Introduction to Formal Proof

Bernard Sufrin

Trinity Term 2018

5: Theories

Theories

▷ The subject matter of predicate logic is “all models (over all signatures)”

∘ Mathematical logicians consider the soundness and completeness of particular deductive systems for the logic, and also consider its decidability.

▷ The subject matter of much of (formal) mathematics and computer science is (in one sense) more constrained

∘ We want to make proofs about models that satisfy certain laws (a.k.a. axioms)

∘ We want to use sound deductive systems to make these proofs

⋆ So we start with a sound deductive system (for FOL) ...

⋆ add a signature $(\mathcal{C}, \mathcal{F}, \mathcal{P})$ and **laws** for the models we are interested in ...

⋆ and see what happens next! (*i.e.* what we can prove)

∘ The domain of discourse is left implicit ...
... though we sometimes have a particular domain of discourse in mind!

∘ If we add a law that leads to contradiction then no model will satisfy our theory!

Example: elementary group theory

▷ Constants: ι

▷ Functions: $\cdot\otimes\cdot$ $\sim\cdot$

▷ Axiom Schemes (Laws):

$$\dfrac{}{T_1 \otimes (T_2 \otimes T_3) = (T_1 \otimes T_2) \otimes T_3}\ \otimes\text{-ass}$$

$$\dfrac{}{\iota \otimes T = T}\ \iota\text{-id}$$

$$\dfrac{}{\sim T \otimes T = \iota}\ \sim$$

▷ Example models: the integers with $\iota = 0, \otimes = +$; the nonzero rationals or reals with $\iota = 1, \otimes = \times$; and (for any set S) the bijective functions in $S \to S$ with $\iota = Id_S, \otimes = \cdot$(composition).

▷ Consequences can be proven using only equational reasoning, for example: $T \otimes \iota = T$

▷ In the “transitive equalities” presentation style the *essence* of the proof is:

$$\begin{array}{lll}
1: & T \otimes \iota & \\
2: & = T \otimes (\sim T \otimes T) & \{ \text{Fold} \sim \\
3: & = (T \otimes \sim T) \otimes T & \{ \text{Unfold } \otimes\text{-ass} \\
4: & = \iota \otimes T & \{ \text{Unfold Thm } T \otimes \sim T = \iota \\
4: & = T & \{ \iota\text{-id}
\end{array}$$

▷ This concise presentation hides the details of the application of the transitivity rules:[1]

1: T⊗ ι =T⊗~T⊗T	Fold ~
2: T⊗~T⊗T=(T⊗~T)⊗T	Unfold ⊗-ass
3: (T⊗~T)⊗T=(ι)⊗T	Unfold Theorem T⊗~T= ι
4: (ι)⊗T=T	ι -id
5: (T⊗~T)⊗T=T	Derived Rule =trans 3,4
6: T⊗~T⊗T=T	Derived Rule =trans 2,5
7: T⊗ ι =T	Derived Rule =trans 1,6

[1] Jape treats ⊗ as right-associative and doesn't fully bracket $T_1 \otimes (T_2 \otimes T_3)$ in displays.

▷ It also relies on a stylized concise form of reporting of “folds” and “unfolds”

$$\dfrac{\dfrac{}{T\otimes\iota=T\otimes{\sim}T\otimes T}\,\text{Fold}\sim \qquad \dfrac{\dfrac{}{T\otimes{\sim}T\otimes T=(T\otimes{\sim}T)\otimes T}\,\text{Unfold}\otimes\text{-ass} \qquad \dfrac{\dfrac{}{(T\otimes{\sim}T)\otimes T=(\iota)\otimes T}\,\text{Unfold Theorem }T\otimes{\sim}T=\iota \qquad \dfrac{}{(\iota)\otimes T=T}\,\iota\text{-id}}{(T\otimes{\sim}T)\otimes T=T}\,\text{Derived Rule =trans}}{T\otimes{\sim}T\otimes T=T}\,\text{Derived Rule =trans}}{T\otimes\iota=T}\,\text{Derived Rule =trans}$$

