

# Completeness for FOL

James Margetson, ported by Tom Ridge

December 12, 2009

## Contents

<b>1</b>	<b>Permutation Lemmas</b>	<b>3</b>
1.1	perm, count equivalence . . . . .	3
1.2	Properties closed under Perm and Contr hold for x iff hold for remdups x . . . . .	4
1.3	List properties closed under Perm, Weak and Contr are mono- tonic in the set of the list . . . . .	4
1.4	Following used in Soundness . . . . .	5
<b>2</b>	<b>Base</b>	<b>5</b>
2.1	Integrate with Isabelle libraries? . . . . .	5
2.2	Summation . . . . .	5
2.3	Termination Measure . . . . .	6
2.4	Functions . . . . .	6
<b>3</b>	<b>Formula</b>	<b>7</b>
3.1	Variables . . . . .	7
3.2	Predicates . . . . .	9
3.3	Formulas . . . . .	9
3.4	formula signs induct, formula signs cases . . . . .	9
3.5	Frees . . . . .	11
3.6	Substitutions . . . . .	12
3.7	Models . . . . .	12
3.8	model, non empty set and positive atom valuation . . . . .	12
3.9	Validity . . . . .	13
<b>4</b>	<b>Sequents</b>	<b>14</b>
4.1	Rules . . . . .	15
4.2	Deductions . . . . .	15
4.3	Basic Rule sets . . . . .	16
4.4	Derived Rules . . . . .	17
4.5	Standard Rule Sets For Predicate Calculus . . . . .	18
4.6	Monotonicity for CutFreePC deductions . . . . .	18

4.7	Tree	19
4.8	Terminal	20
4.9	Inherited	20
4.10	bounded, boundedBy	22
4.11	Inherited Properties- bounded	23
4.12	founded	23
4.13	Inherited Properties- founded	24
4.14	Inherited Properties- finite	24
4.15	path: follows a failing inherited property through tree	24
4.16	Branch	25
4.17	failing branch property: abstracts path defn	26
4.18	Tree induction principles	26
<b>5</b>	<b>Completeness</b>	<b>27</b>
5.1	pseq: type represents a processed sequent	27
5.2	subs: SATAxiom	28
5.3	subs: a CutFreePC justifiable backwards proof step	28
5.4	proofTree(Gamma) says whether tree(Gamma) is a proof	28
5.5	path: considers, contains, costBarrier	29
5.6	path: eventually	29
5.7	path: counter model	29
5.8	subs: finite	29
5.9	inherited: proofTree	30
5.10	pseq: lemma	30
5.11	SATAxiom: proofTree	31
5.12	SATAxioms are deductions: - needed	31
5.13	proofTrees are deductions: subs respects rules - messy start and end	31
5.14	proofTrees are deductions: instance of boundedTreeInduction	31
5.15	contains, considers:	32
5.16	path: nforms = [] implies	32
5.17	path: cases	32
5.18	path: contains not terminal and propagate condition	32
5.19	path: no consider lemmas	33
5.20	path: contains initially	33
5.21	termination: (for EV contains implies EV considers)	33
5.22	costBarrier: lemmas	33
5.23	costBarrier: exp3 lemmas - bit specific...	33
5.24	costBarrier: decreases whilst contains and unconsiders	34
5.25	path: EV contains implies EV considers	34
5.26	EV contains: common lemma	34
5.27	EV contains: FConj,FDisj,FAll	34
5.28	EV contains: lemmas (temporal related)	35
5.29	EV contains: FAToms	35

5.30	EV contains: FEx cases . . . . .	36
5.31	pseq: creates initial pseq . . . . .	36
5.32	EV contains: contain any (i,FEx y) means contain all (i,FEx y) . . . . .	36
5.33	EV contains: contain any (i,FEx y) means contain all (i,FEx y) . . . . .	36
5.34	EV contains: atoms . . . . .	37
5.35	counterModel: lemmas . . . . .	37
5.36	counterModel: all path formula value false - step by step . . .	38
5.37	adequacy . . . . .	38

## 6 Soundness 39

### 1 Permutation Lemmas

```
theory PermutationLemmas
imports Permutation Multiset
begin
```

— following function is very close to that in multisets- now we can make the connection that  $x \dot{\sim} y$  iff the multiset of  $x$  is the same as that of  $y$

