c_2)
\end{array}$$

Besides these binary relations, DAML+OIL also defines a number of *boolean combinations* of classes. These include *intersectionOf*, *unionOf* and *complementOf*. The relation *disjointUnionOf* combines *disjiontWith* and *unionOf*.

$$\begin{array}{l}
\mathit{intersectionOf}, \mathit{unionOf} : \operatorname{seq} \mathit{Class} \rightarrow \mathit{Class} \\
\hline
\forall\, \mathit{cl} : \operatorname{seq} \mathit{Class};\ c : \mathit{Class} \bullet \\
\qquad \mathit{intersectionOf}(\mathit{cl}) = c \Leftrightarrow \mathit{instances}(c) = \bigcap\{x : \operatorname{ran} \mathit{cl} \bullet \mathit{instances}(x)\} \\
\forall\, \mathit{cl} : \operatorname{seq} \mathit{Class};\ c : \mathit{Class} \bullet \\
\qquad \mathit{unionOf}(\mathit{cl}) = c \Leftrightarrow \mathit{instances}(c) = \bigcup\{x : \operatorname{ran} \mathit{cl} \bullet \mathit{instances}(x)\}
\end{array}$$

$$\begin{array}{l}
\mathit{complementOf} : \mathit{Class} \leftrightarrow \mathit{Class} \\
\hline
\forall\, c_1, c_2 : \mathit{Class} \bullet \\
\qquad c_1 \ \underline{\mathit{complementOf}}\ c_2 \Leftrightarrow \mathit{Resource} \setminus \mathit{instances}(c_1) = \mathit{instances}(c_2)
\end{array}$$

$$\begin{array}{l}
\mathit{disjointUnionOf} : \operatorname{seq} \mathit{Class} \rightarrow \mathit{Class} \\
\hline
\forall\, \mathit{cl} : \operatorname{seq} \mathit{Class};\ c : \mathit{Class} \bullet \mathit{disjointUnionOf}(\mathit{cl}) = c \Leftrightarrow \\
\qquad \mathit{unionOf}(\mathit{cl}) = c \wedge \\
\qquad (\forall\, x, y : \mathit{cl} \mid x.1 \neq y.1 \bullet x.2 \ \underline{\mathit{disjointWith}}\ y.2)
\end{array}$$

B.3 Class Enumeration

The class enumeration relation *oneOf* enumerates all the instances of a class.

$$\begin{array}{l}
\mathit{oneOf} : \mathbb{P}\, \mathit{Resource} \rightarrow \mathit{Class} \\
\hline
\forall\, x : \mathbb{P}\, \mathit{Resource};\ c : \mathit{Class} \bullet \mathit{oneOf}(x) = c \Leftrightarrow x = \mathit{instances}(c)
\end{array}$$

B.4 Property Restriction

Besides a class denoted by its name and class enumeration introduced above, class expressions include also *property restrictions*.

A *toClass* element defines the class c_2 of all objects for which the values of property p all belong to the class expression c_1.

$$
\begin{array}{|l}
\mathit{toClass} : (\mathit{Class} \times \mathit{Property}) \to \mathit{Class} \\
\hline
\forall c_1, c_2 : \mathit{Class};\ p : \mathit{Property} \bullet \mathit{toClass}(c_1, p) = c_2 \Leftrightarrow \\
\qquad \mathit{instances}(c_2) = \\
\qquad\qquad \{a : \mathit{Resource} \mid \mathit{sub\_val}(p)(\!|\ \{a\}\ |\!) \subseteq \mathit{instances}(c_1)\}
\end{array}
$$

A *hasValue* element defines the class c of all objects for which the property p has at least one value equal to the named object r or data type value (and perhaps other values as well).

$$
\begin{array}{|l}
\mathit{hasValue} : (\mathit{Resource} \times \mathit{Property}) \to \mathit{Class} \\
\hline
\forall r : \mathit{Resource};\ p : \mathit{Property};\ c : \mathit{Class} \bullet \mathit{hasValue}(r, p) = c \Leftrightarrow \\
\qquad \mathit{instances}(c) = \\
\qquad\qquad \{a : \mathit{Resource} \mid r \in \mathit{sub\_val}(p)(\!|\ \{a\}\ |\!)\}
\end{array}
$$

A *hasClass* element defines the class c_2 of all objects for which at least one value of the property p is a member of the class expression or data type c_1.

