

# Lecture 16 – First-Class Continuations

## COSE212: Programming Languages

Jihyeok Park



2023 Fall

- We will learn about **continuations** with the following topics:
  - **Continuations** (Lecture 14 & 15)
  - **First-Class Continuations** (Lecture 16)
  - **Compiling with continuations** (Lecture 17)
- A **continuation** represents the **rest of the computation**.
  - Continuation Passing Style (CPS)
  - Interpreter of FAE in CPS
  - Small-step operational (reduction) semantics of FAE

- We will learn about **continuations** with the following topics:
  - **Continuations** (Lecture 14 & 15)
  - **First-Class Continuations** (Lecture 16)
  - **Compiling with continuations** (Lecture 17)
- A **continuation** represents the **rest of the computation**.
  - Continuation Passing Style (CPS)
  - Interpreter of FAE in CPS
  - Small-step operational (reduction) semantics of FAE
- In this lecture, let's learn **first-class continuations**.

- We will learn about **continuations** with the following topics:
  - **Continuations** (Lecture 14 & 15)
  - **First-Class Continuations** (Lecture 16)
  - **Compiling with continuations** (Lecture 17)
- A **continuation** represents the **rest of the computation**.
  - Continuation Passing Style (CPS)
  - Interpreter of FAE in CPS
  - Small-step operational (reduction) semantics of FAE
- In this lecture, let's learn **first-class continuations**.
- **KFAE** – FAE with **first-class continuations**
  - Interpreter and Reduction semantics

## 1. First-Class Continuations

## 2. KFAE – FAE with First-Class Continuations

Concrete/Abstract Syntax

## 3. Interpreter and Reduction Semantics for KFAE

Recall: Interpreter and Reduction Semantics for FAE

Interpreter and Reduction Semantics for KFAE

First-Class Continuations

Function Application

Example 1

Example 2

## 4. Control Statements

## 1. First-Class Continuations

## 2. KFAE – FAE with First-Class Continuations

Concrete/Abstract Syntax

## 3. Interpreter and Reduction Semantics for KFAE

Recall: Interpreter and Reduction Semantics for FAE

Interpreter and Reduction Semantics for KFAE

First-Class Continuations

Function Application

Example 1

Example 2

## 4. Control Statements

## Recall: First-Class Citizen

In a programming language, an entity is said to be **first-class citizen** if it is treated as a **value**.

In a programming language, an entity is said to be **first-class citizen** if it is treated as a **value**. In other words, it can be

- ① **assigned** to a **variable**,
- ② **passed** as an **argument** to a function, and
- ③ **returned** from a function.



In a programming language, an entity is said to be **first-class citizen** if it is treated as a **value**. In other words, it can be

- ① **assigned** to a **variable**,
- ② **passed** as an **argument** to a function, and
- ③ **returned** from a function.

For example, Scala supports **first-class functions**.

```
def inc(n: Int): Int = n + 1
// 1. We can assign a function to a variable.
val f: Int => Int =
// 2. We can pass a function as an argument to a function.
List(1, 2, 3).map(inc)           // List(2, 3, 4)
// 3. We can return a function from a function.
def addN(n: Int): Int => Int = m => n + m
val add3: Int => Int = addN(3)
add3(5)                         // 3 + 5 = 8
```

Similarly, a **first-class continuation** means that a continuation is treated as a **value**.

Similarly, a **first-class continuation** means that a continuation is treated as a **value**.

For example, **Racket** supports `let/cc` to create a first-class continuation.

```
; prefix notation  
(* 2 (+ 3 5))
```

```
; infix notation  
; 2 * (3 + 5) = 16
```

- ① Evaluate 2. (Result: 2)
- ② Evaluate 3. (Result: 3)
- ③ Evaluate 5. (Result: 5)
- ④ Add the results of step ② and ③. (Result:  $3 + 5 = 8$ )
- ⑤ Multiply the results of step ① and ④. (Result:  $2 * 8 = 16$ )

```
; first-class continuation with `let/cc`  
(* 2 (let/cc k (+ 3 (k 5))))
```

```
; 2 * 5 = 10
```

For example, Racket supports `let/cc` to create a first-class continuation.

```
; first-class continuation with `let/cc`  
(* 2 (let/cc k (+ 3 (k 5))))           ; 2 * 5 = 10
```

- |   |                                  |
|---|----------------------------------|
| ① Evaluate 2.                               | (Result: 2)                      |
| ② Let $k$ be the continuation of ② – ⑥.     | ( $k$ is $x \Rightarrow 2 * x$ ) |
| ③ Evaluate 3.                               | (Result: 3)                      |
| ④ Evaluate 5.                               | (Result: 5)                      |
| ⑤ Apply the result of step ④ to $k$ .       | (Result of ② – ⑥: 5)             |
| ⑥ Add the results of step ③ and ④ – ⑤.      | (Unreachable)                    |
| ⑦ Multiply the results of step ① and ② – ⑥. | (Result: $2 * 5 = 10$ )          |

For example, Racket supports `let/cc` to create a first-class continuation.

```
; first-class continuation with `let/cc`  
(* 2 (let/cc k (+ 3 (k 5))))           ; 2 * 5 = 10
```

- ① Evaluate 2. (Result: 2)
  - ② Let  $k$  be the continuation of ② – ⑥. ( $k$  is  $x \Rightarrow 2 * x$ )
  - ③ Evaluate 3. (Result: 3)
  - ④ Evaluate 5. (Result: 5)
  - ⑤ Apply the result of step ④ to  $k$ . (Result of ② – ⑥: 5)
  - ⑥ Add the results of step ③ and ④ – ⑤. (Unreachable)
  - ⑦ Multiply the results of step ① and ② – ⑥. (Result:  $2 * 5 = 10$ )
- The step ② defines the continuation of ② – ⑥ as a value in  $k$ .
  - The step ⑤ changes the control flow using  $k$  with the result of ④.

Some functional languages support **first-class continuations**.

- Rakcet

```
(* 2 (let/cc k (+ 3 (k 5))))           ; 2 * 5 = 10
```

- Ruby

```
2 * (callcc { |k| 3 + k.call(5)})     # 2 * 5 = 10
```

- Haskell

```
do
  x <- callCC $ \k -> do
    y <- k 5
    return $ 3 + y
  return $ 2 * x                       -- 2 * 5 = 10
```

- ...

