Harper 1.1

type theory from a computational perspective

- Martin-Löf : Constructive mathematics and computer programming
- Constable et. al. : NuPRL

<u>Plan</u>

1. develop type theory starting w/ computation
   → theory of truth (based on proof)
2. contrast w formalisms
   → theory of ~~pf~~ formal proof

R! a sufficiently expressive PL is a foundation for all of mathematics

Idea: Start w a programming language

- Deterministic operational semantics / transition system
- Assume: some idea of abstract syntax w/ binding and scope — subst for vars
- forms of expressions E
- two judgment forms E val means E is fully evaluated
- $E \mapsto E'$ means one step of simplification of E
- Derived notion of $E \Downarrow E_0$ means $E \mapsto^* E_0$ and E_0 val

Consts: tt val, ff val, Bool val

e.g.

$$\frac{E \mapsto E'}{\text{if}(E_1; E_2)(E) \mapsto \text{if}(E_1; E_2)(E')} \qquad \left\{ \frac{}{\begin{array}{l}\text{if}(E_1; E_2)(tt) \mapsto E_1 \\ \quad\text{"} \quad\quad (ff) \mapsto E_2\end{array}} \right\}$$

TYPES ARE SPECIFICATIONS OF PROGRAM BEHAVIOR

two principal forms of judgment (expressions of knowledge)

A type, $M \in A$ $\left\{ \begin{array}{l} 1. \text{ Behavioral (not structural)} \\ 2. \text{ both } M \text{ and } A \text{ here are programs} \end{array} \right.$

e.g. Bool type, $tt \in$ Bool, $ff \in$ Bool "true by definition"

fact not defn $\left\{ \begin{array}{l} \text{if } M \in \text{Bool and } M_1, M_2 \in A \text{ and } A \text{ type} \\ \text{then if}(M_1, M_2)(M) \in A \end{array} \right.$

E: $\text{if}(17; \text{Garbage})(ff) \in \text{Nat}$
runs by $\mapsto$ simplifies to $17 \in \text{Nat}$

E: $\text{if}(\text{Nat}; \text{Bool})(M)$ type when $M \in$ Bool
b/c any outcome for M induces a simplification to a type.

Harper 1.2 / $\underline{\underline{E}}$ $\quad$ $if(17; tt)(M) \in if(Nat; Bool)(M)$ $\qquad$ $\underline{\underline{R}}$ Specifications/types are programs

- Key idea : type-indexed families of types aka dependent types

e.g. $Seq(n)$ type when $n \in Nat$

$n : nat \gg seq$ type

a family of types indexed by a type $\longrightarrow$ ~~the~~ HYPOTHETICAL JUDGMENT / GENERAL JUDGMENT (Martin-Löf)

$f \in n{:}Nat \longrightarrow Seq(\ )$ (NuPRL notation)

$(\Pi n{:}Nat.\ seq(n))$

critical idea : FUNCTIONALITY

families of types, of elements must respect equality of indices

$\underline{\underline{E}}$ $\quad$ $Seq(2+2)$ "same as" $seq(4)$

- $Seq(if(17;18)(M))$ "same as" $if(seq(17); seq(18))(M)$

$a{:}Bool \gg \left(Seq(if(17;18)(a)) \text{ "same as" } if(seq(17); seq(18))(a) \right)$

---

revise our principle forms of judgment

Judgments

A type $\rightsquigarrow$ $A \doteq A'$ (Exact equality of types)

$M \in A$ $\rightsquigarrow$ $M \doteq M' \in A$ (exact equality of elements at type) "equisatisfaction"

dependent!

* "$\doteq$" is the "same as" from earlier

- $\underline{\underline{E}}$ $\quad$ not $2 \doteq 4 \in Nat$
  is $2 \doteq 4 \in Nat/mod\ 2$

exact equality of elements is a three part judgment, not a chain of two judgments

Scanned by CamScanner

Harper 1.3 (

intention is that if $M \doteq M' \in A$ and $A \doteq A'$ then $M \doteq M' \in A'$

Meaning Explanations aka Semantics (computational)

there exists A_0, ... Conjunctive

1. $A \doteq A'$ means $A \Downarrow A_0$ $A' \Downarrow A_0'$ $\underbrace{A_0 \doteq_0 A_0'}$

A_0 and A_0' are equal type-values
"equal canonical types" (Martin-Löf)

$\underline{E}$ by defn $\underset{val}{Bool} =_0 \underset{val}{Bool}$ i.e. Bool type

