

# Peano arithmetic

Klaus Grue

GRD-2009-11-28.UTC:11:17:40.281380

## Contents

<b>1</b>	<b>Preliminaries</b>	<b>2</b>
1.1	Quantifiers	2
1.2	First and second argument of quantifier	2
1.3	Avoidance	2
1.4	Instantiation	2
1.5	Substitution	3
1.6	Parallel instantiation	3
1.7	Definition	5
1.8	Soundness of definitions	6
1.9	Connectives and function letters	8
<b>2</b>	<b>Theories</b>	<b>8</b>
2.1	Propositional calculus	8
2.2	First order logic	9
2.3	Peano Arithmetic	9
<b>3</b>	<b>Deduction</b>	<b>10</b>
3.1	Auxiliary lemmas	10
3.2	Hooks for the tactic state	15
3.3	Tactic state for hypothetical reasoning	15
3.4	Stating hypotheses	16
3.5	Stating a dummy hypothesis	17
3.6	Handling the first hypothesis	17
3.7	Handling additional hypotheses	18
3.8	Adaption	19
3.9	Ponens	20
3.10	At	21
3.11	Deduction tactic	23
3.12	Test proofs	23
3.13	A proof of $x + y = y + x$	25

# 1 Preliminaries

## 1.1 Quantifiers

The following macro definition allows to write e.g.  $\forall x, y, z: u$  instead of  $\forall x: \forall y: \forall z: u$ .

$[\forall u: v \stackrel{\circ}{=} \lambda x. \text{expand-all} ( x )]$

$[\text{expand-all} ( x ) \stackrel{\bullet}{=}$   
  **let**  $\langle t, s, c \rangle = x$  **in**  
  **let**  $\langle T, u, v \rangle = t$  **in**  
  **let**  $u = \text{stateexpand} ( u, s, c )$  **in**  
  **let**  $v = \text{stateexpand} ( v, s, c )$  **in**  
   $\text{expand-all1} ( t, u, v )]$

$[\text{expand-all1} ( t, u, v ) \stackrel{\bullet}{=}$   
  **if not**  $u \stackrel{r}{=} [x, y]$  **then**  $[_t \text{f.allfunc } \lambda u. v]$  **else**  
   $\text{expand-all1} ( t, u^1, \text{expand-all1} ( t, u^2, v ) )]$

## 1.2 First and second argument of quantifier

$[x^1 \stackrel{\bullet}{=} x^{11}]$

$[x^2 \stackrel{\bullet}{=} x^{12}]$

## 1.3 Avoidance

The side condition  $x \# y$  expresses that  $x$  does not occur free in  $y$ . This side condition is useful to express Mendelsons axioms A5 [1]. We shall also use it in connection with an inference of instantiation defined later.

$[x \# y \stackrel{\bullet}{=} \lambda c. [x] \text{objectavoid} ( c ) [y]]$ . This macro quotes its arguments and pass them on to `objectavoid` which performs the actual computation. During proof checking,  $c$  is instantiated to the cache of the page on which the proof occurs.

## 1.4 Instantiation

The side condition  $a \sim \langle b | x := t \rangle$  expresses that  $a$  is equal to the result of replacing  $x$  by  $t$  in  $b$ . This side condition is useful for expressing the axiom of induction in which one has to replace the induction variable  $x$  by 0 and  $x'$ . The side condition requires  $x$  to be an object variable. The side condition is also useful for expressing axiom A4 (instantiation).

$[a \sim \langle b | x := t \rangle \stackrel{\bullet}{=} \lambda c. \text{sub0} ( [a], [b], [x], [t], c )]$ . This macro quotes its arguments and pass them on to `sub0` which performs the actual computation. During proof checking,  $c$  is instantiated to the cache of the page on which the proof occurs.

[sub0 (  $a$  ,  $b$  ,  $x$  ,  $t$  ,  $c$  )  $\stackrel{\bullet}{=} x$  objectvar (  $c$  ) **and** sub1 (  $a$  ,  $b$  ,  $x$  ,  $t$  ,  $c$  )].  
 Check that  $x$  is an object variable and invoke sub1.

[sub1 (  $a$  ,  $b$  ,  $x$  ,  $t$  ,  $c$  )  $\stackrel{\bullet}{=} \mathbf{if} \ b \stackrel{r}{=} [\forall u: v] \ \mathbf{and} \ b^1 \stackrel{t}{=} x \ \mathbf{then} \ a \stackrel{t}{=} b \ \mathbf{else} \ \mathbf{if} \ b \stackrel{t}{=} x \ \mathbf{then} \ a \stackrel{t}{=} t \ \mathbf{else} \ a \stackrel{r}{=} b \ \mathbf{and} \ \text{sub}^*( a^t , b^t , x , t , c )]$

[sub\* (  $a$  ,  $b$  ,  $x$  ,  $t$  ,  $c$  )  $\stackrel{\bullet}{=} a \in \mathbf{A} \ \mathbf{or} \ \text{sub1} ( a^h , b^h , x , t , c ) \ \mathbf{and} \ \text{sub}^* ( a^t , b^t , x , t , c )]$

## 1.5 Substitution

The function  $\text{f.subst} ( x , s , c )$  defined in the following performs the substitution  $s$  on  $x$  without renaming.

[f.subst (  $x$  ,  $s$  ,  $c$  )  $\stackrel{\bullet}{=} \mathbf{let} \ d = \text{lookup} ( x , s , \top ) \ \mathbf{in} \ \mathbf{if} \ \mathbf{not} \ d \ \mathbf{then} \ d \ \mathbf{else} \ \mathbf{let} \ d = x \ \text{valuedef} ( c ) \ \mathbf{in} \ \mathbf{if} \ d = 1 \ \mathbf{then} \ x \ \mathbf{else} \ \mathbf{if} \ d \neq 0 \ \mathbf{then} \ x^h :: \text{f.subst}^* ( x^t , s , c ) \ \mathbf{else} \ \langle x^h , x^1 , \text{f.subst} ( x^2 , \text{remove} ( x^1 , s ) , c ) \rangle]$

[f.subst\* (  $x$  ,  $s$  ,  $c$  )  $\stackrel{\bullet}{=} \mathbf{if} \ x \in \mathbf{A} \ \mathbf{then} \ \top \ \mathbf{else} \ \text{f.subst} ( x^h , s , c ) :: \text{f.subst}^* ( x^t , s , c )]$

## 1.6 Parallel instantiation

[inst (  $x$  ,  $y$  )  $\stackrel{\bullet}{=} \lambda c. \text{inst0} ( [x] , [y] , c )]$

$\text{inst} ( x , y )$  is suited as a side condition. The side condition is true if  $y$  is an instantiation of  $x$ . Multiple instantiation is legal and is done in parallel. Only universal quantifiers which occur in the root can be instantiated. Universal quantifiers which occur in the the root of  $x$  but which are not instantiated because they also appear in the root of  $y$  may instead be alpha-converted.

[inst0 (  $x$  ,  $y$  ,  $c$  )  $\stackrel{\bullet}{=} \mathbf{not} \ \text{inst1} ( x , y , c )^{\circ h}$ ]

Check that  $y$  is an instance of  $x$ . Return true if it is and false otherwise.

[inst1 (  $x$  ,  $y$  ,  $c$  )  $\stackrel{\bullet}{=} \mathbf{let} \ x :: u = \text{inst-strip} ( x , \top , c ) \ \mathbf{in} \ \mathbf{let} \ y :: v = \text{inst-strip} ( y , \top , c ) \ \mathbf{in}$

**let**  $s = \text{inst-zip} ( u , v )$  **in**  
 $\text{inst2} ( x , y , s , \top , c )]$

Remove universal quantifiers from the root of  $x$  and  $y$ . Accumulate the bound variables in reverse order in  $u$  and  $v$ . Then zip  $u$  and  $v$  in a way which allows  $u$  to be longer than  $v$ . In that way, variables which occur both in  $u$  and  $v$  are effectively alpha-renamed.

$[\text{inst-strip} ( x , v , c ) \stackrel{\bullet}{=} \text{if not } x \text{ metadef} ( c ) \text{ then } x :: v \text{ else}$   
 $\text{if not } x \text{ valuedef} ( c ) \text{ then } x :: v \text{ else}$   
 $\text{if } x \text{ mathdef} ( c ) \neq \text{all then } x :: v \text{ else}$   
 $\text{if } x^t \text{ then } \bullet \text{ else}$   
 $\text{let } x = x^1 \text{ in}$   
 $\text{if not } x \text{ metadef} ( c ) \text{ then } \bullet \text{ else}$   
 $\text{if } x \text{ valuedef} ( c ) \neq 0 \text{ then } \bullet \text{ else}$   
 $\text{inst-strip} ( x^2 , x^1 :: v , c )]$

Remove universal quantifiers from the root of  $x$ . Accumulate the bound variables in  $v$ .

$[\text{inst-zip} ( x , y ) \stackrel{\bullet}{=} \text{if } x \text{ then if } y \text{ then } \top \text{ else } \bullet \text{ else}$   
 $(x^h :: y^h) :: \text{inst-zip} ( x^t , y^t )]$

Zip  $x$  and  $y$  where  $x$  is allowed to be longer than  $y$ .

$[\text{inst2} ( x , y , s , b , c ) \stackrel{\bullet}{=} \text{let } d = \text{lookup} ( x , s , F ) \text{ in}$   
 $\text{if } d \text{ then } (x :: y) :: s \text{ else}$   
 $\text{if } d \in \mathbf{P} \text{ then if inst3} ( d , b , c ) \text{ then } s \text{ else } \bullet \text{ else}$   
 $\text{if not } x \stackrel{r}{=} y \text{ then } \bullet \text{ else}$   
 $\text{if not } x \text{ metadef} ( c ) \text{ then } \bullet \text{ else}$   
 $\text{let } d = x \text{ valuedef} ( c ) \text{ in}$   
 $\text{if } d = 1 \text{ then if } x^1 = y^1 \text{ then } s \text{ else } \bullet \text{ else}$   
 $\text{if } d \neq 0 \text{ then inst2*} ( x^t , y^t , s , b , c ) \text{ else}$   
 $\text{let } v = x^1 \text{ in}$   
 $\text{if not } v = y^1 \text{ then } \bullet \text{ else}$   
 $\text{if not } x \text{ objectvar} ( c ) \text{ then } \bullet \text{ else}$   
 $\text{let } s = \text{remove} ( v , s ) \text{ in}$   
 $\text{inst2} ( x^2 , y^2 , s , b \text{ set+ } v , c )]$

Check that  $y$  is the result of performing the substitution  $s$  on  $x$ . The set  $b$  contains all locally bound variables. The function enriches  $s$  whenever it meets a variable for which it is not yet known how the variable is instantiated. The function returns the enriched  $s$  or signals an error.

