Harper 3

So far, we're developing <u>computational</u>
Type Theory — types are behavioral specifications

key setup
principal forms of judgment
exact type equality

$A = A'$ (A type means $A = A$)

technical device
could define A type
then define when equal

more technically efficient
(less redundant)

when reflexive, happens to be a Type

exact member equality $M = M' \in A$

($M \in A$ means $M \doteq M \in A$)

alternatively: $M \doteq_A M'$ (mix-fix notation)

→ expressions do <u>not</u> intrinsically have a type
same program can ~~specify~~ satisfy many specifications

e.g. $M \in Nat \to Nat$

$M \in Primes \to Primes$

equality depend on what type you're talking about

$\lambda a.\, a \doteq \lambda a.\, |a| \in Nat \to Nat$

but not true for

$\times\ \lambda a.\, a \doteq \lambda a.\, |a| \in Int \to Int$

Definitions of various types

Bool, Nat } inductive types

1 aka Unit - exercise <> val

$A_1 \times A_2$ products

$A_1 \to A_2$ functions

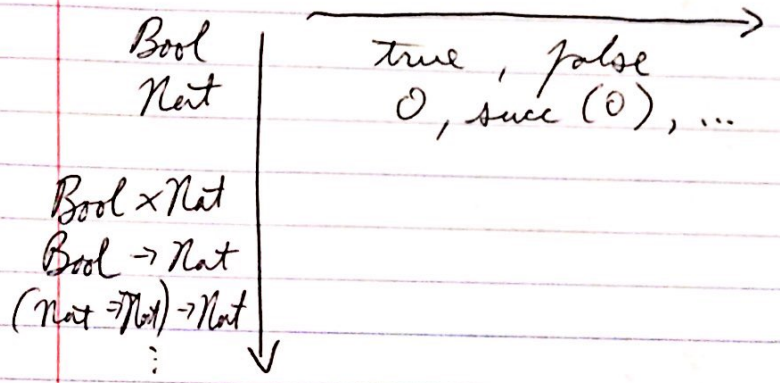
0 aka Void (empty type - exercise)
no entry forms
has nullary case analysis
cas {} (M)

$A_1 + A_2$ exercise

$1 \cdot M,\ 2 \cdot M,$ case $\{a_1 . M_1 ;\ a_2 . M_2\}$ (M)

meaning of $\times$, $\to$
defined ~~determined~~ in terms of A_1, A_2

two axes



types

vertical induction, horizontal

dependent case

$a : A_1 \times A_2$ aka dep product aka sum aka Σ-type $\Sigma a{:}A_1.A_2$

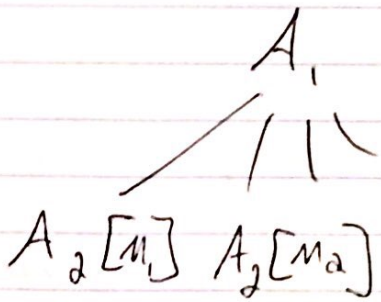
$a : A_1 \to A_2$ dep function aka Π-type $\Pi a{:}A_1.A_2$

to define $a : A_1 \times A_2$

must already have

A_1 defined

$A_2[M_1/a]$ for every $M_1 \in A_1$ infinitely wide not inf. deep



defind in terms of A_1 with
and all instances $A_2[M_1/a]$

A_1 Type

$a : A_1 >> A_2$ Type

trouble w/ quantification across all possible types
point of universes to ensure these are well-defined

Crucial point : everything defined in terms of
evaluation

lots of types arise that don't (as naturally)
in formal setting
example : intersection, union types

back to lecture 1

type theory a theory of computation

(simple - not incl effects, etc)

(program specification)

→ Brouwer as a means to give a notion of truth for logical propositions

"Propositions as types"

aka semantic correspondence

(realizability)

about how programs run

not formal derivations

capule summary

how do we explain these?

- φ prop
- φ true

ϕ prop proposition is a specification
or problem

e.g. there are infinitely many problems

ϕ true solution to problem
satisfies spec
realizer

ϕ^* type $\qquad \phi^* \doteq \phi^*$ (?) $\in$ more troublesome

ϕ^* inhabited (can find a program)
there is
$\exists M \in \phi^*$ get in passant $\exists$
$M \doteq M' \in \phi^*$

notion of <u>type</u> is <u>primary</u>
effective construction, computation

<u>more extensive</u> than merely propositions
merely logic

Typically

$\top^*$	1 aka unit	(trivially true)
$\bot^*$	0 aka Void	
$(\phi_1 \wedge \phi_2)^*$	$\phi_1^* \times \phi_2^*$	
$(\phi_1 \vee \phi_2)^*$	$\phi_1^* + \phi_2^*$	controversial

nota bene: constructive notion of truth

implication

$(\phi_1 \supset \phi_2)^*$ $\quad \phi_1^* \to \phi_2^*$

$(\forall a{:}A.\ \phi_a)^*$ $\quad a{:}A \to \phi_a^*$

$\equiv$

need notion of what we are quantifying over (e.g. Nat)

