Program Verification: Lecture 16

José Meseguer

Computer Science Department
University of Illinois at Urbana-Champaign

Scaling Up Inductive Proofs with NuITP

Theorem proving is the strongest form of program verification. However, it is labor intensive and requires significant user expertise. Therefore, methods to scale up theorem proving verification are very important. The NuITP supports several such methods:

1. **Bundling**: Prove many properties with a single multiclause.
2. **Internalize and Conquer**: Make already proved properties available in subsequent proofs by adding them to the module (internalization); and exploit program equivalences.
3. **Modularize and Conquer**: Inherit already proved properties of sub/super modules in super/sub modules.
4. **Proof Strategies**: Automate large parts of a proof by proof strategies that recursively apply some simplification and/or induction proof rules.

Scaling Up Inductive Proofs with NuITP

In this lecture I will illustrate the effectiveness of scaling up methods (1)–(3) by means of two case studies.

In a first case study I will show how bundling, proving program equivalences, and internalization allow a very short proof of associativity-commutativity of natural number addition.

In a second case study I will show how proving many properties of various functions on lists, trees, and mapping trees to lists can be done with very short proofs using techniques (1)–(3).

Case Study 1

Recall the standard definition of natural number addition by recursion on the right argument in `PEANO+R` (Lecture 15). Let us prove its semantic equivalence with `PEANO+L`:

```
set include BOOL off .

fmod PEANO+L 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 0 + N = N .
    eq s(N) + M = s(N + M) .
endfm
```

Case Study 1 (II)

```
NuITP> set module PEANO+R .

  Module PEANO+R is now active.

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

  Initial goal set.

  Goal Id: 0
  Skolem Ops:
    None
  Executable Hypotheses:
    None
  Non-Executable Hypotheses:
    None
  Goal:
    (Y:Nat =(0 + Y:Nat)) /\ s(X:Nat + Y:Nat) =(s(X:Nat) + Y:Nat)
```

```
NuITP> apply gsi! to 0 on Y: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> set module PEANO+L .

  Module PEANO+L is now active.

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

  Initial goal set.

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

```
    None
  Non-Executable Hypotheses:
    None
  Goal:
    (Y:Nat =(Y:Nat + 0)) /\ s(Y:Nat + X:Nat) =(Y:Nat + s(X:Nat))

NuITP> apply gsi! to 0 on Y: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>
```

Case Study 1 (III)

By using the Lemma Internalization Theorem 2, the following program `PEANO+LR` is semantically equivalent to both `PEANO+L` and `PEANO+R`:

```
set include BOOL off .

fmod PEANO+LR is
     protecting PEANO+R .
    vars N M : Nat .
    *** the already-proved equations of PEANO+L are internalized
    eq 0 + N = N .
    eq s(N) + M = s(N + M) .
endfm
```

Therefore, we can use `PEANO+LR` to prove that $+$ in `PEANO+R` (and of course in `PEANO+L`) is AC:

Case Study 1 (IV)

```
NuITP> set module PEANO+LR .

  Module PEANO+LR is now active.

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

  Initial goal set.

  Goal Id: 0
  Skolem Ops:
    None
  Executable Hypotheses:
    None
  Non-Executable Hypotheses:
    None
  Goal:
    ((X:Nat + Y:Nat) =(Y:Nat + X:Nat)) /\(X:Nat +(Y:Nat + Z:Nat)) =((X:Nat +
```

```
    Y:Nat) + Z:Nat)

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

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

  Goal 0.1 has been proved.

  Goal Id: 0.2
  Skolem Ops:
    K.Nat
  Executable Hypotheses:
    ((X:Nat + K) + Z:Nat) =>(X:Nat +(K + Z:Nat))
  Non-Executable Hypotheses:
    (K + X:Nat) =(X:Nat + K)
  Goal:
    (K + X:Nat) =(X:Nat + K)

NuITP>
```

Since goal 0.2 is identical to its non-executable hypothesis, we can use clause subsumption (`cs`) (which applies in particular when a goal is a substitution instance of a hypothesis) to finish the proof:

```
NuITP> apply cs to 0.2 .

  Clause Subsumption (CS) applied to goal 0.2.

