Funcons-beta: Throwing [*]

The PLanCompS Project

`Throwing.cbs` | PLAIN | PRETTY

Throwing

[*Datatype* throwing
Funcon thrown
Funcon finalise-throwing
Funcon throw
Funcon handle-thrown
Funcon handle-recursively
Funcon catch-else-throw]

Meta-variables $R, S, T, T', T'' <:$ values

Datatype throwing $::=$ thrown$(_ :$ values$)$

thrown(V) is a reason for abrupt termination.

Funcon finalise-throwing$(X : \Rightarrow T) : \Rightarrow T \mid$ null-type
$\rightsquigarrow$ finalise-abrupting(X)

finalise-throwing(X) handles abrupt termination of X due to executing throw(V).

Funcon throw$(V : T) : \Rightarrow$ empty-type
$\rightsquigarrow$ abrupt(thrown(V))

throw(V) abruptly terminates all enclosing computations uTil it is handled.

Funcon handle-thrown$(_ : T' \Rightarrow T, _ : T'' \Rightarrow T) : T' \Rightarrow T$

handle-thrown(X, Y) first evaluates X. If X terminates normally with value V, then V is returned and Y is ignored. If X terminates abruptly with a thrown eTity having value V, then Y is executed with V as given value.

handle-thrown(X, Y) is associative, with throw(given) as unit. handle-thrown$(X,$ else$(Y,$ throw(given)$))$ ensures that if Y fails, the thrown value is re-thrown.

[*] Suggestions for improvement: `plancomps@gmail.com`.
Reports of issues: `https://github.com/plancomps/CBS-beta/issues`.

Rule $$\frac{X \xrightarrow{\text{abrupted}(\ )} X'}{\text{handle-thrown}(X, Y) \xrightarrow{\text{abrupted}(\ )} \text{handle-thrown}(X', Y)}$$

Rule $$\frac{X \xrightarrow{\text{abrupted}(\text{thrown}(V'':\text{values}))} X'}{\text{handle-thrown}(X, Y) \xrightarrow{\text{abrupted}(\ )} \text{give}(V'', Y)}$$

Rule $$\frac{X \xrightarrow{\text{abrupted}(V':\sim\text{throwing})} X'}{\text{handle-thrown}(X, Y) \xrightarrow{\text{abrupted}(V')} \text{handle-thrown}(X', Y)}$$

Rule $\text{handle-thrown}(V : T, Y) \rightsquigarrow V$

Funcon $\text{handle-recursively}(X : S \Rightarrow T, Y : R \Rightarrow T) : S \Rightarrow T$
$\rightsquigarrow \text{handle-thrown}(X, \text{else}(\text{handle-recursively}(Y, Y), \text{throw}(\text{given})))$

$\text{handle-recursively}(X, Y)$ behaves similarly to $\text{handle-thrown}(X, Y)$, except that another copy of the handler attempts to handle any values thrown by Y. Thus, many thrown values may get handled by the same handler.

Funcon $\text{catch-else-throw}(P : \text{values}, Y : \Rightarrow T) : \Rightarrow T$
$\rightsquigarrow \text{else}(\text{case-match}(P, Y), \text{throw}(\text{given}))$

$\text{handle-thrown}(X, \text{catch-else-throw}(P, Y))$ handles those values thrown by X that match pattern P. Other thrown values are re-thrown.