

Completeness for FOL

James Margetson, ported by Tom Ridge

April 10, 2026

Contents

1	Permutation Lemmas	1
1.1	perm, count equivalence	1
1.2	Properties closed under Perm and Contr hold for x iff hold for remdups x	2
1.3	List properties closed under Perm, Weak and Contr are mono- tonic in the set of the list	2
2	Base	3
2.1	Integrate with Isabelle libraries?	3
2.2	Summation	3
2.3	Termination Measure	3
2.4	Functions	3
3	Formula	4
3.1	Variables	4
3.2	Predicates	6
3.3	Formulas	6
3.4	formula signs induct, formula signs cases	6
3.5	Frees	7
3.6	Substitutions	8
3.7	Models	9
3.8	model, non empty set and positive atom valuation	9
3.9	Validity	9
4	Sequents	10
4.1	Rules	11
4.2	Deductions	11
4.3	Basic Rule sets	11
4.4	Derived Rules	13
4.5	Standard Rule Sets For Predicate Calculus	14
4.6	Monotonicity for CutFreePC deductions	14
4.7	Tree	15

4.8	Terminal	15
4.9	Inherited	16
4.10	bounded, boundedBy	17
4.11	Inherited Properties- bounded	18
4.12	founded	19
4.13	Inherited Properties- founded	19
4.14	Inherited Properties- finite	20
4.15	path: follows a failing inherited property through tree	20
4.16	Branch	21
4.17	failing branch property: abstracts path defn	22
4.18	Tree induction principles	22
5	Completeness	22
5.1	pseq: type represents a processed sequent	23
5.2	subs: SATAxiom	23
5.3	subs: a CutFreePC justifiable backwards proof step	23
5.4	proofTree(Gamma) says whether tree(Gamma) is a proof	24
5.5	path: considers, contains, costBarrier	24
5.6	path: eventually	24
5.7	path: counter model	24
5.8	subs: finite	25
5.9	inherited: proofTree	25
5.10	pseq: lemma	26
5.11	SATAxiom: proofTree	26
5.12	SATAxioms are deductions: - needed	26
5.13	proofTrees are deductions: subs respects rules - messy start and end	26
5.14	proofTrees are deductions: instance of boundedTreeInduction	27
5.15	contains, considers:	27
5.16	path: nforms = [] implies	27
5.17	path: cases	27
5.18	path: contains not terminal and propagate condition	28
5.19	termination: (for EV contains implies EV considers)	28
5.20	costBarrier: lemmas	28
5.21	costBarrier: exp3 lemmas - bit specific...	28
5.22	costBarrier: decreases whilst contains and unconsiders	28
5.23	path: EV contains implies EV considers	29
5.24	EV contains: common lemma	29
5.25	EV contains: FConj,FDisj,FAll	29
5.26	EV contains: lemmas (temporal related)	30
5.27	EV contains: FAToms	30
5.28	EV contains: FEx cases	30
5.29	pseq: creates initial pseq	30

5.30 EV contains: contain any (i,FEx y) means contain all (i,FEx y)	31
5.31 EV contains: contain any (i,FEx y) means contain all (i,FEx y)	31
5.32 EV contains: atoms	31
5.33 counterModel: lemmas	32
5.34 counterModel: all path formula value false - step by step . . .	32
5.35 adequacy	32

6 Soundness 33

1 Permutation Lemmas

```

theory PermutationLemmas
imports HOL-Library.Multiset
begin

```

— following function is very close to that in multisets- now we can make the connection that $x < > y$ iff the multiset of x is the same as that of y

1.1 perm, count equivalence

```

lemma count-eq:
  <count-list xs x = Multiset.count (mset xs) x>
  <proof>

```

```

lemma perm-count: mset A = mset B  $\implies$  ( $\forall x$ . count-list A x = count-list B x)
  <proof>

```

```

lemma count-0: ( $\forall x$ . count-list B x = 0) = (B = [])
  <proof>

```

```

lemma count-Suc: count-list B a = Suc m  $\implies$  a  $\in$  set B
  <proof>

```

```

lemma count-perm: !! B. ( $\forall x$ . count-list A x = count-list B x)  $\implies$  mset A =
mset B
  <proof>

```

```

lemma perm-count-conv: mset A = mset B  $\longleftrightarrow$  ( $\forall x$ . count-list A x = count-list
B x)
  <proof>

```

1.2 Properties closed under Perm and Contr hold for x iff hold for remdups x

```

lemma remdups-append: y  $\in$  set ys  $\implies$  remdups (ws@y#ys) = remdups (ws@ys)
  <proof>

```

lemma perm-contr':
assumes *perm*: $\bigwedge xs\ ys. mset\ xs = mset\ ys \implies (P\ xs = P\ ys)$
and *contr'*: $\bigwedge x\ xs. P(x\#\#x\#xs) = P(x\#xs)$
shows $length\ xs = n \implies (P\ xs = P\ (remdups\ xs))$
<proof>

lemma perm-contr:
assumes *perm*: $\bigwedge xs\ ys. mset\ xs = mset\ ys \implies (P\ xs = P\ ys)$
and *contr'*: $\bigwedge x\ xs. P(x\#\#x\#xs) = P(x\#xs)$
shows $(P\ xs = P\ (remdups\ xs))$
<proof>

1.3 List properties closed under Perm, Weak and Contr are monotonic in the set of the list

definition
rem :: 'a => 'a list => 'a list **where**
rem *x* *xs* = *filter* ($\%y. y \sim = x$) *xs*

lemma rem: $x \notin set\ (rem\ x\ xs)$
<proof>

lemma length-rem: $length\ (rem\ x\ xs) \leq length\ xs$
<proof>

lemma rem-notin: $x \notin set\ xs \implies rem\ x\ xs = xs$
<proof>

lemma perm-weak-filter':
assumes *perm*: $\bigwedge xs\ ys. mset\ xs = mset\ ys \implies (P\ xs = P\ ys)$
and *weak*: $\bigwedge x\ xs. P\ xs \implies P(x\#xs)$
and *P*: $P\ (ys@filter\ Q\ xs)$
shows $P\ (ys@xs)$
<proof>

lemma perm-weak-filter:
assumes *perm*: $\bigwedge xs\ ys. mset\ xs = mset\ ys \implies (P\ xs = P\ ys)$
and *weak*: $\bigwedge x\ xs. P\ xs \implies P(x\#xs)$
shows $P\ (filter\ Q\ xs) \implies P\ xs$
<proof>

lemma perm-weak-contr-mono:
assumes *perm*: $\bigwedge xs\ ys. mset\ xs = mset\ ys \implies (P\ xs = P\ ys)$
and *contr*: $\bigwedge x\ xs. P(x\#\#x\#xs) \implies P(x\#xs)$
and *weak*: $\bigwedge x\ xs. P\ xs \implies P(x\#xs)$
and *xy*: $set\ x \subseteq set\ y$
and *Px*: $P\ x$
shows $P\ y$

<proof>

end

2 Base

theory *Base*
imports *PermutationLemmas*
begin

2.1 Integrate with Isabelle libraries?

lemma *natset-finite-max*:
 assumes *a*: *finite A*
 shows *Suc (Max A) ∉ A*
 <proof>

2.2 Summation

primrec *summation* :: $(\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{nat} \Rightarrow \text{nat}$
 where
 summation f 0 = f 0
 | *summation f (Suc n) = f (Suc n) + summation f n*

2.3 Termination Measure

primrec *sumList* :: $\text{nat list} \Rightarrow \text{nat}$
 where
 sumList [] = 0
 | *sumList (x#xs) = x + sumList xs*

2.4 Functions

abbreviation (*input*) *preImage* \equiv *vimage*

abbreviation (*input*) *pre f a* \equiv $f^{-1} \{a\}$

definition
equalOn :: $['a \text{ set}, 'a \Rightarrow 'b, 'a \Rightarrow 'b] \Rightarrow \text{bool}$ **where**
equalOn A f g = $(\forall x \in A. f x = g x)$

lemma *preImage-insert*: $\text{preImage } f (\text{insert } a \ A) = \text{pre } f \ a \cup \text{preImage } f \ A$
<proof>

lemma *equalOn-Un*: $\text{equalOn } (A \cup B) \ f \ g = (\text{equalOn } A \ f \ g \wedge \text{equalOn } B \ f \ g)$
<proof>

lemma *equalOnD*: $\text{equalOn } A \ f \ g \Longrightarrow (\forall x \in A. f x = g x)$
<proof>

lemma *equalOnI*: $(\forall x \in A . f x = g x) \implies \text{equalOn } A f g$
 ⟨*proof*⟩

lemma *equalOn-UnD*: $\text{equalOn } (A \text{ Un } B) f g \implies \text{equalOn } A f g \ \& \ \text{equalOn } B f g$
 ⟨*proof*⟩

lemma *inj-inv-singleton[simp]*: $\llbracket \text{inj } f; f z = y \rrbracket \implies \{x. f x = y\} = \{z\}$
 ⟨*proof*⟩

lemma *finite-pre[simp]*: $\text{inj } f \implies \text{finite } (\text{pre } f x)$
 ⟨*proof*⟩

declare *finite-vimageI [simp]*

end

3 Formula

theory *Formula*
imports *Base*
begin

3.1 Variables

datatype *vbl = X nat*

— FIXME there's a lot of stuff about this datatype that is really just a lifting from nat (what else could it be). Makes me wonder whether things wouldn't be clearer if we just identified vbls with nats

primrec *deX* :: *vbl* \Rightarrow *nat* **where** *deX* (X n) = n

lemma *X-deX[simp]*: *X* (deX a) = a
 ⟨*proof*⟩

definition *zeroX* = X 0

primrec

nextX :: *vbl* \Rightarrow *vbl* **where**
nextX (X n) = X (Suc n)

definition

vblcase :: [*'a, vbl* \Rightarrow *'a, vbl*] \Rightarrow *'a* **where**
vblcase a f n \equiv (@z. (n=zeroX \longrightarrow z=a) \wedge (!x. n=nextX x \longrightarrow z=f(x)))

declare $\llbracket \text{case-translation } \text{vblcase } \text{zeroX } \text{nextX} \rrbracket$

