Program Verification: Lecture 13

José Meseguer

Computer Science Department
University of Illinois at Urbana-Champaign

The ITP Inference Rules

Notice that in the ITP we reason backwards, replacing the main goal G we want to prove by subgoals, $G_1, \ldots, G_n$, such that if we prove each of the subgoals, then we have proved the main goal.

For such an inference to be sound, the implication

$$G_1 \wedge \ldots \wedge G_n \Rightarrow G$$

should always be satisfied, that is, should be semantically valid in the initial algebra $\mathcal{T}_{\Sigma,E}$ on which we are doing the inductive reasoning.

The ITP Inference Rules (II)

Such semantically valid inferences are expressed as inference rules

$$\frac{G_1 \quad \ldots \quad G_n}{G}$$

However, since we are reasoning backwards, from the root of the proof tree to the leaves, the ITP uses such rules in the opposite direction, as rules

$$\frac{G}{G_1 \quad \ldots \quad G_n}$$

We will illustrate through an example such backward reasoning for several ITP inference rules besides the induction rule, and will at the same time justify their soundness.

Linearity of the Number Ordering

Consider the following module defining the order on numbers, which we would like to prove is linear.

```
fmod NATURAL-ORD is
  sort Natural .
  op 0 : -> Natural [ctor] .
  op s : Natural -> Natural [ctor] .
  op _<_ : Natural Natural -> Bool .
  op _=<_ : Natural Natural -> Bool .
  vars N M : Natural .
  eq N < 0 = false .
  eq 0 < s(N) = true .
  eq s(N) < s(M) = N < M .
  ceq N =< M = true if N < M .
  ceq N =< M = true if not(M < N) .
  ceq N =< M = false if M < N .
endfm
```

The `cns` Inference Rule

After entering the goal stating that the order on the naturals is linear, one of the possible ITP inference rules we can invoke is the lemma of constants, which converts universally quantified variables in a goal into constants.

```
Maude> (goal linear : NATURAL-ORD |- A{N:Natural ; M:Natural}
       (((N =< M) or (M =< N)) = (true)) .)

=================================
label-sel: linear@0
=================================
A{N:Natural ; M:Natural}
N:Natural =< M:Natural or M:Natural =< N:Natural = true
+++++++++++++++++++++++++++++++++

Maude> (cns .)
```

```
=================================
label-sel: linear@0
=================================
N*Natural =< M*Natural or M*Natural =< N*Natural = true
+++++++++++++++++++++++++++++++++
```

The $\mathtt{cns}$ Inference Rule (II)

The fact that this is a semantically valid inference is based on the **Constants Lemma**, which states the equivalence between satisfiability of a quantified equation, and of the same equation with the variables transformed into generic constants (we assume $vars(E) \cap vars(t = t') = \emptyset$):

$$E \models_{\Sigma} t = t' \quad \Leftrightarrow \quad E \models_{\Sigma(X)} t = t'.$$

Thanks to the completeness of equational reasoning, this is expressed in the ITP as the $\mathtt{cns}$ rule,

$$\frac{E \vdash_{\Sigma} t = t'}{E \vdash_{\Sigma(X)} t = t'}$$

which is an equivalence by the rewriting definition of $=_E$.

Reasoning by Cases: The `split` Rule

To prove our goal, we can now reason by cases. For each pair of natural numbers n, m, either $n < m = \texttt{true}$, or $n < m = \texttt{false}$. Therefore, we can split a goal involving n and m into two subgoals: one in which we assume $n < m = \texttt{true}$ as an extra hypothesis, and another in which we assume $n < m = \texttt{false}$.

In the ITP this is accomplished by the `split` rule, which, given an unquantified goal without variables (only "generic" constants like n, m) and given a Boolean-valued expression involving some of those generic constants, splits a given goal into two: one assuming the expression `true`, and another assuming it `false`.

