Program Verification: Lecture 15

José Meseguer

Computer Science Department
University of Illinois at Urbana-Champaign

Proving Inductive Theorems with the NuITP

The NuITP is a next-generation inductive theorem proper for Maude replacing the earlier Maude Inductive Theorem Prover (ITP). The NuITP uses advanced symbolic techniques to automate large parts of inductive proofs, thus saving proof time and effort.

In the NuITP, standard induction on the natural numbers is generalized to induction on constructors, using the so-called generator set induction (GSI) inference rule.

To better understand generator set induction we can see how, in the case of the natural numbers, it can directly express standard natural number induction.

Let us see how associativity of addition is proved, first by standard induction, and then by the NuITP using generator set induction.

Standard Proof of Associativity of Addition

We want to prove that the addition operation in the module:

```
fmod PEANO+R is
    sort Nat .
    op 0 : -> Nat [ctor] .
    op s : Nat -> Nat [ctor] .
    op _+_ : Nat Nat -> Nat .
    vars N M L : Nat .
    eq N + 0 = N .
    eq N + s(M) = s(N + M) .
endfm
```

where `PEANO+R` suggests that we recurse on the right (`R`) argument when defining $+$, satisfies the associativity property,

$$(\forall \texttt{N}, \texttt{M}, \texttt{L})\ \texttt{N + (M + L)} = \texttt{(N + M) + L}.$$

Standard Proof of Associativity of Addition (II)

We can prove this property by induction on `L`. That is, we prove it for `L` $=$ `0` (base case) and then assuming that it holds for `L`, we prove it for `s(L)` (induction step).

BaseCase: We need to show,

$$(\forall \texttt{N},\texttt{M})\ \texttt{N + (M + 0)} = \texttt{(N + M) + 0}.$$

We can do this trivially, by simplification with the equation

```
eq N + 0 = N .
```

Standard Proof of Associativity of Addition (III)

InductionStep: We think of $\overline{\texttt{L}}$ as a generic constant (typically written n in textbooks) and assume that the associativity equation (induction hypothesis (IH))

$$(\forall \texttt{N}, \texttt{M})\ \ \texttt{N + (M + }\overline{\texttt{L}}\texttt{)} = \texttt{(N + M) + }\overline{\texttt{L}}$$

holds for that constant. Then, we try to prove the equation,

$$(\forall \texttt{N}, \texttt{M})\ \ \texttt{N} + (\texttt{M} + s(\overline{\texttt{L}})) = \texttt{(N + M)} + s(\overline{\texttt{L}}).$$

using the induction hypothesis. Again, we can do this by simplification with the equations E in `NAT`, and the induction hypothesis IH equation, since we have,

Standard Proof of Associativity of Addition (IV)

$$\mathtt{N} + (\mathtt{M} + s(\overline{\mathtt{L}})) \longrightarrow_E \mathtt{N} + s(\mathtt{M} + \overline{\mathtt{L}})$$

$$\longrightarrow_E s(\mathtt{N} + (\mathtt{M} + \overline{\mathtt{L}})) \longrightarrow_{IH} \mathtt{s((N + M) +}\overline{\mathtt{L}})).$$

and

$$\mathtt{(N + M) + s(L)} \longrightarrow_E \mathtt{s((N + M) + L).}$$

q.e.d

Machine-Assisted Inductive Proofs with Maude's NuITP

Maude's NuITP is an inductive theorem prover supporting proofs by induction in Maude functional modules. The NuITP is a research collaboration involving Francisco Durán at the University of Málaga, Santiago Escobar and Julia Sapiña at the Technical University of Valencia, and José Meseguer at UIUC. It is a Maude program used as follows:

- one first loads in Maude the functional module or modules one wants to reason about
- one then loads the file `NuITP.maude` into Maude.
- one sets one of the modules previously loaded in Maude as the current module and sets a multiclause as the goal to be proved.
- one then gives commands, corresponding to inductive proof steps, or formula simplification steps, to prove the chosen goal.

Proof of + Associativity with Maude's NuITP (I)

To prove the associativity of addition, we first load into Maude `PEANO+R` annotated with an RPO termination order, just as for the MTA. To prevent Maude from also loading `BOOL` we first type:

```
set include BOOL off .

fmod PEANO+R is
    sort Nat .
    op 0 : -> Nat [ctor metadata "0"] .
    op s : Nat -> Nat [ctor metadata "4"] .
    op _+_ : Nat Nat -> Nat [metadata "8"] .
    vars N M L : Nat .
    eq N + 0 = N .
    eq N + s(M) = s(N + M) .
endfm
```

Then we load `NuITP.maude` into Maude and set `PEANO+R` as current module and associativity of + as the goal to be proved as follows:

