CSCI.6962/4962 Software Verification—Fundamental Proof Methods in Computer Science (Arkoudas and Musser)—Chapter 3.1-3.8

Instructor: Carlos Varela
Rensselaer Polytechnic Institute
Spring 2020
Proving Equalities

Goal: to become familiar with *equational proofs* as a type of inference.

- numeric equations
- equality chaining
- terms and sentences as trees
- logic behind equality chaining
- sample proofs
- mathematical induction
- *list equations*
- *polymorphic datatypes*
- *ground terms*
- *top-down proof development*
**Datatype for natural numbers**

```plaintext
datatype N := zero | (S N)
```

This defines a datatype `N` with two constructors: `zero` and `S`. The definition says that the values of sort `N` are:

```
zero
(S zero)
(S (S zero))
(S (S (S zero)))
...
```

- 0 is represented by `zero`,
- \(n + 1\) is obtained by applying the “successor function” `S` to the natural number `n`,
- these are *the only values* that are natural numbers.
Functions over natural numbers

\[
\text{declare Plus: } [N N] \rightarrow N \[+\]
\]

- \text{Plus} takes two natural numbers as inputs and produces a natural number as output.
- The expression \([+\)] at the end of the declaration overloads the built-in symbol + so that it can be used as an alias for \text{Plus} whenever the context allows it.

We can now write the following equivalent terms:

- \((\text{Plus (S zero) zero})\)
- \(((\text{S zero) Plus zero})\)
- \((\text{S zero Plus zero})\)
- \((\text{S zero + zero})\)

after setting the precedence of S to be higher than Plus.
Reasoning about functions

We can express commutativity of addition as follows:

> (forall n m . n Plus m = m Plus n)

Sentence: (forall ?n:N

(forall ?m:N

(= (Plus ?n:N ?m:N)

(Plus ?m:N ?n:N))))

Axioms for + can be added to the global assumption base using universally quantified equations:

assert right-zero := (forall n . n + zero = n)

assert right-nonzero := (forall n m . n + S m = S (n + m))
Instantiating axioms

The meaning of \((S\ zero + \ zero)\) is determined by the equation that is just the special case of right-zero with the ground term \((S\ zero)\) substituted for \(n\).

\[
(!\text{instance right-zero } [(S\ zero)])
\]
to which Athena responds:

**Theorem:** \((= (\text{Plus} (S\ zero) \\
                               \text{zero}) \\
                  (S\ zero))\)

That is, \(1 + 0 = 1\). Likewise:

\[
> (!\text{instance right-nonzero } [\text{zero} (S\ zero)])
\]

**Theorem:** \((= (\text{Plus} \text{zero} \\
                   (S\ (S\ zero))) \\
                   (S\ (\text{Plus} \text{zero} \\
                                  (S\ zero))))\)

CSCI.6962/4962 Software Verification—Fundamental Proof Methods in Computer Science (Arkoudas and Musser)—Chapter 3.1-3.8 – p. 6/38
The instance method

- The first argument to instance is a universally quantified sentence $p$ in the assumption base.
- The second is a list $L$ of terms.

If $p = (\forall v_1 \cdots v_n . q)$ and $L = [t_1 \cdots t_k]$, where $k \leq n$, then instance produces the sentence

$$(\forall v_{k+1} \cdots v_n . q')$$

where $q'$ results from substituting $t_i$ for $v_i$ in $q$, for $i = 1, \ldots, k$.

> (!instance right-nonzero [zero])

**Theorem:** $(\forall ?v303:N)

\quad (= (\text{Plus zero}

\quad \quad (S ?v303:N))

\quad \quad (S (\text{Plus zero } ?v303:N))))$
Equality chaining

What about the meaning of `Plus` for larger ground term inputs, like 

\[(S \ S \ zero \ + \ S \ S \ zero)\]?

In other words, can we now deduce that \(2 + 2 = 4\)?

