Linear Type Systems

Free exactly once

```
void* p = malloc(...);
  ...
free(p);
  ...
free(p);
```

```
void *p = malloc(...);
  ...
// Memory leak!
```

Write in at most one thread

```
void f () {
  (*p)++
}
pthread_create(f);
f();
```

Remember: Structural rules

$$\frac{\Gamma \vdash e : \tau}{\Gamma, x:\tau' \vdash e : \tau} \text{ weakening} \qquad \frac{\Gamma, x:\tau_1, y:\tau_2 \vdash e:\tau}{\Gamma, y:\tau_2, x:\tau_1 \vdash e:\tau} \text{ Exchange}$$

$$\frac{\Gamma, x:\tau', x:\tau' \vdash e:\tau}{\Gamma, x:\tau' \vdash e:\tau} \text{ contraction}$$

Get rid of weakening + contraction
⇒ Linear type system
⇒ Use variables exactly once
(just contraction ⇒ Affine T.S. ⇒ At most once)

$e ::= x \mid () \mid \lambda x{:}\tau.e \mid e\ e \mid (e,e) \mid \text{let } (x,y) = e \text{ in } e$

$\tau ::= \text{unit} \mid \tau \multimap \tau \mid \tau \otimes \tau$

"lolly" "tensor"

? !

fst(x, y)

lost!

$$\frac{}{x:\tau \vdash x:\tau}\ (T\text{-}1)$$

$$\frac{}{\bullet \vdash ():\text{unit}}\ (T\text{-}2)$$

$$\frac{\Gamma, x:\tau_1 \vdash e:\tau_2}{\Gamma \vdash \lambda x:\tau_1.e : \tau_1 \multimap \tau_2}\ (T\text{-}3)$$

$$\frac{\Gamma_1 \vdash e_1 : \tau_1 \multimap \tau_2 \quad \Gamma_2 \vdash e_2 : \tau_1}{\Gamma_1, \Gamma_2 \vdash e_1\ e_2 : \tau_2}\ (T\text{-}4)$$

$$\frac{\Gamma_1 \vdash e_1 : \tau_1 \quad \Gamma_2 \vdash e_2 : \tau_2}{\Gamma_1, \Gamma_2 \vdash (e_1, e_2) : \tau_1 \otimes \tau_2}\ (T\text{-}5)$$

$$\frac{\Gamma_1 \vdash e_1 : \tau_1 \otimes \tau_2 \quad \Gamma_2, x:\tau_1, y:\tau_2 \vdash e_2 : \tau}{\Gamma_1, \Gamma_2 \vdash \text{let } (x,y) = e_1 \text{ in } e_2 : \tau}\ (T\text{-}6)$$

$$\cfrac{\cfrac{\cfrac{\cfrac{}{x:\tau_1 \vdash x:\tau_1} \quad \cfrac{}{y:\tau_2 \vdash y:\tau_2}}{x:\tau_1, y:\tau_2 \vdash (x,y) : \tau_1 \otimes \tau_2}}{x:\tau_1 \vdash \lambda y:\tau_2.(x,y) : \tau_2 \multimap \tau_1 \otimes \tau_2}}{\bullet \vdash \lambda x:\tau_1.\lambda y:\tau_2.(x,y) : \tau_1 \multimap \tau_2 \multimap \tau_1 \otimes \tau_2}$$

x can only go in one ctx!

$$\cfrac{\cfrac{? \vdash x: \quad ? \vdash x:}{x:\tau \vdash (x,x):}}{\bullet \vdash \lambda x:\tau.(x,x):}$$

Return to free, data race examples

void* p=malloc (···)
Want to be able to use p!

Combine linear + non-linear contexts
Idea: separate Pointer + capability

Pointer: non-linear, use to read/write (read)

capability: linear (affine) use to free (write)

non-lin / lin

$\Delta; \Gamma \vdash e : \tau$

$$\frac{\Delta, p:\tau\ \text{ptr}; \Gamma, c:\text{cap} \vdash e:\tau}{\Delta;\Gamma \vdash \text{let } (p,c) = \text{malloc in } e:\tau}$$

$$\frac{\Delta;\Gamma \vdash e : \tau\ \text{ptr} \times \text{cap}}{\Delta;\Gamma \vdash \text{free } e : ()}$$

```
let (p,c) = malloc in
   p := p + p;
   free (p,c) : unit
   free(p,c) X
```

Linear Logic (By Curry-Howard Correspondence)

Logic where facts can be used exactly once
Useful for reasoning about resources

Prop. Logic: VM: I have \$1.50 → I have a candy bar

$$\dfrac{\dfrac{\overline{VM, \$1.50 \vdash \$1.50} \quad \dfrac{\cdots}{VM, \$1.50 \vdash Candy}}{VM, \$1.50, \$1.50 \vdash \$1.50 \wedge Candy}}{VM, \$1.50 \vdash \$1.50 \wedge Candy} \quad \text{OOPS}$$

Linear: VM = \$1.50 ⊸ candy

Needs both \$1.50 and VM

$$\dfrac{X \vdash \$1.50 \qquad \vdash Candy}{VM, \$1.50 \vdash \$1.50 \otimes Candy}$$

Thanks to Cornell 6110