The following is what I call a ‘semantic design’ for Nu's scripting system. The concept of a semantic design is inspired by Conal Elliot’s denotational design - <https://www.youtube.com/watch?v=bmKYiUOEo2A>. The difference is that semantic design does not connect back to an intermediate language like mathematics, but is instead built upon axioms as expressed by just a minimal set of circular type and related function definitions.

Whereas denotational design is a more thorough design treatment that is used in greenfield development to yield high-precision design artifacts, semantic design works well for projects that don’t satisfy any simple denotational design, such as those that are already far into their implementation.

// A value abstraction.

type Value<a> =

| Relation of Value<Relation>

| Address of Value<Address>

| Name of Value<Name>

| Unit of Value<Unit>

get<a> : Name -> Relation -> Value<a> = get<a>

set<a> : Name -> Relation -> Value<a> -> Effect<a> = set<a>

// Augments the environment with a new definition of type a.

type Declare<a> = Declare<a>

declare<a> : Name -> a -> Declare<a> = declare<a>

// An effect on the environment parameterized with a value of type a.

type Effect = Effect

effect<a> : Value<a> -> Effect = effect<a>

// A stream abstraction.

type Stream<a> = Stream<a>

foldStream<a, b> : (Value<a> -> b) -> Stream<a> -> b = foldStream<a, b>

productStream<a, b> : Stream<a> -> Stream<b> -> Stream<a \* b> = productStream<a, b>

sumStream<a, b> : Stream<a> -> Stream<b> -> Stream<a | b> = sumStream<a, b>

eventStream<a> : Address -> Stream<a> = eventStream<a>

propertyStream<a> : Name -> Relation -> Stream<a> = propertyStream<a>

// Domain-level functions. Now that all the axioms are defined, we can create functions that at the domain level that are fully-defined.

define name value = declare name value

variable name stream = declare name stream

equate name relation stream = foldStream (set name relation) stream

handle effect stream = foldStream effect stream