FLOLAC'14

Type theory and logic

Lecture III: natural number arithmetic

3 July 2014

柯向上

Department of Computer Science
University of Oxford

Hsiang-Shang.Ko@cs.ox.ac.uk

Natural numbers

- Formation:

$$\frac{}{\Gamma \vdash \mathbb{N} : \mathcal{U}}\ (\mathbb{N}\mathsf{F})$$

- Introduction:

$$\frac{}{\Gamma \vdash \mathtt{zero} : \mathbb{N}}\ (\mathbb{N}\mathsf{IZ}) \qquad \frac{\Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathtt{suc}\ n : \mathbb{N}}\ (\mathbb{N}\mathsf{IS})$$

- Elimination:

$$\frac{\begin{array}{l}\Gamma \vdash P : \mathbb{N} \to \mathcal{U} \\ \Gamma \vdash z : P\ \mathtt{zero} \\ \Gamma \vdash s : \Pi[x : \mathbb{N}]\ P\,x \to P\,(\mathtt{suc}\ x) \\ \Gamma \vdash n : \mathbb{N}\end{array}}{\Gamma \vdash \mathtt{ind}\ P\ z\ s\ n : P\ n}\ (\mathbb{N}\mathsf{E})$$

Logically this is the *induction principle*; computationally this is *primitive recursion*.

Natural numbers — computation rules

- Computation:

$$\frac{\begin{array}{l}\Gamma \vdash P : \mathbb{N} \to \mathcal{U} \\ \Gamma \vdash z : P\ \mathtt{zero} \\ \Gamma \vdash s : \Pi[x:\mathbb{N}]\ P\,x \to P\,(\mathtt{suc}\ x)\end{array}}{\Gamma \vdash \mathtt{ind}\ P\,z\,s\ \mathtt{zero} = z \in P\ \mathtt{zero}}\ (\mathbb{N}\mathrm{CZ})$$

$$\frac{\begin{array}{l}\Gamma \vdash P : \mathbb{N} \to \mathcal{U} \\ \Gamma \vdash z : P\ \mathtt{zero} \\ \Gamma \vdash s : \Pi[x:\mathbb{N}]\ P\,x \to P\,(\mathtt{suc}\ x) \\ \Gamma \vdash n : \mathbb{N}\end{array}}{\Gamma \vdash \mathtt{ind}\ P\,z\,s\,(\mathtt{suc}\ n) = s\,n\,(\mathtt{ind}\ P\,z\,s\,n) \in P\,(\mathtt{suc}\ n)}\ (\mathbb{N}\mathrm{CS})$$

Exercise. Define addition and multiplication with `ind`.

Inductively defined sets

The set of natural numbers is *inductively defined*.

In general, every element of an inductively defined set is recursively constructed in a finite number of steps.

Accompanying every inductively defined set is an induction principle, which says that the recursive structure of an element can guide computation, and is formulated as elimination and computation rules in type theory.

In Agda, every datatype can be thought of as being inductively defined, and a structurally recursive function on a datatype (usually using pattern matching) can be thought of as syntactic sugar for an invocation of its induction principle.

Identity types

Identity types are also called *propositional equality*, especially when contrasted with judgemental equality.

- Formation:

$$\frac{\Gamma \vdash A : \mathcal{U} \qquad \Gamma \vdash t : A \qquad \Gamma \vdash u : A}{\Gamma \vdash \mathtt{Id}\ A\ t\ u : \mathcal{U}}\ (\mathtt{IdF})$$

- Introduction:

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \mathtt{refl}\ t : \mathtt{Id}\ A\ t\ t}\ (\mathtt{IdI})$$

Exercise. Assume $\Gamma \vdash t = u \in A$ and derive $\Gamma \vdash \mathtt{refl}\ t : \mathtt{Id}\ A\ t\ u$.

Identity types — elimination and computation

- Elimination:

$$\frac{\begin{array}{l}\Gamma \vdash P : A \to \mathcal{U} \\ \Gamma \vdash t : A \\ \Gamma \vdash p : P\,t \\ \Gamma \vdash u : A \\ \Gamma \vdash q : \mathtt{Id}\,A\,t\,u\end{array}}{\Gamma \vdash \mathtt{transport}\,P\,p\,q : P\,u}\ (\mathtt{IdE})$$

- Computation:

$$\frac{\Gamma \vdash P : A \to \mathcal{U} \qquad \Gamma \vdash t : A \qquad \Gamma \vdash p : P\,t}{\Gamma \vdash \mathtt{transport}\,P\,p\,(\mathtt{refl}\,t) = p \in P\,t}\ (\mathtt{IdC})$$

Exercise. Prove

$$\Pi[A:\mathcal{U}]\ \Pi[B:\mathcal{U}]\ \Pi[f:A\to B]$$
$$\Pi[x:A]\ \Pi[y:A]\ \mathtt{Id}\,A\,x\,y \to \mathtt{Id}\,B\,(f\,x)\,(f\,y)$$

