Miscellaneous FOL Examples

October 1, 2005

Contents

1 A simple formulation of First-Order Logic

theory *First-Order-Logic* **imports** *Pure* **begin**

The subsequent theory development illustrates single-sorted intuitionistic first-order logic with equality, formulated within the Pure framework. Actually this is not an example of Isabelle/FOL, but of Isabelle/Pure.

1.1 Syntax

typedecl i
typedecl o

judgment
$\mathit{Trueprop} :: o \Rightarrow \mathit{prop} \quad (\text{-}\ 5)$

1.2 Propositional logic

consts
$\mathit{false} :: o \quad (\bot)$

imp :: $o \Rightarrow o \Rightarrow o$ (**infixr** $\longrightarrow$ *25*)
conj :: $o \Rightarrow o \Rightarrow o$ (**infixr** $\wedge$ *35*)
disj :: $o \Rightarrow o \Rightarrow o$ (**infixr** $\vee$ *30*)

axioms
falseE [*elim*]: $\bot \Longrightarrow A$

impI [*intro*]: $(A \Longrightarrow B) \Longrightarrow A \longrightarrow B$
mp [*dest*]: $A \longrightarrow B \Longrightarrow A \Longrightarrow B$

conjI [*intro*]: $A \Longrightarrow B \Longrightarrow A \wedge B$
conjD1: $A \wedge B \Longrightarrow A$
conjD2: $A \wedge B \Longrightarrow B$

disjE [*elim*]: $A \vee B \Longrightarrow (A \Longrightarrow C) \Longrightarrow (B \Longrightarrow C) \Longrightarrow C$
disjI1 [*intro*]: $A \Longrightarrow A \vee B$
disjI2 [*intro*]: $B \Longrightarrow A \vee B$

theorem *conjE* [*elim*]: $A \wedge B \Longrightarrow (A \Longrightarrow B \Longrightarrow C) \Longrightarrow C$
proof −
assume *ab*: $A \wedge B$
assume *r*: $A \Longrightarrow B \Longrightarrow C$
show *C*
proof (*rule r*)
from *ab* **show** *A* **by** (*rule conjD1*)
from *ab* **show** *B* **by** (*rule conjD2*)
qed
qed

constdefs
true :: *o* (⊤)
$\top \equiv \bot \longrightarrow \bot$
not :: $o \Rightarrow o$ ($\neg$ - [*40*] *40*)
$\neg A \equiv A \longrightarrow \bot$
iff :: $o \Rightarrow o \Rightarrow o$ (**infixr** $\longleftrightarrow$ *25*)
$A \longleftrightarrow B \equiv (A \longrightarrow B) \wedge (B \longrightarrow A)$

theorem *trueI* [*intro*]: ⊤
proof (*unfold true-def*)
show $\bot \longrightarrow \bot$ **..**
qed

theorem *notI* [*intro*]: $(A \Longrightarrow \bot) \Longrightarrow \neg A$
proof (*unfold not-def*)
assume $A \Longrightarrow \bot$
thus $A \longrightarrow \bot$ **..**
qed

theorem *notE* [*elim*]: $\neg\ A \Longrightarrow A \Longrightarrow B$
proof (*unfold not-def*)
  assume $A \longrightarrow \bot$ **and** A
  hence $\bot$ **.. thus** B **..**
qed

theorem *iffI* [*intro*]: $(A \Longrightarrow B) \Longrightarrow (B \Longrightarrow A) \Longrightarrow A \longleftrightarrow B$
proof (*unfold iff-def*)
  assume $A \Longrightarrow B$ **hence** $A \longrightarrow B$ **..**
  moreover assume $B \Longrightarrow A$ **hence** $B \longrightarrow A$ **..**
  ultimately show $(A \longrightarrow B) \wedge (B \longrightarrow A)$ **..**
qed

theorem *iff1* [*elim*]: $A \longleftrightarrow B \Longrightarrow A \Longrightarrow B$
proof (*unfold iff-def*)
  assume $(A \longrightarrow B) \wedge (B \longrightarrow A)$
  hence $A \longrightarrow B$ **..**
  thus $A \Longrightarrow B$ **..**
qed

theorem *iff2* [*elim*]: $A \longleftrightarrow B \Longrightarrow B \Longrightarrow A$
proof (*unfold iff-def*)
  assume $(A \longrightarrow B) \wedge (B \longrightarrow A)$
  hence $B \longrightarrow A$ **..**
  thus $B \Longrightarrow A$ **..**
qed

1.3 Equality

consts
  equal :: $i \Rightarrow i \Rightarrow o$ (**infixl** = *50*)

axioms
  refl [*intro*]: $x = x$
  subst: $x = y \Longrightarrow P(x) \Longrightarrow P(y)$

theorem *trans* [*trans*]: $x = y \Longrightarrow y = z \Longrightarrow x = z$
  by (*rule subst*)

theorem *sym* [*sym*]: $x = y \Longrightarrow y = x$
proof −
  assume $x = y$
  from *this* **and** *refl* **show** $y = x$ **by** (*rule subst*)
qed

1.4 Quantifiers

consts
  All :: $(i \Rightarrow o) \Rightarrow o$ (**binder** $\forall$ *10*)
  Ex :: $(i \Rightarrow o) \Rightarrow o$ (**binder** $\exists$ *10*)

axioms
  allI [*intro*]: $(\bigwedge x.\ P(x)) \Longrightarrow \forall x.\ P(x)$
  allD [*dest*]: $\forall x.\ P(x) \Longrightarrow P(a)$

  exI [*intro*]: $P(a) \Longrightarrow \exists x.\ P(x)$
  exE [*elim*]: $\exists x.\ P(x) \Longrightarrow (\bigwedge x.\ P(x) \Longrightarrow C) \Longrightarrow C$

lemma $(\exists x.\ P(f(x))) \longrightarrow (\exists y.\ P(y))$
proof
  assume $\exists x.\ P(f(x))$
  thus $\exists y.\ P(y)$
  proof
    fix x **assume** $P(f(x))$
    thus *?thesis* **..**
  qed
qed

lemma $(\exists x.\ \forall y.\ R(x,\ y)) \longrightarrow (\forall y.\ \exists x.\ R(x,\ y))$
proof
  assume $\exists x.\ \forall y.\ R(x,\ y)$
  thus $\forall y.\ \exists x.\ R(x,\ y)$
  proof
    fix x **assume** a: $\forall y.\ R(x,\ y)$
    show *?thesis*
    proof
      fix y **from** a **have** $R(x,\ y)$ **..**
      thus $\exists x.\ R(x,\ y)$ **..**
    qed
  qed
qed

end

2 Natural numbers

theory *Natural-Numbers* **imports** *FOL* **begin**

Theory of the natural numbers: Peano's axioms, primitive recursion. (Modernized version of Larry Paulson's theory "Nat".)

typedecl *nat*
arities *nat* :: *term*

consts
  Zero :: *nat* (*0*)
  Suc :: *nat* => *nat*

```
  rec :: [nat, 'a, [nat, 'a] => 'a] => 'a

axioms
  induct [case-names 0 Suc, induct type: nat]:
    P(0) ==> (!!x. P(x) ==> P(Suc(x))) ==> P(n)
  Suc-inject: Suc(m) = Suc(n) ==> m = n
  Suc-neq-0: Suc(m) = 0 ==> R
  rec-0: rec(0, a, f) = a
  rec-Suc: rec(Suc(m), a, f) = f(m, rec(m, a, f))

lemma Suc-n-not-n: Suc(k) ≠ k
proof (induct k)
  show Suc(0) ≠ 0
  proof
    assume Suc(0) = 0
    thus False by (rule Suc-neq-0)
  qed
  fix n assume hyp: Suc(n) ≠ n
  show Suc(Suc(n)) ≠ Suc(n)
  proof
    assume Suc(Suc(n)) = Suc(n)
    hence Suc(n) = n by (rule Suc-inject)
    with hyp show False by contradiction
  qed
qed


constdefs
  add :: [nat, nat] => nat    (infixl + 60)
  m + n == rec(m, n, λx y. Suc(y))

lemma add-0 [simp]: 0 + n = n
  by (unfold add-def) (rule rec-0)

lemma add-Suc [simp]: Suc(m) + n = Suc(m + n)
  by (unfold add-def) (rule rec-Suc)

lemma add-assoc: (k + m) + n = k + (m + n)
  by (induct k) simp-all

lemma add-0-right: m + 0 = m
  by (induct m) simp-all

lemma add-Suc-right: m + Suc(n) = Suc(m + n)
  by (induct m) simp-all

lemma (!!n. f(Suc(n)) = Suc(f(n))) ==> f(i + j) = i + f(j)
proof -
  assume !!n. f(Suc(n)) = Suc(f(n))
```

thus *?thesis* **by** (*induct i*) *simp-all*
qed

end

3 Theory of the natural numbers: Peano's axioms, primitive recursion

theory *Nat*
imports *FOL*
begin

typedecl *nat*
arities *nat* :: *term*

consts
0 :: *nat* (*0*)
Suc :: *nat* => *nat*
rec :: [*nat*, *'a*, [*nat*,*'a*]=>*'a*] => *'a*
add :: [*nat*, *nat*] => *nat* (**infixl** + *60*)

axioms
induct: [| *P*(*0*); !!*x*. *P*(*x*) ==> *P*(*Suc*(*x*)) |] ==> *P*(*n*)
Suc-inject: *Suc*(*m*)=*Suc*(*n*) ==> *m*=*n*
Suc-neq-0: *Suc*(*m*)=*0* ==> *R*
rec-0: *rec*(*0*,*a*,*f*) = *a*
rec-Suc: *rec*(*Suc*(*m*), *a*, *f*) = *f*(*m*, *rec*(*m*,*a*,*f*))
add-def: *m*+*n* == *rec*(*m*, *n*, %*x y*. *Suc*(*y*))

ML ⟪ *use-legacy-bindings* (*the-context* ()) ⟫

end

4 First-Order Logic: PROLOG examples

theory *Prolog*
imports *FOL*
begin

typedecl *'a list*
arities *list* :: (*term*) *term*
consts
Nil :: *'a list*
Cons :: [*'a*, *'a list*]=> *'a list* (**infixr** : *60*)
app :: [*'a list*, *'a list*, *'a list*] => *o*
rev :: [*'a list*, *'a list*] => *o*

axioms
appNil: *app*(*Nil*,*ys*,*ys*)
appCons: *app*(*xs*,*ys*,*zs*) ==> *app*(*x*:*xs*, *ys*, *x*:*zs*)
revNil: *rev*(*Nil*,*Nil*)
revCons: [| *rev*(*xs*,*ys*); *app*(*ys*, *x*:*Nil*, *zs*) |] ==> *rev*(*x*:*xs*, *zs*)

ML ⟪ *use-legacy-bindings* (*the-context* ()) ⟫

end

5 Intuitionistic First-Order Logic

theory *Intuitionistic* **imports** *IFOL* **begin**

Metatheorem (for *propositional* formulae): P is classically provable iff $\neg\neg P$ is intuitionistically provable. Therefore $\neg P$ is classically provable iff it is intuitionistically provable.