Reasoning by Cases: The `split` Rule (II)

In our example we can give the `split` command,

```
Maude> (split on (N*Natural < M*Natural) . )

=================================
label-sel: linear@1.0
=================================
N*Natural =< M*Natural or M*Natural =< N*Natural = true

=================================
label: linear@2.0
=================================
N*Natural =< M*Natural or M*Natural =< N*Natural = true
+++++++++++++++++++++++++++++++++
```

Reasoning by Cases: The `split` Rule (III)

The goals remain the same, but the split hypotheses for each case have been added to each case's module. Using these hypotheses we can now discharge each of the subgoals with the `auto` tactic.

```
Maude> (auto .)

=================================
label-sel: linear@2.0
=================================
N*Natural =< M*Natural or M*Natural =< N*Natural = true
+++++++++++++++++++++++++++++++++

Maude> (auto .)

q.e.d

+++++++++++++++++++++++++++++++++
```

Caveats on the `split` Rule

As already mentioned, the present version of the ITP requires that the goal to which the `split` rule is applied is an unquantified goal without variables, in which only "generic" constants —such as `N*Natural` and `M*Natural` in our example— appear.

This requirement will be relaxed in future ITP versions, but it is assumed and required by the present version. Therefore, applications of `split` to quantified goals with variables are currently forbidden: in the present ITP version we must first transform such variables into generic constants using `cns`.

Applications of `split` Should Protect `BOOL`

Regardless of the current ITP restrictions in the application of `split`, there is a fundamental way in which the application of `split` would be unsound, namely, if we have messed up the Booleans by adding junk and perhaps confusion to them, so that we do not have anymore two different canonical forms, `true`, and `false`, but may have other nonstandard Boolean elements as well.

This can happen because, when defining new predicates with operators of sort `Bool`, we do not give enough equations, thus adding "junk" to `Bool`, or we give the wrong equations, adding "confusion" and possibly also "junk" to `Bool`. Therefore, for an application of `split` to be sound, it is enough to require that the `BOOL` submodule is protected.

Protecting Module Importations

In general, if we have a theory (Σ, E) having a subtheory (Σ', E') with,

$$\Sigma' \subseteq \Sigma \quad \text{and} \quad E' \subseteq E,$$

and the module $\texttt{fmod}(\Sigma, E)\texttt{endfm}$ imports the submodule $\texttt{fmod}(\Sigma', E')\texttt{endfm}$ in `protecting` mode, we require that the unique Σ'-homomorphism

$$-^{E'}_{\mathcal{T}_{\Sigma/E}|_{\Sigma'}} : \mathcal{T}_{\Sigma'/E'} \longrightarrow \mathcal{T}_{\Sigma/E}|_{\Sigma'}$$

is an isomorphism.

Protecting Module Importations (II)

Checking that `BOOL` is protected in a supermodule `fmod`(Σ, E)`endfm` is quite easy, since it reduces to checking that:

1. In (Σ, E) the only constructors of sort `Bool` are `true` and `false`.

2. (Σ, E) is ground confluent, terminating, sort decreasing, and sufficiently complete relative to the given signature Ω of declared constructors; and

3. `true` and `false` are in E-canonical form.

Justification of the `split` Rule

Suppose that we are reasoning inductively about a module `fmod`(Σ, E)`endfm`, which correctly imports `BOOL` in `protecting` mode (**Note**: this must be checked independently, as an implicit proof obligation).

In `BOOL` we have,

$$\mathcal{T}_{\mathtt{BOOL}} \models \mathtt{true} \neq \mathtt{false}$$

and we also have,

$$\mathcal{T}_{\mathtt{BOOL}} \models (\forall x : \mathtt{Bool})\ x = \mathtt{true} \vee x = \mathtt{false}$$

Notice that, by the `protecting` importation, we have, $\mathcal{T}_{\Sigma/E}|_{\Sigma_{\mathtt{BOOL}}} \cong \mathcal{T}_{\mathtt{BOOL}}$.