#### 1.1 perm, count equivalence

```
primrec count :: 'a ⇒ 'a list ⇒ nat
```

```
where
```

```
  count x [] = 0
```

```
| count x (y#ys) = (if x=y then 1 else 0) + count x ys
```

```
lemma perm-count: A <~> B ⇒ (∀ x. count x A = count x B)
<proof>
```

```
lemma count-0: (∀ x. count x B = 0) = (B = [])
<proof>
```

```
lemma count-Suc: count a B = Suc m ⇒ a : set B
<proof>
```

```
lemma count-append: count a (xs@ys) = count a xs + count a ys
<proof>
```

```
lemma count-perm: !! B. (∀ x. count x A = count x B) ⇒ A <~> B
<proof>
```

```
lemma perm-count-conv: A <~> B = (∀ x. count x A = count x B)
<proof>
```

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

**lemma** *remdups-append*:  $y : set\ ys \dashrightarrow remdups\ (ws@y\#ys) = remdups\ (ws@ys)$   
 ⟨proof⟩

**lemma** *perm-contr'*: **assumes** *perm*[rule-format]:  $! xs\ ys.\ xs <\sim\sim> ys \dashrightarrow (P\ xs = P\ ys)$

**and** *contr'*[rule-format]:  $! x\ xs.\ P(x\#x\#xs) = P\ (x\#xs)$   
**shows**  $! xs.\ length\ xs = n \dashrightarrow (P\ xs = P\ (remdups\ xs))$   
 ⟨proof⟩

**lemma** *perm-contr*: **assumes** *perm*:  $! xs\ ys.\ xs <\sim\sim> ys \dashrightarrow (P\ xs = P\ ys)$

**and** *contr'*:  $! x\ xs.\ P(x\#x\#xs) = P\ (x\#xs)$   
**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* ::  $'a \Rightarrow 'a\ list \Rightarrow 'a\ list$  **where**  
*rem*  $x\ xs = filter\ (\%y.\ y \sim = x)\ xs$

**lemma** *rem*:  $x \sim : set\ (rem\ x\ xs)$   
 ⟨proof⟩

**lemma** *length-rem*:  $length\ (rem\ x\ xs) \leq length\ xs$   
 ⟨proof⟩

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

**lemma** *perm-weak-filter'*: **assumes** *perm*[rule-format]:  $! xs\ ys.\ xs <\sim\sim> ys \dashrightarrow (P\ xs = P\ ys)$

**and** *weak*[rule-format]:  $! x\ xs.\ P\ xs \dashrightarrow P\ (x\#xs)$   
**shows**  $! ys.\ P\ (ys@filter\ Q\ xs) \dashrightarrow P\ (ys@xs)$   
 ⟨proof⟩

**lemma** *perm-weak-filter*: **assumes** *perm*:  $! xs\ ys.\ xs <\sim\sim> ys \dashrightarrow (P\ xs = P\ ys)$

**and** *weak*:  $! x\ xs.\ P\ xs \dashrightarrow P\ (x\#xs)$   
**shows**  $P\ (filter\ Q\ xs) \implies P\ xs$   
 ⟨proof⟩

**lemma** *perm-weak-contr-mono*:

**assumes** *perm*:  $! xs\ ys.\ xs <\sim\sim> ys \dashrightarrow (P\ xs = P\ ys)$   
**and** *contr*:  $! x\ xs.\ P\ (x\#x\#xs) \dashrightarrow P\ (x\#xs)$   
**and** *weak*:  $! x\ xs.\ P\ xs \dashrightarrow P\ (x\#xs)$

**and**  $xy: \text{set } x \leq \text{set } y$   
**and**  $Px : P x$   
**shows**  $P y$   
 $\langle \text{proof} \rangle$

## 1.4 Following used in Soundness

**primrec** *multiset-of-list* ::  $'a \text{ list} \Rightarrow 'a \text{ multiset}$   
**where**

$\text{multiset-of-list } [] = \{\#\}$   
 $|\ \text{multiset-of-list } (x\#xs) = \{\#x\# \} + \text{multiset-of-list } xs$

**lemma** *count-count[symmetric]*:  $\text{count } x A = \text{Multiset.count } (\text{multiset-of-list } A) x$   
 $\langle \text{proof} \rangle$

**lemma** *perm-multiset*:  $A <\sim\sim> B = (\text{multiset-of-list } A = \text{multiset-of-list } B)$   
 $\langle \text{proof} \rangle$