Proof: Let Q be the conjuction of the propositions $A \vee \neg A$, one for each atom A in P. Now $\neg\neg Q$ is intuitionistically provable because $\neg\neg(A \vee \neg A)$ is and because double-negation distributes over conjunction. If P is provable classically, then clearly $Q \to P$ is provable intuitionistically, so $\neg\neg(Q \to P)$ is also provable intuitionistically. The latter is intuitionistically equivalent to $\neg\neg Q \to \neg\neg P$, hence to $\neg\neg P$, since $\neg\neg Q$ is intuitionistically provable. Finally, if P is a negation then $\neg\neg P$ is intuitionstically equivalent to P. [Andy Pitts]

lemma ~~(*P*&*Q*) <-> ~~*P* & ~~*Q*
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

lemma ~~ ((~*P* --> *Q*) --> (~*P* --> ~*Q*) --> *P*)
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

Double-negation does NOT distribute over disjunction

lemma ~~(*P*-->*Q*) <-> (~~*P* --> ~~*Q*)
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

lemma ~~~*P* <-> ~*P*
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

lemma ~~((*P* --> *Q* | *R*) --> (*P*-->*Q*) | (*P*-->*R*))
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

lemma (*P*<->*Q*) <-> (*Q*<->*P*)
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

lemma ((*P* --> (*Q* | (*Q*-->*R*))) --> *R*) --> *R*
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

lemma $(((G-->A) --> J) --> D --> E) --> (((H-->B)-->I)-->C-->J)$
$--> (A-->H) --> F --> G --> (((C-->B)-->I)-->D)-->(A-->C)$
$--> (((F-->A)-->B) --> I) --> E$
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

Lemmas for the propositional double-negation translation

lemma $P --> ~~P$
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

lemma $~~(~~P --> P)$
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

lemma $~~P \& ~~(P --> Q) --> ~~Q$
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

The following are classically but not constructively valid. The attempt to prove them terminates quickly!

lemma $((P-->Q) --> P) --> P$
apply (*tactic*⟪*IntPr.fast-tac 1*⟫ | −)
apply (*rule asm-rl*) — Checks that subgoals remain: proof failed.
oops

lemma $(P\&Q-->R) --> (P-->R) | (Q-->R)$
apply (*tactic*⟪*IntPr.fast-tac 1*⟫ | −)
apply (*rule asm-rl*) — Checks that subgoals remain: proof failed.
oops

5.1 de Bruijn formulae

de Bruijn formula with three predicates

lemma $((P<->Q) --> P\&Q\&R) \&$
$((Q<->R) --> P\&Q\&R) \&$
$((R<->P) --> P\&Q\&R) --> P\&Q\&R$
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

de Bruijn formula with five predicates

lemma $((P<->Q) --> P\&Q\&R\&S\&T) \&$
$((Q<->R) --> P\&Q\&R\&S\&T) \&$
$((R<->S) --> P\&Q\&R\&S\&T) \&$
$((S<->T) --> P\&Q\&R\&S\&T) \&$
$((T<->P) --> P\&Q\&R\&S\&T) --> P\&Q\&R\&S\&T$
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

Problem 1.1

lemma $(ALL\ x.\ EX\ y.\ ALL\ z.\ p(x) \&\ q(y) \&\ r(z)) <->$
$(ALL\ z.\ EX\ y.\ ALL\ x.\ p(x) \&\ q(y) \&\ r(z))$
by (*tactic*⟪*IntPr.best-dup-tac 1*⟫) — SLOW

Problem 3.1

lemma ~ (*EX x. ALL y. mem*(*y*,*x*) <-> ~ *mem*(*x*,*x*))
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

Problem 4.1: hopeless!

lemma (*ALL x. p*(*x*) --> *p*(*h*(*x*)) | *p*(*g*(*x*))) & (*EX x. p*(*x*)) & (*ALL x.* ~*p*(*h*(*x*)))
--> (*EX x. p*(*g*(*g*(*g*(*g*(*x*)))))))
oops

5.2 Intuitionistic FOL: propositional problems based on Pelletier.

1

lemma ~~((*P*-->*Q*) <-> (~*Q* --> ~*P*))
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

2

lemma ~~(~~*P* <-> *P*)
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

3

lemma ~(*P*-->*Q*) --> (*Q*-->*P*)
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

4

lemma ~~((~*P*-->*Q*) <-> (~*Q* --> *P*))
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

5

lemma ~~((*P*|*Q*-->*P*|*R*) --> *P*|(*Q*-->*R*))
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

6

lemma ~~(*P* | ~*P*)
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

7

lemma ~~(*P* | ~~~*P*)
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

8. Peirce's law

lemma ~~(((*P*-->*Q*) --> *P*) --> *P*)
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

9

lemma ((*P*|*Q*) & (~*P*|*Q*) & (*P*| ~*Q*)) --> ~ (~*P* | ~*Q*)

by (*tactic*⟪*IntPr.fast-tac 1*⟫)

10

lemma $(Q-->R) --> (R-->P\&Q) --> (P-->(Q|R)) --> (P<->Q)$
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

5.3 11. Proved in each direction (incorrectly, says Pelletier!!)

lemma $P<->P$
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

12. Dijkstra's law

lemma $\sim\sim(((P <-> Q) <-> R) <-> (P <-> (Q <-> R)))$
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

lemma $((P <-> Q) <-> R) --> \sim\sim(P <-> (Q <-> R))$
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

13. Distributive law

lemma $P \mid (Q \& R) <-> (P \mid Q) \& (P \mid R)$
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

14

lemma $\sim\sim((P <-> Q) <-> ((Q \mid \sim P) \& (\sim Q|P)))$
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

15

lemma $\sim\sim((P --> Q) <-> (\sim P \mid Q))$
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

16

lemma $\sim\sim((P-->Q) \mid (Q-->P))$
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

17

lemma $\sim\sim(((P \& (Q-->R))-->S) <-> ((\sim P \mid Q \mid S) \& (\sim P \mid \sim R \mid S)))$
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

Dijkstra's "Golden Rule"

lemma $(P\&Q) <-> P <-> Q <-> (P|Q)$
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

5.4 ****Examples with quantifiers****

5.5 The converse is classical in the following implications...

lemma $(EX\ x.\ P(x)-->Q) --> (ALL\ x.\ P(x)) --> Q$
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

lemma $((ALL\ x.\ P(x))--{>}Q)\ --{>}\ {\sim}\ (ALL\ x.\ P(x)\ \&\ {\sim}Q)$
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

lemma $((ALL\ x.\ {\sim}P(x))--{>}Q)\ \ --{>}\ \ {\sim}\ (ALL\ x.\ {\sim}\ (P(x)|Q))$
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

lemma $(ALL\ x.\ P(x))\ |\ Q\ \ --{>}\ \ (ALL\ x.\ P(x)\ |\ Q)$
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

lemma $(EX\ x.\ P\ --{>}\ Q(x))\ --{>}\ (P\ --{>}\ (EX\ x.\ Q(x)))$
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

5.6 The following are not constructively valid!

The attempt to prove them terminates quickly!

lemma $((ALL\ x.\ P(x))--{>}Q)\ --{>}\ (EX\ x.\ P(x)--{>}Q)$
apply (*tactic*⟪*IntPr.fast-tac 1*⟫ | −)
apply (*rule asm-rl*) — Checks that subgoals remain: proof failed.
oops

lemma $(P\ --{>}\ (EX\ x.\ Q(x)))\ --{>}\ (EX\ x.\ P--{>}Q(x))$
apply (*tactic*⟪*IntPr.fast-tac 1*⟫ | −)
apply (*rule asm-rl*) — Checks that subgoals remain: proof failed.
oops

lemma $(ALL\ x.\ P(x)\ |\ Q)\ --{>}\ ((ALL\ x.\ P(x))\ |\ Q)$
apply (*tactic*⟪*IntPr.fast-tac 1*⟫ | −)
apply (*rule asm-rl*) — Checks that subgoals remain: proof failed.
oops

lemma $(ALL\ x.\ {\sim}{\sim}P(x))\ --{>}\ {\sim}{\sim}(ALL\ x.\ P(x))$
apply (*tactic*⟪*IntPr.fast-tac 1*⟫ | −)
apply (*rule asm-rl*) — Checks that subgoals remain: proof failed.
oops

Classically but not intuitionistically valid. Proved by a bug in 1986!

lemma $EX\ x.\ Q(x)\ --{>}\ (ALL\ x.\ Q(x))$
apply (*tactic*⟪*IntPr.fast-tac 1*⟫ | −)
apply (*rule asm-rl*) — Checks that subgoals remain: proof failed.
oops

5.7 Hard examples with quantifiers

The ones that have not been proved are not known to be valid! Some will require quantifier duplication – not currently available

lemma $\sim\sim(EX\ y.\ ALL\ x.\ P(y)\text{-->}P(x))$
oops — NOT PROVED

19

lemma $\sim\sim(EX\ x.\ ALL\ y\ z.\ (P(y)\text{-->}Q(z))\ \text{-->}\ (P(x)\text{-->}Q(x)))$
oops — NOT PROVED

20

lemma $(ALL\ x\ y.\ EX\ z.\ ALL\ w.\ (P(x)\&Q(y)\text{-->}R(z)\&S(w)))$
$\text{-->}\ (EX\ x\ y.\ P(x)\ \&\ Q(y))\ \text{-->}\ (EX\ z.\ R(z))$
by $(tactic\langle\!\langle IntPr.fast\text{-}tac\ 1\rangle\!\rangle)$

21

lemma $(EX\ x.\ P\text{-->}Q(x))\ \&\ (EX\ x.\ Q(x)\text{-->}P)\ \text{-->}\ \sim\sim(EX\ x.\ P\text{<->}Q(x))$
oops — NOT PROVED; needs quantifier duplication

22

lemma $(ALL\ x.\ P\ \text{<->}\ Q(x))\ \ \text{-->}\ \ (P\ \text{<->}\ (ALL\ x.\ Q(x)))$
by $(tactic\langle\!\langle IntPr.fast\text{-}tac\ 1\rangle\!\rangle)$

23

lemma $\sim\sim\ ((ALL\ x.\ P\ |\ Q(x))\ \ \text{<->}\ \ (P\ |\ (ALL\ x.\ Q(x))))$
by $(tactic\langle\!\langle IntPr.fast\text{-}tac\ 1\rangle\!\rangle)$

24

lemma $\sim(EX\ x.\ S(x)\&Q(x))\ \&\ (ALL\ x.\ P(x)\ \text{-->}\ Q(x)|R(x))\ \&$
$(\sim(EX\ x.\ P(x))\ \text{-->}\ (EX\ x.\ Q(x)))\ \&\ (ALL\ x.\ Q(x)|R(x)\ \text{-->}\ S(x))$
$\text{-->}\ \sim\sim(EX\ x.\ P(x)\&R(x))$

Not clear why *fast-tac*, *best-tac*, *ASTAR* and *ITER-DEEPEN* all take forever

apply $(tactic\langle\!\langle\ IntPr.safe\text{-}tac\rangle\!\rangle)$
apply $(erule\ impE)$
apply $(tactic\langle\!\langle IntPr.fast\text{-}tac\ 1\rangle\!\rangle)$
by $(tactic\langle\!\langle IntPr.fast\text{-}tac\ 1\rangle\!\rangle)$