$$
\begin{array}{|l}
\mathit{hasClass} : (\mathit{Class} \times \mathit{Property}) \to \mathit{Class} \\
\hline
\forall c_1, c_2 : \mathit{Class};\ p : \mathit{Property} \bullet \mathit{hasValue}(c_1, p) = c_2 \Leftrightarrow \\
\qquad \mathit{instances}(c_2) = \\
\qquad\qquad \{a : \mathit{Resource} \mid \mathit{sub\_val}(p)(\!|\ \{a\}\ |\!) \cap \mathit{instances}(c_1) \neq \emptyset\}
\end{array}
$$

DAML+OIL also defines a number of (qualified) cardinality-related constraints. For example, the *cardinality* relation defines the class c of all objects that have exactly n distinct values for the property p, i.e., a is an instance of c if and only if there are exactly n distinct values mapped to a by p. Other relations are similarly defined.

$$\begin{array}{|l}
\mathit{cardinality}, \mathit{minCardinality}, \mathit{maxCardinality} : (\mathbb{N} \times \mathit{Property}) \rightarrow \mathit{Class} \\
\hline
\forall\, n : \mathbb{N};\ p : \mathit{Property};\ c : \mathit{Class} \bullet \mathit{cardinality}(n, p) = c \Leftrightarrow \\
\qquad \mathit{instances}(c) = \{a : \mathit{Resource} \mid \#(\mathit{sub\_val}(p)(\!|\ \{a\}\ |\!)) = n\} \\
\forall\, n : \mathbb{N};\ p : \mathit{Property};\ c : \mathit{Class} \bullet \mathit{minCardinality}(n, p) = c \Leftrightarrow \\
\qquad \mathit{instances}(c) = \{a : \mathit{Resource} \mid \#(\mathit{sub\_val}(p)(\!|\ \{a\}\ |\!)) \geq n\} \\
\forall\, n : \mathbb{N};\ p : \mathit{Property};\ c : \mathit{Class} \bullet \mathit{maxCardinality}(n, p) = c \Leftrightarrow \\
\qquad \mathit{instances}(c) = \{a : \mathit{Resource} \mid \#(\mathit{sub\_val}(p)(\!|\ \{a\}\ |\!)) \leq n\}
\end{array}$$

The qualified cardinality constraints are similarly defined, except that the quantified elements must be from a specific class expression.

$$\begin{array}{|l}
\mathit{cardinalityQ}, \mathit{minCardinalityQ}, \mathit{maxCardinalityQ} : (\mathbb{N} \times \mathit{Class} \times \mathit{Property}) \rightarrow \mathit{Class} \\
\hline
\forall\, n : \mathbb{N};\ c_1, c_2 : \mathit{Class};\ p : \mathit{Property} \bullet \mathit{cardinalityQ}(n, c_1, p) = c_2 \Leftrightarrow \\
\qquad \mathit{instances}(c_2) = \{a : \mathit{Resource} \mid \#(\mathit{sub\_val}(p)(\!|\ \{a\}\ |\!) \cap \mathit{instances}(c_1)) = n\} \\
\forall\, n : \mathbb{N};\ c_1, c_2 : \mathit{Class};\ p : \mathit{Property} \bullet \mathit{minCardinalityQ}(n, c_1, p) = c_2 \Leftrightarrow \\
\qquad \mathit{instances}(c_2) = \{a : \mathit{Resource} \mid \#(\mathit{sub\_val}(p)(\!|\ \{a\}\ |\!) \cap \mathit{instances}(c_1)) \geq n\} \\
\forall\, n : \mathbb{N};\ c_1, c_2 : \mathit{Class};\ p : \mathit{Property} \bullet \mathit{maxCardinalityQ}(n, c_1, p) = c_2 \Leftrightarrow \\
\qquad \mathit{instances}(c_2) = \{a : \mathit{Resource} \mid \#(\mathit{sub\_val}(p)(\!|\ \{a\}\ |\!) \cap \mathit{instances}(c_1)) \leq n\}
\end{array}$$

B.5 Property Elements

In this section, we present the Z semantics of DAML+OIL language constructs for describing properties. These constructs, such as *domain*, *range*, *subPropertyOf*, etc., are translated into Z functions or relations.