$[\text{inst2*} ( x , y , s , b , c ) \stackrel{\bullet}{=} \text{if } x \text{ then } s \text{ else}$

$\text{inst2}^* ( x^t , y^t , \text{inst2} ( x^h , y^h , s , b , c ) , b , c )]$

Elementwise application of  $\text{inst2} ( x , y , s , b , c )$  to lists  $x$  and  $y$ .

$[\text{inst3} ( x , b , c ) \doteq$   
**if not**  $x$  **metadef**  $( c )$  **then F else**  
**if not**  $x$  **mathdef**  $( c )$  **then**  $\text{inst3}^* ( x^t , b , c )$  **else**  
**let**  $d = x$  **valuedef**  $( c )$  **in**  
**if**  $d$  **then not**  $x \in b$  **else**  
**if**  $d = 1$  **then T else**  
**if**  $d \neq 0$  **then**  $\text{inst3}^* ( x^t , b , c )$  **else**  
 $\text{inst3} ( x^2 , b \text{ set- } x^1 , c )]$

Check that  $x$  is suited for insertion, i.e. that the free variable of  $x$  do not occur in  $b$ .

$[\text{inst3}^* ( x , b , c ) \doteq$   
**if**  $x$  **then T else**  $\text{inst3} ( x^h , b , c )$  **and**  $\text{inst3}^* ( x^t , b , c )]$   
 Elementwise application of  $\text{inst3} ( x , b , c )$  to the list  $x$ .

## 1.7 Definition

The side condition  $\text{def} ( a , b )$  allows to formulate a rule of definition which allows to replace definiens by definiendum and vice versa.

$[\text{def} ( a , b ) \doteq \lambda c. \text{def0} ( [a] , [b] , c )]$ .

This macro quotes its arguments and pass them on to  $\text{def0}$  which performs the actual computation. During proof checking,  $c$  is instantiated to the cache of the page on which the proof occurs.

$[\text{def0} ( a , b , c ) \doteq \text{def1} ( a , b , c ) \text{ or } \text{def2} ( a , b , c ) \text{ or } \text{def2} ( b , a , c )]$ .

Test that  $a$  and  $b$  are equal modulo application of definitions.

$[\text{def0}^* ( a , b , c ) \doteq a \in \mathbf{A} \text{ or } \text{def0} ( a^h , b^h , c ) \text{ and } \text{def0}^* ( a^t , b^t , c )]$

Coordinatewise application of  $\text{def0}$  to the lists  $a$  and  $b$ .

$[\text{def1} ( a , b , c ) \doteq$   
**let**  $v = a$  **metadef**  $( c )$  **in**  
**if**  $v = \text{var}$  **then**  $a \stackrel{t}{=} b$  **else**  
**if not**  $v$  **then F else**  
**let**  $v = a$  **valuedef**  $( c )$  **in**  
**if**  $v = 0$  **then**  $a^1 \stackrel{t}{=} b^1$  **and**  $\text{def0} ( a^2 , b^2 , c )$  **else**  
**if**  $v = 1$  **then**  $a^1 = b^1$  **else**  
 $a \stackrel{t}{=} b$  **and**  $\text{def0}^* ( a^t , b^t , c )]$ .

$\text{def1}$  is true if  $a$  and  $b$  are equal modulo definitions applied to sub-terms of  $a$  and  $b$ .

[def2 (  $a$  ,  $b$  ,  $c$  )  $\stackrel{\bullet}{\doteq}$   
 $a$  metadef (  $c$  ) =  $\top$  **and**  $a$  valuedef (  $c$  ) =  $\top$  **and**  
**let**  $v = a$  def (  $c$  , math ) **in**  
**if**  $v$  **or**  $v^{3r} = 0$  **or not** checked-def (  $a^r$  ,  $c$  ) **then**  $F$  **else**  
**let**  $s = \text{zip}$  (  $v^{2t}$  ,  $a^t$  ) **in**  
def3 (  $v^3$  ,  $b$  ,  $s$  ,  $c$  )]

Test that  $b$  is equal to  $a$  according to the definition of  $a$ .

[def3 (  $a$  ,  $b$  ,  $s$  ,  $c$  )  $\stackrel{\bullet}{\doteq}$   
**if not**  $a$  metadef (  $c$  ) **then**  $F$  **else**  
**let**  $v = a$  valuedef (  $c$  ) **in**  
**if**  $v = 0$  **then**  
 $b$  valuedef (  $c$  ) = 0 **and** def3 (  $a^2$  ,  $b^2$  , (  $a^1 :: b^1$  ) ::  $s$  ,  $c$  ) **else**  
**if**  $v = 1$  **then**  $F$  **else**  
**let**  $w = a$  mathdef (  $c$  ) **in**  
**if**  $v$  **and** (  $w$  **or**  $w^r = 0$  ) **then**  $b \stackrel{t}{\doteq} \text{lookup}$  (  $a$  ,  $s$  ,  $\top$  ) **else**  
 $a \stackrel{r}{\doteq} b$  **and** def3\* (  $a^t$  ,  $b^t$  ,  $s$  ,  $c$  )].

True if the right hand side  $a$  is identical to  $b$  after instantiating  $a$  as specified by the association list  $s$ .

[def3\* (  $a$  ,  $b$  ,  $s$  ,  $c$  )  $\stackrel{\bullet}{\doteq}$   
 $a \in \mathbf{A}$  **or** def3 (  $a^h$  ,  $b^h$  ,  $s$  ,  $c$  ) **and** def3\* (  $a^t$  ,  $b^t$  ,  $s$  ,  $c$  )]

Coordinatewise application of def3 to the lists  $a$  and  $b$ .

## 1.8 Soundness of definitions

[**Verifier:**test1  $\wedge_c$  defcheck  $\wedge_c$  proofcheck]

When using the present page as bed page, all tests are executed (test1), all proofs are checked (proofcheck), and the soundness of all definitions are checked (defcheck).

[defcheck  $\doteq \lambda c.$ defcheck1 (  $c$  )]

The defcheck conjunct passes the cache  $c$  on to defcheck1 (  $c$  ).

[checked-def (  $r$  ,  $c$  )  $\stackrel{\bullet}{\doteq}$   
**let**  $x = c[r][\text{codex}][r][0][0][\text{claim}]^3$  **in**  
claiming ( [defcheck] ,  $x$  )]

True if the soundness of definitions on the page with reference  $r$  has been checked.

[defcheck1 (  $c$  )  $\stackrel{\bullet}{\doteq}$   
**let**  $r = c[0]$  **in**  
**let**  $a = c[r][\text{codex}][r]$  **in**  
**let**  $e :: v = \text{defcheck2}$  (  $a$  ,  $c$  ) $^\circ$  **in**  
**if**  $v \neq \top$  **then**  $v$  **else**

**if not  $e$  then  $\top$  else**

[‘In definition soundness checker: unprocessed exception’]]

Look up the array  $a$  of all local definitions on the page being checked.

Pass it to defcheck2 (  $a$  ,  $c$  )

[defcheck2 (  $a$  ,  $c$  )  $\stackrel{\bullet}{=}$

**if  $a = \top$  then  $\top$  else**

**if not  $a^h \in \mathbf{Z}$  then**

defcheck2 (  $a^h$  ,  $c$  ) **and** defcheck2 (  $a^t$  ,  $c$  ) **else**

**let  $v = a^t[0]$  in**

**if  $v$  or  $v^{3r} = 0$  then  $\top$  else**

**if valid-def (  $v^3$  ,  $v^{2t}$  ,  $\langle v^2 \rangle$  ,  $c$  ) then  $\top$  else**

error (  $v^2$  , diag ( ‘Definition soundness check of’ ) form (  $v^2$  ) diag ( ‘failed.’ ) diag ( ‘Error could be in any, transitively used definition’ ) diag ( ‘or could be that the definition is circular.’ ) end diagnose )]

Find all definitions  $v$ . If  $v$  is  $\top$  then the corresponding construct is undefined which is ok. If  $v^{3r} = 0$  then the right hand side is a string indicating that this is a predefined construct of some sort which is ok. Otherwise, call valid-def. Calling valid-def for each definition is not optimal since non-circularity is checked every time.

[valid-def (  $a$  ,  $b$  ,  $s$  ,  $c$  )  $\stackrel{\bullet}{=}$

**if not  $a$  metadef (  $c$  ) then  $\mathbf{F}$  else**

**let  $v = a$  valuedef (  $c$  ) in**

**if  $v = 0$  then valid-def (  $a^2$  ,  $a^1 :: b$  ,  $s$  ,  $c$  ) else**

**if  $v = 1$  then  $\mathbf{F}$  else**

**if not  $v$  then valid-def\* (  $a^t$  ,  $b$  ,  $s$  ,  $c$  ) else**

**let  $v = a$  def (  $c$  , math ) in**

**if  $v$  then  $a \in b$  else**

**if not valid-def\* (  $a^t$  ,  $b$  ,  $s$  ,  $c$  ) then  $\mathbf{F}$  else**

**if  $v^{3r} = 0$  then  $\top$  else**

**if  $v^{2r} \neq c[0]$  then claiming (  $v^{2r}$  ,  $c$  ) else**

**if  $v^2 \in s$  then  $\mathbf{F}$  else**

valid-def (  $v^3$  ,  $v^{2t}$  ,  $v^2 :: s$  ,  $c$  )].

valid-def (  $a$  ,  $b$  ,  $s$  ,  $c$  ) tests that  $a$  is a valid right hand side of a definition.  $b$  is the list of bound variables (parameters plus variables bound by lambdas).  $s$  is a stack of left hand sides used for spotting circular definitions.  $c$  is the cache.

