## Expressions (and reductions)

The usual way to calculate an expression, one step at a time, with '='s:

$$2 \cdot 3 + 4 \cdot 5 = 2 \cdot 3 + 20$$

$$= 6 + 20$$

$$= 26$$

$$2 \cdot 3 + 4 \cdot 5 = 6 + 4 \cdot 5$$

$$= 6 + 20$$

$$= 26$$

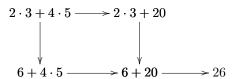
Each '=' corresponds to a ' $\longrightarrow$ ' in the reduction diagram below.

A notation for calculating the value of an expression by calculating the values of all its subexpressions:

$$\underbrace{2 \cdot 3}_{6} + \underbrace{4 \cdot 5}_{20}$$

Each '=' in the previous diagram corresponds to applying one '\_\_\_'.

A reduction diagram for  $2 \cdot 3 + 4 \cdot 5$ : (See Hindley/Seldin, pages 14 and 17)



Note that when we can choose two subexpressions to calculate the ' $\downarrow$ ' evaluates the leftmost one, and the ' $\to$ ' evaluates the rightmost one.

The subexpressions of  $2 \cdot 3 + 4 \cdot 5$ :



Exercise:

Do the same as above for these expressions:

a) 
$$2 \cdot (3+4) + 5 \cdot 6$$

b) 
$$2 + 3 + 4$$

c) 
$$2+3+4+5$$

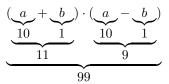
(Improvise when needed)

## Expressions with variables

If a = 5 and b = 2, then:

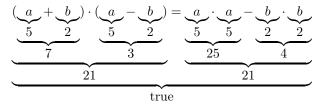
$$\underbrace{\underbrace{\underbrace{\underbrace{a}}_{5} + \underbrace{b}_{2}}_{7} \cdot \underbrace{\underbrace{a}_{5} - \underbrace{b}_{2}}_{21}}_{21}$$

If a = 10 and b = 1, then:

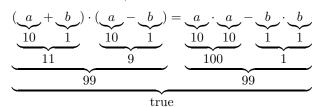


We know – by algebra, which is not for (tiny) children – that  $(a+b)\cdot(a-b)=a\cdot a-b\cdot b$  is true for all  $a,b\in\mathbb{R}$  We know – without algebra – how to test " $(a+b)\cdot(a-b)=a\cdot a-b\cdot b$ " for specific values of a and b...

If a = 5 and b = 2, then:



If a = 10 and b = 1, then:



A notation for (simultaneous) substitution:

$$((x+y)\cdot z)\begin{bmatrix}x:=a+y\\y:=b+z\\z:=c+x\end{bmatrix} = ((a+y)+(b+z))\cdot (c+x).$$

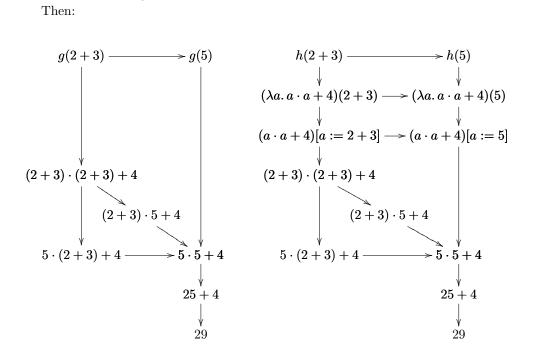
Note that  $((a+b)\cdot(a-b))[\frac{a:=5}{b:=2}] = (5+2)\cdot(5-2)$ .

#### Lambda

A named function:  $g(a) = a \cdot a + 4$ An unnamed function:  $\lambda a. a \cdot a + 4$ 

Let  $h = \lambda a \cdot a \cdot a + 4$ .

Then:



The usual notation for defining functions is like this:

$$\begin{array}{cccc} f: & \mathbb{N} & \to & \mathbb{R} \\ & n & \mapsto & 2+\sqrt{n} \end{array}$$

 $(name): (domain) \rightarrow (codomain)$  $(variable) \mapsto (expression)$ 

It creates named functions (with domains and codomains).

