Completeness for FOL

James Margetson, ported by Tom Ridge

March 17, 2025

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 monotonic 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* ⟹ (∀ *x*. *count-list A x* = *count-list B x*)
⟨*proof*⟩

lemma *count-0*: (∀ *x*. *count-list B x* = *0*) = (*B* = [])
⟨*proof*⟩

lemma *count-Suc*: *count-list B a* = *Suc m* ⟹ *a* ∈ *set B*
⟨*proof*⟩

lemma *count-perm*: !! *B*. (∀ *x*. *count-list A x* = *count-list B x*) ⟹ *mset A* = *mset B*
⟨*proof*⟩

lemma *perm-count-conv*: *mset A* = *mset B* ⟷ (∀ *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* ∈ *set ys* ⟹ *remdups* (*ws*@*y*#*ys*) = *remdups* (*ws*@*ys*)
⟨*proof*⟩

lemma *perm-contr′*:
assumes *perm*: $\bigwedge xs\ ys.\ mset\ xs = mset\ ys \Longrightarrow (P\ xs = P\ ys)$
and *contr′*: $\bigwedge x\ xs.\ P(x\#x\#xs) = P\ (x\#xs)$
shows $length\ xs = n \Longrightarrow (P\ xs = P\ (remdups\ xs))$
⟨*proof*⟩

lemma *perm-contr*:
assumes *perm*: $\bigwedge xs\ ys.\ mset\ xs = mset\ ys \Longrightarrow (P\ xs = P\ ys)$
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*: $length\ (rem\ x\ xs) <= length\ xs$
⟨*proof*⟩

lemma *rem-notin*: $x \notin set\ xs \Longrightarrow rem\ x\ xs = xs$
⟨*proof*⟩

lemma *perm-weak-filter′*:
assumes *perm*: $\bigwedge xs\ ys.\ mset\ xs = mset\ ys \Longrightarrow (P\ xs = P\ ys)$
and *weak*: $\bigwedge x\ xs.\ P\ xs \Longrightarrow P\ (x\#xs)$
and *P*: $P\ (ys@filter\ Q\ xs)$
shows $P\ (ys@xs)$
⟨*proof*⟩

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

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

$\langle proof \rangle$

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*) $\notin$ *A*
 $\langle proof \rangle$

2.2 Summation

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

2.3 Termination Measure

primrec *sumList* :: *nat list* $\Rightarrow$ *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*−' {*a*}

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

lemma *preImage-insert*: *preImage f* (*insert a A*) = *pre f a* $\cup$ *preImage f A*
 $\langle proof \rangle$

lemma *equalOn-Un*: *equalOn* (*A* $\cup$ *B*) *f g* = (*equalOn A f g* $\wedge$ *equalOn B f g*)
 $\langle proof \rangle$

lemma *equalOnD*: *equalOn A f g* $\Longrightarrow$ ($\forall$ *x* $\in$ *A* . *f x* = *g x*)
 $\langle proof \rangle$

lemma *equalOnI*:(∀ $x \in A$. $f\ x = g\ x$) $\Longrightarrow$ *equalOn A f g*
⟨*proof*⟩

lemma *equalOn-UnD*: *equalOn* (*A Un B*) *f g* ==> *equalOn A f g* & *equalOn B f g*
⟨*proof*⟩

lemma *inj-inv-singleton*[*simp*]: ⟦ *inj f*; $f\ z = y$ ⟧ $\Longrightarrow$ $\{x.\ f\ x = y\} = \{z\}$
⟨*proof*⟩

lemma *finite-pre*[*simp*]: *inj f* $\Longrightarrow$ *finite* (*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 is we just identified vbls with nats

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

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

definition *zeroX* = *X 0*

primrec
nextX :: *vbl* ⇒ *vbl* **where**
nextX (*X n*) = *X* (*Suc n*)

definition
vblcase :: [*'a*,*vbl* ⇒ *'a*,*vbl*] ⇒ *'a* **where**
vblcase a f n ≡ (@*z*. (*n*=*zeroX* ⟶ *z*=*a*) ∧ (!*x*. *n*=*nextX x* ⟶ *z*=*f*(*x*)))

declare [[*case-translation vblcase zeroX nextX*]]

definition

freshVar :: *vbl set* ⇒ *vbl* **where**
freshVar vs = *X* (*LEAST n*. *n* ∉ *deX* ‘ *vs*)

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

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

lemma *ind*:
assumes *P zeroX* ⋀ *v*. *P v* ⟹ *P* (*nextX v*)
shows *P v*′
⟨*proof*⟩

lemma *zeroX-nextX*[*iff*]: *zeroX* ≠ *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* ∨ (∃ *y*. *x* = *nextX y*)
⟨*proof*⟩

lemma *vbl-casesE*: ⟦ *x* = *zeroX* ⟹ *P*; ⋀ *y*. *x* = *nextX y* ⟹ *P* ⟧ ⟹ *P*
⟨*proof*⟩

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

lemma *equalOn-vblcaseI*′: *equalOn* (*preImage nextX A*) *f g* ⟹ *equalOn A* (*vblcase x f*) (*vblcase x g*)
⟨*proof*⟩

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

lemma *X-deX-connection*: *X n* ∈ *A* = (*n* ∈ (*deX* ‘ *A*))
⟨*proof*⟩

lemma *finiteFreshVar*: *finite A* ⟹ *freshVar A* ∉ *A*

⟨*proof*⟩

lemma *freshVarI*: $[\![finite\ A;\ B \subseteq A]\!] \Longrightarrow freshVar\ A \notin B$
⟨*proof*⟩

lemma *freshVarI2*: $[\![finite\ A;\ \bigwedge x\ .\ x \notin A \Longrightarrow P\ x]\!] \Longrightarrow P\ (freshVar\ A)$
⟨*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* ⇒ *bool* ⇒ *bool*
where
sign Pos x = *x*
| *sign Neg x* = (¬ *x*)

