CacaoScript: Syntax and Semantics

A simple language for distributed applications on CacaoWeb

Ivan Chollet,Lucius Gregory Meredith,Paul Steckler

August 11, 2014

ToC

ToC

Syntax

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

Core language I

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

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

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

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

Core language II

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

Core language III

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

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

$$\langle ArithmeticExpr2 \rangle \quad ::= \quad \langle ArithmeticExpr2 \rangle \; \texttt{*} \; \langle ArithmeticExpr3 \rangle$$

$$\langle ArithmeticExpr3 \rangle \quad ::= \quad \langle ArithmeticExpr3 \rangle \; \texttt{::} \; \langle ArithmeticExpr4 \rangle$$

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

Comprehensions

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

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

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

Delimited continuations

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

Reflection

ToC

Example programs

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

Sum, products, etc

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

in-place update with delimcc

```
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 update4 : ('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

```
let pnf = new_prompt () in
match push_prompt pnf (fun () -> Done (update4 pnf 1 succ tree1)) with
  | Done tree    -> tree
  | ReqNF (k,c)  ->
      match push_subcont c (fun () -> 100) with Done x -> x
```

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)`, whose evaluation 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

Our sample applications that relied on restartable exceptions had a subtle flaw. Upon the exception restart a new node is added to the tree, changing the height of its branch and potentially requiring rebalancing. We should have written

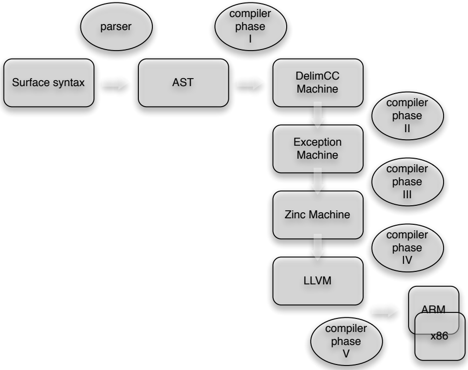
```
let pnf = new_prompt () in
match push_prompt pnf (fun () -> Done (update4 pnf 1 succ tree1)) with
  | Done tree     -> tree
  | ReqNF (k,c)   ->
     rebalance (match push_subcont c (fun () -> 100) with Done x -> x)
```

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

ToC

The pipeline



ToC

The delimCC machine

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

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

Variables	$x, y, \ldots$	Exceptions	$p, \ldots$

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

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 delimCC machine

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

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

ToC

The Zinc machine

ToC

The LLVM

ToC

Conclusions