The following three relations model the relationship between two properties. They are similar to those defined in Section B.2.

$$\begin{array}{|l}
subPropertyOf, samePropertyOf, inverseOf : Property \leftrightarrow Property \\
\hline
\forall p_1, p_2 : Property \bullet \\
\quad p_1 \ \underline{subPropertyOf} \ p_2 \Leftrightarrow sub\_val(p_1) \subseteq sub\_val(p_2) \\
\forall p_1, p_2 : Property \bullet \\
\quad p_1 \ \underline{samePropertyOf} \ p_2 \Leftrightarrow sub\_val(p_1) = sub\_val(p_2) \\
\forall p_1, p_2 : Property \bullet \\
\quad p_1 \ \underline{inverseOf} \ p_2 \Leftrightarrow sub\_val(p_1) = (sub\_val(p_2))^{\sim}
\end{array}$$

The relations *domain* and *range* maps a property to its domain and range, respectively.

$$\begin{array}{|l}
domain, range : Property \rightarrow Class \\
\hline
\forall p : Property; \ c : Class \bullet \\
\quad domain(p) = c \Leftrightarrow \mathrm{dom}(sub\_val(p)) \subseteq instances(c) \\
\forall p : Property; \ c : Class \bullet \\
\quad range(p) = c \Leftrightarrow \mathrm{ran}(sub\_val(p)) \subseteq instances(c)
\end{array}$$

In DAML+OIL, a property can be transitive, unique (functional), or unambiguous (inverse functional). Three properties are defined to model these characteristics.

$$\begin{array}{|l}
TransitiveProperty, UniqueProperty, UnambiguousProperty : \mathbb{P}\, Property \\
\hline
\forall p : Property \bullet \\
p \in TransitiveProperty \Leftrightarrow (\forall x, y, z : Resource \bullet \\
\quad (x, y) \in sub\_val(p) \wedge (y, z) \in sub\_val(p) \Rightarrow (x, z) \in sub\_val(p)) \\
\forall p : Property \bullet \\
p \in UniqueProperty \Leftrightarrow (\forall x, y, z : Resource \bullet \\
\quad (x, y) \in sub\_val(p) \wedge (x, z) \in sub\_val(p) \Rightarrow y = z) \\
\forall p : Property \bullet \\
p \in UnambiguousProperty \Leftrightarrow (\forall x, y, z : Resource \bullet \\
\quad (x, z) \in sub\_val(p) \wedge (y, z) \in sub\_val(p) \Rightarrow x = y)
\end{array}$$

B.6 Instances

DAML+OIL defines two properties to relate pairs of instances: *sameIndividualAs* and *differentIndividualFrom*. They are modeled as relations in Z.

$$\begin{array}{|l}
sameIndividualAs : Resource \leftrightarrow Resource \\
differentIndividualFrom : Resource \leftrightarrow Resource
\end{array}$$

Appendix C

Z Semantics for OWL DL

In this chapter, we present the complete Z semantics for the ontology language OWL DL. The presentation in this chapter will be divided into subsections roughly according to [66].

C.1 Basic Concepts

As in the Z semantics for DAML+OIL, we model *Resource* as a given type.

$[Resource]$

In OWL, the instances of classes are grouped under one concept called *Individual*, which is modeled as a subset of *Resource*.

$| \; Individual : \mathbb{P}\, Resource$

As in DAML+OIL, *Class* and *Property* are similarly defined. Moreover, *Class*, *Property* and *Individual* are mutually disjoint.

$$\begin{array}{|l}
Class : \mathbb{P}\, Resource \\
Property : \mathbb{P}\, Resource \\
\hline
Individual \cap Class = \emptyset \\
Property \cap Class = \emptyset \\
Property \cap Individual = \emptyset
\end{array}$$

Every class holds a number of individuals as its members. The function *instances* maps a class to the set of *Individual*s it holds.

$$\begin{array}{|l}
instances : Class \rightarrow \mathbb{P}\, Individual
\end{array}$$

As in DAML+OIL, OWL also defines the two special classes, *Thing* and *Nothing*.

$$\begin{array}{|l}
Thing : Class \\
Nothing : Class \\
\hline
instances(Thing) = Individual \\
instances(Nothing) = \emptyset
\end{array}$$

In OWL DL, support for data types are more elaborate than in DAML+OIL. Hence, we also tailor the Z semantics towards data types that might appear in the OWL ontologies.