## 1. First-Class Continuations

## 2. KFAE – FAE with First-Class Continuations

### Concrete/Abstract Syntax

## 3. Interpreter and Reduction Semantics for KFAE

Recall: Interpreter and Reduction Semantics for FAE

Interpreter and Reduction Semantics for KFAE

First-Class Continuations

Function Application

Example 1

Example 2

## 4. Control Statements

Now, let's extend FAE into KFAE to support **first-class continuations**.  
(Assume that `val` is supported in FAE as syntactic sugar.)

```
/* KFAE */  
2 * { vcc k; 3 + k(5) } // 2 * 5 = 10
```



Now, let's extend FAE into KFAE to support **first-class continuations**.  
(Assume that `val` is supported in FAE as syntactic sugar.)

```
/* KFAE */  
2 * { vcc k; 3 + k(5) } // 2 * 5 = 10
```

```
/* KFAE */  
{  
  vcc done;           // done = x => x  
  val f = {  
    vcc exit;         // exit = x => val f = x; f(3) * 5  
    2 * done(1 + {  
      vcc k;          // k      = x => val f = { 2 * done(1 + x) }; f(3) * 5  
      exit(k)  
    })  
  };  
  f(3) * 5  
}  
// 1 + 3
```

For KFAE, we need to extend **expressions** of FAE with

- ① **first-class continuations** (`vcc`)

For KFAE, we need to extend **expressions** of FAE with

① **first-class continuations** (`vcc`)

We can extend the **concrete syntax** of FAE as follows:

```
// expressions  
<expr> ::= ... | "vcc" <id> ";" <expr>
```

For KFAE, we need to extend **expressions** of FAE with

## ① first-class continuations (`vcc`)

We can extend the **concrete syntax** of FAE as follows:

```
// expressions  
<expr> ::= ... | "vcc" <id> ";" <expr>
```

and the **abstract syntax** of FAE as follows:

Expressions  $\mathbb{E} \ni e ::= \dots \mid \text{vcc } x; e \quad (\text{Vcc})$

```
enum Expr:  
  ...  
  // first-class continuations  
  case Vcc(name: String, body: Expr)
```

## 1. First-Class Continuations

## 2. KFAE – FAE with First-Class Continuations

Concrete/Abstract Syntax

## 3. Interpreter and Reduction Semantics for KFAE

Recall: Interpreter and Reduction Semantics for FAE

Interpreter and Reduction Semantics for KFAE

First-Class Continuations

Function Application

Example 1

Example 2

## 4. Control Statements

In the previous lecture, we have defined the **first-order representation** of **continuations** with **value stack**:

```
enum Cont:
  case EmptyK
  case EvalK(env: Env, expr: Expr, k: Cont)
  case AddK(k: Cont)
  case MulK(k: Cont)
  case AppK(k: Cont)

type Stack = List[Value]
```

Continuations	$\mathbb{K} \ni \kappa ::= \square$	(EmptyK)
	$(\sigma \vdash e) :: \kappa$	(EvalK)
	$(+)$ :: $\kappa$	(AddK)
	$(\times)$ :: $\kappa$	(MulK)
	$(@)$ :: $\kappa$	(AppK)
Value Stacks	$\mathbb{S} \ni s ::= \blacksquare$   $v :: s$	(List[Value])

Then, we have defined the **reduction relation**  $\rightarrow \in (\mathbb{K} \times \mathbb{S}) \times (\mathbb{K} \times \mathbb{S})$  between **states** consisting of pairs of **continuations** and **value stacks**:

```
def reduce(k: Cont, s: Stack): (Cont, Stack) = ???
```

$$\langle \kappa \parallel s \rangle \rightarrow \langle \kappa' \parallel s' \rangle$$

Then, we have defined the **reduction relation**  $\rightarrow \in (\mathbb{K} \times \mathbb{S}) \times (\mathbb{K} \times \mathbb{S})$  between **states** consisting of pairs of **continuations** and **value stacks**:

```
def reduce(k: Cont, s: Stack): (Cont, Stack) = ???
```

$$\langle \kappa \parallel s \rangle \rightarrow \langle \kappa' \parallel s' \rangle$$

And the eval function **iteratively reduces** the state until it reaches the empty continuation  $\square$  and returns the single value in the value stack:

```
def eval(str: String): String =
  import Cont.*
  def aux(k: Cont, s: Stack): Value = reduce(k, s) match
    case (EmptyK, List(v)) => v
    case (k, s) => aux(k, s)
  aux(EvalK(Map.empty, Expr(str), EmptyK), List.empty).str
```

$$\langle (\emptyset \vdash e) :: \square \parallel \blacksquare \rangle \rightarrow^* \langle \square \parallel v :: \blacksquare \rangle$$



Now, let's extend the interpreter and reduction semantics for FAE to KFAE by adding the **first-class continuations**.

First, we need to extend the values of FAE with **continuation values** consisting of pairs of continuations and value stacks:

```
// values
enum Value:
  case NumV(number: BigInt)
  case CloV(param: String, body: Expr, env: Env)
  case ContV(cont: Cont, stack: Stack)
```

$$\begin{array}{ll} \text{Values } \mathbb{V} \ni v ::= n & (\text{NumV}) \\ & | \langle \lambda x.e, \sigma \rangle \quad (\text{CloV}) \\ & | \langle \kappa \parallel s \rangle \quad (\text{ContV}) \end{array}$$

Then, let's fill out the missing cases in the reduce function and reduction rules for  $\rightarrow$  in the reduction semantics of KFAE.

```
def reduce(k: Cont, s: Stack): (Cont, Stack) = (k, s) match
  case (EvalK(env, expr, k), s) => expr match
    ...
    case Vcc(x, b) => (EvalK(env + (x -> ContV(k, s)), b, k), s)
```

$$\langle \kappa \parallel s \rangle \rightarrow \langle \kappa \parallel s \rangle$$

$$\text{Vcc} \quad \langle (\sigma \vdash \text{vcc } x; e) :: \kappa \parallel s \rangle \rightarrow \langle (\sigma[x \mapsto \langle \kappa \parallel s \rangle] \vdash e) :: \kappa \parallel s \rangle$$

It defines a new immutable binding  $x$  in the environment  $\sigma$  that maps to a **continuation value**  $\langle \kappa \parallel s \rangle$ , and then evaluates the body expression  $e$  in the extended environment  $\sigma[x \mapsto \langle \kappa \parallel s \rangle]$ .

```
def reduce(k: Cont, s: Stack): (Cont, Stack) = (k, s) match
  case (EvalK(env, expr, k), s) => expr match
    ...
    case App(f, e) => (EvalK(env, f, EvalK(env, e, AppK(k))), s)
  ...
  case (AppK(k), a :: f :: s) => f match
    case CloV(p, b, fenv) => (EvalK(fenv + (p -> a), b, k), s)
    case ContV(k1, s1)    => (k1, a :: s1)
    case v                => error(s"not a function: ${v.str}")
```

$$\langle \kappa \parallel s \rangle \rightarrow \langle \kappa \parallel s \rangle$$

