CS6180: Lecture 9

Robert Constable

September 18, 2017

1 Lecture Summary

After talking with a number of students after class and discussing the homework problems, I think it will be good to *review first-order logic* and stress these points: the style of refinement logic proofs, a formulation of intuitionistic first-order logic, iFOL, in refinement logic, and its interpretations in type theory. We will then return to surveying the types available in constructive type theory and how they can be incorporated into the core first-order logic. We will use the logic rules implemented in Nuprl and in particular look at the presentation in a Nuprl article on completeness issues for iFOL [1].

On Thursday we will finish our discussion of the basic types of constructive type theory and consider their logical meaning in more detail.

2 First-Order Logic

For each rule we provide a name that is the *outermost operator* of a proof expression with slots to be filled in as the refinement style proof is developed. The partial proofs are organized as a tree generated in two passes. The first pass is top down, driven by the user creating terms with slots to be filled in on the algorithmic bottom up pass once the downward pass is complete. Here is a simple proof of the intuitionistic tautology $A \Rightarrow (B \Rightarrow A)$.

$$\vdash A \Rightarrow (B \Rightarrow A) \ by \ \lambda(x.slot_1(x))$$

$$x : A \vdash (B \Rightarrow A) \ by \ slot_1(x)$$

In the next step, $slot_1(x)$ is replaced at the leaf of the tree by $\lambda(y.slot_2(x, y))$ to give:

$$\vdash A \Rightarrow (B \Rightarrow A) \ by \ \lambda(x.slot_1(x))$$

$$x : A \vdash (B \Rightarrow A) \ by \ \lambda(y.slot_2(x, y)) \ for \ slot_1(x)$$

$$x : A, y : B \vdash A \ by \ slot_2(x, y)$$

When the proof is complete, we see the slots filled in at each inference step as in:

$$\vdash A \Rightarrow (B \Rightarrow A) \ by \ \lambda(x.\lambda(y.x))$$

$$x : A \vdash (B \Rightarrow A) \ by \ \lambda(y.x)$$

$$x : A, y : B \vdash A \ by \ x$$

We present the rules in a *top down style*, as in tableaux, and as in the style of the *Edinburgh LCF* tactic system [3] and the Nuprl Proof Development System [2]. Here we show the *construction rules* first, often called the *introduction rules* because they introduce the canonical proof terms or called the *right hand side* rules in the sequent calculus because they apply to terms on the right hand side of the turnstile. So typical names seen in the literature are these: for & we say *AndIntro* or *AndR*; for $\Rightarrow$ we say *ImpIntro* or *ImpR*; for $\vee$ we say *OrIn-r* or *OrIn-l*, and for $\forall x$ we say *AllIntro* or *AllR*, and for $\exists$ we say *ExistsIntro* or *ExistsR*.

For each of these construction rules, the constructor needs subterms which build the component pieces of evidence. Thus for *AndR* the full term will have slots for the two pieces of evidence needed, the form will be $AndR(slot1, slot2)$ where the slots are filled in as the proof tree is expanded. When the object to be filled in depends on a new hypothesis to be added to the left hand side of the turnstile, the rule name supplies a unique label for the new hypothesis, so we see a rule name like $ImpR(x.slot(x))$ or $AllR(x.slot(x))$. In the case of the rule for $\exists$, there is a subtlety. The rule name provides two slots, but the second depends on the object built for the first, so we see rule names such as $ExistsR(a; slot(a))$.

For each connective and operator we also have rules for their occurrence on the left of the turnstile. These are the *rules for decomposing* or using or *eliminating* a connective or operator. They tell us how to use the evidence that was built with the corresponding construction rules, and the formula being decomposed is always named by a label in the list of hypotheses, so there is a variable associated with each rule application. Here are typical names: for & we say $AndElim(x)$ or $AndL(x)$. However, there must be more to this rule name because typically new formulas are added to the hypothesis list, one for each of the conjuncts, so we need to provide labels for these formulas. Thus the form of elimination for & is actually $AndL(x; l, r.slot(l, r))$ where l stands for the left conjunct and r for the right one.