  Goal 0.2.1 has been proved.

  qed

NuITP>
```

That is, we have both proved `PEANO+R` $\equiv_{sem}$ `PEANO+L` and $+$ AC with just three applications of `gsi!` and one application of `cs`.

Furthermore, the Lemma Internalization Theorem 3 gives us the additional program equivalence: `PEANO+R` $\equiv_{sem}$ `PEANO+AC` for the program:

```
set include BOOL off .

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

`PEANO+AC` will be much more effective than either `PEANO+R`, `PEANO+L`, or `PEANO+LR` in proving further properties (not just of $+$, but of other functions using $+$), that require knowledge that $+$ is AC.

Modularize and Conquer

The key idea of the modularize and conquer method is to inherit already proved properties of a sub/super theory in a super/sub theory. Recall that a theory inclusion

$$(\Sigma_0, E_0 \cup B_0) \subseteq (\Sigma, E \cup B)$$

is (by definition) protecting iff $\mathbb{T}_{\Sigma/E \cup B}|_{\Sigma_0} \cong \mathbb{T}_{\Sigma_0/E_0 \cup B_0}$.

We will use two theorems that are proved in an Appendix to this lecture, soon to appear:

Theorem (Up Theorem). For a theory inclusion of admissible theories $(\Sigma_0, E_0 \cup B_0) \subseteq (\Sigma, E \cup B)$ sufficiently complete w.r.t. respective constructors Ω_0 and Ω, with respective sort sets S_0 and S s.t. $s \in S, s_0 \in S_0 \wedge s < s_0 \Rightarrow s \in S_0$, s.t. $\Omega|_{S_0} = \Omega_0$, where $\Omega|_{S_0} =_{def} \{(c : w \to s) \in \Omega \mid (w, s) \in S_0^* \times S_0\}$, and s.t.

$(u \in T_{\Omega_0} \wedge u \to_{\vec{E}/B} v) \Rightarrow v \in T_{\Omega_0}$, for φ any unconditional multiclause such that $\varphi \in \mathit{IndThm}_{FOL}(\Sigma_0, E_0 \cup B_0)$ we have $\varphi \in \mathit{IndThm}_{FOL}(\Sigma, E \cup B)$.

The requirement that φ is unconditional is essential. For example, for $(\Sigma_0, E_0 \cup B_0)$ the theory of `PEANO+R`, and $(\Sigma, E \cup B)$ the theory that adds the equation $s(s(s(0))) = 0$ to get the naturals modulo 3. Obviously, $s(s(s(0))) \neq 0 \notin \mathit{IndThm}_{FOL}(\Sigma, E \cup B)$; but $s(s(s(0))) \neq 0 \in \mathit{IndThm}_{FOL}(\Sigma_0, E_0 \cup B_0)$, expressible as the clause $s(s(s(0))) = 0 \to \mathit{false}$. Indeed:

```
NuITP> set module PEANO+R .

  Module PEANO+R is now active.

NuITP> set goal s(s(s(X:Nat))) = 0 -> false .

  Initial goal set.
```

```
  Goal Id: 0
  Skolem Ops:
    None
  Executable Hypotheses:
    None
  Non-Executable Hypotheses:
    None
  Goal:
    0 = s(s(s(X:Nat))) -> false

NuITP> apply eps to 0 .

  Equality Predicate Simplification (EPS) applied to goal 0.

  Goal 0.1 has been proved.

  qed

NuITP>
```

Modularize and Conquer (II)

Theorem (Up and Down Theorem). For a theory inclusion of admissible theories $(\Sigma_0, E_0 \cup B_0) \subseteq (\Sigma, E \cup B)$ such that $(\Sigma, E \cup B)$ protects $(\Sigma_0, E_0 \cup B_0)$ and φ any Σ_0-multiclause we have the equivalence:

$$\varphi \in \mathit{IndThm}_{FOL}(\Sigma_0, E_0 \cup B_0) \quad \Leftrightarrow \quad \varphi \in \mathit{IndThm}_{FOL}(\Sigma, E \cup B).$$

By the Up Theorem, if an unconditional multiclause φ is an inductive theorem of submodule $(\Sigma_0, E_0 \cup B_0)$ it is also an inductive theorem of supermodule $(\Sigma, E \cup B)$.

