CS 671, Automated Reasoning

Lesson 3: *What is a type (II)* January 28, 2001

Semantics

$\boxed{\text{Q:}}$ *How does one usually express semantics?*

Traditionally, the semantics of a formal language is described by *interpreting* its basic constructs in a foundational mathematical theory such as set theory. Essentially this means we map basic language constructs onto set theory concepts and then define inductively the meaning of all possible expressions.

However, we can't do that for type theory for two reasons. $\boxed{\text{Q:}}$ *why?*

First of all, type theory is intended to be a foundational theory itself, that is it should not depend on any other theory but give a precise and direct meaning to intuitively well understood concepts.

Second, set theory is not capable of expressing computational concepts such as value and computation. Although Turing machines and similar computation models could in principle be expressed in set theory, this presupposes that the tables used for describing concrete models are in some canonical form – i.e. basic values and not complex complex expression denoting these values. Set theory, however, does not provide the means to effectively distinguish between these two and thus even the basic computation steps of a Turing machine could lead to undecidable problems like "*is the value I read on the tape the same as the one stored in my transition table*". It doesn't make sense to base a computational theory on a theory that can't even decide that.

Therefore, we will use a more direct approach for expressing the semantics of types and their elements. For this purpose Martin-Löf introduced the concept of *judgments*, that is statements of fundamental truths. These judgments form the foundation of the theory – they are not up for discussion but postulated as facts whose truth is evident according to our intuitive understanding of the concepts we formalize. In a sense, they are the axioms of the theory.

Of course, one better makes sure that only the most fundamental truths are being postulated – those that we easily comprehend – and that other truths can be derived from that. This allows us to check the *consistency* of our theory, i.e. that it doesn't contain any paradoxes. *Soundness*, i.e. whether every judgment is actually true, can obviously not be checked since it relies on our intuitive understanding of what is true. For that reason, judgments are usually only made about facts that are not seriously disputed.

For our theory of natural numbers this means that we only make four judgments.

1. $\mathbb{N}$ is a type
2. 0 is an element of $\mathbb{N}$
3. $\texttt{suc}(e)$ is an element of $\mathbb{N}$ if e is
4. if $e{\downarrow}v$ and v is an element of $\mathbb{N}$ then so is e

Note that these judgments do not define any properties of natural numbers other than that they *are* natural numbers. The first three judgments are quite evident. No one would ever question them. The last is the most crucial one, as it ties judgments to evaluation: *everything that evaluates to a natural number* **is** *a natural number.*

Most mathematical theories proceed in a different way. They would first state their basic axioms (about "is an element" and "is a type") and *then* define evaluation and prove its properties. Many logicians would not accept the fourth judgment as an axiom. Moreover, they would usually want to prove it in the opposite direction, as *subject reduction*

if $e{\downarrow}v$ and e is an element of $\mathbb{N}$ then so is v

Note that the four judgments do not directly talk about the `ind` term. `ind` terms are only natural numbers if they can be evaluated into some. However, the third judgment makes sure that terms like `ind(suc(suc(0));suc(0);x.suc(x))`, which under eager evaluation evaluates to `suc(suc(suc(0)))` but does not evaluate to a primitive value under lazy evaluation, are judged to be elements of $\mathbb{N}$ anyway. The argument goes like this:

(4) `ind(suc(suc(0));suc(0);x.suc(x))` $\downarrow$ `suc(ind(suc(0);suc(0);x.suc(x)))`
so `ind(suc(suc(0));suc(0);x.suc(x))` is an element of $\mathbb{N}$
if `suc(ind(suc(0);suc(0);x.suc(x)))` is

(3) `suc(ind(suc(0);suc(0);x.suc(x)))` is an element of $\mathbb{N}$
if `ind(suc(0);suc(0);x.suc(x))` is.

(4) `ind(suc(0);suc(0);x.suc(x))` $\downarrow$ `suc(ind(0;suc(0);x.suc(x)))`
so `ind(suc(0);suc(0);x.suc(x))` is an element of $\mathbb{N}$
if `suc(ind(0;suc(0);x.suc(x)))` is.

(3) `suc(ind(0;suc(0);x.suc(x)))` is an element of $\mathbb{N}$ if `ind(0;suc(0);x.suc(x))` is.

(4) `ind(0;suc(0);x.suc(x))` $\downarrow$ `suc(0)`
so `ind(0;suc(0);x.suc(x))` is an element of $\mathbb{N}$ if `suc(0)` is.

