

COMP4100 Lectures in 2007

by Malcolm Newey

(Relatively) Advanced Z

Lectures 9-10 March 16-19, 2005

References:

Grassman & Tremblay, Chapter 8

Davies & Woodcock, *Using Z – Specification ...*

<http://www.usingZ.com>

Spivey, *The Z Notation – a reference manual*

<http://spivey.oriel.ox.ac.uk/~mike/zrm/>

Declarations and Definitions

Some of these will be new to you.

- Basic or given types
- Abbreviations
- Data type definitions
- Schema definitions
- Axiomatic definitions
- Generic axiomatic definitions

Basic Types

Given types are collections of individuals that we don't regard as structured (in the current Z specification).

- [Student]
- [City]
- [Date]

Abbreviations

Abbreviations have the form $\langle name \rangle == \langle expression \rangle$.

They define something new in terms of already known things.

(Described in Z syntax as constant declaration.)

Examples:

$$weekend == \{saturday, sunday\}$$

$$evenNums == \{j : \mathbb{Z} \mid j \bmod 2 = 0\}$$

$$evenPred == \lambda n \bullet n \in evenNums$$

Data Type Definitions

Data type definitions have the form:

$$\langle \textit{typename} \rangle ::= t_1 \mid t_2 \mid \dots \mid t_n.$$

They are used to define enumerated types.

Familiar example:

$$\textit{PossibleMessages} ::= \textit{ok} \mid \textit{already_in_class} \mid \textit{full} \mid \textit{not_in_class} \mid \textit{two_errors}$$

Schema Definitions

Schema definitions have the form:

$$\langle \textit{schema name} \rangle \hat{=} \langle \textit{schema expression} \rangle.$$

Familiar example:

$$\textit{AddClass} \hat{=} (\textit{AddClass}_o \wedge \textit{OkMessage}) \vee \textit{AddClassError}$$

Axiomatic Definitions

An axiomatic definition introduces a new object (of some given type) that satisfies some constraint.

$$\frac{}{x : T}$$

$$\frac{}{p(x)}$$

$$\frac{}{\textit{primes} : \mathbb{P}\mathbb{N}}$$

$$\frac{}{\forall p : \textit{primes} \bullet p > 1}$$

$$\frac{}{\forall p : \textit{primes}; j : \mathbb{N} \mid 1 < j < p \bullet (p \bmod j) > 0}$$

$$\frac{}{\textit{classLimit} : \mathbb{N}}$$

$$\frac{}{\textit{classLimit} : \mathbb{N}}$$

Inconsistency of Axiomatic Definitions

If an object x is introduced by an axiomatic definition, there is a danger that a contradiction could be derived. For example:

$$\frac{}{\textit{special} : \mathbb{N}}$$

$$\frac{}{\textit{special} < 0}$$

Showing Consistency of Axiomatic Definitions

Suppose x is defined as follows:

$$\left| \begin{array}{l} x : T \\ \hline p(x) \end{array} \right.$$

Consistency is guaranteed, however, if we can prove

$$\exists y : T \bullet p(y)$$

Consistency Problem: Empty Domains

If an object x is introduced by an axiomatic definition with no predicate part, the chance of inconsistency is not eliminated.

$$\left| \begin{array}{l} x : T \end{array} \right.$$

The danger that a contradiction could be derived lies in the possibility that T might be the empty set.

Generic Axiomatic Definitions

Where we have a structured type with a free type variable (such as $\mathbb{P} T \rightarrow \mathbb{P} T$) we will usually want to define polymorphic constants and functions across this family of types. To do this we use generic axiomatic definitions which look like:

$$\left[\begin{array}{l} T \\ \hline x : S \\ \hline p(x) \end{array} \right.$$

We would expect the the identifier T to be the whole of or a part of the type expression S .

Example: The Empty Set

For every type T we often need to refer to the empty set of objects of type T ; the polymorphic empty set is defined this way:

$$\left[\begin{array}{l} T \\ \hline \emptyset : \mathbb{P} T \\ \hline \forall x : T \bullet x \notin \emptyset \end{array} \right.$$

Example: The Subset Relation

The typical basic operations on sets need to be polymorphic.

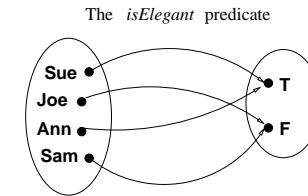
That is, we must be able to talk about the union of two sets of objects of any given type.