definition

freshVar :: *vbl set* \Rightarrow *vbl* **where**
freshVar *vs* = *X* (*LEAST* *n*. *n* \notin *deX* ' *vs*)

lemma *nextX-nextX*[*iff*]: *nextX* *x* = *nextX* *y* = (*x* = *y*)
 ⟨*proof*⟩

lemma *inj-nextX*: *inj nextX*
 ⟨*proof*⟩

lemma *ind*:
 assumes *P zeroX* \wedge *v*. *P v* \implies *P (nextX v)*
 shows *P v'*
 ⟨*proof*⟩

lemma *zeroX-nextX*[*iff*]: *zeroX* \neq *nextX a*
 ⟨*proof*⟩

lemmas *nextX-zeroX*[*iff*] = *not-sym*[*OF zeroX-nextX*]

lemma *nextX*: *nextX* (*X n*) = *X (Suc n)*
 ⟨*proof*⟩

lemma *vblcase-zeroX*[*simp*]: *vblcase a b zeroX* = *a*
 ⟨*proof*⟩

lemma *vblcase-nextX*[*simp*]: *vblcase a b (nextX n)* = *b n*
 ⟨*proof*⟩

lemma *vbl-cases*: *x* = *zeroX* \vee (\exists *y*. *x* = *nextX y*)
 ⟨*proof*⟩

lemma *vbl-casesE*: $\llbracket x = \text{zeroX} \implies P; \wedge y. x = \text{nextX } y \implies P \rrbracket \implies P$
 ⟨*proof*⟩

lemma *comp-vblcase*: *f* \circ *vblcase a b* = *vblcase (f a) (f \circ b)*
 ⟨*proof*⟩

lemma *equalOn-vblcaseI'*: *equalOn (preImage nextX A) f g* \implies *equalOn A (vblcase x f) (vblcase x g)*
 ⟨*proof*⟩

lemma *equalOn-vblcaseI*: (*zeroX* \in *A* \implies *x=y*) \implies *equalOn (preImage nextX A) f g* \implies *equalOn A (vblcase x f) (vblcase y g)*
 ⟨*proof*⟩

lemma *X-deX-connection*: *X n* \in *A* = (*n* \in (*deX* ' *A*))
 ⟨*proof*⟩

lemma *finiteFreshVar*: *finite A* \implies *freshVar A* \notin *A*

<proof>

lemma *freshVarI*: $\llbracket \text{finite } A; B \subseteq A \rrbracket \implies \text{freshVar } A \notin B$
<proof>

lemma *freshVarI2*: $\llbracket \text{finite } A; \bigwedge x . x \notin A \implies P x \rrbracket \implies P (\text{freshVar } A)$
<proof>

lemmas *vblsimps* = *vblcase-zeroX vblcase-nextX zeroX-nextX*
nextX-zeroX nextX-nextX comp-vblcase

3.2 Predicates

datatype *predicate* = *Predicate nat*

datatype *signs* = *Pos* | *Neg*

primrec *sign* :: *signs* \Rightarrow *bool* \Rightarrow *bool*
where
 sign Pos *x* = *x*
 | *sign Neg* *x* = (\neg *x*)

primrec *invSign* :: *signs* \Rightarrow *signs*
where
 invSign Pos = *Neg*
 | *invSign Neg* = *Pos*

3.3 Formulas

datatype *formula* =
 FAtom signs predicate (vbl list)
 | *FConj signs formula formula*
 | *FAll signs formula*

3.4 formula signs induct, formula signs cases

lemma *formula-signs-induct* [*case-names FAtomP FAtomN FConjP FConjN FAllP*
FAllN, cases type: formula]:
 $\llbracket \bigwedge p \text{ vs. } P (\text{FAtom Pos } p \text{ vs});$
 $\bigwedge p \text{ vs. } P (\text{FAtom Neg } p \text{ vs});$
 $\bigwedge A B . \llbracket P A; P B \rrbracket \implies P (\text{FConj Pos } A B);$
 $\bigwedge A B . \llbracket P A; P B \rrbracket \implies P (\text{FConj Neg } A B);$
 $\bigwedge A . \llbracket P A \rrbracket \implies P (\text{FAll Pos } A);$
 $\bigwedge A . \llbracket P A \rrbracket \implies P (\text{FAll Neg } A)$
 $\rrbracket \implies P A$
<proof>

lemma *sizelemmas*: *size A* < *size (FConj z A B)*
size B < *size (FConj z A B)*
size A < *size (FAll z A)*

<proof>

primrec *FNot* :: *formula* \Rightarrow *formula*

where

FNot-FAtom: $FNot (FAtom\ z\ P\ vs) = FAtom\ (invSign\ z)\ P\ vs$

| *FNot-FConj*: $FNot (FConj\ z\ A0\ A1) = FConj\ (invSign\ z)\ (FNot\ A0)\ (FNot\ A1)$

| *FNot-FAll*: $FNot (FAll\ z\ body) = FAll\ (invSign\ z)\ (FNot\ body)$

primrec *neg* :: *signs* \Rightarrow *signs*

where

neg Pos = *Neg*

| *neg Neg* = *Pos*

primrec

dual :: [(*signs* \Rightarrow *signs*), (*signs* \Rightarrow *signs*), (*signs* \Rightarrow *signs*)] \Rightarrow *formula* \Rightarrow *formula*

where

dual-FAtom: $dual\ p\ q\ r\ (FAtom\ z\ P\ vs) = FAtom\ (p\ z)\ P\ vs$

| *dual-FConj*: $dual\ p\ q\ r\ (FConj\ z\ A0\ A1) = FConj\ (q\ z)\ (dual\ p\ q\ r\ A0)\ (dual\ p\ q\ r\ A1)$

| *dual-FAll*: $dual\ p\ q\ r\ (FAll\ z\ body) = FAll\ (r\ z)\ (dual\ p\ q\ r\ body)$

lemma *dualCompose*: $dual\ p\ q\ r\ \circ\ dual\ P\ Q\ R = dual\ (p\ \circ\ P)\ (q\ \circ\ Q)\ (r\ \circ\ R)$

<proof>

lemma *dualFNot'*: $dual\ invSign\ invSign\ invSign = FNot$

<proof>

lemma *dualFNot*: $dual\ invSign\ id\ id\ (FNot\ A) = FNot\ (dual\ invSign\ id\ id\ A)$

<proof>

lemma *dualId*: $dual\ id\ id\ id\ A = A$

<proof>

3.5 Frees

primrec *freeVarsF* :: *formula* \Rightarrow *vbl set*

where

freeVarsFAtom: $freeVarsF (FAtom\ z\ P\ vs) = set\ vs$

| *freeVarsFConj*: $freeVarsF (FConj\ z\ A0\ A1) = (freeVarsF\ A0) \cup (freeVarsF\ A1)$

| *freeVarsFAll*: $freeVarsF (FAll\ z\ body) = preImage\ nextX\ (freeVarsF\ body)$

definition

freeVarsFL :: *formula list* \Rightarrow *vbl set* **where**

freeVarsFL $\Gamma = Union\ (freeVarsF\ ' (set\ \Gamma))$

lemma *freeVarsF-FNot[simp]*: $freeVarsF (FNot\ A) = freeVarsF\ A$

<proof>

lemma *finite-freeVarsF[simp]*: *finite (freeVarsF A)*
 ⟨*proof*⟩

lemma *freeVarsFL-nil[simp]*: *freeVarsFL ([]) = {}*
 ⟨*proof*⟩

lemma *freeVarsFL-cons*: *freeVarsFL (A#Γ) = freeVarsF A ∪ freeVarsFL Γ*
 ⟨*proof*⟩

lemma *finite-freeVarsFL[simp]*: *finite (freeVarsFL Γ)*
 ⟨*proof*⟩

lemma *freeVarsDual*: *freeVarsF (dual p q r A) = freeVarsF A*
 ⟨*proof*⟩

3.6 Substitutions

primrec *subF* :: [*vbl* ⇒ *vbl,formula*] ⇒ *formula*

where

subFAtom: *subF theta (FAtom z P vs) = FAtom z P (map theta vs)*

| *subFConj*: *subF theta (FConj z A0 A1) = FConj z (subF theta A0) (subF theta A1)*

| *subFAll*: *subF theta (FAll z body) =*

FAll z (subF (λv . (case v of zeroX ⇒ zeroX | nextX v ⇒ nextX (theta v)))
body)

lemma *size-subF*: *size (subF theta A) = size (A::formula)*
 ⟨*proof*⟩

lemma *subFNot*: *subF theta (FNot A) = FNot (subF theta A)*
 ⟨*proof*⟩

lemma *subFDual*: *subF theta (dual p q r A) = dual p q r (subF theta A)*
 ⟨*proof*⟩

definition

instanceF :: [*vbl,formula*] ⇒ *formula* **where**

instanceF w body = subF (λv. case v of zeroX ⇒ w | nextX v ⇒ v) body

lemma *size-instance*: *size (instanceF v A) = size A*
 ⟨*proof*⟩

lemma *instanceFDual*: *instanceF u (dual p q r A) = dual p q r (instanceF u A)*
 ⟨*proof*⟩