[valid-def\* (  $a$  ,  $b$  ,  $s$  ,  $c$  )  $\stackrel{\bullet}{=}$

$a \in \mathbf{A}$  or valid-def (  $a^h$  ,  $b$  ,  $s$  ,  $c$  ) **and** valid-def\* (  $a^t$  ,  $b$  ,  $s$  ,  $c$  )].

Conjunction of valid-def applied to all elements of the list  $a$ .

## 1.9 Connectives and function letters

The sidecondition above and the inference rule of definition stated later allows to define new concepts from previously defined concepts. Definitions whose right hand side is a string introduce a new construct.

In Propositional calculus we take implication  $x \Rightarrow y$  and falsehood  $\mathbf{F}$  as elementary and then define a number of logical connectives on basis of that:

$$[x \Rightarrow y \stackrel{\text{math}}{=} \text{'imply'}].$$

$$[\mathbf{F} \stackrel{\text{math}}{=} \text{'ff'}].$$

$$[\neg x \stackrel{\text{math}}{=} x \Rightarrow \mathbf{F}]$$

$$[\mathbf{T} \stackrel{\text{math}}{=} \mathbf{F} \Rightarrow \mathbf{F}]$$

$$[x \vee y \stackrel{\text{math}}{=} \neg x \Rightarrow y]$$

$$[x \wedge y \stackrel{\text{math}}{=} \neg(x \Rightarrow \neg y)]$$

$$[x \Leftarrow y \stackrel{\text{math}}{=} y \Rightarrow x]$$

$$[x \Leftrightarrow y \stackrel{\text{math}}{=} (x \Rightarrow y) \wedge (y \Rightarrow x)]$$

In first order logic we take the universal quantifier as elementary. We have defined  $\forall x: y$  to macro expand to `f.allfunc  $\lambda x.y$`  so it is the unary `f.allfunc`  $x$  construct which represents the universal quantifier.

$$[\text{f.allfunc } x \stackrel{\text{math}}{=} \text{'all'}].$$

In Peano arithmetic we use the function letters  $0$ ,  $x + y$ ,  $x \cdot y$ , and  $x'$  and the relation sign  $x = y$ . All of these except the successor function  $x'$  are value defined on the [base](#) page. We also give a value definition of the successor function:

$$[x' \stackrel{\bullet}{=} x + 1]$$

## 2 Theories

### 2.1 Propositional calculus

The following rules define the propositional calculus Prop. The axioms A1, A2, and A3 and the inference rule MP are taken from Mendelson [1].

The Def rule is the inference of definition. Due to the inference of definition, we may use e.g.  $x \wedge y$ ,  $x \vee y$ , and  $x \Leftrightarrow y$  even though those connectives are not mentioned in the axioms.

Even  $\neg x$  is defined:  $\neg x$  is defined to mean  $x \Rightarrow \mathbf{F}$  where  $\mathbf{F}$  is falsehood.

Apart from Def and the definition of  $\neg x$ , Prop below is defined as in [1].

**Axiom A1:**  $\prod x, y: x \Rightarrow y \Rightarrow x \square$

**Axiom A2:**  $\prod x, y, z: (x \Rightarrow y \Rightarrow z) \Rightarrow (x \Rightarrow y) \Rightarrow x \Rightarrow z \square$

**Axiom A3:**  $\prod x, y: (\neg y \Rightarrow \neg x) \Rightarrow (\neg y \Rightarrow x) \Rightarrow y \square$

**Rule MP:**  $\prod x, y: x \Rightarrow y \vdash x \vdash y \square$

**Rule Def:**  $\prod x, y: \text{def } (x, y) \Vdash x \vdash y \square$

**Theory Prop:**  $A1 \oplus A2 \oplus A3 \oplus \text{MP} \oplus \text{Def} \square$

## 2.2 First order logic

First order logic FOL is Prop extended with A4, AP4, A5, and Gen. Gen is generalization, A4 is the opposite, and A5 is needed for the theorem of deduction.

AP4 is a parallel version of A4 which allows to instantiate several quantifiers at the same time. As an example,  $\forall x, y: x + y = y + x \Rightarrow y + x = x + y$  is correct according to AP4 because  $y + x = x + y$  arises from  $x + y = y + x$  by simultaneous replacement of  $x$  with  $y$  and  $y$  with  $x$ . The verification time of the present paper has been reduce by a factor of five by having parallel object instantiation (AP4) and by having parallel meta instantiation (parallel  $x @ y$ ). Both A4 and AP4 has been included in FOL even though any one of them is sufficient.

Apart from AP4 and the definition of  $\neg x$ , FOL below is defined as in [1].

**Rule A4:**  $\prod t, x, a, b: b \sim \langle a \mid x := t \rangle \Vdash \forall x: a \Rightarrow b \square$

**Rule AP4:**  $\prod a, b: \text{inst } (a, b) \Vdash a \Rightarrow b \square$

**Rule A5:**  $\prod x, a, b: x \# a \Vdash \forall x: (a \Rightarrow b) \Rightarrow a \Rightarrow \forall x: b \square$

**Rule Gen:**  $\prod u, x: x \vdash \forall u: x \square$

**Theory FOL:**  $\text{Prop} \oplus A4 \oplus \text{AP4} \oplus A5 \oplus \text{Gen} \square$

## 2.3 Peano Arithmetic

Peano arithmetic PA is FOL extended with S1 to S9 taken from [1].

**Rule S1:**  $\forall x, y, z: (x = y \Rightarrow x = z \Rightarrow y = z) \square$

**Rule S2:**  $\forall x, y: (x = y \Rightarrow x' = y') \square$

**Rule S3:**  $\forall x: \neg 0 = x' \square$

**Rule S4:**  $\forall x: x' = y' \Rightarrow x = y \quad \square$

**Rule S5:**  $\forall x: x + 0 = x \quad \square$

**Rule S6:**  $\forall x, y: x + y' = (x + y)' \quad \square$

**Rule S7:**  $\forall x: x \cdot 0 = 0 \quad \square$

**Rule S8:**  $\forall x, y: x \cdot y' = x \cdot y + x \quad \square$

**Rule S9:**  $\Pi x, a, z, i: z \sim \langle a \mid x := 0 \rangle \Vdash i \sim \langle a \mid x := x' \rangle \Vdash z \Rightarrow \forall x: (a \Rightarrow i) \Rightarrow a \quad \square$

**Theory PA:**  $\text{FOL} \oplus \text{S1} \oplus \text{S2} \oplus \text{S3} \oplus \text{S4} \oplus \text{S5} \oplus \text{S6} \oplus \text{S7} \oplus \text{S8} \oplus \text{S9} \quad \square$

## 3 Deduction

### 3.1 Auxiliary lemmas

We now prove a number of auxiliary lemmas which are needed by the deduction tactic. Later, we define that Prop, FOL, and PA proofs are tactic expanded by the deduction tactic. We cannot prove the lemmas below using the deduction tactic since the deduction tactic does not work before the lemmas below are proved. For that reason, we disable the deduction tactic in the proofs below by assuming Prop and FOL as premises rather than stating them in the header of the proof.

Lemma Mend1.8 below is Lemma 1.8 in Mendelson: Introduction to Mathematical Logic [1]. In general, the numbering used in [1] has been kept. Some of the proofs also come from [1]. The present text deviates slightly from [1] in that the present text takes implication  $x \Rightarrow y$  and falsehood **F** as primitive whereas [1] takes implication  $x \Rightarrow y$  and negation  $\neg x$  as primitive. All proofs in [1] carry over to the present framework without change since the axioms and inference rules are taken from [1]. But the present text can take advantage of an additional feature, namely that  $\neg x$  is defined to denote  $x \Rightarrow \mathbf{F}$ .

Prop **lemma** Mend1.8:  $\Pi x: x \Rightarrow x \quad \square$

**Proof of** Mend1.8:

L01:	Premise $\gg$	Prop	;
L02:	Arbitrary $\gg$	$x$	;
L03:	A2 $\gg$	$(x \Rightarrow (y \Rightarrow x) \Rightarrow x) \Rightarrow$ $(x \Rightarrow y \Rightarrow x) \Rightarrow x \Rightarrow x$	;
L04:	A1 $\gg$	$x \Rightarrow (y \Rightarrow x) \Rightarrow x$	;
L05:	MP $\triangleright$ L03 $\triangleright$ L04 $\gg$	$(x \Rightarrow y \Rightarrow x) \Rightarrow x \Rightarrow x$	;
L06:	A1 $\gg$	$x \Rightarrow y \Rightarrow x$	;
L07:	MP $\triangleright$ L05 $\triangleright$ L06 $\gg$	$x \Rightarrow x$	$\square$

Prop **lemma** A1':  $\Pi h, a: a \vdash h \Rightarrow a \quad \square$

**Proof of A1':**

L01:	Premise $\gg$	Prop	;
L02:	Arbitrary $\gg$	$h, a$	;
L03:	Premise $\gg$	$a$	;
L04:	MP $\triangleright$ A1 $\triangleright$ L03 $\gg$	$h \Rightarrow a$	□

Prop lemma A2':  $\Pi h, a, b: h \Rightarrow a \Rightarrow b \vdash h \Rightarrow a \vdash h \Rightarrow b$  □

**Proof of A2':**

L01:	Premise $\gg$	Prop	;
L02:	Arbitrary $\gg$	$h, a, b$	;
L03:	Premise $\gg$	$h \Rightarrow a \Rightarrow b$	;
L04:	Premise $\gg$	$h \Rightarrow a$	;
L05:	MP $\triangleright$ (MP $\triangleright$ A2 $\triangleright$ L03) $\triangleright$ L04 $\gg$	$h \Rightarrow b$	□

Prop lemma Mend1.47b:  $\Pi a, b, c: a \Rightarrow b \vdash b \Rightarrow c \vdash a \Rightarrow c$  □

