Funcons-beta: Flowing

The PLanCompS Project

`Funcons-beta/Computations/Normal/Flowing/Flowing.cbs`*

Flowing

[*Funcon* left-to-right
Alias l-to-r
Funcon right-to-left
Alias r-to-l
Funcon sequential
Alias seq
Funcon effect
Funcon choice
Funcon if-true-else
Alias if-else
Funcon while-true
Alias while
Funcon do-while-true
Alias do-while
Funcon interleave
Datatype yielding
Funcon signal
Funcon yielded
Funcon yield
Funcon yield-on-value
Funcon yield-on-abrupt
Funcon atomic]

Meta-variables T <: values T* <: values *

*Suggestions for improvement: `plancomps@gmail.com`.
Issues: `https://github.com/plancomps/CBS-beta/issues`.

Sequencing

Funcon left-to-right(_ : (⇒(T)*)*) : ⇒(T)*

Alias l-to-r = left-to-right

left-to-right(···) computes its arguments sequentially, from left to right, and gives the resulting sequence of values, provided all terminate normally. For example, integer-add(X, Y) may interleave the computations of X and Y, whereas integer-add left-to-right(X, Y) always computes X before Y.

When each argument of left-to-right(···) computes a single value, the type of the result is the same as that of the argument sequence. For instance, when X : T and Y : T′, the result of left-to-right(X, Y) is of type (T, T′). The only effect of wrapping an argument sequence in left-to-right(···) is to ensure that when the arguments are to be evaluated, it is done in the specified order.

Rule $\dfrac{Y \longrightarrow Y'}{\text{left-to-right}(V^* : (T)^*, Y, Z^*) \longrightarrow \text{left-to-right}(V^*, Y', Z^*)}$

Rule left-to-right(V* : (T)*) ⇝ V*

Funcon right-to-left(_ : (⇒(T)*)*) : ⇒(T)*

Alias r-to-l = right-to-left

right-to-left(···) computes its arguments sequentially, from right to left, and gives the resulting sequence of values, provided all terminate normally.

Note that right-to-left(X*) and reverse left-to-right reverse(X*) are not equivalent: reverse(X*) interleaves the evaluation of X*.

Rule $\dfrac{Y \longrightarrow Y'}{\text{right-to-left}(X^*, Y, V^* : (T)^*) \longrightarrow \text{right-to-left}(X^*, Y', V^*)}$

Rule right-to-left(V* : (T)*) ⇝ V*

Funcon sequential(_ : (⇒ null-type)*, _ : ⇒ T) : ⇒ T

Alias seq = sequential

sequential(X, ···) computes its arguments in the given order. On normal termination, it returns the value of the last argument; the other arguments all compute null-value.

Binary sequential(X, Y) is associative, with unit null-value.

Rule $\dfrac{X \longrightarrow X'}{\mathsf{sequential}(X, Y^+) \longrightarrow \mathsf{sequential}(X', Y^+)}$

Rule $\mathsf{sequential}(\mathsf{null\text{-}value}, Y^+) \rightsquigarrow \mathsf{sequential}(Y^+)$

Rule $\mathsf{sequential}(Y) \rightsquigarrow Y$

Funcon $\mathsf{effect}(V^* : T^*) : \Rightarrow \mathsf{null\text{-}type}$
$\rightsquigarrow \mathsf{null\text{-}value}$

$\mathsf{effect}(\cdots)$ interleaves the computations of its arguments, then discards all the computed values.

Choosing

Funcon $\mathsf{choice}(\_ : (\Rightarrow T)^+) : \Rightarrow T$

$\mathsf{choice}(Y, \cdots)$ selects one of its arguments, then computes it. It is associative and commutative.

Rule $\mathsf{choice}(X^*, Y, Z^*) \rightsquigarrow Y$

Funcon $\mathsf{if\text{-}true\text{-}else}(\_ : \mathsf{booleans}, \_ : \Rightarrow T, \_ : \Rightarrow T) : \Rightarrow T$

Alias $\mathsf{if\text{-}else} = \mathsf{if\text{-}true\text{-}else}$

$\mathsf{if\text{-}true\text{-}else}(B, X, Y)$ evaluates B to a Boolean value, then reduces to X or Y, depending on the value of B.

Rule $\mathsf{if\text{-}true\text{-}else}(\mathsf{true}, X, Y) \rightsquigarrow X$

Rule $\mathsf{if\text{-}true\text{-}else}(\mathsf{false}, X, Y) \rightsquigarrow Y$

Iterating

Funcon $\mathsf{while\text{-}true}(B : \Rightarrow \mathsf{booleans}, X : \Rightarrow \mathsf{null\text{-}type}) : \Rightarrow \mathsf{null\text{-}type}$
$\rightsquigarrow \mathsf{if\text{-}true\text{-}else}(B,$
$\mathsf{sequential}(X,$
$\mathsf{while\text{-}true}(B,$
$X)),$
$\mathsf{null\text{-}value})$

Alias $\mathsf{while} = \mathsf{while\text{-}true}$

while-true(B, X) evaluates B to a Boolean value. Depending on the value of B, it either executes X and iterates, or terminates normally.

