

Advanced Generic Definitions

[BGA]

Example 1 : Generic Three - Way Product

Generic function for three-component tuples:

$\equiv [X, Y, Z] =$

$triple : X \times Y \times \mathbb{Z} \rightarrow X \times Y \times \mathbb{Z}$

$getFirst : X \times Y \times \mathbb{Z} \rightarrow X$

$getSecond : X \times Y \times \mathbb{Z} \rightarrow Y$

$getThird : X \times Y \times \mathbb{Z} \rightarrow \mathbb{Z}$

$\forall x : X \bullet \forall y : Y \bullet \forall z : \mathbb{Z} \bullet triple(x, y, z) = (x, y, z) \wedge$

$getFirst(x, y, z) = x \wedge$

$getSecond(x, y, z) = y \wedge$

$getThird(x, y, z) = z$

Projection functions for triples.

Example 2 : Generic Binary Tree (Conceptual)

Binary trees are typically defined for specific types in Z notation rather than as fully polymorphic structures. For a concrete type like N, you would *define* : Tree ::= leaf | node{N cross Tree cross Tree}. Generic trees require complex workarounds beyond standard gedef blocks, as free types in fuzz are not easily parameterized.

Example 6 : Generic State Schema

A generic container with capacity:

Note: Fuzz supports generic schemas using the schema[X] syntax (not gendef):

Container[X] _____

contents : seq X

capacity : \mathbb{N}

#*contents* \leq *capacity*

This schema is parameterized by the element type X.

It can be instantiated as Container[N], Container[Char], etc.

Example 7 : Generic Relation Operations

Relational composition for any types:

$$\frac{[X, Y, Z] \quad \text{compose} : (X \leftrightarrow Y) \times (Y \leftrightarrow Z) \longrightarrow (X \leftrightarrow Z)}{\forall R : X \leftrightarrow Y \bullet \forall S : Y \leftrightarrow Z \bullet \text{compose}(R, S) = R ; S}$$

Composes two relations using forward composition.

Example 8 : Generic Option Type (Conceptual)

Optional values (like Maybe or Option in other languages) are challenging to express generically in Z notation due to limitations with parameterized free types. For a specific type, you would *define* : $Option ::= \text{none} \mid \text{some}\langle T \rangle$. Fully generic option types with operations require advanced patterns beyond standard gedef blocks.

Example 9 : Generic Stack Schema

Note: Define generic schema separately, then declare operations in gedef:

$Stack[X]$
$items : \text{seq } X$
$\text{maxSize} : \mathbb{N}$
$\#items \leq \text{maxSize}$
<hr/>
$[X]$
$\text{emptyStack} : Stack[X]$
$\text{stackSize} : Stack[X] \rightarrow \mathbb{N}$
$\text{emptyStack}.items = \langle \rangle \wedge \text{emptyStack}.maxSize = 100$
$\forall s : Stack[X] \bullet \text{stackSize}(s) = \#s.items$
<hr/>

Generic stack with query operations.

The schema is defined separately using schema[X] syntax, then operations use gedef.

Example 10 : Generic Pair Schema

Use schema[X, Y] syntax for generic schemas:

$OrderedPair[X, Y]$
$first : X$
$second : Y$

A generic pair type.

The schema has two type parameters and can be instantiated as OrderedPair[N, Char], etc.

Example 11 : Generic Zip Function

Combine two sequences into a sequence of pairs:

$[X, Y]$
$\text{zip} : \text{seq } X \times \text{seq } Y \rightarrow \text{seq } (X \times Y)$
$\text{zip}(\langle \rangle, \langle \rangle) = \langle \rangle$
$\forall x : X \bullet \forall y : Y \bullet \forall xs : \text{seq } X \bullet \forall ys : \text{seq } Y \bullet \text{zip}(\langle x \rangle \cap xs, \langle y \rangle \cap ys) =$
$\quad \langle (x, y) \rangle \cap \text{zip}(xs, ys)$
$\forall xs : \text{seq } X \bullet \text{zip}(xs, \langle \rangle) = \langle \rangle$
$\forall ys : \text{seq } Y \bullet \text{zip}(\langle \rangle, ys) = \langle \rangle$

Pairs up corresponding elements from two sequences.

Example 12 : Practical Example - Generic Database TableGA

Note: Define generic schema separately, then query operations in gedef:

$[Key, Value]$

$\begin{array}{l} \text{--- } TableGA[Key, Value] \text{ ---} \\ | \\ entries : Key \rightarrow Value \\ size : \mathbb{N} \\ | \\ size = \#(\text{dom } entries) \end{array}$

$\begin{array}{l} \text{--- } [Key, Value] \text{ ---} \\ | \\ emptyTable : TableGA[Key, Value] \\ tableSize : TableGA[Key, Value] \rightarrow \mathbb{N} \\ allKeys : TableGA[Key, Value] \rightarrow \mathbb{P} Key \\ | \\ emptyTable.entries = \{\} \wedge emptyTable.size = 0 \\ \forall t : TableGA[Key, Value] \bullet tableSize(t) = t.size \\ \forall t : TableGA[Key, Value] \bullet allKeys(t) = \text{dom } t.entries \end{array}$

A generic key-value table with query operations. This demonstrates how generic definitions enable reusable, type-safe data structures.

The TableGA schema is defined with schema[Key, Value] syntax, allowing proper fuzz typechecking.