$\text{App}_1 \quad \langle (\sigma \vdash e_1(e_2)) :: \kappa \parallel s \rangle \rightarrow \langle (\sigma \vdash e_1) :: (\sigma \vdash e_2) :: (@) :: \kappa \parallel s \rangle$   
 $\text{App}_{2,\lambda} \quad \langle (@) :: \kappa \parallel v_2 :: \langle \lambda x.e, \sigma \rangle :: s \rangle \rightarrow \langle (\sigma[x \mapsto v_2] \vdash e) :: \kappa \parallel s \rangle$   
 $\text{App}_{2,\kappa} \quad \langle (@) :: \kappa \parallel v_2 :: \langle \kappa' \parallel s' \rangle :: s \rangle \rightarrow \langle \kappa' \parallel v_2 :: s' \rangle$

The new  $\text{App}_{2,\kappa}$  rule handles when the function expression evaluates to a continuation value  $\langle \kappa' \parallel s' \rangle$ . It changes the control flow to the continuation  $\kappa'$  with the given argument value  $v_2$  and the value stack  $s'$ .

## Example 1

Let's interpret the expression  $2 \times (\text{vcc } k; (3 + k(5)))$ :

## Example 1

Let's interpret the expression  $2 \times (\text{vcc } k; (3 + k(5)))$ :

$\langle (\emptyset \vdash 2 \times (\text{vcc } k; (3 + k(5)))) :: \square \quad \parallel \blacksquare \quad \rangle$

# Example 1

Let's interpret the expression  $2 \times (\text{vcc } k; (3 + k(5)))$ :

$$\begin{array}{l} \text{(Mul}_1\text{)} \quad \langle (\emptyset \vdash 2 \times (\text{vcc } k; (3 + k(5)))) \rangle :: \square \quad \parallel \blacksquare \quad \rangle \\ \rightarrow \quad \langle (\emptyset \vdash 2) \rangle :: \langle (\emptyset \vdash (\text{vcc } k; (3 + k(5)))) \rangle :: (\times) :: \square \quad \parallel \blacksquare \quad \rangle \end{array}$$

# Example 1

Let's interpret the expression  $2 \times (\text{vcc } k; (3 + k(5)))$ :

$$\begin{array}{llll} \text{(Mul}_1\text{)} & \langle (\emptyset \vdash 2 \times (\text{vcc } k; (3 + k(5)))) \rangle :: \square & || \blacksquare & \rangle \\ \rightarrow & \langle (\emptyset \vdash 2) :: (\emptyset \vdash (\text{vcc } k; (3 + k(5)))) \rangle :: (\times) :: \square & || \blacksquare & \rangle \\ \text{(Num)} & & & \\ \rightarrow & \langle (\emptyset \vdash (\text{vcc } k; (3 + k(5)))) \rangle :: (\times) :: \square & || 2 :: \blacksquare & \rangle \end{array}$$

# Example 1

Let's interpret the expression  $2 \times (\text{vcc } k; (3 + k(5)))$ :

$$\begin{array}{llll} \text{(Mul}_1\text{)} & \langle (\emptyset \vdash 2 \times (\text{vcc } k; (3 + k(5)))) \rangle :: \square & || \blacksquare & \rangle \\ \rightarrow & \langle (\emptyset \vdash 2) :: (\emptyset \vdash (\text{vcc } k; (3 + k(5)))) \rangle :: (\times) :: \square & || \blacksquare & \rangle \\ \text{(Num)} & \rightarrow & & \\ \rightarrow & \langle (\emptyset \vdash (\text{vcc } k; (3 + k(5)))) \rangle :: (\times) :: \square & || 2 :: \blacksquare & \rangle \\ \text{(Vcc)} & \rightarrow & & \\ \rightarrow & \langle (\sigma_0 \vdash (3 + k(5))) \rangle :: (\times) :: \square & || 2 :: \blacksquare & \rangle \end{array}$$

$$\text{where } \begin{cases} \sigma_0 = [k \mapsto \langle \kappa_0 \mid s_0 \rangle] \\ \kappa_0 = (\times) :: \square \\ s_0 = 2 :: \blacksquare \end{cases}$$



# Example 1

Let's interpret the expression  $2 \times (\text{vcc } k; (3 + k(5)))$ :

$(\text{Mul}_1)$	$\langle (\emptyset \vdash 2 \times (\text{vcc } k; (3 + k(5)))) :: \square$	$\parallel \blacksquare$	$\rangle$
$\rightarrow$	$\langle (\emptyset \vdash 2) :: (\emptyset \vdash (\text{vcc } k; (3 + k(5)))) :: (\times) :: \square$	$\parallel \blacksquare$	$\rangle$
$(\text{Num})$	$\langle (\emptyset \vdash 2) :: (\emptyset \vdash (\text{vcc } k; (3 + k(5)))) :: (\times) :: \square$	$\parallel 2 :: \blacksquare$	$\rangle$
$\rightarrow$	$\langle (\emptyset \vdash (\text{vcc } k; (3 + k(5)))) :: (\times) :: \square$	$\parallel 2 :: \blacksquare$	$\rangle$
$(\text{Vcc})$	$\langle (\sigma_0 \vdash (3 + k(5))) :: (\times) :: \square$	$\parallel 2 :: \blacksquare$	$\rangle$
$\rightarrow$	$\langle (\sigma_0 \vdash (3 + k(5))) :: (\times) :: \square$	$\parallel 2 :: \blacksquare$	$\rangle$
$(\text{Add}_1)$	$\langle (\sigma_0 \vdash 3) :: (\sigma_0 \vdash k(5)) :: (+) :: (\times) :: \square$	$\parallel 2 :: \blacksquare$	$\rangle$
$\rightarrow$	$\langle (\sigma_0 \vdash 3) :: (\sigma_0 \vdash k(5)) :: (+) :: (\times) :: \square$	$\parallel 2 :: \blacksquare$	$\rangle$

$$\text{where } \begin{cases} \sigma_0 = [k \mapsto \langle \kappa_0 \parallel s_0 \rangle] \\ \kappa_0 = (\times) :: \square \\ s_0 = 2 :: \blacksquare \end{cases}$$

# Example 1

Let's interpret the expression  $2 \times (\text{vcc } k; (3 + k(5)))$ :