Justification of the `split` Rule (II)

Therefore, given any Boolean valued Σ-term $p \in T_\Sigma(X)_{\texttt{Bool}}$, and given any assignment $a : X \longrightarrow T_{\Sigma/E}$, we have,

$$(\mathcal{T}_{\Sigma/E}, a) \not\models (p \neq \texttt{true} \wedge p \neq \texttt{false}).$$

That is, for any such assignment a, the above proposition is equivalent to the identically false proposition, $\bot$, which is never satisfied:

$$(\mathcal{T}_{\Sigma/E}, a) \models (p \neq \texttt{true} \wedge p \neq \texttt{false}) \quad \Leftrightarrow \quad (\mathcal{T}_{\Sigma/E}, a) \models \bot.$$

Now, since for any proposition A we always have the Boolean equivalence, $A \equiv A \vee \bot$, we also have an equivalence $(\forall X)\ A \equiv (\forall X)\ (A \vee \bot)$. Therefore, given an equation $t = t'$ with $vars(t = t) \subseteq X$ we have,

Justification of the `split` Rule (III)

$$\mathcal{T}_{\Sigma/E} \models t = t' \quad \Leftrightarrow \quad \mathcal{T}_{\Sigma/E} \models ((t = t') \vee \bot)$$

or, equivalently,

$$\mathcal{T}_{\Sigma/E} \models (\forall X)\ t = t' \quad \Leftrightarrow \quad \mathcal{T}_{\Sigma/E} \models (\forall X)\ ((t = t') \vee (p \neq \mathtt{true} \wedge p \neq \mathtt{false}))$$

which, using distributivity of disjunction over conjunction, plus the fact that $A \Rightarrow B \equiv (\neg A) \vee B$, is equivalent to,

$$\mathcal{T}_{\Sigma/E} \models (\forall X)\ t = t' \quad \Leftrightarrow \quad \mathcal{T}_{\Sigma/E} \models (\forall X)\ (p = \mathtt{true} \Rightarrow t = t') \wedge (p = \mathtt{false} \Rightarrow t = t'),$$

which, modulo the distribution of $\forall$ over $\wedge$, is our desired justification for `split` as a sound inductive reasoning rule.

Induction on Other Data Structures: Tree Induction

We have already seen examples of how the ITP's `ind` rule applies to natural number induction and to list induction.

Before discussing the most general form of the `ind` rule for any signature of constructors Ω and its justification, we give an example illustrating binary tree induction, in which the data in leaves are seen as depth-zero trees.

The intuitive idea is that to prove an inductive property P about such trees we must show: (1) that P holds for the data elements (**base case**); and (2) that if P holds for the left and right subtrees, then it must hold for their binary join (**induction step**).

Induction on Other Data Structures: Tree Induction (II)

Consider the following module defining binary trees whose nodes are quoted identifiers (constants in the predefined module `QID`), and a reverse function on binary trees.

```
fmod TREE is
protecting QID .
sort Tree .
subsort Qid < Tree .
op _#_ : Tree Tree -> Tree [ctor] .
op rev : Tree -> Tree .
var I : Qid .
vars T T' : Tree .
eq rev(I) = I .
eq rev(T # T') = rev(T') # rev(T) .
endfm
```

Induction on Other Data Structures: Tree Induction (III)

We can apply binary tree induction to prove that for all trees `T` the equation `rev(rev(T)) = T` holds. We can do so by entering the `TREE` module in the ITP and the goal:

```
Maude> (goal rev : TREE |- A{T:Tree}((rev(rev(T:Tree))) = (T:Tree)) .)

=================================
label-sel: rev@0
=================================
A{T:Tree} rev(rev(T:Tree)) = T:Tree
+++++++++++++++++++++++++++++++++
```

Induction on Other Data Structures: Tree Induction (IV)