Yes, and here is one way to do it:

```
(!chain [(S S zero + S S zero)
  = (S (S S zero + S zero)) [right-nonzero]
  = (S S (S S zero + zero)) [right-nonzero]
  = (S S S S zero) [right-zero]
  ])
```

Here we have used `chain`, an Athena method for proving equations by chaining together a sequence of terms connected by equalities.
The chain method

In general,

\[ (!\text{chain } [t_0 = t_1 [p_1] = t_2 [p_2] = \cdots = t_n [p_n]]) \]

- attempts to derive the identity \((t_0 = t_n)\),
- each \(p_i\) must be in the assumption base and
- each equation \((t_{i-1} = t_i)\) must follow from \(p_i\), typically by one of five fundamental axioms of equality, for \(i = 1, \ldots, n\):
  - reflexivity
  - symmetry
  - transitivity
  - functional substitution, or
  - relational substitution
Equality chaining

- $n = 3$,
- $p_1 = p_2 = \text{right-nonzero}$ and $p_3 = \text{right-zero}$,
- Athena responds with the theorem proved:

\[
\text{Theorem: } (= (\text{Plus} (S (S \text{ zero}))
\]
\[
(S (S \text{ zero}))
\]
\[
(S (S (S (S \text{ zero}))))
\]

For each step $i$ (from $t_{i-1}$ to $t_i$), $[p_i]$ is its \textit{justification list} and each $p_i$ is its \textit{justifier}. 
The logic behind equality chaining

A firm foundation for reasoning about equalities is provided by the basic equality axioms:

1. **Reflexivity**: $\forall x. \ x = x$.

2. **Symmetry**: $\forall x y. \ x = y \Rightarrow y = x$.

3. **Transitivity**: $\forall x y z. \ x = y \land y = z \Rightarrow x = z$.

4. **Functional Substitution**: For any $n$-arity function symbol $f$,

   $\forall x_1 \cdots x_n \ y_1 \cdots y_n. \ x_1 = y_1 \land \ldots \land x_n = y_n \Rightarrow \ f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$.

5. **Relational Substitution**: For any $n$-arity relation symbol $R$,

   $\forall x_1 \cdots x_n \ y_1 \cdots y_n. \ x_1 = y_1 \land \ldots \land x_n = y_n \land R(x_1, \ldots, x_n) \Rightarrow \ R(y_1, \ldots, y_n)$.
Terms and sentences as trees

Terms and sentences are tree-structured objects.

- A variable or a constant symbol can be viewed as a simple one-node tree (a leaf node).
- An application of the form \((f \, t_1 \, \cdots \, t_n)\) for \(n > 0\) can be viewed as a tree with
  - the symbol \(f\) at the root and
  - with the trees corresponding to \(t_1, \ldots, t_n\) as its immediate subtrees, arranged from left to right in that order.
Terms and sentences as trees

The term \((\text{Plus} \ (S \ \text{zero}) \ (\text{Plus} \ (S \ x) \ (S \ y)))\) can be depicted as follows:
Abstract syntax

Tree representations of terms depict their *abstract syntax*:

- not specified whether prefix, infix, or postfix notation was used
- no argument separation syntax (e.g., commas, periods, spaces, indentation, etc.)
- these belong to a *concrete syntax*, e.g., Lisp prefix notation is used by Athena to output terms.
Dewey paths

Every node can be assigned a unique list of positive integers $[i_1 \cdots i_m]$ indicating the path that must be traversed in order to get from the root of the tree to the node in question, e.g., for $(\text{Plus} \ (\text{S} \ \text{zero}) \ (\text{Plus} \ x \ y))$:

These integer sequences are called *Dewey paths* (or Dewey positions).
Computing with Dewey paths

The Athena procedure \texttt{positions-and-subterms}

- takes a term and
- produces a list of all positions in the term, each paired in a sublist with the subterm at that position.

For example:

\begin{verbatim}
> (positions-and-subterms (Plus (S zero) (Plus x y)))

List: [[[] (Plus (S zero) (Plus ?x ?y))]
  [[1] (S zero)]
  [[1 1] zero]
  [[2] (Plus ?x ?y)]
  [[2 1] ?x]
  [[2 2] ?y]]
\end{verbatim}
Computing with Dewey paths

subterm is a useful procedure that

• takes a term $t$ and

• a position $I$ (as a list of positive integers) and

• returns the subterm of $t$ that is located at position $I$ in the tree representation of $t$.

```haskell
define (subterm t I) :=
    match I {
        [] => t
        | (list-of i rest) => (subterm (ith (children t) i) rest)
    }
```

The primitive binary procedure $\text{ith}$ takes a list of $n > 0$ values $[V_1 \cdots V_n]$ and an integer $i \in \{1, \ldots, n\}$ and returns $V_i$. 
Computing with Dewey paths

If we only want the node at a given position, we can use the following procedure: subterm-node

```lisp
define (subterm-node t I) := (root (subterm t I))

> (subterm-node (x + S S zero) [2 1])
```

Symbol: S

Another useful procedure is replace-subterm, where

(replace-subterm t I t')

- returns the term obtained from t by replacing the subterm at position I by t',
- provided that the result is well sorted.
- **Exercise:** Define the replace-subterm procedure.
Sentences as trees

Dewey paths also apply to sentences depicted as trees, e.g.:

(forall ?x (not (= zero (S ?x))))
More examples of equality chaining

We begin with the following property:

```
define left-zero := (forall n . zero + n = n)
```

which differs from right-zero in that zero appears as the first input to Plus rather than the second.