By the Up and Down Theorem, if the theory inclusion $(\Sigma_0, E_0 \cup B_0) \subseteq (\Sigma, E \cup B)$ is protecting, it doen't matter where we prove that a Σ_0-multiclause φ is an inductive theorem: we could do it either in the submodule $(\Sigma_0, E_0 \cup B_0)$ or in the supermodule $(\Sigma, E \cup B)$, since it holds for both.

Case Study 2

Consider the following three modules, defining functions on lists, trees, and between trees and lists:

```
fmod NAT-LIST+R is protecting PEANO+R .
    sorts NeList List .  subsorst Nat < NeList < List .
    op nil : -> List [ctor metadata "1"] .
    op _;_ : List List -> List [assoc metadata "5"] .
    op _;_ : NeList NeList -> NeList [ctor assoc metadata "5"] .
    op rev : List -> List [metadata "10"] .   *** list reverse
    op + : List -> Nat [metadata "12"] .    *** adds all numbers in list
    var N : Nat .  vars L : List .
    eq L ; nil = L .                  eq nil ; L = L .
    eq rev(nil) = nil .
    eq rev(N) = N .                   eq rev(N ; L) = rev(L) ; N .
    eq +(nil) = 0 .   eq +(N) = N .  eq +(N ; L) = N + +(L) .
endfm
```

```
fmod NAT-TREE+R is protecting PEANO+R .
    sort  Tree .  subsort Nat < Tree .
    op _^_ : Tree Tree -> Tree [ctor metadata "6"] .
    op rev : Tree -> Tree [metadata "10"] .  *** tree reverse
    op + : Tree -> Nat [metadata "12"] .     *** adds all numbers in tree
    vars N M : Nat .  vars T1 T2 : Tree .
    eq rev(N) = N .
    eq rev(T1 ^ T2) = rev(T2) ^ rev(T1) .
    eq +(N) = N .
    eq +(T1 ^ T2) = +(T1) + +(T2) .
endfm

fmod NAT-TREE-LIST+R is protecting NAT-TREE+R .
  protecting NAT-LIST+R .
  op rev : Nat -> Nat [metadata "10"] .   *** added for preregularity
  op + : Nat -> Nat [metadata "12"] .     *** added for preregularity
  op t2l : Tree -> List [metadata "9"] .  *** maps trees to lists
  var N : Nat .  vars T1 T2 : Tree .
  eq t2l(N) = N .
  eq t2l(T1 ^ T2) = t2l(T1) ; t2l(T2) .
endfm
```

Case Study 2 (II)

We would like to prove the following four inductive theorems about `NAT-LIST+R`:

1. $rev(rev(L\!:\!List)) = L\!:\!List$

2. $rev(L\!:\!List\ ;\ Q\!:\!List) = rev(Q\!:\!List)\ ;\ rev(L\!:\!List)$

3. $+(L\!:\!List\ ;\ Q\!:\!List) = +(L\!:\!List)\ +\ +(Q\!:\!List)$

4. $+(rev(L\!:\!List)) = +(L\!:\!List)$

Likewise, we would like to prove the following two inductive theorems about (respectively) `NAT-TREE+R` and `NAT-TREE-LIST+R`:

1. $rev(rev(T\!:\!Tree)) = T\!:\!Tree$

2. $+(rev(T\!:\!Tree)) = +(T\!:\!Tree)$.

1. $t2l(rev(T\!:\!Tree)) = rev(t2l(T\!:\!Tree))$

2. $+(t2l(T\!:\!Tree)) = +(T\!:\!Tree)$.

However, since, using the Strong Protection Theorem in the Appendix to this lecture it is easy to check that `NAT-TREE-LIST+R` protects `NAT-TREE+R`, using the Up and Down Theorem we can get a shorted proof by bundling these four equations together and proving them in `NAT-TREE-LIST+R`.