We can then try to prove this goal by induction on `T:Tree`.

```
Maude> (ind on T:Tree .)



=================================
label-sel: rev@1.0
=================================
A{V0#0:Qid} rev(rev(V0#0:Qid)) = V0#0:Qid ==>
rev(rev(V0#0:Qid)) = V0#0:Qid

=================================
label: rev@2.0
=================================
A{V0#0:Tree ; V0#1:Tree} rev(rev(V0#1:Tree)) = V0#1:Tree &
rev(rev(V0#0:Tree)) = V0#0:Tree ==>
rev(rev(V0#0:Tree # V0#1:Tree)) = V0#0:Tree # V0#1:Tree
+++++++++++++++++++++++++++++++++
```

Induction on Other Data Structures: Tree Induction (V)

Note that goal `rev@2.0` is the “induction step” in tree induction, whereas the “base case” is goal `rev@1.0`. Both subgoals can then be proved using the `auto` tactic.

```
Maude> (auto .)

==================================
label-sel: rev@2.0
==================================
A{V0#0:Tree ; V0#1:Tree} rev(rev(V0#1:Tree)) = V0#1:Tree &
rev(rev(V0#0:Tree)) = V0#0:Tree ==>
rev(rev(V0#0:Tree # V0#1:Tree)) = V0#0:Tree # V0#1:Tree
++++++++++++++++++++++++++++++++++

Maude> (auto .)

q.e.d
```

Structural Induction

We have already observed how the ITP supports inductive proofs in three cases: natural number induction, list induction, and tree induction. But what is the general form of induction supported by the ITP for a specification having a subsignature Ω of constructors? This general form is called structural induction. It reduces proving an inductive property of the form $(\forall x : s)\ P(x)$, to proving:

- **Base Case**. For any constant $a : nil \longrightarrow s'$ in Ω with $s' \leq s$, the subgoal $P(x \mapsto a)$

  Notation: Given a variable x, the substitution $\{x \mapsto t\}$ mapping x to a term t is abbreviated to $(x \mapsto t)$, and its homomorphic extension is denoted $\_(x \mapsto t)$.

Structural Induction (II)

- **Induction Step**. For each constructor $f : s_1 \dots s_{n_f} \longrightarrow s'$ in Ω with $s' \leq s$, where the sorts $s_{i_1}, \dots, s_{i_{k_f}}$ are those among the $s_1 \dots s_{n_f}$ such that $s_{i_j} \leq s$, $1 \leq j \leq k_f$, the subgoal,

$$\bigwedge_{1 \leq j \leq k_f} (\forall x_{i_j}) P(x \mapsto x_{i_j}) \rightarrow (\forall x_1 : s_1, \dots, x_{n_f} : s_{n_f})\ P(x \mapsto f(x_1, \dots, x_{n_f})).$$

  Note: It may happen that none of the sorts among the $s_1 \dots s_{n_f}$ is s or a subsort of s. In that case, the subgoal has the form $(\forall x_1 : s_1, \dots, x_{n_f} : s_{n_f})\ P(x \mapsto f(x_1, \dots, x_{n_f}))$.

- **Subsorts Without Constructors**. If $s' \leq s$ is a subsort having no constructor constants or operators in a sort $s'' \leq s'$, then we add the subgoal $(\forall y : s')\ P(x \mapsto y)$.

Structural Induction (III)

Note: If the signature Ω of constructors has been fully specified, the base case and the induction step implicitly cover all subsorts, so that the third case should never arise, except perhaps for s' an empty sort, with no terms whatsoever, for which the property P then trivially holds.

Therefore, from now on we will systematically ignore the case of subsorts without constructors in the rest of our theoretical discussions. In practice, however, this case is actually quite useful, and this for two reasons:

Structural Induction (IV)

- to deal with constants in predeclared modules, such as `QID`, which are built-in and are not defined as constructors (we encountered this phenomenon in our tree-reverse example); and

- to generalize these inductive proof methods to parameterized modules, such as `LIST(X)`, where the parametric sort of elements might be a subsort of the sort `List(X)`, but we have no a priori information about the constructors of such a parametric sort of elements.