Proof of + Associativity with Maude's NuITP (II)

```
                   NuITP
          Inductive Theorem Prover
       for Maude Equational Theories
                 alpha 12a


NuITP> set module PEANO+R .

  Module PEANO+R is now active.

NuITP> set goal ((N:Nat + M:Nat) + L:Nat = N:Nat + (M:Nat + L:Nat)) .

  Initial goal set.

  Goal Id: 0
  Skolem Ops:
    None
  Executable Hypotheses:
```

```
    None
  Non-Executable Hypotheses:
    None
  Goal:
    (N:Nat +(M:Nat + L:Nat)) =((N:Nat + M:Nat) + L:Nat)
```

The user can now give commands to the NuITP to prove this goal. The command that exactly corresponds to standard induction on the natural numbers is:

```
apply gsi to 0 on L:Nat with 0 ;; s(K:Nat) .
```

where the generator set used for sort `Nat` is: $\{0, s(K)\}$, corresponding exactly to the base case and induction step of standard induction. Let us explore this concept in more detail.

Generator Sets

For `fmod` $(\Sigma, E \cup B)$ `endfm` an admissible equational program sufficiently complete w.r.t. constructors Ω, a generator set for sort s in Σ, is a finite set of constructor terms of sort s,

$$\{u_1, \ldots, u_n\} \subseteq T_\Omega(X)_s$$

such that any ground constructor term of sort s is a ground instance modulo B of some u_i, i.e., $\forall w \in T_{\Omega_s} \;\; \exists i, 1 \leq i \leq n$, $\exists \gamma \in [vars(u_i) \rightarrow T_\Omega]$, s.t. $w =_B u_i\gamma$.

$\{0, s(K)\}$ is a generator set of sort `Nat`; and $\{0, s(0), s(s(K))\}$ is also a generator set for `Nat`: many choices are possible.

For $\_;\_$ an associative operator of sort `LIst` with `Nat < List`, $\{nil, n, (L; L')\}$, $\{nil, n, (m; L)\}$ and $\{nil, n, (L; m)\}$ are all generator sets of sort `LIst` (with variables $n, m :$ `Nat`, $L, L' :$ `LIst`).

Checking Correctness of Generator Sets

Correctness of a generator set $\{u_1, \ldots, u_n\}$ for a sort s can be reduced to: (i) checking $\{u_1, \ldots, u_n\} \subseteq T_\Omega(X)_s$ and (ii) a sufficient completeness check for a module. For $\{nil, n, (m; L)\}$ the module:

```
fmod GEN-SET-SORT-PREDICATE-FOR-List is protecting TRUTH-VALUE .
sorts Nat List .            subsorts Nat < List .
op 0 : -> Nat [ctor]        op nil : -> List [ctor] .
op s : Nat -> Nat [ctor]    op _;_ : List List -> List [ctor assoc] .
op List : List -> Bool .
eq List(nil) = true .       eq List(n:Nat) = true .
eq List(m:Nat ; L:List) = true .
endfm
```

In the current alpha version of NuITP it is the user's responsibility to check the sufficient completeness of the module defining the sort predicate associated to a generator set using Maude's SCC.

Warning: the variables of a generator set should be fresh, not appearing in any goal. And the u_i should be linear terms.

Proof of + Associativity with Maude's NuITP (III)

```
NuITP> apply gsi to 0 on L:Nat with 0 ;; s(K:Nat) .

  Generator Set Induction (GSI) applied to goal 0.

  Goal Id: 0.1
  Skolem Ops:
    None
  Executable Hypotheses:
    None
  Non-Executable Hypotheses:
    None
  Goal:
    (N:Nat +(M:Nat + 0)) =((N:Nat + M:Nat) + 0)

  Goal Id: 0.2
  Skolem Ops:
    K.Nat
  Executable Hypotheses:
```

```
  ((N:Nat + M:Nat) + K) =>(N:Nat +(M:Nat + K))
Non-Executable Hypotheses:
  None
Goal:
  (N:Nat +(M:Nat + s(K))) =((N:Nat + M:Nat) + s(K))
```

These goals are exactly those generated by standard induction. Note that the role of the generic constant $\overline{\texttt{L}}$ is here played by the Skolem constant `K`.

As in standard induction, all we have left to do is to simplify these goals using: (i) the module's equations; and (ii) the induction hypothesis. In the NuITP this is done with the equality predicate simplification (`eps`) command as follows:

Proof of + Associativity with Maude's NuITP (IV)

```
NuITP> apply eps to 0.1 .

  Equality Predicate Simplification (EPS) applied to goal 0.1.

  Goal 0.1.1 has been proved.

  Unproved goals:

  Goal Id: 0.2
  Skolem Ops:
    K.Nat
  Executable Hypotheses:
    ((N:Nat + M:Nat) + K) =>(N:Nat +(M:Nat + K))
  Non-Executable Hypotheses:
    None
  Goal:
    (N:Nat +(M:Nat + s(K))) =((N:Nat + M:Nat) + s(K))
```

```
NuITP> apply eps to 0.2 .

  Equality Predicate Simplification (EPS) applied to goal 0.2.

  Goal 0.2.1 has been proved.

  qed
```

The `qed` acronym indicates that there are no pending goals and the inductive proof of associativity of $+$ has been finished, exactly as with standard induction.