primrec *invSign* :: *signs* ⇒ *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*]:
$[\![\bigwedge p\ vs.\ P\ (FAtom\ Pos\ p\ vs);$
$\bigwedge p\ vs.\ P\ (FAtom\ Neg\ p\ vs);$
$\bigwedge A\ B\ .\ [\![P\ A;\ P\ B]\!] \Longrightarrow P\ (FConj\ Pos\ A\ B);$
$\bigwedge A\ B\ .\ [\![P\ A;\ P\ B]\!] \Longrightarrow P\ (FConj\ Neg\ A\ B);$
$\bigwedge A\ .\ [\![P\ A]\!] \Longrightarrow P\ (FAll\ Pos\ A);$
$\bigwedge A\ .\ [\![P\ A]\!] \Longrightarrow P\ (FAll\ Neg\ A)$
$]\!] \Longrightarrow P\ A$
⟨*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* ⇒ *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* ⇒ *signs*
where
neg Pos = *Neg*
| *neg Neg* = *Pos*

primrec
dual :: [(*signs* ⇒ *signs*),(*signs* ⇒ *signs*),(*signs* ⇒ *signs*)] ⇒ *formula* ⇒ *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* ∘ *dual P Q R* = *dual* (*p* ∘ *P*) (*q* ∘ *Q*) (*r* ∘ *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* ⇒ *vbl set*
where
freeVarsFAtom: *freeVarsF* (*FAtom z P vs*) = *set vs*
| *freeVarsFConj*: *freeVarsF* (*FConj z A0 A1*) = (*freeVarsF A0*) ∪ (*freeVarsF A1*)

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

definition
freeVarsFL :: *formula list* ⇒ *vbl set* **where**
freeVarsFL Γ = *Union* (*freeVarsF* ‘ (*set* Γ))

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* (λ*v* . (*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* (λ*v*. *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* ⇒ *object*
where *inj-obj*: *inj obj*

3.8 model, non empty set and positive atom valuation

definition *model* = {*z* :: (*object set* ∗ ([*predicate*,*object list*] ⇒ *bool*)). (*fst z* ≠ {})}

typedef *model* = *model*
⟨*proof*⟩

definition
objects :: *model* ⇒ *object set* **where**
objects M = *fst* (*Rep-model M*)

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

lemma *objectsNonEmpty*: *objects M* ≠ {}
⟨*proof*⟩

lemma *modelsNonEmptyI*: *fst* (*Rep-model M*) ≠ {}
⟨*proof*⟩

3.9 Validity

primrec *evalF* :: [*model*,*vbl* ⇒ *object*,*formula*] ⇒ *bool*
where
evalFAtom: *evalF M φ* (*FAtom z P vs*) = *sign z* (*evalP M P* (*map φ vs*))
| *evalFConj*: *evalF M φ* (*FConj z A0 A1*) = *sign z* (*sign z* (*evalF M φ A0*) ∧ *sign z* (*evalF M φ A1*))
| *evalFAll*: *evalF M φ* (*FAll z body*) =
sign z (∀ *x*∈*objects M*. *sign z* (*evalF M* (λ*v* . (*case v of zeroX* ⇒ *x* | *nextX v* ⇒ *φ v*)) *body*))

definition
valid :: *formula* ⇒ *bool* **where**
valid F ≡ (∀ *M φ*. *evalF M φ F* = *True*)

lemma *evalF-FAll*: *evalF M φ* (*FAll Pos A*) = (∀ *x*∈*objects M*. (*evalF M* (*vblcase x φ*) *A*))
⟨*proof*⟩

lemma *evalF-FEx*: *evalF M φ* (*FAll Neg A*) = (∃ *x*∈*objects M*. (*evalF M* (*vblcase x φ*) *A*))
⟨*proof*⟩

lemma *evalF-arg2-cong*: $x = y \Longrightarrow evalF\ M\ p\ x = evalF\ M\ p\ y$
⟨*proof*⟩

lemma *evalF-FNot*: $!!\varphi.\ evalF\ M\ \varphi\ (FNot\ A) = (\neg\ evalF\ M\ \varphi\ A)$
⟨*proof*⟩

lemma *evalF-equiv*: $equalOn\ (freeVarsF\ A)\ f\ g \Longrightarrow evalF\ M\ f\ A = evalF\ M\ g\ A$
⟨*proof*⟩
lemma *evalF-subF-eq*: $evalF\ M\ \varphi\ (subF\ theta\ A) = evalF\ M\ (\varphi \circ theta)\ A$
⟨*proof*⟩

lemma *evalF-instance*: $evalF\ M\ \varphi\ (instanceF\ u\ A) = evalF\ M\ (vblcase\ (\varphi\ u)\ \varphi)\ A$
⟨*proof*⟩

lemma *instanceF-E*: $instanceF\ g\ formula \neq FAll\ signs\ formula$
⟨*proof*⟩

end

4 Sequents

theory *Sequents*
imports *Formula*
begin

type-synonym *sequent* = *formula list*

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

lemma *evalS-nil*[*simp*]: $evalS\ M\ \varphi\ [] = False$
⟨*proof*⟩

lemma *evalS-cons*[*simp*]: $evalS\ M\ \varphi\ (A\ \#\ \Gamma) = (evalF\ M\ \varphi\ A \vee evalS\ M\ \varphi\ \Gamma)$
⟨*proof*⟩

lemma *evalS-append*: $evalS\ M\ \varphi\ (\Gamma\ @\ \Delta) = (evalS\ M\ \varphi\ \Gamma \vee evalS\ M\ \varphi\ \Delta)$
⟨*proof*⟩

lemma *evalS-equiv*: $(equalOn\ (freeVarsFL\ \Gamma)\ f\ g) \Longrightarrow (evalS\ M\ f\ \Gamma = evalS\ M\ g\ \Gamma)$
⟨*proof*⟩

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

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

definition
validS :: *formula list* ⇒ *bool* **where**
$validS\ fs \equiv (\forall M.\ \forall \varphi \in modelAssigns\ M\ .\ evalS\ M\ \varphi\ fs = True)$

4.1 Rules

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

definition
concR :: *rule* ⇒ *sequent* **where**
$concR = (\lambda(conc,prems).\ conc)$

definition
premsR :: *rule* ⇒ *sequent set* **where**
$premsR = (\lambda(conc,prems).\ prems)$

definition
mapRule :: (*formula* ⇒ *formula*) ⇒ *rule* ⇒ *rule* **where**
$mapRule = (\lambda f\ (conc,prems)\ .\ (map\ f\ conc,(map\ f)\ `\ prems))$

lemma *mapRuleI*: $[\![A = map\ f\ a;\ B = map\ f\ `\ b]\!] \Longrightarrow (A,\ B) = mapRule\ f\ (a,\ b)$
⟨*proof*⟩