The usual notation for creating named functions without specifying their domains and codomains is just  $f(n) = 2 + \sqrt{n}$ .

Note that this is:

$$f$$
  $(n)$  =  $2 + \sqrt{n}$   
(name)  $(\text{(variable)})$  =  $(\text{expression})$ 

#### Functions as their graphs

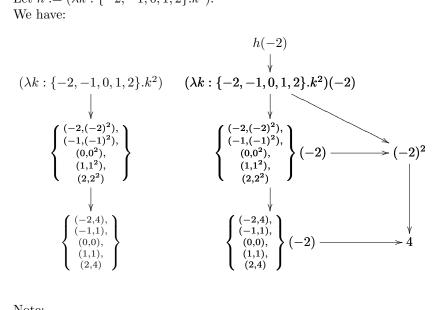
The graph of

$$\begin{array}{cccc} h: & \{-2,-1,0,1,2\} & \to & \{0,1,2,3,4\} \\ & k & \mapsto & k^2 \end{array}$$

is 
$$\{(-2,4), (-1,1), (0,0), (1,1), (2,4)\}$$

We can think that a function is its graph, and that a lambda-expression (with domain) reduces to a graph. Then  $h = \{(-2, 4), (-1, 1), (0, 0), (1, 1), (2, 4)\}$ and  $h(-2) = \{(-2,4), (-1,1), (0,0), (1,1), (2,4)\}(-2) = 4.$ 

Let  $h := (\lambda k : \{-2, -1, 0, 1, 2\}.k^2).$ We have:



Note:

the graph of  $(\lambda n : \mathbb{N}.n^2)$  has infinite points,

the graph of  $(\lambda n : \mathbb{N}.n^2)$  is an infinite set,

the graph of  $(\lambda n : \mathbb{N}.n^2)$  can't be written down explicitly without '...'s...

Mathematicians love infinite sets.

Computers hate infinite sets.

For mathematicians a function is its graph

(↑ remember Discrete Mathematics!)

For computer scientists a function is is a finite program.

Computer scientists love ' $\lambda$ 's!

I love things like this:  $\left\{ \begin{smallmatrix} (3,30),\\ (4,40) \end{smallmatrix} \right\}(3)=30$ 

## Types (introduction)

Let:

$$A = \{1, 2\}$$
  
 $B = \{30, 40\}.$ 

If  $f:A\to B$ , then f is one of these four functions:

$$\substack{1 \mapsto 30 \\ 2 \mapsto 30}, \, \substack{1 \mapsto 30 \\ 2 \mapsto 40}, \, \substack{1 \mapsto 40 \\ 2 \mapsto 30}, \, \substack{1 \mapsto 40 \\ 2 \mapsto 40}$$

or, in other notation,

$$\left\{ \begin{array}{c} (1,30) \\ (2,30) \end{array} \right\}, \left\{ \begin{array}{c} (1,30) \\ (2,40) \end{array} \right\}, \left\{ \begin{array}{c} (1,40) \\ (2,30) \end{array} \right\}, \left\{ \begin{array}{c} (1,40) \\ (2,40) \end{array} \right\}$$

which means that:

$$f \in \left\{ \left. \left\{ \begin{smallmatrix} (1,30) \\ (2,30) \end{smallmatrix} \right\}, \left\{ \begin{smallmatrix} (1,30) \\ (2,40) \end{smallmatrix} \right\}, \left\{ \begin{smallmatrix} (1,40) \\ (2,30) \end{smallmatrix} \right\}, \left\{ \begin{smallmatrix} (1,40) \\ (2,40) \end{smallmatrix} \right\} \right\}$$

Let's use the notation " $A \rightarrow B$ " for "the set of all functions from A to B".