The following is how we define, for example, the subset relation between two sets of objects of type T .

$$\begin{aligned} & \text{---} [T] \text{---} \\ & \text{---} \subseteq \text{---} : \mathbb{P} T \leftrightarrow \mathbb{P} T \\ & \text{---} \forall a, b : \mathbb{P} T \bullet (a \subseteq b) \Leftrightarrow (\forall x : T \bullet x \in a \Rightarrow x \in b) \end{aligned}$$

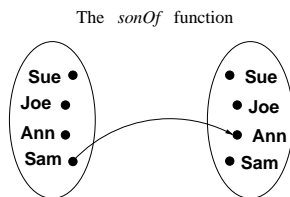
Total Functions

- A total function is one that is *defined* (ie not undefined) on every argument from its domain. For example, the predicate *even* over the integers is total.
- That is, a total function maps every element of its domain to some element of its range.
- **Notation:** $F : X \rightarrow Y$ says that F is total.



Partial Functions

- A function from X to Y is a relation that maps each element of X to at most one element of Y .
- If f is a function, so defined, we say f is a *partial* function to emphasize that the definedness attribute of total functions may not apply. Square root is a function on the integers but is not total.
- **Notation:** $F : X \mapsto Y$ says that F is a partial function.



Total Functions are a subset of Relations

- The declaration $a : X$ says that a is an element of the set X . (Types are sets!)
- Analogously, the declaration $F : X \rightarrow Y$ indicates that F is in the set $X \rightarrow Y$!
- By the same token, $X \mapsto Y$ and $X \leftrightarrow Y$ are sets.
- It follows directly from their definitions that

$$(X \rightarrow Y) \subset (X \mapsto Y) \subset (X \leftrightarrow Y)$$

That is, each total function is a partial function and hence a relation.

Partial Functions - The Official Definition

So, if $X \leftrightarrow Y$ is a set we should be able to define it.

How about this?

$$X \leftrightarrow Y \equiv \{f : X \leftrightarrow Y \mid \forall x : X; y_1, y_2 : Y \bullet ((x \mapsto y_1) \in f) \wedge ((x \mapsto y_2) \in f) \Rightarrow (y_1 = y_2)\}$$

Total Functions - The Definition

It is easy to characterize total functions in terms of partial ones.

$$X \rightarrow Y \equiv \{f : X \leftrightarrow Y \mid \text{dom } f = X\}$$

Ways of Defining Functions

- Defining by abbreviation:

$$\text{square} \equiv \{j : \mathbb{Z} \bullet (j \mapsto j * j)\}$$

- Constructing an axiomatic definition:

$$\begin{array}{l} \text{factorial} : \mathbb{N} \rightarrow \mathbb{N} \\ \hline \text{factorial}(0) = 1 \\ \forall n : \mathbb{N} \bullet n > 0 \Rightarrow \text{factorial}(n) = n * \text{factorial}(n - 1) \end{array}$$

- Using lambda notation (as follows)

Lambda Notation

- Usual form of a lambda term:

$$\lambda \langle \text{declaration} \rangle \bullet \langle \text{expression} \rangle$$

- Example:

$$\lambda m, n : \mathbb{N} \bullet \text{if } m > n \text{ then } m \text{ else } n$$

- General form of a lambda term:

$$\lambda \langle \text{declaration} \rangle \mid \langle \text{predicate} \rangle \bullet \langle \text{expression} \rangle$$

- Example:

$$\lambda m, n : \mathbb{N} \mid m > n \bullet m$$

is a partial function over pairs of natural numbers.

Toolkit Functions

The usual operators that we use to get the domain and range of relations are functions.

$$\begin{array}{l} \text{dom} : (A \leftrightarrow B) \rightarrow \mathbb{P}A \\ \text{ran} : (A \leftrightarrow B) \rightarrow \mathbb{P}B \\ \forall R : A \leftrightarrow B \bullet \text{dom}(R) = \{a : A \mid \exists b : B \bullet (a \mapsto b) \in R\} \\ \forall R : A \leftrightarrow B \bullet \text{ran}(R) = \{b : B \mid \exists a : A \bullet (a \mapsto b) \in R\} \end{array}$$

More Toolkit Functions

The usual operators that we use to restrict the domain and range of relations are also functions.

