User - Defined Squash Function

The squash operator appears in the Z notation glossary but is not built into the fuzz type-checker. However, we can define it ourselves as a user-defined function using an axiomatic specification.

Squash takes a partial function from N1 (positive naturals) to some type X and compacts it into a sequence by removing gaps in the domain. For example, $\{1 \mapsto a, 3 \mapsto b, 7 \mapsto c\}$ becomes $\langle a, b, c \rangle$.

Example 1 : Axiomatic Definition of Squash

We define squash axiomatically by specifying its key properties using only fuzz-compatible notation.

$$[X]$$

$$\begin{array}{|l} \mathit{squashFunc} : (\mathbb{N}_1 \pfun X) \to \operatorname{seq} X \\ \hline \forall f : \mathbb{N}_1 \pfun X \bullet \operatorname{ran}(\mathit{squashFunc}(f)) = \operatorname{ran} f \land \#(\mathit{squashFunc}(f)) = \#(\operatorname{ran} f) \end{array}$$

This axiomatically defines squashFunc (a user-defined version of squash) with two key properties that hold for all partial functions f from N1 to X: (1) squashFunc preserves the *range* : *ran* (squashFunc f) = ran f, and (2) the resulting sequence length equals the range cardinality: # (squashFunc f) = # (ran f). These properties characterize squashFunc as compacting partial functions by removing domain gaps while preserving all elements and their relative order.

Example 2 : Using Squash on a Simple Function

Demonstrate squash on a sparse partial function.

$$\begin{array}{|l} f1 : \mathbb{N}_1 \pfun \mathbb{N} \\ s1 : \operatorname{seq} \mathbb{N} \\ \hline f1 = \{1 \mapsto 10, 3 \mapsto 30, 5 \mapsto 50\} \land s1 = \mathit{squashFunc}(f1) \end{array}$$

The result s1 is $\langle 10, 30, 50 \rangle$, containing exactly the range elements in the order of their original indices.

Example 3 : Squash with Letters

$$[\mathit{Letter}]$$

$$\begin{array}{|l} a : \mathit{Letter} \\ b : \mathit{Letter} \\ c : \mathit{Letter} \\ d : \mathit{Letter} \\ f2 : \mathbb{N}_1 \pfun \mathit{Letter} \\ s2 : \operatorname{seq} \mathit{Letter} \\ \hline f2 = \{2 \mapsto a, 4 \mapsto b, 7 \mapsto c, 9 \mapsto d\} \land s2 = \mathit{squashFunc}(f2) \end{array}$$

The result s2 is $\langle a, b, c, d \rangle$. The gaps at positions 1, 3, 5, 6, 8 are removed.

Example 4 : Empty Function

$$f3 : \mathbb{N}_1 \pfun \mathbb{N}$$
$$s3 : \text{seq } \mathbb{N}$$
$$\vdash f3 = \{\} \land s3 = squashFunc(f3)$$

Squashing an empty function yields an empty *sequence* : $s3 = \langle\rangle$.

Example 5 : Continuous Function (No Gaps)

$$f4 : \mathbb{N}_1 \pfun \mathbb{N}$$
$$s4 : \text{seq } \mathbb{N}$$
$$\vdash f4 = \{1 \mapsto 100, 2 \mapsto 200, 3 \mapsto 300\} \land s4 = squashFunc(f4)$$

When the function has no gaps (dom $f4 = 1..3$), squashFunc simply converts it to a *sequence* : $s4 = \langle 100, 200, 300\rangle$.

Example 6 : Relationship Between Function land Sequence

For any partial function f with a continuous domain starting at 1, squashFunc is the identity.

$$continuous\_N : \mathbb{N}_1 \pfun \mathbb{N}$$
$$s\_continuous\_N : \text{seq } \mathbb{N}$$
$$\vdash \text{dom } continuous\_N = 1 \,..\, \#(\text{ran } continuous\_N) \land s\_continuous\_N = squashFunc(continuous\_N)$$

In this case, continuous_N is already a sequence, and squashFunc $continuous\_N = continuous\_N$ as a sequence.

Example 7 : Squash Preserves Injectivity

If the input function is injective, the output sequence has no duplicates.

$$inj\_func : \mathbb{N}_1 \pfun \mathbb{N}$$
$$inj\_seq : \text{seq } \mathbb{N}$$
$$\vdash (\forall i, j : \text{dom } inj\_func \bullet inj\_func(i) = inj\_func(j) \Rightarrow i = j) \land inj\_seq = squashFunc(inj\_func)$$

Since inj_func is injective (one-to-one), inj_seq contains no duplicate elements.

Example 8 : Practical Use Case - Video Database

$[VideoID, Title]$

$$sparse\_catalog : \mathbb{N}_1 \pfun (VideoID \times Title)$$
$$compact\_catalog : \text{seq } (VideoID \times Title)$$
$$\vdash compact\_catalog = squashFunc(sparse\_catalog)$$

A sparse catalog with gaps (deleted entries) can be compacted into a continuous sequence while preserving the order of remaining entries.