First of all, properties are further divided into 2 broad categories, those relate an individual to another individual and those relate an individual to a value of a particular data type. These two types of properties are called *ObjectProperty* and *DatatypeProperty*, which are disjoint with each other.

$$\begin{array}{|l}
ObjectProperty : \mathbb{P}\, Property \\
DatatypeProperty : \mathbb{P}\, Property \\
\hline
ObjectProperty \cap DatatypeProperty = \emptyset
\end{array}$$

Before presenting the definitions of these properties, define how these properties are mapped to the pairs of resources that they relate. Two functions, *sub_val* and *sub_valD*, are defined in Z.

The function *sub_val* is almost identical to that defined in Appendix B, except that the domain is updated to *ObjectProeprty* and *Resource* is replaced by *Individual*.

$$sub\_val : ObjectProperty \rightarrow (Individual \leftrightarrow Individual)$$

The function *sub_valD* caters for the case where an individual is related to a data item by a property. It is defined as a generic definition where the data type is represented by the generic type X and the domain is changed to *DatatypeProperty*.

$$[X]$$
$$sub\_valD : DatatypeProperty \rightarrow (Individual \leftrightarrow X)$$

C.2 Classes

C.2.1 Class Descrpitions

In this section, we will present the class descriptions, the building blocks for constructing OWL classes.

The simplest form of class description is, according to [66], by referring to the name of the class. In Z, a concept must be declared before it is used. Hence, a class is defined by using an axiomatic definition. When it is referred subsequently, its name will be used, such as the class *Individual* when defining *Class* and *Property* in the previous section.

Enumeration

As in DAML+OIL, OWL defines a class property *oneOf* that completely defines a class by enumerating its instances.

$$
\begin{array}{|l}
\mathit{oneOf} : \mathbb{P}\,\mathit{Individual} \to \mathit{Class} \\
\hline
\forall\, x : \mathbb{P}\,\mathit{Individual};\ y : \mathit{Class} \bullet \mathit{oneOf}(x) = y \Leftrightarrow x = \mathit{instances}(y)
\end{array}
$$

Property Restrictions

In OWL, a property restriction usually describes an anonymous class by constraining its membership through the use of a property. Two kinds of property restrictions are defined: value constraints and cardinality constraints.

In OWL, the value constraints include three properties, namely *allValuesFrom*, *someValuesFrom* and *hasValue*, which are similar to *toClass*, *hasClass* and *hasValue* defined in Appendix B.4.

Since these properties cater for both abstract and concrete values, we transform them into different Z definitions, as detailed below.

$$
\begin{array}{|l}
\mathit{allValuesFrom} : (\mathit{Class} \times \mathit{ObjectProperty}) \to \mathit{Class} \\
\hline
\forall\, c_1 : \mathit{Class};\ p : \mathit{ObjectProperty};\ c_2 : \mathit{Class} \bullet \mathit{allValuesFrom}(c_1, p) = c_2 \Leftrightarrow \\
\qquad \mathit{instances}(c_2) = \\
\qquad\qquad \{a : \mathit{Individual} \mid \mathit{sub\_val}(p)(\!|\ \{a\}\ |\!) \subseteq \mathit{instances}(c_1)\}
\end{array}
$$

The above axiomatic definition of *allValuesFrom* handles the case where an OWL class is constrained by another class and an object property. The following generic definition *allValuesFromD* handles the case where an OWL class is constrained by a (generic) data type and a data type property.

$$
\begin{array}{|l}
[X] \\
\hline
\mathit{allValuesFromD} : (\mathbb{P}\,X \times \mathit{DatatypeProperty}) \to \mathit{Class} \\
\hline
\forall\, d : \mathbb{P}\,X;\ c : \mathit{Class};\ p : \mathit{DatatypeProperty} \bullet \mathit{allValuesFromD}(d, p) = c \Leftrightarrow \\
\qquad \mathit{instances}(c) = \\
\qquad\qquad \{a : \mathit{Individual} \mid (\mathit{sub\_valD}(p)(\!|\ \{a\}\ |\!)) \subseteq d\} \\
\hline
\end{array}
$$

The treatment of *someValuesFrom* and *hasValue* are similar to that of *allValuesFrom*.