$$\begin{array}{l} _ \triangleleft _ : \mathbb{P}A \times (A \leftrightarrow B) \rightarrow (A \leftrightarrow B) \\ _ \triangleright _ : (A \leftrightarrow B) \times \mathbb{P}B \rightarrow (A \leftrightarrow B) \\ \forall Q : \mathbb{P}A; R : A \leftrightarrow B \bullet \\ \quad Q \triangleleft R = \{a : A; b : B \mid (a \in Q) \wedge (a \mapsto b) \in R \bullet a \mapsto b\} \\ \forall R : A \leftrightarrow B; Q : \mathbb{P}B \bullet \\ \quad R \triangleright Q = \{a : A; b : B \mid (a \mapsto b) \in R \wedge (b \in Q) \bullet a \mapsto b\} \end{array}$$

Yet More Toolkit Functions

The domain and range of relations can be restricted in the complementary way to \triangleleft and \triangleright

$$\begin{array}{l} _ \triangleleft _ : \mathbb{P}A \times (A \leftrightarrow B) \rightarrow (A \leftrightarrow B) \\ _ \triangleright _ : (A \leftrightarrow B) \times \mathbb{P}B \rightarrow (A \leftrightarrow B) \\ \forall Q : \mathbb{P}A; R : A \leftrightarrow B \bullet \\ \quad Q \triangleleft R = \{a : A; b : B \mid (a \notin Q) \wedge (a \mapsto b) \in R \bullet a \mapsto b\} \\ \forall R : A \leftrightarrow B; Q : \mathbb{P}B \bullet \\ \quad R \triangleright Q = \{a : A; b : B \mid (a \mapsto b) \in R \wedge (b \notin Q) \bullet a \mapsto b\} \end{array}$$

Injections, Surjections and Bijections

Consider a function $f : X \mapsto Y$.

(Note that the domain of f is contained in X . Similarly, $\text{ran}(f) \subseteq Y$.)

- f is injective if it maps every element of its domain to a distinct element in the range. (It is also called a *1 to 1 function*.)
- f is surjective if its range is the whole of the target set, Y . (It is also described as *onto*.)
- f is bijective if it's both injective and surjective. (Bijections also called *1 to 1 correspondences*.)

Injections, Surjections and Bijections - Notation

When these are properties apply to functions in a Z specification, it may be useful to declare them prominently. Thus there are six new symbols.

$f : X \mapsto Y$	f is partial and injective,
$f : X \twoheadrightarrow Y$	f is total and injective,
$f : X \dashrightarrow Y$	f is partial and surjective,
$f : X \twoheadrightarrow Y$	f is total and surjective,
$f : X \xrightarrow{\sim} Y$	f is total and bijective

Finite Sets

- $x : \mathbb{F} X$ indicates x must be a finite set.
(If $x : \mathbb{P} \mathbb{N}$ then x may be infinite.)
- The type of the cardinality function, $\#$, is $\mathbb{F} X \rightarrow \mathbb{N}$.
- Finite functions have types like $A \mapsto B$.
- Finite injections have types like $A \mapsto B$.

Overriding

If we use the union operation to combine two functions, then we do not necessarily get a function.

If f and g are functions of the same type, then $f \oplus g$ is a function that combines f and g , with g 'taking preference'.

$$\frac{}{_ \oplus _ : (A \leftrightarrow B) \times (A \leftrightarrow B) \rightarrow (A \leftrightarrow B)}$$

$$\frac{\forall f, g : A \leftrightarrow B \bullet}{f \oplus g = (\text{dom } g \triangleleft f) \cup g}$$

Sequences

In mathematics a *sequence* is an ordered collection of objects of some type. In programming we tend to refer to such a thing as a *list*.

Sequences differ from sets in two important ways; they are ordered and they may have some values duplicated.

- $\langle 3, 2, 4, 2, 3 \rangle$
- $\langle \rangle$
- $\langle \langle T, F \rangle, \langle T, T, T \rangle, \langle \rangle \rangle$

Basic Operations on Sequences

Four destructors:

- $head \langle 3, 2, 4, 2, 1 \rangle = 3$
- $tail \langle 3, 2, 4, 2, 1 \rangle = \langle 2, 4, 2, 1 \rangle$
- $last \langle 3, 2, 4, 2, 1 \rangle = 1$
- $front \langle 3, 2, 4, 2, 1 \rangle = \langle 3, 2, 4, 2 \rangle$

One constructor - concatenation:

