CacaoScript: Syntax and Semantics

A simple language for distributed applications on CacaoWeb

Ivan Chollet,Lucius Gregory Meredith,Paul Steckler

August 12, 2014

ToC

ToC

Syntax

- Core language is a mini-OCaml
- with sugar for
  - Comprehensions
  - Delimited continuations
  - Reflection (quote and unquote)

Core language I

$$
\begin{array}{lcl}
\langle \mathit{Expr} \rangle & ::= & \langle \mathit{Expr} \rangle \ \texttt{;} \ \langle \mathit{Expr1} \rangle \\
& | & \langle \mathit{Expr1} \rangle
\end{array}
$$

$$
\begin{array}{lcl}
\langle \mathit{Expr1} \rangle & ::= & \langle \mathit{Expr1} \rangle \ \langle \mathit{ListExpr2} \rangle \ \texttt{;;} \\
& | & \langle \mathit{Expr2} \rangle
\end{array}
$$

$$
\begin{array}{lcl}
\langle \mathit{Expr2} \rangle & ::= & \texttt{let} \ \langle \mathit{Pattern} \rangle = \langle \mathit{Expr2} \rangle \ \texttt{in} \ \langle \mathit{Expr3} \rangle \\
& | & \texttt{let rec} \ \langle \mathit{Pattern} \rangle = \langle \mathit{Expr2} \rangle \ \texttt{in} \ \langle \mathit{Expr3} \rangle \\
& | & \langle \mathit{Expr3} \rangle
\end{array}
$$

$$
\begin{array}{lcl}
\langle \mathit{Expr3} \rangle & ::= & \texttt{fun} \ \langle \mathit{Pattern} \rangle \ \texttt{->} \ \langle \mathit{Expr4} \rangle \\
& | & \langle \mathit{Expr4} \rangle
\end{array}
$$

Core language II

$$
\begin{array}{rcl}
\langle \mathit{Expr4} \rangle & ::= & \mathtt{if}\ \langle \mathit{Expr4} \rangle\ \mathtt{then}\ \langle \mathit{Expr5} \rangle\ \mathtt{else}\ \langle \mathit{Expr5} \rangle \\
& | & \langle \mathit{Expr5} \rangle = \langle \mathit{Expr5} \rangle \\
& | & \langle \mathit{Expr5} \rangle < \langle \mathit{Expr5} \rangle \\
& | & \langle \mathit{Expr5} \rangle > \langle \mathit{Expr5} \rangle \\
& | & \langle \mathit{Expr5} \rangle <= \langle \mathit{Expr5} \rangle \\
& | & \langle \mathit{Expr5} \rangle >= \langle \mathit{Expr5} \rangle \\
& | & \langle \mathit{Expr5} \rangle
\end{array}
$$

Core language III

$$\langle ArithmeticExpr \rangle \quad ::= \quad \langle ArithmeticExpr \rangle \ \texttt{/} \ \langle ArithmeticExpr1 \rangle$$

$$\langle ArithmeticExpr1 \rangle \quad ::= \quad \langle ArithmeticExpr1 \rangle + \langle ArithmeticExpr2 \rangle$$

$$\langle ArithmeticExpr2 \rangle \quad ::= \quad \langle ArithmeticExpr2 \rangle * \langle ArithmeticExpr3 \rangle$$

$$\langle ArithmeticExpr3 \rangle \quad ::= \quad \langle ArithmeticExpr3 \rangle :: \langle ArithmeticExpr4 \rangle$$

$$\langle ArithmeticExpr4 \rangle \quad ::= \quad - \langle ArithmeticExpr5 \rangle$$

Comprehensions

$\texttt{from}\ (\ \langle \mathit{ListBinding} \rangle\ )\ \texttt{yield}\ \langle \mathit{Expr5} \rangle$
$\texttt{from}\ (\ \langle \mathit{ListBinding} \rangle\ )\ \langle \mathit{Expr5} \rangle$
$\texttt{from}\ (\ \langle \mathit{ListBinding} \rangle \mid \langle \mathit{ListPattern} \rangle\ )\ \texttt{yield}\ \langle \mathit{Expr5} \rangle$
$\texttt{from}\ (\ \langle \mathit{ListBinding} \rangle \mid \langle \mathit{ListPattern} \rangle\ )\ \langle \mathit{Expr5} \rangle$

$$\langle \mathit{Binding} \rangle \quad ::= \quad \langle \mathit{Pattern} \rangle <- \langle \mathit{Expr5} \rangle$$

$$\begin{array}{lcl} \langle \mathit{Pattern} \rangle & ::= & \langle \mathit{Symbol} \rangle\ \texttt{(}\ \langle \mathit{ListPattern} \rangle\ \texttt{)} \\ & \mid & \langle \mathit{Variation} \rangle \\ & \mid & \langle \mathit{Value} \rangle \\ & \mid & \langle \mathit{Lyst} \rangle \end{array}$$