Recall we called it Plus-S-property in Chapter 1 (see induction.ath). Similarly:

```
assert left-nonzero := (forall m n . (S n) + m = S (n + m))
```

We are treating left-nonzero as an axiom (we used assert) though it can be proven from right-zero and right-nonzero.
More examples of equality chaining

A multiplication function, Times, takes two natural numbers as inputs and returns their product:

```plaintext
declare Times: [N N] -> N [*]
```

We overloaded * to mean Times when applied to N arguments. Next, the semantics:

```plaintext
assert Times-zero := (forall x . x * zero = zero)
assert Times-nonzero := (forall x y . x * S y = x * y + x)
```

If we read (S n) as n + 1, the second axiom just says

\[ x \cdot (y + 1) = x \cdot y + x. \]

Note that * has a built-in precedence greater than that of +, so that, for example, \((x * y + z)\) is parsed as \((x * y) + z\).
More examples of equality chaining

Let’s also introduce a name one and give its meaning with an equation:

\[
\text{declare one: N} \\
\text{assert one-definition := (one = S zero)}
\]

The proof of the following property:

\[
\text{define Times-right-one := (forall x . x * one = x)}
\]

provides another simple illustration of equality chaining:

\[
\text{conclude Times-right-one} \\
\text{pick-any } x:N \\
(\text{!chain } [(x * one) \\
= (x * S zero) \text{ [one-definition]} \\
= (x * zero + x) \text{ [Times-nonzero]} \\
= (zero + x) \text{ [Times-zero]} \\
= x \text{ [left-zero]}])
\]
More examples of equality chaining

Associativity of Times, which for the moment we will treat as an axiom:

\[ \text{assert Times-associative} := (\forall x \ y \ z . \ (x \ast y) \ast z = x \ast (y \ast z)) \]

An exponentiation function, \(*\ast\), follows:

\[ \text{declare } \ast\ast : [N \ N] \rightarrow N \ [310] \]

We set the precedence of \(*\ast\) higher than that of \(*\) (predefined as 300).
For semantics, we write:

\[ \text{assert Power-right-zero} := (\forall x . \ x \ast\ast \text{zero} = \text{one}) \]

\[ \text{assert Power-right-nonzero} := (\forall x \ n . \ x \ast\ast \text{S n} = x \ast x \ast\ast n) \]

or

\[ x^{n+1} = x \ast x^n \]
**Power square theorem**

Recall the following result from elementary algebra: $(x^2)^n = x^{2n}$

```plaintext
define power-square-theorem := (forall n x . (x * x) ** n = x ** (n + n))
```

If we define the following procedure:

```plaintext
define (power-square-property n) :=
    (forall x . (x * x) ** n = x ** (n + n))
```

we can express `power-square-theorem` as the proposition that every natural number has the `power-square-property`:

```plaintext
(forall n . power-square-property n).
```

Athena can verify that the two formulations are identical:

```plaintext
> (power-square-theorem equals? (forall n . power-square-property n))
Term: true
```

We call `power-square-property` a *property procedure*. 
Power square theorem

So how do we go about proving power-square-theorem?
For this theorem, instantiating only the variable $n$ for a few small values yields:

$$(\forall x . (x \times x) ^* \text{zero} = x ^* (\text{zero} + \text{zero}))$$

$$(\forall x . (x \times x) ^* S \text{zero} = x ^* (S \text{zero} + S \text{zero}))$$

$$(\forall x . (x \times x) ^* S S \text{zero} = x ^* (S S \text{zero} + S S \text{zero}))$$

$$(\forall x . (x \times x) ^* S S S \text{zero} = x ^* (S S S \text{zero} + S S S \text{zero}))$$

\ldots
# Power square theorem

The proof of \((\text{power-square-property zero})\) is simple:

<table>
<thead>
<tr>
<th>conclude power-zero-case := (power-square-property zero)</th>
</tr>
</thead>
<tbody>
<tr>
<td>pick-any (x:N)</td>
</tr>
<tr>
<td>!(chain [((x * x) ** zero)</td>
</tr>
<tr>
<td>= one</td>
</tr>
<tr>
<td>= (x ** zero)</td>
</tr>
<tr>
<td>= (x ** (zero + zero))</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Power square theorem

