

Program Verification: Lecture 16

José Meseguer

University of Illinois at Urbana-Champaign

Guessing Lemmas from Failed Proof Attempts

In inductive theorem proving it often happens that our **first attempt** to prove a property is unsuccessful.

Guessing Lemmas from Failed Proof Attempts

In inductive theorem proving it often happens that our **first attempt** to prove a property is unsuccessful. However, we can **learn from a failed attempt** to **guess a lemma** that will solve the unproved goal.

Guessing Lemmas from Failed Proof Attempts

In inductive theorem proving it often happens that our **first attempt** to prove a property is unsuccessful. However, we can **learn from a failed attempt** to **guess a lemma** that will solve the unproved goal.

In the NuITP the process of **guessing** and **applying** a lemma to a goal is supported by the **lemma enrichment** (le) inference rule.

Guessing Lemmas from Failed Proof Attempts

In inductive theorem proving it often happens that our **first attempt** to prove a property is unsuccessful. However, we can **learn from a failed attempt** to **guess a lemma** that will solve the unproved goal.

In the NuTP the process of **guessing** and **applying** a lemma to a goal is supported by the **lemma enrichment** (le) inference rule.

Of course, although the lemma thus guessed will typically prove the unproved goal, we still need to **prove** that the lemma we have guessed is in fact **true**.

Guessing Lemmas from Failed Proof Attempts

In inductive theorem proving it often happens that our **first attempt** to prove a property is unsuccessful. However, we can **learn from a failed attempt** to **guess a lemma** that will solve the unproved goal.

In the NuTP the process of **guessing** and **applying** a lemma to a goal is supported by the **lemma enrichment** ($1e$) inference rule.

Of course, although the lemma thus guessed will typically prove the unproved goal, we still need to **prove** that the lemma we have guessed is in fact **true**.

Let us see an example, namely, proving **commutativity of addition** in the PEANO+R module.

Proving Commutativity of Addition

Our first attempt to prove **addition commutative** in PEANO+R yields two unproved goals:

```
NuITP> set goal X:Nat + Y:Nat = Y:Nat + X:Nat .
```

Initial goal set.

Goal Id: 0

Generated By: init

Skolem Ops:

None

Executable Hypotheses:

None

Non-Executable Hypotheses:

None

Goal:

$(\$1:\text{Nat} + \$2:\text{Nat}) = (\$2:\text{Nat} + \$1:\text{Nat})$

```
NuITP> apply gsi! to 0 on $1:Nat .
```

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

Proving Commutativity of Addition

Our first attempt to prove **addition commutative** in PEANO+R yields two unproved goals:

```
NuITP> set goal X:Nat + Y:Nat = Y:Nat + X:Nat .
```

Initial goal set.

Goal Id: 0

Generated By: init

Skolem Ops:

None

Executable Hypotheses:

None

Non-Executable Hypotheses:

None

Goal:

$(\$1:\text{Nat} + \$2:\text{Nat}) = (\$2:\text{Nat} + \$1:\text{Nat})$

```
NuITP> apply gsi! to 0 on $1:Nat .
```

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

Proving Commutativity of Addition (II)

Goal Id: 0.1.1

Generated By: EPS

Skolem Ops:

None

Executable Hypotheses:

None

Non-Executable Hypotheses:

None

Goal:

$\$2:\text{Nat} = (0 + \$2:\text{Nat})$

Goal Id: 0.2.1

Generated By: EPS

Skolem Ops:

$\$3.\text{Nat}$

Executable Hypotheses:

None

Non-Executable Hypotheses:

$(\$3 + \$2:\text{Nat}) = (\$2:\text{Nat} + \$3)$

Goal:

$s(\$2:\text{Nat} + \$3) = (s(\$3) + \$2:\text{Nat})$

Proving Commutativity of Addition (II)

Goal Id: 0.1.1

Generated By: EPS

Skolem Ops:

None

Executable Hypotheses:

None

Non-Executable Hypotheses:

None

Goal:

$\$2:\text{Nat} = (0 + \$2:\text{Nat})$

Goal Id: 0.2.1

Generated By: EPS

Skolem Ops:

$\$3.\text{Nat}$

Executable Hypotheses:

None

Non-Executable Hypotheses:

$(\$3 + \$2:\text{Nat}) = (\$2:\text{Nat} + \$3)$

Goal:

$s(\$2:\text{Nat} + \$3) = (s(\$3) + \$2:\text{Nat})$

Proving Commutativity of Addition (III)

However, we can **guess** from the unproved base case (goal 0.0.1):
 $\$2:\text{Nat} = (0 + \$2:\text{Nat})$ the **lemma** $0 + X = X$. In the NuITP we
 can then **apply** this lemma to goal 0.0.1 as follows:

```
NuITP> apply le! to 0.1.1 with 0 + X:Nat = X:Nat .
```

Lemma Enrichment with Equality Predicate Simplification (LE!) applied to
 goal 0.1.1.