$$\dfrac{\dfrac{\dfrac{}{{\sim}T\otimes T=\iota}\,\sim}{T\otimes\iota=T\otimes{\sim}T\otimes T}\,\text{rewriteRL} \qquad \dfrac{\dfrac{}{T\otimes{\sim}T\otimes T=(T\otimes{\sim}T)\otimes T}\,\otimes\text{-ass} \qquad \dfrac{\dfrac{\dfrac{}{T\otimes{\sim}T=\iota}\,\text{Theorem }T\otimes{\sim}T=\iota}{(T\otimes{\sim}T)\otimes T=(\iota)\otimes T}\,\text{rewriteLR} \qquad \dfrac{}{(\iota)\otimes T=T}\,\iota\text{-id}}{(T\otimes{\sim}T)\otimes T=T}\,\text{Derived Rule =trans}}{T\otimes{\sim}T\otimes T=T}\,\text{Derived Rule =trans}}{T\otimes\iota=T}\,\text{Derived Rule =trans}$$

☜11

▷ Inverses are unique

1:	Ti⊗T= ι	assumption
2:	Ti	
3:	= Ti⊗ ι	Fold Theorem T⊗ ι =T
4:	= Ti⊗T⊗~T	Fold Theorem T⊗~T= ι
5:	= (Ti⊗T)⊗~T	Unfold ⊗-ass
6:	= (ι)⊗~T	Unfold hyp
7:	= ~T	Unfold ι -id

▷ Completely formal proofs using associativity can be ... tedious, for example:

1:	T⊗~T	
2:	= ι ⊗T⊗~T	Fold ι -id
3:	= (~(T⊗~T)⊗T⊗~T)⊗T⊗~T	Fold ~
4:	= ~(T⊗~T)⊗((T⊗~T)⊗T⊗~T)	Fold ⊗-ass
5:	= ~(T⊗~T)⊗(T⊗(~T⊗T⊗~T))	Fold ⊗-ass
6:	= ~(T⊗~T)⊗((T⊗~T)⊗T⊗~T)	Unfold ⊗-ass
7:	= ~(T⊗~T)⊗(((T⊗~T)⊗T)⊗~T)	Unfold ⊗-ass
8:	= ~(T⊗~T)⊗((T⊗(~T⊗T))⊗~T)	Fold ⊗-ass
9:	= ~(T⊗~T)⊗((T⊗(ι))⊗~T)	Unfold ~
10:	= ~(T⊗~T)⊗(T⊗(ι ⊗~T))	Fold ⊗-ass
11:	= ~(T⊗~T)⊗(T⊗(~T))	Unfold ι -id
12:	= ι	Unfold ~

Here lines 3-8 could be summarised as: "by associativity of ⊗", and interactive proof assistants should provide some sort of interface that gets on with the details of "flattening, then rebracketing" under the direction of the user.

Example: theory of Natural Numbers

▷ Constants: 0

▷ Functions: succ $\cdot, \cdot + \cdot, \cdot \times \cdot, \ldots$

▷ Laws:[2]

$$\frac{}{\Gamma \vdash \mathsf{succ}(T) \neq 0}\ \text{P3 (0 is the beginning)} \qquad \frac{}{\Gamma \vdash \mathsf{succ}(T_1) = \mathsf{succ}(T_2) \rightarrow T_1 = T_2}\ \text{P4 (injectivity)}$$

$$\frac{\Gamma \vdash \phi(0) \qquad \Gamma, \phi(m) \vdash \phi(\mathsf{succ}(m))}{\Gamma \vdash \phi(T)}\ \text{P5 natinduction } (m \text{ fresh})$$

▷ Nat induction is a schema parameterized by $\phi(\cdot)$ – a formula in which $\cdot$ may appear.

▷ The "intended model" is the natural numbers.

▷ The even numbers also constitute a model; indeed there are infinitely many models! (why?)

[2] These were first listed by Dedekind – but are usually attributed to Peano.

