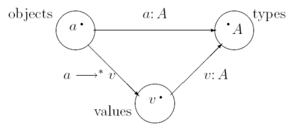
III. Computational meaning and inductive types

- Semantics
  - How to give meanings to (logical) sentences?
  - Model-theoretic semantics v.s. proof-theoretic semantics
  - (More later)
- For type theories:
  - How to understand the judgements in a proof-theoretic semantics?
  - Eg, how to understand the basic judgement "a : A"?
  - Ie, when is "a : A" a correct judgement?

Canonical objects: values of computation

- "a:A" is correct if a computes to a canonical object (value) v such that v:A.



Examples:

- A = Nat, a = 3+4, v = 7.
- A = Nat×Nat, a = (λx:Nat.⟨x,x+1⟩)(2), v = ⟨2,3⟩.

Inductive types

- Typical types in type theory
  - Logical propositions (as explained before)
  - Inductive types
  - Universes
- Examples of inductive types
  - Finite types ($\mathbb{0}$, $\mathbb{1}$, $\mathbb{2}$, …)
  - Types of nats, lists, vectors, trees, …
  - Types of dependent pairs/tuples (modules)
  - Types of ordinals, well-orderings, …

Type of natural numbers

Formation $\quad \dfrac{}{Nat : Type}$

Introduction $\quad \dfrac{}{0 : Nat} \quad \dfrac{n : Nat}{n+1 : Nat}$

Elimination

$$\frac{c : C(0) \qquad f(n) : C(n) \to C(n+1)\ [n : Nat]}{Rec(c,f)\ :\ \Pi x : Nat.\ C(x)}$$

Computation (behaviour of Rec, omitted)

Elimination rule explained

Elimination

$$\frac{c : C(0) \qquad f(n) : C(n) \to C(n+1)\ [n : Nat]}{Rec(c,f)\ :\ \Pi x : Nat.\ C(x)}$$

Elimination and induction:

$$\frac{C(0) \qquad C(n) \supset C(n+1)\ [n : Nat]}{\forall x.\ C(x)}$$

"If C holds for all canonical nats, then C holds for every nat."

- General pattern (for all inductive types):

C holds for all *canonical* objects of …

C holds for *every* object of …

More inductive types: the Boolean type $\mathbb{2}$

Formation $\quad \dfrac{}{2 : Type}$

Introduction $\quad \dfrac{}{true : 2} \quad \dfrac{}{false : 2}$

Elimination

$$\frac{c_1 : C(true) \qquad c_2 : C(false)}{\mathcal{E}_2(c_1,c_2)\ :\ \Pi x : 2.\ C(x)}$$

Computation (omitted)

More inductive types: List(A) & Vect(A,n)

- List(A) – type of lists of objects of type A
  - nil(A) : List(A)
  - cons(A,a,l) : List(A)
- Vect(A,n) – type of lists of length n
  - nil_V(A) : Vect(A,0)
  - $cons_V$(A,n,a,l) : Vect(A,n+1)
- Simple example:
  - Head of a list – what about hd(nil)? (to make it total …)
  - Head of a vector:

    $hd(n) : Vect(A,n+1) \to A$

    $hd(n,[a_1,\ldots,a_{n+1}]) = a_1$

Type universes: a reflection principle

- Collecting (the names of some) types into a type called a *universe.*
- How to define a type-valued function? For example,

  f(0) = Nat

  f(n+1) = f(n) x Nat

  But the "type" of Nat is *not* a type!
- Introduce a type universe U such that Nat : U, then

  $f : Nat \to U$

  This is now "legal".

More inductive types: ∑-types

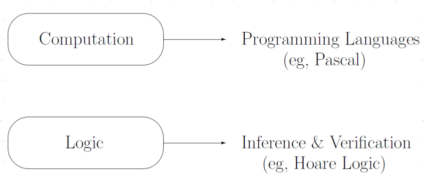
- ∑-types – types of pairs/tuples
  - Intuitively,

    $\Sigma(A,B) = \{ \langle x,y \rangle \mid x : A \;\&\; y : B(x) \}$
  - Type of modules

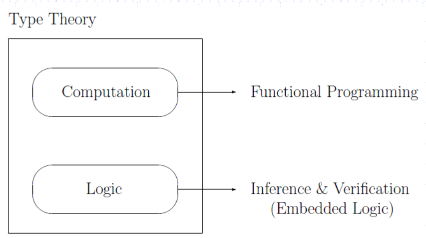
    $\Sigma\,[\, S : U,\ id : S \to S,\ + : S \to S \to S, \ldots \ldots \,]$

    where U is a universe.

Computation and logic in different languages:



Computation and logic in type theory:



Computational meanings

- Semantics
  - Model-theoretic semantics
    - Meanings of logical sentences are given by truth values in models.
    - Tarski, …
  - Proof-theoretic semantics
    - Meanings of logical sentences are given by canonical proofs and computation.
    - Gentzen, Prawitz, Dummett, Tait, Girard, Martin-Löf, Schroeder-Heister.

Data types v.s. logical propositions

- Natural separation between
  - Logical propositions
  - Inductive data types (eg, Nat, List(A), …)
- Philosophy behind the development of
  - ECC/UTT (Luo 1989/1994)
  - Logic-enriched TTs (Gambino & Aczel 2006, Luo 2006)
- Combining data types with propositions
  - Eg, Σ-types:

    Σ(Nat, positive) -- type of positive nats

    Σ(Man, handsome) -- type of handsome men

Revision Questions

- What kinds of types are there in modern TTs?
- What is a type universe? What is the difference between a universe and an inductive type?
- Data types v.s. logical propositions
  - In what sense may one identify them in a TT? What are the caveats?
  - How can one separate them in TTs? What are the advantages?

Selected References

- T. Coquand C. Paulin-Mohring. Inductively defined types. Proc of the Inter Conf on Computer Logic (COLOG-88). LNCS 417, 1990.
- M. Dummett. The Logical Basis of Metaphysics. Harvard University Press, 1993.
- Z. Luo. Computation and Reasoning: A Type Theory for Computer Science. OUP, 1994.
- P. Martin-Löf. Intuitionistic Type Theory. Bibliopolis, 1984.
- B. Nordström, K. Petersson, and J. Smith. Programming in Martin-Löf's Type Theory. OUP, 1990.