**lemma** *set-of-multiset-of-list*:  $\text{set-of } (\text{multiset-of-list } A) = \text{set } A$   
 $\langle \text{proof} \rangle$

**end**

## 2 Base

**theory** *Base*  
**imports** *PermutationLemmas*  
**begin**

### 2.1 Integrate with Isabelle libraries?

— Misc

— FIXME added by tjr, forms basis of a lot of proofs of existence of inf sets  
 — something like this should be in FiniteSet, asserting nats are not finite

**lemma** *natset-finite-max*: **assumes**  $a: \text{finite } A$

**shows**  $\text{Suc } (\text{Max } A) \notin A$

$\langle \text{proof} \rangle$

**lemma** *not-finite-univ*:  $\sim \text{finite } (\text{UNIV}::\text{nat set})$

$\langle \text{proof} \rangle$

**lemma** *LeastI-ex*:  $(\exists x. P (x::'a::\text{wellorder})) \implies P (\text{LEAST } x. P x)$

$\langle \text{proof} \rangle$

### 2.2 Summation

**primrec** *summation* ::  $(\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{nat} \Rightarrow \text{nat}$

**where**

$\text{summation } f 0 = f 0$

|  $\text{summation } f \text{ (Suc } n) = f \text{ (Suc } n) + \text{summation } f \text{ } n$

## 2.3 Termination Measure

**primrec**  $\text{exp} :: [\text{nat}, \text{nat}] \Rightarrow \text{nat}$

**where**

$\text{exp } x \ 0 = 1$

|  $\text{exp } x \text{ (Suc } m) = x * \text{exp } x \ m$

**primrec**  $\text{sumList} :: \text{nat list} \Rightarrow \text{nat}$

**where**

$\text{sumList } [] = 0$

|  $\text{sumList } (x \# xs) = x + \text{sumList } xs$

## 2.4 Functions

**definition**

$\text{preImage} :: ('a \Rightarrow 'b) \Rightarrow 'b \text{ set} \Rightarrow 'a \text{ set}$  **where**

$\text{preImage } f \ A = \{ x . f \ x \in A \}$

**definition**

$\text{pre} :: ('a \Rightarrow 'b) \Rightarrow 'b \Rightarrow 'a \text{ set}$  **where**

$\text{pre } f \ a = \{ x . f \ x = a \}$

**definition**

$\text{equalOn} :: ['a \text{ set}, 'a \Rightarrow 'b, 'a \Rightarrow 'b] \Rightarrow \text{bool}$  **where**

$\text{equalOn } A \ f \ g = (!x:A. f \ x = g \ x)$

**lemma**  $\text{preImage-insert}$ :  $\text{preImage } f \ (\text{insert } a \ A) = \text{pre } f \ a \ \text{Un } \text{preImage } f \ A$

$\langle \text{proof} \rangle$

**lemma**  $\text{preImageI}$ :  $f \ x : A \ ==> x : \text{preImage } f \ A$

$\langle \text{proof} \rangle$

**lemma**  $\text{preImageE}$ :  $x : \text{preImage } f \ A \ ==> f \ x : A$

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

**lemma**  $\text{equalOn-UnD}$ :  $\text{equalOn } (A \ \text{Un } B) \ f \ g \ ==> \text{equalOn } A \ f \ g \ \& \ \text{equalOn } B \ f \ g$

$\langle \text{proof} \rangle$

$\langle \text{proof} \rangle$

**lemma**  $\text{inj-inv-singleton[simp]}$ :  $[\text{inj } f; f \ z = y] \ ==> \{x. f \ x = y\} = \{z\}$

*<proof>*

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

**lemma** *finite-preImage[simp]*:  $\llbracket \text{finite } A; \text{inj } f \rrbracket \implies \text{finite } (\text{preImage } f \ A)$   
*<proof>*

**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 => nat* **where** *deX (X n) = n*

**lemma** *X-deX[simp]*:  $X \ (\text{deX } a) = a$   
*<proof>*

**definition** *zeroX = X 0*

**primrec**

*nextX* :: *vbl => vbl* **where**  
*nextX (X n) = X (Suc n)*

**definition**

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

**translations**

*case p of XCONST zeroX  $\Rightarrow$  a | XCONST nextX y  $\Rightarrow$  b == (CONST vblcase a (%y. b) p)*