$$\begin{array}{|l}
\mathit{someValuesFrom} : (\mathit{Class} \times \mathit{ObjectProperty}) \rightarrow \mathit{Class} \\
\hline
\forall\, c_1, c_2 : \mathit{Class};\ p : \mathit{ObjectProperty} \bullet \mathit{someValuesFrom}(c_1, p) = c_2 \Leftrightarrow \\
\qquad \mathit{instances}(c_2) = \\
\qquad\qquad \{a : \mathit{Individual} \mid \mathit{sub\_val}(p)(\!|\ \{a\}\ |\!) \cap \mathit{instances}(c_1) \neq \emptyset\}
\end{array}$$

$$\begin{array}{|l}
[X] \\
\mathit{someValuesFromD} : (\mathbb{P}\, X \times \mathit{DatatypeProperty}) \rightarrow \mathit{Class} \\
\hline
\forall\, t : \mathbb{P}\, X;\ p : \mathit{DatatypeProperty};\ c : \mathit{Class} \bullet \mathit{someValuesFromD}(t, p) = c \Leftrightarrow \\
\qquad \mathit{instances}(c) = \\
\qquad\qquad \{a : \mathit{Individual} \mid \mathit{sub\_valD}(p)(\!|\ \{a\}\ |\!) \cap t \neq \emptyset\} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\mathit{hasValue} : (\mathit{Individual} \times \mathit{ObjectProperty}) \rightarrow \mathit{Class} \\
\hline
\forall\, r : \mathit{Individual};\ p : \mathit{ObjectProperty};\ c : \mathit{Class} \bullet \mathit{hasValue}(r, p) = c \Leftrightarrow \\
\qquad \mathit{instances}(c) = \\
\qquad\qquad \{a : \mathit{Individual} \mid r \in \mathit{sub\_val}(p)(\!|\ \{a\}\ |\!)\}
\end{array}$$

$$\begin{array}{|l}
[X] \\
\mathit{hasValueD} : (X \times \mathit{DatatypeProperty}) \rightarrow \mathit{Class} \\
\hline
\forall\, r : X;\ p : \mathit{DatatypeProperty};\ c : \mathit{Class} \bullet \mathit{hasValueD}(r, p) = c \Leftrightarrow \\
\qquad \mathit{instances}(c) = \\
\qquad\qquad \{a : \mathit{Individual} \mid r \in \mathit{sub\_valD}(p)(\!|\ \{a\}\ |\!)\} \\
\hline
\end{array}$$

The cardinality property constraints are updated based on those defined in DAML+OIL. In OWL, qualified cardinality constraints are removed as they can be expressed by using unqualified cardinality constraints and value constraints in conjunction.

To cater for data types, each of the cardinality constraints are also transformed into two versions.

$$\begin{array}{|l}
\mathit{maxCardinality} : (\mathbb{N} \times \mathit{ObjectProperty}) \rightarrow \mathit{Class} \\
\hline
\forall\, c : \mathit{Class};\ p : \mathit{ObjectProperty};\ n : \mathbb{N} \bullet \mathit{maxCardinality}(n, p) = c \Leftrightarrow \\
\qquad \mathit{instances}(c) = \{x : \mathit{Individual} \mid \#(\mathit{sub\_val}(p)(\!|\ \{x\}\ |\!)) \leq n\}
\end{array}$$

$$[X]$$
$$maxCardinalityD : (\mathbb{N} \times DatatypeProperty) \rightarrow Class$$
$$\forall c : Class;\ p : DatatypeProperty;\ n : \mathbb{N} \bullet maxCardinalityD(n, p) = c \Leftrightarrow$$
$$\quad instances(c) = \{x : Individual \mid \#[X](sub\_valD(p)(\!|\ \{x\}\ |\!)) \leq n\}$$

$$minCardinality : (\mathbb{N} \times ObjectProperty) \rightarrow Class$$
$$\forall c : Class;\ p : ObjectProperty;\ n : \mathbb{N} \bullet minCardinality(n, p) = c \Leftrightarrow$$
$$\quad instances(c) = \{x : Individual \mid \#(sub\_val(p)(\!|\ \{x\}\ |\!)) \geq n\}$$

$$[X]$$
$$minCardinalityD : (\mathbb{N} \times DatatypeProperty) \rightarrow Class$$
$$\forall c : Class;\ p : DatatypeProperty;\ n : \mathbb{N} \bullet minCardinalityD(n, p) = c \Leftrightarrow$$
$$\quad instances(c) = \{x : Individual \mid \#[X](sub\_valD(p)(\!|\ \{x\}\ |\!)) \geq n\}$$