▷ Axiom schemas with parameters T_1, T_2 (terms)

$$\frac{}{0 + T_2 = T_2}\ +.0 \qquad \frac{}{\mathsf{succ}(T_1) + T_2 = \mathsf{succ}(T_1 + T_2)}\ +.1$$

$$\frac{}{0 \times T_2 = 0}\ \times.0 \qquad \frac{}{\mathsf{succ}(T_1) \times T_2 = T_2 + (T_1 \times T_2)}\ \times.1$$

▷ These schemas characterize addition and multiplication (almost) uniquely (but this needs to be proved)

▷ Consequences: commutativity and associativity of $+, \times$ distributivity of $\times$ through $+$, *etc.*, *etc.*

An inductive proof using substitution (equals-elimination) to rewrite equal subterms within successive formulae

1:	0+(T2+T3)=T2+T3	+'0
2:	0+T2=T2	+'0
3:	T2+T3=(T2)+T3	=-i
4:	T2+T3=(0+T2)+T3	Derived Rule =-e ← 2,3
5:	0+(T2+T3)=(0+T2)+T3	Derived Rule =-e ← 1,4
6:	m+(T2+T3)=(m+T2)+T3	assumption
7:	succ(m)+(T2+T3)=succ(m+T2+T3)	+'1
8:	succ(m)+T2=succ(m+T2)	+'1
9:	(succ(m+T2))+T3=succ((m+T2)+T3)	+'1
10:	m+T2+T3=(m+T2)+T3	hyp 6
11:	succ((m+T2)+T3)=succ((m+T2)+T3)	=-i
12:	succ(m+T2+T3)=succ((m+T2)+T3)	Derived Rule =-e ← 10,11
13:	succ(m+T2+T3)=(succ(m+T2))+T3	Derived Rule =-e ← 9,12
14:	succ(m+T2+T3)=(succ(m)+T2)+T3	Derived Rule =-e ← 8,13
15:	succ(m)+(T2+T3)=(succ(m)+T2)+T3	Derived Rule =-e ← 7,14
16:	T1+(T2+T3)=(T1+T2)+T3	natinduction 5,6-15

The proof consists of successive formulae that are equivalent up to substitution of equal terms

The same proof: using folds and unfolds, and presented in the transformational style

1:	0+(T2+T3)	
2:	= T2+T3	Unfold +'0
3:	= (0+T2)+T3	Fold +'0
4:	m+(T2+T3)=(m+T2)+T3	assumption
5:	succ(m)+(T2+T3)	
6:	= succ(m+T2+T3)	Unfold +'1
7:	= succ((m+T2)+T3)	Unfold hyp
8:	= (succ(m+T2))+T3	Fold +'1
9:	= (succ(m)+T2)+T3	Fold +'1
10:	T1+(T2+T3)=(T1+T2)+T3	natinduction 1-3,4-9

This is not *exactly* the same as the substitutivity proof, but all the "essential" steps in it are the same.

Comparison between the transformational proof and (a compact form) of the substitutive proof

1:	0+(T2+T3)	
2:	= T2+T3	Unfold +'0
3:	= (0+T2)+T3	Fold +'0
4:	m+(T2+T3)=(m+T2)+T3	assumption
5:	succ(m)+(T2+T3)	
6:	= succ(m+T2+T3)	Unfold +'1
7:	= succ((m+T2)+T3)	Unfold hyp
8:	= (succ(m+T2))+T3	Fold +'1
9:	= (succ(m)+T2)+T3	Fold +'1
10:	T1+(T2+T3)=(T1+T2)+T3	natinduction 1-3,4-9

1:	T2+T3=(T2)+T3	=-i
2:	T2+T3=(0+T2)+T3	+'0 1
3:	0+(T2+T3)=(0+T2)+T3	+'0 2
4:	m+(T2+T3)=(m+T2)+T3	assumption
5:	succ((m+T2)+T3)=succ((m+T2)+T3)	=-i
6:	succ(m+T2+T3)=succ((m+T2)+T3)	hyp 5
7:	succ(m+T2+T3)=(succ(m+T2))+T3	+'1 6
8:	succ(m+T2+T3)=(succ(m)+T2)+T3	+'1 7
9:	succ(m)+(T2+T3)=(succ(m)+T2)+T3	+'1 8
10:	T1+(T2+T3)=(T1+T2)+T3	natinduction 3,4-9

(in the compact form, the uses of "=-e ←" are left implicit)

Theories with several types

In which we indicate how to formalize a (rather weak) notion of types thereby supporting proofs in theories in which several "types" are used