If we had instead used the generator set $\{0, s(0), s(s(K))\}$ a somewhat different proof with two "base cases" and one "induction step" would have been created. The user has the freedom to choose a generator set that best matches the recursive equations in the module. In this example the generator set $\{0, s(K)\}$ was a good match; but in other examples other choices may be preferable.

The **gsi!** Command

For many NuITP commands like `gsi` that apply an inductive inference rule, the best strategy before applying another command is to simplify the subgoals just generated using the `eps` command.

This situation is so common, that the NuITP combines both commands into the `gsi!` command, that applies `eps` to each of the goals generated by `gsi`. This can greatly shorten proofs. Let us see the effect for proving associativity of $+$:

```
NuITP> set module PEANO+R .

  Module PEANO+R is now active.

NuITP> set goal ((N:Nat + M:Nat) + L:Nat = N:Nat + (M:Nat + L:Nat)) .

  Initial goal set.
```

```
  Goal Id: 0
  Skolem Ops:
    None
  Executable Hypotheses:
    None
  Non-Executable Hypotheses:
    None
  Goal:
    (N:Nat +(M:Nat + L:Nat)) =((N:Nat + M:Nat) + L:Nat)

NuITP> apply gsi! to 0 on L:Nat with 0 ;; s(K:Nat) .

  Generator Set Induction with Equality Predicate Simplification (GSI!) applied
    to goal 0.

  Goals 0.1 and 0.2 have been proved.

  qed

NuITP>
```

Proving Program Equivalences in NuITP

Recall from the Program Equivalence Theorem in Lecture 14 that `fmod` $(\Sigma, E \cup B)$ `endfm` $\equiv_{sem}$ `fmod` $(\Sigma, E' \cup B')$ `endfm` iff $(\Sigma, E \cup B) \equiv_{ind} (\Sigma, E' \cup B')$ iff (by definition)

$$\mathbb{T}_{\Sigma/E \cup B} \models E' \cup B' \quad and \quad \mathbb{T}_{\Sigma/E' \cup B'} \models E \cup B.$$

In particular, proving program equivalences can be useful for program optimization purposes.

Let us prove that our equational program `PEANO+R` is semantically equivalent to the following program `PEANO+R-FAST`, which runs, roughly, twice as fast.

Proving Program Equivalences in NuITP (II)

```
fmod PEANO+R-FAST is
    sort Nat .
    op 0 : -> Nat [ctor metadata "0"] .
    op s : Nat -> Nat [ctor metadata "4"] .
    op _+_ : Nat Nat -> Nat [metadata "8"] .
    vars N M : Nat .
    eq N + 0 = N .
    eq N + s(0) = s(N) .
    eq N + s(s(M)) = s(s(N + M)) .
endfm
```

Note that a good generator set for this program, matching its recursive equations, is: $\{0, s(0), s(s(K))\}$. Proofs for this module using this generator set will tend to be shorter than proofs using the "vanilla flavored" generator set $\{0, s(K)\}$.

Let us now prove that `PEANO+R` and `PEANO+R-FAST` are equivalent.

Proving Program Equivalences in NuITP (III)

```
NuITP> set goal ((N:Nat + 0 = N:Nat) /\ (N:Nat + s(0) = s(N:Nat)) /\
       (N:Nat + s(s(M:Nat)) = s(s(N:Nat + M:Nat)))) .

  Initial goal set.

  Goal Id: 0
  Skolem Ops:
    None
  Executable Hypotheses:
    None
  Non-Executable Hypotheses:
    None
  Goal:
    (N:Nat =(N:Nat + 0)) /\(s(N:Nat) =(N:Nat + s(0))) /\
    s(s(N:Nat + M:Nat)) =(N:Nat + s(s(M:Nat)))

NuITP> apply eps to 0 .
```

```
  Equality Predicate Simplification (EPS) applied to goal 0.

  Goal 0.1 has been proved.

  qed

NuITP> set module PEANO+R-FAST .

  Module PEANO+R-FAST is now active.

NuITP> set goal ((X:Nat + 0 = X:Nat) /\ (X:Nat + s(Y:Nat) = s(X:Nat + Y:Nat))) .

  Initial goal set.

  Goal Id: 0
  Skolem Ops:
    None
  Executable Hypotheses:
    None
  Non-Executable Hypotheses:
    None
```

```
  Goal:
    (X:Nat =(X:Nat + 0)) /\ s(X:Nat + Y:Nat) =(X:Nat + s(Y:Nat))

NuITP> apply gsi! to 0 on Y:Nat with 0 ;; s(0) ;; s(s(K:Nat)) .

  Generator Set Induction with Equality Predicate Simplification (GSI!) applied
    to goal 0.

  Goals 0.1, 0.2 and 0.3 have been proved.

  qed

NuITP>
```