Goal 0.1.1.2.1 has been proved.

Goal Id: 0.1.1.1

Generated By: LE

Skolem Ops:

None

Executable Hypotheses:

None

Non-Executable Hypotheses:

None

Goal:

$\$3:\text{Nat} = (0 + \$3:\text{Nat})$

Proving Commutativity of Addition (III)

However, we can **guess** from the unproved base case (goal 0.0.1):
 $\$2:\text{Nat} = (0 + \$2:\text{Nat})$ the **lemma** $0 + X = X$. In the NuITP we
 can then **apply** this lemma to goal 0.0.1 as follows:

```
NuITP> apply le! to 0.1.1 with 0 + X:Nat = X:Nat .
```

Lemma Enrichment with Equality Predicate Simplification (LE!) applied to
 goal 0.1.1.

Goal 0.1.1.2.1 has been proved.

Goal Id: 0.1.1.1

Generated By: LE

Skolem Ops:

None

Executable Hypotheses:

None

Non-Executable Hypotheses:

None

Goal:

$\$3:\text{Nat} = (0 + \$3:\text{Nat})$

Proving Commutativity of Addition (IV)

The base case has thus been proved, but we still **need to prove** the guessed lemma (goal 0.1.1.1). We do so as follows:

Proving Commutativity of Addition (IV)

The base case has thus been proved, but we still **need to prove** the guessed lemma (goal 0.1.1.1). We do so as follows:

NuITP> apply gsi! to 0.1.1.1 on \$3:Nat .

Generator Set Induction with Equality Predicate Simplification (GSI!)
applied to goal 0.1.1.1.

Goals 0.1.1.1.1.1 and 0.1.1.1.2.1 have been proved.

Unproven goals:

Goal Id: 0.2.1

Generated By: EPS

Skolem Ops:

\$3.Nat

Executable Hypotheses:

None

Non-Executable Hypotheses:

$(\$3 + \$2:\text{Nat}) = (\$2:\text{Nat} + \$3)$

Goal:

$s(\$2:\text{Nat} + \$3) = (s(\$3) + \$2:\text{Nat})$

Proving Commutativity of Addition (V)

We still need to deal with the unproved **induction step** goal (0.2.1):
 $s(\$2:\text{Nat} + \$3) = (s(\$3) + \$2:\text{Nat})$ for which we can **guess**
the lemma $s(N) + M = s(M + N)$ and apply it to 0.2.1:

Proving Commutativity of Addition (V)

We still need to deal with the unproved **induction step** goal (0.2.1):
 $s(\$2:\text{Nat} + \$3) = (s(\$3) + \$2:\text{Nat})$ for which we can **guess**
 the lemma $s(N) + M = s(M + N)$ and apply it to 0.2.1:

```
NuITP> apply le! to 0.2.1 with s(N:Nat) + M:Nat = s(M:Nat + N:Nat) .
```

Lemma Enrichment with Equality Predicate Simplification (LE!) applied
 to goal 0.2.1.

Goal 0.2.1.2.1 has been proved.

Goal Id: 0.2.1.1

Generated By: LE

Skolem Ops:

$\$3.\text{Nat}$

Executable Hypotheses:

None

Non-Executable Hypotheses:

$(\$3 + \$2:\text{Nat}) = (\$2:\text{Nat} + \$3)$

Goal:

$s(\$4:\text{Nat} + \$5:\text{Nat}) = (s(\$5:\text{Nat}) + \$4:\text{Nat})$

Proving Commutativity of Addition (V)

We still need to deal with the unproved **induction step** goal (0.2.1):
 $s(\$2:\text{Nat} + \$3) = (s(\$3) + \$2:\text{Nat})$ for which we can **guess**
 the lemma $s(N) + M = s(M + N)$ and apply it to 0.2.1:

```
NuITP> apply le! to 0.2.1 with s(N:Nat) + M:Nat = s(M:Nat + N:Nat) .
```

Lemma Enrichment with Equality Predicate Simplification (LE!) applied
 to goal 0.2.1.