25

lemma $(EX\ x.\ P(x))\ \&$
$(ALL\ x.\ L(x)\ \text{-->}\ \sim\ (M(x)\ \&\ R(x)))\ \&$
$(ALL\ x.\ P(x)\ \text{-->}\ (M(x)\ \&\ L(x)))\ \&$
$((ALL\ x.\ P(x)\text{-->}Q(x))\ |\ (EX\ x.\ P(x)\&R(x)))$
$\text{-->}\ (EX\ x.\ Q(x)\&P(x))$
by $(tactic\langle\!\langle IntPr.fast\text{-}tac\ 1\rangle\!\rangle)$

26

lemma $(\sim\sim(EX\ x.\ p(x))\ \text{<->}\ \sim\sim(EX\ x.\ q(x)))\ \&$
$(ALL\ x.\ ALL\ y.\ p(x)\ \&\ q(y)\ \text{-->}\ (r(x)\ \text{<->}\ s(y)))$
$\text{-->}\ ((ALL\ x.\ p(x)\text{-->}r(x))\ \text{<->}\ (ALL\ x.\ q(x)\text{-->}s(x)))$

oops — NOT PROVED

27

lemma $(EX\ x.\ P(x)\ \&\ \sim Q(x))\ \&$
$\quad (ALL\ x.\ P(x) \dashrightarrow R(x))\ \&$
$\quad (ALL\ x.\ M(x)\ \&\ L(x) \dashrightarrow P(x))\ \&$
$\quad ((EX\ x.\ R(x)\ \&\ \sim\ Q(x)) \dashrightarrow (ALL\ x.\ L(x) \dashrightarrow \sim\ R(x)))$
$\dashrightarrow (ALL\ x.\ M(x) \dashrightarrow \sim L(x))$
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

28. AMENDED

lemma $(ALL\ x.\ P(x) \dashrightarrow (ALL\ x.\ Q(x)))\ \&$
$\quad (\sim\sim(ALL\ x.\ Q(x)|R(x)) \dashrightarrow (EX\ x.\ Q(x)\&S(x)))\ \&$
$\quad (\sim\sim(EX\ x.\ S(x)) \dashrightarrow (ALL\ x.\ L(x) \dashrightarrow M(x)))$
$\dashrightarrow (ALL\ x.\ P(x)\ \&\ L(x) \dashrightarrow M(x))$
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

29. Essentially the same as Principia Mathematica *11.71

lemma $(EX\ x.\ P(x))\ \&\ (EX\ y.\ Q(y))$
$\dashrightarrow ((ALL\ x.\ P(x)\dashrightarrow R(x))\ \&\ (ALL\ y.\ Q(y)\dashrightarrow S(y)) \quad <\!\!\dashrightarrow$
$\quad (ALL\ x\ y.\ P(x)\ \&\ Q(y) \dashrightarrow R(x)\ \&\ S(y)))$
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

30

lemma $(ALL\ x.\ (P(x)\ |\ Q(x)) \dashrightarrow \sim\ R(x))\ \&$
$\quad (ALL\ x.\ (Q(x) \dashrightarrow \sim\ S(x)) \dashrightarrow P(x)\ \&\ R(x))$
$\dashrightarrow (ALL\ x.\ \sim\sim S(x))$
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

31

lemma $\sim(EX\ x.\ P(x)\ \&\ (Q(x)\ |\ R(x)))\ \&$
$\quad (EX\ x.\ L(x)\ \&\ P(x))\ \&$
$\quad (ALL\ x.\ \sim\ R(x) \dashrightarrow M(x))$
$\dashrightarrow (EX\ x.\ L(x)\ \&\ M(x))$
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

32

lemma $(ALL\ x.\ P(x)\ \&\ (Q(x)|R(x))\dashrightarrow S(x))\ \&$
$\quad (ALL\ x.\ S(x)\ \&\ R(x) \dashrightarrow L(x))\ \&$
$\quad (ALL\ x.\ M(x) \dashrightarrow R(x))$
$\dashrightarrow (ALL\ x.\ P(x)\ \&\ M(x) \dashrightarrow L(x))$
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

33

lemma $(ALL\ x.\ \sim\sim(P(a)\ \&\ (P(x)\dashrightarrow P(b))\dashrightarrow P(c))) \ <\!\!\dashrightarrow$
$\quad (ALL\ x.\ \sim\sim((\sim P(a)\ |\ P(x)\ |\ P(c))\ \&\ (\sim P(a)\ |\ \sim P(b)\ |\ P(c))))$
apply (*tactic*⟪*IntPr.best-tac 1*⟫)
done

36

lemma (*ALL x. EX y. J(x,y)*) &
    (*ALL x. EX y. G(x,y)*) &
    (*ALL x y. J(x,y) | G(x,y) --> (ALL z. J(y,z) | G(y,z) --> H(x,z)*))
 --> (*ALL x. EX y. H(x,y)*)
by (*tactic⟪IntPr.fast-tac 1⟫*)

37

lemma (*ALL z. EX w. ALL x. EX y.*
        ~~(*P(x,z)-->P(y,w)*) & *P(y,z)* & (*P(y,w) --> (EX u. Q(u,w)*))) &
      (*ALL x z. ~P(x,z) --> (EX y. Q(y,z)*)) &
      (~~(*EX x y. Q(x,y)*) --> (*ALL x. R(x,x)*))
   --> ~~(*ALL x. EX y. R(x,y)*)
oops — NOT PROVED

39

lemma ~ (*EX x. ALL y. F(y,x) <-> ~F(y,y)*)
by (*tactic⟪IntPr.fast-tac 1⟫*)

40. AMENDED

lemma (*EX y. ALL x. F(x,y) <-> F(x,x)*) -->
          ~(*ALL x. EX y. ALL z. F(z,y) <-> ~ F(z,x)*)
by (*tactic⟪IntPr.fast-tac 1⟫*)

44

lemma (*ALL x. f(x) -->*
          (*EX y. g(y) & h(x,y) & (EX y. g(y) & ~ h(x,y)*))) &
          (*EX x. j(x) & (ALL y. g(y) --> h(x,y)*))
          --> (*EX x. j(x) & ~f(x)*))
by (*tactic⟪IntPr.fast-tac 1⟫*)

48

lemma (*a=b | c=d*) & (*a=c | b=d*) --> *a=d | b=c*
by (*tactic⟪IntPr.fast-tac 1⟫*)

51

lemma (*EX z w. ALL x y. P(x,y) <-> (x=z & y=w)*) -->
    (*EX z. ALL x. EX w. (ALL y. P(x,y) <-> y=w*) <-> *x=z*)
by (*tactic⟪IntPr.fast-tac 1⟫*)

52

Almost the same as 51.

lemma (*EX z w. ALL x y. P(x,y) <-> (x=z & y=w)*) -->
    (*EX w. ALL y. EX z. (ALL x. P(x,y) <-> x=z*) <-> *y=w*)
by (*tactic⟪IntPr.fast-tac 1⟫*)

56

lemma (*ALL x. (EX y. P(y) & x=f(y)) --> P(x)) <-> (ALL x. P(x) --> P(f(x)))*
by (*tactic⟪IntPr.fast-tac 1⟫*)

57

lemma *P(f(a,b), f(b,c)) & P(f(b,c), f(a,c)) &*
(ALL x y z. P(x,y) & P(y,z) --> P(x,z)) --> P(f(a,b), f(a,c))
by (*tactic⟪IntPr.fast-tac 1⟫*)

60

lemma *ALL x. P(x,f(x)) <-> (EX y. (ALL z. P(z,y) --> P(z,f(x))) & P(x,y))*
by (*tactic⟪IntPr.fast-tac 1⟫*)

end

6 Classical Predicate Calculus Problems

theory *Classical* **imports** *FOL* **begin**

lemma (*P --> Q | R) --> (P-->Q) | (P-->R)*
by *blast*

If and only if

lemma (*P<->Q) <-> (Q<->P)*
by *blast*

lemma *~ (P <-> ~P)*
by *blast*

Sample problems from F. J. Pelletier, Seventy-Five Problems for Testing Automatic Theorem Provers, J. Automated Reasoning 2 (1986), 191-216. Errata, JAR 4 (1988), 236-236.

The hardest problems – judging by experience with several theorem provers, including matrix ones – are 34 and 43.

6.1 Pelletier's examples

1

lemma (*P-->Q) <-> (~Q --> ~P)*
by *blast*

2

lemma *~ ~ P <-> P*

by *blast*

3

lemma $^{\sim}(P--{>}Q)$ $--{>}$ $(Q--{>}P)$
by *blast*

4

lemma $(^{\sim}P--{>}Q)$ $<{->}$ $(^{\sim}Q$ $--{>}$ $P)$
by *blast*

5

lemma $((P|Q)--{>}(P|R))$ $--{>}$ $(P|(Q--{>}R))$
by *blast*

6

lemma P | $^{\sim}$ P
by *blast*

7

lemma P | $^{\sim}$ $^{\sim}$ $^{\sim}$ P
by *blast*

8. Peirce's law

lemma $((P--{>}Q)$ $--{>}$ $P)$ $--{>}$ P
by *blast*

9

lemma $((P|Q)$ & $(^{\sim}P|Q)$ & $(P|$ $^{\sim}Q))$ $--{>}$ $^{\sim}$ $(^{\sim}P$ | $^{\sim}Q)$
by *blast*

10

lemma $(Q--{>}R)$ & $(R--{>}P\&Q)$ & $(P--{>}Q|R)$ $--{>}$ $(P{<}{->}Q)$
by *blast*

11. Proved in each direction (incorrectly, says Pelletier!!)

lemma $P{<}{->}P$
by *blast*

12. "Dijkstra's law"

lemma $((P$ $<{->}$ $Q)$ $<{->}$ $R)$ $<{->}$ $(P$ $<{->}$ $(Q$ $<{->}$ $R))$
by *blast*

13. Distributive law

lemma P | $(Q$ & $R)$ $<{->}$ $(P$ | $Q)$ & $(P$ | $R)$
by *blast*

14

lemma $(P <-> Q) <-> ((Q \mid \sim P) \;\&\; (\sim Q \mid P))$
by *blast*

15

lemma $(P --> Q) <-> (\sim P \mid Q)$
by *blast*

16

lemma $(P --> Q) \mid (Q --> P)$
by *blast*

17

lemma $((P \;\&\; (Q --> R)) --> S) <-> ((\sim P \mid Q \mid S) \;\&\; (\sim P \mid \sim R \mid S))$
by *blast*

6.2 Classical Logic: examples with quantifiers

lemma $(\forall x.\ P(x) \;\&\; Q(x)) <-> (\forall x.\ P(x)) \;\&\; (\forall x.\ Q(x))$
by *blast*

lemma $(\exists x.\ P --> Q(x)) <-> (P --> (\exists x.\ Q(x)))$
by *blast*

lemma $(\exists x.\ P(x) --> Q) <-> (\forall x.\ P(x)) --> Q$
by *blast*

lemma $(\forall x.\ P(x)) \mid Q <-> (\forall x.\ P(x) \mid Q)$
by *blast*

Discussed in Avron, Gentzen-Type Systems, Resolution and Tableaux, JAR 10 (265-281), 1993. Proof is trivial!

lemma $\sim((\exists x. \sim P(x)) \;\&\; ((\exists x.\ P(x)) \mid (\exists x.\ P(x) \;\&\; Q(x))) \;\&\; \sim (\exists x.\ P(x)))$
by *blast*

