22-1

```
(define-syntax-rule (or2 x y)
  (let ([tmp x])
    (if tmp
        tmp
        y)))
```

```
(if x
    x
    y)
```

x = (printf ...)

```
(let ([tmp0 #f])
  (or2 #f tmp0))
```

⇓

```
(let ([tmp0 #t])
  (let ([tmp1 #f])
    (if tmp1
        tmp1
        tmp0)))
```

```
(define tmp (gensym 'if))
`(let ([,tmp ,x])
   (if ,tmp
       ,tmp
       ,y))
```

"Hygeine"

```
(let ([+ -])
  (mac1 2 3))
```

⇒ 5 ~~⇒ -1~~

~~(define (gensym 'if) +)~~

```
(dsr (mac1 x y)
     (+ x y))
```

```
(dsr (POTUS pw)
     (when (= pw PASS)
       (launch-missiles!)))
```

```
(define x 5)
(dsr (mac2 y)
     (let ([x (+ y 1)])
       x))
(mac2 3)
```

```
(define x 5)
(dsr (weird ~~#f~~ x xe b)
     (let ([x xe]) b))
(dsr (mac2 y)
     (weird x (+ y 1) x))
(mac2 3)
```

2-2 blue

```
(let ([tmp #t])
  (or2 #f tmp))
```

```
(dsr (or2 x y)
  (let ([tmp x]) (if tmp tmp y)))
```

```
(let_b ([tmp_b #t])
  (or2_br #f tmp_br))
```

```
(dsr (or2 x y)
  (let_b ([tmp_b x_b]) (if tmp_b tmp_b y_b)))
```

=>

```
(let_b ([tmp_b #t])
  (let_b ([tmp_b #f])
    (if_b tmp_b tmp_b tmp_br)))
```

=>

```
(let_b ([tmp_b #t])
  (let_b̄ ([tmp_b̄ #f])
    (if_b̄ tmp_b̄ tmp_b̄ tmp_b)))
```

o.p. painted #o

macro invocation i, is painted #i

macro transcription i is paint #ī

" Env ⊢ e → e' "

env = id → denotation

denot = special + macro + id

special = (λ, let-syntax)

macro = (pattern × rewrite)$^+$ × env

ident ∈ env

lookup ∈ env × id → denot

bind ∈ env × id × denot → env

divert ∈ env × env → env

lookup(ident, x) = x

lookup(bind(e,x,v), y) = v $^{x=y}$ o.w. (x ≠ y) lookup(e, y)

divert(e, ident) = e

divert(e, bind(e', x, v)) = bind(divert(e,e'), x, v)

$$\frac{\text{lookup}(e,x) \in \text{identifier}}{e \vdash x \to \text{lookup}(e,x)}$$

$$\frac{\text{lookup}(e, k_0) = \text{lambda} \quad \text{bind}(e, x, x') \vdash E \to E' \quad (x' \text{ is fresh})}{e \vdash (k_0\ (x)\ E) \to (\text{lambda}\ (x')\ E')}$$

$$\frac{\text{lookup}(e, k_0) = \text{let-syntax} \quad \text{bind}(e, k, \langle \tau, e \rangle) \vdash E \to E'}{e \vdash (k_0\ ((k\ \tau))\ E) \to E'}$$

$$\frac{\text{lookup}(e,k) = \langle \tau, e' \rangle \quad \text{transcribe}((k\ .\ \_), \tau, e, e') = \langle E, e'' \rangle \quad e'' \vdash E \to E'}{e \vdash (k\ .\ \_) \to E'}$$

$$\frac{e \vdash E_0 \to E_0' \qquad e \vdash E_1 \to E_1'}{e \vdash (E_0\ E_1) \to (E_0'\ E_1')}$$

$$\frac{\text{match}(E, \pi, e_{use}, e_{def}) = \text{nomatch} \quad \text{transcribe}(E, \tau', e_{use}, e_{def}) = \langle E', e' \rangle}{\text{transcribe}(E, \langle \langle \pi, p \rangle, \tau' \rangle, e_{use}, e_{def}) = \langle E', e' \rangle}$$

$$\frac{\text{match}(E, \pi, e_{use}, e_{def}) = \sigma \quad \text{rewrite}(p, \sigma, e_{def}) = \langle E', e_{new} \rangle}{= \langle E', \text{divert}(e_{use}, e_{new}) \rangle}$$

12-4/ match(E, ?v, euse, edef) = {?v ↦ E}

$$\frac{\text{lookup}(edef, x') = \text{lookup}(euse, x)}{\text{match}(x, x', euse, edef) = \{\}}$$

---

rewrite(p, σ, edef) = ⟨ rewrite'(p, σ') , enew ⟩

enew = bind(... bind(ident, x'1, d1) ... , x'n, dn)

x1 ... xn = all identifiers in p

x'1 ... x'n = all fresh

d1 ... dn = lookup(edef, xi)

σ' = σ ∪ { xi ↦ x'i }

rewrite'(?v, σ) = σ(?v)

rewrite'(x, σ) = σ(x)