$(\text{Mul}_1)$	$\langle (\emptyset \vdash 2 \times (\text{vcc } k; (3 + k(5)))) \rangle :: \square$	$\parallel \blacksquare$	$\rangle$
$\rightarrow$	$\langle (\emptyset \vdash 2) :: (\emptyset \vdash (\text{vcc } k; (3 + k(5)))) \rangle :: (\times) :: \square$	$\parallel \blacksquare$	$\rangle$
$(\text{Num})$	$\langle (\emptyset \vdash 2) \rangle :: (\times) :: \square$	$\parallel 2 :: \blacksquare$	$\rangle$
$(\text{Vcc})$	$\langle (\sigma_0 \vdash (3 + k(5))) \rangle :: (\times) :: \square$	$\parallel 2 :: \blacksquare$	$\rangle$
$\rightarrow$	$\langle (\sigma_0 \vdash 3) :: (\sigma_0 \vdash k(5)) \rangle :: (+) :: (\times) :: \square$	$\parallel 2 :: \blacksquare$	$\rangle$
$(\text{Add}_1)$	$\langle (\sigma_0 \vdash 3) \rangle :: (+) :: (\times) :: \square$	$\parallel 3 :: 2 :: \blacksquare$	$\rangle$
$\rightarrow$	$\langle (\sigma_0 \vdash k(5)) \rangle :: (+) :: (\times) :: \square$		
$(\text{Num})$			

$$\text{where } \begin{cases} \sigma_0 = [k \mapsto \langle \kappa_0 \parallel s_0 \rangle] \\ \kappa_0 = (\times) :: \square \\ s_0 = 2 :: \blacksquare \end{cases}$$

# Example 1

Let's interpret the expression  $2 \times (\text{vcc } k; (3 + k(5)))$ :

$(\text{Mul}_1)$	$\langle (\emptyset \vdash 2 \times (\text{vcc } k; (3 + k(5)))) \rangle :: \square$	$\parallel \blacksquare$	$\rangle$
$\rightarrow$	$\langle (\emptyset \vdash 2) :: (\emptyset \vdash (\text{vcc } k; (3 + k(5)))) \rangle :: (\times) :: \square$	$\parallel \blacksquare$	$\rangle$
$(\text{Num})$	$\langle (\emptyset \vdash (\text{vcc } k; (3 + k(5)))) \rangle :: (\times) :: \square$	$\parallel 2 :: \blacksquare$	$\rangle$
$(\text{Vcc})$	$\langle (\sigma_0 \vdash (3 + k(5))) \rangle :: (\times) :: \square$	$\parallel 2 :: \blacksquare$	$\rangle$
$\rightarrow$	$\langle (\sigma_0 \vdash 3) :: (\sigma_0 \vdash k(5)) \rangle :: (+) :: (\times) :: \square$	$\parallel 2 :: \blacksquare$	$\rangle$
$(\text{Add}_1)$	$\langle (\sigma_0 \vdash k(5)) \rangle :: (+) :: (\times) :: \square$	$\parallel 3 :: 2 :: \blacksquare$	$\rangle$
$\rightarrow$	$\langle (\sigma_0 \vdash k) :: (\sigma_0 \vdash 5) \rangle :: (@) :: (+) :: (\times) :: \square$	$\parallel 3 :: 2 :: \blacksquare$	$\rangle$
$(\text{App}_1)$			

$$\text{where } \begin{cases} \sigma_0 = [k \mapsto \langle \kappa_0 \parallel s_0 \rangle] \\ \kappa_0 = (\times) :: \square \\ s_0 = 2 :: \blacksquare \end{cases}$$

# Example 1

Let's interpret the expression  $2 \times (\text{vcc } k; (3 + k(5)))$ :

	$\langle (\emptyset \vdash 2 \times (\text{vcc } k; (3 + k(5)))) :: \square \rangle$	$\parallel \blacksquare$	$\rangle$
$(\text{Mul}_1)$	$\rightarrow \langle (\emptyset \vdash 2) :: (\emptyset \vdash (\text{vcc } k; (3 + k(5)))) :: (\times) :: \square \rangle$	$\parallel \blacksquare$	$\rangle$
$(\text{Num})$	$\rightarrow \langle (\emptyset \vdash (\text{vcc } k; (3 + k(5)))) :: (\times) :: \square \rangle$	$\parallel 2 :: \blacksquare$	$\rangle$
$(\text{Vcc})$	$\rightarrow \langle (\sigma_0 \vdash (3 + k(5))) :: (\times) :: \square \rangle$	$\parallel 2 :: \blacksquare$	$\rangle$
$(\text{Add}_1)$	$\rightarrow \langle (\sigma_0 \vdash 3) :: (\sigma_0 \vdash k(5)) :: (+) :: (\times) :: \square \rangle$	$\parallel 2 :: \blacksquare$	$\rangle$
$(\text{Num})$	$\rightarrow \langle (\sigma_0 \vdash k(5)) :: (+) :: (\times) :: \square \rangle$	$\parallel 3 :: 2 :: \blacksquare$	$\rangle$
$(\text{App}_1)$	$\rightarrow \langle (\sigma_0 \vdash k) :: (\sigma_0 \vdash 5) :: (@) :: (+) :: (\times) :: \square \rangle$	$\parallel 3 :: 2 :: \blacksquare$	$\rangle$
$(\text{Id})$	$\rightarrow \langle (\sigma_0 \vdash 5) :: (@) :: (+) :: (\times) :: \square \rangle$	$\parallel \langle \kappa_0 \parallel s_1 \rangle :: 3 :: 2 :: \blacksquare$	$\rangle$

$$\text{where } \begin{cases} \sigma_0 = [k \mapsto \langle \kappa_0 \parallel s_0 \rangle] \\ \kappa_0 = (\times) :: \square \\ s_0 = 2 :: \blacksquare \end{cases}$$

# Example 1

Let's interpret the expression  $2 \times (\text{vcc } k; (3 + k(5)))$ :

	$\langle (\emptyset \vdash 2 \times (\text{vcc } k; (3 + k(5)))) :: \square \rangle$	$\parallel \blacksquare$	$\rangle$
$(\text{Mul}_1)$	$\rightarrow \langle (\emptyset \vdash 2) :: (\emptyset \vdash (\text{vcc } k; (3 + k(5)))) :: (\times) :: \square \rangle$	$\parallel \blacksquare$	$\rangle$
$(\text{Num})$	$\rightarrow \langle (\emptyset \vdash (\text{vcc } k; (3 + k(5)))) :: (\times) :: \square \rangle$	$\parallel 2 :: \blacksquare$	$\rangle$
$(\text{Vcc})$	$\rightarrow \langle (\sigma_0 \vdash (3 + k(5))) :: (\times) :: \square \rangle$	$\parallel 2 :: \blacksquare$	$\rangle$
$(\text{Add}_1)$	$\rightarrow \langle (\sigma_0 \vdash 3) :: (\sigma_0 \vdash k(5)) :: (+) :: (\times) :: \square \rangle$	$\parallel 2 :: \blacksquare$	$\rangle$
$(\text{Num})$	$\rightarrow \langle (\sigma_0 \vdash k(5)) :: (+) :: (\times) :: \square \rangle$	$\parallel 3 :: 2 :: \blacksquare$	$\rangle$
$(\text{App}_1)$	$\rightarrow \langle (\sigma_0 \vdash k) :: (\sigma_0 \vdash 5) :: (@) :: (+) :: (\times) :: \square \rangle$	$\parallel 3 :: 2 :: \blacksquare$	$\rangle$
$(\text{Id})$	$\rightarrow \langle (\sigma_0 \vdash 5) :: (@) :: (+) :: (\times) :: \square \rangle$	$\parallel \langle \kappa_0 \parallel s_1 \rangle :: 3 :: 2 :: \blacksquare$	$\rangle$
$(\text{Num})$	$\rightarrow \langle (@) :: (+) :: (\times) :: \square \rangle$	$\parallel 5 :: \langle \kappa_0 \parallel s_1 \rangle :: 3 :: 2 :: \blacksquare$	$\rangle$