6.3 Problems requiring quantifier duplication

Theorem B of Peter Andrews, Theorem Proving via General Matings, JACM 28 (1981).

lemma $(\exists x.\ \forall y.\ P(x) <-> P(y)) --> ((\exists x.\ P(x)) <-> (\forall y.\ P(y)))$
by *blast*

Needs multiple instantiation of ALL.

lemma $(\forall x.\ P(x) --> P(f(x))) \;\&\; P(d) --> P(f(f(f(d))))$
by *blast*

Needs double instantiation of the quantifier

lemma $\exists x.\ P(x) --> P(a) \;\&\; P(b)$

by *blast*

lemma $\exists z.\ P(z) \longrightarrow (\forall x.\ P(x))$
by *blast*

lemma $\exists x.\ (\exists y.\ P(y)) \longrightarrow P(x)$
by *blast*

V. Lifschitz, What Is the Inverse Method?, JAR 5 (1989), 1–23. NOT PROVED

lemma $\exists x\ x'.\ \forall y.\ \exists z\ z'.$
$(\sim P(y,y) \mid P(x,x) \mid \sim S(z,x))\ \&$
$(S(x,y) \mid \sim S(y,z) \mid Q(z',z'))\ \&$
$(Q(x',y) \mid \sim Q(y,z') \mid S(x',x'))$
oops

6.4 Hard examples with quantifiers

18

lemma $\exists y.\ \forall x.\ P(y) \longrightarrow P(x)$
by *blast*

19

lemma $\exists x.\ \forall y\ z.\ (P(y) \longrightarrow Q(z)) \longrightarrow (P(x) \longrightarrow Q(x))$
by *blast*

20

lemma $(\forall x\ y.\ \exists z.\ \forall w.\ (P(x) \& Q(y) \longrightarrow R(z) \& S(w)))$
$\longrightarrow (\exists x\ y.\ P(x)\ \&\ Q(y)) \longrightarrow (\exists z.\ R(z))$
by *blast*

21

lemma $(\exists x.\ P \longrightarrow Q(x))\ \&\ (\exists x.\ Q(x) \longrightarrow P) \longrightarrow (\exists x.\ P <-> Q(x))$
by *blast*

22

lemma $(\forall x.\ P <-> Q(x)) \longrightarrow (P <-> (\forall x.\ Q(x)))$
by *blast*

23

lemma $(\forall x.\ P \mid Q(x)) <-> (P \mid (\forall x.\ Q(x)))$
by *blast*

24

lemma $\sim(\exists x.\ S(x) \& Q(x))\ \&\ (\forall x.\ P(x) \longrightarrow Q(x) \mid R(x))\ \&$
$(\sim(\exists x.\ P(x)) \longrightarrow (\exists x.\ Q(x)))\ \&\ (\forall x.\ Q(x) \mid R(x) \longrightarrow S(x))$
$\longrightarrow (\exists x.\ P(x) \& R(x))$

by *blast*

25

lemma $(\exists x.\ P(x))\ \&$
$\quad (\forall x.\ L(x) \dashrightarrow \sim (M(x)\ \&\ R(x)))\ \&$
$\quad (\forall x.\ P(x) \dashrightarrow (M(x)\ \&\ L(x)))\ \&$
$\quad ((\forall x.\ P(x) \dashrightarrow Q(x)) \mid (\exists x.\ P(x) \& R(x)))$
$\dashrightarrow (\exists x.\ Q(x) \& P(x))$
by *blast*

26

lemma $((\exists x.\ p(x)) <\!\!\dashrightarrow (\exists x.\ q(x)))\ \&$
$\quad (\forall x.\ \forall y.\ p(x)\ \&\ q(y) \dashrightarrow (r(x) <\!\!\dashrightarrow s(y)))$
$\dashrightarrow ((\forall x.\ p(x) \dashrightarrow r(x)) <\!\!\dashrightarrow (\forall x.\ q(x) \dashrightarrow s(x)))$
by *blast*

27

lemma $(\exists x.\ P(x)\ \&\ \sim Q(x))\ \&$
$\quad (\forall x.\ P(x) \dashrightarrow R(x))\ \&$
$\quad (\forall x.\ M(x)\ \&\ L(x) \dashrightarrow P(x))\ \&$
$\quad ((\exists x.\ R(x)\ \&\ \sim Q(x)) \dashrightarrow (\forall x.\ L(x) \dashrightarrow \sim R(x)))$
$\dashrightarrow (\forall x.\ M(x) \dashrightarrow \sim L(x))$
by *blast*

28. AMENDED

lemma $(\forall x.\ P(x) \dashrightarrow (\forall x.\ Q(x)))\ \&$
$\quad ((\forall x.\ Q(x) | R(x)) \dashrightarrow (\exists x.\ Q(x) \& S(x)))\ \&$
$\quad ((\exists x.\ S(x)) \dashrightarrow (\forall x.\ L(x) \dashrightarrow M(x)))$
$\dashrightarrow (\forall x.\ P(x)\ \&\ L(x) \dashrightarrow M(x))$
by *blast*

29. Essentially the same as Principia Mathematica *11.71

lemma $(\exists x.\ P(x))\ \&\ (\exists y.\ Q(y))$
$\dashrightarrow ((\forall x.\ P(x) \dashrightarrow R(x))\ \&\ (\forall y.\ Q(y) \dashrightarrow S(y)) \quad <\!\!\dashrightarrow$
$\quad (\forall x\ y.\ P(x)\ \&\ Q(y) \dashrightarrow R(x)\ \&\ S(y)))$
by *blast*

30

lemma $(\forall x.\ P(x) \mid Q(x) \dashrightarrow \sim R(x))\ \&$
$\quad (\forall x.\ (Q(x) \dashrightarrow \sim S(x)) \dashrightarrow P(x)\ \&\ R(x))$
$\dashrightarrow (\forall x.\ S(x))$
by *blast*

31

lemma $\sim(\exists x.\ P(x)\ \&\ (Q(x) \mid R(x)))\ \&$
$\quad (\exists x.\ L(x)\ \&\ P(x))\ \&$
$\quad (\forall x.\ \sim R(x) \dashrightarrow M(x))$
$\dashrightarrow (\exists x.\ L(x)\ \&\ M(x))$

by *blast*

32

lemma $(\forall x.\ P(x)\ \&\ (Q(x)|R(x))\text{--}{>}S(x))\ \&$
$(\forall x.\ S(x)\ \&\ R(x)\ \text{--}{>}\ L(x))\ \&$
$(\forall x.\ M(x)\ \text{--}{>}\ R(x))$
$\text{--}{>}\ (\forall x.\ P(x)\ \&\ M(x)\ \text{--}{>}\ L(x))$
by *blast*

33

lemma $(\forall x.\ P(a)\ \&\ (P(x)\text{--}{>}P(b))\text{--}{>}P(c))\ \ <\text{->}$
$(\forall x.\ (\sim P(a)\ |\ P(x)\ |\ P(c))\ \&\ (\sim P(a)\ |\ \sim P(b)\ |\ P(c)))$
by *blast*

34 AMENDED (TWICE!!). Andrews's challenge

lemma $((\exists x.\ \forall y.\ p(x)\ <\text{->}\ p(y))\ \ <\text{->}$
$((\exists x.\ q(x))\ <\text{->}\ (\forall y.\ p(y))))\ \ <\text{->}$
$((\exists x.\ \forall y.\ q(x)\ <\text{->}\ q(y))\ \ <\text{->}$
$((\exists x.\ p(x))\ <\text{->}\ (\forall y.\ q(y))))$
by *blast*

35

lemma $\exists x\ y.\ P(x,y)\ \text{--}{>}\ \ (\forall u\ v.\ P(u,v))$
by *blast*

36

lemma $(\forall x.\ \exists y.\ J(x,y))\ \&$
$(\forall x.\ \exists y.\ G(x,y))\ \&$
$(\forall x\ y.\ J(x,y)\ |\ G(x,y)\ \text{--}{>}\ (\forall z.\ J(y,z)\ |\ G(y,z)\ \text{--}{>}\ H(x,z)))$
$\text{--}{>}\ (\forall x.\ \exists y.\ H(x,y))$
by *blast*

37

lemma $(\forall z.\ \exists w.\ \forall x.\ \exists y.$
$(P(x,z)\text{--}{>}P(y,w))\ \&\ P(y,z)\ \&\ (P(y,w)\ \text{--}{>}\ (\exists u.\ Q(u,w))))\ \&$
$(\forall x\ z.\ \sim P(x,z)\ \text{--}{>}\ (\exists y.\ Q(y,z)))\ \&$
$((\exists x\ y.\ Q(x,y))\ \text{--}{>}\ (\forall x.\ R(x,x)))$
$\text{--}{>}\ (\forall x.\ \exists y.\ R(x,y))$
by *blast*

38

lemma $(\forall x.\ p(a)\ \&\ (p(x)\ \text{--}{>}\ (\exists y.\ p(y)\ \&\ r(x,y)))\ \text{--}{>}$
$(\exists z.\ \exists w.\ p(z)\ \&\ r(x,w)\ \&\ r(w,z)))\ \ <\text{->}$
$(\forall x.\ (\sim p(a)\ |\ p(x)\ |\ (\exists z.\ \exists w.\ p(z)\ \&\ r(x,w)\ \&\ r(w,z)))\ \&$
$(\sim p(a)\ |\ \sim(\exists y.\ p(y)\ \&\ r(x,y))\ |$
$(\exists z.\ \exists w.\ p(z)\ \&\ r(x,w)\ \&\ r(w,z))))$
by *blast*

39

lemma ~ (∃ *x*. ∀ *y*. *F*(*y*,*x*) <-> ~*F*(*y*,*y*))
by *blast*

40. AMENDED

lemma (∃ *y*. ∀ *x*. *F*(*x*,*y*) <-> *F*(*x*,*x*)) -->
~(∀ *x*. ∃ *y*. ∀ *z*. *F*(*z*,*y*) <-> ~ *F*(*z*,*x*))
by *blast*

41

lemma (∀ *z*. ∃ *y*. ∀ *x*. *f*(*x*,*y*) <-> *f*(*x*,*z*) & ~ *f*(*x*,*x*))
--> ~ (∃ *z*. ∀ *x*. *f*(*x*,*z*))
by *blast*

42

lemma ~ (∃ *y*. ∀ *x*. *p*(*x*,*y*) <-> ~ (∃ *z*. *p*(*x*,*z*) & *p*(*z*,*x*)))
by *blast*

43

lemma (∀ *x*. ∀ *y*. *q*(*x*,*y*) <-> (∀ *z*. *p*(*z*,*x*) <-> *p*(*z*,*y*)))
--> (∀ *x*. ∀ *y*. *q*(*x*,*y*) <-> *q*(*y*,*x*))
by *blast*

44

lemma (∀ *x*. *f*(*x*) --> (∃ *y*. *g*(*y*) & *h*(*x*,*y*) & (∃ *y*. *g*(*y*) & ~ *h*(*x*,*y*)))) &
(∃ *x*. *j*(*x*) & (∀ *y*. *g*(*y*) --> *h*(*x*,*y*)))
--> (∃ *x*. *j*(*x*) & ~*f*(*x*))
by *blast*