4.2 Deductions

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

inductive-set
deductions :: *rule set* ⇒ *formula list set*
for *rules* :: *rule set*

where
inferI: $[\![(conc,prems) \in rules;\ prems \in Pow(deductions(rules))$
$]\!] \Longrightarrow conc \in deductions(rules)$

lemma *mono-deductions*: $[\![A \subseteq B]\!] \Longrightarrow deductions(A) \subseteq deductions(B)$
⟨*proof*⟩

4.3 Basic Rule sets

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

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

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

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

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

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

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

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

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

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

lemma *AxiomI*: $[\![Axioms \subseteq A]\!] \Longrightarrow [FAtom\ Pos\ p\ vs,FAtom\ Neg\ p\ vs] \in deductions(A)$
⟨*proof*⟩

lemma *DAxiomsI*: $[\![DAxioms \subseteq A]\!] \Longrightarrow [FAtom\ Neg\ p\ vs,FAtom\ Pos\ p\ vs] \in deductions(A)$
⟨*proof*⟩

lemma *DisjI*: $[\![A0\#A1\#\Gamma \in deductions(A);\ Disjs \subseteq A]\!] \Longrightarrow (FConj\ Neg\ A0\ A1\#\Gamma) \in deductions(A)$
⟨*proof*⟩

lemma *ConjI*: $[\![(A0\#\Gamma) \in deductions(A);\ (A1\#\Delta) \in deductions(A);\ Conjs \subseteq A]\!] \Longrightarrow FConj\ Pos\ A0\ A1\#\Gamma\ @\ \Delta \in deductions(A)$
⟨*proof*⟩

lemma *AllI*: $[\![instanceF\ w\ A\#\Gamma \in deductions(R);\ w \notin freeVarsFL\ (FAll\ Pos\ A\#\Gamma);\ Alls \subseteq R]\!] \Longrightarrow (FAll\ Pos\ A\#\Gamma) \in deductions(R)$
⟨*proof*⟩

lemma *ExI*: $[\![instanceF\ w\ A\#\Gamma \in deductions(R);\ Exs \subseteq R]\!] \Longrightarrow (FAll\ Neg\ A\#\Gamma) \in deductions(R)$
⟨*proof*⟩

lemma *WeakI*: $[\![\Gamma \in deductions\ R;\ Weaks \subseteq R]\!] \Longrightarrow A\#\Gamma \in deductions(R)$
⟨*proof*⟩

lemma *ContrI*: $[\![A\#A\#\Gamma \in deductions\ R;\ Contrs \subseteq R]\!] \Longrightarrow A\#\Gamma \in deductions(R)$
⟨*proof*⟩

lemma *PermI*: $[\![Gamma' \in deductions\ R;\ mset\ \Gamma = mset\ Gamma';\ Perms \subseteq R]\!] \Longrightarrow \Gamma \in deductions(R)$
⟨*proof*⟩

4.4 Derived Rules

lemma *WeakI1*: $[\![\Gamma \in deductions(A);\ Weaks \subseteq A]\!] \Longrightarrow (\Delta\ @\ \Gamma) \in deductions(A)$
⟨*proof*⟩

lemma *WeakI2*: $[\![\Gamma \in deductions(A);\ Perms \subseteq A;\ Weaks \subseteq A]\!] \Longrightarrow (\Gamma\ @\ \Delta) \in deductions(A)$
⟨*proof*⟩

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

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

lemma *DisjI2*: $[\![(A0\#\Gamma) \in deductions(A);\ Disjs \subseteq A;\ Weaks \subseteq A;\ Perms \subseteq A]\!] \Longrightarrow FConj\ Neg\ A0\ A1\#\Gamma \in deductions(A)$
⟨*proof*⟩
lemma *perm-tmp4*: $Perms \subseteq R \Longrightarrow A\ @\ (a\ \#\ list)\ @\ (a\ \#\ list) \in deductions\ R \Longrightarrow (a\ \#\ a\ \#\ A)\ @\ list\ @\ list \in deductions\ R$
⟨*proof*⟩

lemma *weaken-append*:
$Contrs \subseteq R \Longrightarrow Perms \subseteq R \Longrightarrow A\ @\ \Gamma\ @\ \Gamma \in deductions(R) \Longrightarrow A\ @\ \Gamma \in deductions(R)$
⟨*proof*⟩

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

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

4.5 Standard Rule Sets For Predicate Calculus

definition
PC :: *rule set* **where**
PC = *Union* {*Perms*,*Axioms*,*Conjs*,*Disjs*,*Alls*,*Exs*,*Weaks*,*Contrs*,*Cuts*}

definition
CutFreePC :: *rule set* **where**
CutFreePC = *Union* {*Perms*,*Axioms*,*Conjs*,*Disjs*,*Alls*,*Exs*,*Weaks*,*Contrs*}

lemma *rulesInPCs*: *Axioms* ⊆ *PC Axioms* ⊆ *CutFreePC*
Conjs ⊆ *PC Conjs* ⊆ *CutFreePC*
Disjs ⊆ *PC Disjs* ⊆ *CutFreePC*
Alls ⊆ *PC Alls* ⊆ *CutFreePC*
Exs ⊆ *PC Exs* ⊆ *CutFreePC*
Weaks ⊆ *PC Weaks* ⊆ *CutFreePC*
Contrs ⊆ *PC Contrs* ⊆ *CutFreePC*
Perms ⊆ *PC Perms* ⊆ *CutFreePC*
Cuts ⊆ *PC*
CutFreePC ⊆ *PC*
⟨*proof*⟩