3.7 Models

typedecl

object

axiomatization $obj :: nat \Rightarrow object$
where $inj-obj: inj\ obj$

3.8 model, non empty set and positive atom valuation

definition $model = \{z :: (object\ set * ([predicate,object\ list] \Rightarrow bool)). (fst\ z \neq \{\})\}$

typedef $model = model$
 $\langle proof \rangle$

definition
 $objects :: model \Rightarrow object\ set$ **where**
 $objects\ M = fst\ (Rep-model\ M)$

definition
 $evalP :: model \Rightarrow predicate \Rightarrow object\ list \Rightarrow bool$ **where**
 $evalP\ M = snd\ (Rep-model\ M)$

lemma $objectsNonEmpty: objects\ M \neq \{\}$
 $\langle proof \rangle$

lemma $modelsNonEmptyI: fst\ (Rep-model\ M) \neq \{\}$
 $\langle proof \rangle$

3.9 Validity

primrec $evalF :: [model,vbl \Rightarrow object,formula] \Rightarrow bool$
where
 $evalFAtom: evalF\ M\ \varphi\ (FAtom\ z\ P\ vs) = sign\ z\ (evalP\ M\ P\ (map\ \varphi\ vs))$
 $| evalFConj: evalF\ M\ \varphi\ (FConj\ z\ A0\ A1) = sign\ z\ (sign\ z\ (evalF\ M\ \varphi\ A0) \wedge sign\ z\ (evalF\ M\ \varphi\ A1))$
 $| evalFAll: evalF\ M\ \varphi\ (FAll\ z\ body) =$
 $sign\ z\ (\forall x \in objects\ M. sign\ z\ (evalF\ M\ (\lambda v. (case\ v\ of\ zeroX \Rightarrow x\ | nextX\ v \Rightarrow \varphi\ v))\ body))$

definition
 $valid :: formula \Rightarrow bool$ **where**
 $valid\ F \equiv (\forall M\ \varphi. evalF\ M\ \varphi\ F = True)$

lemma $evalF-FAll: evalF\ M\ \varphi\ (FAll\ Pos\ A) = (\forall x \in objects\ M. (evalF\ M\ (vblcase\ x\ \varphi)\ A))$
 $\langle proof \rangle$

lemma $evalF-FEx: evalF\ M\ \varphi\ (FAll\ Neg\ A) = (\exists x \in objects\ M. (evalF\ M\ (vblcase\ x\ \varphi)\ A))$
 $\langle proof \rangle$

lemma *evalF-arg2-cong*: $x = y \implies \text{evalF } M \text{ p } x = \text{evalF } M \text{ p } y$
<proof>

lemma *evalF-FNot*: $\forall \varphi. \text{evalF } M \ \varphi \ (\text{FNot } A) = (\neg \text{evalF } M \ \varphi \ A)$
<proof>

lemma *evalF-equiv*: $\text{equalOn } (\text{freeVarsF } A) \ f \ g \implies \text{evalF } M \ f \ A = \text{evalF } M \ g \ A$
<proof>

lemma *evalF-subF-eq*: $\text{evalF } M \ \varphi \ (\text{subF } \theta \ A) = \text{evalF } M \ (\varphi \circ \theta) \ A$
<proof>

lemma *evalF-instance*: $\text{evalF } M \ \varphi \ (\text{instanceF } u \ A) = \text{evalF } M \ (\text{vblcase } (\varphi \ u) \ \varphi) \ A$
<proof>

lemma *instanceF-E*: $\text{instanceF } g \ \text{formula} \neq \text{FAll signs formula}$
<proof>

end

4 Sequents

theory *Sequents*
imports *Formula*
begin

type-synonym *sequent* = *formula list*

definition
evalS :: $[\text{model}, \text{vbl} \Rightarrow \text{object}, \text{formula list}] \Rightarrow \text{bool}$ **where**
 $\text{evalS } M \ \varphi \ fs \equiv (\exists f \in \text{set } fs . \text{evalF } M \ \varphi \ f = \text{True})$

lemma *evalS-nil[simp]*: $\text{evalS } M \ \varphi \ [] = \text{False}$
<proof>

lemma *evalS-cons[simp]*: $\text{evalS } M \ \varphi \ (A \ \# \ \Gamma) = (\text{evalF } M \ \varphi \ A \vee \text{evalS } M \ \varphi \ \Gamma)$
<proof>

lemma *evalS-append*: $\text{evalS } M \ \varphi \ (\Gamma \ @ \ \Delta) = (\text{evalS } M \ \varphi \ \Gamma \vee \text{evalS } M \ \varphi \ \Delta)$
<proof>

lemma *evalS-equiv*: $(\text{equalOn } (\text{freeVarsFL } \Gamma) \ f \ g) \implies (\text{evalS } M \ f \ \Gamma = \text{evalS } M \ g \ \Gamma)$
<proof>

definition
modelAssigns :: $[\text{model}] \Rightarrow (\text{vbl} \Rightarrow \text{object}) \ \text{set}$ **where**
 $\text{modelAssigns } M = \{ \varphi . \text{range } \varphi \subseteq \text{objects } M \}$

lemma *modelAssigns-iff* [*simp*]: $f \in \text{modelAssigns } M \longleftrightarrow \text{range } f \subseteq \text{objects } M$
 ⟨*proof*⟩

definition

validS :: *formula list* \Rightarrow *bool* **where**
validS *fs* $\equiv (\forall M. \forall \varphi \in \text{modelAssigns } M . \text{evalS } M \varphi \text{ fs} = \text{True})$

4.1 Rules

type-synonym *rule* = *sequent* * (*sequent set*)

definition

concR :: *rule* \Rightarrow *sequent* **where**
concR = $(\lambda(\text{conc}, \text{prems}). \text{conc})$

definition

premsR :: *rule* \Rightarrow *sequent set* **where**
premsR = $(\lambda(\text{conc}, \text{prems}). \text{prems})$

definition

mapRule :: (*formula* \Rightarrow *formula*) \Rightarrow *rule* \Rightarrow *rule* **where**
mapRule = $(\lambda f (\text{conc}, \text{prems}). (\text{map } f \text{ conc}, (\text{map } f) \text{ 'prems}))$

lemma *mapRuleI*: $\llbracket A = \text{map } f \text{ a}; B = \text{map } f \text{ ' b} \rrbracket \Longrightarrow (A, B) = \text{mapRule } f \text{ (a, b)}$
 ⟨*proof*⟩

4.2 Deductions

lemmas *Powp-mono* [*mono*] = *Pow-mono* [*to-pred pred-subset-eq*]

inductive-set

deductions :: *rule set* \Rightarrow *formula list set*
for *rules* :: *rule set*

where

inferI: $\llbracket (\text{conc}, \text{prems}) \in \text{rules}; \text{prems} \in \text{Pow}(\text{deductions}(\text{rules})) \rrbracket \Longrightarrow \text{conc} \in \text{deductions}(\text{rules})$

lemma *mono-deductions*: $\llbracket A \subseteq B \rrbracket \Longrightarrow \text{deductions}(A) \subseteq \text{deductions}(B)$
 ⟨*proof*⟩

4.3 Basic Rule sets

definition

Axioms = $\{z. \exists p \text{ vs. } z = ([\text{FAtom Pos } p \text{ vs}, \text{FAtom Neg } p \text{ vs}], \{\}) \}$

definition

Conjs = $\{z. \exists A0 \text{ A1 } \Delta \Gamma. z = (\text{FConj Pos } A0 \text{ A1 } \# \Gamma @ \Delta, \{A0 \# \Gamma, A1 \# \Delta\}) \}$

definition

$$Disjs = \{z. \exists A0 A1 \quad \Gamma. z = (FConj Neg A0 A1 \# \Gamma, \{A0 \# A1 \# \Gamma\}) \}$$

definition

$$Alls = \{z. \exists A x \quad \Gamma. z = (FAll Pos A \# \Gamma, \{instanceF x A \# \Gamma\}) \wedge x \notin freeVarsFL (FAll Pos A \# \Gamma) \}$$

definition

$$Exs = \{z. \exists A x \quad \Gamma. z = (FAll Neg A \# \Gamma, \{instanceF x A \# \Gamma\}) \}$$

definition

$$Weaks = \{z. \exists A \quad \Gamma. z = (A \# \Gamma, \{\Gamma\}) \}$$

definition

$$Contrs = \{z. \exists A \quad \Gamma. z = (A \# \Gamma, \{A \# A \# \Gamma\}) \}$$

definition

$$Cuts = \{z. \exists C \Delta \quad \Gamma. z = (\Gamma @ \Delta, \{C \# \Gamma, FNot C \# \Delta\}) \}$$

definition

$$Perms = \{z. \exists \Gamma \Delta \quad . z = (\Gamma, \{\Delta\}) \wedge mset \Gamma = mset \Delta \}$$

definition

$$DAxioms = \{z. \exists p vs. \quad z = ([FAtom Neg p vs, FAtom Pos p vs], \{\}) \}$$

lemma *AxiomI*: $\llbracket Axioms \subseteq A \rrbracket \implies [FAtom Pos p vs, FAtom Neg p vs] \in deductions(A)$
<proof>

lemma *DAxiomsI*: $\llbracket DAxioms \subseteq A \rrbracket \implies [FAtom Neg p vs, FAtom Pos p vs] \in deductions(A)$
<proof>

lemma *DisjI*: $\llbracket A0 \# A1 \# \Gamma \in deductions(A); Disjs \subseteq A \rrbracket \implies (FConj Neg A0 A1 \# \Gamma) \in deductions(A)$
<proof>

lemma *ConjI*: $\llbracket (A0 \# \Gamma) \in deductions(A); (A1 \# \Delta) \in deductions(A); Conjs \subseteq A \rrbracket \implies FConj Pos A0 A1 \# \Gamma @ \Delta \in deductions(A)$
<proof>

lemma *AllI*: $\llbracket instanceF w A \# \Gamma \in deductions(R); w \notin freeVarsFL (FAll Pos A \# \Gamma); Alls \subseteq R \rrbracket \implies (FAll Pos A \# \Gamma) \in deductions(R)$
<proof>

lemma *ExI*: $\llbracket instanceF w A \# \Gamma \in deductions(R); Exs \subseteq R \rrbracket \implies (FAll Neg A \# \Gamma) \in deductions(R)$
<proof>

lemma *WeakI*: $[[\Gamma \in \text{deductions } R; \text{Weaks} \subseteq R]] \implies A\#\Gamma \in \text{deductions}(R)$
 ⟨proof⟩

lemma *ContrI*: $[[A\#A\#\Gamma \in \text{deductions } R; \text{Contrs} \subseteq R]] \implies A\#\Gamma \in \text{deductions}(R)$
 ⟨proof⟩

lemma *PermI*: $[[\text{Gamma}' \in \text{deductions } R; \text{mset } \Gamma = \text{mset } \text{Gamma}'; \text{Perms} \subseteq R]] \implies \Gamma \in \text{deductions}(R)$
 ⟨proof⟩

