

Announcements

- HW6 Graded
- Late Days column on HuskyCT updated
- HW7 due tonight (Saturday with 2 late days)
 - I'll post solutions Sunday so please be on time!

Final Exam, Tue. 12/9 8-10am, ITE 125

- Content: Everything!
 - Including this week's lectures, though these lectures were high-level so the questions will be too.
 - Focus will be on material since the midterm, though this built on pre-midterm stuff.
- Format: Similar to midterm (true-false + 4-5 longer questions)
 - True-false: I give a statement, you agree or disagree **and explain** (explanation is very important but still keep it to 1-2 sentences)
- 100 points, 120 minutes. Keep to ~1 point/minute
- Reference material (posted today or tomorrow) will be provided
- You can bring **three** 8.5x11" note sheets with any content
- Best way to study: review homeworks and the midterm

Continuations and Wrap-up

CSE 5095-002, Fall 2025

Continuations capture “the rest of the stuff to do”

Kind of like evaluation contexts $E[e]$

Ex. $\underbrace{(\text{fst } (\text{fst } (\text{fst } (o))))}_{\text{continuation}} [\text{fst } (7, 8)]$

First-class continuations

- Type α cont = continuation expecting a α .
- `throw` : α cont $\rightarrow \alpha \rightarrow \beta$
 - `throw k v` “calls” the continuation `k` with value `v`

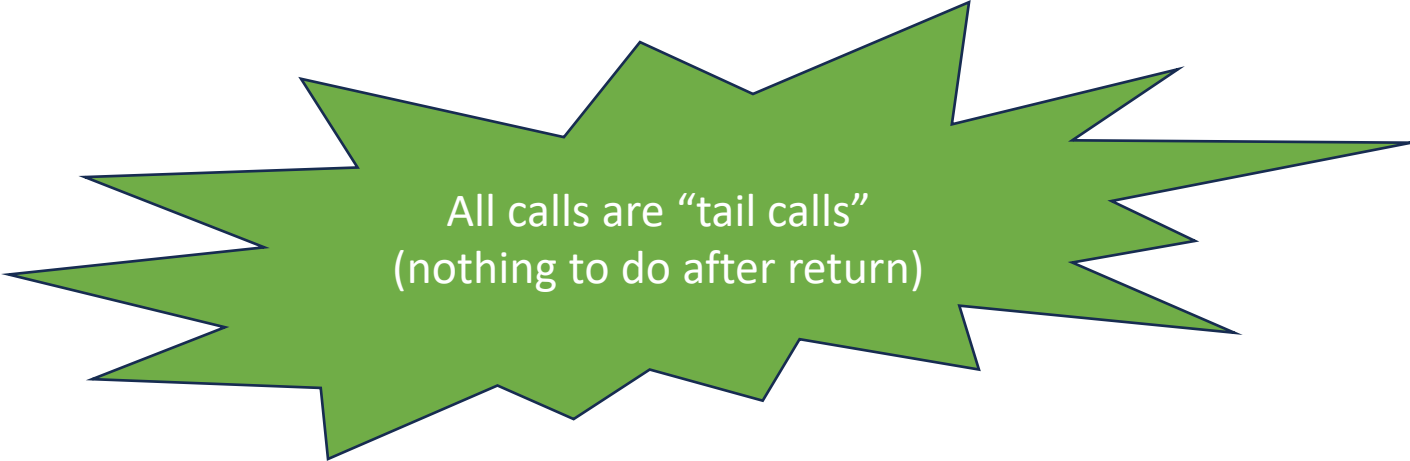
Example: tail recursion

```
fix mult_all =  $\lambda$ l : intlist.  
  case unroll l of  
    {_. 1;  
     (h, t). h * (mult_all t) }
```



Example: tail calls/recursion

```
fix mult_all =  $\lambda l : \text{intlist}. \lambda k : \text{int} \text{ cont}.$   
  case unroll l of  
    {_. throw k 1;  
     (h, t). mult_all t ( $\lambda v. \text{throw } k (v * h)$ )}
```



All calls are “tail calls”
(nothing to do after return)

Example: short-circuiting

```
fix mult_all =  $\lambda l : \text{intlist}. \lambda k : \text{int} \text{ cont}.$   
  case unroll l of  
    {_. throw k 1;  
     (h, t). if h = 0 then throw k 0  
              else mult_all t ( $\lambda v. \text{throw } k (v * h)$ )}
```

Example: exceptions

Normal
control flow

Exceptional
control flow

$\lambda a : \text{int}. \lambda b : \text{int}. \lambda k : \text{int cont}. \lambda f : \text{string cont}.$
 if $b \neq 0$ then throw k (a / b)
 else throw f “Divide by zero”

But how do we get a continuation?

- `call/cc` : $(\alpha \text{ cont} \rightarrow \beta) \rightarrow \alpha$
 - “Call with current continuation”
 - `call/cc f` captures the current continuation as an object and passes it to `f`.
 - $E[\text{call/cc } f] \rightarrow f \text{ “E”}$
- e.g., `call/cc (mult_all [1; 2; 3; 4; 5])`

A continuation is a function

- From α to... what?
- Whatever the result of the computation is: need some designated “result type”
- Things work out surprisingly nicely if we choose `void`

So, under Curry-Howard, α cont = $\alpha \rightarrow \perp = \neg \alpha$

The typing rule for call/cc is interesting under Curry-Howard...

$$\frac{\Gamma \vdash e : (\alpha \rightarrow \text{void}) \rightarrow \text{void}}{\Gamma \vdash \text{callcc } e : \alpha}$$

$$\frac{\Gamma \vdash \neg\neg A \text{ true}}{\Gamma \vdash A \text{ true}}$$

STLC with call/cc = Classical Logic!