**Proof of Mend1.47b:**

L01:	Premise $\gg$	Prop	;
L02:	Arbitrary $\gg$	$a, b, c$	;
L03:	Premise $\gg$	$a \Rightarrow b$	;
L04:	Premise $\gg$	$b \Rightarrow c$	;
L05:	A1' $\triangleright$ L04 $\gg$	$a \Rightarrow b \Rightarrow c$	;
L06:	A2' $\triangleright$ L05 $\triangleright$ L03 $\gg$	$a \Rightarrow c$	□

Prop lemma Mend1.47c:  $\Pi a, b, c: a \Rightarrow b \Rightarrow c \vdash b \Rightarrow a \Rightarrow c$  □

**Proof of Mend1.47c:**

L01:	Premise $\gg$	Prop	;
L02:	Arbitrary $\gg$	$a, b, c$	;
L03:	Premise $\gg$	$a \Rightarrow b \Rightarrow c$	;
L04:	MP $\triangleright$ A2 $\triangleright$ L03 $\gg$	$(a \Rightarrow b) \Rightarrow a \Rightarrow c$	;
L05:	A1' $\triangleright$ L04 $\gg$	$b \Rightarrow (a \Rightarrow b) \Rightarrow a \Rightarrow c$	;
L06:	MP $\triangleright$ A2 $\triangleright$ L05 $\gg$	$(b \Rightarrow a \Rightarrow b) \Rightarrow b \Rightarrow a \Rightarrow c$	;
L07:	MP $\triangleright$ L06 $\triangleright$ A1 $\gg$	$b \Rightarrow a \Rightarrow c$	□

Prop lemma Mend1.47d:  $\Pi a, b: (\neg a \Rightarrow \neg b) \Rightarrow b \Rightarrow a$  □

**Proof of Mend1.47d:**

L01:	Premise $\gg$	Prop	;
L02:	Arbitrary $\gg$	$a, b$	;
L03:	A3 $\gg$	$(\neg a \Rightarrow \neg b) \Rightarrow (\neg a \Rightarrow b) \Rightarrow a$	;
L04:	Mend1.47c $\triangleright$ L03 $\gg$	$(\neg a \Rightarrow b) \Rightarrow (\neg a \Rightarrow \neg b) \Rightarrow a$	;
L05:	A1 $\gg$	$b \Rightarrow \neg a \Rightarrow b$	;
L06:	Mend1.47b $\triangleright$ L05 $\triangleright$ L04 $\gg$	$b \Rightarrow (\neg a \Rightarrow \neg b) \Rightarrow a$	;
L07:	Mend1.47c $\triangleright$ L06 $\gg$	$(\neg a \Rightarrow \neg b) \Rightarrow b \Rightarrow a$	□

Prop lemma Mend1.11b:  $\Pi a: a \Rightarrow \neg\neg a$  □

**Proof of Mend1.11b:**

L01:	Premise $\gg$	Prop	;
L02:	Arbitrary $\gg$	$a$	;
L03:	Mend1.8 $\gg$	$(a \Rightarrow \mathbf{F}) \Rightarrow a \Rightarrow \mathbf{F}$	;
L04:	Mend1.47c $\triangleright$ L03 $\gg$	$a \Rightarrow (a \Rightarrow \mathbf{F}) \Rightarrow \mathbf{F}$	;
L05:	Def $\triangleright$ L04 $\gg$	$a \Rightarrow \neg a \Rightarrow \mathbf{F}$	;
L06:	Def $\triangleright$ L05 $\gg$	$a \Rightarrow \neg\neg a$	□

Prop **lemma** Mend1.48d:  $\prod h, x: h \wedge x \Rightarrow h$  □

**Proof of Mend1.48d:**

L01:	Premise $\gg$	Prop	;
L02:	Arbitrary $\gg$	$h, x$	;
L03:	Mend1.8 $\gg$	$\mathbf{F} \Rightarrow \mathbf{F}$	;
L04:	Def $\triangleright$ L03 $\gg$	$\neg\mathbf{F}$	;
L05:	A1' $\triangleright$ L04 $\gg$	$\neg\neg x \Rightarrow \neg\mathbf{F}$	;
L06:	MP $\triangleright$ Mend1.47d $\triangleright$ L05 $\gg$	$\mathbf{F} \Rightarrow \neg x$	;
L07:	A1' $\triangleright$ L06 $\gg$	$h \Rightarrow \mathbf{F} \Rightarrow \neg x$	;
L08:	MP $\triangleright$ A2 $\triangleright$ L07 $\gg$	$(h \Rightarrow \mathbf{F}) \Rightarrow h \Rightarrow \neg x$	;
L09:	Def $\triangleright$ L08 $\gg$	$\neg h \Rightarrow h \Rightarrow \neg x$	;
L10:	Mend1.11b $\gg$	$(h \Rightarrow \neg x) \Rightarrow \neg\neg(h \Rightarrow \neg x)$	;
L11:	Mend1.47b $\triangleright$ L09 $\triangleright$ L10 $\gg$	$\neg h \Rightarrow \neg\neg(h \Rightarrow \neg x)$	;
L12:	MP $\triangleright$ Mend1.47d $\triangleright$ L11 $\gg$	$\neg(h \Rightarrow \neg x) \Rightarrow h$	;
L13:	Def $\triangleright$ L12 $\gg$	$h \wedge x \Rightarrow h$	□

Prop **lemma** Mend1.48e:  $\prod h, x: h \wedge x \Rightarrow x$  □

**Proof of Mend1.48e:**

L01:	Premise $\gg$	Prop	;
L02:	Arbitrary $\gg$	$h, x$	;
L03:	A1 $\gg$	$\neg x \Rightarrow h \Rightarrow \neg x$	;
L04:	Mend1.11b $\gg$	$(h \Rightarrow \neg x) \Rightarrow \neg\neg(h \Rightarrow \neg x)$	;
L05:	Mend1.47b $\triangleright$ L03 $\triangleright$ L04 $\gg$	$\neg x \Rightarrow \neg\neg(h \Rightarrow \neg x)$	;
L06:	MP $\triangleright$ Mend1.47d $\triangleright$ L05 $\gg$	$\neg(h \Rightarrow \neg x) \Rightarrow x$	;
L07:	Def $\triangleright$ L06 $\gg$	$h \wedge x \Rightarrow x$	□

Prop **lemma** Lnexthyp:  $\prod h, x, y: h \Rightarrow y \vdash h \wedge x \Rightarrow y$  □

**Proof of Lnexthyp:**

L01:	Premise $\gg$	Prop	;
L02:	Arbitrary $\gg$	$h, x, y$	;
L03:	Premise $\gg$	$h \Rightarrow y$	;
L04:	Mend1.48d $\gg$	$h \wedge x \Rightarrow h$	;
L05:	Mend1.47b $\triangleright$ L04 $\triangleright$ L03 $\gg$	$h \wedge x \Rightarrow y$	□

Prop **lemma** Lcurry:  $\prod h, x, y: h \wedge x \Rightarrow y \vdash h \Rightarrow x \Rightarrow y$  □

**Proof of Lcurry:**

L01:	Premise $\gg$	Prop	;
------	---------------	------	---

L02:	Arbitrary $\gg$	$h, x, y$	;
L03:	Premise $\gg$	$h \wedge x \Rightarrow y$	;
L04:	Def $\triangleright$ L03 $\gg$	$\neg(h \Rightarrow \neg x) \Rightarrow y$	;
L05:	Mend1.11b $\gg$	$y \Rightarrow \neg\neg y$	;
L06:	Mend1.47b $\triangleright$ L04 $\triangleright$ L05 $\gg$	$\neg(h \Rightarrow \neg x) \Rightarrow \neg\neg y$	;
L07:	MP $\triangleright$ Mend1.47d $\triangleright$ L06 $\gg$	$\neg y \Rightarrow h \Rightarrow \neg x$	;
L08:	Mend1.47c $\triangleright$ L07 $\gg$	$h \Rightarrow \neg y \Rightarrow \neg x$	;
L09:	Mend1.47d $\gg$	$(\neg y \Rightarrow \neg x) \Rightarrow x \Rightarrow y$	;
L10:	Mend1.47b $\triangleright$ L08 $\triangleright$ L09 $\gg$	$h \Rightarrow x \Rightarrow y$	□

Prop **lemma** LTruth: **T** □

**Proof of LTruth:**

L01:	Premise $\gg$	Prop	;
L02:	Mend1.8 $\gg$	<b>F</b> $\Rightarrow$ <b>F</b>	;
L03:	Def $\triangleright$ L02 $\gg$	<b>T</b>	□

Prop **lemma** LElimHyp:  $\Pi x: \mathbf{T} \Rightarrow x \vdash x$  □

**Proof of LElimHyp:**

L01:	Premise $\gg$	Prop	;
L02:	Arbitrary $\gg$	$x$	;
L03:	Premise $\gg$	<b>T</b> $\Rightarrow x$	;
L04:	LTruth $\gg$	<b>T</b>	;
L05:	MP $\triangleright$ L03 $\triangleright$ L04 $\gg$	$x$	□

FOL **lemma** LA4':  $\Pi h, t, x, a, b: b \sim \langle a \mid x := t \rangle \Vdash h \Rightarrow \forall x: a \vdash h \Rightarrow b$  □

**Proof of LA4':**

L01:	Premise $\gg$	FOL	;
L02:	Arbitrary $\gg$	$h, t, x, a, b$	;
L03:	Condition $\gg$	$b \sim \langle a \mid x := t \rangle$	;
L04:	Premise $\gg$	$h \Rightarrow \forall x: a$	;
L05:	A4 $\triangleright$ L03 $\gg$	$\forall x: a \Rightarrow b$	;
L06:	Mend1.47b $\triangleright$ L04 $\triangleright$ L05 $\gg$	$h \Rightarrow b$	□

FOL **lemma** LAP4':  $\Pi h, a, b: \text{inst} ( a , b ) \Vdash h \Rightarrow a \vdash h \Rightarrow b$  □

**Proof of LAP4':**

L01:	Premise $\gg$	FOL	;
L02:	Arbitrary $\gg$	$h, a, b$	;
L03:	Condition $\gg$	$\text{inst} ( a , b )$	;
L04:	Premise $\gg$	$h \Rightarrow a$	;
L05:	AP4 $\triangleright$ L03 $\gg$	$a \Rightarrow b$	;
L06:	Mend1.47b $\triangleright$ L04 $\triangleright$ L05 $\gg$	$h \Rightarrow b$	□