▷ For example suppose we want to build a theory of lists of numbers?
- ◦ We expect to be able to prove things about all numbers
- ◦ We expect to be able to prove things about all lists
- ◦ We expect to be able to characterize functions recursively on lists and on numbers
- ◦ The "untyped" induction and definition rules are no longer quite enough

Example: typed theory of natural numbers

▷ The main idea: supplement signatures with "typing predicates"

- Constants: 0
- Functions: succ $\cdot, \cdot + \cdot, \cdot \times \cdot, \ldots$
- Predicates: $\mathbb{N}(\cdot)$ meaning "· is a natural number"
- Laws:

$$\frac{}{\mathbb{N}(0)}\ \textsf{N0} \qquad \frac{\mathbb{N}(T)}{\mathbb{N}(\textsf{succ}(T))}\ \textsf{Nsucc}$$

$$\frac{\mathbb{N}(T)}{\textsf{succ}(T) \neq 0}\ \textsf{P3} \qquad \frac{\mathbb{N}(T_1) \qquad \mathbb{N}(T_2)}{\textsf{succ}(T_1) = \textsf{succ}(T_2) \to T_1 = T_2}\ \textsf{P4}$$

$$\frac{\Gamma \vdash \mathbb{N}(T) \qquad \Gamma \vdash \phi(0) \qquad \Gamma, \mathbb{N}(m), \phi(m) \vdash \phi(\textsf{succ}(m))}{\Gamma \vdash \phi(T)}\ \text{(nat induction)(fresh } m\text{)}$$

(Here $\phi(.)$ is – as usual – a "formula abstraction")

▷ Arithmetic expressions are also typed:

$$\frac{\mathbb{N}(T_1) \qquad \mathbb{N}(T_2)}{\mathbb{N}(T_1 + T_2)}\ \textsf{N+} \qquad \frac{\mathbb{N}(T_1) \qquad \mathbb{N}(T_2)}{\mathbb{N}(T_1 \times T_2)}\ \textsf{N}\times$$

▷ Arithmetic operator definitions get the expected typing antecedents, for example:

$$\frac{\mathbb{N}(T_1) \qquad \mathbb{N}(T_2)}{0 + T_2 = T_2}\ +.0 \qquad \frac{\mathbb{N}(T_1) \qquad \mathbb{N}(T_2)}{\textsf{succ}(T_1) + T_2 = \textsf{succ}(T_1 + T_2)}\ +.1$$

▷ Theorems now have type premisses, for example

$$\frac{}{\mathbb{N}(T_1), \mathbb{N}(T_2), \mathbb{N}(T_3) \vdash T_1 + (T_2 + T_3) = (T_1 + T_2) + T_3}\ +\text{-assoc}$$

▷ The typing antecedents of rules needed in the proof of a theorem are trivial to prove from the typing premisses

Example: typed theory of heterogeneous lists

- Constants: nil
- Functions: $\cdot : \cdot, \cdot +\!\!+ \cdot \ldots$
- Predicates: $\mathbb{L}(\cdot)$
- Laws: (the elements of a *heterogeneous* list need not have the same type)

$$\frac{}{\mathbb{L}(nil)}\ \textsf{Lnil} \qquad \frac{\mathbb{L}(TS)}{\mathbb{L}(T : TS)}\ \textsf{L:}$$

$$\frac{\mathbb{L}(TS)}{T : TS \neq nil}\ \textsf{:-} \qquad \frac{\mathbb{L}(TS) \qquad \mathbb{L}(TS')}{T : TS = T' : TS' \to T = T' \land TS = TS'}\ \textsf{:-inj}$$

$$\frac{\Gamma \vdash \mathbb{L}(T) \qquad \Gamma \vdash \phi(nil) \qquad \Gamma, \mathbb{L}(xs), \phi(xs) \vdash \phi(x : xs)}{\Gamma \vdash \phi(T)}\ \text{(list induction)(fresh } x, xs\text{)}$$

(Here $\phi(.)$ is – as usual – a "formula abstraction")