Goal 0.2.1.2.1 has been proved.

Goal Id: 0.2.1.1

Generated By: LE

Skolem Ops:

$\$3.\text{Nat}$

Executable Hypotheses:

None

Non-Executable Hypotheses:

$(\$3 + \$2:\text{Nat}) = (\$2:\text{Nat} + \$3)$

Goal:

$s(\$4:\text{Nat} + \$5:\text{Nat}) = (s(\$5:\text{Nat}) + \$4:\text{Nat})$

Proving Commutativity of Addition (VI)

Lastly, we need to prove the guessed lemma (0.2.1.1):

```
NuITP> apply gsi! to 0.2.1.1 on $4:Nat .
```

```
Generator Set Induction with Equality Predicate Simplification (GSI!)
  applied to goal 0.2.1.1.
```

```
Goal Id: 0.2.1.1.1.1
```

```
Generated By: EPS
```

```
Skolem Ops:
```

```
  $3.Nat
```

```
Executable Hypotheses:
```

```
  None
```

```
Non-Executable Hypotheses:
```

```
  ($3 + $2:Nat) =($2:Nat + $3)
```

```
Goal:
```

```
  $5:Nat =(0 + $5:Nat)
```

Proving Commutativity of Addition (VI)

Lastly, we need to prove the guessed lemma (0.2.1.1):

```
NuITP> apply gsi! to 0.2.1.1 on $4:Nat .
```

```
Generator Set Induction with Equality Predicate Simplification (GSI!)
  applied to goal 0.2.1.1.
```

```
Goal Id: 0.2.1.1.1.1
```

```
Generated By: EPS
```

```
Skolem Ops:
```

```
  $3.Nat
```

```
Executable Hypotheses:
```

```
  None
```

```
Non-Executable Hypotheses:
```

```
  ($3 + $2:Nat) =($2:Nat + $3)
```

```
Goal:
```

```
  $5:Nat =(0 + $5:Nat)
```

Proving Commutativity of Addition (VII)

Goal Id: 0.2.1.1.2.1

Generated By: EPS

Skolem Ops:

\$3.Nat

\$6.Nat

Executable Hypotheses:

None

Non-Executable Hypotheses:

$(\$3 + \$2:\text{Nat}) = (\$2:\text{Nat} + \$3)$

$s(\$6 + \$5:\text{Nat}) = (s(\$5:\text{Nat}) + \$6)$

Goal:

$s(\$6 + \$5:\text{Nat}) = (s(\$6) + \$5:\text{Nat})$

Proving Commutativity of Addition (VII)

Goal Id: 0.2.1.1.2.1

Generated By: EPS

Skolem Ops:

\$3.Nat

\$6.Nat

Executable Hypotheses:

None

Non-Executable Hypotheses:

$(\$3 + \$2:\text{Nat}) = (\$2:\text{Nat} + \$3)$

$s(\$6 + \$5:\text{Nat}) = (s(\$5:\text{Nat}) + \$6)$

Goal:

$s(\$6 + \$5:\text{Nat}) = (s(\$6) + \$5:\text{Nat})$

The **base case** for the guessed Lemma 0.1.1.1 is $\$5:\text{Nat} = (0 + \$5:\text{Nat})$, which we can prove with our previous lemma $0 + X = X$, so I leave this part for the reader. For the **induction step** (goal 0.2.1.1.2.1) we just apply `gsi` to finish the proof:

Proving Commutativity of Addition (VIII)

```
NuITP> apply gsi! to 0.2.1.1.2.1 on $5:Nat .
```

```
Generator Set Induction with Equality Predicate Simplification (GSI!) applied  
to goal 0.2.1.1.2.1.
```

```
Goals 0.2.1.1.2.1.1.1 and 0.2.1.1.2.1.2.1 have been proved.
```

```
qed
```

Proving Commutativity of Addition (VIII)

```
NuITP> apply gsi! to 0.2.1.1.2.1 on $5:Nat .
```

```
Generator Set Induction with Equality Predicate Simplification (GSI!) applied  
to goal 0.2.1.1.2.1.
```

```
Goals 0.2.1.1.2.1.1.1 and 0.2.1.1.2.1.2.1 have been proved.
```

```
qed
```

The general **heuristic** to deal with unproven goals could be summarized with the motto: **generalize and conquer!**

Proving Commutativity of Addition (VIII)

```
NuITP> apply gsi! to 0.2.1.1.2.1 on $5:Nat .
```

Generator Set Induction with Equality Predicate Simplification (GSI!) applied to goal 0.2.1.1.2.1.