- $\langle d, o, g \rangle \hat{\ } \langle w, o, o, d \rangle = \langle d, o, g, w, o, o, d \rangle$

The types of *head*, *last* are both $seq A \mapsto A$

The types of *tail*, *front* are both $seq A \mapsto seq A$

The type of $\hat{\ }$ is $(seq A \times seq A) \mapsto seq A$

More Standard Operations on Sequences

- Length – $\# : seq A \mapsto \mathbb{N}$
 $\# \langle 3, 2, 4, 2, 1 \rangle = 5$
- Reverse – $rev : seq A \mapsto seq A$
 $rev \langle 3, 2, 4, 2, 1 \rangle = \langle 1, 2, 4, 2, 3 \rangle$
- Filter – $\upharpoonright : seq A \times seq A \mapsto seq A$
 $\langle 3, 2, 4, 2, 1 \rangle \upharpoonright \langle 1, 2 \rangle = \langle 2, 2, 1 \rangle$
- Prefix – $\subseteq : seq A \times seq A \mapsto \mathbb{B}$
 $\langle 3, 2, 4 \rangle \subseteq \langle 3, 2, 4, 2, 1 \rangle = true$

Indexing on a Sequence

In mathematics, a sequence x of, say, four elements is often written as x_1, x_2, x_3, x_4 .

- The function $\{(1 \mapsto x_1), (2 \mapsto x_2), (3 \mapsto x_3), (4 \mapsto x_4)\}$ indexes the sequence appropriately.
- That function can be used as a model for the sequence.
- The domain and range of the function match our expectations as to the domain and range of the sequence.

Sequences as Functions

Z takes the line that sequences *are* the functions suggested above.

So $\langle 5, 7, 1 \rangle$ is defined to *be* the function $\{(1 \mapsto 5), (2 \mapsto 7), (3 \mapsto 1)\}$.

A variable S which is a sequence of values of type X is declared as $S : seq X$.

It follows that the set $seq X$ is $\{s : \mathbb{N} \mapsto X \mid (\exists n : \mathbb{N} \bullet \text{dom } s = 1..n)\}$
Recall that the range $1..3$ is the set $\{1, 2, 3\}$. Accordingly, the expression $1..0$ indicates the empty set.

Consequently, indexing is easy: the n th element of s is $s(n)$.

Prefix Revisited

- When we introduced it, \subseteq may have seemed like an odd choice as name for the prefix relation.
- Perhaps we might have thought that \subseteq would be a good name for a *subsequence* relationship.
- Fortuitously, the subset relation matches what we want as the prefix relation. If sequence A, of size n , is a subset of sequence B, then each maplet ($j \mapsto x_j$) in A must also be in B. That is the n elements of A are the first n elements of B.

Defining Head and Tail

$[A]$
$head : seq A \leftrightarrow A$ $tail : seq A \leftrightarrow seq A$
$\forall s : seq A \mid (s \neq \langle \rangle) \bullet ($ $ head(s) = s(1) \wedge$ $ \#(tail\ s) = \#s - 1 \wedge$ $ \forall j : 1..(\#s - 1) \bullet (tail\ s)j = s(j + 1))$

Defining Functions on Sequences

- Sequences will be used in a Z specification to model system components or aspects that can be thought of as being *lists* or *arrays* of items.
- We can use the intuitions we have from Haskell to code the predicates in axiomatic definitions of operations on sequences.
We have *head* and *tail* and we can code *cons* as

$$x\ cons\ y = \langle x \rangle \hat{\ } y$$
- The induction principle for lists that we know can be applied to sequences.

Defining Reverse

$[A]$
$rev : seq A \rightarrow seq A$
$rev \langle \rangle = \langle \rangle$ $\forall x : A; s : seq A \bullet rev(\langle x \rangle \hat{\ } s) = (rev\ s) \hat{\ } \langle x \rangle$

Defining Filter

$$\begin{array}{l} \text{[A]} \\ \hline \upharpoonright : \text{seq } A \times \text{seq } A \rightarrow \text{seq } A \\ \forall s : \text{seq } A \bullet \langle \rangle \upharpoonright s = \langle \rangle \\ \forall x : A; s, t : \text{seq } A \bullet (\langle x \rangle \hat{\ } s) \upharpoonright t = \\ \quad \text{if } x \in \text{ran } t \text{ then } \langle x \rangle \hat{\ } (s \upharpoonright t) \text{ else } s \upharpoonright t \end{array}$$

Structural Induction Principle for Sequences

$$\frac{P(\langle \rangle) \quad \forall x : A; s : \text{seq } A \bullet P(s) \Rightarrow P(\langle x \rangle \hat{\ } s)}{\forall s : \text{seq } A \bullet P(s)}$$

What are Bags?

