Lecture 3: Induction for sequent calculus

Dale Miller

September 2000

Natural number induction

We incorporate induction by introducing natural numbers using $z : nt$ for zero and s: $nt \to nt$ for successor and using the predicate nat: $nt \to o$. The rules for this new predicate are

$$\frac{}{\Gamma \longrightarrow nat(z)}\ nat\mathcal{R} \qquad\qquad \frac{\Gamma \longrightarrow nat(I)}{\Gamma \longrightarrow nat(s(I))}\ nat\mathcal{R}$$

$$\frac{\longrightarrow B\,z \quad B\,j \longrightarrow B\,(s(j)) \quad B\,I, \Gamma \longrightarrow C}{nat(I), \Gamma \longrightarrow C}\ nat\mathcal{L}$$

Here I, B, and C are schematic variables of these inference rules, and j is a variable not free in B (an eigenvariable).

The left-introduction rule encodes induction: B represents the property that is proved by induction; the third premise of that inference rule witnesses the fact that, in general, B will express a property stronger than C.

Intuitionistic Peano arithmetic

The axiom schemes for IPA can be grouped into the following collections.

1. Axioms for first-order intuitionistic logic.
2. Axioms for equality: reflexivity, symmetry, transitivity, and substitution.
3. The two formulas

$$\forall x \forall y (s(x) = s(y) \supset x = y) \quad \text{and} \quad \forall x (z = s(x) \supset \perp) \ .$$

4. The axioms of induction: all formulas of the form

$$\varphi(z) \wedge \forall j (\varphi(j) \supset \varphi(s(j))) \supset \forall x \varphi(x) \ ,$$

   where $\varphi(x)$ ranges over formulas with at most the variable x free.

There are two inference rules for IPA: *Modus Ponens* allows the formula B to be inferred from the formulas $A \supset B$ and A, while *Universal Generalization* allows the formula $\forall x B$ to be inferred from B.

Strength of $FO\lambda^{\Delta\mathbb{N}}$

$FO\lambda^{\Delta\mathbb{N}}$ is at least as strong as IPA and the only definition that is needed to show this is

$$I = I \triangleq \top \ .$$

I leave it as an exercise to show that the axioms of equality and for zero and successor can be proved simply in $FO\lambda^{\Delta\mathbb{N}}$.

Showing that induction follows from $nat\mathcal{L}$ is only a bit harder since in IPA inductions can be applied to all variables and in $FO\lambda^{\Delta\mathbb{N}}$ it is only applied to *nat* predicates. This too is easy enough to address.

Cut-elimination and the subformula property

Cut-elimination holds for $FO\lambda^{\Delta\mathbb{N}}$ [McDowell & Miller].

Cut-elimination does yield a subformula property since the invariant formula B used in the $nat\mathcal{L}$ rule is not necessarily a subformula of the conclusion of that inference rule.

In fact, the following inference rule is instance of $nat\mathcal{L}$:

$$\frac{\longrightarrow B \quad B, \Gamma \longrightarrow C}{nat(I), \Gamma \longrightarrow C}$$

This inference rule resembles the cut rule except that it requires a *nat* assumption.

The cut-elimination theorem still provides a strong basis for reasoning about proofs in $FO\lambda^{\Delta\mathbb{N}}$.

Also this formulation of the induction principle is natural and close to the one used in actual mathematical practice: that is, invariants must be, at times, clever inventions that are not simply rearrangements of subformulas.

Complete induction

Theorem. For any formula $C : o$, predicate $B : nt \to o$, term $I : nt$, multiset Δ of formulas, and eigenvariable $j : nt$ such that j is not free in B, the following rule is derivable in $FO\lambda^{\Delta\mathbb{N}}$ using the definition for less-than:

$$\frac{nat(j), \forall k(nat(k) \supset k < j \supset B\,k) \longrightarrow B\,j \quad B\,I, \Delta \longrightarrow C}{nat(I), \Delta \longrightarrow C} \,.$$