4.6 Monotonicity for CutFreePC deductions

definition
inDed :: *formula list* ⇒ *bool* **where**
inDed xs ≡ *xs* ∈ *deductions CutFreePC*

lemma *perm*: *mset xs* = *mset ys* ⟹ (*inDed xs* = *inDed ys*)
⟨*proof*⟩

lemma *contr*: *inDed* (*x*#*x*#*xs*) ⟹ *inDed* (*x*#*xs*)
⟨*proof*⟩

lemma *weak*: *inDed xs* ⟹ *inDed* (*x*#*xs*)
⟨*proof*⟩

lemma *inDed-mono′*[*simplified inDed-def*]: *set x* ⊆ *set y* ⟹ *inDed x* ⟹ *inDed y*
⟨*proof*⟩

lemma *inDed-mono*[*simplified inDed-def*]: *inDed x* ⟹ *set x* ⊆ *set y* ⟹ *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** γ :: $'a$
— 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*: $[\![(n,delta) \in tree\ subs\ \gamma;\ sigma \in subs\ delta]\!]$
$\Longrightarrow (Suc\ n,sigma) \in tree\ subs\ \gamma$

declare *tree.cases* [*elim*]
declare *tree.intros* [*intro*]

lemma *tree0Eq*: $(0,y) \in tree\ subs\ \gamma = (y = \gamma)$
⟨*proof*⟩

lemma *tree1Eq*:
$(Suc\ n,Y) \in tree\ subs\ \gamma \longleftrightarrow (\exists\, sigma \in subs\ \gamma\ .\ (n,Y) \in tree\ subs\ sigma)$
⟨*proof*⟩

definition
incLevel :: $nat * 'a \Rightarrow nat * 'a$ **where**
$incLevel = (\%\ (n,a).\ (Suc\ n,a))$

lemma *injIncLevel*: *inj incLevel*
⟨*proof*⟩

lemma *treeEquation*: $tree\ subs\ \gamma = insert\ (0,\gamma)\ (\bigcup delta{\in}subs\ \gamma.\ incLevel\ `\ tree\ subs\ delta)$
⟨*proof*⟩

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*: $terminal\ subs\ \Gamma \Longrightarrow x\ ^{\sim}{:}\ subs\ \Gamma$
⟨*proof*⟩

lemma *terminalI*: $x \in subs\ \Gamma \Longrightarrow \neg\ terminal\ subs\ \Gamma$
⟨*proof*⟩

4.9 Inherited

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

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

lemma *inheritedUn*[*rule-format*]: $inherited\ subs\ P \longrightarrow P\ A \longrightarrow P\ B \longrightarrow P\ (A\ Un\ B)$
⟨*proof*⟩

lemma *inheritedIncLevel*[*rule-format*]: $inherited\ subs\ P \longrightarrow P\ A \longrightarrow P\ (incLevel\ `\ A)$
⟨*proof*⟩

lemma *inheritedEmpty*[*rule-format*]: $inherited\ subs\ P \longrightarrow P\ \{\}$
⟨*proof*⟩

lemma *inheritedInsert*[*rule-format*]:
$inherited\ subs\ P \longrightarrow {}^{\sim}(terminal\ subs\ \Gamma) \longrightarrow P\ A \longrightarrow P\ (insert\ (n,\Gamma)\ A)$
⟨*proof*⟩

lemma *inheritedI*[*rule-format*]: $[\![\forall A\ B\ .\ (P\ A \wedge P\ B) = P\ (A\ Un\ B);$
$\forall A\ .\ P\ A = P\ (incLevel\ `\ A);$
$\forall n\ \Gamma\ A\ .\ {}^{\sim}(terminal\ subs\ \Gamma) \longrightarrow P\ A = P\ (insert\ (n,\Gamma)\ A);$
$P\ \{\}]\!] \Longrightarrow inherited\ subs\ P$
⟨*proof*⟩

lemma *inheritedUnEq*[*rule-format*, *symmetric*]: $inherited\ subs\ P \longrightarrow (P\ A \wedge P\ B) = P\ (A\ Un\ B)$
⟨*proof*⟩

lemma *inheritedIncLevelEq*[*rule-format*, *symmetric*]: $inherited\ subs\ P \longrightarrow P\ A = P\ (incLevel\ `\ A)$
⟨*proof*⟩

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

lemmas $inheritedUnD = iffD1[OF\ inheritedUnEq]$

lemmas $inheritedInsertD = inheritedInsertEq[THEN\ iffD1]$

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

lemma *inheritedUNEq*:
finite A $\Longrightarrow$ *inherited subs P* $\Longrightarrow$ ($\forall$ *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*(*tree subs delta*) = ($\forall$ *sigma*$\in$*subs delta*. *P*(*tree subs sigma*))
⟨*proof*⟩

lemma *inheritedPropagate*:
⟦$\neg$ *P* (*tree subs delta*); *inherited subs P*; *fans subs*; $\neg$ *terminal subs delta*⟧
$\Longrightarrow$ $\exists$ *sigma*$\in$*subs delta*. $\neg$ *P* (*tree subs sigma*)
⟨*proof*⟩

lemma *inheritedViaSub*:
⟦*inherited subs P*; *fans subs*; *P* (*tree subs delta*); *sigma* $\in$ *subs delta*⟧ $\Longrightarrow$ *P* (*tree subs sigma*)
⟨*proof*⟩

lemma *inheritedJoin*:
⟦*inherited subs P*; *inherited subs Q*⟧ $\Longrightarrow$ *inherited subs* (λ*x*. *P x* $\wedge$ *Q x*)
⟨*proof*⟩

lemma *inheritedJoinI*:
⟦*inherited subs P*; *inherited subs Q*; *R* = (λ*x*. *P x* $\wedge$ *Q x*)⟧
$\Longrightarrow$ *inherited subs R*
⟨*proof*⟩

