Lecture 4: Reasoning about Logic Specifications

Dale Miller
Penn State University

September 2000

The meta-logic

We shall use $FO\lambda^{\Delta\mathbb{N}}$ as our meta-logic: the logic in which we model "mathematics level" reasoning about computation. Clearly this is a powerful logic. Can we use it convenienty to actually reason about computations?

This question is still a current research topic.

- Some successes are in McDowell's thesis. Examples include the proof of a number of properties of evaluation and typing for a small functional programming language (PCF) with references.
- Eriksson built the PI proof editor several years ago and it was used to develop a number of proofs.
- Wajs and Miller are working on a prototype theorem prover. It is being built in λProlog.
- Libraries of theorems need to be built.
- Target applications: semantically deep results for specification and programming languages, such as, the λ-calculus and the π-calculus. Other domains will be considered later.

Issues in encoding an object logic

We shall now encode a proof system for a specific object logic to illustrate how we might reason about computation specified in logic.

The trick here is to think of proofs, formulas, contexts, etc, as being just any other data structure.

But these have bound variables in them!

Problem: Isn't unification of structures with bound variables complex? Higher-order unification is undecidable and CSUs cannot be effectively computed.

Approach: In practice, most higher-order unification actually falls into a rather small subset that is decidable and has most general unifiers. (It pays to do examples!) One such subset, called L_λ or *higher-order pattern unification*, is widely implemented.

Problem: Induction on second order types, for example, $(tm \to tm) \to tm$, is not possible since tm appears negatively in the source type (that is, the corresponding operator is not monotone).

Approach: We shall move to doing induction on sequent calculus proofs, not natural deduction proofs. The problem goes away in this style of representation.

A subset of intuitionistic logic

For brevity we will restrict our discussion here to a fragment of the logic containing implication and quantification. The full calculus can be handled but it adds no new insights.

We use the type i for terms of the object logic, the type atm for atoms (atomic propositions) and the type prp for general propositions.

$$\begin{array}{lclclcl} \langle\;\rangle & : & atm \to prp & \qquad & \bigwedge & : & (i \to prp) \to prp \\ \Rightarrow & : & prp \to prp \to prp & \qquad & \bigvee & : & (i \to prp) \to prp \quad . \end{array}$$

The constant $\langle\;\rangle$ coerces atoms into propositions: object-level predicates will be constants that build meta-level terms of type atm.

The constant $\Rightarrow$ represents the implication connective and $\bigwedge$ and $\bigvee$ encode universal and existential quantification at type i.

Notice that we are using the λ-abstraction of $FO\lambda^{\Delta\mathbb{N}}$'s term language to represent the variable binding of the two object logic quantifiers.

Explicit encoding of sequents

$$seq \quad : \quad nt \rightarrow prplst \rightarrow prp \rightarrow o \ .$$

The first argument is an induction measure and will be displayed as a subscript.

The second argument is a list of terms of type *prp* and represents the left side of the sequent. We will assume that *prplst* is the same as the type *lst* introduced earlier using *prp* for the type of elements. In particular we will assume that we have constructors *nil* and ::, and a predicate *element* as defined in $\mathcal{D}(list(prp))$.

The third argument to *seq* corresponds to the right side of the sequent.

Object-level provability

$$
\begin{aligned}
seq_I\ L\ \langle A\rangle &\triangleq element\ \langle A\rangle\ L\\
\\
seq_{(S(I))}\ L\ (B \Rightarrow C) &\triangleq seq_I\ (B :: L)\ C\\
seq_{(S(I))}\ L\ (\bigwedge B) &\triangleq \forall x\ seq_I\ L\ (B\,x)\\
seq_{(S(I))}\ L\ (\bigvee B) &\triangleq \exists x\ seq_I\ L\ (B\,x)\\
\\
seq_{(S(I))}\ L\ D &\triangleq \exists b \exists c (element\ (b \Rightarrow c)\ L\ \wedge\ seq_I\ (c :: L)\ D\ \wedge\ seq_I\ L\ b)\\
seq_{(S(I))}\ L\ C &\triangleq \exists b (element\ \bigwedge b\ L\ \wedge\ \exists x\ seq_I\ ((b\,x) :: L)\ C)\\
seq_{(S(I))}\ L\ C &\triangleq \exists b (element\ \bigvee b\ L\ \wedge\ \forall x\ seq_I\ ((b\,x) :: L)\ C)
\end{aligned}
$$

Proving some of the meta-theory of the object-logic

Consider proving the fact that if the object-level sequent

$$B \supset A \longrightarrow A$$

is provable, then the sequent

$$B \supset A \longrightarrow B$$

must also be provable.