For \( n \equiv (S \quad \text{zero}) \), first consider the following proof:

\[
\text{conclude power-one-case := (power-square-property (S zero))}
\]

\[
\text{pick-any x:N}
\]

\[
(!\text{combine-equations}
\]

\[
(!\text{chain } (((x \times x) \times (x \times x) \times x) \times x) \times x)
\]

\[
= (((x \times x) \times (x \times x) \times x) \times x) \times x) \times x)
\]

\[
= ((x \times x) \times (x \times x) \times x) \times x) \times x)
\]

\[
= ((x \times x) \times x) \times x) \times x)
\]

\[
= ((x \times x) \times x) \times x)
\]

\[
= ((x \times x) \times x)
\]

\[
= (x \times x)
\]

\[
= (x \times x)
\]

\[
= (x \times x)
\]

\[
= (x \times x)
\]

\[
= (x \times x)
\]

\[
= (x \times x)
\]

\[
= (x \times x)
\]

\[
= (x \times x)
\]

\[
= (x \times x)
\]

\[
= (x \times x)
\]

\[
= (x \times x)
\]

\[
= (x \times x)
\]

\[
= (x \times x)
\]

\[
= (x \times x)
\]
Power square theorem

The structure of this proof, combines two applications of chain:

\[ t_0 \xrightarrow{t_1} t_n = u_m \xrightarrow{u_1} u_0 \]

If we can rewrite each side of the equation to the very same term, then we can combine the two chain conclusions to obtain the proof.
The **combine-equations method**

The method combine-equations does just that:

\[
(!\text{combine-equations } (s_0 = s_n) \ (t_0 = t_m))
\]

proves \((s_0 = t_0)\) when

- both \((s_0 = s_n)\) and \((t_0 = t_m)\) are in the assumption base and
- \(s_n\) and \(t_m\) are identical.

**chain** allows the direction of rewriting to be indicated on each step:

- If \(\rightarrow\) is used, chain only attempts to rewrite left-to-right:
  \[t_i \rightarrow t_{i+1}\].
- If \(\leftarrow\) is used, it only attempts to rewrite right-to-left:
  \[(t_{i+1} \rightarrow t_i)\].
- If \(\equiv\) is specified, chain first tries left-to-right, and if that fails, it tries right-to-left.
Power square theorem

The proof given for power-one-case can be shortened by taking advantage of the power-zero-case theorem, as follows:

<table>
<thead>
<tr>
<th>Conclude (forall x . (x * x) ** S zero = x ** (S zero + S zero))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pick-Any x:N</td>
</tr>
<tr>
<td>(!combine-equations</td>
</tr>
<tr>
<td>(!chain (((x * x) ** S zero)</td>
</tr>
<tr>
<td>--&gt; ((x * x) * ((x * x) ** zero)) [Power-right-nonzero]</td>
</tr>
<tr>
<td>--&gt; ((x * x) * (x ** (zero + zero))) [power-zero-case]</td>
</tr>
<tr>
<td>--&gt; (x * x * x ** (zero + zero)) [Times-associative]])</td>
</tr>
<tr>
<td>(!chain [(x ** (S zero + S zero))</td>
</tr>
<tr>
<td>--&gt; (x ** (S (S zero + zero))) [right-nonzero]</td>
</tr>
<tr>
<td>--&gt; (x ** (S (S (zero + zero)))) [left-nonzero]</td>
</tr>
<tr>
<td>--&gt; (x * (x ** (S (zero + zero)))) [Power-right-nonzero]</td>
</tr>
<tr>
<td>--&gt; (x * x * x ** (zero + zero)) [Power-right-nonzero]</td>
</tr>
<tr>
<td>]])</td>
</tr>
</tbody>
</table>

This proof is not just shorter, it can be generalized for any value of n.
Power square theorem

```plaintext
define power-square-step :=
  method (n)
    let {previous-result := (power-square-property n)}
    conclude (power-square-property (S n))
    pick-any x:N
      (!combine-equations
        (!chain [((x * x) ** S n)]
          --> (((x * x) * ((x * x) ** n))) [Power-right-nonzero]
          --> (((x * x) * (x ** (n + n)))) [previous-result]
          --> (x * x * (x ** (n + n))) [Times-associative]])
        (!chain [(x ** (S (S n + n)))]
          --> (x ** (S (S n + n))) [right-nonzero]
          --> (x ** (S S (n + n))) [left-nonzero]
          --> (x * (x ** (S (n + n)))) [Power-right-nonzero]
          --> (x * x * (x ** (n + n))) [Power-right-nonzero]]))
```

if we apply power-square-step to \( n \), we obtain theorem

\((\text{power-square-property } (S \ n) )\)
Power square theorem

Encapsulating the $n \equiv 0$ case in a separate method, power-square-base:

```plaintext
define power-square-base :=
    method ()
    conclude (power-square-property zero)
    pick-any x:N
    (!chain [((x * x) ** zero) = one [Power-right-zero]]
      = (x ** zero) [Power-right-zero]
      = (x ** (zero + zero)) [right-zero]]
```

