Bob Harper (Lecture 1) [2018/07/16] ①

Computational Type Theory

Thesis: A sufficiently expressive programming language is a foundation for all of mathematics.

- Martin-Löf Constructive math & computer programming
- Constable et al NuPRL System & semantics

Plan:

1. develop type theory starting w/ computation

   $\longrightarrow$ theory of TRUTH (based on proof)

2. contrast w/ formalisms (Theory of formal proof)

(Goal: cubical, higher-dimensional type theory)

Idea Start with a programming language
Deterministic operational semantics
Assume: some idea of abstract syntax w/ binding & scope — subst. for vars, etc.

Forms of expression: E
Two judgment forms:

1. E val means E is fully evaluated eg) tt val, ff val
2. $E \longmapsto E'$ means one step of simplification of E.
3. Derived notion $E \Downarrow E_0$ means $E \longmapsto^* E_0$ val

eg) if $(E_1; E_2)(E)$

$$\frac{E \longmapsto E'}{\text{if}(E_1; E_2)\ E \longmapsto \text{if}(E_1, E_2)\ E'}$$

$$\frac{}{\begin{cases} \text{if}(E_1, E_2)(tt) \longmapsto E_1 \\ \phantom{\text{if}(E_1, E_2)}(ff) \longmapsto E_2 \end{cases}}$$

Types are SPECIFICATIONS of BEHAVIOR!

Two principal forms of judgment (expression of knowledge)

A type
$M \in A$
1. behavioral (not structural)
2. both M & A here are programs.

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

Fact (not a def) if $M \in$ Bool and $M_1, M_2 \in A$ (type)
then $\mathrm{if}(M_1, M_2)(M) \in A$

Examples

1. $\mathrm{if}(17; \mathrm{loop})(tt) \in \mathrm{Nat}$
   runs by simplifying to $17 \in \mathrm{Nat}$

2. $\mathrm{if}(\mathrm{Nat}, \mathrm{Bool})(M) \in \mathrm{Type}$ when $M \in \mathrm{Bool}$
   b/c any outcome for M induces a simplification to a type.

3. $\mathrm{if}(17; tt)(M) \in \mathrm{if}(\mathrm{Nat}, \mathrm{Bool})(M)$!

SPECS/TYPES ARE PROGRAMS.

Key Idea: type indexed Families of types (aka dependent types)

eg) $\mathrm{seq}(n)$ type when $n \in \mathrm{Nat}$

$$\boxed{n : \mathrm{Nat} \gg \mathrm{seq}\ n \text{ type}}$$ hypothetical judgment

families of types indexed by a type

eg) $\langle 0, 1, \ldots, 9 \rangle \in \mathrm{seq}(10)$

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

Alt. notation: $f \in \Pi\, n : \mathrm{Nat} . \mathrm{seq}(n)$

Critical idea: Functionality

families (of types, of elements) must respect equality of indices.

What is equality?

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

$Bool \gg seq(\text{if}(17; 18)(a))$

"same as" $\text{if}(\underbrace{seq(17)}_{\text{types}}; \underbrace{seq(18)}_{})(a)$

The complexity of such expressions & "same as" relations enormous; but the collection of true statements in any formalism is relentlessly recursively enumerable (Gödel's thm).

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 A!

"satisfaction"

(It's a 3-place relation) $\rightarrow (M, M'; A) \in \doteq$

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

is: $2 \doteq 4 \in Nat/2$

$$\boxed{\text{Intention: if } M \doteq M' \in A \ \& \ A \doteq A' \text{ then } M \doteq M' \in A'.}$$

Meaning Explanations aka semantics (computational)

1. $A \doteq A'$ means $(\exists A_0\ A \Downarrow A_0 \wedge \exists A_0'\ A' \Downarrow A_0' \wedge \underbrace{A_0 \doteq_{val} A_0'}_{\text{A_0 \& A_0' are equal type-vals}})$

equal "canonical" types

eg) by def. $Bool \doteq_0 Bool$

ie $Bool$: type.

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

(i.e. $A \Downarrow A_0 \quad A_0 \doteq_0 A_0$)

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

"Functionality"

<u>check:</u> $a : A \gg \underline{B\ type} \to B \doteq B$.

means $M \doteq M' \in A$

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

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

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

$\doteq B[M'/a]$

(assuming that

$a : A \gg \underbrace{B \doteq B}_{B\ type}$)

Booleans

1. $Bool \doteq_0 Bool$ i.e. $Bool$ type.

   i.e. $Bool$ is a type (names a type).

2. $M_0 \doteq M_0 \in_0 Bool$ is the <u>strongest</u>

   relation R such that

   $tt \doteq_0 tt \in Bool$ (i.e. $tt \in Bool$)

   and $ff \doteq_0 ff \in Bool$ (i.e. $ff \in_0 Bool$)

a) the stated conditions hold
b) nothing else!

Strongest $R \subseteq Exp \times Exp$ (programs) s.t.

$$tt \; R \; tt \;\wedge\; ff \; R \; ff.$$

You "must" accept this as a valid defn.

Prop/Fact/Claim

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

(this is <u>not</u> a defn here.)

Proof:

key: $\in Bool$ is given by a universal property (least rel. containing $tt \in Bool$ and $ff \in Bool$)

Fix A type $M_1 \in A$, $M_2 \in A$,

WTS. if $M \in Bool$ then $if(M_1, M_2)(M) \in A$

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

suffices to show

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

Lemma ("head expansion" or "reverse execution")

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

EX prove it using the defs in terms of eval to canonical form.

(conclusion of proof)

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

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

use reverse execution lemma $\boxed{\text{T\&I}}$

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