Formal Methods: Lecture 11

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 model $T_{\Sigma,E}$ 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 |- {N:Natural ; M:Natural}
       (((N:Natural =< M:Natural) or (M:Natural =< N:Natural)) = true) .)

==================================
linear$0
==================================


|-{N:Natural ; M:Natural}(N:Natural =< M:Natural or M:Natural =< N:Natural =
    true)

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

```
Maude> (cns linear$0 .)

==================================
linear$0
==================================


|- N*Natural =< M*Natural or M*Natural =< N*Natural = true
```

The 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,

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

Thanks to the completeness of equational reasoning, this is expressed in the ITP as the cns rule,

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

Justification of the cns Rule

We can justify the validity of the **Constants Lemma** not only for unconditional equations, but also for conditional ones as follows. Given a conditional equation

$$(\forall X)\ \varphi = (\forall X)\ t = t' \quad \Leftarrow \quad u_1 = v_1 \wedge \ldots \wedge u_n = v_n,$$

we have $E \models_\Sigma (\forall X)\ \varphi$ iff, by definition,

$$(\forall (A,a) \in \mathbf{Alg}_{\Sigma(X)})\ A \models_\Sigma E \quad \Rightarrow \quad (A,a) \models_{\Sigma(X)} (\forall \emptyset)\ \varphi,$$

which is equivalent, by **Ex**6.2, to

$$(\forall (A,a) \in \mathbf{Alg}_{\Sigma(X)})\ (A,a) \models_{\Sigma(X)} E \quad \Rightarrow \quad (A,a) \models_{\Sigma(X)} (\forall \emptyset)\ \varphi,$$

which, by definition of satisfaction, is precisely $E \models_{\Sigma(X)} (\forall \emptyset)\ \varphi$, as desired.

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 linear$0 on (N*Natural < M*Natural) . )

==================================
linear$1.0
==================================


|- N*Natural =< M*Natural or M*Natural =< N*Natural = true
==================================
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 linear$1.0 .)

==================================
linear$2.0
==================================

|- N*Natural =< M*Natural or M*Natural =< N*Natural = true

++++++++++++++++++++++++++++++++++

Maude> (auto linear$2.0 .)

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` Must 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, sound application of `split` 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 `fmod`(Σ, E)`endfm` imports the submodule `fmod`(Σ', E')`endfm` in `protecting` mode, we require that the unique Σ'-homomorphism

$$eval^{E'}_{T_{\Sigma,E}|_{\Sigma'}} : T_{\Sigma'/E'} \longrightarrow T_{\Sigma,E}|_{\Sigma'}$$

is an isomorphism.

Justification of the `split` Rule

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

In `BOOL` we have,

$$T_{\texttt{BOOL}} \models \texttt{true} \neq \texttt{false}$$

and we also have,

$$T_{\texttt{BOOL}} \models (\forall x : \texttt{Bool})\ x = \texttt{true} \vee x = \texttt{false}$$

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

Justification of the `split` Rule (II)

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

$$(T_{\Sigma/E}, a) \not\models (p \neq \mathtt{true} \wedge p \neq \mathtt{false}).$$

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

$$(T_{\Sigma/E}, a) \models (p \neq \mathtt{true} \wedge p \neq \mathtt{false}) \quad \Leftrightarrow \quad (T_{\Sigma/E}, a) \models \bot.$$

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 $(\forall X)\, t = t'$, we have,

Justification of the `split` Rule (III)

$$T_{\Sigma/E} \models (\forall X)\, t = t' \;\; \Leftrightarrow \;\; T_{\Sigma/E} \models (\forall X)\, ((t = t') \vee \bot)$$

or, equivalently,

$$T_{\Sigma/E} \models (\forall X)\, t = t' \;\; \Leftrightarrow \;\; T_{\Sigma/E} \models (\forall X)\, ((t = t') \vee (p \neq \texttt{true} \wedge p \neq \texttt{false}))$$

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

$$T_{\Sigma/E} \models (\forall X)\, t = t' \;\; \Leftrightarrow \;\; T_{\Sigma/E} \models (\forall X)\, (p = \texttt{true} \Rightarrow t = t') \wedge (p = \texttt{false} \Rightarrow t =$$

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 in the next lecture, 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 |- {T:Tree}((rev(rev(T:Tree))) = (T:Tree)) .)

==================================
rev$0
==================================


|-{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 the variable T:Tree.

```
Maude> (ind rev$0 on T:Tree .)

==================================
rev$1.0
==================================


|-{V2#0:Tree ; V2#1:Tree}(rev(rev(V2#1:Tree))= V2#1:Tree & rev(rev(V2#0:Tree))=
    V2#0:Tree ==> rev(rev(V2#0:Tree # V2#1:Tree))= V2#0:Tree # V2#1:Tree)
==================================
rev$2.0
==================================


|-{V2#0:Qid}(rev(rev(V2#0:Qid))= V2#0:Qid)
```

Induction on Other Data Structures: Tree Induction (V)

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

```
Maude> (auto rev$1.0 .)

==================================
rev$2.0
==================================


|-{V2#0:Qid}(rev(rev(V2#0:Qid))= V2#0:Qid)

++++++++++++++++++++++++++++++++++

Maude> (auto rev$2.0 .)

q.e.d
```