▷ Catenation and reverse defined in the usual way but with typing information added

$$\frac{\mathbb{L}(T)}{\mathbb{L}(rev(T))}\ \textsf{Lrev} \qquad \frac{}{rev(nil) = nil}\ \textsf{rev.0} \qquad \frac{\mathbb{L}(TS)}{rev(T : TS) = rev(T) +\!\!+ (T : nil)}\ \textsf{rev.1}$$

$$\frac{\mathbb{L}(TS) \qquad \mathbb{L}(TS')}{\mathbb{L}(TS +\!\!+ TS')}\ \textsf{L}+\!\!+ \qquad \frac{\mathbb{L}(TS)}{nil +\!\!+ TS = TS}\ +\!\!+ .0 \qquad \frac{\mathbb{L}(TS) \qquad \mathbb{L}(TS')}{(T : TS) +\!\!+ TS' = T : (TS +\!\!+ TS')}\ +\!\!+ .1$$

▷ "Typed" theories can be mixed: antecedents stop us inferring nonsense, *e.g.* $nil +\!\!+ 0 = 0$

▷ Theorems (as expected) have typing premisses, for example:

$$\frac{}{\mathbb{L}(T_1), \mathbb{L}(T_2), \mathbb{L}(T_3) \vdash T_1 +\!\!+ (T_2 +\!\!+ T_3) = (T_1 +\!\!+ T_2) + T_3}\ +\!\!+\text{-assoc}$$

▷ The formal proofs of these theorems are a lot like those we know and love from FP

▷ Some additional, but trivial, work is needed to satisfy the typing antecedents: compare

```
 1: | L(T1)                              | assumption
 2: | L(T2)                              | assumption
 3: | L(T3)                              | assumption
 4: | L(T2++T3)                          | L++ 2,3
 5: | nil++T2=T2                         | ++'0 2
 6: | nil++(T2++T3)                      |
 7: |   = T2++T3                         | ++'0 4
 8: |   = (nil++T2)++T3                  | rewriteRL 5
 9: | | L(xs)                            | assumption
10: | | xs++(T2++T3)=(xs++T2)++T3        | assumption
11: | | L(T2++T3)                        | L++ 2,3
12: | | L(xs++T2)                        | L++ 9,2
13: | | (x:(xs++T2))++T3=x:((xs++T2)++T3)| ++'1 12,3
14: | | x:xs++T2=x:(xs++T2)              | ++'1 9,2
15: | | x:xs++(T2++T3)                   |
16: | |   = x:(xs++T2++T3)               | ++'1 9,11
17: | |   = x:((xs++T2)++T3)             | rewriteLR 10
18: | |   = (x:(xs++T2))++T3             | rewriteRL 13
19: | |   = (x:xs++T2)++T3               | rewriteRL 14
20: | T1++(T2++T3)=(T1++T2)++T3          | listinduction 6-8,9-19,1
```

```
 1: | L(T1)                              | assumption
 2: | L(T2)                              | assumption
 3: | L(T3)                              | assumption
 4: | nil++(T2++T3)                      |
 5: |   = T2++T3                         | Unfold ++'0
 6: |   = (nil++T2)++T3                  | Fold ++'0,hyp
 7: | | L(xs)                            | assumption
 8: | | xs++(T2++T3)=(xs++T2)++T3        | assumption
 9: | | x:xs++(T2++T3)                   |
10: | |   = x:(xs++T2++T3)               | Unfold ++'1
11: | |   = x:((xs++T2)++T3)             | Unfold 8
12: | |   = (x:(xs++T2))++T3             | Fold ++'1,L++,hyp,hyp,hyp
13: | |   = (x:xs++T2)++T3               | Fold ++'1,hyp,hyp
14: | T1++(T2++T3)=(T1++T2)++T3          | listinduction 4-6,7-13,1
```

Formal treatment of generalised induction hypotheses

▷ Notice that we have not used the logical quantifiers in the inductive proof of ++-assoc.

