1 Arithmetic examples

```
let two = 1 + ( 1 / 1 );;
let oneAndAHalf =
  1.0 +. 1.0 /. ( 1.0 +. 1.0 /. 1.0 );;
let oneAndTwoThirds =
  1.0 +. ( 1.0 /. ( 1.0 +. ( 1.0 /. ( 1.0 +. 1.0 ) ) ) );;
```

2 λ-calculus examples

```
let id = fun x -> x ;;
id( id ) ;;
id( id ) == id ;;
```

3 Slightly more realistic examples

Convert continued fraction list notation to float value

```
let reduceContinuedFraction = fun( elems : int list ) ->
let characteristic : int = List.hd( elems ) in
let mantissa : int list = List.tl( elems ) in
let reducer =
  List.fold_left
    ( fun acc e -> ( fun r -> acc( 1.0 /. ( float( e ) +. r ) ) ) )
    ( fun r -> float( characteristic ) +. r )
    mantissa
in reducer( 0.0 );;
```

4 Delimited continuation examples

```
(*
  Oleg's delimited continuation example from his Delimited Control
  in OCaml paper. This is the final definition of update and
  includes the ``time traveling'' client that rebalaces the tree
  on update.
*)
type '(k, 'v) tree =
    | Empty
    | Node of '(k, 'v) tree * 'k * 'v * '(k, 'v) tree

type '(k',v) res =
    Done of '(k',v) tree
    | ReqNF of 'k * '(v',(k',v) res) subcont

let rec update : '(k',v) res prompt ->'
  k -> '(v'->v) -> '(k',v) tree -> '(k',v) tree =
  fun pnf k f ->
    let rec loop = function
      | Empty ->
          Node(
            Empty,
            k,
            take_subcont pnf (fun c () -> ReqNF (k,c)),
            Empty
          )
      | Node (l,k1,v1,r) ->
          begin
            match compare k k1 with
              | 0 -> Node(l,k1,f v1,r)
              | n when n < 0 -> Node(loop l,k1,v1,r)
              | _            -> Node(l,k1,v1,loop r)
          end
    in loop

let pnf = new_prompt () in
 match push_prompt pnf (fun () -> Done (update pnf 1 succ tree1)) with
   | Done tree    -> tree
   | ReqNF (k,c)  ->
      rebalance (match push_subcont c (fun () -> 100) with Done x -> x)
```

Delimcc implements restartable exceptions with multiple, explicit restarts. The value of the type `subcont` is the restart object, created by `take_subcont` as it raises an exception. Passing the restart object to the function `push_subcont` resumes the interrupted computation. The function `update` not only throws the exception when the key is not found; it also collects the data needed for recovery – the exception object c and the missing key – and packs them into the envelope `ReqNF`.

The function call `push subcont c (fun () -> 100)` resumes the evaluation of `update4` as if the expression `take subcont pnf (...)` returned `100`. We have started with the expression `Done (update4 pnf 1 succ tree1)`,whos eevaluation was interrupted by the exception; `push_prompt` has caught the exception, `yielding ReqNF (k,c)` rather than the value `Done tree` expected as the result of our expression. The restarted expression does not raise any further exceptions, finishing normally, with the result `Done tree`. The result becomes the value yielded by `push_subcont`. (The last `Done x` pattern-match in the sample application is therefore total.)

Upon the exception restart a new node is added to the tree, changing the height of its branch and potentially requiring rebalancing. We may need to rebalance the tree only after the key lookup failure and the addition of a new node. The optimal solution is to proceed upon the assumption of no rebalancing; if we eventually discover that the key was missing and a new node has to be adjoined, we go 'back in time' and add the call to rebalance at the beginning.

5 Concurrency examples

```
(*
  Predicate to check that a string begins with a letter between
  'M' - 'Z' or 'm' - 'z'
*)
let mThruZ =
  fun s -> let fc = Char.code( Char.uppercase( s.[0] ) )
    in ( fc > 76 ) && ( fc < 91 );;

(*
  Create a one time pipe from chan1 to chan2:
   Read user profiles of the form profile( fstName, lstName, data )
   from channel chan1, selecting for those with
   lstName beginning with a letter between 'M' - 'Z' or 'm' - 'z'.
   Publish to channel chan2 records of the form
   candidate( lstName, fstName )
*)
from(
  e <- ( chan1 ? ( profile( fstName, lstName, _ ) ) )
  | mThruZ( e( lstName ) )
) chan2 ! candidate( e( lstName ), e( fstName ) )

(*
  Create a standing pipe from chan1 to chan2.
*)
from(
  e <- ( chan1 ?* ( profile( fstName, lstName, _ ) ) )
  | mThruZ( e( lstName ) )
) chan2 ! candidate( e( lstName ), e( fstName ) )

(*
  Suggest trades between matching bids and asks.
*)
from(
  bid <- ( bids ?* ( bid( bidCommodity, bidPrice, bidContact, _ ) ) );
  ask <- ( asks ?* ( ask( askCommodity, askPrice, askContact, _ ) ) )
  | bid( bidCommodity ) == ask( askCommodity )
    && spread( bid( bidPrice ), ask( askPrice ) )
) let tradeChan = newChan() in
    ( bid( bidContact )
      ! suggest( tradeChan, bid( bidCommodity ), ask( askPrice ) ) );
    ( ask( askContact )
      ! suggest( tradeChan, ask( askCommodity ), bid( bidPrice ) ) )
```

6 Compilation pathway for ordinary code

In symbols, suppose $C : OCamlSource \to ZincByteCode$, and $[\![-]\!]$ is the desugaring map from above then our compiler, $CS : CacaoScriptSource \to ZincByteCode$ is just the composition of C with $[\![-]\!] = C([\![-]\!])$.

$CS($ `let two = 1 + ( 1 / 1 );;` $)$
$= C([\![$ `let two = 1 + ( 1 / 1 );;` $]\!])$
$= C($ `let two = 1 + ( 1 / 1 );;` $)$

```
0       @=0       CONST 1
1       @=1       PUSHCONST 1
2       @=2       DIVINT
3       @=3       PUSHCONST 1
4       @=4       ADDINT
5       @=5       PUSHACC 0
6       @=6       MAKEBLOCK 1 0
7       @=8       POP 1
8       @=10      SETGLOBAL 0
```

$CS($ `let id = fun x -> x in let rslt = id( id ) in id == rslt ;;` $)$
$= C([\![$ `let id = fun x -> x in let rslt = id( id ) in id == rslt ;;` $]\!])$
$= C($ `let id = fun x -> x in let rslt = id( id ) in id == rslt ;;` $)$

```
0       @=0       BRANCH 3

1       @=2       ACC 0
2       @=3       RETURN 1

3       @=5       CLOSURE 0 1
4       @=8       PUSHACC 0
5       @=9       PUSHACC 1
6       @=10      APPLY 1
7       @=11      PUSHACC 0
8       @=12      PUSHACC 2
9       @=13      EQ
10      @=14      POP 2
11      @=16      ATOM 0
12      @=17      SETGLOBAL 0
```