4.4 Derived Rules

lemma *WeakI1*: $[[\Gamma \in \text{deductions}(A); \text{Weaks} \subseteq A]] \implies (\Delta @ \Gamma) \in \text{deductions}(A)$
 ⟨proof⟩

lemma *WeakI2*: $[[\Gamma \in \text{deductions}(A); \text{Perms} \subseteq A; \text{Weaks} \subseteq A]] \implies (\Gamma @ \Delta) \in \text{deductions}(A)$
 ⟨proof⟩

lemma *SATAxiomI*: $[[\text{Axioms} \subseteq A; \text{Weaks} \subseteq A; \text{Perms} \subseteq A; \text{forms} = [\text{FAtom Pos } n \text{ vs, FAtom Neg } n \text{ vs}] @ \Gamma]] \implies \text{forms} \in \text{deductions}(A)$
 ⟨proof⟩

lemma *DisjI1*: $[[A1\#\Gamma \in \text{deductions}(A); \text{Disjs} \subseteq A; \text{Weaks} \subseteq A]] \implies \text{FConj Neg } A0 \ A1\#\Gamma \in \text{deductions}(A)$
 ⟨proof⟩

lemma *DisjI2*: $[[A0\#\Gamma \in \text{deductions}(A); \text{Disjs} \subseteq A; \text{Weaks} \subseteq A; \text{Perms} \subseteq A]] \implies \text{FConj Neg } A0 \ A1\#\Gamma \in \text{deductions}(A)$
 ⟨proof⟩

lemma *perm-tmp4*: $\text{Perms} \subseteq R \implies A @ (a \# \text{list}) @ (a \# \text{list}) \in \text{deductions } R$
 $\implies (a \# a \# A) @ \text{list} @ \text{list} \in \text{deductions } R$
 ⟨proof⟩

lemma *weaken-append*:

$\text{Contrs} \subseteq R \implies \text{Perms} \subseteq R \implies A @ \Gamma @ \Gamma \in \text{deductions}(R) \implies A @ \Gamma \in \text{deductions}(R)$
 ⟨proof⟩

lemma *ListWeakI*: $\text{Perms} \subseteq R \implies \text{Contrs} \subseteq R \implies x \# \Gamma @ \Gamma \in \text{deductions}(R)$
 $\implies x \# \Gamma \in \text{deductions}(R)$
 ⟨proof⟩

lemma *ConjI'*: $[[A0\#\Gamma \in \text{deductions}(A); (A1\#\Gamma) \in \text{deductions}(A); \text{Contrs} \subseteq A; \text{Conjs} \subseteq A; \text{Perms} \subseteq A]] \implies \text{FConj Pos } A0 \ A1\#\Gamma \in \text{deductions}(A)$
 ⟨proof⟩

4.5 Standard Rule Sets For Predicate Calculus

definition

PC :: rule set **where**

$PC = \text{Union } \{Perms, Axioms, Conjs, Disjs, Alls, Exs, Weaks, Contrs, Cuts\}$

definition

$CutFreePC$:: rule set **where**

$CutFreePC = \text{Union } \{Perms, Axioms, Conjs, Disjs, Alls, Exs, Weaks, Contrs\}$

lemma *rulesInPCs*: $Axioms \subseteq PC$ $Axioms \subseteq CutFreePC$

$Conjs \subseteq PC$ $Conjs \subseteq CutFreePC$

$Disjs \subseteq PC$ $Disjs \subseteq CutFreePC$

$Alls \subseteq PC$ $Alls \subseteq CutFreePC$

$Exs \subseteq PC$ $Exs \subseteq CutFreePC$

$Weaks \subseteq PC$ $Weaks \subseteq CutFreePC$

$Contrs \subseteq PC$ $Contrs \subseteq CutFreePC$

$Perms \subseteq PC$ $Perms \subseteq CutFreePC$

$Cuts \subseteq PC$

$CutFreePC \subseteq PC$

<proof>

4.6 Monotonicity for CutFreePC deductions

definition

$inDed$:: formula list \Rightarrow bool **where**

$inDed\ xs \equiv xs \in \text{deductions } CutFreePC$

lemma *perm*: $mset\ xs = mset\ ys \implies (inDed\ xs = inDed\ ys)$

<proof>

lemma *contr*: $inDed\ (x\#\ x\#\ xs) \implies inDed\ (x\#\ xs)$

<proof>

lemma *weak*: $inDed\ xs \implies inDed\ (x\#\ xs)$

<proof>

lemma *inDed-mono'[simplified inDed-def]*: $set\ x \subseteq set\ y \implies inDed\ x \implies inDed\ y$

<proof>

lemma *inDed-mono[simplified inDed-def]*: $inDed\ x \implies set\ x \subseteq set\ y \implies inDed\ y$

<proof>

end

theory *Tree*

imports *Main*

begin

4.7 Tree

inductive-set

$tree :: ['a \Rightarrow 'a\ set, 'a] \Rightarrow (nat * 'a)\ set$

for $subs :: 'a \Rightarrow 'a\ set$ **and** $\gamma :: 'a$

— This set represents the nodes in a tree which may represent a proof of γ .

Only storing the annotation and its level.

where

$tree0: (0, \gamma) \in tree\ subs\ \gamma$

| $tree1: [(n, delta) \in tree\ subs\ \gamma; sigma \in subs\ delta]$
 $\implies (Suc\ n, sigma) \in tree\ subs\ \gamma$

declare $tree.cases$ [elim]

declare $tree.intros$ [intro]

lemma $tree0Eq: (0, y) \in tree\ subs\ \gamma = (y = \gamma)$

$\langle proof \rangle$

lemma $tree1Eq:$

$(Suc\ n, Y) \in tree\ subs\ \gamma \longleftrightarrow (\exists\ sigma \in subs\ \gamma. (n, Y) \in tree\ subs\ sigma)$

$\langle proof \rangle$

definition

$incLevel :: nat * 'a \Rightarrow nat * 'a$ **where**

$incLevel = (\% (n, a). (Suc\ n, a))$

lemma $injIncLevel: inj\ incLevel$

$\langle proof \rangle$

lemma $treeEquation: tree\ subs\ \gamma = insert\ (0, \gamma)\ (\bigcup\ delta \in subs\ \gamma. incLevel\ ` tree\ subs\ delta)$

$\langle proof \rangle$

definition

$fans :: ['a \Rightarrow 'a\ set] \Rightarrow bool$ **where**

$fans\ subs \equiv (\forall x. finite\ (subs\ x))$

4.8 Terminal

definition

$terminal :: ['a \Rightarrow 'a\ set, 'a] \Rightarrow bool$ **where**

$terminal\ subs\ delta \equiv subs\ (delta) = \{\}$

lemma $terminalD: terminal\ subs\ \Gamma \implies x \sim: subs\ \Gamma$

$\langle proof \rangle$

lemma $terminalI: x \in subs\ \Gamma \implies \neg terminal\ subs\ \Gamma$

$\langle proof \rangle$

4.9 Inherited

definition

$inherited :: ['a \Rightarrow 'a \text{ set}, (nat * 'a) \text{ set} \Rightarrow bool] \Rightarrow bool$ **where**
 $inherited \text{ subs } P \equiv (\forall A B. (P A \wedge P B) = P (A \text{ Un } B))$
 $\wedge (\forall A. P A = P (\text{incLevel } ' A))$
 $\wedge (\forall n \Gamma A. \neg(\text{terminal subs } \Gamma) \longrightarrow P A = P (\text{insert } (n, \Gamma) A))$
 $\wedge (P \{\})$

— FIXME tjr why does it have to be invariant under inserting nonterminal nodes?

lemma $inheritedUn[\text{rule-format}]$: $inherited \text{ subs } P \longrightarrow P A \longrightarrow P B \longrightarrow P (A \text{ Un } B)$

$\langle \text{proof} \rangle$

lemma $inheritedIncLevel[\text{rule-format}]$: $inherited \text{ subs } P \longrightarrow P A \longrightarrow P (\text{incLevel } ' A)$

$\langle \text{proof} \rangle$

lemma $inheritedEmpty[\text{rule-format}]$: $inherited \text{ subs } P \longrightarrow P \{\}$

$\langle \text{proof} \rangle$

lemma $inheritedInsert[\text{rule-format}]$:

$inherited \text{ subs } P \longrightarrow \sim(\text{terminal subs } \Gamma) \longrightarrow P A \longrightarrow P (\text{insert } (n, \Gamma) A)$

$\langle \text{proof} \rangle$

lemma $inheritedI[\text{rule-format}]$: $\llbracket \forall A B. (P A \wedge P B) = P (A \text{ Un } B);$

$\forall A. P A = P (\text{incLevel } ' A);$

$\forall n \Gamma A. \sim(\text{terminal subs } \Gamma) \longrightarrow P A = P (\text{insert } (n, \Gamma) A);$

$P \{\} \rrbracket \Longrightarrow inherited \text{ subs } P$

$\langle \text{proof} \rangle$

lemma $inheritedUnEq[\text{rule-format}, \text{symmetric}]$: $inherited \text{ subs } P \longrightarrow (P A \wedge P B) = P (A \text{ Un } B)$

$\langle \text{proof} \rangle$

lemma $inheritedIncLevelEq[\text{rule-format}, \text{symmetric}]$: $inherited \text{ subs } P \longrightarrow P A = P (\text{incLevel } ' A)$

$\langle \text{proof} \rangle$

lemma $inheritedInsertEq[\text{rule-format}, \text{symmetric}]$: $inherited \text{ subs } P \longrightarrow \sim(\text{terminal subs } \Gamma) \longrightarrow P A = P (\text{insert } (n, \Gamma) A)$

$\langle \text{proof} \rangle$

lemmas $inheritedUnD = iffD1[OF inheritedUnEq]$

lemmas $inheritedInsertD = inheritedInsertEq[THEN iffD1]$

lemmas *inheritedIncLevelD* = *inheritedIncLevelEq*[*THEN iffD1*]

lemma *inheritedUNEq*:

finite A \implies *inherited subs P* \implies $(\forall x \in A. P (B x)) = P (\bigcup a \in A. B a)$
<proof>

lemmas *inheritedUN* = *inheritedUNEq*[*THEN iffD1*]

lemmas *inheritedUND*[*rule-format*] = *inheritedUNEq*[*THEN iffD2*]

lemma *inheritedPropagateEq*:

assumes *inherited subs P*

and fans subs

and \neg (*terminal subs delta*)

shows $P(\text{tree subs delta}) = (\forall \text{sigma} \in \text{subs delta}. P(\text{tree subs sigma}))$