2. $M \doteq M' \in A$ means, where A type (i.e. $A \Downarrow A_0$ $A_0 \doteq_0 A_0$),

means $M \Downarrow M_0$ & $M' \Downarrow M_0'$ & $M_0 =_0 M_0' \in A_0$ (equal values in a type value)

3. $a : A \gg B \doteq B'$ means if $M \doteq M' \in A$ then $B[M/a] \doteq B'[M'/a]$ ("functionality")

Check $a : A \gg B$ type means $M' \doteq M' \in A$ implies $B[M/a] \doteq B[M'/a]$

4. $a : A \gg N \doteq N' \in B$ means if $M \doteq M' \in A$ then $N[M/a] \doteq N'[M'/a] \in B[M/a]$ $(\doteq B[M'/a])$

(assuming $a : A \gg B \doteq B$)

Booleans

1. $Bool \doteq_0 Bool$ i.e. $Bool\ type_0$ i.e. Bool is a type
$\Updownarrow$
names

2. $M_0 \doteq M_0' \in_0 Bool$ is the strongest relation s.t. $tt \doteq_0 tt \in Bool$ (i.e. $tt \in_0 Bool$)
$ff \doteq_0 ff \in Bool$ (i.e. $ff \in_0 Bool$)

in other words

strongest $R \subseteq Exp \times Exp$ s.t. $R(tt, tt)$, $R(ff, ff)$

a. stated conditions hold
b. nothing else

$\underline{R}$ you "must" accept this as a valid definition

Harper 1.4

Prop / Fact / Claim

- if $M \in Bool$ and A type and $M_1 \in A$ and $M_2 \in A$, then $if(M_1 ; M_2)(M) \in A$

Pf How to prove it? key: $\in Bool$ is given by a universal property - least containing $tt \in Bool$, $ff \in Bool$

- Fix A type, $M_1 \in A$, $M_2 \in A$
- if $M \in Bool$ then $if(M_1 ; M_2)(M) \in A$ / for $M \in Bool$ means $M \Downarrow M_0$ $M_0 = tt$ or $M_0 = ff$
- Sufficient to Show (STS): $if(M_1 ; M_2)(tt) \in A$
  " " " $(ff) \in A$ } cuz "if" evaluates its principle arg.

a. $if(M_1 ; M_2)(tt) \mapsto M_1 \in A$

b. $if(M_1 ; M_2)(ff) \mapsto M_2 \in A$

- L "head expansion" aka "reverse execution"

if $M' \in A$ and $M \mapsto M'$ then $M \in A$

Ex prove it using the defns in terms of eval to canonical form

1. Bool is inductively defined
2. typing is closed under head expansion.

R this lemma is like type preservation in retrograde, yet behavioral whereas type preservation is static.

Scanned by CamScanner

Harper 2.1

A type system consists of

$A \doteq A'$ (w/ A type iff $A \doteq A$)

$M \doteq M' \in A$ (w $M \in A$ iff $M \doteq M \in A$)

symmetric

transitive

if $A \doteq A'$ and $M \doteq M' \in A$ then $M \doteq M' \in A'$

$\doteq$ is a partial equivalence.

$\doteq$ we omit reflexivity.

we assert the existence of certain type systems - defined in terms of evaluation

Hypotheticals express functionality

$a : A \gg B$ type means B is a family that depends functionally on $a{:}A$

$M \doteq M' \in A$ implies $B[M/a] \doteq B[M'/a]$

A type, $a : A \gg B$ type

a presupposition

$a : A \gg N \in B$ means B is a family of elements

$M \doteq M' \in A$ implies $N[M/a] \doteq N[M'/a] \in B[M/a] \doteq B[M'/a]$

Similarly for $B \doteq B'$, $N \doteq N' \in B$

there exists a type system containing Booleans

$Bool \doteq Bool$ i.e, Bool type

$M \doteq M' \in Bool$ iff ($M \Downarrow true$ and $M' \Downarrow true$) or ($M \Downarrow false$ and $M' \Downarrow false$)

Fact if $a : Bool \gg B$ type and $M_1 \in B[true/a]$ and $M_2 \in B[false/a]$ and $M \in Bool$ then $if(M_1 ; M_2)(M) \in B[M/a]$

means you prove

Pf either $M \Downarrow true$ or $M \Downarrow false$

$\therefore M \doteq true \in Bool$ by head expansion

$\therefore M \doteq false \in Bool$ by head expansion

$$\frac{M_1 \in B[true/a]}{if(M_1 ; M_2)(M) \doteq if(M_1 ; M_2)(true) \doteq M_1 \in B[true/a] \doteq B[M/a]}$$