Structural Induction (V)

Ignoring the case of subsorts without constructors, this then becomes an inductive inference rule of the form,

$$\frac{\bigwedge_i P(x \mapsto a_i) \wedge \bigwedge_l \bigwedge_{1 \leq j \leq k_{f_l}} (\forall x_{i_j}) P(x \mapsto x_{i_j}) \Rightarrow (\forall x_1, \ldots, x_{n_{f_l}}) \ P(x \mapsto f_l(x_1, \ldots, x_{n_{f_l}}))}{(\forall x : s) \ P(x)}$$

where the a_i and the f_j include all the constructor constants and operators meeting the properties specified above.

In the ITP this rule is used backwards as the `ind` rule,

$$\frac{(\forall x : s) \ P(x)}{\bigwedge_i P(x \mapsto a_i) \wedge \bigwedge_l \bigwedge_{1 \leq j \leq k_{f_l}} (\forall x_{i_j}) P(x \mapsto x_{i_j}) \Rightarrow (\forall x_1, \ldots, x_{n_{f_l}}) \ P(x \mapsto f_l(x_1, \ldots, x_{n_{f_l}}))}$$

Justification of the `ind` Rule

Why is `ind` a sound inference rule? First consider:

Lemma: For (Σ, E) ground confluent, sort-decreasing, and terminating with subsignature of constructors Ω, given any Σ-equation $t = t'$ with $X = vars(t = t')$ we have:

$$\mathcal{T}_{\Sigma/E} \models t = t' \quad \Leftrightarrow \quad \forall \theta \in [X \to T_\Omega]\; \mathcal{T}_{\Sigma/E} \models t\theta = t'\theta.$$

Proof: Since $\mathcal{T}_{\Sigma/E} \cong \mathcal{C}_{\Sigma/E}$ it is enough to prove that

$$\mathcal{C}_{\Sigma/E} \models t = t' \quad \Leftrightarrow \quad \forall \theta \in [X \to T_\Omega]\; \mathcal{C}_{\Sigma/E} \models t\theta = t'\theta.$$

But, since $C_{\Sigma/E} \subseteq T_\Omega$, any $a : X \longrightarrow C_{\Sigma/E}$ is a substitution $\theta : X \longrightarrow T_\Omega$, exactly one of the form $\theta = \theta!_E$. Furthermore, for each $\theta \in [X \to T_\Omega]$ we have the equivalence,

$$\mathcal{C}_{\Sigma/E} \models t\theta = t'\theta \quad \Leftrightarrow \quad (t\theta)!_E = (t(\theta!_E))!_E = (t'(\theta!_E))!_E = (t'\theta)!_E.$$

Justification of the `ind` Rule (II)

But since any $\theta : X \longrightarrow C_{\Sigma/E}$ satisfies $\theta = \theta!_E$, $\forall\, \theta \in [X \to T_\Omega]\ (t(\theta!_E))!_E = (t'(\theta!_E))!_E$ exactly means $\mathcal{C}_{\Sigma/E} \models t = t'$. q.e.d.

Notice that the above Lemma easily generalizes to the modulo A case, that is, to theories $(\Sigma, E \cup A)$ with E ground confluent, sort-decreasing, and terminating modulo A and Σ preregular modulo A. Our justification of the `ind` rule in what follows works just the same for the modulo A case.

Justification of the `ind` Rule (III)

Notice that the argument of the above lemma does not depend on our formula being actually an equation: by reasoning inductively on the structure of formulas we can show that the lemma applies to any universally-quantified first-order formula of the form $(\forall x : s)\ P(x)$ (P itself can have other quantifiers).

Therefore, we have reduced the problem of proving an inductive property, $(\forall x : s)\ P(x)$, to that of proving that for all $t \in T_{\Omega,s}$ the instantiated property $P(x \mapsto t)$ holds.

Here is where structural induction steps in as a method, namely, by analyzing more closely what it means to prove something for all $t \in T_{\Omega,s}$.