A *bag* is a collection of objects where

- Order doesn't matter;
- Multiple instances are allowed and the number of these instances does matter.

Examples:

- $\llbracket G, R, G, R, G \rrbracket$ is a bag with 5 elements.
- $\llbracket G, G, G, R, R \rrbracket$ is the same bag as above.
- $\llbracket G, G, R, R \rrbracket$ is different to the above.

Bag Notation

- The empty bag is denoted by $\llbracket \rrbracket$.
- To declare b to be a bag of items of type X , write $b : \text{bag } X$.
- Bag membership is tested with relation called 'in'.
 $(5 \text{ in } \llbracket 2, 3, 5, 4, 5, 1 \rrbracket) = \text{true}$.
- If x has multiplicities n_1 and n_2 in bags b_1 and b_2 then it will have multiplicity $(n_1 + n_2)$ in the union of the two bags, $b_1 \uplus b_2$.
- The function call $(\text{count } b \ x)$ gives the multiplicity of x in bag b .

Bags Modeled!

- The definition:
 $\text{bag } X ::= X \mapsto \mathbb{N}_1$
 specifies how 'bag of X' is modeled.
- Why is it a partial function?
- Why is the range \mathbb{N}_1 ?

The Definition of *count*

$$\begin{array}{l} \text{count} : \text{bag } X \mapsto (X \rightarrow \mathbb{N}) \\ \forall x : X; B : \text{bag } X \bullet \text{count } B = (\lambda x : X \bullet 0) \oplus B \end{array}$$

- What are the arrows? Why are they so chosen?
- Why does this definition work?

The Definition of *bag union*

$$\begin{array}{l} _ \uplus _ : (\text{bag } A \times \text{bag } A) \rightarrow \text{bag } A \\ \forall x : A; b_1, b_2 : \text{bag } A \bullet \text{count}(b_1 \uplus b_2)x = (\text{count } b_1 x) + (\text{count } b_2 x) \end{array}$$

- Why?
- Why use *count*?

Connection with Sequences

Sequences and bags both can have multiple occurrences of an element.
 Removing order from a sequence gives a bag.

$$\begin{array}{l} \text{items} : \text{seq } X \rightarrow \text{bag } X \\ \forall S : \text{seq } X \bullet \text{items } S = \{x : X \mid (x \text{ in } S) \bullet (x \mapsto \#(S \triangleright \{x\}))\} \end{array}$$

Example:

- $\langle 2, 3, 5, 2, 3 \rangle$ is modeled by $\{1 \mapsto 2, 2 \mapsto 3, 3 \mapsto 5, 4 \mapsto 2, 5 \mapsto 3\}$
- $(2 \mapsto \#(\langle 2, 3, 5, 2, 3 \rangle \triangleright \{2\}))$
 $= (2 \mapsto \#(\{1 \mapsto 2, 4 \mapsto 2\})) = (2 \mapsto 2)$ etc.
- $\text{items}\langle 2, 3, 5, 2, 3 \rangle = \{2 \mapsto 2, 3 \mapsto 2, 5 \mapsto 1\} = \llbracket 2, 2, 3, 3, 5 \rrbracket$

Schemas as Types

Vector

$X : \mathbb{R}$

$Y : \mathbb{R}$

$Z : \mathbb{R}$

$$(X^2 + Y^2 + Z^2) < 100$$

- A schema with variable declarations D and predicate P denotes the set of all objects with components that are instances of the variables in D that satisfy P .
- So the above example is a schema denoting the set of real number triples with the sum of squares less than 100.
- It is the set of vectors in Cartesian space less than 10 long.

Projection Functions

Assume that variable v is declared by $v : \text{Vector}$ where *Vector* is defined as follows:

Vector

$x : \mathbb{R}$

$y : \mathbb{R}$

$z : \mathbb{R}$

$$(x^2 + y^2 + z^2) < 100$$

- The X, Y and Z components of v are $x(v)$, $y(v)$ and $z(v)$.
- Alternative notation is $v.x$, $v.y$ and $v.z$