Funcons-beta: Flowing *

The PLanCompS Project

`Flowing.cbs` | PLAIN | PRETTY

OUTLINE

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`.
Reports of issues: `https://github.com/plancomps/CBS-beta/issues`.

Sequencing

Funcon left-to-right(_ : ($\Rightarrow(T)^*)^*$) : $\Rightarrow(T)^*$

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

left-to-right($\cdots$) 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($\cdots$) 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($\cdots$) is to ensure that when the arguments are to be evaluated, it is done in the specified order.

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

Rule left-to-right($V^* : (T)^*$) $\rightsquigarrow V^*$

Funcon right-to-left(_ : ($\Rightarrow(T)^*)^*$) : $\Rightarrow(T)^*$

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

right-to-left($\cdots$) 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^*.

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

Rule right-to-left($V^* : (T)^*$) $\rightsquigarrow V^*$

Funcon sequential(_ : ($\Rightarrow$ null-type)*, _ : $\Rightarrow T$) : $\Rightarrow T$

Alias seq = sequential

sequential($X, \cdots$) 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.

$$\textit{Rule}\quad \frac{X \longrightarrow X'}{\text{sequential}(X, Y^+) \longrightarrow \text{sequential}(X', Y^+)}$$

Rule sequential(null-value, Y^+) $\rightsquigarrow$ sequential(Y^+)

Rule sequential(Y) $\rightsquigarrow Y$

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

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

Choosing

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

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

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

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

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

Iterating

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

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

$\mathsf{while\text{-}true}(B, X)$ 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 $\mathsf{handle\text{-}break}(\mathsf{while\text{-}true}(B, X))$, and that of abruptly continuing the iteration by $\mathsf{while\text{-}true}(B, \mathsf{handle\text{-}continue}(X))$.

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

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

$\mathsf{do\text{-}while\text{-}true}(X, B)$ is equivalent to $\mathsf{sequential}(X, \mathsf{while\text{-}true}(B, X))$.

Interleaving

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

$\mathsf{interleave}(\cdots)$ 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.

$\mathsf{atomic}(X)$ prevents interleaving in X, except after transitions that emit a $\mathsf{yielded}(\mathsf{signal})$.

Rule $\mathsf{interleave}(V^* : T^*) \leadsto V^*$

Datatype $\mathsf{yielding} ::= \mathsf{signal}$

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

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

Funcon $\mathsf{yield} : \Rightarrow \mathsf{null\text{-}type}$
$\leadsto \mathsf{yield\text{-}on\text{-}value}(\mathsf{null\text{-}value})$

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

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

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

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

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

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

Rule $$\frac{X \xrightarrow{\textsf{abrupt}(\ )} X'}{\textsf{yield-on-abrupt}(X) \xrightarrow{\textsf{abrupt}(\ )} \textsf{yield-on-abrupt}(X')}$$

Rule $\textsf{yield-on-abrupt}(V : T) \rightsquigarrow V$

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

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

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

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

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

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