Justification of the `ind` Rule (IV)

Theorem. (*Soundness of Structural Induction*). For (Σ, E) ground confluent, sort-decreasing, and terminating with subsignature of constructors Ω, if we have

$$\mathcal{T}_{\Sigma/E} \models \bigwedge_{i} P(x \mapsto a_i) \wedge \bigwedge_{l} \bigwedge_{1 \leq j \leq k_{f_l}} (\forall x_{i_j}) P(x \mapsto x_{i_j}) \Rightarrow (\forall x_1, \ldots, x_{n_{f_l}}) P(x \mapsto f_l(x_1, \ldots, x_{n_{f_l}}))$$

then we also have

$$\mathcal{T}_{\Sigma/E} \models (\forall x : s)\ P(x).$$

Proof. Suppose not. This exactly means that the hypothesis holds and there exists a ground constructor term $t \in T_{\Omega,s}$ such that $\mathcal{T}_{\Sigma/E} \not\models P(x \mapsto t)$. Choose such $t \in T_{\Omega,s}$ of smallest depth possible. That is any other $t' \in T_{\Omega,s}$ such that $\mathcal{T}_{\Sigma/E} \not\models P(x \mapsto t')$ must have tree depth greater or equal to that of t.

Justification of the `ind` Rule (V)

Suc a term t cannot be a constant a_i of sort less or equal to s, since we have $\mathcal{T}_{\Sigma/E} \models \bigwedge_i P(x \mapsto a_i)$. Therefore, t must be of the form $t = f_q(t_1, \ldots, t_{n_{f_q}})$. But by the minimal depth assumption on t, we must have $\mathcal{T}_{\Sigma/E} \models P(x \mapsto t_{i_j})$, $1 \leq j \leq k_{f_q}$. Which by the theorem's hypothesis implies $\mathcal{T}_{\Sigma/E} \models P(x \mapsto f_q(t_1, \ldots, t_{n_{f_q}}))$. That is, $\mathcal{T}_{\Sigma/E} \models P(x \mapsto t)$, contradicting the assumption $\mathcal{T}_{\Sigma/E} \not\models P(x \mapsto t)$. q.e.d.

Need to Check Sufficient Completeness

All this is fine, but there is a pending issue. How do we know that the declared subsignature of constructors is correct? We need to check that it is sufficiently complete, for example using the SCC tool.

Also, as discussed in the justification of the `split` rule, the user may have overlooked giving enough equations for the defined functions, and then it becomes impossible to simplify every ground term to a constructor term.

Finally, a quick glance at our proof of the lemma involved in justifying the soundness of the `ind` rule shows that the reduction of proving $P(x)$ to proving $P(x \mapsto t)$ for each $t \in T_{\Omega,s}$ works just the same under weaker assumptions than (Σ, E) ground confluent, sort-decreasing, and terminating with subsignature of constructors Ω.

Exercises

Ex.13.1 Generalize the proof justifying the soundness of the `split` rule to a considerably weaker condition than protecting. Specifically, show that the `split` rule is sound to prove inductive properties about a module $\texttt{fmod}(\Sigma, E)\texttt{endfm}$ including `BOOL` as a submodule if and only if the homomorphism

$$\_\mathcal{T}_{\Sigma/E}|_{\texttt{BOOL}} : \mathcal{T}_{\texttt{BOOL}} \longrightarrow \mathcal{T}_{\Sigma/E}|_{\texttt{BOOL}}$$

is surjective. Explain why, since `true` and `false` are the only constructors of sort `Bool`, the above surjectivity property, essential to be sure that an application of the `split` rule is correct, can be automatically checked using the Maude SCC tool under quite general assumptions on (Σ, E).

Exercises (II)

Ex.13.2 Generalize the proof justifying the soundness of the `ind` rule to a considerably weaker condition. Specifically, show that the `ind` rule is sound to prove inductive properties about a module `fmod`(Σ, E)`endfm` if and only if the structural induction scheme uses a subsignature Ω on the same sorts S such that the unique homomorphism

$$\_{\mathcal{T}_{\Sigma/E}|_{\Omega}} : \mathcal{T}_{\Omega} \longrightarrow \mathcal{T}_{\Sigma,E}|_{\Omega}$$

is surjective.