**definition**

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

**lemma** *nextX-nextX*[*iff*]:  $\text{nextX } x = \text{nextX } y = (x = y)$   
(*proof*)

**lemma** *inj-nextX*:  $\text{inj } \text{nextX}$   
(*proof*)

**lemma** *ind'*:  $P \text{ zeroX} \implies (! v . P v \longrightarrow P (\text{nextX } v)) \implies P v'$   
(*proof*)

**lemma** *ind*:  $\llbracket P \text{ zeroX}; \bigwedge v . P v \implies P (\text{nextX } v) \rrbracket \implies P v'$   
(*proof*)

**lemma** *zeroX-nextX*[*iff*]:  $\text{zeroX} \sim = \text{nextX } a$  — FIXME iff?  
(*proof*)

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

**lemma** *nextX*:  $\text{nextX } (X n) = X (\text{Suc } n)$   
(*proof*)

**lemma** *vblcase-zeroX*[*simp*]:  $\text{vblcase } a b \text{ zeroX} = a$   
(*proof*)

**lemma** *vblcase-nextX*[*simp*]:  $\text{vblcase } a b (\text{nextX } n) = b n$   
(*proof*)

**lemma** *vbl-cases*:  $x = \text{zeroX} \mid (? y . x = \text{nextX } y)$   
(*proof*)

**lemma** *vbl-casesE*:  $\llbracket x = \text{zeroX} \implies P; \bigwedge y . x = \text{nextX } y \implies P \rrbracket \implies P$   
(*proof*)

**lemma** *comp-vblcase*:  $f \circ \text{vblcase } a b = \text{vblcase } (f a) (f \circ b)$   
(*proof*)

**lemma** *equalOn-vblcaseI'*:  $\text{equalOn } (\text{preImage } \text{nextX } A) f g \implies \text{equalOn } A (\text{vblcase } x f) (\text{vblcase } x g)$   
(*proof*)

**lemma** *equalOn-vblcaseI*:  $(\text{zeroX} : A \longrightarrow x=y) \implies \text{equalOn } (\text{preImage } \text{nextX } A) f g \implies \text{equalOn } A (\text{vblcase } x f) (\text{vblcase } y g)$   
(*proof*)

**lemma** *X-deX-connection*:  $X n : A = (n : (\text{deX } ' A))$   
(*proof*)

**lemma** *finiteFreshVar*:  $\text{finite } A \implies \text{freshVar } A \sim : A$   
(*proof*)

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

**lemma** *freshVarI2*:  $\text{finite } A \implies !x . x \sim: A \dashrightarrow P x \implies P (\text{freshVar } A)$   
*<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*

**lemma** *signsE*:  $\llbracket \text{signs} = \text{Neg} \implies P; \text{signs} = \text{Pos} \implies P \rrbracket \implies P$   
*<proof>*

**lemma** *expand-signs-case*:  $Q(\text{signs-case } v\text{pos } v\text{neg } F) = ($   
 $(F = \text{Pos} \dashrightarrow Q (v\text{pos})) \ \&$   
 $(F = \text{Neg} \dashrightarrow Q (v\text{neg}))$   
 $)$   
*<proof>*

**primrec** *sign* :: *signs*  $\Rightarrow$  *bool*  $\Rightarrow$  *bool*

**where**

*sign Pos* *x* = *x*  
 | *sign Neg* *x* =  $(\neg x)$

**lemma** *sign-arg-cong*:  $x = y \implies \text{sign } z x = \text{sign } z y$  *<proof>*

**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*:  $\llbracket$   
 ! *p vs*. *P* (*FAtom Pos* *p vs*);  
 ! *p vs*. *P* (*FAtom Neg* *p vs*);  
 !! *A B* .  $\llbracket P A; P B \rrbracket \implies P (\text{FConj Pos } A B)$ ;  
 !! *A B* .  $\llbracket P A; P B \rrbracket \implies P (\text{FConj Neg } A B)$ ;  
 $\rrbracket$

```

!! A . [| P A |] ==> P (FAll Pos A);
!! A . [| P A |] ==> P (FAll Neg A)
|]
==> P A
<proof>