(3,2) `suc(0)` is is an element of $\mathbb{N}$ if 0 is, which is the case.

That means, we can judge expressions to be natural numbers even if our evaluation mechanism cannot reduce them to primitive values. However, we only judge expressions to be numbers, if they actually correspond to a number constant. Expressions like `suc(n)` or `ind(suc(suc(0));m;x.suc(x))` are *not* natural numbers, because there is no judgment about variables to *be* numbers. They are just placeholders for numbers, but not numbers themselves.

Note also, that we do not judge that variables are not numbers. It would be a terrible mistake to introduce such a judgment, as this would create inconsistencies with later extensions to, for instance, functions. There is also no need to introduce such a judgment. Our four judgments provide no evidence that standalone variables are natural numbers so we can't consider them as such.

This insight leads to another interesting observation. Judgments only state what is known to be the case. In contrast to logical propositions or conjectures, which may turn out to be true, false, or undecidable, judgments are "the final authority" or the unchangeable truths of our theory – hence the word "judgment". And we only care about what we can know to be true, which may include knowing that certain things are not the case. But we don't care about the things that we cannot know – or cannot prove – because these are useless for reasoning. This does, of course, not exclude reasoning under unproven assumptions, but in this case the assumptions will be explicitly stated as such, as in the case of the third and fourth judgment.

Proof Theory

The purpose of a proof theory is to express semantical reasoning by syntactical manipulations of expressions. For this purpose we have to introduce a more formal notation for the statements that we want to prove. So far there are two kinds of statements we can make.

1. T is a type, *which we express as* T `type`
2. e is an element of T, *which we express as* $e \in T$

where e and T are expressions of our object language. To *prove* such a statement formally, we will have to describe *proof rules* that reflect the four judgments of our small theory. For the first two judgments, this is straightforward, as both of them are *primitive judgments*. In our proof theory, these are reflected by two axioms

1. $\mathbb{N}$ `type`
2. $0 \in \mathbb{N}$

The third judgment, "$\texttt{suc}(e)$ is an element of $\mathbb{N}$ if e is", is a *hypothetical judgment*, because it judges $\texttt{suc}(e) \in \mathbb{N}$ under the assumption that e is an element of $\mathbb{N}$. There are several styles to represent such a hypothetical judgment.

In *natural deduction* style, one would draw a line and write the assumptions on top and the conclusion below

3. $\dfrac{e \in \mathbb{N}}{\texttt{suc}(e) \in \mathbb{N}}$

Proofs are then built as trees, with the axioms on top and each link corresponding to an instance of such a proof rule.

Another option would be to represent the hypothetical judgment as a *sequent* of the form

3. $e{:}\mathbb{N} \vdash \texttt{suc}(e) \in \mathbb{N}$

which is usually read as "*under the assumption (or* hypothesis*) that* e *is an element of* $\mathbb{N}$ *we conclude that* $\texttt{suc}(e)$ *is an element of* $\mathbb{N}$". The turnstyle $\vdash$ is a notation that is borrowed from the *sequent calculus* and indicates *logical deduction*.

The use of the colon : on the left and the element symbol $\in$ on the right of a sequent has historical reasons. The colon is a standard notation for the *declaration* of (object-)variables and introducing *labels*, which traditionally expect (object-)variables to occur on the left hand side of the colon. In contrast to that the element symbol can be applied to compound expressions and not just to variables.

Proofs would combine sequents using a variation of the *modus ponens* rule, which says that two sequents may be combined if the *conclusion* of one sequent, i.e. its right hand side, matches one of hypotheses of the other.

$$\frac{H_1,\ldots,H_n \vdash C \qquad H'_1,\ldots,C,\ \ldots,\ H'_n \vdash C'}{H_1,\ldots,H_n,\ H'_1,\ldots,\ H'_n \vdash C'}$$

The sequent style requires a small modification in the formalization of the first two judgments, which now have to be written as sequents without hypotheses. We use a small dot $\cdot$ to denote empty hypotheses lists.

1. $\cdot \vdash \mathbb{N}$ `type`
2. $\cdot \vdash$ `0` $\in \mathbb{N}$

Sequent proofs have the advantage that all the information needed to make a judgment is presented *locally*, that is within the sequent, while in natural deduction style one has to consider the whole proof tree to see what assumptions have been made.