45

lemma (∀ *x*. *f*(*x*) & (∀ *y*. *g*(*y*) & *h*(*x*,*y*) --> *j*(*x*,*y*))
--> (∀ *y*. *g*(*y*) & *h*(*x*,*y*) --> *k*(*y*))) &
~ (∃ *y*. *l*(*y*) & *k*(*y*)) &
(∃ *x*. *f*(*x*) & (∀ *y*. *h*(*x*,*y*) --> *l*(*y*))
& (∀ *y*. *g*(*y*) & *h*(*x*,*y*) --> *j*(*x*,*y*)))
--> (∃ *x*. *f*(*x*) & ~ (∃ *y*. *g*(*y*) & *h*(*x*,*y*)))
by *blast*

46

lemma (∀ *x*. *f*(*x*) & (∀ *y*. *f*(*y*) & *h*(*y*,*x*) --> *g*(*y*)) --> *g*(*x*)) &
((∃ *x*. *f*(*x*) & ~*g*(*x*)) -->
(∃ *x*. *f*(*x*) & ~*g*(*x*) & (∀ *y*. *f*(*y*) & ~*g*(*y*) --> *j*(*x*,*y*)))) &
(∀ *x* *y*. *f*(*x*) & *f*(*y*) & *h*(*x*,*y*) --> ~*j*(*y*,*x*))
--> (∀ *x*. *f*(*x*) --> *g*(*x*))
by *blast*

6.5 Problems (mainly) involving equality or functions

48

lemma $(a{=}b \mid c{=}d)$ & $(a{=}c \mid b{=}d)$ $-->$ $a{=}d \mid b{=}c$
by *blast*

49 NOT PROVED AUTOMATICALLY. Hard because it involves substitution for Vars the type constraint ensures that x,y,z have the same type as a,b,u.

lemma $(\exists\, x\ y{::}'a.\ \forall\, z.\ z{=}x \mid z{=}y)$ & $P(a)$ & $P(b)$ & $a^{\sim}{=}b$
$-->$ $(\forall\, u{::}'a.\ P(u))$
apply *safe*
apply (*rule-tac* $x = a$ **in** *allE*, *assumption*)
apply (*rule-tac* $x = b$ **in** *allE*, *assumption*, *fast*)
— blast's treatment of equality can't do it
done

50. (What has this to do with equality?)

lemma $(\forall\, x.\ P(a,x) \mid (\forall\, y.\ P(x,y)))$ $-->$ $(\exists\, x.\ \forall\, y.\ P(x,y))$
by *blast*

51

lemma $(\exists\, z\ w.\ \forall\, x\ y.\ P(x,y) <->\ (x{=}z\ \&\ y{=}w))$ $-->$
$(\exists\, z.\ \forall\, x.\ \exists\, w.\ (\forall\, y.\ P(x,y) <->\ y{=}w) <->\ x{=}z)$
by *blast*

52

Almost the same as 51.

lemma $(\exists\, z\ w.\ \forall\, x\ y.\ P(x,y) <->\ (x{=}z\ \&\ y{=}w))$ $-->$
$(\exists\, w.\ \forall\, y.\ \exists\, z.\ (\forall\, x.\ P(x,y) <->\ x{=}z) <->\ y{=}w)$
by *blast*

55

Non-equational version, from Manthey and Bry, CADE-9 (Springer, 1988). fast DISCOVERS who killed Agatha.

lemma $lives(agatha)$ & $lives(butler)$ & $lives(charles)$ &
$(killed(agatha,agatha) \mid killed(butler,agatha) \mid killed(charles,agatha))$ &
$(\forall\, x\ y.\ killed(x,y)$ $-->$ $hates(x,y)$ & $^{\sim}richer(x,y))$ &
$(\forall\, x.\ hates(agatha,x)$ $-->$ $^{\sim}hates(charles,x))$ &
$(hates(agatha,agatha)$ & $hates(agatha,charles))$ &
$(\forall\, x.\ lives(x)$ & $^{\sim}richer(x,agatha)$ $-->$ $hates(butler,x))$ &
$(\forall\, x.\ hates(agatha,x)$ $-->$ $hates(butler,x))$ &
$(\forall\, x.\ ^{\sim}hates(x,agatha) \mid ^{\sim}hates(x,butler) \mid ^{\sim}hates(x,charles))$ $-->$
$killed(?who,agatha)$
by *fast* — MUCH faster than blast

56

lemma $(\forall x.\ (\exists y.\ P(y)\ \&\ x{=}f(y)) \longrightarrow P(x)) \longleftrightarrow (\forall x.\ P(x) \longrightarrow P(f(x)))$
by *blast*

57

lemma $P(f(a,b),\ f(b,c))\ \&\ P(f(b,c),\ f(a,c))\ \&$
$(\forall x\ y\ z.\ P(x,y)\ \&\ P(y,z) \longrightarrow P(x,z)) \longrightarrow P(f(a,b),\ f(a,c))$
by *blast*

58 NOT PROVED AUTOMATICALLY

lemma $(\forall x\ y.\ f(x){=}g(y)) \longrightarrow (\forall x\ y.\ f(f(x)){=}f(g(y)))$
by (*slow elim*: *subst-context*)

59

lemma $(\forall x.\ P(x) \longleftrightarrow \sim P(f(x))) \longrightarrow (\exists x.\ P(x)\ \&\ \sim P(f(x)))$
by *blast*

60

lemma $\forall x.\ P(x,f(x)) \longleftrightarrow (\exists y.\ (\forall z.\ P(z,y) \longrightarrow P(z,f(x)))\ \&\ P(x,y))$
by *blast*

62 as corrected in JAR 18 (1997), page 135

lemma $(\forall x.\ p(a)\ \&\ (p(x) \longrightarrow p(f(x))) \longrightarrow p(f(f(x))))\ \longleftrightarrow$
$(\forall x.\ (\sim p(a) \mid p(x) \mid p(f(f(x))))\ \&$
$(\sim p(a) \mid \sim p(f(x)) \mid p(f(f(x)))))$
by *blast*

From Davis, Obvious Logical Inferences, IJCAI-81, 530-531 fast indeed copes!

lemma $(\forall x.\ F(x)\ \&\ \sim G(x) \longrightarrow (\exists y.\ H(x,y)\ \&\ J(y)))\ \&$
$(\exists x.\ K(x)\ \&\ F(x)\ \&\ (\forall y.\ H(x,y) \longrightarrow K(y)))\ \&$
$(\forall x.\ K(x) \longrightarrow \sim G(x)) \longrightarrow (\exists x.\ K(x)\ \&\ J(x))$
by *fast*

From Rudnicki, Obvious Inferences, JAR 3 (1987), 383-393. It does seem obvious!

lemma $(\forall x.\ F(x)\ \&\ \sim G(x) \longrightarrow (\exists y.\ H(x,y)\ \&\ J(y)))\ \&$
$(\exists x.\ K(x)\ \&\ F(x)\ \&\ (\forall y.\ H(x,y) \longrightarrow K(y)))\ \&$
$(\forall x.\ K(x) \longrightarrow \sim G(x)) \longrightarrow (\exists x.\ K(x) \longrightarrow \sim G(x))$
by *fast*

Halting problem: Formulation of Li Dafa (AAR Newsletter 27, Oct 1994.) author U. Egly

lemma $((\exists x.\ A(x)\ \&\ (\forall y.\ C(y) \longrightarrow (\forall z.\ D(x,y,z)))) \longrightarrow$
$(\exists w.\ C(w)\ \&\ (\forall y.\ C(y) \longrightarrow (\forall z.\ D(w,y,z)))))$
&

```
 (∀w. C(w) & (∀u. C(u) --> (∀v. D(w,u,v))) -->
      (∀y z.
         (C(y) &  P(y,z) --> Q(w,y,z) & OO(w,g)) &
         (C(y) & ~P(y,z) --> Q(w,y,z) & OO(w,b))))
 &
 (∀w. C(w) &
  (∀y z.
     (C(y) & P(y,z) --> Q(w,y,z) & OO(w,g)) &
     (C(y) & ~P(y,z) --> Q(w,y,z) & OO(w,b))) -->
  (∃v. C(v) &
       (∀y. ((C(y) & Q(w,y,y)) & OO(w,g) --> ~P(v,y)) &
            ((C(y) & Q(w,y,y)) & OO(w,b) --> P(v,y) & OO(v,b)))))
 -->
  ~ (∃x. A(x) & (∀y. C(y) --> (∀z. D(x,y,z))))
by (tactic⟪Blast.depth-tac (claset ()) 12 1⟫)
  — Needed because the search for depths below 12 is very slow
```

Halting problem II: credited to M. Bruschi by Li Dafa in JAR 18(1), p.105

```
lemma ((∃x. A(x) & (∀y. C(y) --> (∀z. D(x,y,z)))) -->
  (∃w. C(w) & (∀y. C(y) --> (∀z. D(w,y,z)))))
 &
 (∀w. C(w) & (∀u. C(u) --> (∀v. D(w,u,v))) -->
      (∀y z.
         (C(y) &  P(y,z) --> Q(w,y,z) & OO(w,g)) &
         (C(y) & ~P(y,z) --> Q(w,y,z) & OO(w,b))))
 &
 ((∃w. C(w) & (∀y. (C(y) &  P(y,y) --> Q(w,y,y) & OO(w,g)) &
                   (C(y) & ~P(y,y) --> Q(w,y,y) & OO(w,b))))
  -->
  (∃v. C(v) & (∀y. (C(y) &  P(y,y) --> P(v,y) & OO(v,g)) &
                   (C(y) & ~P(y,y) --> P(v,y) & OO(v,b)))))
 -->
 ((∃v. C(v) & (∀y. (C(y) &  P(y,y) --> P(v,y) & OO(v,g)) &
                   (C(y) & ~P(y,y) --> P(v,y) & OO(v,b))))
  -->
  (∃u. C(u) & (∀y. (C(y) &  P(y,y) --> ~P(u,y)) &
                   (C(y) & ~P(y,y) --> P(u,y) & OO(u,b)))))
  -->
  ~ (∃x. A(x) & (∀y. C(y) --> (∀z. D(x,y,z))))
by blast
```

Challenge found on info-hol

```
lemma ∀x. ∃v w. ∀y z. P(x) & Q(y) --> (P(v) | R(w)) & (R(z) --> Q(v))
by blast
```

Attributed to Lewis Carroll by S. G. Pulman. The first or last assumption can be deleted.

```
lemma (∀x. honest(x) & industrious(x) --> healthy(x)) &
     ~ (∃x. grocer(x) & healthy(x)) &
```

$(\forall x.\ industrious(x)\ \&\ grocer(x) \longrightarrow honest(x))\ \&$
$(\forall x.\ cyclist(x) \longrightarrow industrious(x))\ \&$
$(\forall x.\ \sim healthy(x)\ \&\ cyclist(x) \longrightarrow \sim honest(x))$
$\longrightarrow (\forall x.\ grocer(x) \longrightarrow \sim cyclist(x))$
by *blast*

end

7 First-Order Logic: the 'if' example

theory *If* **imports** *FOL* **begin**

constdefs
$if :: [o,o,o]{=}{>}o$
$if(P,Q,R) == P\&Q \mid \sim P\&R$

lemma *ifI*:
$[\!|\ P \Longrightarrow Q;\ \sim P \Longrightarrow R\ |\!] \Longrightarrow if(P,Q,R)$
apply (*simp add*: *if-def*, *blast*)
done

lemma *ifE*:
$[\!|\ if(P,Q,R);\ [\!|\ P;\ Q\ |\!] \Longrightarrow S;\ [\!|\ \sim P;\ R\ |\!] \Longrightarrow S\ |\!] \Longrightarrow S$
apply (*simp add*: *if-def*, *blast*)
done

lemma *if-commute*: $if(P, if(Q,A,B), if(Q,C,D)) \longleftrightarrow if(Q, if(P,A,C), if(P,B,D))$
apply (*rule iffI*)
apply (*erule ifE*)
apply (*erule ifE*)
apply (*rule ifI*)
apply (*rule ifI*)
oops