▷ Consider the "catenate-reverse-to" function +<+ defined by

$$\frac{\mathbb{L}(T_1) \qquad \mathbb{L}(T_2)}{\mathbb{L}(T_1 \mathbin{+\!\!<\!\!+} T_2)}\ \mathrm{L{+}{<}{+}} \qquad \frac{\mathbb{L}(T_2)}{nil \mathbin{+\!\!<\!\!+} T_2 = T_2}\ {+}{<}{+}.0 \qquad \frac{\mathbb{L}(T_1) \qquad \mathbb{L}(T_2)}{(T : T_1) \mathbin{+\!\!<\!\!+} T_2 = T_1 \mathbin{+\!\!<\!\!+} (T : T_2)}\ {+}{<}{+}.1$$

▷ We want to prove (by induction on T_1) that

$$\mathbb{L}(T_1), \mathbb{L}(T_2) \vdash T_1 \mathbin{+\!\!<\!\!+} T_2 = rev(T_1) \mathbin{+\!\!+} T_2$$

▷ Using the list induction recipe blindly we start the proof as follows:

```
 1: | L(T1)                          | assumption
 2: | L(T2)                          | assumption
 3: | nil+<+T2                       |
 4: |   = T2                         | Unfold +<+'0,hyp
 5: |   = nil++T2                    | Fold ++'0,hyp
 6: |   = rev nil++T2                | Fold rev'0
 7: | | L(xs)                        | assumption
 8: | | xs+<+T2=rev xs++T2           | assumption
 9: | | x:xs+<+T2                    |
10: | |   = [xs+<+(x:T2)]            | Unfold +<+'1
    | |   . . .                      |
11: | |   = rev xs++(x:(T2))         |
12: | |   = rev xs++(x:(nil++T2))    | Fold ++'0,hyp
13: | |   = rev xs++(x:nil++T2)      | Fold ++'1,Lnil,hyp
14: | |   = (rev xs++(x:nil))++T2    | Unfold Theorem L(T1), L(T2), L(T3) ⊢ T1++(T2++T3)=(T1++T2)++T3,Lrev,hyp,L:,Lnil,hyp
15: | |   = rev(x:xs)++T2            | Fold rev'1,hyp
16: | L(T1)                          | hyp 1
17: | T1+<+T2=rev T1++T2             | listinduction 3-6,7-15,16
```

▷ But the proof stalls (why?) at the crux – on attempting use the induction hypothesis (8) to bridge 10 and 11

▷ At this point a lecturer or tutor usually uses the mantra: "there is nothing special about T_2"

▷ This *appears* to allow the proof to be completed, but the result is highly informal.

This problem can (*only*) be overcome by proving a more general result.

```
 1: | L(xs)                                      | assumption
 2: | | L(ys)                                    | assumption
 3: | | nil+<+ys                                 |
 4: | |   = ys                                   | Unfold +<+'0
 5: | |   = nil++ys                              | Fold ++'0,hyp
 6: | |   = rev nil++ys                          | Fold rev'0
 7: | L(ys)→nil+<+ys=rev nil++ys                 | ⊢→ 2-6
 8: | ∀ys.L(ys)→nil+<+ys=rev nil++ys             | ⊢∀ 7
 9: | | L(xs1)                                   | assumption
10: | | ∀ys.L(ys)→xs1+<+ys=rev xs1++ys           | assumption
11: | | | L(ys)                                  | assumption
12: | | | | L(x:(ys))→xs1+<+x:(ys)=rev xs1++x:(ys) | assumption
13: | | | | L(x:(ys))                            | L: 11
14: | | | | | xs1+<+x:(ys)=rev xs1++x:(ys)       | assumption
15: | | | | xs1+<+(x:ys)=rev xs1++(x:(ys))       | →⊢ 12,13,14-14
16: | | | x:xs1+<+ys                             |
17: | | |   = xs1+<+(x:ys)                       | Unfold +<+'1
18: | | |   = rev xs1++(x:(ys))                  | ∀⊢ 10,12-15
19: | | |   = rev xs1++(x:(nil++ys))             | Fold ++'0,hyp
20: | | |   = rev xs1++(x:nil++ys)               | Fold ++'1,Lnil,hyp
21: | | |   = (rev xs1++(x:nil))++ys             | Unfold Theorem L(T1), L(T2), L(T3) ⊢ T1++(T2++T3)=(T1++T2)++T3,Lrev,hyp,L:,Lnil,hyp
22: | | |   = rev(x:xs1)++ys                     | Fold rev'1,hyp,rev'1,hyp,rev'1,hyp,rev'1,hyp
23: | | L(ys)→x:xs1+<+ys=rev(x:xs1)++ys          | ⊢→ 11-22
24: | | ∀ys.L(ys)→x:xs1+<+ys=rev(x:xs1)++ys      | ⊢∀ 23
25: | ∀ys.L(ys)→xs+<+ys=rev xs++ys               | listinduction 8,9-24,1
26: L(xs)→(∀ys.L(ys)→xs+<+ys=rev xs++ys)         | ⊢→ 1-25
27: ∀xs.L(xs)→(∀ys.L(ys)→xs+<+ys=rev xs++ys)     | ⊢∀ 26
```