Id is an equivalence relation

Id is obviously reflexive — we can derive

$$\vdash \lambda A.\, \lambda x.\, \mathtt{refl}\ x : \Pi[A : \mathcal{U}]\ \Pi[x : A]\ \mathtt{Id}\ A\ x\ x$$

Exercise. Prove that Id is symmetric and transitive, i.e.,

$$\Pi[A : \mathcal{U}]\ \Pi[x : A]\ \Pi[y : A]\ \mathtt{Id}\ A\ x\ y \to \mathtt{Id}\ A\ y\ x$$

and

$$\Pi[A : \mathcal{U}]\ \Pi[x : A]\ \Pi[y : A]\ \Pi[z : A]$$
$$\mathtt{Id}\ A\ x\ y \to \mathtt{Id}\ A\ y\ z \to \mathtt{Id}\ A\ x\ z$$

Identity types — general elimination and computation

- Elimination:

$$\frac{\begin{array}{l}\Gamma \vdash t : A \\ \Gamma \vdash P : \Pi[x : A]\ \mathtt{Id}\ A\ t\ x \to \mathcal{U} \\ \Gamma \vdash p : P\ t\ (\mathtt{refl}\ t) \\ \Gamma \vdash u : A \\ \Gamma \vdash q : \mathtt{Id}\ A\ t\ u\end{array}}{\Gamma \vdash \mathtt{J}\ P\ p\ q : P\ u\ q}\ (\mathtt{IdE})$$

- Computation:

$$\frac{\begin{array}{l}\Gamma \vdash t : A \\ \Gamma \vdash P : \Pi[x : A]\ \mathtt{Id}\ A\ t\ x \to \mathcal{U} \\ \Gamma \vdash p : P\ t\ (\mathtt{refl}\ t)\end{array}}{\Gamma \vdash \mathtt{J}\ P\ p\ (\mathtt{refl}\ t) = p\ \in P\ t\ (\mathtt{refl}\ t)}\ (\mathtt{IdC})$$

Peano axioms

Peano axioms specify an *equational theory* of natural number arithmetic; all of them are provable in type theory.

- Zero is a natural number. If n is a natural number, so is the successor of n.
  — The introduction rules.
- Equality on natural numbers is an equivalence relation.
  — We use $\mathtt{Id}$, which has been proved to be an equivalence relation.
- The successor operation is an injective function, i.e.,

$$\Pi[m:\mathbb{N}]\ \Pi[n:\mathbb{N}]\ \mathtt{Id}\ \mathbb{N}\ m\ n \leftrightarrow \mathtt{Id}\ \mathbb{N}\ (\mathtt{suc}\ m)\ (\mathtt{suc}\ n)$$

- The successor operation never yields zero, i.e.,

$$\Pi[n:\mathbb{N}]\ \neg\, \mathtt{Id}\ \mathbb{N}\ (\mathtt{suc}\ n)\ \mathtt{zero}$$

Peano axioms

- Addition satisfies

$$\Pi[n:\mathbb{N}]\ \mathtt{Id}\ \mathbb{N}\ (\mathtt{zero}+n)\ n$$

  and

$$\Pi[m:\mathbb{N}]\ \Pi[n:\mathbb{N}]\ \mathtt{Id}\ \mathbb{N}\ ((\mathtt{suc}\ m)+n)\ (\mathtt{suc}\ (m+n))$$

- Multiplication satisfies

$$\Pi[n:\mathbb{N}]\ \mathtt{Id}\ \mathbb{N}\ (\mathtt{zero}\times n)\ \mathtt{zero}$$

  and

$$\Pi[m:\mathbb{N}]\ \Pi[n:\mathbb{N}]\ \mathtt{Id}\ \mathbb{N}\ ((\mathtt{suc}\ m)\times n)\ (n+m\times n)$$

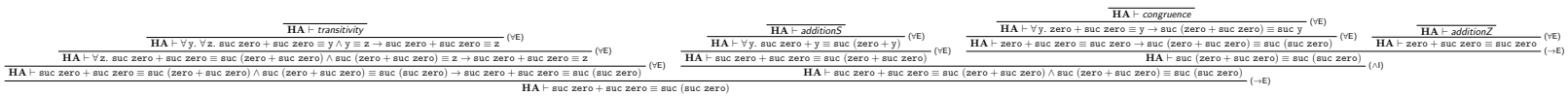
- The induction principle holds for natural numbers.
  — The elimination rule.

Computational foundation

In type theory, Peano "axioms" (formulated as propositions) are merely consequences, and do not play an important role in actual theorem proving.

We now have a more natural foundation of mathematics based on the idea of typed computation.

The infamous proof of $1 + 1 = 2$



is just an automatic check by computation in type theory.