Overview
Syntax
Example programs
The pipeline
The delimCC machine
The exception machine
Eliminating the middle men
The Zinc machine
LLVM
Conclusions

CacaoScript: Syntax and Semantics

A simple language for distributed applications on CacaoWeb

Ivan Chollet,Lucius Gregory Meredith,Paul Steckler

August 12, 2014

ToC

1. Overview
2. Syntax
   - Core language
   - Sugar
     - Comprehensions
     - Delimited continuations
     - Reflection
3. Example programs
   - The basics: arithmetic
   - The basics: abstraction and application
   - More realistic examples
   - More realistic examples: concurrency
4. The pipeline

ToC

1. Overview
2. Syntax
   - Core language
   - Sugar
     - Comprehensions
     - Delimited continuations
     - Reflection
3. Example programs
   - The basics: arithmetic
   - The basics: abstraction and application
   - More realistic examples
   - More realistic examples: concurrency
4. The pipeline

Syntax

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

Core language I

$$
\begin{array}{lcl}
\langle Expr \rangle & ::= & \langle Expr \rangle \ \texttt{;} \ \langle Expr1 \rangle \\
& | & \langle Expr1 \rangle \\
\langle Expr1 \rangle & ::= & \langle Expr1 \rangle \ \langle ListExpr2 \rangle \ \texttt{;;} \\
& | & \langle Expr2 \rangle \\
\langle Expr2 \rangle & ::= & \texttt{let} \ \langle Pattern \rangle = \langle Expr2 \rangle \ \texttt{in} \ \langle Expr3 \rangle \\
& | & \texttt{let rec} \ \langle Pattern \rangle = \langle Expr2 \rangle \ \texttt{in} \ \langle Expr3 \rangle \\
& | & \langle Expr3 \rangle \\
\langle Expr3 \rangle & ::= & \texttt{fun} \ \langle Pattern \rangle \ \texttt{->} \ \langle Expr4 \rangle \\
& | & \langle Expr4 \rangle
\end{array}
$$

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

$$\langle ArithmeticExpr \rangle ::= \langle ArithmeticExpr \rangle \ / \ \langle ArithmeticExpr1 \rangle$$

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

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

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

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

Comprehensions

`from (` $\langle ListBinding \rangle$ `)` `yield` $\langle Expr5 \rangle$

`from (` $\langle ListBinding \rangle$ `)` $\langle Expr5 \rangle$

`from (` $\langle ListBinding \rangle$ | $\langle ListPattern \rangle$ `)` `yield` $\langle Expr5 \rangle$

`from (` $\langle ListBinding \rangle$ | $\langle ListPattern \rangle$ `)` $\langle Expr5 \rangle$

$$\begin{array}{lcl} \langle Binding \rangle & ::= & \langle Pattern \rangle <- \langle Expr5 \rangle \\ \\ \langle Pattern \rangle & ::= & \langle Symbol \rangle \texttt{ (} \langle ListPattern \rangle \texttt{)} \\ & | & \langle Variation \rangle \\ & | & \langle Value \rangle \\ & | & \langle Lyst \rangle \end{array}$$

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

1. Overview
2. Syntax
   - Core language
   - Sugar
     - Comprehensions
     - Delimited continuations
     - Reflection
3. Example programs
   - The basics: arithmetic
   - The basics: abstraction and application
   - More realistic examples
   - More realistic examples: concurrency
4. The pipeline

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 ) )
```

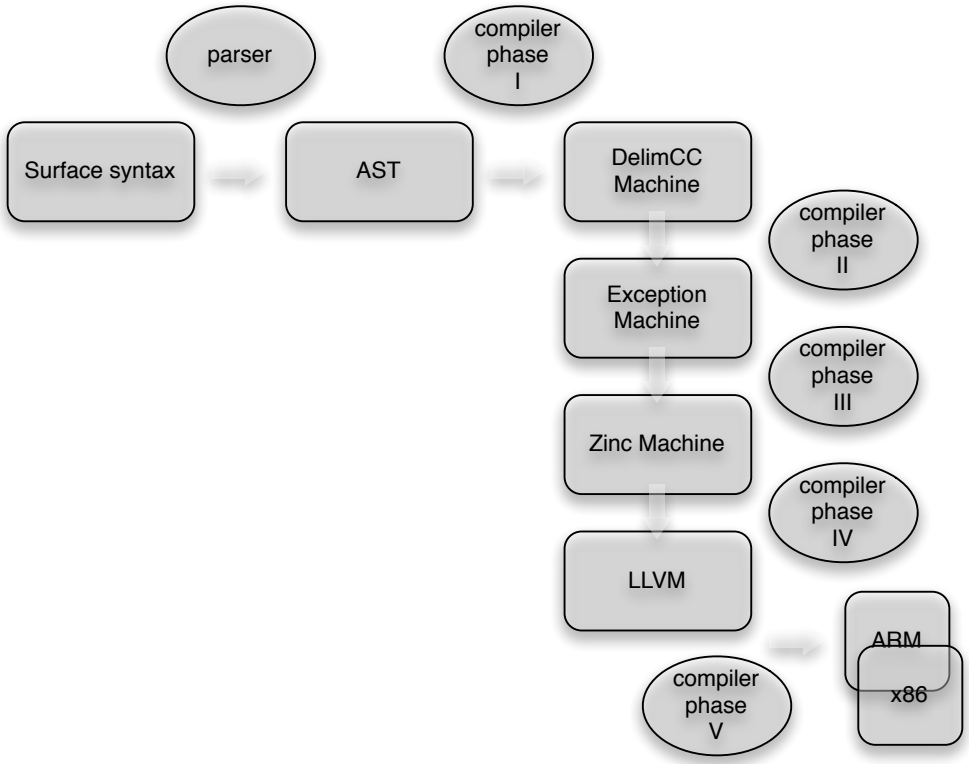
concurrency