$$cardinality : (\mathbb{N} \times ObjectProperty) \rightarrow Class$$
$$\forall c : Class;\ p : ObjectProperty;\ n : \mathbb{N} \bullet cardinality(n, p) = c \Leftrightarrow$$
$$\quad instances(c) = \{x : Individual \mid \#(sub\_val(p)(\!|\ \{x\}\ |\!)) = n\}$$

$$[X]$$
$$cardinalityD : (\mathbb{N} \times DatatypeProperty) \rightarrow Class$$
$$\forall c : Class;\ p : DatatypeProperty;\ n : \mathbb{N} \bullet cardinalityD(n, p) = c \Leftrightarrow$$
$$\quad instances(c) = \{x : Individual \mid \#[X](sub\_valD(p)(\!|\ \{x\}\ |\!)) = n\}$$

Boolean Combinations

A class description can also be one of the three boolean combinations, namely class intersection, union and complement. The property *disjointUnionOf* defined in DAML+OIL is removed from OWL as its effect can be achieved by using *disjointWith* and *unionOf* in conjunction.

$$\begin{array}{|l}
\mathit{intersectionOf} : \operatorname{seq} \mathit{Class} \rightarrow \mathit{Class} \\
\hline
\forall\, \mathit{cl} : \operatorname{seq} \mathit{Class};\ c : \mathit{Class} \bullet \mathit{intersectionOf}(\mathit{cl}) = c \Leftrightarrow \\
\qquad \mathit{instances}(c) = \bigcap\{x : \operatorname{ran} \mathit{cl} \bullet \mathit{instances}(x)\}
\end{array}$$

$$\begin{array}{|l}
\mathit{unionOf} : \operatorname{seq} \mathit{Class} \rightarrow \mathit{Class} \\
\hline
\forall\, \mathit{cl} : \operatorname{seq} \mathit{Class};\ c : \mathit{Class} \bullet \mathit{unionOf}(\mathit{cl}) = c \Leftrightarrow \\
\qquad \mathit{instances}(c) = \bigcup\{x : \operatorname{ran} \mathit{cl} \bullet \mathit{instances}(x)\}
\end{array}$$

$$\begin{array}{|l}
\mathit{complementOf} : \mathit{Class} \leftrightarrow \mathit{Class} \\
\hline
\forall\, c1, c2 : \mathit{Class} \bullet \\
\qquad c1 \ \underline{\mathit{complementOf}}\ c2 \Leftrightarrow \mathit{Individual} \setminus \mathit{instances}(c1) = \mathit{instances}(c2)
\end{array}$$

C.2.2 Class Axioms

This section contains three properties that state the inter-class relationship, namely *subClassOf*, *equivalentClass* and *disjointWith*.

The *subClassOf* is identical to that in DAML+OIL, which states that class c_1 is a sub class of c_2 if its instances is a subset of c_2.

$$\begin{array}{|l}
\mathit{subClassOf} : \mathit{Class} \leftrightarrow \mathit{Class} \\
\hline
\forall\, c1, c2 : \mathit{Class} \bullet c1 \ \underline{\mathit{subClassOf}}\ c2 \Leftrightarrow \mathit{instances}(c1) \subseteq \mathit{instances}(c2)
\end{array}$$

As the name suggests, *eqivalentClass* states the conditions under which two classes are equivalent.

$$\begin{array}{|l}
\mathit{equivalentClass} : \mathit{Class} \leftrightarrow \mathit{Class} \\
\hline
\forall\, c1, c2 : \mathit{Class} \bullet c1 \ \underline{\mathit{equivalentClass}}\ c2 \Leftrightarrow \mathit{instances}(c1) = \mathit{instances}(c2)
\end{array}$$

Two classes are disjoint with each other if and only if the intersection of their instances is an empty set.

$$\begin{array}{|l}
\mathit{disjointWith} : \mathit{Class} \leftrightarrow \mathit{Class} \\
\hline
\forall c1, c2 : \mathit{Class} \bullet c1 \ \underline{\mathit{disjointWith}}\ c2 \Leftrightarrow \mathit{instances}(c1) \cap \mathit{instances}(c2) = \emptyset
\end{array}$$

C.3 Properties

C.3.1 RDF Schema Property Constructs

As stated in Section 2.1, RDF Schema can be regarded as the first ontology language. It defines a number of language constructs for describing properties. In this section, we present the transformation of these constructs, namely *subPropertyOf*, *domain* and *range*. In OWL DL, all these three properties can be applied to both object properties and datatype properties.