▷ The technique we use is to arrange to prove a *more general* result by induction on xs, namely:

$$\mathbb{L}(xs) \vdash \forall ys \cdot \mathbb{L}(ys) \to xs +\!\!\lessdot\!\!+ ys = rev(xs) +\!\!+ ys$$

which will give us the more general induction hypothesis we were seeking in the stalled proof.

▷ This is done by setting up a proof of the more general theorem

$$\vdash \forall xs \cdot \mathbb{L}(xs) \to \forall ys \cdot \mathbb{L}(ys) \to xs +\!\!\lessdot\!\!+ ys = rev(xs) +\!\!+ ys$$

▷ The result we originally set out to prove, namely:

$$\mathbb{L}(T_1), \mathbb{L}(T_2) \vdash T_1 +\!\!\lessdot\!\!+ T_2 = rev(T_1) +\!\!+ T_2$$

can now be established (by $\forall$-e followed by $\to$-e) from this theorem.

Contents

Extending Predicate Logic 1
Theories 1
Example: elementary group theory 2
Example: theory of Natural Numbers 7
Theories with several types 12
Theories with several types 12
Example: typed theory of natural numbers 13
Example: typed theory of heterogeneous lists 15
Indispensability of the logical quantifiers 18
Formal treatment of generalised induction hypotheses 18

Note 1: 4 ☞

The proof trees shown here were made with Jape, using a system including derived equational “rewrite” rules

$$\frac{\Gamma \vdash T_1 = T_2}{\Gamma \vdash T[T_1/\chi] = T[T_2/\chi]}\ \text{rewriteLR}$$

$$\frac{\Gamma \vdash T_1 = T_2}{\Gamma \vdash T[T_2/\chi] = T[T_1/\chi]}\ \text{rewriteRL}$$

These rules enable the direct use of (equational) laws to rewrite one side or the other of an equation while conducting a goal-directed proof search in an equational theory.

The rule $rewriteLR$ is used when we make an “Unfold” goal transformation, such as the move at the at the top right of the tree where we use the theorem $(T\otimes \sim T) = \iota$ to transform the proof goal $(T\otimes \sim T)\otimes T = T$ into the proof goal $\iota \otimes T = T$ – whence it is closed by application of the ι-id axiom.

The rule $rewriteRL$ is used when we make a “Fold” goal transformation, such as the move at the top left of the tree where the rule the goal $T\otimes\iota = T\otimes(\sim T\otimes T)$ is transformed by using the $\sim$ axiom $\sim T\otimes T = \iota$ to rewrite $T\otimes\iota$ as $T\otimes(\sim T\otimes T)$.

Note 2: 7 ☞

By convention we may omit the “$\Gamma\vdash$” in presenting P3 and P4; because they are free of antecedents. It should be clear that we cannot do so in presenting the induction rule, since one of its antecedents transforms the collection of hypotheses.

Note 3: 8 ☞

Here, and in future, we apply the convention of omitting the $\Gamma\vdash$ when presenting antecedent-free rules.

Note 4: A derived equality rule 9 ☞

In our inductive proof of associativity of $+$ we used the derived rule

$$\frac{T_1 = T_2 \qquad \varphi[T_2/\chi]}{\varphi[T_1/\chi]}\ \text{=-e}\leftarrow$$

to simplify the backward proof-search we did in Jape.

Note 5:
17 ☞
There are two different views of a single proof shown here. The proof was made with Jape set up to satisfy the typing laws automatically.