$(\exists a{:}A.\ \phi_a)^*$ $\quad a{:}A \times \phi_a^*$

N.B. controversial

to prove something exists, need to show it "constructive existence"

(in M: Hilf paper)

$(\forall a{:}A\ \exists b{:}B\ R(a,b))$ "R is total"

$\supset$ ($\longrightarrow$)

$(\exists f{:}A\to B\ \forall a{:}A.\ R(a,f(a)))$ f: "choice function"

true

F: $(a{:}A \longrightarrow (b{:}B \times R(a,b)))$

$\to (f.\ A\to B \times (a{:}A \to R(a,f(a))))$ axiom of choice

$f \triangleq \lambda a.(F(a)\cdot 1)$ "fact" of choice (theorem in this theory)

what to do about equality

$(M =_A M')^*$ (?)

How do we interpret <u>equality</u> as a <u>type</u>?

Introduce [Equality Types]

$Eq_A(M_1, M_2) \doteq_{\mathbb{B}} Eq_A(M_1', M_2')$

iff $A = A'$ $M_1 \doteq M_1' \in A$ $M_2 \doteq M_2' \in A$

$M \in Eq_A(M_1, M_2)$ iff

$M \Downarrow *$ and

$M_1 \doteq M_2 \in A$

subsingleton (has at most one element)

↳ idea: "at most true" → nothing about why it's true

$Eq_{Nat}(2,3)$ - uninhabited

$Eq_{Nat}(2, 1+1)$ - inhabited

$M \in Eq_{\to}(M_1; M_2)$ iff $M_1 \doteq M_2 \in A$

corresponds to

$(M_1 =_A M_2)^* \simeq Eq_A(M_1; M_2)$

"works"

Exercise. show that $Eq_A(-,-)$ is

the <u>least</u> reflexive (binary) relation

1) reflexive $* \in Eq_A(M, M)$ whenever $M \in A$

2) (if R is reflexive)

to show T.S. $Eq_A(M,N) \to R(M,N)$ "equality induction"

suffices to show S.T.S. R is reflexive $a : A \to \_ \in R(a,a)$

Formalisms - formal type theory is inductively defined by a collection of rules for deriving

$\Gamma \vdash A \text{ type}$ $\qquad$ $\dfrac{\Gamma \vdash M = M' : A}{\Gamma \vdash A \equiv A'}$ "definitional equality"

$\Gamma \vdash M : A$ $\qquad$ $\Gamma \vdash M \equiv M' : A$

↳ choice is somewhat up for grabs

$$\frac{}{\Gamma, x:A, \Gamma' \vdash x : A}$$

$$\frac{\Gamma \vdash A_1 \text{ type} \quad \Gamma \vdash A_2 \text{ type}}{\Gamma \vdash A_1 \times A_2 \text{ type}}$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash A \equiv A'}{\Gamma \vdash M : A'}$$

$$\frac{\Gamma \vdash M_1 : A_1 \quad \Gamma \vdash M_2 : A_2}{\Gamma \vdash \langle M_1, M_2 \rangle : A_1 \times A_2}$$

$$\frac{\Gamma \vdash M : A_1 \times A_2}{\Gamma \vdash M \cdot i : A_i} \; (i = 1, 2)$$

$$\frac{\Gamma \vdash M_1 : A_1 \quad \Gamma \vdash M_2 : A_2}{\Gamma \vdash \langle M_1, M_2 \rangle \cdot i \equiv M_i : A_i} \checkmark$$

$$\frac{\Gamma \vdash M : A_1 \times A_2}{\Gamma \vdash \langle M \cdot 1, M \cdot 2 \rangle = M : A_1 \times A_2} \; \text{?} \; \text{(sometimes taken)}$$

There exist a completion of rule

$\Gamma \vdash A \equiv A'$

inherently computable (recursively) enumerable

(Harper cont)

Design requirements

- all judgments should be decidable.
  (complexity doesn't come up, can be bad)
- type checking decidable
  ⇔
- defn'l equivalence dec. (for some choice of def. eq.)

structural properties of entailment

formalism is a useful(?) approximation to the truth (has to be an approximation)

can apply correspondence in a formal sense
("exactly same" as rules for formal proof with elements removed)

via formal correspondence w/ formal logics
then type checking same as formal proof checking
"derivation checking"

How to axiomatize equality?
(formalize)

First cut (violates precepts of decidability)

"equality reflection"

$$\frac{\Gamma \vdash M : Eq_A(M_1; M_2)}{\Gamma \vdash M_1 \equiv M_2 : A} \quad (ER)$$

problematic ✗

$$\frac{\Gamma \vdash M_1 \equiv M_2 : A}{\Gamma \vdash refl_A(M_1) : Eq_A(M_1; M_2)}$$

↖ "ETT" (this approach)

have to reason backwards
(need to do if proof search
for 'M' on the top)
threatens or compromises decidability

What do you do?
↳ axiomatize least reflexive relation
(but doesn't mean the same thing
in particular for function types)