$$\begin{array}{l} \text{Then } (A \to B) = \Big\{ \left\{ \begin{smallmatrix} (1,30) \\ (2,30) \end{smallmatrix} \right\}, \left\{ \begin{smallmatrix} (1,30) \\ (2,40) \end{smallmatrix} \right\}, \left\{ \begin{smallmatrix} (1,40) \\ (2,30) \end{smallmatrix} \right\}, \left\{ \begin{smallmatrix} (1,40) \\ (2,40) \end{smallmatrix} \right\} \Big\} \\ \text{and } f: A \to B \\ \text{means } f \in (A \to B). \end{array}$$

In Type Theory and  $\lambda$ -calculus "a:A" is pronounced "a is of type A", and the meaning of this is roughly " $a \in B$ ".

(We'll see the differences between ' $\in$ ' and ':' (much) later).

Note that:

1. if  $f: A \to B$  and a: A then f(a): B

2. if a:A and b:B then  $(a,b):A\times B$ 

3. if  $p: A \times B$  then  $\pi p: A$  and  $\pi' p: B$ , where

' $\pi$ ' means 'first projection' and

' $\pi'$ ' means 'second projection';

if p = (2, 30) then  $\pi p = 2$ ,  $\pi' p = 30$ .

If  $p: A \times B$  and  $g: B \to C$ , then:

$$\underbrace{(\pi\underbrace{p}_{:A\times B}, \underbrace{g}_{:B\to C}\underbrace{(\pi'\underbrace{p}_{:A\times B})))}_{:A\times C}$$

## Typed $\lambda$ -calculus: trees

$$A = \{1, 2\}$$

$$B = \{3, 4\}$$

$$C = \{30, 40\}$$

$$D = \{10, 20\}$$

$$A \times B = \left\{ \begin{pmatrix} (1, 3), (1, 4), \\ (2, 3), (2, 4) \end{pmatrix} \right\}$$

$$B \to C = \left\{ \begin{pmatrix} (3, 30), \\ (4, 30) \end{pmatrix}, \begin{pmatrix} (3, 30), \\ (4, 40) \end{pmatrix}, \begin{pmatrix} (3, 40), \\ (4, 40) \end{pmatrix} \right\}$$

If we know [the values of] a, b, f then we know [the value of] (a, f(b)). If (a, b) = (2, 3) and  $f = {(3,30), (4,40)}$  then (a, f(b)) = (2, 30).

$$\frac{(a,b)}{\frac{a}{a}} \pi \frac{\frac{(a,b)}{b} \pi'}{\frac{f(b)}{(a,f(b))}} \text{ pair } \frac{\frac{(2,3)}{3} \pi'}{\frac{2}{3} \pi'} \frac{\left\{(3,30),(4,40)\right\}}{\frac{30}{30}} \text{ pair } \text{app}$$

If we know the types of a, b, f we know the type of (a, f(b)). If we know the types of p, f we know the type of  $(\pi p, f(\pi'p))$ . If we know the types of p, f we know the types of  $(\lambda p : A \times B.(\pi p, f(\pi'p)))$ .

$$\frac{(a,b):A\times B}{a:A} \stackrel{\pi}{\pi} \frac{\underbrace{(a,b):A\times B}_{} \stackrel{\pi'}{\pi'} \quad f:B\to C}{f(b):C} \text{ pair}} \text{ app}$$

$$\frac{p:A\times B}{\frac{\pi p:A}{(\pi p,f(\pi'p)):A\times C}}\frac{\frac{p:A\times B}{\pi'p:B}}{\frac{f(\pi'p):C}{(\pi p,f(\pi'p)):A\times C}}\operatorname{pair}(\lambda p:A\times B.(\pi p,f(\pi'p))):A\times B\to A\times C}$$

### Types: exercises

Let: 
$$A = \{1, 2\}$$
 
$$B = \{3, 4\}$$
 
$$C = \{30, 40\}$$
 
$$D = \{10, 20\}$$
 
$$f = \left\{ \substack{(3,30), \\ (4,40)} \right\}$$
 
$$g = \left\{ \substack{(1,10), \\ (2,20)} \right\}$$
 Note that  $f: B \to C$  and  $g: A \to D$ .