Case Study 2 (III)

Since both NAT-LIST+R and NAT-TREE+R have functions + that extend natural number additions to lists (resp. trees), this is a good example of a case where the remark at the end of Case Study 1 that PEANO+AC will be much more effective than PEANO+R in proving properties of other functions using + applies. Therefore, we would like to carry out those proofs using semantically equivalent programs that internalize the knowledge that natural number addition is AC, namely,

```
set include BOOL off .

fmod NAT-LIST+AC is protecting PEANO+AC .
    sorts  NeList List .  subsort Nat < NeList < List .
    op nil : -> List [ctor metadata "1"] .
    op _;_ : List List -> List [assoc metadata "5"] .
    op _;_ : NeList NeList -> NeList [ctor assoc metadata "5"] .
```

```
    op rev : List -> List [metadata "10"] .   *** list reverse
    op + : List -> Nat [metadata "12"] .      *** adds all numbers in list
    var N : Nat .  vars L : List .
    eq L ; nil = L .                  eq nil ; L = L .
    eq rev(nil) = nil .
    eq rev(N) = N .                   eq rev(N ; L) = rev(L) ; N .
    eq +(nil) = 0 .   eq +(N) = N .  eq +(N ; L) = N + +(L) .
endfm

set include BOOL off .

fmod NAT-TREE+AC is protecting PEANO+AC .
    sort  Tree .  subsort Nat < Tree .
    op _^_ : Tree Tree -> Tree [ctor metadata "6"] .
    op rev : Tree -> Tree [metadata "10"] .  *** tree reverse
    op + : Tree -> Nat [metadata "12"] .     *** adds all numbers in tree
    vars N M : Nat .  vars T1 T2 : Tree .
    eq rev(N) = N .              eq rev(T1 ^ T2) = rev(T2) ^ rev(T1) .
    eq +(N) = N .                eq +(T1 ^ T2) = +(T1) + +(T2) .
endfm
```

Case Study 2 (IV)

The fact that we indeed have program equivalences
`NAT-LIST+R` $\equiv_{sem}$ `NAT-LIST+AC` and
`NAT-TREE+R` $\equiv_{sem}$ `NAT-TREE+AC`, follows directly form the program equivalence `NAT+R` $\equiv_{sem}$ `NAT+AC` by the Up Theorem and the Lemma Internalization Theorem 3.

So, let us start by proving the four inductive theorems for `NAT-LIST+R` using `NAT-LIST+AC`:

```
NuITP> set goal (rev(rev(L:List)) = L:List) /\
(rev(L:List ; Q:List) = (rev(Q:List) ; rev(L:List))) /\
(+(L:List ; Q:List) = (+(L:List) + +(Q:List))) /\ (+(rev(L:List)) = +(L:List)) .

  Initial goal set.

  Goal Id: 0
  Skolem Ops:
```

```
    None
  Executable Hypotheses:
    None
  Non-Executable Hypotheses:
    None
  Goal:
    (L:List = rev(rev(L:List))) /\(+(L:List) = +(rev(L:List))) /\(+(L:List ;
    Q:List) = +(Q:List) + +(L:List)) /\ rev(L:List ; Q:List) = rev(Q:List) ;
    rev(L:List)

NuITP> apply gsi! to 0 on L:List with nil ;; X:Nat ;; (Y:Nat ; L3:NeList) .

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

  Goals 0.1 and 0.2 have been proved.

  Goal Id: 0.3
  Skolem Ops:
    L3.NeList
    Y.Nat
```

```
Executable Hypotheses:
  +(L3 ; Q:List) => +(L3) + +(Q:List)
  +(rev(L3)) => +(L3)
  rev(L3 ; Q:List) => rev(Q:List) ; rev(L3)
  rev(rev(L3)) => L3
Non-Executable Hypotheses:
  None
Goal:
  (+(rev(L3) ; Y) = Y + +(L3)) /\ rev(rev(L3) ; Y) = Y ; L3
```