Examples

The following formulas are derivable in $FO\lambda^{\Delta\mathbb{N}}$ using the definition for less-than:

$$\forall i(nat((s(i))) \supset nat(i))$$

$$\forall i(nat(i) \supset \forall j(i < j \supset nat(j)))$$

$$\forall i(nat(i) \supset i < (s(i)))$$

$$\forall i(nat(i) \supset \forall j(i < (s(j)) \supset i \leq j))$$

$$\forall i(nat(i) \supset \forall j \forall k(i < j \supset j < k \supset i < k))$$

$$\forall i(nat(i) \supset \forall j(nat(j) \supset \exists k(nat(k) \wedge i < k \wedge j < k)))$$

$$\forall i(nat(i) \supset \forall j \forall k(sum\, i\ (s(j))\ k \supset sum\ (s(i))\ j\ k))$$

$$\forall i(nat(i) \supset \forall j(nat(j) \supset \exists k(nat(k) \wedge sum\ i\ j\ k)))$$

$$\forall i(nat(i) \supset \forall j \forall k(nat(j) \supset sum\ i\ j\ k \supset i \leq k))$$

$$\forall i(nat(i) \supset \forall j \forall k(nat(j) \supset sum\ (s(i))\ j\ k \supset j < k))\ .$$

Revisiting simulation and bisimulation

Let ϕ_s and ϕ_b be the $\mathcal{T}_{\mathcal{D}}$-related predicate for simulation and bisimulation.

Define the following

- $\sqsubseteq_0$ to be the universal relation on states (that is, $S \times S$).
- $\sqsubseteq_{i+1} := \phi_s(\sqsubseteq_i)$.
- $\sqsubseteq_\omega \;:=\; \bigcap_i \sqsubseteq_i$.
- $\equiv_0$ to be the universal relation on states (that is, $S \times S$).
- $\equiv_{i+1} := \phi_s(\equiv_i)$.
- $\equiv_\omega \;:=\; \bigcap_i \equiv_i$.

It is easy to show that $\sqsubseteq \subseteq \sqsubseteq_\omega$ and $\equiv \subseteq \equiv_\omega$.

The converses are also true when the ats is finitely branching.

Finitely branching in CCS is guarenteed if all recursion variables in μ-expressions are prefixed.

Definition of the stratification relations

$$
\begin{array}{rcl}
sim(P,Q) & \triangleq & \forall K.nat(K) \supset ssim(K,P,Q) \\
ssim(z,P,Q) & \triangleq & \top \\
ssim((s\ K),P,Q) & \triangleq & \forall A \forall P'.\ one(P,A,P') \supset \\
 & & \qquad \exists Q'.\ one(Q,A,Q') \wedge ssim(K,P',Q') \\
bisim(P,Q) & \triangleq & \forall K.nat(K) \supset sbisim(K,P,Q) \\
sbisim(z,P,Q) & \triangleq & \top \\
sbisim((s\ K),P,Q) & \triangleq & [\forall A \forall P'.\ one(P,A,P') \supset \\
 & & \qquad \exists Q'.\ one(Q,A,Q') \wedge sbisim(K,P',Q')] \wedge \\
 & & [\forall A \forall Q'.\ one(Q,A,Q') \supset \\
 & & \qquad \exists P'.\ one(P,A,P') \wedge sbisim(K,Q',P')]
\end{array}
$$

The usual theorems are now provable

Consider, for example, CCS. Then the following are some simple example of provable formulas.

$$\forall P \ \mathit{bisim}(P + 0, P)$$

$$\forall P \ \mathit{bisim}(P + P, P)$$

$$\forall P \ \mathit{bisim}(P, P)$$

$$\forall P \forall Q \forall R \ \mathit{bisim}(P + 0, P) \supset \mathit{bisim}(P + R, Q + R)$$

Interesting challenge: Can this be generalized to, say, the π-calculus.

One challenge here is the proper treatment of bound names (λ-tree syntax useful here?).

Another challenge is determining which bisimulations (and there are several) can be properly encoded.