$$\text{where } \begin{cases} \sigma_0 = [k \mapsto \langle \kappa_0 \parallel s_0 \rangle] \\ \kappa_0 = (\times) :: \square \\ s_0 = 2 :: \blacksquare \end{cases}$$

# Example 1

Let's interpret the expression  $2 \times (\text{vcc } k; (3 + k(5)))$ :

	$\langle (\emptyset \vdash 2 \times (\text{vcc } k; (3 + k(5)))) :: \square \rangle$	$\parallel \blacksquare$	$\rangle$
$(\text{Mul}_1)$	$\rightarrow \langle (\emptyset \vdash 2) :: (\emptyset \vdash (\text{vcc } k; (3 + k(5)))) :: (\times) :: \square \rangle$	$\parallel \blacksquare$	$\rangle$
$(\text{Num})$	$\rightarrow \langle (\emptyset \vdash (\text{vcc } k; (3 + k(5)))) :: (\times) :: \square \rangle$	$\parallel 2 :: \blacksquare$	$\rangle$
$(\text{Vcc})$	$\rightarrow \langle (\sigma_0 \vdash (3 + k(5))) :: (\times) :: \square \rangle$	$\parallel 2 :: \blacksquare$	$\rangle$
$(\text{Add}_1)$	$\rightarrow \langle (\sigma_0 \vdash 3) :: (\sigma_0 \vdash k(5)) :: (+) :: (\times) :: \square \rangle$	$\parallel 2 :: \blacksquare$	$\rangle$
$(\text{Num})$	$\rightarrow \langle (\sigma_0 \vdash k(5)) :: (+) :: (\times) :: \square \rangle$	$\parallel 3 :: 2 :: \blacksquare$	$\rangle$
$(\text{App}_1)$	$\rightarrow \langle (\sigma_0 \vdash k) :: (\sigma_0 \vdash 5) :: (@) :: (+) :: (\times) :: \square \rangle$	$\parallel 3 :: 2 :: \blacksquare$	$\rangle$
$(\text{Id})$	$\rightarrow \langle (\sigma_0 \vdash 5) :: (@) :: (+) :: (\times) :: \square \rangle$	$\parallel \langle \kappa_0 \parallel s_1 \rangle :: 3 :: 2 :: \blacksquare$	$\rangle$
$(\text{Num})$	$\rightarrow \langle (@) :: (+) :: (\times) :: \square \rangle$	$\parallel 5 :: \langle \kappa_0 \parallel s_1 \rangle :: 3 :: 2 :: \blacksquare$	$\rangle$
$(\text{App}_{2,\kappa})$	$\rightarrow \langle (\times) :: \square \rangle$	$\parallel 5 :: 2 :: \blacksquare$	$\rangle$

$$\text{where } \begin{cases} \sigma_0 = [k \mapsto \langle \kappa_0 \parallel s_0 \rangle] \\ \kappa_0 = (\times) :: \square \\ s_0 = 2 :: \blacksquare \end{cases}$$

# Example 1

Let's interpret the expression  $2 \times (\text{vcc } k; (3 + k(5)))$ :

	$\langle (\emptyset \vdash 2 \times (\text{vcc } k; (3 + k(5)))) :: \square \rangle$	$\parallel \blacksquare$	$\rangle$
$(\text{Mul}_1)$	$\rightarrow \langle (\emptyset \vdash 2) :: (\emptyset \vdash (\text{vcc } k; (3 + k(5)))) :: (\times) :: \square \rangle$	$\parallel \blacksquare$	$\rangle$
$(\text{Num})$	$\rightarrow \langle (\emptyset \vdash (\text{vcc } k; (3 + k(5)))) :: (\times) :: \square \rangle$	$\parallel 2 :: \blacksquare$	$\rangle$
$(\text{Vcc})$	$\rightarrow \langle (\sigma_0 \vdash (3 + k(5))) :: (\times) :: \square \rangle$	$\parallel 2 :: \blacksquare$	$\rangle$
$(\text{Add}_1)$	$\rightarrow \langle (\sigma_0 \vdash 3) :: (\sigma_0 \vdash k(5)) :: (+) :: (\times) :: \square \rangle$	$\parallel 2 :: \blacksquare$	$\rangle$
$(\text{Num})$	$\rightarrow \langle (\sigma_0 \vdash k(5)) :: (+) :: (\times) :: \square \rangle$	$\parallel 3 :: 2 :: \blacksquare$	$\rangle$
$(\text{App}_1)$	$\rightarrow \langle (\sigma_0 \vdash k) :: (\sigma_0 \vdash 5) :: (@) :: (+) :: (\times) :: \square \rangle$	$\parallel 3 :: 2 :: \blacksquare$	$\rangle$
$(\text{Id})$	$\rightarrow \langle (\sigma_0 \vdash 5) :: (@) :: (+) :: (\times) :: \square \rangle$	$\parallel \langle \kappa_0 \parallel s_1 \rangle :: 3 :: 2 :: \blacksquare$	$\rangle$
$(\text{Num})$	$\rightarrow \langle (@) :: (+) :: (\times) :: \square \rangle$	$\parallel 5 :: \langle \kappa_0 \parallel s_1 \rangle :: 3 :: 2 :: \blacksquare$	$\rangle$
$(\text{App}_{2,\kappa})$	$\rightarrow \langle (\times) :: \square \rangle$	$\parallel 5 :: 2 :: \blacksquare$	$\rangle$
$(\text{Mul}_2)$	$\rightarrow \langle \square \rangle$	$\parallel 10 :: \blacksquare$	$\rangle$

$$\text{where } \begin{cases} \sigma_0 = [k \mapsto \langle \kappa_0 \parallel s_0 \rangle] \\ \kappa_0 = (\times) :: \square \\ s_0 = 2 :: \blacksquare \end{cases}$$

## Example 2

Let's interpret the expression  $(\lambda x. (\text{vcc } r; r(x + 1) \times 2))(3)$ :