4.10 bounded, boundedBy

definition
boundedBy :: *nat* $\Rightarrow$ (*nat* $*$ *'a*) *set* $\Rightarrow$ *bool* **where**
boundedBy N A $\equiv$ ($\forall$ (*n*,*delta*) $\in$ *A*. *n* < *N*)

definition
bounded :: (*nat* $*$ *'a*) *set* $\Rightarrow$ *bool* **where**
bounded A $\equiv$ ($\exists$ *N* . *boundedBy N A*)

lemma *boundedByEmpty*[*simp*]: *boundedBy N* {}
⟨*proof*⟩

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

lemma *boundedByUn*: *boundedBy N* (*A Un B*) = (*boundedBy N A* ∧ *boundedBy N B*)
⟨*proof*⟩

lemma *boundedByIncLevel′*: *boundedBy* (*Suc N*) (*incLevel* ‘ *A*) = *boundedBy N A*
⟨*proof*⟩

lemma *boundedByAdd1*: *boundedBy N B* ⟹ *boundedBy* (*N*+*M*) *B*
⟨*proof*⟩

lemma *boundedByAdd2*: *boundedBy M B* ⟹ *boundedBy* (*N*+*M*) *B*
⟨*proof*⟩

lemma *boundedByMono*: *boundedBy m B* ⟹ *m* < *M* ⟹ *boundedBy M B*
⟨*proof*⟩

lemmas *boundedByMonoD* = *boundedByMono*

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

lemma *boundedBySuc′*: *boundedBy N A* ⟹ *boundedBy* (*Suc N*) *A*
⟨*proof*⟩

lemma *boundedByIncLevel*: *boundedBy n* (*incLevel* ‘ (*tree subs* γ)) ⟷ (∃ *m* . *n* = *Suc m* ∧ *boundedBy m* (*tree subs* γ))
⟨*proof*⟩

lemma *boundedByUN*: *boundedBy N* (*UN x*:*A*. *B x*) = (!*x*:*A*. *boundedBy N* (*B x*))
⟨*proof*⟩

lemma *boundedBySuc*[*rule-format*]: *sigma* ∈ *subs* Γ ⟹ *boundedBy* (*Suc n*) (*tree subs* Γ) ⟹ *boundedBy n* (*tree subs sigma*)
⟨*proof*⟩

4.11 Inherited Properties- bounded

lemma *boundedEmpty*: *bounded* {}
⟨*proof*⟩

lemma *boundedUn*: *bounded* (*A Un B*) ⟷ (*bounded A* ∧ *bounded B*)
⟨*proof*⟩

lemma *boundedIncLevel*: *bounded* (*incLevel*‘ *A*) ⟷ (*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*: $founded\ subs\ P\ (tree\ subs\ \Gamma) \Longrightarrow sigma \in subs\ \Gamma \Longrightarrow founded\ subs\ P\ (tree\ subs\ sigma)$
⟨*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 γ ≡ (*SOME sigma*. (*sigma*:*subs* γ ∧ ~*P*(*tree subs sigma*)))

lemma *failingSubProps*:
⟦*inherited subs P*; ¬ *P* (*tree subs* γ); ¬ *terminal subs* γ; *fans subs*⟧
⟹ *failingSub subs P* γ ∈ *subs* γ ∧ ¬ *P* (*tree subs* (*failingSub subs P* γ))
⟨*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* γ ∈ *subs* γ
⟨*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*: $path\ subs\ \gamma\ P\ 0 = \gamma$
⟨*proof*⟩

lemma *pathSubs*:
$[\![inherited\ subs\ P;\ fans\ subs;\ \neg\ P\ (tree\ subs\ \gamma);\ \neg\ terminal\ subs\ (path\ subs\ \gamma\ P\ n)]\!]$
$\Longrightarrow path\ subs\ \gamma\ P\ (Suc\ n) \in subs\ (path\ subs\ \gamma\ P\ n)$
⟨*proof*⟩

lemma *pathStops*: $terminal\ subs\ (path\ subs\ \gamma\ P\ n) \Longrightarrow path\ subs\ \gamma\ P\ (Suc\ n) = path\ subs\ \gamma\ P\ n$
⟨*proof*⟩

4.16 Branch

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

lemma *branch0*: $branch\ subs\ \Gamma\ f \Longrightarrow f\ 0 = \Gamma$
⟨*proof*⟩

lemma *branchStops*: $branch\ subs\ \Gamma\ f \Longrightarrow terminal\ subs\ (f\ n) \Longrightarrow f\ (Suc\ n) = f\ n$
⟨*proof*⟩

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

lemma *branchI*: $[\![f\ 0 = \Gamma;$
$\forall n\ .\ terminal\ subs\ (f\ n) \longrightarrow f\ (Suc\ n) = f\ n;$
$\forall n\ .\ \neg\ terminal\ subs\ (f\ n) \longrightarrow f\ (Suc\ n) \in subs\ (f\ n)]\!] \Longrightarrow branch\ subs\ \Gamma\ f$
⟨*proof*⟩

lemma *branchTerminalPropagates*: $branch\ subs\ \Gamma\ f \Longrightarrow terminal\ subs\ (f\ m) \Longrightarrow terminal\ subs\ (f\ (m + n))$
⟨*proof*⟩

lemma *branchTerminalMono*:
$branch\ subs\ \Gamma\ f \Longrightarrow m < n \Longrightarrow terminal\ subs\ (f\ m) \Longrightarrow terminal\ subs\ (f\ n)$
⟨*proof*⟩

lemma *branchPath*:
$[\![inherited\ subs\ P;\ fans\ subs;\ \neg\ P(tree\ subs\ \gamma)]\!]$
$\Longrightarrow branch\ subs\ \gamma\ (path\ subs\ \gamma\ P)$
⟨*proof*⟩

4.17 failing branch property: abstracts path defn

lemma *failingBranchExistence*:
  $[\![$*inherited subs P*; *fans subs*; $\sim P(tree\ subs\ \gamma)]\!]$
  $\Longrightarrow \exists f\ .\ branch\ subs\ \gamma\ f \wedge (\forall n\ .\ \neg\ P(tree\ subs\ (f\ n)))$
  ⟨*proof*⟩

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

lemma *infBranchI*: $[\![f\ 0 = \Gamma; \forall n\ .\ f\ (Suc\ n) \in subs\ (f\ n)]\!] \Longrightarrow infBranch\ subs\ \Gamma\ f$
  ⟨*proof*⟩