- a) Evaluate  $A \times B$ .
- b) Evaluate  $A \to D$ .
- c) Evaluate  $(\pi p, f(\pi'p))$  for each of the four possible values of  $p: A \times B$ .
- d) Evaluate  $\lambda p: A \times B.(\pi p, f(\pi'p))$ .
- e) Is this true?

$$(\lambda p : A \times B . (\pi p, f(\pi'p))) = \begin{cases} ((1,3), (1,30)), \\ ((1,4), (1,40)), \\ ((2,3), (2,30)), \\ ((2,4), (2,40)) \end{cases}$$

- f) Let p = (2,3). Evaluate  $(g(\pi p), f(\pi' p))$ .
- g) Check that if  $p: A \times B$  then  $(g(\pi p), f(\pi'p)): D \times C$ .
- h) Check that

$$(\lambda p : A \times B . (g(\pi p), f(\pi' p))) : A \times B \to D \times C.$$

i) Evaluate  $(\lambda p: A \times B.(g(\pi p), f(\pi'p)))$ .

## Type inference

Here is another notation for checking types:

$$(\lambda \underbrace{p}_{:A \times B} : A \times B. \ (\pi \underbrace{p}_{:A \times B}, \underbrace{f}_{:B \to C} \underbrace{(\pi' \underbrace{p}_{:A \times B})))$$

$$\vdots A \times B \xrightarrow{:A \times C}$$

$$\vdots A \times B \xrightarrow{:A \times C}$$

Compare it with:

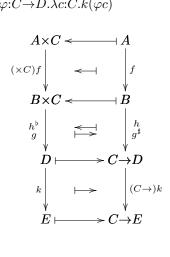
$$\frac{p:A\times B}{\frac{\pi p:A}{(\pi p,f(\pi'p)):A\times C}}\pi \frac{\frac{p:A\times B}{\pi'p:B}\pi'}{\frac{f(\pi'p):C}{(\pi p,f(\pi'p)):A\times C}} \text{ pair } \\ \frac{(\lambda p:A\times B.(\pi p,f(\pi'p))):A\times B\to A\times C}{\lambda}$$

#### Exercise:

Infer the type of each of the terms below (at the right of the ':='). Use the two notations above.

The types of f, g, h, k are shown in the diagram below.

- a)  $(\times C)f := \lambda p:A \times C.(f(\pi p), \pi' p)$ b)  $h^{\flat} := \lambda q:B \times C.(h(\pi q))(\pi' q)$ c)  $g^{\sharp} := \lambda b:B.\lambda c:C.g(b,c)$ d)  $(C \rightarrow)k := \lambda \varphi:C \rightarrow D.\lambda c:C.k(\varphi c)$



#### Term inference

Exercises:

$$\cfrac{\cfrac{p:A\times C}{:A}}{\cfrac{:B}{:B\times C}}\pi$$
 app 
$$\cfrac{p:A\times C}{:C}\pi'$$
 
$$\cfrac{:B\times C}{:A\times C\to B\times C}\lambda$$

$$\frac{q:B\times C}{:C} \ \pi' \quad \frac{\begin{array}{c} q:B\times C \\ \hline :B \end{array} \ \pi \quad h:B\to (C\to D) \\ \hline \begin{array}{c} :C\to D \\ \hline \vdots \\ \hline \\ :B\times C\to D \end{array} \ \text{app}$$

$$\begin{array}{c|c} \frac{b:B \quad c:C}{:B\times C} \text{ pair } & g:B\times C\to D \\ \hline & \vdots \\ \hline & \vdots \\ \hline & \vdots \\ \hline & \vdots \\ B\to (C\to D) \end{array} \text{ app }$$