Rule names such as $AndR$, $AndL$, OrR_l, OrR_r, OrL, and so forth are suggestive in terms of the details of the proof system, but they are not suggestive of the structure of the evidence, the semantics. We will use rule names that define the computational forms of evidence. The *evaluation rules* for these proof terms are given as in ITT or CTT, for instance in the book *Implementing*

Mathematics [2] or in the Nuprl Reference Manual [4].

So instead of $AndR(a;b)$ where a and b are the subterms built by a completed proof by progressively filling in open slots, we use $pair(a;b)$ or even more succinctly $< a, b >$, and for the corresponding decomposition rule we use $spread(x; l, r.t(l, r))$ where the binding variables l, r have a scope that is the subterm $t(l, r)$. This term is a compromise between using more familiar operators for decomposing a pair p such as $first(p)$ and $second(p)$ or $p.1$ and $p.2$ with the usual meanings, e.g., $first(< a, b >) =< a, b > .1 = a$. The reason to use *spread* is that we need to indicate how the subformulas of $A\&B$ will be named in the hypothesis list.

The decomposition rules for $A \Rightarrow B$ and $\forall x.B(x)$ are the most difficult to motivate and use intuitively. Since the evidence for $A \Rightarrow B$ is a function $\lambda(x.b(x))$, a reader might expect to see a decomposition rule name such as $apply(f;a)$ or abbreviated to $ap(f;a)$. However, a Gentzen sequent-style proof rule for decomposing an implication has this form:

$H, f : A \Rightarrow B, H' \vdash G$ *by* $ImpL$ *on* f
1. $H, f : A \Rightarrow B, H' \vdash A$
2. $H, f : A \Rightarrow B, v : B, H' \vdash G$

As the proof proceeds, the two subgoals 1 and 2 with conclusions A and G respectively will be refined, say with proof terms $g(f, v)$ and a respectively. We need to indicate that the value v is $ap(f;a)$, but at the point where the rule is applied, we only have slots for these subterms and a name v for the new hypothesis B. So the rule form is $apseq(f; slot_a; v.slot_g(v))$ where we know that v will be assigned the value $ap(f; slot_a)$ to "sequence" the two subgoals properly. So *apseq* is a sequencing operator as well as an application, and when the subterms are created, we can evaluate the term further as we show below. We thus express the rule as follows.

$H, f : A \Rightarrow B, H' \vdash G$ *by* $apseq(f; slot_a; v.slot_g(v))$
$H, f : A \Rightarrow B, v : B, H' \vdash G$ *by* $slot_g(v)$
$H, f : A \Rightarrow B, H' \vdash A$ *by* $slot_a$

We evaluate the term $apseq(f; a; v.g(v))$ to $g(ap(f;a))$ or more succinctly to $g(f(a))$. This simplification can only be done on the final bottom up pass of creating a closed proof expression, one with no slots.

Semantic consistency With this introduction, the following rules should make sense. They define what we will call the *pure proof expressions*. It should also be clear that we can easily prove by induction on the structure of proofs that there is computational evidence for every provable formula and that the evidence is polymorphic (uniform). This provides a simple semantic consistency proof for iFOL and easy demonstrations that specific formulas such as $P \vee \sim P$ are not provable.

2.1 First-order refinement style proof rules over domain of discourse D

Minimal Logic

Construction rules

- **And Construction**

  $H \ \vdash \ A\&B$ *by* $pair(slot_a; slot_b)$
  $H \ \vdash A$ *by* $slot_a$
  $H \ \vdash B$ *by* $slot_b$

- **Exists Construction**

  $H \ \vdash \ \exists x.B(x)$ *by* $pair(d; slot_b(d))$
  $H \ \vdash d \in D$ *by* $obj(d)$
  $H \ \vdash B(d)$ *by* $slot_b(d)$

- **Implication Construction**

  $H \ \vdash \ A \Rightarrow B$ *by* $\lambda(x.slot_b(x))$ *new* x
  $H, \ x : A \vdash B$ *by* $slot_b(x)$

- **All Construction**

  $H \ \vdash \ \forall x.B(x)$ *by* $\lambda(x.slot_b(x))$ *new* x
  $H, \ x : D \vdash B(x)$ *by* $slot_b(x)$