Representing hypothetical judgments directly as sequents, however, has certain problems. The notation $e{:}\mathbb{N} \vdash$ `suc(`e`)` $\in \mathbb{N}$ suggests that e is a variable which is declared to be of type $\mathbb{N}$. This means that we cannot use the sequent for reasoning about the successor of arbitrary expressions (unless we break with tradition, which may lead to all kinds of new problems) and have to introduce additional proof mechanisms for doing so. Furthermore, the declaration $e{:}\mathbb{N}$ assumes that we know that $\mathbb{N}$ is a type, but this assumption is not stated explicitly.

The proof calculus of NUPRL's type theory uses sequent-syle reasoning but avoids introducing assumptions unless it is essential to have them (for instance when reasoning about the use of variables, as we will se below). As in natural deduction, hypothetical judgments will be represented by proof rules. These rules will, however, be based on sequents, which means that each node in a proof tree will be self-contained. Furthermore, the rules will be given in a *top-down fashion* and thus support reasoning by *refinement*: a proof task, or *goal*, will be decomposed into smaller goals until we have reached an axiom.[2]

In *Refinement Logic* the rules representing the first three judgment of our small theory of natural numbers are the following

1. $\bar{H} \vdash \mathbb{N}$ `type` **by** $\mathbb{N}$`-rule`
2. $\bar{H} \vdash$ `0` $\in \mathbb{N}$ **by** `0-rule` (read "$\mathbb{N}$ is a type and 0 is a member of $\mathbb{N}$")
3. $\bar{H} \vdash$ `suc(`e`)` $\in \mathbb{N}$ **by** `suc-rule`
   $\bar{H} \vdash e \in \mathbb{N}$

where $\bar{H}$ is a placeholder for an arbitrary list of hypotheses.

Applying a rule like the third one to a concrete proof goal means matching its first line, the *main goal*, to the goal and generating a *subgoal* that consists of the instantiated second line. The name of the rule is left next to the main goal to indicate which rule was applied.

A refinement proof for `suc(suc(0))` $\in \mathbb{N}$, for instance, proceeds as follows

$\cdot \vdash$ `suc(suc(0))` $\in \mathbb{N}$ **by** `suc-rule`
  $\cdot \vdash$ `suc(0)` $\in \mathbb{N}$ **by** `suc-rule`
    $\cdot \vdash$ `0` $\in \mathbb{N}$ **by** `0-rule`

[2] In contrast to that, most logic books use a bottom-up style of reasoning, which requires one to start with the axioms and to combine them into a proof of the goal. While this style may be appropriate for *presenting* proofs, it does not reflect the way proofs are found and is therefore less suited for interactive and automated theorem proving.

In general, a refinement logic rule has the following form

$$\begin{array}{l} \textit{main goal} \quad \textbf{by} \quad \textit{rule name} \\ \qquad \textit{subgoal}_1 \\ \qquad \quad \vdots \\ \qquad \textit{subgoal}_n \end{array}$$

That is, a proof rule decomposes a main goal into several subgoal sequences in a way that proofs for all these subgoals provide sufficient evidence for the truth of the main goal.

So far, the representation of the judgments of our theory of natural numbers was straightforward. The fourth judgment, however, cannot be represented in the same way, as it involves the concept of evaluation, which is not part of our object language.

4. if $e\downarrow v$ and v is an element of $\mathbb{N}$ then so is e

In fact, this judgment is far more general than the other three, as it ties membership to evaluation without considering concrete terms of our object language. The only way to represent it in our proof theory is to explicitly state a proof rule about the `ind` term that reflects the above judgment.

$\boxed{\text{Q:}}$ *What would be a meaningful rule for the* `ind` *term?*

For `ind(`e`;`$base$`;`x`.`up`)` to reduce to a natural number, the expression e must evaluate to one (otherwise the reduction rules don't apply). Also $base$ must be a natural number (in case e reduces to 0), and up must be one, provided the expression that we substitute for x is one. These insights are represented by the following rule.

4. $\bar{H} \vdash \texttt{ind}(e;base;x.up) \in \mathbb{N}$ **by** `ind-rule`
   $\bar{H} \vdash e \in \mathbb{N}$
   $\bar{H} \vdash base \in \mathbb{N}$
   $\bar{H}, \; x{:}\mathbb{N} \vdash up \in \mathbb{N}$

Remarks: