Harper 1

Type theory from comp. perspective

constructive

Type theory as if prog. lang.

→ Martin-Löf Constructive Mathematics and Computer Programming

Constable et al. NuPRL System & Semantics

Plan

1. Type theory starting from computation
2. → theory of truth (based on proof)
   ↳ based on computation
2. contrast that w/ formalism
   → theory of formal proof

cubical type theory from comp. perspective

just because you can play this video game known as Coq does not mean you know type theory

Idea: start w/ a programming language

(HoTT is lacking as comp content)

Deterministic operational semantics

Assume: some idea of abstract syntax w/ binding & scope — subst. for vars.

Forms of expression w/ binding & scope E

judgment forms $\boxed{E\ \mathrm{val}}$ means E is fully evaluated eg. $\mathrm{tt}\ \mathrm{val}$, $\mathrm{ff}\ \mathrm{val}$

transition $\boxed{E \mapsto E'}$ one step of simplification of E.

derived notion $E \Downarrow E_0$ means $E \mapsto^* E_0\ \mathrm{val}$

$E \Downarrow V$

eg. if $(N, P)(M)$
if $(E_1, E_2)(E)$

$$\frac{E \mapsto E'}{\mathrm{if}\ (E_1, E_2)(E) \longmapsto \mathrm{if}\ (E_1, E_2)(E')}$$

---

if $(E_1, E_2)(tt) \longmapsto E_1$

$\ldots \quad (ff) \longmapsto E_2$

"binary decision diagrams"

Shannon expansion

Types are <u>specifications</u> of program <u>behavior</u>.

(not "pieces of syntax")

~~eg~~

Two principal forms of <u>judgment</u>

(expressions of knowledge)

Brower premise:
mathematics as human activity

ep inf as fin w/ algorithm

A type

$M \in A$ — M program runs how A says it does

- behavioral, not structural
- both M and A M, A are <u>programs</u>

need Bool val judgment

eg)

Bool type

tt ∈ Bool ff ∈ Bool

("true by defn")

but this is not the defn

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

then if $(M_1; M_2)(M) \in A$

turns out to be a fact, not a defn

eg) if $(17, \underset{\text{(GARBAGE)}}{\text{"17"}})(tt) \in$ Nat ↪ assuming have that

if you run it → runs by simplifying to $17 \in$ Nat

eg) if $(Nat, Bool)(n)$ Type then $n \in$ Bool

any outcome for M induces a simplification to A type

eg) if $(17, \frac{tt}{ff})(M) \in$ if $(Nat, Bool)(M)$!

(need determin semantics)

because of n on both sides needs be same result

Specs / types are programs

(there is nothing else)

Key idea: families of types

type-indexed families of types

aka dependent types

eg) seq (n) type when $n \in Nat$

$n : Nat \gg seq(n)\,type$

↑ not same thing as turnstile

family of types indexed by a type

hypothetical general judgment

eg: $\langle 0, 1, \ldots, n-1 \rangle \in seq\ (n)$

$\langle 0, 1, \ldots, 9 \rangle \in seq\ (10)$

$f \in n : Nat \longrightarrow seq\ (n)$

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

notion of functionality

typ judgm / gen judgment

Critical idea: Functionality

families of types of elements

must respect equality of indices

eg. seq (2+2) "same as" seq (4)

seq (if (17;18)(M)) "same as"

if (seq(17); seq(18)) (M)

types →

a : Bool ⊢ seq (if (17, 18)(a))

"same as" "same type"

~~seq~~ if (seq(17), seq(18))(a)

↳ generic in 'a'

revise principal forms of judgment

Judgments

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

$M \in A$ $\rightsquigarrow$ $M \doteq M' \in A$

satisfaction (equisatisfaction) (exact equality of elements at type A)

depends completely on A

eg) not $2 \doteq 4 \in Nat$

is: $2 \doteq 4 \in Nat/2$ (Evens) } equality depends on type

in TT, equality is independent of type !?

generalizing $M \in A$ to $M \doteq M' \in A$

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

later add variables $[x_1 \ldots x_n]$

coordinate axes n-cubes

notation for kind of presheaf

Meaning explanations aka Semantics (computational)

1. $A \doteq A'$ means

$A \Downarrow A_0$ & $A' \Downarrow A_0'$ & $\underbrace{A_0 \doteq_0 A_0'}$

A_0, A_0' are equal type-values
equal canonical types

eg) by definition $\underbrace{Bool}_{val} \doteq_0 \underbrace{Bool}_{val}$ ie. Bool is type$_0$

2. $M \doteq M' \in A$ ~~means~~, where A type

(ie. $A \Downarrow A_0$ $A_0 \doteq_0 A_0'$)

means $M \Downarrow M_0$ & $M' \Downarrow M_0'$ & $M_0 \doteq_0 M_0' \in A_0$

(equal values in a type value)

3. $a:A \gg B \doteq B'$ means

if $M \doteq M' \in A$ then $B[M/a] = B'[M'/a]$

"functionality"

check $a:A \gg B$ type $\longrightarrow B \doteq B$

means

$M \doteq M' \in A$

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

4. $a : A \gg N \doteq N \in B$

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

means

if $M \doteq M' \in A$

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

$\doteq B[M'/a]$

Booleans:

1. $Bool \doteq_0 Bool$ ie. Bool type.

ie. Bool is a type

Bool names a type

(bec. type determined by what comes next)

(not going to say '17 type')

2. membership relⁿ

$M_0 = M_0' \in_0 Bool$ is the strongest relation (least)

such that

$tt \doteq_0 tt \in Bool$ (ie. $tt \in_0 Bool$)

$ff \doteq_0 ff \in Bool$ (ie. $ff \in_0 Bool$)

$\_\_ \& \doteq_0 \_\_ (\in Bool)$

strongest/least { (a.) the stated conditions/facts hold
(b.) nothing else! is true about that relation

relation $\mathcal{R}$ on program values ($\doteq_0$)

[Strongest $R \subseteq Exp \times Exp$
s.t. $R(tt, tt)$
$R(ff, ff)$]

You "must" accept this as a valid defn
(w/ Gödel's thm)

Prop / Fact / Claim

If $M \in Bool$ and A type and $M_1 \in A$ and $M_2 \in A$
then if $(M_1 ; M_2)(M) \in A$
(fact, not a defn)

Pf:
How to prove it?

Key $\in Bool$ is given by universal property
- least containing
$tt \in Bool$
$ff \in Bool$

Fix A type, consider the property
$M_1 \in A_1, M_2 \in A_2$

If $M \in Bool$ then ... M.

$$if\ (M_1; M_2)(M) \in A$$

its sufficient to show

for $M \in Bool$ means $M \Downarrow M_0$ $M_0 = tt$ or $M_0 = ff$

$$\left.\begin{array}{l} if\ (M_1; M_2)(tt) \in A \\ (ff) \in A \end{array}\right\}$$ because "if" evaluates its principal argument

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

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

Lemma (head-expansion) (reverse execution) "typing is closed under reverse execution"

If $M' \in A$ and $M \mapsto M'$,
then $M \in A$

Ex. prove it using defns in terms of eval to canon form

$if\ (M_1; M_2)(M) \longmapsto^* if\ (M_1; M_2)(tt)$
$\longmapsto^*$ (ff)

$\boxtimes$

(Using)

1) ~~using~~ fact Bool being inductively defined ("strongest relation")

2) typing is closed under head expansion