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 multiclaise.

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.
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).
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
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) \(\land\) (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>
By using the Lemma Internalization Theorem 2, the following program \texttt{PEANO+LR} is semantically equivalent to both \texttt{PEANO+L} and \texttt{PEANO+R}:

\begin{verbatim}
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
\end{verbatim}

Therefore, we can use \texttt{PEANO+LR} to prove that $+$ in \texttt{PEANO+R} (and of course in \texttt{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) /

Initial goal set.

Goal Id: 0
Skolem Ops:
  None
Executable Hypotheses:
  None
Non-Executable Hypotheses:
  None
Goal:
  ((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 \( \text{PEANO+R} \equiv_{sem} \text{PEANO+L} \) 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: \( \text{PEANO+R} \equiv_{sem} \text{PEANO+AC} \) 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 \(T_{\Sigma/E \cup B}|_{\Sigma_0} \cong T_{\Sigma_0/E_0 \cup B_0}\).

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 \(\Omega_0\) and \(\Omega\), with respective sort sets \(S_0\) and \(S\) s.t. \(s \in S, s_0 \in S_0 \land s < s_0 \Rightarrow s \in S_0\), s.t. \(\Omega|_{S_0} = \Omega_0\), where \(\Omega|_{S_0} =_{def} \{(c : w \rightarrow s) \in \Omega \mid (w, s) \in S_0^* \times S_0\}\), and s.t.
\((u \in T_{\Omega_0} \land u \rightarrow_{E/B} v) \Rightarrow v \in T_{\Omega_0}\), for \(\varphi\) any unconditional multiclause such that \(\varphi \in \text{IndThm}_{FOL}(\Sigma_0, E_0 \cup B_0)\) we have \(\varphi \in \text{IndThm}_{FOL}(\Sigma, E \cup B)\).

The requirement that \(\varphi\) is unconditional is essential. For example, for \((\Sigma_0, E_0 \cup B_0)\) the theory of \texttt{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, \(s(s(s(0))) \neq 0 \notin \text{IndThm}_{FOL}(\Sigma, E \cup B)\); but \(s(s(s(0))) \neq 0 \in \text{IndThm}_{FOL}(\Sigma_0, E_0 \cup B_0)\), expressible as the clause \(s(s(s(0))) = 0 \rightarrow \text{false}\). Indeed:

\texttt{NuITP> set module PEANO+R .}

\texttt{Module PEANO+R is now active.}

\texttt{NuITP> set goal s(s(s(X:Nat))) = 0 -> false .}

\texttt{Initial goal set.}
Goal Id: 0
Skolem Ops:
  None
Executable Hypotheses:
  None
Non-Executable Hypotheses:
  None
Goal:
  \[ 0 = s(s(s(X:\text{Nat}))) \rightarrow \text{false} \]

NuITP> apply eps to 0 .

Equality Predicate Simplification (EPS) applied to goal 0.

Goal 0.1 has been proved.

qed

NuITP>
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 $\varphi$ any $\Sigma_0$-multiclause we have the equivalence:

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

By the Up Theorem, if an unconditional multiclause $\varphi$ 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 doesn’t matter where we prove that a $\Sigma_0$-multiclause $\varphi$ 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:

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

18
We would like to prove the following four inductive theorems about \texttt{NAT-LIST+R}:

1. \( \text{rev}(\text{rev}(\text{L}:\text{List})) = \text{L}:\text{List} \)

2. \( \text{rev}(\text{L}:\text{List} ; \text{Q}:\text{List}) = \text{rev}(\text{Q}:\text{List}) ; \text{rev}(\text{L}:\text{List}) \)

3. \( +(\text{L}:\text{List} ; \text{Q}:\text{List}) = + (\text{L}:\text{List}) + + (\text{Q}:\text{List}) \)

4. \( +(\text{rev}(\text{L}:\text{List})) = +(\text{L}:\text{List}) \)

Likewise, we would like to prove the following two inductive theorems about (respectively) \texttt{NAT-TREE+R} and \texttt{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 \textsc{NAT-TREE-LIST+r} protects \textsc{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 \textsc{NAT-TREE-LIST+r}.
Since both \texttt{NAT-LIST+R} and \texttt{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 \texttt{PEANO+AC} will be much more effective than \texttt{PEANO+R} in proving properties of other functions using + applies. Therefore, we would like to carry out those proofs using \textit{semantically equivalent} programs that \texttt{internalize} the knowledge that natural number addition is AC, namely,

\begin{verbatim}
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"] .
\end{verbatim}
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 .
opt _^_ : Tree Tree -> Tree [ctor metadata "6"] .
opt 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
The fact that we indeed have program equivalences
\[ \text{NAT-LIST} + R \equiv_{sem} \text{NAT-LIST} + AC \] and
\[ \text{NAT-TREE} + R \equiv_{sem} \text{NAT-TREE} + AC, \]
follows directly from the program equivalence \( \text{NAT} + R \equiv_{sem} \text{NAT} + AC \) by the Up Theorem and the Lemma Internalization Theorem 3.

So, let us start by proving the four inductive theorems for \( \text{NAT-LIST} + R \) using \( \text{NAT-LIST} + AC \):

```
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:
\[(L:\text{List} = \text{rev}(\text{rev}(L:\text{List}))) \land (+\text{(L:\text{List})} = +\text{(rev}(L:\text{List}))) \land (+\text{(L:\text{List} \cup Q:\text{List})} = +\text{(Q:\text{List})} + +\text{(L:\text{List})}) \land \text{rev}(L:\text{List} \cup Q:\text{List}) = \text{rev}(Q:\text{List}); \text{rev}(L:\text{List})\]

NuITP> apply gsi! to 0 on L:\text{List} with nil ;; X:Nat ;; (Y:Nat ; L3:\text{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:\text{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)) \(\land\) 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:
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:

\((+\text{List} ; \text{Q:List}) = (+\text{List}) + +(\text{Q:List})) \land \text{rev(\text{L:List} ; \text{Q:List}) = rev(\text{Q:List}) ; rev(\text{L:List})}\)

NuITP> apply gsi! to 0.3.1 on \text{L:List} with nil ;; \text{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
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:
  \((T:Tree = \text{rev}(\text{rev}(T:Tree))) \lor (+ (T:Tree) = + (\text{rev}(T:Tree))) \lor (+ (T:Tree) = + (t2l(T:Tree))) \lor \text{rev}(t2l(T:Tree)) = t2l(\text{rev}(T:Tree))\)

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!.
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:


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.