4.18 Tree induction principles

lemma *boundedTreeInduction'*:
 $[\![$ *fans subs*;
   $\forall delta.\ \neg\ terminal\ subs\ delta \longrightarrow (\forall sigma \in subs\ delta.\ P\ sigma) \longrightarrow P\ delta$
$]\!]$
  $\Longrightarrow \forall \Gamma.\ boundedBy\ m\ (tree\ subs\ \Gamma) \longrightarrow founded\ subs\ P\ (tree\ subs\ \Gamma) \longrightarrow P\ \Gamma$
⟨*proof*⟩

lemma *boundedTreeInduction*:
  $[\![$*fans subs*;
    $bounded\ (tree\ subs\ \Gamma); founded\ subs\ P\ (tree\ subs\ \Gamma);$
  $\bigwedge delta.\ [\![\neg\ terminal\ subs\ delta; (\forall sigma \in subs\ delta.\ P\ sigma)]\!] \Longrightarrow P\ delta$
  $]\!] \Longrightarrow P\ \Gamma$
  ⟨*proof*⟩

lemma *boundedTreeInduction2*:
 $[\![$*fans subs*;
   $\forall delta.\ (\forall sigma \in subs\ delta.\ P\ sigma) \longrightarrow P\ delta]\!]$
  $\Longrightarrow boundedBy\ m\ (tree\ subs\ \Gamma) \longrightarrow P\ \Gamma$
⟨*proof*⟩

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* ⇒ *formula list* **where**
sequent = (λ(*atoms*,*nforms*) . *map snd nforms* @ *map* (λ(*z*,*p*,*vs*) . *FAtom z p vs*) *atoms*)

definition
pseq :: *formula list* ⇒ *pseq* **where**
pseq fs = ([],*map* (λ*f*.(*0*,*f*)) *fs*)

definition *atoms* :: *pseq* ⇒ *atom list* **where** *atoms* = *fst*
definition *nforms* :: *pseq* ⇒ *nform list* **where** *nforms* = *snd*

5.2 subs: SATAxiom

definition
SATAxiom :: *formula list* ⇒ *bool* **where**
SATAxiom fs ≡ (∃ *n vs* . *FAtom Pos n vs* ∈ *set fs* ∧ *FAtom Neg n vs* ∈ *set fs*)

5.3 subs: a CutFreePC justifiable backwards proof step

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

definition
subsFConj :: [*atom list*,(*nat* ∗ *formula*) *list*,*signs*,*formula*,*formula*] ⇒ *pseq set* **where**

```
subsFConj atms nAs z A0 A1 =
  (case z of
    Pos ⇒ { (atms,(0,A0)#nAs),(atms,(0,A1)#nAs) }
  | Neg ⇒ { (atms,(0,A0)#(0,A1)#nAs) })
```

definition
subsFAll :: [*atom list*,(*nat* ∗ *formula*) *list*,*nat*,*signs*,*formula*,*vbl set*] ⇒ *pseq set* **where**

```
subsFAll atms nAs n z A frees =
  (case z of
    Pos ⇒ { let v = freshVar frees in (atms,(0,instanceF v A)#nAs) }
  | Neg ⇒ { (atms,(0,instanceF (X n) A)#nAs @ [(Suc n,FAll Neg A)]) })
```

definition
subs :: *pseq* ⇒ *pseq set* **where**

```
subs = (λpseq .
          if SATAxiom (sequent pseq) then
             {}
          else let (atms,nforms) = pseq
               in  case nforms of
                      []      ⇒ {}
                    | nA#nAs ⇒ let (n,A) = nA
```

in (*case A of*
FAtom z P vs ⇒ *subsFAtom atms nAs z P vs*
| *FConj z A0 A1* ⇒ *subsFConj atms nAs z A0 A1*
| *FAll z A* ⇒ *subsFAll atms nAs n z A*
(*freeVarsFL* (*sequent pseq*))))

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

definition
proofTree :: (*nat* ∗ *pseq*) *set* ⇒ *bool* **where**
proofTree A ⟷ *bounded A* ∧ *founded subs* (*SATAxiom* ∘ *sequent*) *A*

5.5 path: considers, contains, costBarrier

definition
considers :: [*nat* ⇒ *pseq*,*nat* ∗ *formula*,*nat*] ⇒ *bool* **where**
considers f nA n = (*case* (*snd* (*f n*)) *of* [] ⇒ *False* | *x#xs* ⇒ *x*=*nA*)

definition
contains :: [*nat* ⇒ *pseq*,*nat* ∗ *formula*,*nat*] ⇒ *bool* **where**
contains f nA n ⟷ *nA* ∈ *set* (*snd* (*f n*))

abbreviation (*input*) *power3* ≡ *power* (*3*::*nat*)

definition
costBarrier :: [*nat* ∗ *formula*,*pseq*] ⇒ *nat* **where**

costBarrier nA = (λ(*atms*,*nAs*).
let barrier = *takeWhile* (λ*x*. *nA* ≠ *x*) *nAs*
in let costs = *map* (*power3* ∘ *size* ∘ *snd*) *barrier*
in sumList costs)

5.6 path: eventually

definition
EV :: [*nat* ⇒ *bool*] ⇒ *bool* **where**
EV f ≡ (∃ *n* . *f n*)

5.7 path: counter model

definition
counterM :: (*nat* ⇒ *pseq*) ⇒ *object set* **where**
counterM f ≡ *range obj*

definition
counterEvalP :: (*nat* ⇒ *pseq*) ⇒ *predicate* ⇒ *object list* ⇒ *bool* **where**
counterEvalP f = (λ*p args* . ! *i* . ¬ (*EV* (*contains f* (*i*,*FAtom Pos p* (*map* (*X* ∘ *inv obj*) *args*)))))

definition

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

primrec *counterAssign* :: *vbl* ⇒ *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*:
¬ *SATAxiom* (*sequent* γ) ⟹
subs γ =
(*case nforms* γ *of*
[] ⇒ {}
| *nA#nAs* ⇒ *let* (*n*,*A*) = *nA*
in (*case A of*
FAtom z P vs ⇒ *subsFAtom* (*atoms* γ) *nAs z P vs*
| *FConj z A0 A1* ⇒ *subsFConj* (*atoms* γ) *nAs z A0 A1*
| *FAll z A* ⇒ *subsFAll* (*atoms* γ) *nAs n z A* (*freeVarsFL*
(*sequent* γ))))
⟨*proof*⟩