Prop **lemma** MP':  $\Pi h, a, b: h \Rightarrow a \Rightarrow b \vdash h \Rightarrow a \vdash h \Rightarrow b$  □

**Proof of MP':**

L01:	Premise $\gg$	Prop	;
L02:	Arbitrary $\gg$	$h, a, b$	;
L03:	Premise $\gg$	$h \Rightarrow a \Rightarrow b$	;
L04:	Premise $\gg$	$h \Rightarrow a$	;
L05:	A2 $\gg$	$(h \Rightarrow a \Rightarrow b) \Rightarrow (h \Rightarrow a) \Rightarrow h \Rightarrow b$	;
L06:	MP $\triangleright$ L05 $\triangleright$ L03 $\gg$	$(h \Rightarrow a) \Rightarrow h \Rightarrow b$	;
L07:	MP $\triangleright$ L06 $\triangleright$ L04 $\gg$	$h \Rightarrow b$	$\square$

FOL lemma Gen1:  $\Pi h, x, a: x \# h \Vdash h \Rightarrow a \vdash h \Rightarrow \forall x: a \square$

**Proof of Gen1:**

L01:	Premise $\gg$	FOL	;
L02:	Arbitrary $\gg$	$h, x, a$	;
L03:	Condition $\gg$	$x \# h$	;
L04:	Premise $\gg$	$h \Rightarrow a$	;
L05:	Gen $\triangleright$ L04 $\gg$	$\forall x: (h \Rightarrow a)$	;
L06:	A5 $\triangleright$ L03 $\gg$	$\forall x: (h \Rightarrow a) \Rightarrow (h \Rightarrow \forall x: a)$	;
L07:	MP $\triangleright$ L06 $\triangleright$ L05 $\gg$	$h \Rightarrow \forall x: a$	$\square$

FOL lemma Gen2:  $\Pi h, x, y, a: x \# h \Vdash y \# h \Vdash h \Rightarrow a \vdash h \Rightarrow \forall x, y: a \square$

**Proof of Gen2:**

L01:	Premise $\gg$	FOL	;
L02:	Arbitrary $\gg$	$h, x, y, a$	;
L03:	Condition $\gg$	$x \# h$	;
L04:	Condition $\gg$	$y \# h$	;
L05:	Premise $\gg$	$h \Rightarrow a$	;
L06:	Gen1 $\triangleright$ L04 $\triangleright$ L05 $\gg$	$h \Rightarrow \forall y: a$	;
L07:	Gen1 $\triangleright$ L03 $\triangleright$ L06 $\gg$	$h \Rightarrow \forall x, y: a$	$\square$

FOL lemma Gen3:  $\Pi h, x, y, z, a: x \# h \Vdash y \# h \Vdash z \# h \Vdash h \Rightarrow a \vdash h \Rightarrow \forall x, y, z: a \square$

**Proof of Gen3:**

L01:	Premise $\gg$	FOL	;
L02:	Arbitrary $\gg$	$h, x, y, z, a$	;
L03:	Condition $\gg$	$x \# h$	;
L04:	Condition $\gg$	$y \# h$	;
L05:	Condition $\gg$	$z \# h$	;
L06:	Premise $\gg$	$h \Rightarrow a$	;
L07:	Gen2 $\triangleright$ L04 $\triangleright$ L05 $\triangleright$ L06 $\gg$	$h \Rightarrow \forall y, z: a$	;
L08:	Gen1 $\triangleright$ L03 $\triangleright$ L07 $\gg$	$h \Rightarrow \forall x, y, z: a$	$\square$

PA lemma Induction:  $\Pi x, h, a, z, i: z \sim \langle a \mid x := 0 \rangle \Vdash i \sim \langle a \mid x := x' \rangle \Vdash x \# h \Vdash h \Rightarrow z \vdash h \Rightarrow a \Rightarrow i \vdash h \Rightarrow a \square$

**Proof of Induction:**

L01:	Premise $\gg$	PA	;
------	---------------	----	---

L02:	Arbitrary $\gg$	$x, h, a, z, i$	;
L03:	Condition $\gg$	$z \sim \langle a \mid x := 0 \rangle$	;
L04:	Condition $\gg$	$i \sim \langle a \mid x := x' \rangle$	;
L05:	Condition $\gg$	$x \# h$	;
L06:	Premise $\gg$	$h \Rightarrow z$	;
L07:	Premise $\gg$	$h \Rightarrow a \Rightarrow i$	;
L08:	Gen1 $\triangleright$ L05 $\triangleright$ L07 $\gg$	$h \Rightarrow \forall x: (a \Rightarrow i)$	;
L09:	S9 $\triangleright$ L03 $\triangleright$ L04 $\gg$	$z \Rightarrow \forall x: (a \Rightarrow i) \Rightarrow a$	;
L10:	A1' $\triangleright$ L09 $\gg$	$h \Rightarrow z \Rightarrow \forall x: (a \Rightarrow i) \Rightarrow a$	;
L11:	A2' $\triangleright$ L10 $\triangleright$ L06 $\gg$	$h \Rightarrow \forall x: (a \Rightarrow i) \Rightarrow a$	;
L12:	A2' $\triangleright$ L11 $\triangleright$ L08 $\gg$	$h \Rightarrow a$	□

### 3.2 Hooks for the tactic state

[hook-hyp  $\stackrel{\bullet}{=} \text{hyp}$ ]

Hypothesis hook. Contains the “working hypothesis” which is the conjunction of all assumed hypotheses (or  $\top$  if no hypotheses are assumed).

### 3.3 Tactic state for hypothetical reasoning

During hypothetical reasoning, unitac-hyp is installed at the hook-unitac hook of the tactic state. The effect is that occurrences of  $x \gg y$  are changed to  $x \gg \text{hyp} \Rightarrow y$  where hyp is the current hypothesis. References to lemmas and rules are modified likewise. Modification only affects proof lines, lemmas, and rules which conclude on object term. Proof lines, lemmas, and rules which conclude a meta term are unaffected.

[is-meta-term (  $t$  ,  $c$  )  $\stackrel{\bullet}{=} \text{is-meta-term}$ ]

```

let  $d = t$  metadef (  $c$  ) in
if  $d \neq \top$  and  $d \neq \text{var}$  then  $\top$  else
not  $t$  stmt-rhs (  $c$  )

```

True if  $t$  is a meta-term (i.e. no object term) or if  $t$  is statement defined to denote some other term.

[unitac-hyp  $\stackrel{\bullet}{=} \top$  [hook-lemma  $\rightarrow$  [ $\lambda u.$ unitac-lemma-hyp (  $u$  )]] [hook-rule  $\rightarrow$  [ $\lambda u.$ unitac-rule (  $u$  )]] [hook-conclude  $\rightarrow$  [ $\lambda u.$ unitac-conclude-hyp (  $u$  )]]]

Structure containing modified lemma, rule, and conclude tactics.

[unitac-lemma-hyp (  $x$  )  $\stackrel{\bullet}{=} \text{unitac-lemma-hyp}$ ]

```

let (  $t, s, c$  ) =  $x$  in
let (  $\top, T, r$  ) =  $t$  stmt-rhs (  $c$  ) in
if is-meta-term (  $r$  ,  $c$  ) then unitac-lemma-std (  $x$  ) else
let  $h = s$  [hook-hyp] in
let  $a' = \text{unitac-theo}$  (  $T$  ,  $s$  ,  $c$  ) in
let  $a = [t \underline{a'}; \underline{t}^{\text{ID}} \triangleright]$  in

```

```

let  $a = [{}_a \underline{a}; (A1^{ID\triangleright} @ \underline{h} @ \underline{r})^{\triangleright}]$  in
let  $r = [{}_r \underline{h} \Rightarrow \underline{r}]$  in
 $s[\text{hook-arg} \rightarrow a][\text{hook-res} \rightarrow r]$ .

```

Modified lemma tactic. Calls unmodified tactic for meta terms.

```

[unitac-rule-hyp (  $x$  )  $\doteq$ 
let  $\langle t, s, c \rangle = x$  in
let  $r = t$  stmt-rhs (  $c$  ) in
if is-meta-term (  $r, c$  ) then unitac-rule-std (  $x$  ) else
let  $h = s[\text{hook-hyp}]$  in
let  $a = \text{unitac-theo} ( t, s, c )$  in
let  $a = [{}_r \underline{a}^D]$  in
let  $a = [{}_a \underline{a}; (A1^{ID\triangleright} @ \underline{h} @ \underline{r})^{\triangleright}]$  in
let  $r = [{}_r \underline{h} \Rightarrow \underline{r}]$  in
 $s[\text{hook-arg} \rightarrow a][\text{hook-res} \rightarrow r]$ 

```

Modified rule tactic. Calls unmodified tactic for meta terms.

```

[unitac-conclude-hyp (  $x$  )  $\doteq$ 
let  $\langle t, s, c \rangle = x$  in
let  $\langle T, q, r \rangle = t$  in
if is-meta-term (  $r, c$  ) then unitac-conclude-std (  $x$  ) else
let  $h = s[\text{hook-hyp}]$  in
let  $s = \text{unieval} ( q, s, c )$  in
let  $s = \text{f.unitac-adapt} ( r \text{ mathdef } ( c ), s, c )$  in
let  $r = [{}_r \underline{h} \Rightarrow \underline{r}]$  in
let  $u = \text{unify} ( s[\text{hook-res}], r, s[\text{hook-uni}] )$  in
 $s[\text{hook-uni} \rightarrow u][\text{hook-res} \rightarrow r]$ 

```

Modified conclude tactic. Calls unmodified tactic for meta terms.