This can be formalized in $FO\lambda^{\Delta\mathbb{N}}$, along with the previously mentioned types and definitions, to be the following meta-logic statement.

$$\longrightarrow \forall a \forall b (\exists i \ seq_i \ ((b \Rightarrow \langle a \rangle) :: nil) \ \langle a \rangle \supset \exists j \ seq_j \ ((b \Rightarrow \langle a \rangle) :: nil) \ b) \ .$$

Applying the $\forall\mathcal{R}$, $\supset\mathcal{R}$, and $\exists\mathcal{L}$ rules yields the sequent

$$seq_i \ ((b \Rightarrow \langle a \rangle) :: nil) \ \langle a \rangle \longrightarrow \exists j \ seq_j \ ((b \Rightarrow \langle a \rangle) :: nil) \ b \ .$$

Now we apply the *def*$\mathcal{L}$ rule to the judgement on the left, which yields four sequents, again corresponding to the cases where the derivation of a ends with the initial rule or any of the three left rules:

$$element\ \langle a\rangle\ ((b \Rightarrow \langle a\rangle) :: nil) \longrightarrow \exists j\ seq_j\ ((b \Rightarrow \langle a\rangle) :: nil)\ b$$

$$\begin{aligned}
&\exists b' \exists c' (element\ (b' \Rightarrow c')\ ((b \Rightarrow \langle a\rangle) :: nil) \wedge \\
&\qquad seq_{i'}\ (c' :: (b \Rightarrow \langle a\rangle) :: nil)\ \langle a\rangle \wedge \\
&\qquad\qquad seq_{i'}\ ((b \Rightarrow \langle a\rangle) :: nil)\ b') \longrightarrow \exists j\ seq_j\ ((b \Rightarrow \langle a\rangle) :: nil)\ b
\end{aligned}$$

$$\begin{aligned}
&\exists b' (element\ \bigwedge b'\ ((b \Rightarrow \langle a\rangle) :: nil) \wedge \\
&\quad \exists x\ seq_{i'}\ ((b'\ x) :: (b \Rightarrow \langle a\rangle) :: nil)\ \langle a\rangle) \longrightarrow \exists j\ seq_j\ ((b \Rightarrow \langle a\rangle) :: nil)\ b
\end{aligned}$$

$$\begin{aligned}
&\exists b' (element\ \bigvee b'\ ((b \Rightarrow \langle a\rangle) :: nil) \wedge \\
&\quad \forall x\ seq_{i'}\ ((b'\ x) :: (b \Rightarrow \langle a\rangle) :: nil)\ \langle a\rangle) \longrightarrow \exists j\ seq_j\ ((b \Rightarrow \langle a\rangle) :: nil)\ b
\end{aligned}$$

.

Three of these sequents will be easy to prove by having $\mathit{def}\mathcal{L}$ recognize that the *element* predicate call fails to hold. Only one sequent is not trivial, namely,

$$\begin{array}{l}\exists b' \exists c' (\mathit{element}\ (b' \Rightarrow c')\ ((b \Rightarrow \langle a \rangle) :: \mathit{nil}) \wedge \\ \qquad\qquad \mathit{seq}_{i'}\ (c' :: (b \Rightarrow \langle a \rangle) :: \mathit{nil})\ \langle a \rangle \wedge \\ \qquad\qquad\qquad \mathit{seq}_{i'}\ ((b \Rightarrow \langle a \rangle) :: \mathit{nil})\ b') \longrightarrow \exists j\ \mathit{seq}_j\ ((b \Rightarrow \langle a \rangle) :: \mathit{nil})\ b\end{array}$$