The key observation at this point is that we could easily prove each of these two conjuncts using their executable hypotheses plus two of the original inductive theorems we wanted to prove, namely, $+(L\!:\!List\,;\,Q\!:\!List) = +(L\!:\!List)\ +\ +(Q\!:\!List)$ and $rev(L\!:\!List\,;\,Q\!:\!List) = rev(Q:List)\,;\,rev(L\!:\!List)$, used as lemmas by means of the `le!` command, and bundled as a single conjunctive formula:

```
NuITP> apply le! to 0.3 with (rev(L:List ; Q:List) = rev(Q:List) ; rev(L:List))
 /\ (+(L:List ; Q:List) = (+(L:List) + +(Q:List))) .

  Lemma Enrichment with Equality Predicate Simplification (LE!) applied to
    goal 0.3.

  Goal 0.3.2 has been proved.

  Goal Id: 0.3.1
  Skolem Ops:
    None
  Executable Hypotheses:
    None
  Non-Executable Hypotheses:
    None
  Goal:
    (+(L:List ; Q:List) = +(Q:List) + +(L:List)) /\ rev(L:List ; Q:List) =
    rev(Q:List) ; rev(L:List)

NuITP> apply gsi! to 0.3.1 on L:List with nil ;; X:Nat ;; (Y:Nat ; L3:NeList) .
```

```
  Generator Set Induction with Equality Predicate Simplification (GSI!)
    applied to goal 0.3.1.

  Goals 0.3.1.1, 0.3.1.2 and 0.3.1.3 have been proved.

  qed

NuITP>
```

This only leaves us proving the two inductive theorems for `NAT-TREE+AC` and the two for `NAT-TREE-LIST+AC`, which, by `NAT-TREE-LIST+AC` protecting `NAT-TREE+AC`, we can prove together as a conjunctive bundle of for theorems in `NAT-TREE-LIST+AC`. However, since several of these theorems involve properties of functions in `NAT-LIST+AC`, it would be silly to attempt such a proof without first internalizing, thanks to the Up Theorem and the Lemma Internalization Theorem 2, the four theorems we have already proved about `NAT-LIST+AC` as follows:

```
set include BOOL off .

fmod NAT-TREE-LIST+AC-ENRICHED is protecting NAT-TREE+AC .
  protecting NAT-LIST+AC .
  op rev : Nat -> Nat [metadata "10"] .   *** added for preregularity
  op + : Nat -> Nat [metadata "12"] .     *** added for preregularity
  op t2l : Tree -> List [metadata "9"] .  *** maps trees to lists
  var N : Nat .  vars T1 T2 : Tree .
  eq t2l(N) = N .                   eq t2l(T1 ^ T2) = t2l(T1) ; t2l(T2) .
  eq rev(rev(L:List)) = L:List .                             *** internalized
  eq +(L:List ; Q:List) = +(Q:List) + +(L:List) .            *** internalized
  eq rev(L:List ; Q:List) = rev(Q:List) ; rev(L:List) . *** internalized
  eq +(rev(L:List)) = +(L:List) .                            *** internalized
endfm
```

Case Study 2 (V)

We can in fact prove those four inductive theorems by a single application of `gsi!` as follows:

```
NuITP> set module NAT-TREE-LIST+AC-ENRICHED . .

  Module NAT-TREE-LIST+AC-ENRICHED is now active.

NuITP> set goal (rev(rev(T:Tree))= T:Tree) /\
(+(rev(T:Tree)) = +(T:Tree)) /\
(t2l(rev(T:Tree)) = rev(t2l(T:Tree))) /\ (+(t2l(T:Tree)) = +(T:Tree)) .

  Initial goal set.

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

```
  Non-Executable Hypotheses:
    None
  Goal:
    (T:Tree = rev(rev(T:Tree))) /\(+(T:Tree) = +(rev(T:Tree))) /\(+(T:Tree) =
    +(t2l(T:Tree))) /\ rev(t2l(T:Tree)) = t2l(rev(T:Tree))

NuITP> apply gsi! to 0 on T:Tree with X:Nat ;; (P:Tree ^ Q:Tree) .

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

  Goals 0.1 and 0.2 have been proved.

  qed

NuITP>
```

In summary, we have proved eight inductive theorems and established seven program equivalences with just: three applications of `gsi!` and one application of `le!`.

Soundness of NuITP's Inference System

The theoretical basis, inference rules, and proof of soundness for the inductive inference system $\vdash_{ind}$ on which the NuITP is based are described in detail in the paper:

> J. Meseguer and S. Skeirik, "Inductive Reasoning with Equality Predicates, Contextual Rewriting and Variant-Based Simplification" (submitted for publication).

which will soon be made available on the CS 476 web page. However, since this is still an unpublished paper, I ask of all CS 476 students to please use of this paper in a private manner, solely for purposes of following the CS 476 Fall 2022 course, since a finished and improved version of it has not yet appeared in print.