Trying again from the beginning in order to use *blast*

declare *ifI* [*intro*!]
declare *ifE* [*elim*!]

lemma *if-commute*: $if(P, if(Q,A,B), if(Q,C,D)) \longleftrightarrow if(Q, if(P,A,C), if(P,B,D))$
by *blast*

lemma $if(if(P,Q,R),\ A,\ B) \longleftrightarrow if(P,\ if(Q,A,B),\ if(R,A,B))$
by *blast*

Trying again from the beginning in order to prove from the definitions

lemma *if*(*if*(*P*,*Q*,*R*), *A*, *B*) <-> *if*(*P*, *if*(*Q*,*A*,*B*), *if*(*R*,*A*,*B*))
by (*simp add*: *if-def*, *blast*)

An invalid formula. High-level rules permit a simpler diagnosis

lemma *if*(*if*(*P*,*Q*,*R*), *A*, *B*) <-> *if*(*P*, *if*(*Q*,*A*,*B*), *if*(*R*,*B*,*A*))
apply *auto*
— The next step will fail unless subgoals remain
apply (*tactic all-tac*)
oops

Trying again from the beginning in order to prove from the definitions

lemma *if*(*if*(*P*,*Q*,*R*), *A*, *B*) <-> *if*(*P*, *if*(*Q*,*A*,*B*), *if*(*R*,*B*,*A*))
apply (*simp add*: *if-def*, *auto*)
— The next step will fail unless subgoals remain
apply (*tactic all-tac*)
oops

end

theory *NatClass*
imports *FOL*
begin

This is an abstract version of theory *Nat*. Instead of axiomatizing a single type *nat* we define the class of all these types (up to isomorphism).

Note: The *rec* operator had to be made *monomorphic*, because class axioms may not contain more than one type variable.

consts
0 :: *'a* (*0*)
Suc :: *'a* => *'a*
rec :: [*'a*, *'a*, [*'a*, *'a*] => *'a*] => *'a*

axclass
nat < *term*
induct: [| *P*(*0*); !!*x*. *P*(*x*) ==> *P*(*Suc*(*x*)) |] ==> *P*(*n*)
Suc-inject: *Suc*(*m*) = *Suc*(*n*) ==> *m* = *n*
Suc-neq-0: *Suc*(*m*) = *0* ==> *R*
rec-0: *rec*(*0*, *a*, *f*) = *a*
rec-Suc: *rec*(*Suc*(*m*), *a*, *f*) = *f*(*m*, *rec*(*m*, *a*, *f*))

constdefs
add :: [*'a*::*nat*, *'a*] => *'a* (**infixl** + *60*)
m + *n* == *rec*(*m*, *n*, %*x y*. *Suc*(*y*))

ML ⟪ *use-legacy-bindings* (*the-context* ()) ⟫

ML ⟪ *open nat-class* ⟫

end

8 Theory for examples of simplification and induction on the natural numbers

theory *Nat2*
imports *FOL*
begin

typedecl *nat*
arities *nat* :: *term*

consts
succ :: *nat* => *nat*
pred :: *nat* => *nat*
0 :: *nat* (*0*)
add :: [*nat*,*nat*] => *nat* (**infixr** + *90*)
lt :: [*nat*,*nat*] => *o* (**infixr** < *70*)
leq :: [*nat*,*nat*] => *o* (**infixr** <= *70*)

axioms
pred-0: *pred*(*0*) = *0*
pred-succ: *pred*(*succ*(*m*)) = *m*

plus-0: *0*+*n* = *n*
plus-succ: *succ*(*m*)+*n* = *succ*(*m*+*n*)

nat-distinct1: ~ *0*=*succ*(*n*)
nat-distinct2: ~ *succ*(*m*)=*0*
succ-inject: *succ*(*m*)=*succ*(*n*) <-> *m*=*n*

leq-0: *0* <= *n*
leq-succ-succ: *succ*(*m*)<=*succ*(*n*) <-> *m*<=*n*
leq-succ-0: ~ *succ*(*m*) <= *0*

lt-0-succ: *0* < *succ*(*n*)
lt-succ-succ: *succ*(*m*)<*succ*(*n*) <-> *m*<*n*
lt-0: ~ *m* < *0*

nat-ind: [| *P*(*0*); *ALL n*. *P*(*n*)-->*P*(*succ*(*n*)) |] ==> *All*(*P*)

ML ⟪ *use-legacy-bindings* (*the-context* ()) ⟫

end

9 Examples of simplification and induction on lists

theory *List*
imports *Nat2*
begin

typedecl *'a list*
arities *list* :: (*term*) *term*

consts
hd :: *'a list => 'a*
tl :: *'a list => 'a list*
forall :: *['a list, 'a => o] => o*
len :: *'a list => nat*
at :: *['a list, nat] => 'a*
Nil :: *'a list* ([])
Cons :: *['a, 'a list] => 'a list* (**infixr** . *80*)
app :: *['a list, 'a list] => 'a list* (**infixr** ++ *70*)

axioms
list-ind: [| *P*([]); *ALL x l. P(l)-->P(x . l)* |] ==> *All*(*P*)

forall-cong:
[| *l = l'*; !!*x*. *P*(*x*)<->*P'*(*x*) |] ==> *forall*(*l*,*P*) <-> *forall*(*l'*,*P'*)

list-distinct1: ~[] = *x . l*
list-distinct2: ~*x . l* = []

list-free: *x . l = x' . l'* <-> *x=x'* & *l=l'*

app-nil: []++*l* = *l*
app-cons: (*x . l*)++*l'* = *x* .(*l*++*l'*)
tl-eq: *tl*(*m . q*) = *q*
hd-eq: *hd*(*m . q*) = *m*

forall-nil: *forall*([],*P*)
forall-cons: *forall*(*x . l*,*P*) <-> *P*(*x*) & *forall*(*l*,*P*)

len-nil: *len*([]) = *0*
len-cons: *len*(*m . q*) = *succ*(*len*(*q*))

at-0: *at*(*m . q*,*0*) = *m*
at-succ: *at*(*m . q*,*succ*(*n*)) = *at*(*q*,*n*)

ML ⟪ *use-legacy-bindings* (*the-context* ()) ⟫

end

10 Example of Declaring an Oracle

theory *IffOracle*
imports *FOL*
begin

10.1 Oracle declaration

This oracle makes tautologies of the form $P \leftrightarrow P \leftrightarrow P \leftrightarrow P$. The length is specified by an integer, which is checked to be even and positive.

```
oracle iff-oracle (int) = ⟪
  let
    fun mk-iff 1 = Var ((P, 0), FOLogic.oT)
      | mk-iff n = FOLogic.iff $ Var ((P, 0), FOLogic.oT) $ mk-iff (n - 1);
  in
    fn thy => fn n =>
      if n > 0 andalso n mod 2 = 0
      then FOLogic.mk-Trueprop (mk-iff n)
      else raise Fail (iff-oracle: ^ string-of-int n)
  end
⟫
```

10.2 Oracle as low-level rule

```
ML ⟪ iff-oracle (the-context ()) 2 ⟫
ML ⟪ iff-oracle (the-context ()) 10 ⟫
ML ⟪ #der (Thm.rep-thm it) ⟫
```

These oracle calls had better fail

```
ML ⟪
  (iff-oracle (the-context ()) 5; raise ERROR)
    handle Fail - => warning Oracle failed, as expected
⟫

ML ⟪
  (iff-oracle (the-context ()) 1; raise ERROR)
    handle Fail - => warning Oracle failed, as expected
⟫
```

10.3 Oracle as proof method

```
method-setup iff = ⟪
  Method.simple-args Args.nat (fn n => fn ctxt =>
    Method.SIMPLE-METHOD
      (HEADGOAL (Tactic.rtac (iff-oracle (ProofContext.theory-of ctxt) n))
        handle Fail - => no-tac))
⟫ iff oracle
```

lemma *A <-> A*
by (*iff 2*)

lemma *A <-> A <-> A <-> A <-> A <-> A <-> A <-> A <-> A <-> A*
by (*iff 10*)

lemma *A <-> A <-> A <-> A <-> A*
apply (*iff 5*)*?*
oops

lemma *A*
apply (*iff 1*)*?*
oops

end

11 Test of Locale Interpretation

theory *LocaleTest*
imports *FOL*
begin

ML ⟪ *set quick-and-dirty* ⟫
ML ⟪ *set Toplevel.debug* ⟫
ML ⟪ *set show-hyps* ⟫
ML ⟪ *set show-sorts* ⟫

ML ⟪
```
fun check-thm name = let
  val thy = the-context ();
  val thm = get-thm thy (Name name);
  val {prop, hyps, ...} = rep-thm thm;
  val prems = Logic.strip-imp-prems prop;
  val - = if null hyps then ()
      else error (Theorem " ^ quote name ^ " has meta hyps.\n ^
        Consistency check of locales package failed.);
  val - = if null prems then ()
      else error (Theorem " ^ quote name ^ " has premises.\n ^
        Consistency check of locales package failed.);
in () end;
```
⟫

12 Context Elements and Locale Expressions

Naming convention for global objects: prefixes L and l

12.1 Renaming with Syntax

locale (**open**) *LS* = *var mult* +
  assumes *mult*(*x*, *y*) = *mult*(*y*, *x*)

print-locale *LS*

locale *LS′* = *LS mult* (**infixl** ∗∗ *60*)

print-locale *LS′*

locale *LT* = *var mult* (**infixl** ∗∗ *60*) +
  assumes *x* ∗∗ *y* = *y* ∗∗ *x*

locale *LU* = *LT mult* (**infixl** ∗∗ *60*) + *LT add* (**infixl** ++ *55*) + *var h* +
  assumes *hom*: *h*(*x* ∗∗ *y*) = *h*(*x*) ++ *h*(*y*)

locale *LV* = *LU* - *add*

12.2 Constrains

locale *LZ* = **fixes** *a* (**structure**)
locale *LZ′* = *LZ* +
  constrains *a* :: *′a* => *′b*
  assumes *a* (*x* :: *′a*) = *a* (*y*)
print-locale *LZ′*