### 3.4 Stating hypotheses

The following constructs allow to add a hypothesis to the working hypothesis of a proof.

```

[L*:Hypothesis  $\gg *; * \stackrel{\text{locate}}{=} \text{line} :: 1]$ 

```

The definition above ensures that errors in hypothesis lines are properly located (somewhat academic since hypothesis lines cannot be in error).

```

[Ll:Hypothesis  $\gg x; n \doteq \text{Line } l : \text{Hypothesis } \gg x; n]$ 

```

```

[Line  $l : \text{Hypothesis } \gg x; n \stackrel{\text{tactic}}{=} \lambda u. \text{tactic-hyp} ( u )]$ 

```

The tactic definition above adds  $x$  to the working hypothesis and tactic evaluates  $n$ .

```
[tactic-hyp ( u ) ≐
  let ⟨t, s, c⟩ = u in
  let h = s[hook-hyp] in
  if h then tactic-first-hyp ( t , s , c ) else tactic-next-hyp ( h , t ,
  s , c )]
```

When more hypotheses are assumed, the first hypothesis is treated differently from the next ones.

### 3.5 Stating a dummy hypothesis

The following constructs allow to state  $\mathbf{T}$  as a hypothesis and removes  $\mathbf{T}$  again afterwards. The construct has no effect when the hypothesis is non-empty. The constructs below are useful in connection with constructs like  $x \supseteq y$ ,  $x \supseteq$ ,  $x \text{ @ } y$ , and  $x \text{ @}$  which require the hypothesis to be non-empty.

```
[L*: Deduction; *  $\stackrel{\text{locate}}{=}$  line :: 1]
```

```
[tactic-ded ( l , n )  $\stackrel{\text{locate}}{=}$  line :: 2]
```

The definition above ensures that errors in dummy hypothesis lines are properly located (somewhat academic since hypothesis lines cannot be in error).

```
[Ll: Deduction; n ≐ Line l : Deduction ; n]
```

```
[Line l : Deduction ; n  $\stackrel{\text{tactic}}{=}$  λu.tactic-dummyhyp ( u )]
```

```
[tactic-ded ( l , n )  $\stackrel{\text{tactic}}{=}$  λu.tactic-dummyhyp ( u )]
```

The tactic definition above adds  $\mathbf{T}$  to the working hypothesis, tactic evaluates  $n$ , and removes  $\mathbf{T}$  again.

```
[tactic-dummyhyp ( u ) ≐
  let ⟨t, s, c⟩ = u in
  let ⟨T, T, n⟩ = t in
  let h = s[hook-hyp] in
  if not h then taceval ( n , s , c ) else
  let s = tactic-first-hyp ( ⟨T, T, [T], n⟩ , s , c ) in
  let r = s[hook-res]2 in
  let a = s[hook-arg] in
  let a = [a a; (LElimHypID▷ @ r)▷] in
  s[hook-arg→a][hook-res→r2]]
```

### 3.6 Handling the first hypthesis

```
[tactic-first-hyp ( t , s , c ) ≐
  let ⟨⟨r, i⟩ = [x ≫ y] in
  let ⟨T, l, x, n⟩ = t in
```

```

let  $s = s[\text{hook-hyp} \rightarrow x]$  in
let  $s = s[\text{hook-unitac} \rightarrow \text{unitac-hyp}]$  in
let  $s = s[\text{hook-pre} \rightarrow \text{add-first-hyp}^* (x, s[\text{hook-pre}], c)]$  in
let  $x' = [{}_t \underline{x} \Rightarrow \underline{x}]$  in
let  $s = \text{tactic-push} ( \text{hook-pre} , \langle l, x', [x' \underline{x}^I] \rangle , s )$  in
let  $s = \text{taceval} ( n , s , c )$  in
 $s[\text{hook-arg} \rightarrow [{}_t \text{Mend1.8}^{\text{ID}\triangleright} @ \underline{x}; s[\text{hook-arg}]]]$ 

```

The definition above first ensures that  $x \gg y$  adds the working hypothesis to  $y$ . Then it sets the working hypothesis to  $x$ . Then it adds  $x$  to all previously proved proof lines. Then it associates the label  $l$  with the conclusion  $x \Rightarrow x$ . Then it tactic evaluates the remainder of the proof. Finally it adds a proof of  $x \Rightarrow x$ .

```

 $[\text{add-first-hyp}^* ( h , p , c )] \stackrel{\bullet}{=}$ 
if  $p$  then  $\top$  else  $\text{add-first-hyp} ( h , p^h , c ) :: \text{add-first-hyp}^* ( h , p^t , c )]$ 

```

```

 $[\text{add-first-hyp} ( h , p , c )] \stackrel{\bullet}{=}$ 
let  $\langle l, r, a \rangle = p$  in
if  $\text{is-meta-term} ( r , c )$  then  $p$  else
let  $a = [{}_a \text{A1}^{\text{ID}\triangleright} @ \underline{h} @ \underline{r}; \underline{a}]$  in
let  $r = [{}_r \underline{h} \Rightarrow \underline{r}]$  in
 $\langle l, r, a \rangle]$ 

```

### 3.7 Handling additional hypotheses

```

 $[\text{tactic-next-hyp} ( h , t , s , c )] \stackrel{\bullet}{=}$ 
let  $\langle \top, l, x, n \rangle = t$  in
let  $h' = [{}_t \underline{h} \wedge \underline{x}]$  in
let  $s = s[\text{hook-hyp} \rightarrow h']$  in
let  $s = s[\text{hook-pre} \rightarrow \text{add-next-hyp}^* ( x , s[\text{hook-pre}], c )]$  in
let  $x' = [{}_t \underline{h}' \Rightarrow \underline{x}]$  in
let  $s = \text{tactic-push} ( \text{hook-pre} , \langle l, x', [x' \underline{x}^I] \rangle , s )$  in
let  $s = \text{taceval} ( n , s , c )$  in
let  $a = s[\text{hook-arg}]$  in
let  $r = s[\text{hook-res}]$  in
if not  $r \stackrel{r}{=} [x \Rightarrow y]$  or not  $r^1 \stackrel{r}{=} [x \wedge y]$  then
 $\text{error} ( x , \text{LocateProofLine} ( x , c ) \text{diag} ( \text{'Malformed result of hypothetical reasoning'} ) \text{disp} ( r ) \text{end diagnose} )$  else
let  $a' = [{}_t \text{Mend1.48e}^{\text{ID}\triangleright} @ \underline{h} @ \underline{x}]$  in
let  $a'' = [{}_t (\text{Lcurry}^{\text{ID}\triangleright} @ \underline{r}^{11} @ \underline{r}^{12} @ \underline{r}^2) \triangleright]$  in
let  $r = [{}_t \underline{r}^{11} \Rightarrow \underline{r}^{12} \Rightarrow \underline{r}^2]$  in
 $s[\text{hook-arg} \rightarrow [{}_t \underline{a}'; s[\text{hook-arg}]; \underline{a}'']][\text{hook-res} \rightarrow r]$ 

```

The definitions above adds the hypothesis to the working hypothesis. Then it adds  $x$  to all previously proved proof lines. Then it adds  $x$  to

all previously proved proof lines. Then it associates the label  $l$  with the conclusion  $h \wedge x \Rightarrow x$ . Then it tactic evaluates the remainder of the proof. Finally it adds a proof of  $h \wedge x \Rightarrow x$  and Curries the result from  $h \wedge x \Rightarrow b$  to  $h \Rightarrow x \Rightarrow b$  where  $h$  was the working hypothesis before the hypothesis  $x$  was added.

```
[add-next-hyp* ( x , p , c ) ≐
  if p then ⊤ else add-next-hyp ( x , ph , c ) :: add-next-hyp* ( x ,
  pt , c )]
```

```
[add-next-hyp ( x , p , c ) ≐
  let ⟨l, t, a⟩ = p in
  if is-meta-term ( t , c ) then p else
  let ⟨⊤, h, r⟩ = t in
  let a = [a a; (LnexthypID▷ @ h @ x @ r)▷] in
  let r = [r h ∧ x ⇒ r] in
  ⟨l, r, a⟩]
```

### 3.8 Adaption

```
[f.unitac-adapt ( t , s , c ) ≐
  if t = var then s else
  let r = s[hook-res] in
  let d = r metadef ( c ) in
  if not d then s else
  if not r ≐ [x ⇒ y] then error ( r , LocateProofLine ( r , c )
  diag ( ‘Malformed result of hypothetical reasoning’ ) disp ( r ) end
  diagnose ) else
  let h = r1 in
  let r = r2 in
  if not r metadef ( c ) then s else
  let d = r mathdef ( c ) in
  if d = t then s else
  let a = s[hook-arg] in
  if d = all then f.unitac-adapt-all ( t , h , a , r , r , s[hook-idx] ,
  ⊤ , s , c ) else
  if d = imply then
  let x = r1 in
  let y = r2 in
  let a = [t a; (h ⇒ x)I; (MPID▷ @ h @ x @ y)▷▷] in
  let r = [t h ⇒ y] in
  f.unitac-adapt ( t , s[hook-arg→a][hook-res→r] , c ) else
  if t then s else
  error ( a ,
  diag ( ‘Cannot convert’ ) disp ( r )
  diag ( ‘to type ‘‘ ’ ) diag ( t ) diag ( ‘’’ ) end diagnose ).
```

Add  $\text{mp}$  and  $\text{at}$  operations to the argumentation inside  $s$  until the root of the conclusion is of type  $t$ . As examples,  $t$  could be “imply”, “all”, “var”, or  $\top$ . In the latter case, a maximum of  $\text{mp}$  and  $\text{at}$  operations are added. When  $t$  is “var”, a minimum of operators are added (meaning that  $s$  is returned unchanged).

```
[f.unitac-adapt-all ( t , h , a , R , r , i , b , s , c ) ≐
  if r mathdef ( c ) = all then
  let u = r1 in
  let r = r2 in
  let v = univar ( a , u , i ) in
  let b = ( u :: v ) :: b in
  f.unitac-adapt-all ( t , h , a , R , r , i + 1 , b , s , c ) else
  let s = s[hook-idx→i] in
  let r' = f.subst ( r , b , c ) in
  let a = [a a; (LAP4ID▷ @ h @ R @ r')*▷] in
  let r = [r' h ⇒ r'] in
  s[hook-arg→a][hook-res→r]]
```

Add  $\text{at}$  operations to the argumentation inside  $s$  until the root of the conclusion differs from “all”. In a call to the functions,  $t$  indicates what to adapt to,  $h$  is the hypothesis,  $a$  is the argumentation,  $r$  is the result of the argumentation,  $R$  is the original result which is passed down unchanged,  $i$  is an index for generating fresh variables,  $b$  is a substitution,  $s$  is a tactic state, and  $c$  is a cache.