Harper 2.2

HW ① $if(M_1; M_2)(true) \doteq M_1 \in B[{}^{true}/_a]$

② $if(M_1; M_2)(false) \doteq M_2 \in B[{}^{false}/_a]$

③ $M \doteq if(true; false)(M) \in Bool$

RTS

if $a : Bool \gg P \in B$ then $P[{}^M/_a] \doteq if(P[{}^{true}/_a], P[{}^{false}/_a])(M)$ | "shannon expansion" (BDD's) binary decision diagrams. "pivot on M"

E

∃ type system containing natural numbers

$Nat \doteq Nat$ gives us an induction principle.

$M \doteq M' \in Nat$ is strongest s.t.

either $M \Downarrow 0, M' \Downarrow 0$

or $M \Downarrow succ(n), M' \Downarrow succ(n')$

w/ $n \doteq n' \in Nat$

Nat is given axiomatically

$$\frac{}{0\ Nat} \qquad \frac{}{succ(M)\ Nat}$$

notice: M is anything, needn't be a Nat.

consider $fix(a, succ(a)) \mapsto \underbrace{succ(fix(a, succ(a)))}_{\omega} \mapsto$!!

COINDUCTION

ω, a point at infinity, inhabits the greatest solution to specification:

if $M \doteq M' \in Nat$ then $M \Downarrow 0, M' \Downarrow 0$ or

$M \Downarrow succ(N)\ M' \Downarrow succ(N')\ N \doteq N' \in CoNat$

Check HW $\omega \notin Nat$

if $M \mapsto M'$, $rec(M_0; a, b.M_1)(M)$

$\mapsto rec(M_0; a, b.M_1)(M')$

$R(0) \mapsto M_0$

$R(succ(M)) \mapsto M_1[M, R(M)/a, b]$

predicate (checking if basecase) result of recursive call.

Fact suppose $a : Nat \gg B\ type$ $M_0 \in B[{}^0/_a]$ $a : Nat, b : B \gg M_1 \in B[{}^{succ(a)}/_a]$

then $M \in Nat$ then $R(M) \in B[{}^M/_a]$

Pf ① $M \Downarrow 0$ $M \doteq 0 \in Nat$ $M_0 \in B[{}^0/_a] \doteq B[{}^M/_a]$

$R(M) \doteq R(0) \doteq M_0$

i.e. $R(M) \in B[{}^M/_a]$

② $M \Downarrow succ(N)$ IH: $R(N) \in B[{}^N/_a]$

⟶ HW finish proof, a lot like conditionals

Harper 2.3

R type systems are about specifying protocols to be obeyed

Products

$A_1 \times A_2 \doteq A_1' \times A_2'$ iff $A_1 \doteq A_1'$ $A_2 \doteq A_2'$

$M \doteq M' \in A_1 \times A_2$ iff $M \Downarrow \langle M_1, M_2 \rangle$ $M_1 \doteq M_1' \in A_1$
$M' \Downarrow \langle M_1', M_2' \rangle$ $M_2 \doteq M_2' \in A_2$

Fact ← not dfn

Suppose A_1 type A_2 type, if $M \in A_1 \times A_2$ then $M \cdot 1 \in A_1$ and $M \cdot 2 \in A_2$

(where $\dfrac{M \mapsto M'}{M \cdot i \mapsto M' \cdot i}$ $\dfrac{}{\langle M_1, M_2 \rangle \cdot i \mapsto M_i} \langle i = 1,2 \rangle$)

we know $M \Downarrow \langle M_1, M_2 \rangle$ with $M_1 \in A_1$

$M \cdot 1 \mapsto^* \langle M_1, M_2 \rangle \cdot 1 \mapsto M_1 \in A_1$

Fact

(A_1, A_2 types)

if $M_1 \in A_1$ then $\langle M_1, M_2 \rangle \cdot 1 \doteq M_1 \in A_1$

no requirement on M_2

$\langle M_1, M_2 \rangle \cdot 1$

$\dfrac{}{\lambda a.M \text{ val}}$

$\dfrac{M \mapsto M'}{ap(M, M_1) \mapsto ap(M', M_1)}$

$\dfrac{}{ap(\lambda a.M_2, M) \mapsto M_2[M/a]}$

Functions

$A_1 \to A_2 \doteq A_1' \to A_2'$ iff $A_1 \doteq A_1'$ and $A_2 \doteq A_2'$

$M \doteq M' \in A_1 \to A_2$ iff $M \Downarrow \lambda a.M_2$ $M' \Downarrow \lambda a.M_2'$ $a : A_1 \gg M_2 \doteq M_2' \in A_2$