<proof>

lemma *inheritedPropagate*:

$\llbracket \neg P (\text{tree subs delta}); \text{inherited subs } P; \text{fans subs}; \neg \text{terminal subs delta} \rrbracket$

$\implies \exists \text{sigma} \in \text{subs delta}. \neg P (\text{tree subs sigma})$

<proof>

lemma *inheritedViaSub*:

$\llbracket \text{inherited subs } P; \text{fans subs}; P (\text{tree subs delta}); \text{sigma} \in \text{subs delta} \rrbracket \implies P (\text{tree subs sigma})$

<proof>

lemma *inheritedJoin*:

$\llbracket \text{inherited subs } P; \text{inherited subs } Q \rrbracket \implies \text{inherited subs } (\lambda x. P x \wedge Q x)$

<proof>

lemma *inheritedJoinI*:

$\llbracket \text{inherited subs } P; \text{inherited subs } Q; R = (\lambda x. P x \wedge Q x) \rrbracket$

$\implies \text{inherited subs } R$

<proof>

4.10 bounded, boundedBy

definition

boundedBy :: *nat* \Rightarrow (*nat* * 'a) *set* \Rightarrow *bool* **where**

boundedBy N A \equiv $(\forall (n, \text{delta}) \in A. n < N)$

definition

bounded :: (*nat* * 'a) *set* \Rightarrow *bool* **where**

bounded A \equiv $(\exists N . \text{boundedBy } N A)$

lemma *boundedByEmpty*[*simp*]: *boundedBy N {}*

<proof>

lemma *boundedByInsert*: $\text{boundedBy } N (\text{insert } (n, \text{delta}) B) = (n < N \wedge \text{boundedBy } N B)$
 ⟨proof⟩

lemma *boundedByUn*: $\text{boundedBy } N (A \text{ Un } B) = (\text{boundedBy } N A \wedge \text{boundedBy } N B)$
 ⟨proof⟩

lemma *boundedByIncLevel'*: $\text{boundedBy } (\text{Suc } N) (\text{incLevel } ' A) = \text{boundedBy } N A$
 ⟨proof⟩

lemma *boundedByAdd1*: $\text{boundedBy } N B \implies \text{boundedBy } (N+M) B$
 ⟨proof⟩

lemma *boundedByAdd2*: $\text{boundedBy } M B \implies \text{boundedBy } (N+M) B$
 ⟨proof⟩

lemma *boundedByMono*: $\text{boundedBy } m B \implies m < M \implies \text{boundedBy } M B$
 ⟨proof⟩

lemmas *boundedByMonoD* = *boundedByMono*

lemma *boundedBy0*: $\text{boundedBy } 0 A = (A = \{\})$
 ⟨proof⟩

lemma *boundedBySuc'*: $\text{boundedBy } N A \implies \text{boundedBy } (\text{Suc } N) A$
 ⟨proof⟩

lemma *boundedByIncLevel*: $\text{boundedBy } n (\text{incLevel } ' (\text{tree subs } \gamma)) \longleftrightarrow (\exists m . n = \text{Suc } m \wedge \text{boundedBy } m (\text{tree subs } \gamma))$
 ⟨proof⟩

lemma *boundedByUN*: $\text{boundedBy } N (\text{UN } x:A. B x) = (!x:A. \text{boundedBy } N (B x))$
 ⟨proof⟩

lemma *boundedBySuc[rule-format]*: $\text{sigma} \in \text{subs } \Gamma \implies \text{boundedBy } (\text{Suc } n) (\text{tree subs } \Gamma) \implies \text{boundedBy } n (\text{tree subs } \text{sigma})$
 ⟨proof⟩

4.11 Inherited Properties- bounded

lemma *boundedEmpty*: $\text{bounded } \{\}$
 ⟨proof⟩

lemma *boundedUn*: $\text{bounded } (A \text{ Un } B) \longleftrightarrow (\text{bounded } A \wedge \text{bounded } B)$
 ⟨proof⟩

lemma *boundedIncLevel*: $\text{bounded } (\text{incLevel } ' A) \longleftrightarrow (\text{bounded } A)$

<proof>

lemma *boundedInsert*: *bounded (insert a B) \longleftrightarrow (bounded B)*
<proof>

lemma *inheritedBounded*: *inherited subs bounded*
<proof>

4.12 founded

definition

founded :: [*'a \Rightarrow 'a set, 'a \Rightarrow bool, (nat * 'a) set*] \Rightarrow *bool* **where**
founded subs Pred = (%A. !(n,delta):A. terminal subs delta \longrightarrow Pred delta)

lemma *foundedD*: *founded subs P (tree subs delta) \Longrightarrow terminal subs delta \Longrightarrow P delta*
<proof>

lemma *foundedMono*: *[[founded subs P A; $\forall x. P x \longrightarrow Q x$]] \Longrightarrow founded subs Q A*
<proof>

lemma *foundedSubs*: *founded subs P (tree subs Γ) \Longrightarrow sigma \in subs Γ \Longrightarrow founded subs P (tree subs sigma)*
<proof>

4.13 Inherited Properties- founded

lemma *foundedInsert*: *\neg terminal subs delta \Longrightarrow founded subs P (insert (n,delta) B) = (founded subs P B)*
<proof>

lemma *foundedUn*: *(founded subs P (A Un B)) = (founded subs P A \wedge founded subs P B)*
<proof>

lemma *foundedIncLevel*: *founded subs P (incLevel ' A) = (founded subs P A)*
<proof>

lemma *foundedEmpty*: *founded subs P {}*
<proof>

lemma *inheritedFounded*: *inherited subs (founded subs P)*
<proof>

4.14 Inherited Properties- finite

lemma *finiteUn*: *finite (A Un B) = (finite A \wedge finite B)*
<proof>

lemma *finiteIncLevel*: $finite (incLevel \ 'A) = finite \ A$
 ⟨proof⟩

lemma *inheritedFinite*: *inherited subs finite*
 ⟨proof⟩

4.15 path: follows a failing inherited property through tree

definition

failingSub :: [*a* ⇒ *a* set, (nat * *a*) set ⇒ bool, *a*] ⇒ *a* **where**
failingSub *subs* *P* $\gamma \equiv (SOME \ sigma. (sigma:subs \ \gamma \wedge \sim P(tree \ subs \ sigma)))$

lemma *failingSubProps*:

[[*inherited subs* *P*; ¬ *P* (tree *subs* γ); ¬ *terminal subs* γ ; *fans subs*]]
 ⇒ *failingSub* *subs* *P* $\gamma \in subs \ \gamma \wedge \neg P (tree \ subs (failingSub \ subs \ P \ \gamma))$
 ⟨proof⟩

lemma *failingSubFailsI*:

[[*inherited subs* *P*; ¬ *P* (tree *subs* γ); ¬ *terminal subs* γ ; *fans subs*]]
 ⇒ ¬ *P* (tree *subs* (*failingSub* *subs* *P* γ))
 ⟨proof⟩

lemmas *failingSubFailsE* = *failingSubFailsI*[*THEN notE*]

lemma *failingSubSubs*:

[[*inherited subs* *P*; ¬ *P* (tree *subs* γ); ¬ *terminal subs* γ ; *fans subs*]]
 ⇒ *failingSub* *subs* *P* $\gamma \in subs \ \gamma$
 ⟨proof⟩

primrec *path* :: [*a* ⇒ *a* set, *a*, (nat * *a*) set ⇒ bool, nat] ⇒ *a*

where

path0: *path* *subs* γ *P* 0 = γ
 | *pathSuc*: *path* *subs* γ *P* (*Suc* *n*) = (if *terminal subs* (*path* *subs* γ *P* *n*)
 then *path* *subs* γ *P* *n*
 else *failingSub* *subs* *P* (*path* *subs* γ *P* *n*))

lemma *pathFailsP*:

[[*inherited subs* *P*; *fans subs*; ¬ *P* (tree *subs* γ)]]
 ⇒ ¬ *P* (tree *subs* (*path* *subs* γ *P* *n*))
 ⟨proof⟩

lemmas *PpathE* = *pathFailsP*[*THEN notE*]

lemma *pathTerminal*:

[[*inherited subs* *P*; *fans subs*; *terminal subs* γ]]
 ⇒ *terminal subs* (*path* *subs* γ *P* *n*)
 ⟨proof⟩

lemma *pathStarts*: $\text{path subs } \gamma P 0 = \gamma$
 ⟨proof⟩

lemma *pathSubs*:
 $\llbracket \text{inherited subs } P; \text{ fans subs}; \neg P (\text{tree subs } \gamma); \neg \text{terminal subs } (\text{path subs } \gamma P n) \rrbracket$
 $\implies \text{path subs } \gamma P (\text{Suc } n) \in \text{subs } (\text{path subs } \gamma P n)$
 ⟨proof⟩

lemma *pathStops*: $\text{terminal subs } (\text{path subs } \gamma P n) \implies \text{path subs } \gamma P (\text{Suc } n) = \text{path subs } \gamma P n$
 ⟨proof⟩