## Example 2

Let's interpret the expression  $(\lambda x. (\text{vcc } r; r(x + 1) \times 2))(3)$ :

$\langle (\emptyset \vdash (\lambda x. (\text{vcc } r; r(x + 1) \times 2))(3)) :: \square \quad \quad \quad || \blacksquare \quad \quad \quad \rangle$

where {

## Example 2

Let's interpret the expression  $(\lambda x.(\text{vcc } r; r(x + 1) \times 2))(3)$ :

$$\begin{array}{lcl} (\text{App}_1) & \langle (\emptyset \vdash (\lambda x.(\text{vcc } r; r(x + 1) \times 2))(3)) :: \square & \parallel \blacksquare \rangle \\ \xrightarrow{\quad} & \langle (\emptyset \vdash (\lambda x.(\text{vcc } r; r(x + 1) \times 2))) :: (\emptyset \vdash 3) :: (@) :: \square & \parallel \blacksquare \rangle \end{array}$$

where {

## Example 2

Let's interpret the expression  $(\lambda x.(\text{vcc } r; r(x + 1) \times 2))(3)$ :

$$\begin{array}{lll}
 (\text{App}_1) & \langle (\emptyset \vdash (\lambda x.(\text{vcc } r; r(x + 1) \times 2))(3)) :: \square & || \blacksquare \rangle \\
 \rightarrow & \langle (\emptyset \vdash (\lambda x.(\text{vcc } r; r(x + 1) \times 2))) :: (\emptyset \vdash 3) :: (@) :: \square & || \blacksquare \rangle \\
 (\text{Fun}) & \langle (\emptyset \vdash 3) :: (@) :: \square & || \langle \lambda x.e_0, \emptyset \rangle :: \blacksquare \rangle \\
 \rightarrow & & 
 \end{array}$$

$$\text{where } \left\{ \begin{array}{l} e_0 = \text{vcc } r; r(x + 1) \times 2 \end{array} \right.$$

## Example 2

Let's interpret the expression  $(\lambda x.(\text{vcc } r; r(x + 1) \times 2))(3)$ :

$$\begin{array}{llll} \text{(App}_1\text{)} & \langle (\emptyset \vdash (\lambda x.(\text{vcc } r; r(x + 1) \times 2))(3)) :: \square & || \blacksquare & \rangle \\ \rightarrow & & & \\ \text{(Fun)} & \langle (\emptyset \vdash (\lambda x.(\text{vcc } r; r(x + 1) \times 2))) :: (\emptyset \vdash 3) :: (@) :: \square & || \blacksquare & \rangle \\ \rightarrow & & & \\ \text{(Num)} & \langle (\emptyset \vdash 3) :: (@) :: \square & || \langle \lambda x.e_0, \emptyset \rangle :: \blacksquare & \rangle \\ \rightarrow & \langle (@) :: \square & || 3 :: \langle \lambda x.e_0, \emptyset \rangle :: \blacksquare & \rangle \end{array}$$

$$\text{where } \left\{ \begin{array}{l} e_0 = \text{vcc } r; r(x + 1) \times 2 \end{array} \right.$$

## Example 2

Let's interpret the expression  $(\lambda x.(\text{vcc } r; r(x + 1) \times 2))(3)$ :

$$\begin{array}{ll}
 (\text{App}_1) & \langle (\emptyset \vdash (\lambda x.(\text{vcc } r; r(x + 1) \times 2))(3)) :: \square \quad \parallel \blacksquare \quad \rangle \\
 \rightarrow & \langle (\emptyset \vdash (\lambda x.(\text{vcc } r; r(x + 1) \times 2))) :: (\emptyset \vdash 3) :: (@) :: \square \parallel \blacksquare \quad \rangle \\
 (\text{Fun}) & \langle (\emptyset \vdash 3) :: (@) :: \square \quad \parallel \langle \lambda x.e_0, \emptyset \rangle :: \blacksquare \quad \rangle \\
 \rightarrow & \langle (@) :: \square \quad \parallel 3 :: \langle \lambda x.e_0, \emptyset \rangle :: \blacksquare \quad \rangle \\
 (\text{Num}) & \langle (@) :: \square \quad \parallel 3 :: \langle \lambda x.e_0, \emptyset \rangle :: \blacksquare \quad \rangle \\
 \rightarrow & \langle (@) :: \square \quad \parallel 3 :: \langle \lambda x.e_0, \emptyset \rangle :: \blacksquare \quad \rangle \\
 (\text{App}_{2,\lambda}) & \langle (\sigma_0 \vdash \text{vcc } r; r(x + 1) \times 2) :: \square \quad \parallel \blacksquare \quad \rangle \\
 \rightarrow & \langle (\sigma_0 \vdash \text{vcc } r; r(x + 1) \times 2) :: \square \quad \parallel \blacksquare \quad \rangle
 \end{array}$$

$$\text{where } \begin{cases} e_0 & = \text{vcc } r; r(x + 1) \times 2 \\ \sigma_0 & = [x \mapsto 3] \end{cases}$$

Let's interpret the expression  $(\lambda x.(\text{vcc } r; r(x + 1) \times 2))(3)$ :

$$\begin{array}{lll}
 & \langle (\emptyset \vdash (\lambda x.(\text{vcc } r; r(x + 1) \times 2))(3)) :: \square & || \blacksquare \rangle \\
 (\text{App}_1) \rightarrow & \langle (\emptyset \vdash (\lambda x.(\text{vcc } r; r(x + 1) \times 2))) :: (\emptyset \vdash 3) :: (@) :: \square & || \blacksquare \rangle \\
 (\text{Fun}) \rightarrow & \langle (\emptyset \vdash 3) :: (@) :: \square & || \langle \lambda x.e_0, \emptyset \rangle :: \blacksquare \rangle \\
 (\text{Num}) \rightarrow & \langle (@) :: \square & || 3 :: \langle \lambda x.e_0, \emptyset \rangle :: \blacksquare \rangle \\
 (\text{App}_{2,\lambda}) \rightarrow & \langle (\sigma_0 \vdash \text{vcc } r; r(x + 1) \times 2) :: \square & || \blacksquare \rangle \\
 (\text{Vcc}) \rightarrow & \langle (\sigma_1 \vdash r(x + 1) \times 2) :: \square & || \blacksquare \rangle
 \end{array}$$

$$\text{where } \begin{cases} e_0 &= \text{vcc } r; r(x + 1) \times 2 \\ \sigma_0 &= [x \mapsto 3] \\ \sigma_1 &= \sigma_0[r \mapsto \langle \square || \blacksquare \rangle] \end{cases}$$