```

**lemma formula-signs-cases: !!P.**

```

[| !! p vs . P (FAtom Pos p vs);
!! p vs . P (FAtom Neg p vs);
!! f1 f2 . P (FConj Pos f1 f2);
!! f1 f2 . P (FConj Neg f1 f2);
!! f1 . P (FAll Pos f1);
!! f1 . P (FAll Neg f1) |]
==> P A
<proof>

```

**lemma strong-formula-induct': !A. (! B. size B < size A --> P B) --> P A**

```

==> ! A. size A = n --> P (A::formula)
<proof>

```

**lemma strong-formula-induct: (! A. (! B. size B < size A --> P B) --> P A)**

```

==> P (A::formula)
<proof>

```

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

```

size B < size (FConj z A B)
size A < size (FAll z A)
<proof>

```

**lemma expand-formula-case:**

```

Q(formula-case fatom fconj fall F) = (
(! z P vs . F = FAtom z P vs --> Q (fatom z P vs)) &
(! z A0 A1 . F = FConj z A0 A1 --> Q (fconj z A0 A1)) &
(! z A . F = FAll z A --> Q (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**

$dual\text{-}FAtom: dual\ p\ q\ r\ (FAtom\ z\ P\ vs) = FAtom\ (p\ z)\ P\ vs$   
 $| dual\text{-}FConj: dual\ p\ q\ r\ (FConj\ z\ A0\ A1) = FConj\ (q\ z)\ (dual\ p\ q\ r\ A0)\ (dual\ p\ q\ r\ A1)$   
 $| dual\text{-}FAll: dual\ p\ q\ r\ (FAll\ z\ body) = FAll\ (r\ z)\ (dual\ p\ q\ r\ body)$

**lemma**  $dualCompose: dual\ p\ q\ r\ o\ dual\ P\ Q\ R = dual\ (p\ o\ P)\ (q\ o\ Q)\ (r\ o\ R)$   
 $\langle proof \rangle$

**lemma**  $dualFNot': dual\ invSign\ invSign\ invSign = FNot$   
 $\langle proof \rangle$

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

**lemma**  $dualId: dual\ id\ id\ id\ A = A$   
 $\langle proof \rangle$

### 3.5 Frees

**primrec**  $freeVarsF :: formula ==> vbl\ set$

**where**

$freeVarsFAtom: freeVarsF\ (FAtom\ z\ P\ vs) = set\ vs$   
 $| freeVarsFConj: freeVarsF\ (FConj\ z\ A0\ A1) = (freeVarsF\ A0)\ Un\ (freeVarsF\ A1)$   
 $| freeVarsFAll: freeVarsF\ (FAll\ z\ body) = preImage\ nextX\ (freeVarsF\ body)$

**definition**

$freeVarsFL :: formula\ list ==> vbl\ set$  **where**  
 $freeVarsFL\ gamma = Union\ (freeVarsF\ ` (set\ gamma))$

**lemma**  $freeVarsF\text{-}FNot[simp]: freeVarsF\ (FNot\ A) = freeVarsF\ A$   
 $\langle proof \rangle$

**lemma**  $finite\text{-}freeVarsF[simp]: finite\ (freeVarsF\ A)$   
 $\langle proof \rangle$

**lemma**  $freeVarsFL\text{-}nil[simp]: freeVarsFL\ ([] ) = \{\}$   
 $\langle proof \rangle$

**lemma**  $freeVarsFL\text{-}cons: freeVarsFL\ (A\#\Gamma) = freeVarsF\ A \cup freeVarsFL\ \Gamma$   
 $\langle proof \rangle$

**lemma**  $finite\text{-}freeVarsFL[simp]: finite\ (freeVarsFL\ gamma)$   
 $\langle proof \rangle$

**lemma** *freeVarsDual*:  $\text{freeVarsF } (\text{dual } p \ q \ r \ A) = \text{freeVarsF } A$   
 ⟨*proof*⟩

### 3.6 Substitutions

**primrec** *subF* ::  $[vbl \Rightarrow vbl, \text{formula}] \Rightarrow \text{formula}$

**where**

*subFAtom*:  $\text{subF } \theta \ (FAtom \ z \ P \ vs) = FAtom \ z \ P \ (\text{map } \theta \ vs)$   
 | *subFConj*:  $\text{subF } \theta \ (FConj \ z \ A0 \ A1) = FConj \ z \ (\text{subF } \theta \ A0) \ (\text{subF } \theta \ A1)$   
 | *subFAll*:  $\text{subF } \theta \ (FAll \ z \ body) =$   