- Just to be sure: $A \vee \neg A = \alpha + \alpha \text{ cont}$
- `call/cc` $(\lambda k : (\alpha + \alpha \text{ cont}) \text{ cont}.$
 $\text{throw } k \text{ (inr } (\lambda v : \alpha. \text{ throw } k \text{ (inl } v))))$

Continuation Passing Style (CPS): All function calls are tail calls

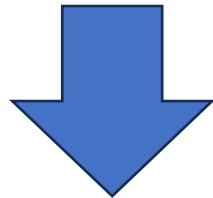
- Difficult to program in, but some efficiency benefits (and helpful for implementing first-class continuations)
- Can be done automatically (and is by some compilers for functional languages) (“CPS transformation”)

With CPS translation, we don't need call/cc but types change a little

call/cc ($\lambda k : (\alpha + \alpha \text{ cont}) \text{ cont}.$
 throw k (inr ($\lambda v : \alpha.$ throw k (inl v))))

$$\alpha + \alpha \text{ cont} = A \vee \neg A$$

Not true constructively



($\lambda k : (\alpha + \alpha \text{ cont}) \text{ cont}.$ k (inr ($\lambda v : \alpha.$ k (inl v))))

$$(\alpha + \alpha \text{ cont}) \text{ cont cont} = \neg \neg (A \vee \neg A)$$

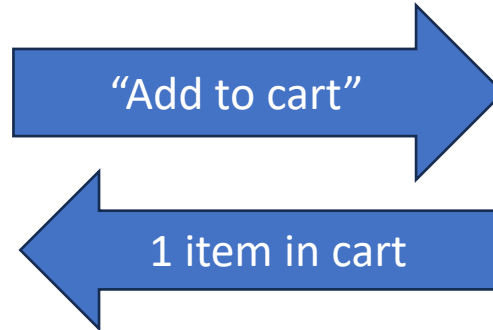
True constructively
(proof is above)

So does that mean that if A is true classically,
 $\neg\neg A$ is true constructively?

- Yes [Glivenko, 1929]

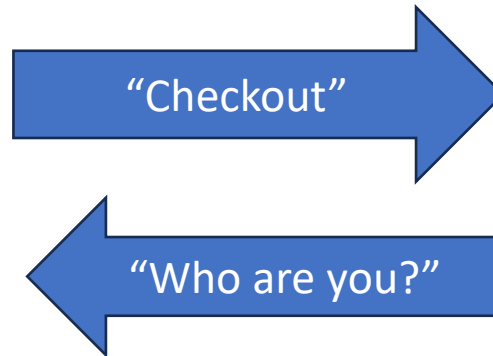
Another application of continuations: stateless web services

Black Friday:



amazon

Wednesday:



amazon

Another application of continuations: stateless web services

Black Friday:



amazon

Wednesday:



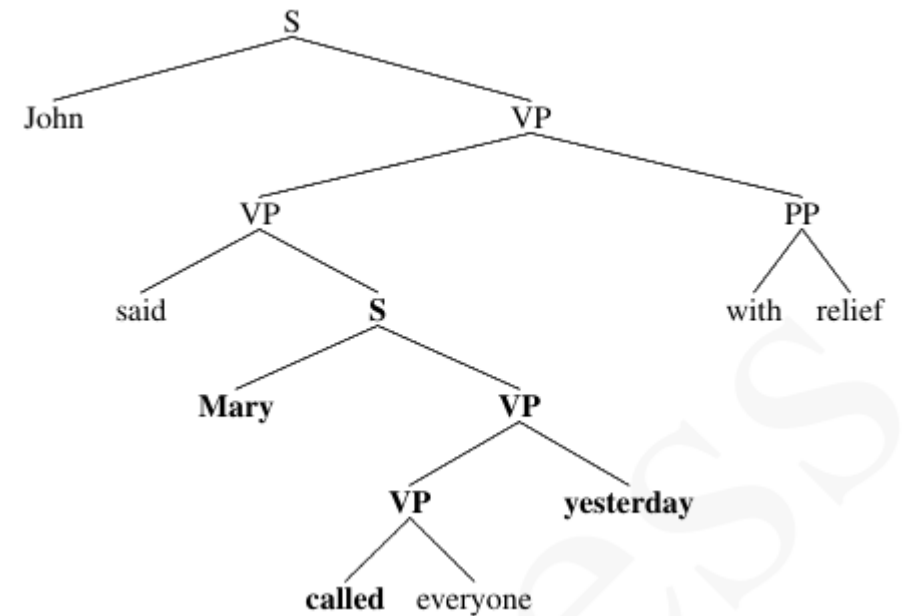
amazon

(λ action. ...) "Checkout"

Another application of continuations: natural language

- “John said Mary called everyone yesterday with relief”
- Q: What happened to everyone?

$(\lambda x. \text{Mary called } x \text{ yesterday})$



Barker, Chris & Shan, Chung-chieh. (2014). Continuations and Natural Language.
10.1093/acprof:oso/9780199575015.001.0001.

Another application of continuations: natural language

An occasional sailor passed by (misplaced modifier)

k occasional

Occasionally, *k* ($\lambda x. x$)

Occasionally, a sailor passed by

$(\lambda x. ?)$ this class

- ... unfortunately not a lot of classes at Uconn
- Research/independent study!
- Advanced Topics in Types and Programming Languages (ed. Pierce)
- Software Foundations (Pierce, <https://softwarefoundations.cis.upenn.edu>)