#### Term inference: answers

$$\frac{p:A\times C}{\frac{\pi p:A}{p:A}}\frac{\pi}{f:A\to B} \text{ app } \frac{p:A\times C}{\pi'p:C}\frac{\pi'}{\pi'}$$

$$\frac{f(\pi p):B}{(f(\pi p),\pi'p):B\times C} \text{ pair }$$

$$\frac{f(\pi p):B\times C}{\lambda p:A\times C.(f(\pi p),\pi'p):A\times C\to B\times C}\lambda$$

$$\frac{q:B\times C}{\pi'q:C}\frac{\pi'}{\pi'}\frac{\frac{q:B\times C}{\pi q:B}}{\frac{\pi q:B}{h(\pi q):C\to D}}\frac{\pi}{h(\pi q):C\to D} \text{ app }$$

$$\frac{h(\pi q)(\pi'q):D}{\lambda q:B\times C.h(\pi q)(\pi'q):B\times C\to D}\lambda$$

$$\frac{b:B \quad c:C}{(b,c):B\times C} \text{ pair } \frac{g:B\times C\to D}{g:B\times C\to D} \text{ app }$$

$$\frac{g(b,c):D}{\lambda c:C.g(b,c):C\to D}\lambda$$

$$\frac{\lambda b:B.\lambda c:C.g(b,c):C\to D}{\lambda b:B.\lambda c:C.g(b,c):B\to (C\to D)}\lambda$$

$$\frac{c:C \quad \varphi:C\to D}{\mu c:C.k(\varphi c):C\to E} \text{ app }$$

$$\frac{k(\varphi c):E}{\lambda c:C.k(\varphi c):(C\to E)}\lambda$$

#### Contexts and '⊢'

Suppose that A, B, C are known, and are sets.

(Jargon: "fix sets A, B, C".)

Then this

$$\underbrace{p: A \times B, f: B \to C}_{\text{"context": a series of declarations like}} \vdash \underbrace{f(\pi'p): C}_{term:type}$$

Means:

"In this context the expression expr makes sense, is not error, and its result is of type type."

Note that calculating  $f(\pi'p)$  yields error if we do not know the values of f or p.

What happens if we add contexts to each term:type in a tree? The two bottom nodes in

$$\frac{p:A\times B}{\frac{\pi p:A}{(\pi p,f(\pi'p)):A\times C}}\pi \frac{\frac{p:A\times B}{\pi'p:B}\pi'}{\frac{f(\pi'p):C}{(\lambda p:A\times B.(\pi p,f(\pi'p))):A\times C}}\operatorname{pair}\frac{(\pi p,f(\pi'p)):A\times C}{\lambda}$$

would become:

$$f: B \to C, p: A \times B \vdash (\pi p, f(\pi'p)): A \times C$$
$$f: B \to C \vdash (\lambda p: A \times B.(\pi p, f(\pi'p))): A \times B \to A \times C$$

After the rule ' $\lambda$ ' the 'p' is no longer needed!

If we add the contexts and omit the types, the tree becomes:

$$\frac{\frac{p \vdash p}{p \vdash \pi p} \ \pi}{\frac{p \vdash p}{f \vdash \pi p} \ \pi'} \frac{f \vdash f}{f \vdash f} \ \mathsf{app}} \quad \underset{f}{\mathsf{pair}} \quad \frac{[p]^1}{\frac{\pi' p}{f}} \frac{\pi'}{f} \frac{f}{f(\pi' p)} \ \mathsf{app}}{\frac{\pi p}{f} \vdash (\lambda p : A \times B . (\pi p, f(\pi' p)))} \ \lambda \qquad \leadsto \qquad \frac{[p]^1}{\frac{\pi p}{f}} \pi \frac{\frac{[p]^1}{\pi' p} \ \pi'}{f(\pi' p)} \frac{f}{\mathsf{pair}} \ \mathsf{app}} \frac{\mathsf{app}}{(\lambda p : A \times B . (\pi p, f(\pi' p)))} \ \lambda ; 1$$

Notational trick:

below the bar ' $\lambda$ ; 1' the value of p is no longer needed; we say that the p is "discharged" (from the list of hypotheses) and we mark the 'p' on the leaves of the tree with ' $[\cdot]^1$ '; a ' $[\cdot]^1$ ' on a hypothesis means: "below the bar ' $\lambda$ ; 1' I am no longer a hypothesis".

