The CPS Transform

Dr. Mattox Beckman

University of Illinois at Urbana-Champaign
Department of Computer Science

Objectives

You should be able to ...

You've seen how to write CPS functions by hand, but we want you to know the mathematical definition.

After today's lecture, you will

- Convert a direct-style function into CPS:
  - Both simple and complex, involving nested continuations.

The CPS Transform, Simple Expressions

Top Level Declaraion To convert a declaration, add a continuation argument to it and then convert the body.

$$C[\![f\ arg\ =\ e)]\!] \Rightarrow f\ arg\ k\ =\ C[\![e]\!]_k$$

Simple Expressions A simple expression in tail position should be passed to a continuation instead of returned.

$$C[\![a]\!]_k \Rightarrow\ k\ a$$

- "Simple" = "No available function calls."
- $f\ a$ is available in $3 + f\ a$, but not in $\lambda x.x + f\ a$.

Try converting these functions ...

```
f x = x
pi1 a b = a
const x = 10
```

Simple Expression Examples

Before:

```
f x = x
pi1 a b = a
const x = 10
```

After:

```
f x k = k x
pi1 a b k = k a
const x k = k 10
```

The CPS Transform, Function Calls

Function Call on Simple Argument To a function call in tail position (where `arg` is simple), pass the current continuation.

$$C[\![f\ arg]\!]_k \Rightarrow f\ arg\ k$$

Function Call on Non-simple Argument If `arg` is not simple, we need to convert it first.

$$C[\![f\ arg]\!]_k \Rightarrow C[\![arg]\!]_{(\lambda v.f\ v\ k)},\ \text{where } v \text{ is fresh.}$$

Try converting these functions.

```
foo 0 = 0
foo n | n < 0       = foo n
      | otherwise = inc (foo n)
```

Example

```haskell
foo 0 = 0
foo n | n < 0     = foo n
      | otherwise = inc (foo n)
```

```haskell
foo 0 k = k 0
foo n k | n < 0     = foo n k
        | otherwise = foo n (\v -> inc v k)
```

The CPS Transform, Operators

Operator with Two Simple Arguments If both arguments are simple, then the whole thing is simple.

$$C[\![e_1 + e_2]\!]_k \Rightarrow\ k(e_1 + e_2)$$

Operator with One Simple Argument If e_2 is simple, we transform e_1.

$$C[\![e_1 + e_2]\!]_k \Rightarrow\ C[\![e_1]\!]_{(\lambda v -> k(v+e2))} \text{where } v \text{ is fresh.}$$

Operator with No Simple Arguments If both need to be transformed ...

$$C[\![e_1 + e_2]\!]_k \Rightarrow\ C[\![e_1]\!]_{(\lambda v_1 -> C[\![e_2]\!]_{\lambda v_2 -> k(v_1+v_2)})} \text{where } v_1 \text{ and } v_2 \text{ are fresh.}$$

Notice that we need to nest the continuations!

Examples

```
foo a b = a + b
bar a b = inc a + b
baz a b = a + inc b
quux a b = inc a + inc b
```

Examples

```
foo a b = a + b
bar a b = inc a + b
baz a b = a + inc b
quux a b = inc a + inc b
```

```
foo a b k = k (a + b)
```

Examples

```
foo a b = a + b
bar a b = inc a + b
baz a b = a + inc b
quux a b = inc a + inc b
```

```
foo a b k = k (a + b)
bar a b k = inc a (\v -> k (v + b))
```

Examples

```
foo a b = a + b
bar a b = inc a + b
baz a b = a + inc b
quux a b = inc a + inc b
```

```
foo a b k = k (a + b)
bar a b k = inc a (\v -> k (v + b))
baz a b k = inc b (\v -> k (a + v))
```

Examples

```
foo a b = a + b
bar a b = inc a + b
baz a b = a + inc b
quux a b = inc a + inc b
```

```
foo a b k = k (a + b)
bar a b k = inc a (\v -> k (v + b))
baz a b k = inc b (\v -> k (a + v))
quux a b k = inc a (\v1 -> inc b (\v2 -> k (v1 + v2)))
```

References

[DF90] Olivier Danvy and Andrzej Filinski. "Abstracting control". In: *Proceedings of the 1990 ACM conference on LISP ...* (1990), pp. 151–160. ISSN: 1098-6596. DOI: http://doi.acm.org.ezp-prod1.hul.harvard.edu/10.1145/91556.91622.

[DF92] Oliver Danvy and Andrzex Filinski. "Representing Control: a Study of the CPS Transformation". In: *Mathematical Structures in Computer Science* 2.04 (1992), p. 361. ISSN: 0960-1295. DOI: 10.1017/S0960129500001535.

[Rey93] John C. Reynolds. "The discoveries of continuations". In: *LISP and Symbolic Computation* 6.3 (Nov. 1993), pp. 233–247. ISSN: 1573-0557. DOI: 10.1007/BF01019459. URL: https://doi.org/10.1007/BF01019459.