Goals 0.2.1.1.2.1.1.1 and 0.2.1.1.2.1.2.1 have been proved.

```
qed
```

The general **heuristic** to deal with unproven goals could be summarized with the motto: **generalize and conquer!** That, is we **guess** the needed lemma by **generalizing** the unproved goal.

Internalize and Conquer

Proving lemmas is tedious.

Internalize and Conquer

Proving lemmas is tedious. Can we do better and achieve **shorter** and perhaps **lemmaless** proofs?

Internalize and Conquer

Proving lemmas is tedious. Can we do better and achieve **shorter** and perhaps **lemmaless** proofs? The answer is **Yes!!** The better proof method has the motto: **internalize and conquer!**

Internalize and Conquer

Proving lemmas is tedious. Can we do better and achieve **shorter** and perhaps **lemmaless** proofs? The answer is **Yes!!** The better proof method has the motto: **internalize and conquer!** It is supported by the NuTP based on the Lemma Internalization Theorems 2 and 3 of Lecture 14.

Internalize and Conquer

Proving lemmas is tedious. Can we do better and achieve **shorter** and perhaps **lemmaless** proofs? The answer is **Yes!!** The better proof method has the motto: **internalize and conquer!** It is supported by the NuTP based on the Lemma Internalization Theorems 2 and 3 of Lecture 14.

This method is based on two simple ideas:

Internalize and Conquer

Proving lemmas is tedious. Can we do better and achieve **shorter** and perhaps **lemmaless** proofs? The answer is **Yes!!** The better proof method has the motto: **internalize and conquer!** It is supported by the NuTP based on the Lemma Internalization Theorems 2 and 3 of Lecture 14.

This method is based on two simple ideas:

- 1 **Low Hanging Fruit:**

Internalize and Conquer

Proving lemmas is tedious. Can we do better and achieve **shorter** and perhaps **lemmaless** proofs? The answer is **Yes!!** The better proof method has the motto: **internalize and conquer!** It is supported by the NuTP based on the Lemma Internalization Theorems 2 and 3 of Lecture 14.

This method is based on two simple ideas:

- 1 **Low Hanging Fruit:** If you have to prove several properties, **arrange them in order of difficulty and dependence.**

Internalize and Conquer

Proving lemmas is tedious. Can we do better and achieve **shorter** and perhaps **lemmaless** proofs? The answer is **Yes!!** The better proof method has the motto: **internalize and conquer!** It is supported by the NuTP based on the Lemma Internalization Theorems 2 and 3 of Lecture 14.

This method is based on two simple ideas:

- 1 **Low Hanging Fruit:** If you have to prove several properties, **arrange them in order of difficulty and dependence**. E.g., prove associativity of addition before commutativity, and both before associativity and commutativity of multiplication.

Internalize and Conquer

Proving lemmas is tedious. Can we do better and achieve **shorter** and perhaps **lemmaless** proofs? The answer is **Yes!!** The better proof method has the motto: **internalize and conquer!** It is supported by the NuTP based on the Lemma Internalization Theorems 2 and 3 of Lecture 14.

This method is based on two simple ideas:

- 1 **Low Hanging Fruit:** If you have to prove several properties, **arrange them in order of difficulty and dependence**. E.g., prove associativity of addition before commutativity, and both before associativity and commutativity of multiplication.
- 2 **Internalize Everything!**

Internalize and Conquer

Proving lemmas is tedious. Can we do better and achieve **shorter** and perhaps **lemmaless** proofs? The answer is **Yes!!** The better proof method has the motto: **internalize and conquer!** It is supported by the NuTP based on the Lemma Internalization Theorems 2 and 3 of Lecture 14.

This method is based on two simple ideas:

- 1 **Low Hanging Fruit:** If you have to prove several properties, **arrange them in order of difficulty and dependence**. E.g., prove associativity of addition before commutativity, and both before associativity and commutativity of multiplication.
- 2 **Internalize Everything!** Use every single property you have already proved to prove harder properties by **internalizing** it.