In this case, $b' \Rightarrow c'$ does match the first element of the list, so we must consider the case where the left $\Rightarrow$ rule was applied to $(b \Rightarrow \langle a \rangle)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\top, \mathit{seq}_{i'}\ ((b \Rightarrow \langle a \rangle) :: \mathit{nil})\ b \longrightarrow \exists j \ldots \qquad \dfrac{}{\mathit{element}\ (b' \Rightarrow c')\ \mathit{nil}, \ldots \longrightarrow \exists j \ldots}\ \mathit{def}\mathcal{L}}{\mathit{element}\ (b' \Rightarrow c')\ ((b \Rightarrow \langle a \rangle) :: \mathit{nil}), \mathit{seq}_{i'}\ ((b \Rightarrow \langle a \rangle) :: \mathit{nil})\ b' \longrightarrow \exists j \ldots}\ \mathit{def}\mathcal{L}}{\mathit{element}\ (b' \Rightarrow c')\ \ldots \wedge \ldots, \mathit{element}\ (b' \Rightarrow c')\ \ldots \wedge \ldots \longrightarrow \exists j \ldots}\ \wedge\mathcal{L}}{\mathit{element}\ (b' \Rightarrow c')\ ((b \Rightarrow \langle a \rangle) :: \mathit{nil}) \wedge \ldots \longrightarrow \exists j\ \mathit{seq}_j\ ((b \Rightarrow \langle a \rangle) :: \mathit{nil})\ b}\ c\mathcal{L}}{\ldots \longrightarrow \exists j\ \mathit{seq}_j\ ((b \Rightarrow \langle a \rangle) :: \mathit{nil})\ b}\ \exists\mathcal{L} \quad .$$

But the unproved sequent is easily derived by choosing j to be i':

$$\dfrac{\dfrac{}{\top, \mathit{seq}_{i'}\ ((b \Rightarrow \langle a \rangle) :: \mathit{nil})\ b \longrightarrow \mathit{seq}_{i'}\ ((b \Rightarrow \langle a \rangle) :: \mathit{nil})\ b}\ \mathit{init}}{\top, \mathit{seq}_{i'}\ ((b \Rightarrow \langle a \rangle) :: \mathit{nil})\ b \longrightarrow \exists j\ \mathit{seq}_j\ ((b \Rightarrow \langle a \rangle) :: \mathit{nil})\ b}\ \exists\mathcal{R} \quad .$$

Another example and a problem

Suppose we know that the sequent

$$\longrightarrow \bigwedge y_1 \bigwedge y_2 (p\, y_1\, t_1 \Rightarrow p\, y_2\, t_2 \Rightarrow p\, y_2\, t_3)$$

is derivable in intuitionistic logic (not $FO\lambda^{\Delta\mathbb{N}}$) for some predicate constant p and some terms t_1, t_2, and t_3.

The derivation must end with applications of the right rules for $\bigwedge$ and $\Rightarrow$.

Thus we know that the sequent

$$p\, y_1\, t_1, p\, y_2\, t_2 \longrightarrow p\, y_2\, t_3$$

is derivable. Since p is a predicate constant, these formulas are all atomic, so the only rule that applies is the initial rule.

The eigenvariable condition for the application of the right rule for $\bigwedge$ guarantees that y_1 and y_2 are distinct, so the initial rule must apply to the second hypothesis.

Therefore, it must be the case that t_2 and t_3 are the same term.

Now let us try to capture this reasoning in $FO\lambda^{\Delta\mathbb{N}}$ using our current encoding of intuitionistic logic.

To do this, we will need some way to indicate term identity, and so we introduce the predicate $\equiv$ of type $i \to i \to o$ defined by the clause

$$X \equiv X \triangleq \top .$$

We then want to derive the sequent

$$\longrightarrow \forall p \forall t_1 \forall t_2 \forall t_3 (\exists i\ seq_i\ nil\ \bigwedge y_1 \bigwedge y_2 (\langle p\, y_1\, t_1 \rangle \Rightarrow \langle p\, y_2\, t_2 \rangle \Rightarrow \langle p\, y_2\, t_3 \rangle) \supset t_2 \equiv t_3) \quad .$$

The only way to proceed is by applying $\forall\mathcal{R}$ and $\supset\mathcal{R}$, yielding

$$\exists i\ seq_i\ nil\ \bigwedge y_1 \bigwedge y_2 (\langle p\, y_1\, t_1 \rangle \Rightarrow \langle p\, y_2\, t_2 \rangle \Rightarrow \langle p\, y_2\, t_3 \rangle) \longrightarrow t_2 \equiv t_3 \quad .$$

There is nothing more that we can do on the right, since the definitional clause for $\equiv$ does not apply. Applying $\exists\mathcal{L}$ gives us the sequent

$$seq_i\ nil\ \bigwedge y_1 \bigwedge y_2 (\langle p\, y_1\, t_1 \rangle \Rightarrow \langle p\, y_2\, t_2 \rangle \Rightarrow \langle p\, y_2\, t_3 \rangle) \longrightarrow t_2 \equiv t_3 \quad .$$