Let's interpret the expression  $(\lambda x.(\text{vcc } r; r(x + 1) \times 2))(3)$ :

$$\begin{array}{lll}
 & \langle (\emptyset \vdash (\lambda x.(\text{vcc } r; r(x + 1) \times 2))(3)) :: \square & || \blacksquare \rangle \\
 (\text{App}_1) \xrightarrow{\quad} & \langle (\emptyset \vdash (\lambda x.(\text{vcc } r; r(x + 1) \times 2))) :: (\emptyset \vdash 3) :: (@) :: \square & || \blacksquare \rangle \\
 (\text{Fun}) \xrightarrow{\quad} & \langle (\emptyset \vdash 3) :: (@) :: \square & || \langle \lambda x.e_0, \emptyset \rangle :: \blacksquare \rangle \\
 (\text{Num}) \xrightarrow{\quad} & \langle (@) :: \square & || 3 :: \langle \lambda x.e_0, \emptyset \rangle :: \blacksquare \rangle \\
 (\text{App}_{2,\lambda}) \xrightarrow{\quad} & \langle (\sigma_0 \vdash \text{vcc } r; r(x + 1) \times 2) :: \square & || \blacksquare \rangle \\
 (\text{Vcc}) \xrightarrow{\quad} & \langle (\sigma_1 \vdash r(x + 1) \times 2) :: \square & || \blacksquare \rangle \\
 (\text{Mul}_1) \xrightarrow{\quad} & \langle (\sigma_1 \vdash r(x + 1)) :: (\sigma_1 \vdash 2) :: (@) :: (\times) :: \square & || \blacksquare \rangle
 \end{array}$$

$$\text{where } \begin{cases} e_0 & = & \text{vcc } r; r(x + 1) \times 2 \\ \sigma_0 & = & [x \mapsto 3] \\ \sigma_1 & = & \sigma_0[r \mapsto \langle \square || \blacksquare \rangle] \end{cases}$$

Let's interpret the expression  $(\lambda x.(\text{vcc } r; r(x + 1) \times 2))(3)$ :

	$\langle (\emptyset \vdash (\lambda x.(\text{vcc } r; r(x + 1) \times 2))(3)) :: \square \rangle$	$\parallel \blacksquare \rangle$
$(\text{App}_1)$ $\rightarrow$	$\langle (\emptyset \vdash (\lambda x.(\text{vcc } r; r(x + 1) \times 2))) :: (\emptyset \vdash 3) :: (@) :: \square \rangle$	$\parallel \blacksquare \rangle$
$(\text{Fun})$ $\rightarrow$	$\langle (\emptyset \vdash 3) :: (@) :: \square \rangle$	$\parallel \langle \lambda x.e_0, \emptyset \rangle :: \blacksquare \rangle$
$(\text{Num})$ $\rightarrow$	$\langle (@) :: \square \rangle$	$\parallel 3 :: \langle \lambda x.e_0, \emptyset \rangle :: \blacksquare \rangle$
$(\text{App}_{2,\lambda})$ $\rightarrow$	$\langle (\sigma_0 \vdash \text{vcc } r; r(x + 1) \times 2) :: \square \rangle$	$\parallel \blacksquare \rangle$
$(\text{Vcc})$ $\rightarrow$	$\langle (\sigma_1 \vdash r(x + 1) \times 2) :: \square \rangle$	$\parallel \blacksquare \rangle$
$(\text{Mul}_1)$ $\rightarrow$	$\langle (\sigma_1 \vdash r(x + 1)) :: (\sigma_1 \vdash 2) :: (@) :: (\times) :: \square \rangle$	$\parallel \blacksquare \rangle$
$(\text{App}_1)$ $\rightarrow$	$\langle (\sigma_1 \vdash r) :: (\sigma_1 \vdash x + 1) :: (@) :: (\sigma_1 \vdash 2) :: (\times) :: \square \rangle$	$\parallel \blacksquare \rangle$
...		

$$\text{where } \begin{cases} e_0 &= \text{vcc } r; r(x + 1) \times 2 \\ \sigma_0 &= [x \mapsto 3] \\ \sigma_1 &= \sigma_0[r \mapsto \langle \square \parallel \blacksquare \rangle] \end{cases}$$



## Example 2

Let's interpret the expression  $(\lambda x.(\text{vcc } r; r(x + 1) \times 2))(3)$ :

$$\begin{array}{lll}
 (\text{App}_1) & \langle (\emptyset \vdash (\lambda x.(\text{vcc } r; r(x + 1) \times 2))(3)) :: \square & || \blacksquare \rangle \\
 \rightarrow & \langle (\emptyset \vdash (\lambda x.(\text{vcc } r; r(x + 1) \times 2))) :: (\emptyset \vdash 3) :: (@) :: \square & || \blacksquare \rangle \\
 (\text{Fun}) & \rightarrow & \langle (\emptyset \vdash 3) :: (@) :: \square & || \langle \lambda x.e_0, \emptyset \rangle :: \blacksquare \rangle \\
 (\text{Num}) & \rightarrow & \langle (@) :: \square & || 3 :: \langle \lambda x.e_0, \emptyset \rangle :: \blacksquare \rangle \\
 (\text{App}_{2,\lambda}) & \rightarrow & \langle (\sigma_0 \vdash \text{vcc } r; r(x + 1) \times 2) :: \square & || \blacksquare \rangle \\
 (\text{Vcc}) & \rightarrow & \langle (\sigma_1 \vdash r(x + 1) \times 2) :: \square & || \blacksquare \rangle \\
 (\text{Mul}_1) & \rightarrow & \langle (\sigma_1 \vdash r(x + 1)) :: (\sigma_1 \vdash 2) :: (@) :: (\times) :: \square & || \blacksquare \rangle \\
 (\text{App}_1) & \rightarrow & \langle (\sigma_1 \vdash r) :: (\sigma_1 \vdash x + 1) :: (@) :: (\sigma_1 \vdash 2) :: (\times) :: \square & || \blacksquare \rangle \\
 \dots & & \\
 \rightarrow^* & \langle (@) :: (\sigma_1 \vdash 2) :: (\times) :: \square & || 4 :: \langle \square || \blacksquare \rangle :: \blacksquare \rangle
 \end{array}$$

$$\text{where } \begin{cases} e_0 & = & \text{vcc } r; r(x + 1) \times 2 \\ \sigma_0 & = & [x \mapsto 3] \\ \sigma_1 & = & \sigma_0[r \mapsto \langle \square || \blacksquare \rangle] \end{cases}$$

## Example 2

Let's interpret the expression  $(\lambda x. (\text{vcc } r; r(x + 1) \times 2))(3)$ :