4.16 Branch

definition

$\text{branch} :: ['a \Rightarrow 'a \text{ set}, 'a, \text{nat} \Rightarrow 'a] \Rightarrow \text{bool}$ **where**
 $\text{branch subs } \Gamma f \iff f 0 = \Gamma$
 $\wedge (!n . \text{terminal subs } (f n) \longrightarrow f (\text{Suc } n) = f n)$
 $\wedge (!n . \neg \text{terminal subs } (f n) \longrightarrow f (\text{Suc } n) \in \text{subs } (f n))$

lemma *branch0*: $\text{branch subs } \Gamma f \implies f 0 = \Gamma$
 ⟨proof⟩

lemma *branchStops*: $\text{branch subs } \Gamma f \implies \text{terminal subs } (f n) \implies f (\text{Suc } n) = f n$
 ⟨proof⟩

lemma *branchSubs*: $\text{branch subs } \Gamma f \implies \neg \text{terminal subs } (f n) \implies f (\text{Suc } n) \in \text{subs } (f n)$
 ⟨proof⟩

lemma *branchI*: $\llbracket f 0 = \Gamma; \forall n . \text{terminal subs } (f n) \longrightarrow f (\text{Suc } n) = f n; \forall n . \neg \text{terminal subs } (f n) \longrightarrow f (\text{Suc } n) \in \text{subs } (f n) \rrbracket \implies \text{branch subs } \Gamma f$
 ⟨proof⟩

lemma *branchTerminalPropagates*: $\text{branch subs } \Gamma f \implies \text{terminal subs } (f m) \implies \text{terminal subs } (f (m + n))$
 ⟨proof⟩

lemma *branchTerminalMono*:
 $\text{branch subs } \Gamma f \implies m < n \implies \text{terminal subs } (f m) \implies \text{terminal subs } (f n)$
 ⟨proof⟩

lemma *branchPath*:
 $\llbracket \text{inherited subs } P; \text{ fans subs}; \neg P (\text{tree subs } \gamma) \rrbracket$
 $\implies \text{branch subs } \gamma (\text{path subs } \gamma P)$
 ⟨proof⟩

4.17 failing branch property: abstracts path defn

lemma *failingBranchExistence*:

$\llbracket \text{inherited subs } P; \text{ fans subs}; \sim P(\text{tree subs } \gamma) \rrbracket$
 $\implies \exists f . \text{branch subs } \gamma f \wedge (\forall n . \neg P(\text{tree subs } (f n)))$
 $\langle \text{proof} \rangle$

definition

$\text{infBranch} :: ['a \Rightarrow 'a \text{ set}, 'a, \text{nat} \Rightarrow 'a] \Rightarrow \text{bool}$ **where**
 $\text{infBranch subs } \Gamma f \longleftrightarrow f 0 = \Gamma \wedge (\forall n . f (\text{Suc } n) \in \text{subs } (f n))$

lemma *infBranchI*: $\llbracket f 0 = \Gamma; \forall n . f (\text{Suc } n) \in \text{subs } (f n) \rrbracket \implies \text{infBranch subs } \Gamma f$
 $\langle \text{proof} \rangle$

4.18 Tree induction principles

lemma *boundedTreeInduction'*:

$\llbracket \text{fans subs};$
 $\quad \forall \text{delta} . \neg \text{terminal subs delta} \longrightarrow (\forall \text{sigma} \in \text{subs delta} . P \text{ sigma}) \longrightarrow P \text{ delta}$
 \rrbracket
 $\implies \forall \Gamma . \text{boundedBy } m (\text{tree subs } \Gamma) \longrightarrow \text{founded subs } P (\text{tree subs } \Gamma) \longrightarrow P \Gamma$
 $\langle \text{proof} \rangle$

lemma *boundedTreeInduction*:

$\llbracket \text{fans subs};$
 $\quad \text{bounded } (\text{tree subs } \Gamma); \text{founded subs } P (\text{tree subs } \Gamma);$
 $\quad \bigwedge \text{delta} . \llbracket \neg \text{terminal subs delta}; (\forall \text{sigma} \in \text{subs delta} . P \text{ sigma}) \rrbracket \implies P \text{ delta}$
 $\rrbracket \implies P \Gamma$
 $\langle \text{proof} \rangle$

lemma *boundedTreeInduction2*:

$\llbracket \text{fans subs};$
 $\quad \forall \text{delta} . (\forall \text{sigma} \in \text{subs delta} . P \text{ sigma}) \longrightarrow P \text{ delta} \rrbracket$
 $\implies \text{boundedBy } m (\text{tree subs } \Gamma) \longrightarrow P \Gamma$
 $\langle \text{proof} \rangle$

end

5 Completeness

theory *Completeness*

imports *Tree Sequents*

begin

5.1 pseq: type represents a processed sequent

type-synonym *atom* = (*signs* * *predicate* * *vbl list*)

type-synonym *nform* = (*nat* * *formula*)

type-synonym *pseq* = (*atom list* * *nform list*)

definition

$sequent :: pseq \Rightarrow formula\ list$ **where**
 $sequent = (\lambda(atoms,nforms) . map\ snd\ nforms\ @\ map\ (\lambda(z,p,vs) . FAtom\ z\ p\ vs)\ atoms)$

definition

$pseq :: formula\ list \Rightarrow pseq$ **where**
 $pseq\ fs = ([], map\ (\lambda f.(0,f))\ fs)$

definition $atoms :: pseq \Rightarrow atom\ list$ **where** $atoms = fst$

definition $nforms :: pseq \Rightarrow nform\ list$ **where** $nforms = snd$

5.2 subs: SATAxiom

definition

$SATAxiom :: formula\ list \Rightarrow bool$ **where**
 $SATAxiom\ fs \equiv (\exists n\ vs . FAtom\ Pos\ n\ vs \in set\ fs \wedge FAtom\ Neg\ n\ vs \in set\ fs)$

5.3 subs: a CutFreePC justifiable backwards proof step

definition

$subsFAtom :: [atom\ list,(nat * formula)\ list,signs,predicate,vbl\ list] \Rightarrow pseq\ set$
where
 $subsFAtom\ atms\ nAs\ z\ P\ vs = \{ ((z,P,vs)\#atms,nAs) \}$

definition

$subsFConj :: [atom\ list,(nat * formula)\ list,signs,formula,formula] \Rightarrow pseq\ set$
where
 $subsFConj\ atms\ nAs\ z\ A0\ A1 =$
 (case z of
 Pos $\Rightarrow \{ (atms,(0,A0)\#nAs),(atms,(0,A1)\#nAs) \}$
 Neg $\Rightarrow \{ (atms,(0,A0)\#(0,A1)\#nAs) \}$)

definition

$subsFAll :: [atom\ list,(nat * formula)\ list,nat,signs,formula,vbl\ set] \Rightarrow pseq\ set$
where
 $subsFAll\ atms\ nAs\ n\ z\ A\ frees =$
 (case z of
 Pos $\Rightarrow \{ let\ v = freshVar\ frees\ in\ (atms,(0,instanceF\ v\ A)\#nAs) \}$
 Neg $\Rightarrow \{ (atms,(0,instanceF\ (X\ n)\ A)\#nAs\ @\ [(Suc\ n,FAll\ Neg\ A)]) \}$)

definition

$subs :: pseq \Rightarrow pseq\ set$ **where**
 $subs = (\lambda pseq .$
 if $SATAxiom\ (sequent\ pseq)$ then
 $\{ \}$
 else let $(atms,nforms) = pseq$
 in case nforms of
 $[] \Rightarrow \{ \}$
 $| nA\#nAs \Rightarrow let\ (n,A) = nA$

$$\text{in } (\text{case } A \text{ of}$$

$$\quad \text{FAtom } z \ P \ vs \Rightarrow \text{subsFAtom } \text{atms } nAs \ z \ P \ vs$$

$$\quad | \ \text{FConj } z \ A0 \ A1 \Rightarrow \text{subsFConj } \text{atms } nAs \ z \ A0 \ A1$$

$$\quad | \ \text{FAll } z \ A \quad \Rightarrow \text{subsFAll } \text{atms } nAs \ n \ z \ A$$

$$(\text{freeVarsFL } (\text{sequent } pseq))))$$

5.4 proofTree(Gamma) says whether tree(Gamma) is a proof

definition

$$\text{proofTree} :: (\text{nat} * \text{pseq}) \text{ set} \Rightarrow \text{bool} \ \mathbf{where}$$

$$\text{proofTree } A \iff \text{bounded } A \wedge \text{founded subs } (\text{SATAxiom} \circ \text{sequent}) \ A$$

5.5 path: considers, contains, costBarrier

definition

$$\text{considers} :: [\text{nat} \Rightarrow \text{pseq}, \text{nat} * \text{formula}, \text{nat}] \Rightarrow \text{bool} \ \mathbf{where}$$

$$\text{considers } f \ nA \ n = (\text{case } (\text{snd } (f \ n)) \ \text{of } [] \Rightarrow \text{False} \mid x \# xs \Rightarrow x = nA)$$

definition

$$\text{contains} :: [\text{nat} \Rightarrow \text{pseq}, \text{nat} * \text{formula}, \text{nat}] \Rightarrow \text{bool} \ \mathbf{where}$$

$$\text{contains } f \ nA \ n \iff nA \in \text{set } (\text{snd } (f \ n))$$

abbreviation (*input*) $\text{power3} \equiv \text{power } (3 :: \text{nat})$

definition

$$\text{costBarrier} :: [\text{nat} * \text{formula}, \text{pseq}] \Rightarrow \text{nat} \ \mathbf{where}$$

$$\text{costBarrier } nA = (\lambda(\text{atms}, nAs).$$

$$\quad \text{let barrier} = \text{takeWhile } (\lambda x. \ nA \neq x) \ nAs$$

$$\quad \text{in let costs} = \text{map } (\text{power3} \circ \text{size} \circ \text{snd}) \ \text{barrier}$$

$$\quad \text{in sumList costs})$$

5.6 path: eventually

definition

$$\text{EV} :: [\text{nat} \Rightarrow \text{bool}] \Rightarrow \text{bool} \ \mathbf{where}$$

$$\text{EV } f \equiv (\exists n . f \ n)$$

5.7 path: counter model

definition

$$\text{counterM} :: (\text{nat} \Rightarrow \text{pseq}) \Rightarrow \text{object set} \ \mathbf{where}$$

$$\text{counterM } f \equiv \text{range } \text{obj}$$

definition

$$\text{counterEvalP} :: (\text{nat} \Rightarrow \text{pseq}) \Rightarrow \text{predicate} \Rightarrow \text{object list} \Rightarrow \text{bool} \ \mathbf{where}$$

$$\text{counterEvalP } f = (\lambda p \ \text{args} . ! \ i . \neg (\text{EV } (\text{contains } f \ (i, \text{FAtom } \text{Pos } p \ (\text{map } (X \circ \text{inv } \text{obj}) \ \text{args}))))))$$

definition

counterModel :: (nat \Rightarrow pseq) \Rightarrow model **where**
counterModel f = Abs-model (counterM f, counterEvalP f)

primrec *counterAssign* :: vbl \Rightarrow object
where *counterAssign* (X n) = obj n