Now we want to reason about the derivation of $\bigwedge y_1 \bigwedge y_2 \ldots$ to conclude that $t_2 \equiv t_3$. In the informal proof, we reasoned that this derivation must end with the right rule for $\bigwedge$; we do the same thing here using $def\mathcal{L}$, which yields the sequent

$$\forall y_1\ seq_{i_1}\ nil \bigwedge y_2(\langle p\, y_1\, t_1\rangle \Rightarrow \langle p\, y_2\, t_2\rangle \Rightarrow \langle p\, y_2\, t_3\rangle) \longrightarrow t_2 \equiv t_3 \quad ,$$

as well as three other sequents corresponding to the cases where the object logic derivation ends with the application of one of the left rules.

Since these latter three sequents represent cases that are not applicable, they are easily derivable as shown in the previous example; we thus focus on the sequent shown above.

Before we can proceed to apply $def\mathcal{L}$ again for the second use of the right rule for $\bigwedge$, we must first apply $\forall\mathcal{L}$, which requires supplying a substitution term for y_1. For this proof, it doesn't matter what term we use for y_1, as long as it is something that does not unify with the term we supply for y_2.

So let x_1 and x_2 be two distinct, non-unifiable terms of type i.

If we use x_1 for y_1, and then apply $def\mathcal{L}$ and $\forall\mathcal{L}$ again using x_2 for y_2, we get

$$seq_{i_2}\ nil\,(\langle p\, x_1\, t_1\rangle \Rightarrow \langle p\, x_2\, t_2\rangle \Rightarrow \langle p\, x_2\, t_3\rangle) \longrightarrow t_2 \equiv t_3 \quad .$$

We now apply $\mathit{def}\mathcal{L}$ two more times, each of which corresponds to reasoning that the object logic derivation must proceed with a use of the right rule for $\Rightarrow$. This yields the sequent

$$\mathit{seq}_{i_3}\ (\langle p\, x_2\, t_2\rangle :: \langle p\, x_1\, t_1\rangle :: \mathit{nil})\ \langle p\, x_2\, t_3\rangle \longrightarrow t_2 \equiv t_3 \quad .$$

Another application of $\mathit{def}\mathcal{L}$ reflects the fact that in the object logic derivation only the initial rule now applies:

$$\mathit{element}\ \langle p\, x_2\, t_3\rangle\ (\langle p\, x_2\, t_2\rangle :: \langle p\, x_1\, t_1\rangle :: \mathit{nil}) \longrightarrow t_2 \equiv t_3 \quad .$$

For $\langle p\, x_2\, t_3\rangle$ to be the first element of the list, t_2 and t_3 must be the same, and this is what we want to prove. We have chosen x_1 and x_2 to be terms that do not unify, so $\langle p\, x_2\, t_3\rangle$ cannot be the other element of the list. This reasoning is represented formally by the $FO\lambda^{\Delta\mathbb{N}}$ derivation

$$\dfrac{\dfrac{\dfrac{}{\top \longrightarrow \top}\ \top\mathcal{R}}{\top \longrightarrow t_2 \equiv t_2}\ \mathit{def}\mathcal{R} \qquad \dfrac{\dfrac{}{\mathit{element}\ \langle p\, x_2\, t_3\rangle\ \mathit{nil} \longrightarrow t_2 \equiv t_3}\ \mathit{def}\mathcal{L}}{\mathit{element}\ \langle p\, x_2\, t_3\rangle\ (\langle p\, x_1\, t_1\rangle :: \mathit{nil}) \longrightarrow t_2 \equiv t_3}\ \mathit{def}\mathcal{L}}{\mathit{element}\ \langle p\, x_2\, t_3\rangle\ (\langle p\, x_2\, t_2\rangle :: \langle p\, x_1\, t_1\rangle :: \mathit{nil}) \longrightarrow t_2 \equiv t_3}\ \mathit{def}\mathcal{L} \quad .$$

But this is, of course, rather unfortunate

If we are able to construct the two non-unifiable terms x_1 and x_2, we are able to conduct this analysis in $FO\lambda^{\Delta\mathbb{N}}$.

But the need for these two terms is rather disturbing.

The informal proof is independent of the type of y_1 and y_2 and the term structure of this type.

In fact, the informal proof is valid even for a type that is uninhabited; this is obviously not the case for our representation in $FO\lambda^{\Delta\mathbb{N}}$.

The problem is that our representation of object-level quantification in terms of $FO\lambda^{\Delta\mathbb{N}}$ quantification doesn't allow us to examine a derivation that is generic over certain terms.