```
(*
  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

1. Overview
2. Syntax
   - Core language
   - Sugar
     - Comprehensions
     - Delimited continuations
     - Reflection
3. Example programs
   - The basics: arithmetic
   - The basics: abstraction and application
   - More realistic examples
   - More realistic examples: concurrency
4. The pipeline

The pipeline



ToC

1. Overview
2. Syntax
   - Core language
   - Sugar
     - Comprehensions
     - Delimited continuations
     - Reflection
3. Example programs
   - The basics: arithmetic
   - The basics: abstraction and application
   - More realistic examples
   - More realistic examples: concurrency
4. The pipeline

Overview
Syntax
Example programs
The pipeline
The delimCC machine
The exception machine
Eliminating the middle men
The Zinc machine
LLVM
Conclusions

The delimCC machine

Variables $x, y, \ldots$ Prompts $p, q \in N$

Expressions $e ::= v \mid e\,e \mid \texttt{newP} \mid \texttt{pushP}\,e\,e \mid \texttt{takeSC}\,e\,e \mid \texttt{pushSC}\,e\,e$

Values $v ::= x \mid \lambda x.\,e \mid p \mid D$

Contexts $D ::= \Box \mid De \mid vD \mid \texttt{pushP}\,D\,e \mid \texttt{pushSC}\,D\,e \mid \texttt{takeSC}\,D\,e \mid \texttt{takeSC}\,p\,D \mid \texttt{pushP}\,p\,D$

Single Frame $::= \Box e \mid v\Box \mid \texttt{pushP}\,\Box\,e \mid \texttt{pushSC}\,\Box\,e \mid \texttt{takeSC}\,\Box\,e \mid \texttt{takeSC}\,p\,\Box \mid \texttt{pushP}\,p\,\Box$

Transitions between configurations (e, D, q)

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

ToC

1. Overview
2. Syntax
   - Core language
   - Sugar
     - Comprehensions
     - Delimited continuations
     - Reflection
3. Example programs
   - The basics: arithmetic
   - The basics: abstraction and application
   - More realistic examples
   - More realistic examples: concurrency
4. The pipeline

Overview
Syntax
Example programs
The pipeline
The delimCC machine
The exception machine
Eliminating the middle men
The Zinc machine
LLVM
Conclusions

The exception machine

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

Expressions $e ::= v \mid e\,e \mid \mathtt{raise}_p\, e \mid \mathtt{try}_p\, e\, e$

Values $v ::= x \mid \lambda x.\, e$

Contexts $D ::= \Box \mid De \mid vD \mid \mathtt{raise}_p\, D \mid \mathtt{try}_p\, D\, e$

Single Frame $::= \Box e \mid v\Box \mid \mathtt{raise}_p\, \Box \mid \mathtt{try}_p\, \Box\, e$

Transitions between configurations (e, D)

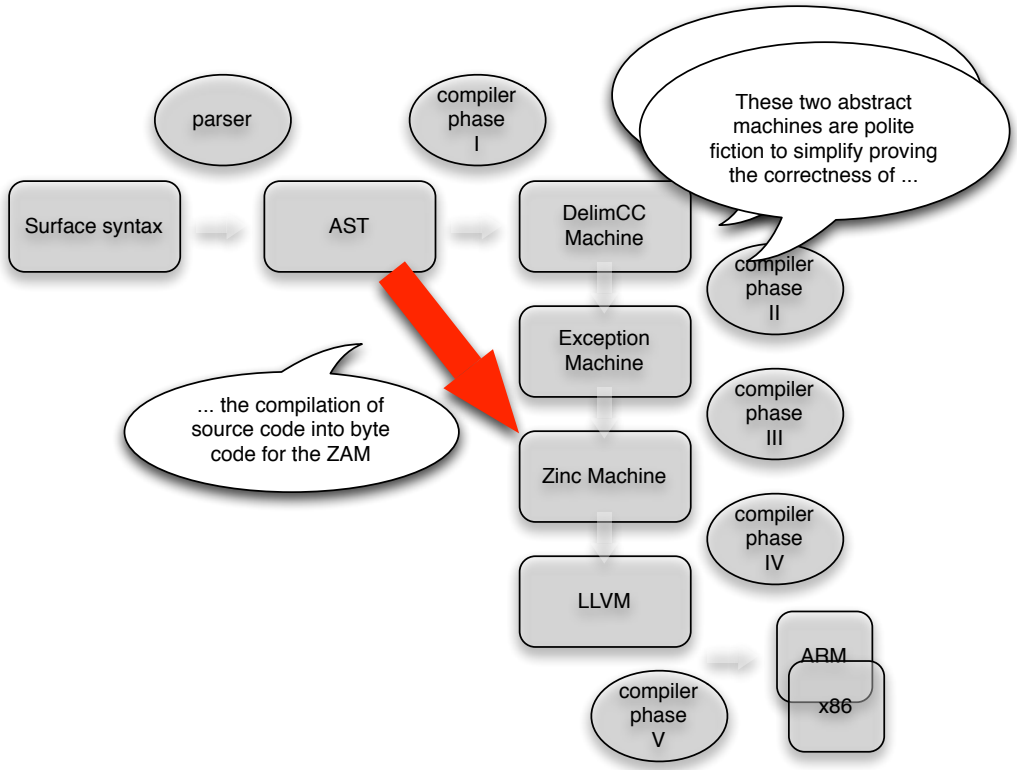
$$
\begin{array}{rcll}
(ee', D) & \mapsto & (e, D[\Box e']) & e \text{ non-value} \\
(ve, D) & \mapsto & (e, D[v\Box]) & e \text{ non-value} \\
(\mathtt{raise}_p\, e, D) & \mapsto & (e, D[\mathtt{raise}_p\, \Box]) & e \text{ non-value} \\
((\lambda x.\, e)v, D) & \mapsto & (e[v/x], D) & \\
(\mathtt{try}_p\,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

