

Funcons-beta: Throwing *

The P_{LAN}CompS 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 $R, S, T, T', T'' <: \text{values}$

Datatype **throwing** ::= **thrown**($_ : \text{values}$)

thrown(V) is a reason for abrupt termination.

Funcon **finalise-throwing**($X : \Rightarrow T$) : $\Rightarrow T$ | **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**($X, \text{else}(Y, \text{throw}(\text{given}))$) 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>.

$$\begin{array}{l}
\text{Rule} \quad \frac{X \xrightarrow{\text{abrupted}(\)} X'}{\text{handle-throw}(X, Y) \xrightarrow{\text{abrupted}(\)} \text{handle-throw}(X', Y)} \\
\text{Rule} \quad \frac{X \xrightarrow{\text{abrupted}(\text{thrown}(V'':\text{values}))} X'}{\text{handle-throw}(X, Y) \xrightarrow{\text{abrupted}(\)} \text{give}(V'', Y)} \\
\text{Rule} \quad \frac{X \xrightarrow{\text{abrupted}(V':\sim \text{throwing})} X'}{\text{handle-throw}(X, Y) \xrightarrow{\text{abrupted}(V')} \text{handle-throw}(X', Y)} \\
\text{Rule} \quad \text{handle-throw}(V : T, Y) \rightsquigarrow V
\end{array}$$

$$\begin{array}{l}
\text{Funcon} \quad \text{handle-recursively}(X : S \Rightarrow T, Y : R \Rightarrow T) : S \Rightarrow T \\
\rightsquigarrow \text{handle-throw}(X, \text{else}(\text{handle-recursively}(Y, Y), \text{throw}(\text{given})))
\end{array}$$

`handle-recursively`(X, Y) behaves similarly to `handle-throw`(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.

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

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