The effect of abruptly breaking the iteration is obtained by the combination handle-break(while-true(B, X)), and that of abruptly continuing the iteration by while-true$(B,$ handle-continue$(X))$.

Funcon do-while-true$(X :\Rightarrow$ null-type$, B :\Rightarrow$ booleans$) :\Rightarrow$ null-type
$\rightsquigarrow$ sequential$(X,$
if-true-else$(B,$
do-while-true$(X,$
$B),$
null-value$))$

Alias do-while = do-while-true

do-while-true(X, B) is equivalent to sequential$(X,$ while-true$(B, X))$.

Interleaving

Funcon interleave$(_ : T^*) :\Rightarrow T^*$

interleave$(\cdots)$ computes its arguments in any order, possibly interleaved, and returns the resulting sequence of values, provided all terminate normally. Fairness of interleaving is not required, so pure left-to-right computation is allowed.

atomic(X) prevents interleaving in X, except after transitions that emit a yielded(signal).

Rule interleave$(V^* : T^*) \rightsquigarrow V^*$

Datatype yielding ::= signal

Entity $\_ \xrightarrow{\text{yielded}(\_ : \text{yielding}^?)} \_$

yielded(signal) in a label on a transition allows interleaving at that point in the enclosing atomic computation. yielded( ) indicates interleaving at that point in an atomic computation is not allowed.

Funcon yield $:\Rightarrow$ null-type
$\rightsquigarrow$ yield-on-value(null-value)

Funcon $\mathsf{yield\text{-}on\text{-}value}(\_ : T) : \Rightarrow T$

$\mathsf{yield\text{-}on\text{-}value}(X)$ allows interleaving in an enclosing atomic computation on normal termination of X.

$$\textit{Rule}\ \ \mathsf{yield\text{-}on\text{-}value}(V : T) \xrightarrow{\mathsf{yielded}(\mathsf{signal})} V$$

Funcon $\mathsf{yield\text{-}on\text{-}abrupt}(\_ : \Rightarrow T) : \Rightarrow T$

$\mathsf{yield\text{-}on\text{-}abrupt}(X)$ ensures that abrupt termination of X is propagated through an enclosing atomic computation.

$$\textit{Rule}\ \ \frac{X \xrightarrow{\mathsf{abrupt}(V : T), \mathsf{yielded}(\_^?)} X'}{\mathsf{yield\text{-}on\text{-}abrupt}(X) \xrightarrow{\mathsf{abrupt}(V), \mathsf{yielded}(\mathsf{signal})} \mathsf{yield\text{-}on\text{-}abrupt}(X')}$$

$$\textit{Rule}\ \ \frac{X \xrightarrow{\mathsf{abrupt}(\ )} X'}{\mathsf{yield\text{-}on\text{-}abrupt}(X) \xrightarrow{\mathsf{abrupt}(\ )} \mathsf{yield\text{-}on\text{-}abrupt}(X')}$$

$$\textit{Rule}\ \ \mathsf{yield\text{-}on\text{-}abrupt}(V : T) \rightsquigarrow V$$

Funcon $\mathsf{atomic}(\_ : \Rightarrow T) : \Rightarrow T$

$\mathsf{atomic}(X)$ computes X, but controls its potential interleaving with other computations: interleaving is only allowed following a transition of X that emits $\mathsf{yielded}(\mathsf{signal})$.

$$\textit{Rule}\ \ \frac{X \xrightarrow{\mathsf{yielded}(\ )}_1 X' \qquad \mathsf{atomic}(X') \xrightarrow{\mathsf{yielded}(\ )}_2 X''}{\mathsf{atomic}(X) \xrightarrow{\mathsf{yielded}(\ )}_1; \xrightarrow{\mathsf{yielded}(\ )}_2 X''}$$

$$\textit{Rule}\ \ \frac{X \xrightarrow{\mathsf{yielded}(\ )} V \qquad V : T}{\mathsf{atomic}(X) \xrightarrow{\mathsf{yielded}(\ )} V}$$

$$\textit{Rule}\ \ \mathsf{atomic}(V : T) \rightsquigarrow V$$

$$\textit{Rule}\ \ \frac{X \xrightarrow{\mathsf{yielded}(\mathsf{signal})} X'}{\mathsf{atomic}(X) \xrightarrow{\mathsf{yielded}(\ )} \mathsf{atomic}(X')}$$