

Funcons-beta: Functions *

The P_{LAN}CompS Project

Functions.cbs | PLAIN | PRETTY

Functions

```
[ Datatype functions
  Funcon function
  Funcon apply
  Funcon supply
  Funcon compose
  Funcon uncurry
  Funcon curry
  Funcon partial-apply ]
```

Meta-variables T, T', T_1, T_2 <: values

Datatype `functions`(T, T') ::= `function`(A : `abstractions`($T \Rightarrow T'$))

`functions`(T, T') consists of abstractions whose bodies may depend on a given value of type T , and whose executions normally compute values of type T' . `function`(`abstraction`(X)) evaluates to a function with dynamic bindings, `function`(`closure`(X)) computes a function with static bindings.

Funcon `apply`($_ : \text{functions}(T, T'), _ : T$) : $\Rightarrow T'$

`apply`(F, V) applies the function F to the argument value V . This corresponds to call by value; using thunks as argument values corresponds to call by name. Moreover, using tuples as argument values corresponds to application to multiple arguments.

Rule `apply`(`function`(`abstraction`(X)), $V : T$) \rightsquigarrow `give`(V, X)

Funcon `supply`($_ : \text{functions}(T, T'), _ : T$) : \Rightarrow `thunks`(T')

`supply`(F, V) determines the argument value of a function application, but returns a thunk that defers executing the body of the function.

Rule `supply`(`function`(`abstraction`(X)), $V : T$) \rightsquigarrow `thunk`(`abstraction`(`give`(V, X)))

Funcon `compose`($_ : \text{functions}(T_2, T'), _ : \text{functions}(T_1, T_2)$) : \Rightarrow `functions`(T_1, T')

*Suggestions for improvement: plancomps@gmail.com.
Reports of issues: <https://github.com/plancomps/CBS-beta/issues>.

`compose`(F_2, F_1) returns the function that applies F_1 to its argument, then applies F_2 to the result of F_1 .

Rule `compose(function(abstraction(Y)), function(abstraction(X))) \rightsquigarrow
 function(abstraction(give(X, Y)))`

Funcon `uncurry($F : \text{functions}(T_1, \text{functions}(T_2, T'))$)`
 `: \Rightarrow functions(tuples(T_1, T_2), T')`
 \rightsquigarrow `function(`
 `abstraction(`
 `apply(`
 `apply(F , checked index(1, tuple-elements given)),`
 `checked index(2, tuple-elements given))`
 `)`

`uncurry`(F) takes a curried function F and returns a function that takes a pair of arguments..

Funcon `curry($F : \text{functions}(\text{tuples}(T_1, T_2), T')$) : \Rightarrow functions(T_1 , functions(T_2, T'))`
 \rightsquigarrow `function(abstraction(partial-apply(F , given)))`

`curry`(F) takes a function F that takes a pair of arguments, and returns the corresponding 'curried' function.

Funcon `partial-apply($F : \text{functions}(\text{tuples}(T_1, T_2), T'), V : T_1$) : \Rightarrow functions(T_2, T')`
 \rightsquigarrow `function(abstraction(apply(F , tuple(V , given))))`

`partial-apply`(F, V) takes a function F that takes a pair of arguments, and determines the first argument, returning a function of the second argument.