Let us see this method in action in an **arithmetic case study**:

An Example: Natural Number Arithmetic

```

fmod NATURAL-ARITH is
  sorts Nat NzNat .
  subsort NzNat < Nat .
  op 0 : -> Nat [ctor metadata "1"] .
  op s : Nat -> NzNat [ctor metadata "2"] .
  op +_+ : Nat Nat -> Nat [metadata "3"] .
  op *_* : Nat Nat -> Nat [metadata "4"] .
  op *_* : NzNat NzNat -> NzNat [metadata "5"] .
  op ^_^ : NzNat Nat -> NzNat [metadata "6"] .   *** exponentiation
  vars n m k : Nat .  vars n' k' m' : NzNat .
  eq n + 0 = n .
  eq n + s(m) = s(n + m) .
  eq n * 0 = 0 .
  eq n * s(m) = n + (n * m) .
  eq n' ^ 0 = s(0) .
  eq n' ^ s(m) = n' * (n' ^ m) .
endfm

```

An Example: Natural Number Arithmetic

```
fmod NATURAL-ARITH is
  sorts Nat NzNat .
  subsort NzNat < Nat .
  op 0 : -> Nat [ctor metadata "1"] .
  op s : Nat -> NzNat [ctor metadata "2"] .
  op +_ : Nat Nat -> Nat [metadata "3"] .
  op *_ : Nat Nat -> Nat [metadata "4"] .
  op *_ : NzNat NzNat -> NzNat [metadata "5"] .
  op ^_ : NzNat Nat -> NzNat [metadata "6"] .   *** exponentiation
  vars n m k : Nat .  vars n' k' m' : NzNat .
  eq n + 0 = n .
  eq n + s(m) = s(n + m) .
  eq n * 0 = 0 .
  eq n * s(m) = n + (n * m) .
  eq n' ^ 0 = s(0) .
  eq n' ^ s(m) = n' * (n' ^ m) .
endfm
```

This module's **canonical term algebra** $\mathbb{C}_{\Sigma/E,B}$ is just \mathbb{N} with the $s_{\mathbb{N}}$, $+_{\mathbb{N}}$, $*_{\mathbb{N}}$, and $(-)^{(-)}_{\mathbb{N}}$ functions.

An Example: Natural Number Arithmetic

```

fmod NATURAL-ARITH is
  sorts Nat NzNat .
  subsort NzNat < Nat .
  op 0 : -> Nat [ctor metadata "1"] .
  op s : Nat -> NzNat [ctor metadata "2"] .
  op +_ : Nat Nat -> Nat [metadata "3"] .
  op *_ : Nat Nat -> Nat [metadata "4"] .
  op *_ : NzNat NzNat -> NzNat [metadata "5"] .
  op ^_ : NzNat Nat -> NzNat [metadata "6"] .   *** exponentiation
  vars n m k : Nat .  vars n' k' m' : NzNat .
  eq n + 0 = n .
  eq n + s(m) = s(n + m) .
  eq n * 0 = 0 .
  eq n * s(m) = n + (n * m) .
  eq n' ^ 0 = s(0) .
  eq n' ^ s(m) = n' * (n' ^ m) .
endfm

```

This module's **canonical term algebra** $\mathbb{C}_{\Sigma/E,B}$ is just \mathbb{N} with the $s_{\mathbb{N}}$, $+_{\mathbb{N}}$, $*_{\mathbb{N}}$, and $(-)_{\mathbb{N}}^{(-)}$ functions. Consider some **properties** of NATURAL-ARITH:

Arithmetic Properties of NATURAL-ARITH

NATURAL-ARITH **should** enjoy the following **arithmetic properties**:

Arithmetic Properties of NATURAL-ARITH

NATURAL-ARITH **should** enjoy the following **arithmetic properties**:

- 1 $+$ _{\mathbb{N}} should be **associative** and **commutative**

Arithmetic Properties of NATURAL-ARITH

NATURAL-ARITH should enjoy the following arithmetic properties:

- 1 $+$ _{\mathbb{N}} should be associative and commutative
- 2 $*$ _{\mathbb{N}} should be left and right distributive over $+$ _{\mathbb{N}}

Arithmetic Properties of NATURAL-ARITH

NATURAL-ARITH **should** enjoy the following **arithmetic properties**:

- 1 $+\mathbb{N}$ should be **associative** and **commutative**
- 2 $*\mathbb{N}$ should be **left** and **right distributive** over $+\mathbb{N}$
- 3 $*\mathbb{N}$ should be **associative** and **commutative**

Arithmetic Properties of NATURAL-ARITH

NATURAL-ARITH **should** enjoy the following **arithmetic properties**:

- 1 $+_{\mathbb{N}}$ should be **associative** and **commutative**
- 2 $*_{\mathbb{N}}$ should be **left** and **right distributive** over $+_{\mathbb{N}}$
- 3 $*_{\mathbb{N}}$ should be **associative** and **commutative**
- 4 $(-)^{(-)}_{\mathbb{N}}$ should enjoy the property: $x^{y+z} = x^y * x^z$

Arithmetic Properties of NATURAL-ARITH

NATURAL-ARITH **should** enjoy the following **arithmetic properties**:

- ① $+_{\mathbb{N}}$ should be **associative** and **commutative**
- ② $*_{\mathbb{N}}$ should be **left** and **right distributive** over $+_{\mathbb{N}}$
- ③ $*_{\mathbb{N}}$ should be **associative** and **commutative**
- ④ $(-)_{\mathbb{N}}^{(-)}$ should enjoy the property: $x^{y+z} = x^y * x^z$

Reflecting on the (tedious) proof of commutativity for $+_{\mathbb{N}}$, which required the lemmas $0 + M = M$ and $s(N) + M = s(M + N)$, we can see that proving commutativity of $+_{\mathbb{N}}$ would have been trivial if we had first proved and **internalized** that programs PEANO+R and PEANO+L (whose equations are the above lemmas) are **semantically equivalent**.

Arithmetic Properties of NATURAL-ARITH

NATURAL-ARITH **should** enjoy the following **arithmetic properties**:

- ① $+\mathbb{N}$ should be **associative** and **commutative**
- ② $*\mathbb{N}$ should be **left** and **right distributive** over $+\mathbb{N}$
- ③ $*\mathbb{N}$ should be **associative** and **commutative**
- ④ $(-)\mathbb{N}^{(-)}$ should enjoy the property: $x^{y+z} = x^y * x^z$

Reflecting on the (tedious) proof of commutativity for $+\mathbb{N}$, which required the lemmas $0 + M = M$ and $s(N) + M = s(M + N)$, we can see that proving commutativity of $+\mathbb{N}$ would have been trivial if we had first proved and **internalized** that programs PEANO+R and PEANO+L (whose equations are the above lemmas) are **semantically equivalent**. The analogous observation holds for proving commutativity of $*\mathbb{N}$. Let's prove all these arithmetic properties!

Proving Properties of NATURAL-ARITH

```
=====
      NuITP (alpha 22)
      Inductive Theorem Prover
      for Maude Equational Theories
=====
      Copyright 2021-2023
      Universitat Politècnica de València
```

```
NuITP> set module NATURAL-ARITH .
```

```
Module NATURAL-ARITH is now active.
```

Proving Properties of NATURAL-ARITH

```

=====
      NuITP (alpha 22)
    Inductive Theorem Prover
  for Maude Equational Theories
=====
      Copyright 2021-2023
  Universitat Politècnica de València
  
```

```
NuITP> set module NATURAL-ARITH .
```

```
Module NATURAL-ARITH is now active.
```

By the Program Equivalence Theorem in Lecture 14, We can next prove the **semantic equivalence** between the left- and right-recursive definitions of addition if we prove the following goal by generator set induction using standard induction (SIND):

Proving Properties of NATURAL-ARITH (II)

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

```
...
```

```
Goal Id: 0
```

```
...
```

```
Goal:
```

```
($2:Nat =(0 + $2:Nat)) /\ s($1:Nat + $2:Nat) =(s($1:Nat) + $2:Nat)
```

```
NuITP> apply gsi! to 0 on $2:Nat .
```

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

Goals 0.1.1 and 0.2.1 have been proved.

```
qed
```

Proving Properties of NATURAL-ARITH (II)

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

```
...
```

```
Goal Id: 0
```

```
...
```

```
Goal:
```

```
($2:Nat =(0 + $2:Nat)) /\ s($1:Nat + $2:Nat) =(s($1:Nat) + $2:Nat)
```

```
NuITP> apply gsi! to 0 on $2:Nat .
```