Delimited continuations

$$
\begin{array}{lcl}
\langle Expr5 \rangle & ::= & \texttt{newP} \\
 & | & \texttt{pushP}\ \langle Expr5 \rangle\ \langle Expr5 \rangle \\
 & | & \texttt{takeSC}\ \langle Expr5 \rangle\ \langle Expr5 \rangle \\
 & | & \texttt{pushSC}\ \langle Expr5 \rangle\ \langle Expr5 \rangle
\end{array}
$$

Reflection

ToC

Example programs

- simple arithmetic
- obligatory lambda abstraction application example
- in-place update of a key-value map
- concurrency examples

Sum, products, etc

We can write down any continued fraction such as $P/Q = a + 1/(b + 1/(c + 1/(d + \ldots)))$

```
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 ) ) ) );;
```

$(\lambda x.x)(\lambda x.x)$

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

Continued fractions

```
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 );;
```

in-place update with delimcc

```
(*
  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
```

in-place update with delimcc

```
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
```

in-place update with delimcc

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`.

in-place update with delimcc

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.)

in-place update with delimcc

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.

concurrency

```
(*
  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 );;
```

concurrency

```
(*
  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 ) )
```

concurrency

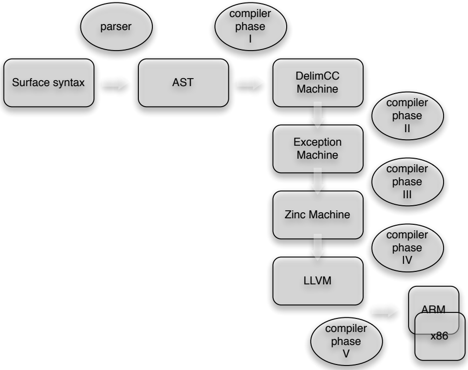
```
(*
  Create a standing pipe from chan1 to chan2.
*)
from(
  e <- ( chan1 ?* ( profile( fstName, lstName, _ ) ) )
  | mThruZ( e( lstName ) )
) chan2 ! candidate( e( lstName ), e( fstName ) )
```

concurrency

```
(*
  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 ) ) )
```

ToC

The pipeline



ToC

The delimCC machine

$$
\begin{array}{llll}
\text{Variables} & x, y, \ldots & \text{Prompts} & p, q \in N \\
\text{Expressions} & e & ::= & v \mid e\,e \mid \texttt{newP} \mid \texttt{pushP}\,e\,e \mid \texttt{takeSC}\,e\,e \mid \texttt{pushSC}\,e\,e \\
\text{Values} & v & ::= & x \mid \lambda x.\,e \mid p \mid D \\
\text{Contexts} & D & ::= & \square \mid De \mid vD \mid \texttt{pushP}\,D\,e \mid \texttt{pushSC}\,D\,e \mid \texttt{takeSC}\,D\,e \\
 & & & \mid \texttt{takeSC}\,p\,D \mid \texttt{pushP}\,p\,D \\
\text{Single Frame} & & ::= & \square e \mid v\square \mid \texttt{pushP}\,\square\,e \mid \texttt{pushSC}\,\square\,e \mid \texttt{takeSC}\,\square\,e \\
 & & & \mid \texttt{takeSC}\,p\,\square \mid \texttt{pushP}\,p\,\square
\end{array}
$$

Transitions between configurations (e, D, q)

$$
\begin{array}{rcll}
(ee', D, q) & \mapsto & (e, D[\square e'], q) & e \text{ non-value} \\
(ve, D, q) & \mapsto & (e, D[v\square], q) & e \text{ non-value} \\
(\texttt{pushP}\,ee', D, q) & \mapsto & (e, D[\texttt{pushP}\,\square e'], q) & e \text{ non-value} \\
(\texttt{takeSC}\,ee', D, q) & \mapsto & (e, D[\texttt{takeSC}\,\square e'], q) & e \text{ non-value} \\
(\texttt{takeSC}\,pe, D, q) & \mapsto & (e, D[\texttt{takeSC}\,p\square], q) & e \text{ non-value} \\
(\texttt{pushSC}\,ee', D, q) & \mapsto & (e, D[\texttt{pushSC}\,\square e'], q) & e \text{ non-value} \\
 & & & \\
((\lambda x.\,e)v, D, q) & \mapsto & (e[v/x], D, q) & \\
(\texttt{newP}, D, q) & \mapsto & (q, D, q+1) & \\
(\texttt{pushP}\,pe, D, q) & \mapsto & (e, D[\texttt{pushP}\,p\square], q) & \\
(\texttt{takeSC}\,pv, D, q) & \mapsto & (vD_1, D_2, q) & D_2[\texttt{pushP}\,pD_1] = D,\ \texttt{pushP}\,pD' \notin D_1 \\
(\texttt{pushSC}\,D'e, D, q) & \mapsto & (e, D[D'], q) & \\
 & & & \\
(v, D[D_1], q) & \mapsto & (D_1[v], D, q) & D_1 \text{ single frame} \\
(\texttt{pushP}\,pv, D, q) & \mapsto & (v, D, q) &
\end{array}
$$