13 Interpretation

Naming convention for global objects: prefixes I and i

interpretation input syntax

locale *IL*
locale *IM* = **fixes** *a* **and** *b* **and** *c*

interpretation *test* [*simp*]: *IL* + *IM a b c* [*x y z*] **.**

print-interps *IL*
print-interps *IM*

interpretation *test* [*simp*]: *IL* **print-interps** *IM* **.**

interpretation *IL* **.**

Processing of locale expression

locale *IA* = **fixes** *a* **assumes** *asm-A*: *a* = *a*

locale (**open**) *IB* = **fixes** *b* **assumes** *asm-B* [*simp*]: *b* = *b*

locale *IC* = *IA* + *IB* + **assumes** *asm-C*: *c* = *c*

locale *ID* = *IA* + *IB* + **fixes** *d* **defines** *def-D*: *d* == (*a* = *b*)

theorem (**in** *IA*)
  includes *ID*
  shows *True* **..**

theorem (**in** *ID*) *True* **..**

typedecl *i*
arities *i* :: *term*

interpretation *i1*: *IC* [*X*::*i* *Y*::*i*] **by** (*auto intro*: *IA.intro IC-axioms.intro*)

print-interps *IA*

thm *i1.a.asm-A* **thm** *LocaleTest.i1.a.asm-A*
thm *i1.asm-A* **thm** *LocaleTest.i1.asm-A*

ML ⟪ *check-thm i1.a.asm-A* ⟫

interpretation *IC* [*W*::*i* *Z*::*i*] **.**
interpretation *IC* [*W*::'*a* *Z*::*i*] **by** (*auto intro*: *IA.intro IC-axioms.intro*)

print-interps *IA*

thm *asm-C* **thm** *a-b.asm-C* **thm** *LocaleTest.a-b.asm-C* **thm** *LocaleTest.a-b.asm-C*

ML ⟪ *check-thm asm-C* ⟫

interpretation *i2*: *ID* [*X* *Y*::*i* *Y* = *X*] **by** (*simp add*: *eq-commute*)

print-interps *IA*
print-interps *ID*

interpretation *i3*: *ID* [*X* *Y*::*i*] **.**

print-interps *IA*
print-interps *IB*
print-interps *IC*
print-interps *ID*

interpretation *i10*: *ID* + *ID* *a′* *b′* *d′* [*X* *Y*::*i* - *u* *v*::*i* -] **.**

corollary (**in** *ID*) *th-x*: *True* **..**

thm *i2.th-x* **thm** *i3.th-x*

ML ⟪ *check-thm i2.th-x*; *check-thm i3.th-x* ⟫

lemma (**in** *ID*) *th-y*: *d* == (*a* = *b*) **.**

thm *i2.th-y* **thm** *i3.th-y*

ML ⟪ *check-thm i2.th-y*; *check-thm i3.th-y* ⟫

lemmas (**in** *ID*) *th-z* = *th-y*

thm *i2.th-z*

ML ⟪ *check-thm i2.th-z* ⟫

13.1 Interpretation in Proof Contexts

locale *IF* = **fixes** *f* **assumes** *asm-F*: *f* & *f* −−> *f*

theorem *True*
proof −
fix *alpha*::*i* **and** *beta*::*′a* **and** *gamma*::*o*

have *alpha-A*: *IA*(*alpha*) **by** (*auto intro*: *IA.intro*)
interpret *i5*: *IA* [*alpha*] **.**
print-interps *IA*
interpret *i6*: *IC* [*alpha beta*] **by** (*auto intro*: *IC-axioms.intro*)
print-interps *IA*
print-interps *IC*
interpret *i11*: *IF* [*gamma*] **by** (*fast intro*: *IF.intro*)
thm *i11.asm-F*

qed *rule*

theorem (**in** *IA*) *True*
proof −
  print-interps *IA*
  fix *beta* **and** *gamma*
  interpret *i9*: *ID* [*a beta* -]

    apply − **apply** (*rule refl*) **apply** *assumption* **done**
qed *rule*

ML ⟪ *reset show-sorts* ⟫

locale *IE* = **fixes** *e* **defines** *e-def*: *e*(*x*) == *x* & *x*
  notes *e-def2* = *e-def*

lemma (**in** *IE*) *True* **thm** *e-def* **by** *fast*

interpretation *i7*: *IE* [%*x*. *x*] **by** *simp*

thm *i7.e-def2*

ML ⟪ *check-thm i7.e-def2* ⟫

locale *IE′* = **fixes** *e* **defines** *e-def*: *e* == (%*x*. *x* & *x*)
  notes *e-def2* = *e-def*

interpretation *i7′*: *IE′* [(%*x*. *x*)] **by** *simp*

thm *i7′.e-def2*

ML ⟪ *check-thm i7′.e-def2* ⟫

locale (**open**) *IG* = **fixes** *g* **assumes** *asm-G*: *g* −−> *x*
  notes *asm-G2* = *asm-G*

interpretation *i8*: *IG* [*False*] **by** *fast*

thm *i8.asm-G2*

ML ⟪ *check-thm i8.asm-G2* ⟫

Locale without assumptions

locale *IL1* = **notes** *rev-conjI* [*intro*] = *conjI* [*THEN iffD1* [*OF conj-commute*]]

lemma [| *P*; *Q* |] ==> *P* & *Q*
proof −
  interpret *my*: *IL1* **.**

No chained fact required.

  assume *Q* **and** *P*

order reversed

  then show *P* & *Q* **..**

Applies ⟦*?P1*; *?Q1*⟧ ⟹ *?Q1* ∧ *?P1*.

qed

locale *IL11* = **notes** *rev-conjI* = *conjI* [*THEN iffD1* [*OF conj-commute*]]

lemma [| *P*; *Q* |] ==> *P* & *Q*
proof −
  interpret [*intro*]: *IL11* **.**

Attribute supplied at instantiation.

  assume *Q* **and** *P*
  then show *P* & *Q* **..**
qed

13.2 Simple locale with assumptions

consts *ibin* :: [*i*, *i*] => *i* (**infixl** # *60*)

axioms *i-assoc*: (*x* # *y*) # *z* = *x* # (*y* # *z*)
  i-comm: *x* # *y* = *y* # *x*

locale *IL2* =
  fixes *OP* (**infixl** + *60*)
  assumes *assoc*: (*x* + *y*) + *z* = *x* + (*y* + *z*)
    and *comm*: *x* + *y* = *y* + *x*

lemma (**in** *IL2*) *lcomm*: *x* + (*y* + *z*) = *y* + (*x* + *z*)
proof −
  have *x* + (*y* + *z*) = (*x* + *y*) + *z* **by** (*simp add*: *assoc*)
  also have ... = (*y* + *x*) + *z* **by** (*simp add*: *comm*)
  also have ... = *y* + (*x* + *z*) **by** (*simp add*: *assoc*)
  finally show *?thesis* **.**
qed

lemmas (**in** *IL2*) *AC* = *comm assoc lcomm*

lemma (*x*::*i*) # *y* # *z* # *w* = *y* # *x* # *w* # *z*
proof −

interpret *my*: *IL2* [*op #*] **by** (*rule IL2.intro* [*of op #*, *OF i-assoc i-comm*])
show *?thesis* **by** (*simp only*: *my.OP.AC*)
qed

13.3 Nested locale with assumptions

locale *IL3* =
fixes *OP* (**infixl** + *60*)
assumes *assoc*: (*x* + *y*) + *z* = *x* + (*y* + *z*)

locale *IL4* = *IL3* +
assumes *comm*: *x* + *y* = *y* + *x*

lemma (**in** *IL4*) *lcomm*: *x* + (*y* + *z*) = *y* + (*x* + *z*)
proof −
have *x* + (*y* + *z*) = (*x* + *y*) + *z* **by** (*simp add*: *assoc*)
also have ... = (*y* + *x*) + *z* **by** (*simp add*: *comm*)
also have ... = *y* + (*x* + *z*) **by** (*simp add*: *assoc*)
finally show *?thesis* **.**
qed

lemmas (**in** *IL4*) *AC* = *comm assoc lcomm*

lemma (*x*::*i*) # *y* # *z* # *w* = *y* # *x* # *w* # *z*
proof −
interpret *my*: *IL4* [*op #*]
by (*auto intro*: *IL3.intro IL4-axioms.intro i-assoc i-comm*)
show *?thesis* **by** (*simp only*: *my.OP.AC*)
qed

Locale with definition

This example is admittedly not very creative :-)

locale *IL5* = *IL4* + *var A* +
defines *A-def*: *A* == *True*

lemma (**in** *IL5*) *lem*: *A*
by (*unfold A-def*) *rule*

lemma *IL5*(*op #*) ==> *True*
proof −
assume *IL5*(*op #*)
then interpret *IL5* [*op #*] **by** (*auto intro*: *IL5.axioms*)
show *?thesis* **by** (*rule lem*)
qed

Interpretation in a context with target

lemma (**in** *IL4*)
fixes *A* (**infixl** $ *60*)

assumes *A*: *IL4*(*A*)
shows (*x*::*i*) $ *y* $ *z* $ *w* = *y* $ *x* $ *w* $ *z*
proof −
from *A* **interpret** *A*: *IL4* [*A*] **by** (*auto intro*: *IL4.axioms*)
show *?thesis* **by** (*simp only*: *A.OP.AC*)
qed

14 Interpretation in Locales

Naming convention for global objects: prefixes R and r

locale (**open**) *Rsemi* = *var prod* (**infixl** ** *65*) +
assumes *assoc*: (*x* ** *y*) ** *z* = *x* ** (*y* ** *z*)

locale (**open**) *Rlgrp* = *Rsemi* + *var one* + *var inv* +
assumes *lone*: *one* ** *x* = *x*
and *linv*: *inv*(*x*) ** *x* = *one*

lemma (**in** *Rlgrp*) *lcancel*:
x ** *y* = *x* ** *z* <−> *y* = *z*
proof
assume *x* ** *y* = *x* ** *z*
then have *inv*(*x*) ** *x* ** *y* = *inv*(*x*) ** *x* ** *z* **by** (*simp add*: *assoc*)
then show *y* = *z* **by** (*simp add*: *lone linv*)
qed *simp*

locale (**open**) *Rrgrp* = *Rsemi* + *var one* + *var inv* +
assumes *rone*: *x* ** *one* = *x*
and *rinv*: *x* ** *inv*(*x*) = *one*

lemma (**in** *Rrgrp*) *rcancel*:
y ** *x* = *z* ** *x* <−> *y* = *z*
proof
assume *y* ** *x* = *z* ** *x*
then have *y* ** (*x* ** *inv*(*x*)) = *z* ** (*x* ** *inv*(*x*))
by (*simp add*: *assoc* [*symmetric*])
then show *y* = *z* **by** (*simp add*: *rone rinv*)
qed *simp*

interpretation *Rlgrp* < *Rrgrp*
proof −
{
fix *x*
have *inv*(*x*) ** *x* ** *one* = *inv*(*x*) ** *x* **by** (*simp add*: *linv lone*)
then show *x* ** *one* = *x* **by** (*simp add*: *assoc lcancel*)
}
note *rone* = *this*
{
fix *x*

```
    have inv(x) ** x ** inv(x) = inv(x) ** one
      by (simp add: linv lone rone)
    then show x ** inv(x) = one by (simp add: assoc lcancel)
  }
qed



print-locale Rlgrp



lemma (in Rlgrp)
  y ** x = z ** x <-> y = z
  apply (rule rcancel)
  print-interps Rrgrp thm lcancel rcancel
  done



interpretation Rrgrp < Rlgrp
  proof -
    {
      fix x
      have one ** (x ** inv(x)) = x ** inv(x) by (simp add: rinv rone)
      then show one ** x = x by (simp add: assoc [symmetric] rcancel)
    }
    note lone = this
    {
      fix x
      have inv(x) ** (x ** inv(x)) = one ** inv(x)
        by (simp add: rinv lone rone)
      then show inv(x) ** x = one by (simp add: assoc [symmetric] rcancel)
    }
  qed



print-locale! Rrgrp
print-locale! Rlgrp



locale RA5 = var A + var B + var C + var D + var E +
  assumes eq: A <-> B <-> C <-> D <-> E

interpretation RA5 < RA5 - - D E C
print-facts
print-interps RA5
```

using *A-B-C-D-E.eq* **apply** (*blast intro*: *RA5.intro*) **done**

interpretation *RA5* < *RA5 C - E - A*
print-facts
print-interps *RA5*
using *A-B-C-D-E.eq* **apply** (*blast intro*: *RA5.intro*) **done**

interpretation *RA5* < *RA5 B C A - -*
print-facts
print-interps *RA5*
using *A-B-C-D-E.eq* **apply** (*blast intro*: *RA5.intro*) **done**

lemma (**in** *RA5*) *True*
print-facts
print-interps *RA5*
..