#### Curry-Howard: introduction

We are learning a system called "the simply-typed  $\lambda$ -calculus (with binary products)" — system  $\lambda 1$ , for short.

In  $\lambda 1$  in its fullest form, its objects are trees of '...  $\vdash term : type$ 's, but we saw (evidence) that we can:

- reconstruct the full tree from just the 'term : type's,
- write just ': type's (except on the leaves, to get the var names),
- reconstruct the full tree from just the bottom 'term : type'...

For example, we can reconstruct the whole tree, with contexts, from:

$$\frac{[p:A\times B]^1}{:A} \stackrel{\pi}{\times} \frac{\frac{[p:A\times B]^1}{:B} \stackrel{\pi'}{\pi'} \quad f:B\to C}{:C} \text{ app}}{:A\times C} \stackrel{\text{pair}}{\xrightarrow{:A\times B\to A\times C}} \lambda$$

If we erase the terms and the ':'s and leave only the types, we get something that is strikingly similar to a tree in Natural Deduction,

$$\frac{[A \times B]^1}{A} \pi \frac{\frac{[A \times B]^1}{B} \pi' \quad B \to C}{\frac{C}{A \times B \to A \times C} \lambda} \text{ pair}$$

$$\xrightarrow{P} & E_1 \qquad \frac{P \& Q]^1}{Q} \& E_2 \qquad Q \to R \\
\frac{P \& Q}{P \& R \to P \& Q} \to I; 1$$

which talks about logic.

## Curry-Howard: Natural Deduction

The tree

$$\frac{[P\&Q]^1}{\frac{P}{P}} \&E_1 \qquad \frac{[P\&Q]^1}{Q} \&E_2 \qquad Q \to R \\ \frac{P}{P} \&E_1 \qquad \frac{R}{P\&Q} \&I \\ \frac{P\&Q}{P\&R \to P\&Q} \to I;1$$

is in  $ND_{\&\to}$  (or in  $IPL_{\&\to}$ ), the fragment of Natural Deduction (or intuitionistic predicate logic) that only has the connectives & and  $\to$ .

Its rules are:

$$\frac{P Q}{P \& Q} \& I \qquad \frac{P \& Q}{P} \& E_1 \qquad \frac{P \& Q}{Q} \& E_2$$

$$P [Q]^1$$

$$\vdots$$

$$\frac{R}{Q \to R} \to I \qquad \frac{P P \to Q}{Q} \to E$$

New rules (for  $\top$ ,  $\bot$ ,  $\lor$ ): (not yet — see the whiteboard for 20170418)

## Planar Heyting Algebras

We read sections 1–7 of:

http://angg.twu.net/LATEX/2017planar-has.pdf

Let 
$$B = {32 \atop 22 \atop 20 \atop 11 \atop 10 \atop 00} \atop 10 \atop 00}$$
 .

Exercises:

Calculate and represent in positional notation when possible:

- a)  $\lambda lr:B.l$
- b)  $\lambda lr:B.r$
- c)  $\lambda lr:B.(l \leq 1)$
- d)  $\lambda lr:B.(r \geq 1)$
- e)  $\lambda lr:B.lr \leq 11$
- f)  $\lambda lr:B.lr\&12$
- g)  $\lambda lr:B$ . valid  $(\langle l+1,r\rangle)$
- h)  $\lambda lr:B.lr$  leftof 11
- i)  $\lambda lr$ :B.lr leftof 12
- j)  $\lambda lr:B.lr$  above 11
- k)  $\lambda lr:B$ . ne (lr)
- 1)  $\lambda lr:B$ . nw (lr)
- m)  $20 \rightarrow 11$
- n)  $02 \rightarrow 11$
- o)  $22 \to 11$
- p)  $00 \to 11$
- q)  $\lambda lr:B.\neg lr$
- r)  $\lambda lr:B.\neg\neg lr$
- s)  $\lambda lr$ : $B.lr = \neg \neg lr$