Example: a simple theorem

$$\forall x \forall y (nat(x) \supset sum\ x\ z\ y \supset x = y).$$

Prove the sequent

$$nat(x), sum\ x\ z\ y \longrightarrow x = y.$$

Chose the invariant $B = \lambda w.\ \forall y.\ sum\ w\ z\ y \supset w = y$. The base case and inductive case are proved below. The third case is immediate.

$$\dfrac{\dfrac{\dfrac{}{\longrightarrow z = z}\ def\mathcal{R}}{sum\ z\ z\ y \longrightarrow z = y}\ def\mathcal{L}}{\longrightarrow \forall y.\ sum\ z\ z\ y \supset z = y}\ \forall\mathcal{R}, \supset\mathcal{R}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{sum\ j\ z\ y' \longrightarrow sum\ j\ z\ y'} \quad \dfrac{\longrightarrow (s\ j) = (s\ j)}{j = y' \longrightarrow (s\ j) = (s\ y')}\ def\mathcal{L}}{sum\ j\ z\ y' \supset j = y', sum\ j\ z\ y' \longrightarrow (s\ j) = (s\ y')}\ \supset\mathcal{L}}{\forall y(sum\ j\ z\ y \supset j = y), sum\ j\ z\ y' \longrightarrow (s\ j) = (s\ y')}\ \forall\mathcal{L}}{\forall y(sum\ j\ z\ y \supset j = y), sum\ (s\ j)\ z\ y \longrightarrow (s\ j) = y}\ def\mathcal{L},\ y \mapsto (s\ y')}{\forall y(sum\ j\ z\ y \supset j = y) \longrightarrow \forall y(sum\ (s\ j)\ z\ y \supset (s\ j) = y)}\ \forall\mathcal{R}, \supset\mathcal{R}$$

Lists

Introduce a type *lst* for lists over an arbitrary but fixed type τ.

This type has two constructors, $nil : lst$ and the infix operator :: of type $\tau \to lst \to lst$.

On the next slide we provide definitions of the following predicates:

$$
\begin{array}{rcl@{\qquad}rcl}
length & : & lst \to nt \to o & split & : & lst \to lst \to lst \to o \\
list & : & lst \to o & permute & : & lst \to lst \to o \\
element & : & \tau \to lst \to o & & &
\end{array}
$$

These predicates will be useful, in part, because the encoding of the left-hand-side of object-level sequent calculus will be as lists and such operations will be needed to manipulate such contexts.

$$
\begin{array}{rcl}
\mathit{length}\ \mathit{nil}\ z & \triangleq & \top \\
\mathit{length}\ (X :: L)\ (s(I)) & \triangleq & \mathit{length}\ L\ I \\
\\
\mathit{list}\ L & \triangleq & \exists i(\mathit{nat}(i) \wedge \mathit{length}\ L\ i) \\
\\
\mathit{element}\ X\ (X :: L) & \triangleq & \top \\
\mathit{element}\ X\ (Y :: L) & \triangleq & \mathit{element}\ X\ L \\
\\
\mathit{split}\ \mathit{nil}\ \mathit{nil}\ \mathit{nil} & \triangleq & \top \\
\mathit{split}\ (X :: L_1)\ (X :: L_2)\ L_3 & \triangleq & \mathit{split}\ L_1\ L_2\ L_3 \\
\mathit{split}\ (X :: L_1)\ L_2\ (X :: L_3) & \triangleq & \mathit{split}\ L_1\ L_2\ L_3 \\
\\
\mathit{permute}\ \mathit{nil}\ \mathit{nil} & \triangleq & \top \\
\mathit{permute}\ (X :: L_1)\ L_2 & \triangleq & \exists l(\mathit{split}\ L_2\ (X :: \mathit{nil})\ l \wedge \mathit{permute}\ L_1\ l\ )
\end{array}
$$

An induction rule for lists

We now derive an induction rule for lists from the induction rule for natural numbers ($nat\mathcal{L}$) using the length of a list as our measure.

For any formula $C : o$, predicate $B : lst \to o$, term $L : lst$, multiset Γ of formulas, and eigenvariables $x : \tau$ and $l : lst$ such that x and l are not free in B, the following rule is derivable in $FO\lambda^{\Delta\mathbb{N}}$ using the definition for lists.

$$\frac{\longrightarrow B\,nil \quad B\,l \longrightarrow B\,(x::l) \quad B\,L, \Gamma \longrightarrow C}{list\ L, \Gamma \longrightarrow C}\ listL.$$

The following rules are also derviable:

$$\frac{}{\Delta \longrightarrow list\ nil}\ listR \qquad \frac{\Delta \longrightarrow list\ L}{\Delta \longrightarrow list\ (X::L)}\ listR.$$

These rules are simple consequences of the definition of the *list* predicate definition.

We can thus move from viewing *list* as a definition to viewing it as a new predicate on which we can do induction.