5.9 inherited: proofTree

lemma *proofTree-def2*: *proofTree* = (λ*x*. *bounded x* ∧ *founded subs* (*SATAxiom* ∘ *sequent*) *x*)
⟨*proof*⟩

lemma *inheritedProofTree*: *inherited subs proofTree*
⟨*proof*⟩

lemma *proofTreeI*: ⟦*bounded A*; *founded subs* (*SATAxiom* ∘ *sequent*) *A*⟧ ⟹ *proofTree A*
⟨*proof*⟩

lemma *proofTreeBounded*: *proofTree A* ⟹ *bounded A*
⟨*proof*⟩

lemma *proofTreeTerminal*:
⟦*proofTree A*; (*n*, *delta*) ∈ *A*; *terminal subs delta*⟧ ⟹ *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* γ) $\Longrightarrow$ *terminal subs* γ
⟨*proof*⟩

lemma *SATAxiomBounded*:*SATAxiom* (*sequent* γ) $\Longrightarrow$ *bounded* (*tree subs* γ)
⟨*proof*⟩

lemma *SATAxiomFounded*: *SATAxiom* (*sequent* γ) $\Longrightarrow$ *founded subs* (*SATAxiom* $\circ$ *sequent*) (*tree subs* γ)
⟨*proof*⟩

lemma *SATAxiomProofTree*: *SATAxiom* (*sequent* γ) $\Longrightarrow$ *proofTree* (*tree subs* γ)
⟨*proof*⟩

lemma *SATAxiomEq*: (*proofTree* (*tree subs* γ) $\wedge$ *terminal subs* γ) = *SATAxiom* (*sequent* γ)
⟨*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 ⟦¬ *SATAxiom* (*sequent* (*ats*, (*n*, *f*) # *list*));
¬ *terminal subs* (*ats*, (*n*, *f*) # *list*);
∀ *sigma*∈*subs* (*ats*, (*n*, *f*) # *list*). *sequent sigma* ∈ *deductions CutFreePC*⟧
$\Longrightarrow$ *sequent* (*ats*, (*n*, *f*) # *list*) ∈ *deductions CutFreePC*
⟨*proof*⟩

lemma *subsJustified*:
assumes ¬ *terminal subs* γ
and ∀ *sigma* ∈ *subs* γ. *sequent sigma* ∈ *deductions* (*CutFreePC*)

shows *sequent* $\gamma \in$ *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* $\gamma \in$ *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*) = [] $\Longrightarrow$ $\neg$ *contains f iA n*
⟨*proof*⟩

lemma *nformsTerminal*: *nforms* (*f n*) = [] $\Longrightarrow$ *terminal subs* (*f n*)
⟨*proof*⟩

lemma *nformsStops*:
⟦*branch subs* γ *f*; $\bigwedge$*n*. $\neg$ *proofTree* (*tree subs* (*f n*)); *nforms* (*f n*) = []⟧
$\Longrightarrow$ *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 γ f* ¬ *proofTree* (*tree subs* (*f n*)) *contains f iA n*
shows ¬ *terminal subs* (*f n*)
⟨*proof*⟩

lemma *containsPropagates*:
assumes *branch subs γ f*
and ⋀*n*. ¬ *proofTree* (*tree subs* (*f n*))
and *contains f iA n*
shows *contains f iA* (*Suc n*) ∨ *considers f iA n*
⟨*proof*⟩

5.19 termination: (for EV contains implies EV considers)

lemma *terminationRule* [*rule-format*]:
! *n*. *P n* ⟶ (¬ (*P* (*Suc n*)) | (*P* (*Suc n*) ∧ *msrFn* (*Suc n*) < (*msrFn*::*nat* ⇒ *nat*) *n*)) ⟹ *P m* ⟶ (∃ *n* . *P n* ∧ ¬ (*P* (*Suc n*)))
(**is** - ⟹ *?P m*)
⟨*proof*⟩

5.20 costBarrier: lemmas

5.21 costBarrier: exp3 lemmas - bit specific...

lemma *exp1*: *power3 A* + *power3 B* < *3* ∗ (*power3 A* ∗ *power3 B*)
⟨*proof*⟩

lemma *exp1′*: *power3 A* < *3* ∗ ((*power3 A*) ∗ (*power3 B*)) + *C*
⟨*proof*⟩

lemma *exp2*: *Suc 0* < *3* ∗ *power3* (*B*)
⟨*proof*⟩

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 ⟦¬ *SATAxiom* (*sequent* (*a*, (*num*, *fm*) # *list*)); *iA* ≠ (*num*, *fm*);
¬ *proofTree* (*tree subs* (*a*, (*num*, *fm*) # *list*));
fSucn ∈ *subs* (*a*, (*num*, *fm*) # *list*); *iA* ∈ *set list*⟧
⟹ *costBarrier iA fSucn* < *costBarrier iA* (*a*, (*num*, *fm*) # *list*)
⟨*proof*⟩

lemma *costBarrierDecreases*:
assumes *branch subs γ f*
and ⋀*n*. ¬ *proofTree* (*tree subs* (*f n*))
and *contains f iA n*
and ¬ *considers f iA n*
shows *costBarrier iA* (*f* (*Suc n*)) < *costBarrier iA* (*f n*)

⟨*proof*⟩

5.23 path: EV contains implies EV considers

lemma *considersContains*: *considers f iA n* $\Longrightarrow$ *contains f iA n*
⟨*proof*⟩

lemma *containsConsiders*:
assumes *branch subs* γ *f*
and $\bigwedge$*n*. ¬ *proofTree* (*tree subs* (*f n*))
and *EV* (*contains f iA*)
shows *EV* (*considers f iA*)
⟨*proof*⟩

5.24 EV contains: common lemma

lemma *lemmA*:
assumes *branch subs* γ *f*
and $\bigwedge$*n*. ¬ *proofTree* (*tree subs* (*f n*))
and *EV* (*contains f* (*i*, *A*))
obtains *n nAs* **where** ¬ *SATAxiom* (*sequent* (*f n*))
nforms (*f n*) = (*i*,*A*) # *nAs f* (*Suc n*) ∈ *subs* (*f n*)
⟨*proof*⟩