$$\begin{array}{|l}
[X] \\
\hline
\mathit{subPropertyOf} : \mathit{Property} \leftrightarrow \mathit{Property} \\
\hline
\forall p_1, p_2 : \mathit{Property} \bullet p_1 \ \underline{\mathit{subPropertyOf}}\ p_2 \Leftrightarrow \\
\quad (p_1 \in \mathit{ObjectProperty} \land p_2 \in \mathit{ObjectProperty}) \Rightarrow \mathit{sub\_val}(p_1) \subseteq \mathit{sub\_val}(p_2) \land \\
\quad (p_1 \in \mathit{DatatypeProperty} \land p_2 \in \mathit{DatatypeProperty}) \Rightarrow \\
\qquad \mathit{sub\_valD}[X](p_1) \subseteq \mathit{sub\_valD}[X](p_2) \\
\hline
\end{array}$$

The following two properties return the domain and range of a property respectively.

$$\begin{array}{|l}
[X] \\
\hline
\mathit{domain} : \mathit{Property} \rightarrow \mathit{Class} \\
\hline
\forall p : \mathit{Property};\ c : \mathit{Class} \bullet \mathit{domain}(p) = c \Leftrightarrow \\
\quad p \in \mathit{ObjectProperty} \Rightarrow \mathrm{dom}(\mathit{sub\_val}(p)) \subseteq \mathit{instances}(c) \land \\
\quad p \in \mathit{DatatypeProperty} \Rightarrow \mathrm{dom}(\mathit{sub\_valD}[X](p)) \subseteq \mathit{instances}(c) \\
\hline
\end{array}$$

The property *range* defined in RDF Schema returns the range of the property. Since

OWL DL allows this property to be applied to both object and datatype properties, as before, we transform it to two versions in Z, one for each kind of properties.

$$\begin{array}{l}
\mathit{range} : \mathit{ObjectProperty} \rightarrow \mathit{Class} \\
\hline
\forall\, p : \mathit{ObjectProperty};\; c : \mathit{Class} \bullet \mathit{range}(p) = c \Leftrightarrow \\
\quad \operatorname{ran}(\mathit{sub\_val}(p)) \subseteq \mathit{instances}(c)
\end{array}$$

$$\begin{array}{l}
[X] \\
\hline
\mathit{rangeD} : \mathit{DatatypeProperty} \rightarrow \mathbb{P}\, X \\
\hline
\forall\, p : \mathit{DatatypeProperty};\; d : \mathbb{P}\, X \bullet \mathit{rangeD}(p) = d \Leftrightarrow \\
\quad \operatorname{ran}(\mathit{sub\_valD}(p)) \subseteq d
\end{array}$$

C.3.2 Relations to Other Properties

A property is equivalent to another property if its property extension is the same as that of the other. This property is also defined for both object and datatype properties.

$$\begin{array}{l}
[X] \\
\hline
\mathit{equivalentProperty} : \mathit{Property} \leftrightarrow \mathit{Property} \\
\hline
\forall\, p_1, p_2 : \mathit{Property} \bullet p_1 \,\underline{\mathit{equivalentProperty}}\, p_2 \Leftrightarrow \\
\quad ((p_1 \in \mathit{ObjectProperty} \wedge p_2 \in \mathit{ObjectProperty}) \Rightarrow \\
\qquad \mathit{sub\_val}(p_1) = \mathit{sub\_val}(p_2)) \wedge \\
\quad ((p_1 \in \mathit{DatatypeProperty} \wedge p_2 \in \mathit{DatatypeProperty}) \Rightarrow \\
\qquad \mathit{sub\_valD}[X](p_1) = \mathit{sub\_valD}[X](p_2))
\end{array}$$

The inverse of a property is another property with their domains and ranges flipped. It is only applicable to object properties as the domain and range of such properties must be of the same type.

$$\begin{array}{l}
\mathit{inverseOf} : \mathit{ObjectProperty} \leftrightarrow \mathit{ObjectProperty} \\
\hline
\forall\, p_1, p_2 : \mathit{ObjectProperty} \bullet p_1 \,\underline{\mathit{inverseOf}}\, p_2 \Leftrightarrow \\
\quad \mathit{sub\_val}(p_1) = (\mathit{sub\_val}(p_2))^{\sim}
\end{array}$$