Fact if $M \in A_1 \to A_2$ and $M_1 \in A_1$ then $ap(M, M_1) \in A_2$

Pf (HW)

Fact if $M, M' \in A_1 \to A_2$ and $a : A_1 \gg ap(M, a) \doteq ap(M', a) \in A_2$ then $M \doteq M' \in A_1 \to A_2$ "function extensionality"

Pf (HW)

this fact is verified by the rule

$$\frac{\Gamma \vdash M : A_1 \to A_2 \quad \Gamma \vdash M_1 : A_1}{\Gamma \vdash ap(M, M_1) : A_2}$$

the rule is "protocol" or "syntax"

Q what is the quantifier complexity of $M \doteq M' \in Nat \to Nat$?

"$\forall M_1 \doteq M_1' \in Nat$
$\exists P_1 \doteq P_1' \in Nat$
$ap(M, M_1) \doteq ap(M', M_1') \in Nat \to N$
where $ap(M, M_1) \Downarrow P$
$ap(M', M_1') \Downarrow P'$"

answer $\forall\exists$ or Π^0_2

Scanned by CamScanner

Harper 2.4

Dependent products

$a : A_1 \times A_2 \doteq a : A_1' \times A_2'$

iff $A_1 \doteq A_1'$ $a : A_1 \gg A_2 \doteq A_2'$

A_2 is a family

$M \doteq M' \in a : A_1 \times A_2$ iff

$M \Downarrow \langle M_1 ; M_2 \rangle$ $M' \Downarrow \langle M_1' ; M_2' \rangle$ $M_1 \doteq M_1' \in A_1$

$M_2 \doteq M_2' \in A_2[M_1/a] \doteq A_2[M_1'/a]$

Dependent Functions

$a_1 : A_1 \to A_2 \doteq a : A_1' \to A_2'$ iff $A_1 \doteq A_1'$

$a : A_1 \gg A_2 \doteq A_2'$

$M \doteq M' \in a : A_1 \to A_2$ iff $M \Downarrow \lambda a . M_2$ $M' \Downarrow \lambda a_1 M_2'$

$a : A_1 \gg M_2 \doteq M_2' \in A_2(a)$

if $M_1' \doteq M_1 \in A_1$ then $M_2[M_1/a] \doteq M_2[M_1'/a] \in A_2[M_1/a]$

$\doteq A_2[M_1'/a]$

Fact ① if $M \in a : A_1 \times A_2$ then

$fst(M) \in A_1$ and $snd(M) \in A_2[fst(M)/a]$

② if $M \in a : A_1 \to A_2$ and $M_1 \in A_1$ then $ap(M, M_1) \in A_2[M_1/a]$

$(\Sigma a : A_1, A_2)$ $(\Pi a : A_1, A_2)$

Pf (HW)

Bool, Nat, $a_1 : A \times A_2$ $a : A_1 \to A_2$ are inherently computational

Harper 3.1

So far we're developing computational type theory
types are behavioral specifications.

(exact) type equality $A \doteq A'$ (A type means $A \doteq A$)

(exact) member equality $M \doteq M' \in A$ ($M \in A$ means $M \doteq M \in A$) Syntax: $M \doteq_A M$

$\longrightarrow$ expressions do not intrinsically "have" a type, the same program can satisfy many specs.

Definitions of types

Bool, Nat } inductive types

1 aka unit exercise $\langle\rangle$ val ("void" in C-like languages)

$A_1 \times A_2$ products

$A_1 \to A_2$ functions

0 aka void $\quad$ $\mathrm{case}\{\}(M)$

$A_1 + A_2$ $\quad$ $1 \cdot M$, $2 \cdot M$, $\mathrm{case}\{a_1.M_1 ; a_2.M_2\}(M)$

$a{:}A_1 \times A_2$ dependent product aka Σ-type $\Sigma a{:}A_1, A_2$: to defn $a{:}A_1 \times A_2$, must already have A_1 defnd, $A_2[M/a]$ for every $M_1 \in A_1$

A_1 / $A_2[M_1]$ $A_2[M_2]$ ...