$(\text{App}_1)$	$\langle (\emptyset \vdash (\lambda x. (\text{vcc } r; r(x + 1) \times 2))(3)) :: \square \rangle$	$\parallel \blacksquare \rangle$
$\xrightarrow{\quad}$	$\langle (\emptyset \vdash (\lambda x. (\text{vcc } r; r(x + 1) \times 2))) :: (\emptyset \vdash 3) :: (@) :: \square \rangle$	$\parallel \blacksquare \rangle$
$(\text{Fun})$	$\langle (\emptyset \vdash 3) :: (@) :: \square \rangle$	$\parallel \langle \lambda x. e_0, \emptyset \rangle :: \blacksquare \rangle$
$\xrightarrow{\quad}$	$\langle (@) :: \square \rangle$	$\parallel 3 :: \langle \lambda x. e_0, \emptyset \rangle :: \blacksquare \rangle$
$(\text{Num})$	$\langle (@) :: \square \rangle$	$\parallel 3 :: \langle \lambda x. e_0, \emptyset \rangle :: \blacksquare \rangle$
$\xrightarrow{\quad}$	$\langle (@) :: \square \rangle$	$\parallel 3 :: \langle \lambda x. e_0, \emptyset \rangle :: \blacksquare \rangle$
$(\text{App}_{2,\lambda})$	$\langle (\sigma_0 \vdash \text{vcc } r; r(x + 1) \times 2) :: \square \rangle$	$\parallel \blacksquare \rangle$
$\xrightarrow{\quad}$	$\langle (\sigma_1 \vdash r(x + 1) \times 2) :: \square \rangle$	$\parallel \blacksquare \rangle$
$(\text{Vcc})$	$\langle (\sigma_1 \vdash r(x + 1) \times 2) :: \square \rangle$	$\parallel \blacksquare \rangle$
$\xrightarrow{\quad}$	$\langle (\sigma_1 \vdash r(x + 1) \times 2) :: \square \rangle$	$\parallel \blacksquare \rangle$
$(\text{Mul}_1)$	$\langle (\sigma_1 \vdash r(x + 1)) :: (\sigma_1 \vdash 2) :: (@) :: (\times) :: \square \rangle$	$\parallel \blacksquare \rangle$
$\xrightarrow{\quad}$	$\langle (\sigma_1 \vdash r(x + 1)) :: (\sigma_1 \vdash 2) :: (@) :: (\times) :: \square \rangle$	$\parallel \blacksquare \rangle$
$(\text{App}_1)$	$\langle (\sigma_1 \vdash r) :: (\sigma_1 \vdash x + 1) :: (@) :: (\sigma_1 \vdash 2) :: (\times) :: \square \rangle$	$\parallel \blacksquare \rangle$
$\xrightarrow{\quad}$	$\langle (\sigma_1 \vdash r) :: (\sigma_1 \vdash x + 1) :: (@) :: (\sigma_1 \vdash 2) :: (\times) :: \square \rangle$	$\parallel \blacksquare \rangle$
$\dots$		
$\xrightarrow{*}$	$\langle (@) :: (\sigma_1 \vdash 2) :: (\times) :: \square \rangle$	$\parallel 4 :: \langle \square \parallel \blacksquare \rangle :: \blacksquare \rangle$
$(\text{App}_{2,\kappa})$	$\langle \square \rangle$	$\parallel 4 :: \blacksquare \rangle$

$$\text{where } \begin{cases} e_0 &= \text{vcc } r; r(x + 1) \times 2 \\ \sigma_0 &= [x \mapsto 3] \\ \sigma_1 &= \sigma_0[r \mapsto \langle \square \parallel \blacksquare \rangle] \end{cases}$$

1. First-Class Continuations
2. KFAE – FAE with First-Class Continuations  
Concrete/Abstract Syntax
3. Interpreter and Reduction Semantics for KFAE  
Recall: Interpreter and Reduction Semantics for FAE  
Interpreter and Reduction Semantics for KFAE  
First-Class Continuations  
Function Application  
Example 1  
Example 2
4. Control Statements

Many real-world programming languages support **control statements** to change the **control-flow** of a program.

Many real-world programming languages support **control statements** to change the **control-flow** of a program.

For example, C++ supports **break**, **continue**, and **return** statements:

```
int sumEvenUntilZero(int xs[], int len) {  
    if (len <= 0) return 0;           // directly return 0 if len <= 0  
    int sum = 0;  
    for (int i = 0; i < len; i++) {  
        if (xs[i] == 0) break;        // stop the loop if xs[i] == 0  
        if (xs[i] % 2 == 1) continue; // skip the rest if xs[i] is odd  
        sum += xs[i];  
    }  
    return sum;                       // finally return the sum  
}  
  
int xs[] = {4, 1, 3, 2, 0, 6, 5, 8};  
sumEvenUntilZero(xs, 8);             // 4 + 2 = 6
```

Let's represent them using **first-class continuations**!

- `return` statement:

```
x => body
```

- **return** statement:

```
x => body
```

means

```
x => { vcc return;  
      body // return(e) directly returns e to the caller  
}
```

- **return** statement:

```
x => body
```

means

```
x => { vcc return;  
      body // return(e) directly returns e to the caller  
}
```

- **break** and **continue** statements:

```
while (cond) body
```



- **return** statement:

```
x => body
```

means

```
x => { vcc return;  
      body // return(e) directly returns e to the caller  
    }
```

- **break** and **continue** statements:

```
while (cond) body
```

means

```
{ vcc break;  
  while (cond) { vcc continue;  
    body // continue(e)/break(e) jumps to the next/end of the loop  
  }  
}
```

We can represent other control statements similarly, but think for yourself!

- exception in Python

```
try:
    x = y / z
except ZeroDivisionError:
    x = 0
```

- generator in JavaScript

```
const foo = function* () { yield 'a'; yield 'b'; yield 'c'; };
let str = '';
for (const c of foo()) { str = str + c; }
str // 'abc'
```

- coroutines in Kotlin
- async/await in C#
- ...

## 1. First-Class Continuations

## 2. KFAE – FAE with First-Class Continuations

Concrete/Abstract Syntax

## 3. Interpreter and Reduction Semantics for KFAE

Recall: Interpreter and Reduction Semantics for FAE

Interpreter and Reduction Semantics for KFAE

First-Class Continuations

Function Application

Example 1

Example 2

## 4. Control Statements

- Please see this document<sup>1</sup> on GitHub.
  - Implement reduce function.
- It is just an exercise, and you **don't need to submit** anything.
- However, some exam questions might be related to this exercise.

---

<sup>1</sup><https://github.com/ku-plrg-classroom/docs/tree/main/cose212/kfae>.

- Compiling with Continuations

Jihyeok Park

[jihyeok\\_park@korea.ac.kr](mailto:jihyeok_park@korea.ac.kr)

<https://plrg.korea.ac.kr>