15-1/ Safety

(÷ 1 0) (5 3)

E[(u v)] ⟶ E[abort "Tried to call non-function"]
where u ∉ p, λ

<v, _, kapp (u) _ () k>
⟼ <abort "Tried to call non-function", _, k>

15-2/ switch (c->tag) {

```
    case C_IF: ....
         :
    case C_LAM: ...
    case C_NUM:
    case C_BOOL:
      switch (k->tag) {
        case K_IF: ...
        case K_APP:
          if (k->es->tag == EMPTY) {
            if (k->vs[0]->tag == CLO) { ...}
                                 == PRIM) { ...}
          else { ... do the abort sqn ...
```

15-3 / A <u>safe</u> kernel of language

is when the lowest level enforces the invariants

of the languages.

.... This is expensive!

... the code in Delta for addition ...

(((num_t *) args[0]) -> n
+ ((num_t *) args[1]) -> n)

make_num( )

(+ true false) $\longrightarrow$ 1

struct {
bool_t b; }

make_bool(true)

15-4/ JS, Py, Ruby, Racket, ...
all intend to be safe

C is not like, nor is C++

<u>unsafe</u>

15-5/ How to have an unsafe kernel (fast) AND a safe language?

OLD:

+ is the kernel's plus

NEW:

+ is in the standard library

unsafe-+ is the kernel's plus (#%+)

15-6

```
(define (+ x y)
  (if (and (number? x)
           (number? y))
      (unsafe-+ x y)
      (throw "+: given non-numbers")))
```

Must add number? to the kernel

box? boolean? pair? unit? inl? inr?

continuation? function? primitive-arity

function-arity primitive?

15-7/ (+ 1 2 3) (+ 1)

desugar [f, x_0, ..., x_n] =

(if (and (~~function?~~ procedure? f)

(= n (~~function-arity~~ procedure-arity f)))

(f x_0 ... x_n)

(throw "not a fun or wrong arg"))

(define (procedure? x)

(or (function? x) (primitive? x) (continuation? x)))

(define (procedure-arity x)

(if (function? x) (function-arity x)

(if (primitive? x) (primitive-arity x) 1)))

J ⇒ J

15-8) (+ 1 2) (if True x y) ⇒ x

⇒ desugar

(if (and (procedure? +) (= (procedure-arity +) 2))

(+ 1 2) ...)

$\xRightarrow{\text{optimizer}}$ (+ 1 2)

⇒ (if (and (num? 1) (num? 2))

(unsafe-+ 1 2) ...)

⇒ (unsafe-+ 1 2)

(1 line 3)

15-9/ violation of invariant:

- what was sent — true → val
- what was supposed — num → ctc
- who sent it — you → pos label
- who recv'd it — + → neg label

Software contracts — express and enforce invariants

```
(protect ctc val pos neg)
→ (if (ctc val) val
      (error "expected " ctc " but got" val
             " from " pos " to " neg))
```

15-10] desugar ["+", x, y] ⇒
(unsafe-+ (protect number? x "line 27" "+")
(protect number? y "line 29" "+")]

15-11 A flat contract is one on an atomic value.

number? bool?

Concat Map : (Num ⇒ Str) x List(Num) ⇒ Str

protect (listof ctc) v pos neg ⇒

(and (list? v) (checkall ctc v pos neg)

(check all ctc [] pos neg ⇒ true

checkall ctc (a:d) pos neg ⇒

(and (protect ctc a pos neg)

(checkall ctc d pos neg)

15-12/

dom, rng

protect (Num → Str) x pos neg ⇒

(if (and (procedure? x) (= (procedure-arity x) 1))

(λ (arg)

(protect rng (x (protect dom arg neg pos))

pos neg)

higher-order contract