Although the formula $\forall y\ seq_i\ L\ (B\, y)$ indicates that the proposition $B\, y$ is derivable from the hypotheses in L for any y, it does not indicate that the derivation is the same for all y, and we cannot examine that derivation generically.

All we can do is use the $\forall\mathcal{L}$ rule, which requires us to substitute a specific term for y, and then examine the derivation for that specific term.

Explicit eigenvariable encoding

To solve this problem we must explicitly keep track of the eigenvariables introduced by the quantifier rules. We do not wish to abandon, however, our higher-order abstract syntax representation of quantification.

Naively, we might like to replace the quantification by λ-abstraction in the rule for $\bigwedge$:

$$seq_{(S(I))}\ L\ (\bigwedge B) \quad \triangleq \quad \lambda x\ seq_I\ L\ (B\,x)\ .$$

This does not work, of course, since the body of this clause now has type $i \to o$ instead of type o.

Also, we would need to address "judgments" with varying numbers of abstractions, such as,

$$\lambda x_1 \ldots \lambda x_n\ seq_I\ (L\,x_1 \ldots x_n)\ (B\,x_1 \ldots x_n)$$

for arbitrary $n \geq 0$.

Making a list context of the eigenvariables

First consider "uncurrying" this expression by replacing the λ-abstractions over $x_1, \ldots, x_n$ by a single λ-abstraction over the n-tuple $(x_1, \ldots, x_n)$:

$$\lambda x.seq_I\ (L\,(\pi_1\ x) \ldots (\pi_n\ x))\ (B\,(\pi_1\ x) \ldots (\pi_n\ x))\ \ .$$

Now we can deal with the arbitrary n by replacing the n-tuple with a list, and using $fst\ x$ in place of $\pi_1\ x$, $fst\ (rst\ x)$ in place of $\pi_2\ x$, $fst\ (rst\ (rst\ x))$ in place of $\pi_3\ x$, etc.

Finally, we push the λ-abstraction into the *seq* predicate by changing its type:

$$\mathit{seq} \quad : \quad \mathit{nt} \to (\mathit{evs} \to \mathit{prplst}) \to (\mathit{evs} \to \mathit{prp}) \to o\ \ .$$

Here evs is a new type representing a list of eigenvariables.

We have already seen the two operators on this type, $fst\colon evs \to i$ and $rst\colon evs \to evs$; $fst\ l$ represents the first eigenvariable in the list l, and $rst\ l$ represents the remainder of the list.

The right rule for $\bigwedge$ is now encoded as follows:

$$seq_{(S(I))}\, L\, (\lambda l \bigwedge x (B\, l\, x)) \quad \triangleq \quad seq_I\, (\lambda l'\, L (rst\; l'))\, (\lambda l'\, B\, (rst\; l')\, (fst\; l')) \;\; .$$

The bound variable l' in the body of the clause should be thought of as a list whose length is one longer than the length of the bound variable l in the head of the clause; $fst\; l'$ represents the new eigenvariable, and $rst\; l'$ represents the eigenvariables in l. The left rule for $\bigvee$ is similarly modified:

$$\begin{aligned} seq_{(S(I))}\, L\, C \quad \triangleq \quad & \exists b (element\, (\lambda l \bigvee x (b\, l\, x))\, L \wedge \\ & \qquad seq_I\, (\lambda l'\, (b\, (rst\; l')\, (fst\; l')) :: (L\, (rst\; l')))\, (\lambda l'\, C\, (rst\; l')) \;\; . \end{aligned}$$

The remainder of the clauses are only modified to reflect the change in the type of *seq*.

The type of the predicate *element* also changes:

$$element \quad : \quad (evs \to prp) \to (evs \to prplst) \to o \;\; .$$

$$seq_I\ L\ \lambda l\ \langle (A\,l)\rangle \triangleq element\ \lambda l\ \langle (A\,l)\rangle\ L$$

$$\begin{aligned}
seq_{(S(I))}\ L\ \lambda l\ ((B\,l) \Rightarrow (C\,l)) &\triangleq seq_I\ \lambda l\ ((B\,l) :: (L\,l))\ C \\
seq_{(S(I))}\ L\ (\lambda l \bigwedge x (B\,l\,x)) &\triangleq seq_I\ (\lambda l'\ L\,(rst\ l'))\ (\lambda l'\ B\,(rst\ l')\,(fst\ l')) \\
seq_{(S(I))}\ L\ (\lambda l \bigvee x (B\,l\,x)) &\triangleq \exists x\ seq_I\ L\ (\lambda l\ B\,l\,(x\,l))
\end{aligned}$$