5.8 subs: finite

lemma *finite-subs*: finite (subs γ)
 <proof>

lemma *fansSubs*: fans subs
 <proof>

lemma *subs-def2*:

\neg SATAxiom (sequent γ) \Longrightarrow
 subs γ =
 (case nforms γ of
 [] \Rightarrow {}
 | nA#nAs \Rightarrow let (n,A) = nA
 in (case A of
 FAtom z P vs \Rightarrow subsFAtom (atoms γ) nAs z P vs
 FConj z A0 A1 \Rightarrow subsFConj (atoms γ) nAs z A0 A1
 FAll z A \Rightarrow subsFAll (atoms γ) nAs n z A (freeVarsFL
 (sequent γ))))
 <proof>

5.9 inherited: proofTree

lemma *proofTree-def2*: proofTree = (λx . bounded x \wedge founded subs (SATAxiom \circ sequent) x)
 <proof>

lemma *inheritedProofTree*: inherited subs proofTree
 <proof>

lemma *proofTreeI*: \llbracket bounded A; founded subs (SATAxiom \circ sequent) A $\rrbracket \Longrightarrow$ proofTree A
 <proof>

lemma *proofTreeBounded*: proofTree A \Longrightarrow bounded A
 <proof>

lemma *proofTreeTerminal*:

\llbracket proofTree A; (n, delta) \in A; terminal subs delta $\rrbracket \Longrightarrow$ SATAxiom (sequent delta)
 <proof>

5.10 pseq: lemma

lemma *snd-o-Pair*: $(snd \circ (Pair\ x)) = (\lambda x. x)$
<proof>

lemma *sequent-pseq*: $sequent\ (pseq\ fs) = fs$
<proof>

5.11 SATAxiom: proofTree

lemma *SATAxiomTerminal*[*rule-format*]: $SATAxiom\ (sequent\ \gamma) \implies terminal\ subs\ \gamma$
<proof>

lemma *SATAxiomBounded*: $SATAxiom\ (sequent\ \gamma) \implies bounded\ (tree\ subs\ \gamma)$
<proof>

lemma *SATAxiomFounded*: $SATAxiom\ (sequent\ \gamma) \implies founded\ subs\ (SATAxiom\ \circ\ sequent)\ (tree\ subs\ \gamma)$
<proof>

lemma *SATAxiomProofTree*: $SATAxiom\ (sequent\ \gamma) \implies proofTree\ (tree\ subs\ \gamma)$
<proof>

lemma *SATAxiomEq*: $(proofTree\ (tree\ subs\ \gamma) \wedge terminal\ subs\ \gamma) = SATAxiom\ (sequent\ \gamma)$
<proof>

5.12 SATAxioms are deductions: - needed

lemma *SAT-deduction*:
 assumes *SATAxiom* *x*
 shows $x \in deductions\ CutFreePC$
<proof>

5.13 proofTrees are deductions: subs respects rules - messy start and end

lemma *subsJustified'*:
 notes *ss = subs-def2 nforms-def Let-def atoms-def sequent-def subsFAtom-def subsFConj-def subsFAll-def*
 shows $\llbracket \neg SATAxiom\ (sequent\ (ats,\ (n,\ f)\ \# list));$
 $\neg terminal\ subs\ (ats,\ (n,\ f)\ \# list);$
 $\forall sigma \in subs\ (ats,\ (n,\ f)\ \# list). sequent\ sigma \in deductions\ CutFreePC \rrbracket$
 $\implies sequent\ (ats,\ (n,\ f)\ \# list) \in deductions\ CutFreePC$
<proof>

lemma *subsJustified*:
 assumes $\neg terminal\ subs\ \gamma$
 and $\forall sigma \in subs\ \gamma. sequent\ sigma \in deductions\ (CutFreePC)$

shows *sequent* $\gamma \in \text{deductions (CutFreePC)}$
<proof>

5.14 proofTrees are deductions: instance of boundedTreeInduction

lemmas *proofTreeD* = *proofTree-def* [THEN *iffD1*]

lemma *proofTreeDeductionD*:
assumes *proofTree(tree subs γ)*
shows *sequent* $\gamma \in \text{deductions (CutFreePC)}$
<proof>

5.15 contains, considers:

lemma *contains-def2*: *contains f iA n* = (*iA* \in *set (nforms (f n))*)
<proof>

lemma *considers-def2*: *considers f iA n* = (\exists *nAs* . *nforms (f n)* = *iA#nAs*)
<proof>

lemmas *containsI* = *contains-def2*[THEN *iffD2*]

5.16 path: nforms = [] implies

lemma *nformsNoContains*:
nforms (f n) = [] \implies \neg *contains f iA n*
<proof>

lemma *nformsTerminal*: *nforms (f n)* = [] \implies *terminal subs (f n)*
<proof>

lemma *nformsStops*:
[[*branch subs γ f*; $\bigwedge n$. \neg *proofTree (tree subs (f n))*; *nforms (f n)* = []]
 \implies *nforms (f (Suc n))* = [] \wedge *atoms (f (Suc n))* = *atoms (f n)*
<proof>

5.17 path: cases

lemma *terminalNFormCases*: *terminal subs (f n)* \vee (\exists *i A nAs* . *nforms (f n)* = *(i,A)#nAs*)
<proof>

lemma *cases[elim-format]*: *terminal subs (f n)* \vee (\neg (*terminal subs (f n)* \wedge (\exists *i A nAs* . *nforms (f n)* = *(i,A)#nAs*)))
<proof>

5.18 path: contains not terminal and propagate condition

lemma *containsNotTerminal*:

assumes *branch subs* $\gamma f \neg \text{proofTree (tree subs (f n)) contains f iA n}$
shows $\neg \text{terminal subs (f n)}$
 $\langle \text{proof} \rangle$

lemma *containsPropagates*:
assumes *branch subs* γf
and $\bigwedge n. \neg \text{proofTree (tree subs (f n))}$
and *contains f iA n*
shows *contains f iA (Suc n) \vee considers f iA n*
 $\langle \text{proof} \rangle$

5.19 termination: (for EV contains implies EV considers)

lemma *terminationRule* [rule-format]:
 $! n. P n \longrightarrow (\neg (P (Suc n)) \mid (P (Suc n) \wedge \text{msrFn (Suc n) < (msrFn::nat} \Rightarrow$
 $\text{nat) n})) \Longrightarrow P m \longrightarrow (\exists n. P n \wedge \neg (P (Suc n)))$
(is $- \Longrightarrow ?P m)$
 $\langle \text{proof} \rangle$

5.20 costBarrier: lemmas

5.21 costBarrier: exp3 lemmas - bit specific...

lemma *exp1*: $\text{power3 } A + \text{power3 } B < 3 * (\text{power3 } A * \text{power3 } B)$
 $\langle \text{proof} \rangle$

lemma *exp1'*: $\text{power3 } A < 3 * ((\text{power3 } A) * (\text{power3 } B)) + C$
 $\langle \text{proof} \rangle$

lemma *exp2*: $\text{Suc } 0 < 3 * \text{power3 } (B)$
 $\langle \text{proof} \rangle$

5.22 costBarrier: decreases whilst contains and unconsiders

lemma *costBarrierDecreases'*:
notes *ss = subs-def2 nforms-def subsFAtom-def subsFConj-def subsFAll-def costBarrier-def*
shows $\llbracket \neg \text{SATAxiom (sequent (a, (num, fm) \# list)); iA \neq (num, fm);}$
 $\neg \text{proofTree (tree subs (a, (num, fm) \# list));}$
 $\text{fSucn} \in \text{subs (a, (num, fm) \# list); iA} \in \text{set list} \rrbracket$
 $\Longrightarrow \text{costBarrier iA fSucn} < \text{costBarrier iA (a, (num, fm) \# list)}$
 $\langle \text{proof} \rangle$

lemma *costBarrierDecreases*:
assumes *branch subs* γf
and $\bigwedge n. \neg \text{proofTree (tree subs (f n))}$
and *contains f iA n*
and $\neg \text{considers f iA n}$
shows $\text{costBarrier iA (f (Suc n))} < \text{costBarrier iA (f n)}$

<proof>

5.23 path: EV contains implies EV considers

lemma *considersContains*: $\text{considers } f \ iA \ n \implies \text{contains } f \ iA \ n$
<proof>

lemma *containsConsiders*:
 assumes *branch subs* $\gamma \ f$
 and $\bigwedge n. \neg \text{proofTree } (\text{tree } \text{subs } (f \ n))$
 and $EV \ (\text{contains } f \ iA)$
 shows $EV \ (\text{considers } f \ iA)$
<proof>

5.24 EV contains: common lemma

lemma *lemmaA*:
 assumes *branch subs* $\gamma \ f$
 and $\bigwedge n. \neg \text{proofTree } (\text{tree } \text{subs } (f \ n))$
 and $EV \ (\text{contains } f \ (i, A))$
 obtains $n \ nAs$ **where** $\neg \text{SATAxiom } (\text{sequent } (f \ n))$
 $n\text{forms } (f \ n) = (i, A) \ \# \ nAs \ f \ (\text{Suc } n) \in \text{subs } (f \ n)$
<proof>