#### Algebraic structures

A ring is a 6-uple

$$(R, 0_R, 1_R, +_R, -_R, \cdot_R)$$

where  $R, 0_R, \ldots, \cdot_R$  have the following types,

$$\begin{split} &R \text{ is a set,} \\ &0_R \in R, \\ &1_R \in R, \\ &+_R : R \times R \to R, \\ &-_R : R \to R \text{ (unary minus),} \\ &\cdot_R : R \to R, \end{split}$$

and where the components obey these equations  $(\forall a, b, c \in R)$ :

$$a+0_R = 0_R + a = a$$
,  $a+b = b+a$ ,  $a+(b+c) = (a+b)+c$ ,  $a+(-a) = 0$ ,  $a \cdot 1_R = 1_R \cdot a = a$ ,  $a \cdot b = b \cdot a$ ,  $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ ,  $a \cdot (b+c) = a \cdot b + a \cdot c$ .

A proto-ring is a 6-uple  $(R, 0_R, 1_R, +_R, -_R, \cdot_R)$  that obeys the typing conditions of a ring.

A ring is a proto-ring plus the assurance that it obeys the ring equations.

A proto-Heyting Algebra is a 7-uple

$$H = (\Omega, <_H, \top_H, \bot_H, \&_H, \lor_H, \to_H)$$

in which:

$$\begin{array}{l} \Omega \text{ is a set (the "set of truth values"),} \\ \leq_H \subset \Omega \times \Omega \text{ (partial order),} \\ \top_H \in \Omega, \\ \bot_H \in \Omega, \\ \&_H : \Omega \times \Omega \to \Omega \\ \lor_H : \Omega \times \Omega \to \Omega \\ \to_H : \Omega \times \Omega \to \Omega \end{array}$$

Sometimes we add operations '¬' and  $\leftrightarrow$  to a (proto-)HA H,

$$H = (\Omega, \leq_H, \top_H, \bot_H, \&_H, \lor_H, \rightarrow_H, \lnot_H, \leftrightarrow_H)$$

by defining them as 
$$\neg P := P \to \bot$$
 and  $P \leftrightarrow Q := (P \to Q)\&(Q \to P)$  (i.e.,  $\neg_H P := P \to_H \bot_H$  and  $P \leftrightarrow_H Q := (P \to_H Q)\&_H(Q \to_H P)$ ).

This abuse of language is very common: R "="  $(R, 0_R, 1_R, +_R, -_R, \cdot_R)$ .

# Protocategories

A protocategory is a 4-uple

$$\mathbf{C} = (\mathbf{C}_0, \mathrm{Hom}_{\mathbf{C}}, \mathrm{id}_{\mathbf{C}}, \circ_{\mathbf{C}})$$

where

 $\mathbf{C}_0$  is a set (more precisely a "class"),  $\mathrm{Hom}_{\mathbf{C}}: \mathbf{C}_0 \times \mathbf{C}_0 \to \mathbf{Sets},$  $\mathrm{id}_{\mathbf{C}}(A) \in \mathrm{Hom}_{\mathbf{C}}(A,A),$ 

 $(\circ_{\mathbf{C}})_{ABC}: \operatorname{Hom}_{\mathbf{C}}(A, C) \times \operatorname{Hom}_{\mathbf{C}}(A, B) \to \operatorname{Hom}_{\mathbf{C}}(A, C).$ 

A *categoru* is a protocategory plus the assurance that identities behave as expected and composition is associative.

Sometimes we add an operation ';' to a category,

$$\mathbf{C} = (\mathbf{C}_0, \operatorname{Hom}_{\mathbf{C}}, \operatorname{id}_{\mathbf{C}}, \circ_{\mathbf{C}}, ;_{\mathbf{C}})$$

where ';' is the composition in other order:  $f \circ g = g$ ; f.