ToC

The exception machine

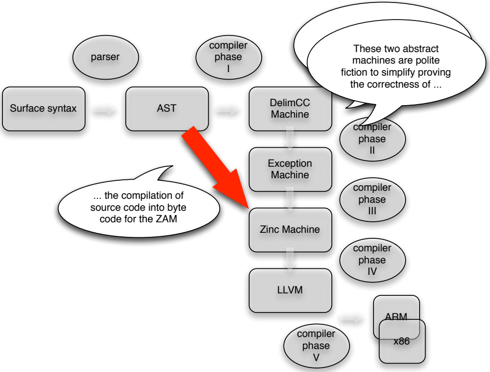
$$
\begin{array}{lllll}
\text{Variables} & x, y, \ldots & & \text{Exceptions} & p, \ldots \\
\text{Expressions} & e ::= v \mid e\,e \mid \mathtt{raise}_p\, e \mid \mathtt{try}_p\, e\, e \\
\text{Values} & v ::= x \mid \lambda x.\, e \\
\text{Contexts} & D ::= \Box \mid De \mid vD \mid \mathtt{raise}_p\, D \mid \mathtt{try}_p\, D\, e \\
\text{Single Frame} & ::= \Box e \mid v\Box \mid \mathtt{raise}_p\, \Box \mid \mathtt{try}_p\, \Box\, e
\end{array}
$$

Transitions between configurations (e, D)

$$
\begin{array}{rcll}
(ee', D) & \mapsto & (e, D[\Box e']) & e \text{ non-value} \\
(ve, D) & \mapsto & (e, D[v\Box]) & e \text{ non-value} \\
(\mathtt{raise}_p\, e, D) & \mapsto & (e, D[\mathtt{raise}_p\, \Box]) & e \text{ non-value} \\
((\lambda x.\, e)v, D) & \mapsto & (e[v/x], D) & \\
(\mathtt{try}_p\, ee', D) & \mapsto & (e, D[\mathtt{try}_p\, \Box e']) & \\
(\mathtt{raise}_p\, v, D) & \mapsto & (e'v, D_2) & D_2[\mathtt{try}_p\, D_1 e'] = D,\ \mathtt{try}_p\, D'e \notin D_1 \\
(v, D[D_1]) & \mapsto & (D_1[v], D) & D_1 \text{ single frame} \\
(\mathtt{try}_p\, ve', D) & \mapsto & (v, D) &
\end{array}
$$

ToC

Eliminating the middle men



Eliminating the delimCC machine

$$
\begin{array}{lcll}
\lfloor \mathtt{takeSC}\, v\, (\lambda\_.\, e) \rfloor & = & \mathtt{raise}_{p_0}(\lambda\_.\, \lfloor e \rfloor, \lfloor v \rfloor) & \\
\lfloor \mathtt{pushP}\, v\, e \rfloor & = & \mathtt{try}_{p_0} \lfloor e \rfloor\ \mathtt{TH}_{\lfloor v \rfloor} & \\
\lfloor x \rfloor & = & x & \\
\lfloor p \rfloor & = & q & \\
\lfloor \lambda x.\, e \rfloor & = & \lambda x.\, \lfloor e \rfloor & \\
\lfloor e_1\, e_2 \rfloor & = & \lfloor e_1 \rfloor\, \lfloor e_2 \rfloor & \\
\lfloor \mathtt{newP} \rfloor & = & \mathtt{newQ} & \\
\lfloor \mathtt{pushP}\, e\, e' \rfloor & = & \lfloor (\lambda x.\, \mathtt{pushP}\, x\, e')e \rfloor & e \text{ non-value, } x \text{ fresh} \\
\lfloor \mathtt{takeSC}\, e\, \lambda\_.\, e' \rfloor & = & \lfloor (\lambda x.\, \mathtt{takeSC}\, x\, \lambda\_.\, e')e \rfloor & e \text{ non-value, } x \text{ fresh}
\end{array}
$$

$$\mathtt{TH}_q = \lambda y.\, \mathtt{if}\ (\lambda y_2.\, (q, y_2))(\mathtt{snd}\, y)\ \mathtt{then}\ \mathtt{fst}\, y\, ()\ \mathtt{else}\ \mathtt{raise}_{p_0}\, y$$

ToC

The Zinc machine

ToC

The LLVM

ToC

Conclusions