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

Goals 0.1.1 and 0.2.1 have been proved.

```
qed
```

The NuITP allows us to apply the Lemma Internalization Theorem 2 in Lecture 14 to **internalize** the just-proved semantically equivalent left-recursive equations for $+$ we just proved by **adding them to the current module** as follows:

Proving Properties of NATURAL-ARITH (III)

```
NuITP> internalize .
```

Proving Properties of NATURAL-ARITH (III)

```
NuITP> internalize .
```

Now that we have internalized the above semantic equivalence, we prove the **associativity** of $+$, which succeeds with one blow:

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

```
...
```

```
Goal Id: 0
```

```
...
```

```
Goal:
```

```
($1:Nat + ($2:Nat + $3:Nat)) = (($1:Nat + $2:Nat) + $3:Nat)
```

```
NuITP> apply gsi! to 0 on $3:Nat .
```

Generator Set Induction with Equality Predicate Simplification (GSI!) applied

Goals 0.1.1 and 0.2.1 have been proved.

```
qed
```

Proving Properties of NATURAL-ARITH (IV)

The NuTP allows us to apply the Lemma Internalization Theorem 3 of Lecture 14 to internalize associativity as an **axiom**. From now on, all simplification with $+$ expressions can be done **modulo** associativity:

Proving Properties of NATURAL-ARITH (IV)

The NuITP allows us to apply the Lemma Internalization Theorem 3 of Lecture 14 to internalize associativity as an **axiom**. From now on, all simplification with $+$ expressions can be done **modulo** associativity:

```
NuITP> internalize as assoc .
```

Proving Properties of NATURAL-ARITH (IV)

The NuITP allows us to apply the Lemma Internalization Theorem 3 of Lecture 14 to internalize associativity as an **axiom**. From now on, all simplification with $+$ expressions can be done **modulo** associativity:

```
NuITP> internalize as assoc .
```

Next we prove **commutativity** for $+$, which succeeds with one blow:

Proving Properties of NATURAL-ARITH (IV)

The NuITP allows us to apply the Lemma Internalization Theorem 3 of Lecture 14 to internalize associativity as an **axiom**. From now on, all simplification with $+$ expressions can be done **modulo** associativity:

```
NuITP> internalize as assoc .
```

Next we prove **commutativity** for $+$, which succeeds with one blow:

```
NuITP> set goal (X:Nat + Y:Nat = Y:Nat + X:Nat) .
```

```
...
```

```
Goal Id: 0
```

```
...
```

```
Goal:
```

```
($1:Nat + $2:Nat) = $2:Nat + $1:Nat
```

```
NuITP> apply gsi! to 0 on $1:Nat .
```

```
Generator Set Induction with Equality Predicate Simplification (GSI!) applied
```

```
Goals 0.1.1 and 0.2.1 have been proved.
```

Proving Properties of NATURAL-ARITH (V)

```
NuITP> internalize as comm .
```

Proving Properties of NATURAL-ARITH (V)

```
NuITP> internalize as comm .
```

From now on, all simplification with $+$ expressions can be done **modulo** associativity **and** commutativity.

Proving Properties of NATURAL-ARITH (V)

```
NuITP> internalize as comm .
```

From now on, all simplification with $+$ expressions can be done **modulo** associativity **and** commutativity. Next we prove **right distributivity** of $*$ over $+$, which succeeds with one blow:

Proving Properties of NATURAL-ARITH (V)

```
NuITP> internalize as comm .
```

From now on, all simplification with $+$ expressions can be done **modulo** associativity **and** commutativity. Next we prove **right distributivity** of $*$ over $+$, which succeeds with one blow:

```
NuITP> set goal X:Nat * (Y:Nat + Z:Nat) = (X:Nat * Y:Nat) + (X:Nat * Z:Nat) .
```

```
...
```

```
Goal Id: 0
```

```
...
```

```
Goal:
```

```
($1:Nat * $3:Nat + $2:Nat) =($1:Nat * $3:Nat) +($1:Nat * $2:Nat)
```

```
NuITP> apply gsi! to 0 on $2:Nat .
```

Generator Set Induction with Equality Predicate Simplification (GSI!) applied

Goals 0.1.1 and 0.2.1 have been proved.

```
qed
```

```
NuITP> internalize .
```

Proving Properties of NATURAL-ARITH (VI)

To overcome $*$'s recursion on its **second** argument we proceed as for $+$, which succeeds with one blow:

Proving Properties of NATURAL-ARITH (VI)

To overcome $*$'s recursion on its **second** argument we proceed as for $+$, which succeeds with one blow:

```
NuITP> set goal ((0 * Y:Nat = 0) /\ (s(X:Nat) * Y:Nat) = (Y:Nat + (X:Nat * Y:Nat)
...
Goal Id: 0
...
Goal:
  (0 = (0 * $2:Nat)) /\ (s($1:Nat) * $2:Nat) = $2:Nat + ($1:Nat * $2:Nat)
```

```
NuITP> apply gsi! to 0 on $2:Nat .
```

Generator Set Induction with Equality Predicate Simplification (GSI!) applied

Goals 0.1.1 and 0.2.1 have been proved.

qed

```
NuITP> internalize .
```

Proving Properties of NATURAL-ARITH (VII)

Next we prove **left distributivity** of $*$ over $+$, which succeeds with one blow:

Proving Properties of NATURAL-ARITH (VII)

Next we prove **left distributivity** of $*$ over $+$, which succeeds with one blow:

```
NuITP> set goal (Y:Nat + Z:Nat) * X:Nat = (Y:Nat * X:Nat) + (Z:Nat * X:Nat) .
...
Goal Id: 0
...
Goal:
  (($3:Nat + $2:Nat) * $1:Nat) = ($3:Nat * $1:Nat) + ($2:Nat * $1:Nat)
```

```
NuITP> apply gsi! to 0 on $2:Nat .
```

Generator Set Induction with Equality Predicate Simplification (GSI!) applied

Goals 0.1.1 and 0.2.1 have been proved.

qed

```
NuITP> internalize .
```

Proving Properties of NATURAL-ARITH (VIII)

Next we prove **associativity** of $*$, which succeeds with one blow:

Proving Properties of NATURAL-ARITH (VIII)

Next we prove **associativity** of $*$, which succeeds with one blow:

```
NuITP> set goal X:Nat * (Y:Nat * Z:Nat) = (X:Nat * Y:Nat) * Z:Nat .
```

```
...
```

```
Goal Id: 0
```

```
...
```

```
Goal:
```

```
  ($1:Nat * ($2:Nat * $3:Nat)) = (($1:Nat * $2:Nat) * $3:Nat)
```

```
NuITP> apply gsi! to 0 on $3:Nat .
```

Generator Set Induction with Equality Predicate Simplification (GSI!) applied

Goals 0.1.1 and 0.2.1 have been proved.

qed

```
NuITP> internalize as assoc .
```


Proving Properties of NATURAL-ARITH (IX)

Next we prove **commutativity** of $*$, which succeeds with one blow:

Proving Properties of NATURAL-ARITH (IX)

Next we prove **commutativity** of $*$, which succeeds with one blow:

```
NuITP> set goal (X:Nat * Y:Nat = Y:Nat * X:Nat) .
```

```
...
```

```
Goal Id: 0
```

```
...
```

```
Goal:
```

```
  ($1:Nat * $2:Nat) = $2:Nat * $1:Nat
```

```
NuITP> apply gsi! to 0 on $2:Nat .
```

Generator Set Induction with Equality Predicate Simplification (GSI!) applied

Goals 0.1.1 and 0.2.1 have been proved.

qed

```
NuITP> internalize as comm .
```

Proving Properties of NATURAL-ARITH (X)

We finish proving **all** arithmetic properties (1)–(4) from slide 4 by proving the equality $x^{y+z} = x^y * x^z$ of **exponentiation**, which succeeds with one blow:

Proving Properties of NATURAL-ARITH (X)

We finish proving **all** arithmetic properties (1)–(4) from slide 4 by proving the equality $x^{y+z} = x^y * x^z$ of **exponentiation**, which succeeds with one blow:

```
NuITP> set goal (X:NzNat ^ (Y:Nat + Z:Nat) = (X:NzNat ^ Y:Nat) * (X:NzNat ^ Z:Nat))
...
Goal Id: 0
...
Goal:
  ((($1:NzNat ^ $3:Nat) * ($1:NzNat ^ $2:Nat)) = ($1:NzNat ^ $3:Nat + $2:Nat))
```

```
NuITP> apply gsi! to 0 on $2:Nat .
```

Generator Set Induction with Equality Predicate Simplification (GSI!) applied

Goals 0.1.1 and 0.2.1 have been proved.

qed