- **Or Construction**

  $H \ \vdash \ A \vee B$ *by* $inl(slot_l)$
  $H \ \vdash A$ *by* $slot_l$

  $H \ \vdash \ A \vee B$ *by* $inr(slot_r)$
  $H \ \vdash B$ *by* $slot_r$

Decomposition rules

- **And Decomposition**

  $H, x : A\&B, H' \vdash G$ *by* $spread(x; l, r.slot_g(l, r))$ *new* l, r

$H, l : A, r : B, H' \vdash G$ *by* $slot_g(l, r)$

- **Exists Decomposition**

  $H, x : \exists y.B(y), H' \vdash G$ *by* $spread(x; d, r.slot_g(d, r))$ *new* d, r

  $H, d : D, r : B(d), H' \vdash G$ *by* $slot_g(d, r)$

- **Implication Decomposition**

  $H, f : A \Rightarrow B, H' \vdash G$ *by* $apseq(f; slot_a; v.slot_g[ap(f; slot_a)/v])$ *new* v[1]
  $H, f : A \Rightarrow B, H' \vdash A$ *by* $slot_a$

  $H, f : A \Rightarrow B, H', v : B \vdash G$ *by* $slot_g(v)$

- **All Decomposition**

  $H, f : \forall x.B(x), H' \vdash G$ *by* $apseq(f; d; v.slot_g[ap(f; d)/v])$

  $H, f : \forall x.B(x), H' \vdash d \in D$ *by* $obj(d)$

  $H, f : \forall x.B(x), H', v : B(d) \vdash G$ *by* $slot_g(v)$[2]

- **Or Decomposition**

  $H, y : A \vee B, H' \vdash G$ *by* $decide(y; l.leftslot(l); r.rightslot(r))$

  1. $H, l : A, H' \vdash G$ *by* $leftslot(l)$

  2. $H, r : B, H' \vdash G$ *by* $rightslot(r)$

- **Hypothesis**

  $H, d : D, H' \vdash \ d \in D$ *by* $obj(d)$

  $H, x : A, H' \vdash \ A$ *by* $hyp(x)$
  We usually abbreviate the justifications to *by* d and *by* x respectively.

Intuitionistic Rules

[1] This notation shows that $ap(f; slot_a)$ is substituted for v in $g(v)$. In the CTT logic we stipulate in the rule that $v = ap(f; slot_a)$ *in* B.

[2] In the CTT logic, we use equality to stipulate that $v = ap(f; d)$ *in* $B(v)$ just before the hypothesis $v : B(d)$.

- **False Decomposition**

  $H, f : False, H' \vdash G$ *by* $any(f)$

  This is the rule that distinguishes intuitionistic from minimal logic, called "ex falso quodlibet". We use the constant $False$ for intuitionistic formulas and $\bot$ for minimal ones to distinguish the logics. In practice, we would use only one constant, say $\bot$, and simply add the above rule with $\bot$ for $False$ to axiomatize iFOL. However, for our results it's especially important to be clear about the difference, so we use both notations.

Note that we use the term d to denote objects in the domain of discourse D. In the classical evidence semantics, we assume that D is non-empty by postulating the existence of some d_0 in it. Also note that in the rule for $False$ Decomposition, it is important to use the $any(f)$ term which allows us to thread the explanation for how $False$ was derived into the justification for G.

Classical Rules

- **Non-empty Domain of Discourse**

  $H \vdash d_0 \in D$ *by* $obj(d_0)$

- **Law of Excluded Middle (LEM)**

  Define $\sim A$ as $(A \Rightarrow False)$

  $H \vdash (A \vee \sim A)$ *by* **magic**(A)

  Note that this is the only rule that mentions a formula in the rule name.

References

[1] Robert Constable and Mark Bickford. Intuitionistic Completeness of First-Order Logic. *Annals of Pure and Applied Logic*, 165(1):164–198, January 2014.

[2] Robert L. Constable, Stuart F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, Douglas J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, James T. Sasaki, and Scott F. Smith. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, NJ, 1986.

[3] Michael Gordon, Robin Milner, and Christopher Wadsworth. *Edinburgh LCF: a mechanized logic of computation*, volume 78 of *Lecture Notes in Computer Science.* Springer-Verlag, NY, 1979.

[4] Christoph Kreitz. The Nuprl Proof Development System, version 5, Reference Manual and User's Guide. Cornell University, Ithaca, NY, 2002. `http://www.nuprl.org/html/02cucs-NuprlManual.pdf`.