m \ n   merge, n's fields overriding m's fields
-object-level only? how should metafields merge?

λx. M   respondent: object → object
- args are tupled-up by name

a and: [b]   (#and:, a, [b])

x ifTrue: [y] ifFalse: [z]   (#ifTrue:ifFalse:, x, [y], [z])

x copy.   (#copy, x)
x asString   (#asString, x)

1 + 2   (#+, 1, 2)

x asString.   x.asString
x if true: [y] false: [z]   x if true: [y] false: [z]

a and [b]   a and: [b]   a and: [b]

```
(  (                      ))
 x foo if notNil: [blah]
((       )                 )
-------------------------------
 x foo bar if true: [blah]
(     (                   ))
((         )               )
```

x.foo.bar if true: [blah]
(x foo bar) if true: [blah]
x foo (bar if true: [blah])

Referential Transparency as cure for agoraphobia

Dispatch:

S V m_1: a_1 m_2: a_2 ⋯ → #V, (subject: S m_1: a_1 m_2: a_2 ⋯)

Candidates from
S meta at: #subject selector: #V
a_1 meta at: #m_1 selector: #V
a_2 meta at: #m_2 selector #V
etc.

Need to retain both PMD's advantages without introducing ambiguity into the method lookup
- does PMD need a partial-order over roles? (modifiers)
Arbitrary conflict resolution? with "resend" taking the strain?

foo bar zot: z vs. foo bar quux: z
where #bar defd on (subject: foo class, zot: z class)
and also on (subject: foo class quux: z class)

evaluated in an environment where only globals are present?
Or, required to be a Symbol? "global" var.

Method Definition:

Subjectexpr $\overset{\text{adjective}}{\text{verb}}$ modifier: Objectexpr ⋯
expr.
expr.
expr.

Subject (punct-op) Object
expr.
expr.

Looping? (let loop ((a₁ v₁) (a₂ v₂)) x₁ x₂ ⋯)
⇒ loop ~~##~~ [a_1:v_1 a_2:v_2 | x_1. x_2. ⋯].
name [arg_1 : $init_1$ arg_2 : $init_2$ | exp. exp]

Magic 'keywords':
- subject
- resend
- @, (), [], ←

Locations?
Transactions? + PE...

Array new size: 6
Socket new host: h port: p.

x ← 3.
[illegible]

{ car. cdr } ← 1 :: ()

Installing a method (other than implicitly by field update) requires access to the metalevel.

let x = 3 in
let (car, cdr) = 1 :: nil in
...

✓ myBlock@(x: 1 y: 2).
✓ thunk @ ().
× thunk\().
× myBlock\(x: 1 y: 2)

let 1 :: nil => x ← .car

[expr. expr. expr]
[field field field | expr. expr]
~~x . asString~~

[illegible] x. as: String

x. toHex

x: toHex

x :toHex

x :asString :hex

Subject verb modifier: argument
(adjective above verb)
Subject (punct-op) Object

(field: value field: value)
myPair\(car: 6).
m ← m\(car: 6).
(car: car, cdr: cdr) ← m.
~~[illegible]~~
a and :[b]

~~[illegible]~~ subject? NO: This is not method update syntax
p ← p\(verb: [field field | expr. expr]) — this is field update syntax

(123 + 123) :asByteArray :hex print on: Console :writeStream.

123 asByteArray hex print on: Console writeStream.

"Boolean if" ? no. "True if true: Block", "False if false: Block"
"Object if notNil: Block", "Nil if nil: Block"