interpretation *RA5* < *RA5 - C D B -* **.**

locale (**open**) *RA1* = *var A* + *var B* + **assumes** *p*: *A* <-> *B*
locale (**open**) *RA2* = *var A* + *var B* + **assumes** *q*: *A* & *B* |~ *A* & ~ *B*
locale (**open**) *RA3* = *var A* + *var B* + **assumes** *r*: (*A* --> *B*) & (*B* --> *A*)

interpretation *RA1* < *RA2*
print-facts
using *p* **apply** *fast* **done**
interpretation *RA2* < *RA3*
print-facts
using *q* **apply** *fast* **done**
interpretation *RA3* < *RA1*
print-facts
using *r* **apply** *fast* **done**

locale (**open**) *RB1* = *var A* + *var B* + **assumes** *p*: *A* <-> *B*
locale (**open**) *RB2* = *var A* + *var B* + **assumes** *q*: *A* & *B* |~ *A* & ~ *B*
locale (**open**) *RB3* = *var A* + *var B* + **assumes** *r*: (*A* --> *B*) & (*B* --> *A*)

interpretation *RB1* < *RB2*
print-facts
using *p* **apply** *fast* **done**
interpretation *RB3* < *RB1*
print-facts
using *r* **apply** *fast* **done**
interpretation *RB2* < *RB3*

```
  print-facts
  using q apply fast done

lemma (in RB1) True
  print-facts
  ..



locale Rpsemi = var prod (infixl ** 65) +
  assumes assoc: (x ** y) ** z = x ** (y ** z)

locale Rplgrp = Rpsemi + var one + var inv +
  assumes lone: one ** x = x
    and linv: inv(x) ** x = one

lemma (in Rplgrp) lcancel:
  x ** y = x ** z <-> y = z
proof
  assume x ** y = x ** z
  then have inv(x) ** x ** y = inv(x) ** x ** z by (simp add: assoc)
  then show y = z by (simp add: lone linv)
qed simp

locale Rprgrp = Rpsemi + var one + var inv +
  assumes rone: x ** one = x
    and rinv: x ** inv(x) = one

lemma (in Rprgrp) rcancel:
  y ** x = z ** x <-> y = z
proof
  assume y ** x = z ** x
  then have y ** (x ** inv(x)) = z ** (x ** inv(x))
    by (simp add: assoc [symmetric])
  then show y = z by (simp add: rone rinv)
qed simp

interpretation Rplgrp < Rprgrp
  proof (rule Rprgrp-axioms.intro)
    {
      fix x
      have inv(x) ** x ** one = inv(x) ** x by (simp add: linv lone)
      then show x ** one = x by (simp add: assoc lcancel)
    }
    note rone = this
    {
      fix x
      have inv(x) ** x ** inv(x) = inv(x) ** one
```

```
      by (simp add: linv lone rone)
    then show x ** inv(x) = one by (simp add: assoc lcancel)
  }
qed
```

```
print-locale Rplgrp
```

```
lemma (in Rplgrp)
  y ** x = z ** x <-> y = z
  apply (rule rcancel)
  print-interps Rprgrp thm lcancel rcancel
  done
```

```
interpretation Rprgrp < Rplgrp
  proof (rule Rplgrp-axioms.intro)
    {
      fix x
      have one ** (x ** inv(x)) = x ** inv(x) by (simp add: rinv rone)
      then show one ** x = x by (simp add: assoc [symmetric] rcancel)
    }
    note lone = this
    {
      fix x
      have inv(x) ** (x ** inv(x)) = one ** inv(x)
        by (simp add: rinv lone rone)
      then show inv(x) ** x = one by (simp add: assoc [symmetric] rcancel)
    }
  qed
```

```
print-locale Rprgrp
print-locale Rplgrp
```

14.1 Interaction of Interpretation in Theories and Locales: in Locale, then in Theory

```
consts
  rone :: i
  rinv :: i => i

axioms
  r-one : rone # x = x
```

```
  r-inv : rinv(x) # x = rone

interpretation Rbool: Rlgrp [op # rone rinv]
proof −
  fix x y z
  {
    show (x # y) # z = x # (y # z) by (rule i-assoc)
  next
    show rone # x = x by (rule r-one)
  next
    show rinv(x) # x = rone by (rule r-inv)
  }
qed



print-interps Rrgrp
print-interps Rlgrp

lemma y # x = z # x <−> y = z by (rule Rbool.rcancel)



lemma (in Rrgrp) new-cancel:
  b ** a = c ** a <−> b = c
  by (rule rcancel)

thm Rbool.new-cancel

ML ⟪ check-thm Rbool.new-cancel ⟫

lemma b # a = c # a <−> b = c by (rule Rbool.new-cancel)
```

14.2 Interaction of Interpretation in Theories and Locales: in Theory, then in Locale

```
locale Rqsemi = var prod (infixl ** 65) +
  assumes assoc: (x ** y) ** z = x ** (y ** z)

locale Rqlgrp = Rqsemi + var one + var inv +
  assumes lone: one ** x = x
    and linv: inv(x) ** x = one

lemma (in Rqlgrp) lcancel:
  x ** y = x ** z <−> y = z
proof
  assume x ** y = x ** z
  then have inv(x) ** x ** y = inv(x) ** x ** z by (simp add: assoc)
  then show y = z by (simp add: lone linv)
```

```
qed simp

locale Rqrgrp = Rqsemi + var one + var inv +
  assumes rone: x ** one = x
    and rinv: x ** inv(x) = one

lemma (in Rqrgrp) rcancel:
  y ** x = z ** x <-> y = z
proof
  assume y ** x = z ** x
  then have y ** (x ** inv(x)) = z ** (x ** inv(x))
    by (simp add: assoc [symmetric])
  then show y = z by (simp add: rone rinv)
qed simp

interpretation Rqrgrp < Rprgrp
proof -
  show Rpsemi(op **)
    apply (rule Rpsemi.intro) apply (rule assoc) done
next
  show Rprgrp-axioms(op **, one, inv)
    apply (rule Rprgrp-axioms.intro) apply (rule rone) apply (rule rinv) done
qed

interpretation R2: Rqlgrp [op # rone rinv]
proof -
  apply-end (rule Rqsemi.intro)
  fix x y z
  {
    show (x # y) # z = x # (y # z) by (rule i-assoc)
  next
  apply-end (rule Rqlgrp-axioms.intro)
    show rone # x = x by (rule r-one)
  next
    show rinv(x) # x = rone by (rule r-inv)
  }
qed

print-interps Rqsemi
print-interps Rqlgrp
print-interps Rplgrp


interpretation Rqlgrp < Rqrgrp
  proof (rule Rqrgrp-axioms.intro)
    {
      fix x
      have inv(x) ** x ** one = inv(x) ** x by (simp add: linv lone)
      then show x ** one = x by (simp add: assoc lcancel)
```

```
  }
  note rone = this
  {
    fix x
    have inv(x) ** x ** inv(x) = inv(x) ** one
      by (simp add: linv lone rone)
    then show x ** inv(x) = one by (simp add: assoc lcancel)
  }
qed

print-interps! Rqrgrp
print-interps! Rpsemi
print-interps! Rprgrp
print-interps! Rplgrp
thm R2.rcancel
thm R2.lcancel

ML ⟨⟨ check-thm R2.rcancel; check-thm R2.lcancel ⟩⟩
```

14.3 Generation of Witness Theorems for Transitive Interpretations

```
locale Rtriv = var x +
  assumes x: x = x

locale Rtriv2 = var x + var y +
  assumes x: x = x and y: y = y

interpretation Rtriv2 < Rtriv x
  apply (rule Rtriv.intro)
  apply (rule x)
  done

interpretation Rtriv2 < Rtriv y
  apply (rule Rtriv.intro)
  apply (rule y)
  done

print-locale Rtriv2

locale Rtriv3 = var x + var y + var z +
  assumes x: x = x and y: y = y and z: z = z

interpretation Rtriv3 < Rtriv2 x y
  apply (rule Rtriv2.intro)
  apply (rule x)
  apply (rule y)
  done
```

print-locale *Rtriv3*

interpretation *Rtriv3* < *Rtriv2 x z*
apply (*rule Rtriv2.intro*)
apply (*rule x-y-z.x*)
apply (*rule z*)
done

ML ⟨⟨ *set show-types* ⟩⟩

print-locale *Rtriv3*

14.4 Normalisation Replaces Assumed Element by Derived Element

typedecl ('*a*, '*b*) *pair*
arities *pair* :: (*term*, *term*) *term*

consts
pair :: ['*a*, '*b*] => ('*a*, '*b*) *pair*
fst :: ('*a*, '*b*) *pair* => '*a*
snd :: ('*a*, '*b*) *pair* => '*b*

axioms
fst [*simp*]: *fst*(*pair*(*x*, *y*)) = *x*
snd [*simp*]: *snd*(*pair*(*x*, *y*)) = *y*

locale *Rpair* = *var prod* (**infixl** ** 65*) + *var prodP* (**infixl** *** 65*) +
defines *P-def*: *x* *** *y* == *pair*(*fst*(*x*) ** *fst*(*y*), *snd*(*x*) ** *snd*(*y*))

locale *Rpair-semi* = *Rpair* + *Rpsemi*

interpretation *Rpair-semi* < *Rpsemi prodP* (**infixl** *** 65*)
proof (*rule Rpsemi.intro*)
fix *x y z*
show (*x* *** *y*) *** *z* = *x* *** (*y* *** *z*)
by (*unfold P-def*) (*simp add*: *assoc*)
qed

locale *Rsemi-rev* = *Rpsemi* + *var rprod* (**infixl** ++ 65*) +
defines *r-def*: *x* ++ *y* == *y* ** *x*

lemma (**in** *Rsemi-rev*) *r-assoc*:
(*x* ++ *y*) ++ *z* = *x* ++ (*y* ++ *z*)
by (*simp add*: *r-def assoc*)

lemma (**in** *Rpair-semi*)
includes *Rsemi-rev prodP* (**infixl** *** 65*) *rprodP* (**infixl** +++ 65*)
constrains *prod* :: ['*a*, '*a*] => '*a*

and *rprodP* :: [('*a*, '*a*) *pair*, ('*a*, '*a*) *pair*] => ('*a*, '*a*) *pair*
shows (*x* +++ *y*) +++ *z* = *x* +++ (*y* +++ *z*)
apply (*rule r-assoc*) **done**

end