$a{:}A_1 \to A_2$ dependent function aka Π-type $\Pi a{:}A_1, A_2$

a type theory is a theory of computation (program specification)

Brouwer: as a means to give a notion of truth for logical propositions. "PROPOSITIONS AS TYPES" "SEMANTIC CORRESPONDENCE"

logic	types	
$\top^*$	unit, 1	
$\bot^*$	void, 0	
$(\varphi_1 \wedge \varphi_2)^*$	$\varphi_1^* \times \varphi_2^*$	
$(\varphi_1 \vee \varphi_2)^*$	$\varphi_1^* + \varphi_2^*$	(NB) } constructive
$(\varphi_1 \supset \varphi_2)^*$	$\varphi_1^* \to \varphi_2^*$	impl/arrow
$(\forall a{:}A.\ \varphi_a)^*$	$a{:}A \to \varphi_a^*$	dependent arrow
$(\exists a{:}A.\ \varphi_a)^*$	$a{:}A \times \varphi_a^*$	(NB) { constructive existence
$(M =_A M)^*$	(next page)	

what does this mean? { φ prop $\longmapsto$ specification/problem $\longmapsto$ φ^* type
φ true $\longmapsto$ solution/satisfier/realizer $\longmapsto$ φ^* inhabited, i.e., there is $M \in \varphi^*$ and $M \doteq M' \in \varphi^*$

the notion of type is primary and more extensive than "mere" logic.

"construction" means a program inhabits a type (Brouwer)

fact, ~~axiom~~ of choice

logic { $\forall a{:}A\ \exists b{:}B\ R(a,b)$ "R is total"
$\supset\ \exists f{:}A \to B\ \forall a{:}A\ R(a, f(a))$ "choice function f" (the function f is a witness)

tt { a fact: $F :: (a{:}A \to (b{:}B \times R(a,b))) \to (f{:}A \to B \times (a{:}A \to R(a, f(a))))$

$f \triangleq \lambda a.(F(a) \cdot 1)$

Harper 3.2

How to interpret equality as a type?

Equality types

$Eq_A(M_1, M_2) \doteq Eq_{A'}(M_1', M_2')$ iff $A \doteq A'$ $\quad M_1 \doteq M_1' \in A$ $\quad M_2 \doteq M_2' \in A$

$M \in Eq_A(M_1, M_2)$ iff $M \Downarrow *$ and $M_1 \doteq M_2 \in A$ "subsingleton"

E $Eq_{Nat}(2,7)$ uninhabited

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

R an equation is at most true

$*$ is the trivial inhabitant

$\exists M \in Eq_A(M_1, M_2)$ iff $M_1 \doteq M_2 \in A$ where $M \Downarrow *$

$(M_1 =_A M_2)^* = Eq_A(M_1, M_2)$ "works"

HW axiomatize inductively that $Eq_A(-,-)$ is the least reflexive relation on A

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

② TS : $Eq_A(M,N) \longrightarrow R(M,N)$ "equality induction"

STS : R is reflexive $a : A \gg \_ \in R(a,b)$

Formalisms – formal type theory is inductively defined by rules for deriving

$\Gamma \vdash A \text{ type}$ $\qquad \Gamma \vdash A \equiv A$ $\qquad$ "definitional equality"

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

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

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

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

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

$$\frac{\Gamma \vdash M : A_1 \times A_2}{\Gamma \vdash M \cdot i : A_i} \langle i = 1, 2 \rangle$$

etc.

✓, have it

$$\frac{\Gamma \vdash M_1 : A_1 \quad \Gamma \vdash M_2 : A_2}{\Gamma \vdash \langle M_1, M_2 \rangle \cdot k \equiv M_k : A_k}$$

?, might have it

$$\frac{\Gamma \vdash M : A_1 \times A_2}{\Gamma \vdash \langle M \cdot 1, M \cdot 2 \rangle \equiv M : A_1 \times A_2}$$

a design requirement: all judgments ought to be decidable

- type checking
- defn'l equivalence (for some choice of defn eq)

Structural properties of entailment

derivation

Composition of rules

$\Gamma \vdash A \equiv A'$

Formalism is a useful approximation to the truth.

- DERIVATIONS ARE INHERENTLY COMPUTABLY ENUMERABLE
- Via formal correspondence to formal logics, type-checking and derivation checking is the same thing.