### 3.9 Ponens

$[x \triangleright y \stackrel{\text{unitac}}{=} \lambda u. \text{f.unitac-ponens} ( u )]$

$[x \triangleright \stackrel{\text{unitac}}{=} \lambda u. \text{f.unitac-Ponens} ( u )]$

```
[f.unitac-ponens ( u ) ≐
  let ( t , s , c ) = u in
  let s = unieval ( t1 , s , c ) in
  let s = f.unitac-adapt ( imply , s , c ) in
  let a = s[hook-arg] in
  let r = s[hook-res] in
  if not r  $\stackrel{\text{r}}{=} [x \Rightarrow y]$  then
  error ( t , LocateProofLine ( t , c ) diag ( ‘Malformed result of
  hypothetical reasoning’ ) disp ( r ) end diagnose ) else
  let h = r1 in
  let r = r2 in
  let x = r1 in
  let y = r2 in
  let s = unieval ( t2 , s , c ) in
  let a' = s[hook-arg] in
```

```

let  $r' = s[\text{hook-res}]$  in
if not  $r' \stackrel{r}{=} [x \Rightarrow y]$  then
error (  $t$  , LocateProofLine (  $t$  ,  $c$  ) diag ( ‘Malformed result of
hypothetical reasoning’ ) disp (  $r'$  ) end diagnose ) else
let  $r' = r'^2$  in
let  $u = s[\text{hook-uni}]$  in
let  $u = \text{unify} ( x , r' , u )$  in
let  $s = s[\text{hook-uni} \rightarrow u]$  in
let  $a = [t \underline{a}; a'; (\text{MP}^{\text{ID}\triangleright} @ \underline{h} @ \underline{x} @ \underline{y})^{\triangleright\triangleright}]$  in
let  $r = [t \underline{h} \Rightarrow y]$  in
 $s[\text{hook-arg} \rightarrow a][\text{hook-res} \rightarrow r]$ 

```

If the argumentation has form  $a \triangleright a'$  then convert  $a$  into something whose result has form  $h \Rightarrow x \Rightarrow y$ . Convert  $a'$  into something whose result has form  $h \Rightarrow x'$ . Unify  $x$  with  $x'$  and return argumentation  $(a; a'; \text{A2} @ \dots)$  and conclusion  $h \Rightarrow y$ .

```

[f.unitac-Ponens (  $u$  )  $\stackrel{\bullet}{=}$ 
let  $\langle t, s, c \rangle = u$  in
let  $s = \text{unieval} ( t^1 , s , c )$  in
let  $s = \text{f.unitac-adapt} ( \text{imply} , s , c )$  in
let  $a = s[\text{hook-arg}]$  in
let  $r = s[\text{hook-res}]$  in
if not  $r \stackrel{r}{=} [x \Rightarrow y]$  then
error (  $t$  , LocateProofLine (  $t$  ,  $c$  ) diag ( ‘Malformed result of
hypothetical reasoning’ ) disp (  $r$  ) end diagnose ) else
let  $h = r^1$  in
let  $r = r^2$  in
let  $x = r^1$  in
let  $y = r^2$  in
let  $a = [t \underline{a}; (h \Rightarrow x)^1; (\text{MP}^{\text{ID}\triangleright} @ \underline{h} @ \underline{x} @ \underline{y})^{\triangleright\triangleright}]$  in
let  $r = [t \underline{h} \Rightarrow y]$  in
 $s[\text{hook-arg} \rightarrow a][\text{hook-res} \rightarrow r]$ 

```

### 3.10 At

$[x @ \stackrel{\text{unitac}}{=} \lambda u. \text{f.unitac-At} ( u )]$

$[x @ y \stackrel{\text{unitac}}{=} \lambda u. \text{f.unitac-at} ( u )]$

```

[f.unitac-At (  $u$  )  $\stackrel{\bullet}{=}$ 
let  $\langle t, s, c \rangle = u$  in
let  $s = \text{unieval} ( t^1 , s , c )$  in
let  $s = \text{f.unitac-adapt} ( \text{all} , s , c )$  in
let  $a = s[\text{hook-arg}]$  in
let  $r = s[\text{hook-res}]$  in
if not  $r \stackrel{r}{=} [x \Rightarrow y]$  then

```

error (  $t$  , LocateProofLine (  $t$  ,  $c$  ) diag ( ‘Malformed result of hypothetical reasoning’ ) disp (  $r$  ) end diagnose ) **else**

**let**  $h = r^1$  **in**

**let**  $r = r^2$  **in**

**let**  $x = r^1$  **in**

**let**  $y = r^2$  **in**

**let**  $i = s[\text{hook-idx}]$  **in**

**let**  $s = s[\text{hook-idx} \rightarrow i + 1]$  **in**

**let**  $v = \text{univar} ( a , x , i )$  **in**

**let**  $r' = \text{f.subst} ( y , \langle x :: v \rangle , c )$  **in**

**let**  $a = [t \underline{a}; (\text{LAP4}^{\text{ID}\triangleright} @ \underline{h} @ \underline{r} @ \underline{r}')^* \triangleright ]$  **in**

**let**  $r = [r \underline{h} \Rightarrow \underline{r}']$  **in**

$s[\text{hook-arg} \rightarrow a][\text{hook-res} \rightarrow r]$

If the argumentation has form  $a^{\textcircled{a}}$  then convert  $a$  into something whose conclusion is expected to be of form  $h \Rightarrow \forall x: y$ . Then replace  $x$  by a fresh variable  $v$  in  $y$  and return conclusion  $h \Rightarrow y$  and a suitable argumentation.

[f.unitac-at (  $u$  )  $\doteq$

**let**  $\langle t, s, c \rangle = u$  **in**

**let**  $s = \text{unieval} ( t^1 , s , c )$  **in**

**let**  $s = \text{f.unitac-adapt} ( \text{all} , s , c )$  **in**

**let**  $a = s[\text{hook-arg}]$  **in**

**let**  $r = s[\text{hook-res}]$  **in**

**if not**  $r \stackrel{\text{r}}{=} [x \Rightarrow y]$  **then**

error (  $t$  , LocateProofLine (  $t$  ,  $c$  ) diag ( ‘Malformed result of hypothetical reasoning’ ) disp (  $r$  ) end diagnose ) **else**

**let**  $h = r^1$  **in**

**let**  $r = r^2$  **in**

**let**  $x = r^1$  **in**

**let**  $y = r^2$  **in**

**let**  $i = s[\text{hook-idx}]$  **in**

**let**  $s = s[\text{hook-idx} \rightarrow i + 1]$  **in**

**let**  $v = \text{univar} ( a , x , i )$  **in**

**let**  $r' = \text{f.subst} ( y , \langle x :: v \rangle , c )$  **in**

**let**  $a = [t \underline{a}; (\text{LAP4}^{\text{ID}\triangleright} @ \underline{h} @ \underline{r} @ \underline{r}')^* \triangleright ]$  **in**

**let**  $r = [r \underline{h} \Rightarrow \underline{r}']$  **in**

**let**  $u = s[\text{hook-uni}]$  **in**

**let**  $u = \text{unify} ( t^2 , v , u )$  **in**

**let**  $s = s[\text{hook-uni} \rightarrow u]$  **in**

$s[\text{hook-arg} \rightarrow a][\text{hook-res} \rightarrow r]$

If the argumentation has form  $a @ a'$  then convert  $a$  into something whose conclusion is expected to be of form  $\Pi x: y$ . Then replace  $x$  by a fresh meta variable  $v$  in  $y$ , unify  $v$  with  $a'$ , and return argumentation  $a @ v$  and conclusion  $y$ .

### 3.11 Deduction tactic

We now state that Prop, FOL, and PA proofs may use deduction:

[Prop  $\stackrel{\text{tactic}}{=} \lambda u. \text{tactic-Prop} ( u )$ ].

[FOL  $\stackrel{\text{tactic}}{=} \lambda u. \text{tactic-FOL} ( u )$ ].

[PA  $\stackrel{\text{tactic}}{=} \lambda u. \text{tactic-FOL} ( u )$ ].

[tactic-Prop (  $u$  )  $\stackrel{\bullet}{=}$   
**let**  $\langle t, s, c \rangle = u$  **in**  
**let**  $\langle \top, x, y \rangle = t$  **in**  
**let**  $t = [t \underline{x} \vdash (\text{Prop Rule} \gg \text{Prop}; \text{tactic-ded} ( \top, \underline{y} ))]$  **in**  
 taceval (  $t, s, c$  )]

[tactic-FOL (  $u$  )  $\stackrel{\bullet}{=}$   
**let**  $\langle t, s, c \rangle = u$  **in**  
**let**  $\langle \top, x, y \rangle = t$  **in**  
**let**  $t = [t \underline{x} \vdash (\text{Prop Rule} \gg \text{Prop}; \text{FOL Rule} \gg \text{FOL}; \text{tactic-ded} ( \top, \underline{y} ))]$  **in**  
**let**  $r = \text{taceval} ( t, s, c )$  **in**  
 $r$ ]

### 3.12 Test proofs

Elementary two level deduction. Import of L03 into inner deduction.

FOL **lemma** lemma-x:  $x \Rightarrow y \Rightarrow x \square$

**Proof of lemma-x:**

L01:	Premise $\gg$	FOL	;
L02:	Prop Rule $\gg$	Prop	;
L03:	Hypothesis $\gg$	$x$	;
L04:	Hypothesis $\gg$	$y$	;
L05:	L03 $\gg$	$x$	$\square$

Import of lemma.

FOL **lemma** lemma-y:  $h \Rightarrow x \Rightarrow y \Rightarrow x \square$

**Proof of lemma-y:**

L01:	Premise $\gg$	FOL	;
L02:	Prop Rule $\gg$	Prop	;
L03:	Hypothesis $\gg$	$h$	;
L04:	lemma-x $\gg$	$x \Rightarrow y \Rightarrow x$	$\square$

Import or rule.