5.25 EV contains: FConj,FDisj,FAll

lemma *evContainsConj*:
assumes *EV* (*contains f* (*i*, *FConj Pos A0 A1*))
and *branch subs* γ *f*
and $\bigwedge$*n*. ¬ *proofTree* (*tree subs* (*f n*))
shows *EV* (*contains f* (*0*, *A0*)) ∨ *EV* (*contains f* (*0*, *A1*))
⟨*proof*⟩

lemma *evContainsDisj*:
assumes *EV* (*contains f* (*i*, *FConj Neg A0 A1*))
and *branch subs* γ *f*
and $\bigwedge$*n*. ¬ *proofTree* (*tree subs* (*f n*))
shows *EV* (*contains f* (*0*, *A0*)) ∧ *EV* (*contains f* (*0*, *A1*))
⟨*proof*⟩

lemma *evContainsAll*:
assumes *EV* (*contains f* (*i*,*FAll Pos body*))
and *branch subs* γ *f*
and $\bigwedge$*n*. ¬ *proofTree* (*tree subs* (*f n*))
shows ∃ *v* . *EV* (*contains f* (*0*,*instanceF v body*))
⟨*proof*⟩

lemma *evContainsEx-instance*:
assumes *EV* (*contains f* (*i*,*FAll Neg body*))
and *branch subs* γ *f*

and $\bigwedge n.\ \neg\ proofTree\ (tree\ subs\ (f\ n))$
shows $EV\ (contains\ f\ (0,instanceF\ (X\ i)\ body))$
⟨*proof*⟩

lemma *evContainsEx-repeat*:
assumes $EV\ (contains\ f\ (i,FAll\ Neg\ body))$
and $branch\ subs\ \gamma\ f$
and $\bigwedge n.\ \neg\ proofTree\ (tree\ subs\ (f\ n))$
shows $EV\ (contains\ f\ (Suc\ i,FAll\ Neg\ body))$
⟨*proof*⟩

5.26 EV contains: lemmas (temporal related)

5.27 EV contains: FAtoms

lemma *notTerminalNotSATAxiom*: $\neg\ terminal\ subs\ \gamma \Longrightarrow \neg\ SATAxiom\ (sequent\ \gamma)$
⟨*proof*⟩

lemma *notTerminalNforms*: $\neg\ terminal\ subs\ (f\ n) \Longrightarrow nforms\ (f\ n) \neq []$
⟨*proof*⟩

lemma *atomsPropagate*:
assumes f: $branch\ subs\ \gamma\ f$ **and** x: $x \in set\ (atoms\ (f\ n))$
shows $x \in set\ (atoms\ (f\ (Suc\ n)))$
⟨*proof*⟩

5.28 EV contains: FEx cases

lemma *evContainsEx0-allRepeats*:
$[\![branch\ subs\ \gamma\ f;\ \bigwedge n\ .\ \neg\ proofTree\ (tree\ subs\ (f\ n));$
$EV\ (contains\ f\ (0,FAll\ Neg\ A))]\!]$
$\Longrightarrow EV\ (contains\ f\ (i,FAll\ Neg\ A))$
⟨*proof*⟩

lemma *evContainsEx0-allInstances*:
$[\![branch\ subs\ \gamma\ f;\ \bigwedge n.\ \neg\ proofTree\ (tree\ subs\ (f\ n));$
$EV\ (contains\ f\ (0,FAll\ Neg\ A))]\!]$
$\Longrightarrow EV\ (contains\ f\ (0,instanceF\ (X\ i)\ A))$
⟨*proof*⟩

5.29 pseq: creates initial pseq

lemma *containsPSeq0D*: $branch\ subs\ (pseq\ fs)\ f \Longrightarrow contains\ f\ (i,A)\ 0 \Longrightarrow i=0$
⟨*proof*⟩

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** $P\ n\ \neg\ P\ (Suc\ n)$
⟨*proof*⟩

lemma *containsNotTerminal′*:
$[\![$ *branch subs* γ *f*; $\bigwedge n.\ \neg$ *proofTree* (*tree subs* (*f n*)); *contains f iA n* $]\!] \Longrightarrow \neg$ (*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*))$]\!]$
$\Longrightarrow$ *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*))$]\!]$
$\Longrightarrow$ *EV* (*contains f* (*0*,*instanceF v body*))
⟨*proof*⟩

5.32 EV contains: atoms

lemma *atomsInSequentI*:
(*z*,*P*,*vs*) $\in$ *set* (*fst ps*) $\Longrightarrow$ *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 $\exists\, n.\ \forall\, m.\ \mathit{FAtom}\ z\ P\ \mathit{vs} \in \mathit{set}\ (\mathit{sequent}\ (f\ (n + m)))$
⟨*proof*⟩

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

5.33 counterModel: lemmas

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

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

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

lemma *map-X-map-counterAssign* [*simp*]: *map X* (*map* (*inv obj*) (*map counterAssign xs*)) = *xs*
⟨*proof*⟩

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

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

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

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

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

5.35 adequacy

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

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

lemma *validProofTree*:
  assumes ¬ *proofTree* (*tree subs* (*pseq fs*))
  shows ¬ *validS fs*
⟨*proof*⟩

theorem *adequacy*[*simplified sequent-pseq*]: *validS fs* ⟹ (*sequent* (*pseq fs*)) ∈ *deductions CutFreePC*
  ⟨*proof*⟩

end

6 Soundness

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

lemma *permutation-validS*: *mset fs* = *mset gs* ⟹ (*validS fs* = *validS gs*)
  ⟨*proof*⟩

lemma *modelAssigns-vblcase*: φ ∈ *modelAssigns M* ⟹ *x* ∈ *objects M* ⟹ *vblcase x* φ ∈ *modelAssigns M*
  ⟨*proof*⟩

lemma *soundnessFAll*:
    assumes *u* ∉ *freeVarsFL* (*FAll Pos A* # *Gamma*)
    and *validS* (*instanceF u A* # *Gamma*)
  shows *validS* (*FAll Pos A* # *Gamma*)
⟨*proof*⟩

lemma *soundnessFEx*: *validS* (*instanceF x A* # *Gamma*) ⟹ *validS* (*FAll Neg A* # *Gamma*)
  ⟨*proof*⟩

lemma *soundnessFCut*: ⟦*validS* (*C* # *Gamma*); *validS* (*FNot C* # *Delta*)⟧ ⟹ *validS* (*Gamma* @ *Delta*)
  ⟨*proof*⟩

lemma *soundness*: *fs* : *deductions*(*PC*) ⟹ *validS fs*
⟨*proof*⟩

theorem *completeness*: *fs* ∈ *deductions* (*PC*) ⟷ *validS fs*
  ⟨*proof*⟩

end