$$\begin{aligned}
seq_{(S(I))}\ L\ D &\triangleq \exists b \exists c (element\ \lambda l\ ((b\,l) \Rightarrow (c\,l))\ L \wedge \\
&\qquad seq_I\ \lambda l\ ((c\,l) :: (L\,l))\ D \wedge \\
&\qquad seq_I\ L\ b) \\
seq_{(S(I))}\ L\ C &\triangleq \exists b (element\ (\lambda l \bigwedge x (b\,l\,x))\ L \wedge \\
&\qquad \exists x\ seq_I\ \lambda l\ ((b\,l\,(x\,l)) :: (L\,l))\ C) \\
seq_{(S(I))}\ L\ C &\triangleq \exists b (element\ (\lambda l \bigvee x (b\,l\,x))\ L \wedge \\
&\qquad seq_I\ \lambda l'\ ((b\,(rst\ l')\,(fst\ l')) :: (L\,(rst\ l')))\ (\lambda l'\ C\,(rst\ l')))
\end{aligned}$$

$$\begin{aligned}
element\ X\ \lambda l\ ((X\,l) :: (L\,l)) &\triangleq \top \\
element\ X\ \lambda l\ ((Y\,l) :: (L\,l)) &\triangleq element\ X\ L
\end{aligned}$$

A smaller subset of intuitionistic logic

Consider the fragment of second-order intuitionistic logic given by the grammar

$$
\begin{array}{rcl}
D & ::= & A \mid G \Rightarrow A \mid \bigwedge_{\alpha} x.D \mid \bigwedge_{\alpha\to\alpha} x.D \\
G & ::= & A \mid tt \mid G \,\&\, G \mid A \Rightarrow G \mid \bigwedge_{\alpha} x.G \ ,
\end{array}
$$

where A ranges over atomic formulas and α ranges over ground types. D and G represent definite clauses and goal formulas, respectively.

Although this seems like a rather simple fragment, higher-order abstract syntax encodings generally fall within the set of definite clauses given by this grammar.

The set of goal formulas can be encoded using the following constants:

$$
\begin{array}{rclrclrcl}
\langle\,\rangle & : & atm \to prp & \& & : & prp \to prp \to prp & \bigwedge & : & (i \to prp) \to prp \\
tt & : & prp & \Rightarrow & : & atm \to prp \to prp & & & .
\end{array}
$$

Notice that the antecendent of the implication is restricted to be atomic.

The sequent calculus

$$seq \quad : \quad nt \to atmlst \to prp \to o \ ,$$

where *atmlst* is the same as the type *lst* except we use *atm* for the type of elements.

Since the antecedents are atomic, only the initial and right rules are necessary:

$$\begin{array}{rcl}
seq_I\ (A' :: L)\ \langle A \rangle & \triangleq & element\ A\ (A' :: L) \\
seq_I\ L\ tt & \triangleq & \top \\
seq_{(S(I))}\ L\ (B\ \&\ C) & \triangleq & seq_I\ L\ B\ \wedge\ seq_I\ L\ C \\
seq_{(S(I))}\ L\ (A \Rightarrow B) & \triangleq & seq_I\ (A :: L)\ B \\
seq_{(S(I))}\ L\ (\bigwedge B) & \triangleq & \forall x\ seq_I\ L\ (B\,x) \\
seq_{(S(I))}\ L\ \langle A \rangle & \triangleq & \exists b (prog\ A\ b \wedge seq_I\ L\ b)
\end{array}$$

Here, the object-level logic program is stored in the *prog* predicate.

Admissibility of Rules for Intuitionistic Object Logic

The following formulas are derivable in $FO\lambda^{\Delta\mathbb{N}}$ using the definition we have seen so far.

Specialization Rule:

$$\forall i \forall b \forall l (nat(i) \supset seq_{(S(i))}\, l \bigwedge b \supset \forall x\, seq_i\, l\, (b\, x))$$

Cut Rule:

$$\forall a \forall b \forall l ((a :: l) \vdash b \supset l \vdash \langle a \rangle \supset l \vdash b)$$

Structural Rules:

$$\forall i \forall b \forall l \forall l' (nat(i) \supset \forall a (element\ a\ l \supset element\ a\ l') \supset seq_i\ l\ b \supset seq_i\ l'\ b)$$