5.25 EV contains: FConj,FDisj,FAll

lemma *evContainsConj*:
 assumes $EV \ (\text{contains } f \ (i, \text{FConj Pos } A0 \ A1))$
 and *branch subs* $\gamma \ f$
 and $\bigwedge n. \neg \text{proofTree } (\text{tree } \text{subs } (f \ n))$
 shows $EV \ (\text{contains } f \ (0, A0)) \vee EV \ (\text{contains } f \ (0, A1))$
<proof>

lemma *evContainsDisj*:
 assumes $EV \ (\text{contains } f \ (i, \text{FConj Neg } A0 \ A1))$
 and *branch subs* $\gamma \ f$
 and $\bigwedge n. \neg \text{proofTree } (\text{tree } \text{subs } (f \ n))$
 shows $EV \ (\text{contains } f \ (0, A0)) \wedge EV \ (\text{contains } f \ (0, A1))$
<proof>

lemma *evContainsAll*:
 assumes $EV \ (\text{contains } f \ (i, \text{FAll Pos } \text{body}))$
 and *branch subs* $\gamma \ f$
 and $\bigwedge n. \neg \text{proofTree } (\text{tree } \text{subs } (f \ n))$
 shows $\exists v. EV \ (\text{contains } f \ (0, \text{instanceF } v \ \text{body}))$
<proof>

lemma *evContainsEx-instance*:
 assumes $EV \ (\text{contains } f \ (i, \text{FAll Neg } \text{body}))$
 and *branch subs* $\gamma \ f$

and $\bigwedge n. \neg \text{proofTree } (\text{tree subs } (f n))$
shows $EV \text{ (contains } f \text{ (0,instanceF } (X i) \text{ body))}$
 $\langle \text{proof} \rangle$

lemma *evContainsEx-repeat*:
assumes $EV \text{ (contains } f \text{ (i, Fall Neg body))}$
and $\text{branch subs } \gamma f$
and $\bigwedge n. \neg \text{proofTree } (\text{tree subs } (f n))$
shows $EV \text{ (contains } f \text{ (Suc i, Fall Neg body))}$
 $\langle \text{proof} \rangle$

5.26 EV contains: lemmas (temporal related)

5.27 EV contains: FAtoms

lemma *notTerminalNotSATAxiom*: $\neg \text{terminal subs } \gamma \implies \neg \text{SATAxiom } (\text{sequent } \gamma)$
 $\langle \text{proof} \rangle$

lemma *notTerminalNforms*: $\neg \text{terminal subs } (f n) \implies \text{nforms } (f n) \neq []$
 $\langle \text{proof} \rangle$

lemma *atomsPropagate*:
assumes $f: \text{branch subs } \gamma f$ **and** $x: x \in \text{set } (\text{atoms } (f n))$
shows $x \in \text{set } (\text{atoms } (f \text{ (Suc } n)))$
 $\langle \text{proof} \rangle$

5.28 EV contains: FEx cases

lemma *evContainsEx0-allRepeats*:
 $\llbracket \text{branch subs } \gamma f; \bigwedge n. \neg \text{proofTree } (\text{tree subs } (f n));$
 $EV \text{ (contains } f \text{ (0, Fall Neg A))} \rrbracket$
 $\implies EV \text{ (contains } f \text{ (i, Fall Neg A))}$
 $\langle \text{proof} \rangle$

lemma *evContainsEx0-allInstances*:
 $\llbracket \text{branch subs } \gamma f; \bigwedge n. \neg \text{proofTree } (\text{tree subs } (f n));$
 $EV \text{ (contains } f \text{ (0, Fall Neg A))} \rrbracket$
 $\implies EV \text{ (contains } f \text{ (0, instanceF } (X i) A))$
 $\langle \text{proof} \rangle$

5.29 pseq: creates initial pseq

lemma *containsPSeq0D*: $\text{branch subs } (\text{pseq } fs) f \implies \text{contains } f \text{ (i,A) } 0 \implies i=0$
 $\langle \text{proof} \rangle$

5.30 EV contains: contain any (i,FEx y) means contain all (i,FEx y)

lemma *natPredCases*:

obtains $\forall n. P n \mid \neg P 0 \mid n$ **where** $P n \rightarrow P (Suc n)$
 ⟨proof⟩

lemma *containsNotTerminal'*:

[[*branch subs* γf ; $\bigwedge n. \neg proofTree (tree\ subs\ (f\ n))$; *contains f iA n*]] $\implies \neg$
 (*terminal subs (f n)*)
 ⟨proof⟩

lemma *evContainsExSuc-containsEx*:

assumes *branch subs (pseq fs) f*
and $\bigwedge n. \neg proofTree (tree\ subs\ (f\ n))$
and *EV (contains f (Suc i, Fall Neg body))*
shows *EV (contains f (i, Fall Neg body))*
 ⟨proof⟩

5.31 EV contains: contain any (i,FEx y) means contain all (i,FEx y)

lemma *evContainsEx-containsEx0*:

[[*branch subs (pseq fs) f*; $\bigwedge n. \neg proofTree (tree\ subs\ (f\ n))$;
EV (contains f (i, Fall Neg A))]]
 \implies *EV (contains f (0, Fall Neg A))*
 ⟨proof⟩

lemma *evContainsExval*:

[[*EV (contains f (i, Fall Neg body))*; *branch subs (pseq fs) f*;
 $\bigwedge n. \neg proofTree (tree\ subs\ (f\ n))$]]
 \implies *EV (contains f (0, instanceF v body))*
 ⟨proof⟩

5.32 EV contains: atoms

lemma *atomsInSequentI*:

$(z, P, vs) \in set\ (fst\ ps) \implies FAtom\ z\ P\ vs \in set\ (sequent\ ps)$
 ⟨proof⟩

lemma *evContainsAtom1*:

assumes *branch subs (pseq fs) f*
and $\bigwedge n. \neg proofTree (tree\ subs\ (f\ n))$
and *EV (contains f (i, FAtom z P vs))*
shows $\exists n. (z, P, vs) \in set\ (fst\ (f\ n))$
 ⟨proof⟩

lemmas *atomsPropagate' = atomsPropagate[simplified atoms-def, simplified]*

lemma *evContainsAtom*:

assumes *branch subs (pseq fs) f*
and $\bigwedge n. \neg proofTree (tree\ subs\ (f\ n))$
and *EV (contains f (i, FAtom z P vs))*

shows $\exists n. \forall m. \text{FAtom } z \text{ P } vs \in \text{set } (\text{sequent } (f (n + m)))$
 ⟨proof⟩

lemma *notEvContainsBothAtoms*:
 $\llbracket \text{branch subs } (\text{pseq fs}) f; \bigwedge n. \neg \text{proofTree } (\text{tree subs } (f n)) \rrbracket$
 $\implies \neg \text{EV } (\text{contains } f (i, \text{FAtom Pos } p \text{ vs})) \vee$
 $\neg \text{EV } (\text{contains } f (j, \text{FAtom Neg } p \text{ vs}))$
 ⟨proof⟩

5.33 counterModel: lemmas

lemma *counterModelInRepn*: $(\text{counterM } f, \text{counterEvalP } f) \in \text{model}$
 ⟨proof⟩

lemmas *Abs-counterModel-inverse = counterModelInRepn*[*THEN Abs-model-inverse*]

lemma *inv-obj-obj*: $\text{inv obj } (\text{obj } n) = n$
 ⟨proof⟩

lemma *map-X-map-counterAssign* [*simp*]: $\text{map } X (\text{map } (\text{inv obj}) (\text{map } \text{counterAssign } xs)) = xs$
 ⟨proof⟩

lemma *objectsCounterModel*: $\text{objects } (\text{counterModel } f) = \{ z . \exists i . z = \text{obj } i \}$
 ⟨proof⟩

lemma *inCounterM*: $\text{counterAssign } v \in \text{objects } (\text{counterModel } f)$
 ⟨proof⟩

lemma *evalPCounterModel*: $M = \text{counterModel } f \implies \text{evalP } M = \text{counterEvalP } f$
 ⟨proof⟩

5.34 counterModel: all path formula value false - step by step

lemma *path-evalF*:
assumes $\text{branch subs } (\text{pseq fs}) f \bigwedge n. \neg \text{proofTree } (\text{tree subs } (f n))$
shows $(\exists i . \text{EV } (\text{contains } f (i, A))) \implies \neg \text{evalF } (\text{counterModel } f) \text{ counterAssign } A$
 ⟨proof⟩

5.35 adequacy

lemma *counterAssignModelAssign*: $\text{counterAssign} \in \text{modelAssigns } (\text{counterModel } \gamma)$
 ⟨proof⟩

lemma *branch-contains-initially*: $\text{branch subs } (\text{pseq fs}) f \implies x \in \text{set } fs \implies \text{contains } f (0, x) 0$
 ⟨proof⟩

lemma *validProofTree*:
assumes $\neg \text{proofTree } (\text{tree subs } (\text{pseq fs}))$
shows $\neg \text{validS fs}$
 $\langle \text{proof} \rangle$

theorem *adequacy[simplified sequent-pseq]*: $\text{validS fs} \implies (\text{sequent } (\text{pseq fs})) \in \text{deductions CutFreePC}$
 $\langle \text{proof} \rangle$

end

6 Soundness

theory *Soundness* **imports** *Completeness* **begin**

lemma *permutation-validS*: $\text{mset fs} = \text{mset gs} \implies (\text{validS fs} = \text{validS gs})$
 $\langle \text{proof} \rangle$

lemma *modelAssigns-vblcase*: $\varphi \in \text{modelAssigns } M \implies x \in \text{objects } M \implies \text{vblcase } x \varphi \in \text{modelAssigns } M$
 $\langle \text{proof} \rangle$

lemma *soundnessFAll*:
assumes $u \notin \text{freeVarsFL } (\text{FAll Pos } A \# \text{Gamma})$
and $\text{validS } (\text{instanceF } u A \# \text{Gamma})$
shows $\text{validS } (\text{FAll Pos } A \# \text{Gamma})$
 $\langle \text{proof} \rangle$

lemma *soundnessFEx*: $\text{validS } (\text{instanceF } x A \# \text{Gamma}) \implies \text{validS } (\text{FAll Neg } A \# \text{Gamma})$
 $\langle \text{proof} \rangle$

lemma *soundnessFCut*: $\llbracket \text{validS } (C \# \text{Gamma}); \text{validS } (\text{FNot } C \# \text{Delta}) \rrbracket \implies \text{validS } (\text{Gamma} @ \text{Delta})$
 $\langle \text{proof} \rangle$

lemma *soundness*: $\text{fs} : \text{deductions}(PC) \implies \text{validS fs}$
 $\langle \text{proof} \rangle$

theorem *completeness*: $\text{fs} \in \text{deductions } (PC) \longleftrightarrow \text{validS fs}$
 $\langle \text{proof} \rangle$

end