ToC

1. Overview
2. Syntax
   - Core language
   - Sugar
     - Comprehensions
     - Delimited continuations
     - Reflection
3. Example programs
   - The basics: arithmetic
   - The basics: abstraction and application
   - More realistic examples
   - More realistic examples: concurrency
4. The pipeline

Overview
Syntax
Example programs
The pipeline
The delimCC machine
The exception machine
Eliminating the middle men
The Zinc machine
LLVM
Conclusions

Eliminating the middle men



Eliminating the delimCC machine

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

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

Overview
Syntax
Example programs
The pipeline
The delimCC machine
The exception machine
Eliminating the middle men
The Zinc machine
LLVM
Conclusions

Eliminating the Exception machine

The Exception machine is really only a simplification of the the Zinc machine. It embeds directly into the Zinc machine. It's sole purpose is to provide an abstract target for the elimination of delimited continuation sugar to any abstract machine that supports exceptions such as the Zinc machine or the JVM or ...

Eliminating the Exception machine

Thus, of our examples, only those that incur a dependency on

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

will have a different compilation path.

Overview
Syntax
Example programs
The pipeline
The delimCC machine
The exception machine
Eliminating the middle men
The Zinc machine
LLVM
Conclusions

Eliminating the middle men

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

Eliminating the Exception machine

To illustrate the point:

$$
\begin{aligned}
& CS(\texttt{ let two = 1 + (1 / 1);; }) \\
= & \, C([\![\texttt{ let two = 1 + (1 / 1);; }]\!]) \\
= & \, C(\texttt{ let two = 1 + (1 / 1);; })
\end{aligned}
$$

ToC

1. Overview
2. Syntax
   - Core language
   - Sugar
     - Comprehensions
     - Delimited continuations
     - Reflection
3. Example programs
   - The basics: arithmetic
   - The basics: abstraction and application
   - More realistic examples
   - More realistic examples: concurrency
4. The pipeline

Overview
Syntax
Example programs
The pipeline
The delimCC machine
The exception machine
Eliminating the middle men
The Zinc machine
LLVM
Conclusions

The Zinc machine

This is well documented.

Compiling our examples to the Zinc machine

$C($ `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
```

Compiling our examples to the Zinc machine

$$
\begin{aligned}
&CS(\texttt{ let id = fun x -> x in let rslt = id(id) in id == rslt ;; }) \\
&= C([\![\texttt{ let id = fun x -> x in let rslt = id(id) in id == rslt ;; }]\!]) \\
&= C(\texttt{ let id = fun x -> x in let rslt = id(id) in id == rslt ;; })
\end{aligned}
$$

```
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
```

ToC

1. Overview
2. Syntax
   - Core language
   - Sugar
     - Comprehensions
     - Delimited continuations
     - Reflection
3. Example programs
   - The basics: arithmetic
   - The basics: abstraction and application
   - More realistic examples
   - More realistic examples: concurrency
4. The pipeline

The LLVM

The purpose of targeting LLVM from the Zinc machine byte code (the subject of various student projects in the OCaml community) is to be able to target a variety of hardware architectures, notably the mobile platforms.

ToC

1. Overview
2. Syntax
   - Core language
   - Sugar
     - Comprehensions
     - Delimited continuations
     - Reflection
3. Example programs
   - The basics: arithmetic
   - The basics: abstraction and application
   - More realistic examples
   - More realistic examples: concurrency
4. The pipeline

Overview
Syntax
Example programs
The pipeline
The delimCC machine
The exception machine
Eliminating the middle men
The Zinc machine
LLVM
Conclusions

Conclusions