PA **lemma** lemma-z:  $h \Rightarrow i \Rightarrow x + 0 = x \square$

**Proof of** lemma-z:

L01:	Premise $\gg$	PA	;
L02:	FOL Rule $\gg$	FOL	;
L03:	Prop Rule $\gg$	Prop	;
L04:	Hypothesis $\gg$	$h$	;
L05:	Hypothesis $\gg$	$i$	;
L06:	S5 $\gg$	$x + 0 = x$	$\square$

Proof that contains a dummy hypothesis (Line L04) so that  $x \sqsupseteq y$  works.

PA **lemma** lemma-u:  $x = x \square$

**Proof of** lemma-u:

L01:	Premise $\gg$	PA	;
L02:	FOL Rule $\gg$	FOL	;
L03:	Prop Rule $\gg$	Prop	;
L04:	Deduction		;
L05:	S5 $\gg$	$x + 0 = x$	;
L06:	S1 $\sqsupseteq$ L05 $\sqsupseteq$ L05 $\gg$	$x = x$	$\square$

Fully specified instantiation.

PA **lemma** lemma-v:  $h \Rightarrow i \Rightarrow x + 0 = x \square$

**Proof of** lemma-v:

L01:	Premise $\gg$	PA	;
L02:	FOL Rule $\gg$	FOL	;
L03:	Prop Rule $\gg$	Prop	;
L04:	Hypothesis $\gg$	$h$	;
L05:	Hypothesis $\gg$	$i$	;
L06:	S5 @ $x \gg$	$x + 0 = x$	$\square$

Partially specified instantiation.

PA **lemma** lemma-w:  $h \Rightarrow i \Rightarrow x + 0 = x \square$

**Proof of** lemma-w:

L01:	Premise $\gg$	PA	;
L02:	FOL Rule $\gg$	FOL	;
L03:	Prop Rule $\gg$	Prop	;
L04:	Hypothesis $\gg$	$h$	;
L05:	Hypothesis $\gg$	$i$	;
L06:	S5@ $\gg$	$x + 0 = x$	$\square$

### 3.13 A proof of $x + y = y + x$

PA lemma 3.2a:  $\forall x: x = x$   $\square$

PA proof of 3.2a:

L01:	S5 $\gg$	$x + 0 = x$	;
L02:	S1 $\supseteq$ L01 $\supseteq$ L01 $\gg$	$x = x$	;
L03:	Gen1 $\triangleright$ L02 $\gg$	$\forall x: x = x$	$\square$

PA lemma 3.2b:  $\forall x, y: (x = y \Rightarrow y = x)$   $\square$

PA proof of 3.2b:

L00:	Block $\gg$	Begin	;
L01:	Hypothesis $\gg$	$x = y$	;
L02:	3.2a $\gg$	$x = x$	;
L03:	S1 $\supseteq$ L01 $\supseteq$ L02 $\gg$	$y = x$	;
L04:	Block $\gg$	End	;
L05:	Gen2 $\triangleright$ L04 $\gg$	$\forall x, y: (x = y \Rightarrow y = x)$	$\square$

PA lemma 3.2c:  $\forall x, y, z: (x = y \Rightarrow y = z \Rightarrow x = z)$   $\square$

PA proof of 3.2c:

L01:	Block $\gg$	Begin	;
L02:	Hypothesis $\gg$	$x = y$	;
L03:	Hypothesis $\gg$	$y = z$	;
L04:	3.2b $\supseteq$ L02 $\gg$	$y = x$	;
L05:	S1 $\supseteq$ L04 $\supseteq$ L03 $\gg$	$x = z$	;
L06:	Block $\gg$	End	;
L07:	Gen3 $\triangleright$ L06 $\gg$	$\forall x, y, z:$ $(x = y \Rightarrow y = z \Rightarrow x = z)$	$\square$

PA lemma 3.2d:  $\forall x, y, z: (x = z \Rightarrow y = z \Rightarrow x = y)$   $\square$

PA proof of 3.2d:

L00:	Block $\gg$	Begin	;
L01:	Hypothesis $\gg$	$x = z$	;
L02:	Hypothesis $\gg$	$y = z$	;
L03:	3.2b $\supseteq$ L02 $\gg$	$z = y$	;
L04:	3.2c $\supseteq$ L01 $\supseteq$ L03 $\gg$	$x = y$	;
L05:	Block $\gg$	End	;
L06:	Gen3 $\triangleright$ L05 $\gg$	$\forall x, y, z:$ $(x = z \Rightarrow y = z \Rightarrow x = y)$	$\square$

PA lemma 3.2e:  $\forall x, y, z: (x = y \Rightarrow x + z = y + z)$   $\square$

PA proof of 3.2e:

L01:	Block $\gg$	Begin	;
L02:	Hypothesis $\gg$	$x = y$	;

L03:	S5 $\gg$	$x + 0 = x$	;
L04:	3.2c $\supseteq$ L03 $\supseteq$ L02 $\gg$	$x + 0 = y$	;
L05:	S5 $\gg$	$y + 0 = y$	;
L06:	3.2d $\supseteq$ L04 $\supseteq$ L05 $\gg$	$x + 0 = y + 0$	;
L07:	Block $\gg$	End	;
L08:	Block $\gg$	Begin	;
L09:	Hypothesis $\gg$	$x = y \Rightarrow x + z = y + z$	;
L10:	Hypothesis $\gg$	$x = y$	;
L11:	MP' $\triangleright$ L09 $\triangleright$ L10 $\gg$	$x + z = y + z$	;
L12:	S2 $\supseteq$ L11 $\gg$	$(x + z)' = (y + z)'$	;
L13:	S6 $\gg$	$x + z' = (x + z)'$	;
L14:	3.2c $\supseteq$ L13 $\supseteq$ L12 $\gg$	$x + z' = (y + z)'$	;
L15:	S6 $\gg$	$y + z' = (y + z)'$	;
L16:	3.2d $\supseteq$ L14 $\supseteq$ L15 $\gg$	$x + z' = y + z'$	;
L17:	Block $\gg$	End	;
L18:	Induction @ $z \triangleright$ L07 $\triangleright$ L17 $\gg$	$x = y \Rightarrow x + z = y + z$	;
L19:	Gen3 $\triangleright$ L18 $\gg$	$\forall x, y, z:$ $(x = y \Rightarrow x + z = y + z)$	□

PA lemma 3.2f:  $\forall x: x = 0 + x$  □

PA proof of 3.2f:

L01:	S5 $\gg$	$0 + 0 = 0$	;
L02:	3.2b $\supseteq$ L01 $\gg$	$0 = 0 + 0$	;
L03:	Block $\gg$	Begin	;
L04:	Hypothesis $\gg$	$x = 0 + x$	;
L05:	S2 $\supseteq$ L04 $\gg$	$x' = (0 + x)'$	;
L06:	S6 $\gg$	$0 + x' = (0 + x)'$	;
L07:	3.2d $\supseteq$ L05 $\supseteq$ L06 $\gg$	$x' = 0 + x'$	;
L08:	Block $\gg$	End	;
L09:	Induction @ $x \triangleright$ L02 $\triangleright$ L08 $\gg$	$x = 0 + x$	;
L10:	Gen1 $\triangleright$ L09 $\gg$	$\forall x: x = 0 + x$	□

PA lemma 3.2g:  $\forall x, y: x' + y = (x + y)'$  □

PA proof of 3.2g:

L01:	S5 $\gg$	$x' + 0 = x'$	;
L02:	S5 $\gg$	$x + 0 = x$	;
L03:	S2 $\supseteq$ L02 $\gg$	$(x + 0)' = x'$	;
L04:	3.2d $\supseteq$ L01 $\supseteq$ L03 $\gg$	$x' + 0 = (x + 0)'$	;
L05:	Block $\gg$	Begin	;
L06:	Hypothesis $\gg$	$x' + y = (x + y)'$	;
L07:	S2 $\supseteq$ L06 $\gg$	$(x' + y)' = (x + y)''$	;
L08:	S6 $\gg$	$x' + y' = (x' + y)'$	;
L09:	3.2c $\supseteq$ L08 $\supseteq$ L07 $\gg$	$x' + y' = (x + y)''$	;
L10:	S6 $\gg$	$x + y' = (x + y)''$	;
L11:	S2 $\supseteq$ L10 $\gg$	$(x + y')' = (x + y)''$	;

L12:	3.2d $\supseteq$ L09 $\supseteq$ L11 $\gg$	$x' + y' = (x + y)'$	;
L13:	Block $\gg$	End	;
L14:	Induction @ $y \triangleright$ L04 $\triangleright$ L13 $\gg$	$x' + y = (x + y)'$	;
L15:	Gen2 $\triangleright$ L14 $\gg$	$\forall x, y: x' + y = (x + y)'$	□

PA lemma 3.2h:  $\forall x, y: x + y = y + x$  □

PA proof of 3.2h:

L01:	S5 $\gg$	$x + 0 = x$	;
L02:	3.2f $\gg$	$x = 0 + x$	;
L03:	3.2c $\supseteq$ L01 $\supseteq$ L02 $\gg$	$x + 0 = 0 + x$	;
L04:	Block $\gg$	Begin	;
L05:	Hypothesis $\gg$	$x + y = y + x$	;
L06:	S2 $\supseteq$ L05 $\gg$	$(x + y)' = (y + x)'$	;
L07:	S6 $\gg$	$x + y' = (x + y)'$	;
L08:	3.2c $\supseteq$ L07 $\supseteq$ L06 $\gg$	$x + y' = (y + x)'$	;
L09:	3.2g $\gg$	$y' + x = (y + x)'$	;
L10:	3.2d $\supseteq$ L08 $\supseteq$ L09 $\gg$	$x + y' = y' + x$	;
L11:	Block $\gg$	End	;
L12:	Induction @ $y \triangleright$ L03 $\triangleright$ L11 $\gg$	$x + y = y + x$	;
L13:	Gen2 $\triangleright$ L12 $\gg$	$\forall x, y: x + y = y + x$	□

## References

- [1] E. Mendelson. *Introduction to Mathematical Logic*. Wadsworth and Brooks, 3. edition, 1987.