C.3.3 Global Cardinality Constraints on Properties

A functional property is a property that can have only one (unique) value in its range for each instance in its domain.

$$
\begin{array}{l}
[X] \\
\hline
\mathit{functionalProperty} : \mathbb{P}\, \mathit{Property} \\
\hline
\forall\, p : \mathit{Property} \bullet p \in \mathit{functionalProperty} \Leftrightarrow \\
\quad (p \in \mathit{ObjectProperty} \Rightarrow (\forall\, a : \mathrm{dom}(\mathit{sub\_val}(p)) \bullet \\
\qquad \#(\mathit{sub\_val}(p)(\!|\, \{a\} \,|\!)) = 1)) \land \\
\quad (p \in \mathit{DatatypeProperty} \Rightarrow (\forall\, a : \mathrm{dom}(\mathit{sub\_valD}[X](p)) \bullet \\
\qquad \#(\mathit{sub\_valD}[X](p)(\!|\, \{a\} \,|\!)) = 1))
\end{array}
$$

An object property can be declared to be inverse-functional. If a property is declared to be inverse-functional, the object of a property statement uniquely determines the subject (some individual).

$$
\begin{array}{l}
\mathit{InverseFunctionalProperty} : \mathbb{P}\, \mathit{ObjectProperty} \\
\hline
\forall\, p : \mathit{ObjectProperty} \bullet p \in \mathit{InverseFunctionalProperty} \Leftrightarrow \\
\quad (\forall\, a, b, c : \mathit{Individual} \mid (a, c) \in \mathit{sub\_val}(p) \land (b, c) \in \mathit{sub\_val}(p) \bullet a = b)
\end{array}
$$

C.3.4 Logical Characteristics of Properties

An object property can also be declared as being transitive. Formally speaking, if pairs of individuals (a, b) and (b, c) are instances (members of the property extension) of property p, then we can infer that (a, c) is also an instance of p.

$$
\begin{array}{l}
\mathit{TransitiveProperty} : \mathbb{P}\, \mathit{ObjectProperty} \\
\hline
\forall\, p : \mathit{ObjectProperty} \bullet p \in \mathit{TransitiveProperty} \Leftrightarrow \\
\quad (\forall\, a, b, c : \mathit{Individual} \bullet (a, b) \in \mathit{sub\_val}(p) \land (b, c) \in \mathit{sub\_val}(p) \Rightarrow \\
\qquad (a, c) \in \mathit{sub\_val}(p))
\end{array}
$$

A symmetric property is a property for which holds that if the pair (x, y) is an instance of a property p, then the pair (y, x) is also an instance of p. As the same reason above, *SymmetricProperty* is a sub set of *ObjectProperty*.

$$\begin{array}{|l}
\mathit{SymmetricProperty} : \mathbb{P}\, \mathit{ObjectProperty} \\
\hline
\forall\, p : \mathit{ObjectProperty} \bullet p \in \mathit{SymmetricProperty} \Leftrightarrow \\
\qquad (\forall\, a, b : \mathit{Individual} \bullet (a, b) \in \mathit{sub\_val}(p) \Rightarrow (b, a) \in \mathit{sub\_val}(p))
\end{array}$$

C.4 Individuals

This section describes the properties that OWL defines for individuals.

C.4.1 Individual Identity

OWL provides three properties for stating the identity of an individual.

In OWL DL, the *sameAs* property states that two individuals are same as each other.

$$\begin{array}{|l}
\mathit{sameAs} : \mathit{Individual} \leftrightarrow \mathit{Individual}
\end{array}$$

On the contrary to *sameAs*, *differentFrom* states that two individuals are actually different.

$$\begin{array}{|l}
\mathit{differentFrom} : \mathit{Individual} \leftrightarrow \mathit{Individual}
\end{array}$$

The property *AllDifferent* is defined in OWL for convenience to state the pairwise disjointness among a list of individuals.

$$\begin{array}{|l}
\mathit{AllDifferent} : \mathbb{P}(\mathrm{seq}\, \mathit{Individual}) \\
\hline
\forall\, \mathit{ins} : \mathrm{seq}\, \mathit{Individual} \bullet \mathit{ins} \in \mathit{AllDifferent} \Leftrightarrow \\
(\forall\, x, y : \mathit{ins} \mid x.1 \neq y.1 \bullet x.2\ \underline{\mathit{differentFrom}}\ y.2)
\end{array}$$