The following sequence of calls could be extended to obtain the proof of (power-square-property $n$) for any natural number $n$:

```plaintext
(!power-square-base)
(!power-square-step zero)
(!power-square-step (S zero))
```
The principle of mathematical induction

To prove $\forall n . P(n)$ where $n$ ranges over the natural numbers, it suffices to prove:

1. **Basis case**: $P(0)$.
2. **Induction step**: $\forall n . P(n) \Rightarrow P(n + 1)$.

In the induction step, the antecedent assumption $P(n)$ is called the *induction hypothesis*. 
The **by-induction** proof construct

This principle is embodied in Athena’s by-induction proof construct. We can use it to prove `power-square-theorem` as follows:

```
by-induction power-square-theorem {
    zero => (!power-square-base)
    | (S n) => (!power-square-step n)
}
```

The keyword **by-induction** is followed by the sentence to be derived, which is a goal of the form

\[ \forall n : N . P(n), \]

followed by a number of *clauses*, enclosed in curly braces and separated by `|`, expressing the cases that together are sufficient to complete the proof.
The by-induction proof construct

There are usually two clauses (there can be more):

- one that expresses the basis case, corresponding to $P(0)$, and
- the other expressing the induction step (or “inductive step”), corresponding to

$$\forall n . P(n) \Rightarrow P(n + 1).$$

Each clause is essentially a pair consisting of

- a constructor pattern $\pi_i$ that represents one of the cases of the inductive argument, and
- a corresponding subproof $D_i$.

The arrow keyword $\Rightarrow$ separates $\pi_i$ from $D_i$.

The subproof $D_i$ will be evaluated in the original assumption base augmented with all appropriate inductive hypotheses.
Power square theorem

The induction-step sentence for our example can be written in Athena as follows:

\[(\forall n . \text{power-square-property } n \implies \text{power-square-property } (S \ n))\]

If we were trying to prove this sentence from scratch, without the benefit of by-induction, we could do it with a proof along the following lines:

```
pick-any n:N
assume induction-hypothesis := (power-square-property n)
conclude (power-square-property (S n))
(!power-square-step n)
```

But with by-induction it is not necessary to write this much detail: pick-any, assume, and conclude are implicit.
A schema for inductive proofs

# Start by defining a unary `\"property procedure\"':

define \( P \) \( t \) := ...

# Then use it to define a goal which says that
# every object has this property:

define goal := (forall \( n : \mathbb{N} \) . \( P \) \( n \))

# Finally, prove the goal by induction:

by-induction goal {
    zero  => conclude \( P \) zero
    \( \text{(!basis-case ...)} \)
    | (n as \( (S \) \( m \)) =>
        conclude \( P \) \( n \) \hspace{3cm} \# Here the assumption base contains
        \( \text{(!induction-step ...)} \hspace{1cm} \# the inductive hypothesis} (P \) \( m \)).
    \}
}
A simpler proof by induction

Recall the left-zero property that we defined earlier:

\[
\text{define } \text{left-zero} := (\forall n. \text{zero} + n = n)
\]

Here is a proof using by-induction:

\[
\text{by-induction } \text{left-zero} \{
\begin{array}{l}
\text{zero } \Rightarrow \text{conclude } (\text{zero} + \text{zero} = \text{zero}) \\
\quad (\text{!chain } [(\text{zero} + \text{zero}) \Rightarrow \text{zero } [\text{right-zero}])
\end{array}
\]

\[
| \text{(n as (S m)) } \Rightarrow
\begin{array}{l}
\quad \text{conclude } (\text{zero} + n = n) \\
\quad \text{let } \{\text{induction-hypothesis} := (\text{zero} + m = m)\}
\end{array}
\]

\[
\quad (\text{!chain } [(\text{zero} + \text{S m})
\quad \Rightarrow (\text{S (zero} + m)) \quad [\text{right-nonzero}]
\quad \Rightarrow (\text{S m}) \quad [\text{induction-hypothesis}])
\}
\]