 $FAll \ z \ (\text{subF } (\%v \ . \ (\text{case } v \ \text{of } \text{zeroX} \Rightarrow \text{zeroX} \ | \ \text{nextX } v \Rightarrow \text{nextX } (\theta \ v))))$   
 $body$

**lemma** *size-subF*:  $!!\theta. \text{size } (\text{subF } \theta \ A) = \text{size } (A::\text{formula})$   
 ⟨*proof*⟩

**lemma** *subFNot*:  $!!\theta. \text{subF } \theta \ (FNot \ A) = FNot \ (\text{subF } \theta \ A)$   
 ⟨*proof*⟩

**lemma** *subFDual*:  $!!\theta. \text{subF } \theta \ (\text{dual } p \ q \ r \ A) = \text{dual } p \ q \ r \ (\text{subF } \theta \ A)$   
 ⟨*proof*⟩

**definition**

*instanceF* ::  $[vbl, \text{formula}] \Rightarrow \text{formula}$  **where**  
*instanceF*  $w \ body = \text{subF } (\%v. \text{case } v \ \text{of } \text{zeroX} \Rightarrow w \ | \ \text{nextX } v \Rightarrow v) \ body$

**lemma** *size-instance*:  $!!v. \text{size } (\text{instanceF } v \ A) = \text{size } (A::\text{formula})$   
 ⟨*proof*⟩

**lemma** *instanceFDual*:  $\text{instanceF } u \ (\text{dual } p \ q \ r \ A) = \text{dual } p \ q \ r \ (\text{instanceF } u \ A)$   
 ⟨*proof*⟩

### 3.7 Models

**typedecl**

*object*

**axiomatization** *obj* ::  $\text{nat} \Rightarrow \text{object}$

**where** *inj-obj*:  $\text{inj } \text{obj}$

### 3.8 model, non empty set and positive atom valuation

**typedef** *model* =  $\{ z :: (\text{object set} * ([\text{predicate}, \text{object list}] \Rightarrow \text{bool})) \cdot (\text{fst } z \ \sim = \{\}) \}$  ⟨*proof*⟩

**definition**

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

**definition**

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

**lemma**  $evalP\text{-}arg2\text{-}cong: x = y \Rightarrow evalP\ M\ p\ x = evalP\ M\ p\ y$   $\langle proof \rangle$

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

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

### 3.9 Validity

**primrec**  $evalF :: [model, vbl \Rightarrow object, formula] \Rightarrow bool$

**where**

$evalFAtom: evalF\ M\ phi\ (FAtom\ z\ P\ vs) = sign\ z\ (evalP\ M\ P\ (map\ phi\ vs))$   
 $| evalFConj: evalF\ M\ phi\ (FConj\ z\ A0\ A1) = sign\ z\ (sign\ z\ (evalF\ M\ phi\ A0) \&$   
 $sign\ z\ (evalF\ M\ phi\ A1))$   
 $| evalFAll: evalF\ M\ phi\ (FAll\ z\ body) = sign\ z\ (!x: (objects\ M).$   
 $sign\ z$   
 $(evalF\ M\ (\%v . (case\ v\ of$   
 $zeroX \Rightarrow x$   
 $| nextX\ v \Rightarrow phi\ v))\ body))$

**definition**

$valid :: formula \Rightarrow bool$  **where**  
 $valid\ F \longleftrightarrow (\forall\ M\ phi. evalF\ M\ phi\ F = True)$

**lemma**  $evalF\text{-}FAll: evalF\ M\ phi\ (FAll\ Pos\ A) = (!x: (objects\ M). (evalF\ M\ (vblcase$   
 $x\ (\%v . phi\ v))\ A))$   
 $\langle proof \rangle$

**lemma**  $evalF\text{-}FEx: evalF\ M\ phi\ (FAll\ Neg\ A) = (?x:(objects\ M). (evalF\ M$   
 $(vblcase\ x\ (\%v . phi\ v))\ A))$   
 $\langle proof \rangle$

**lemma**  $evalF\text{-}arg2\text{-}cong: x = y \Rightarrow evalF\ M\ p\ x = evalF\ M\ p\ y$   $\langle proof \rangle$

**lemma**  $evalF\text{-}FNot: !phi. evalF\ M\ phi\ (FNot\ A) = (\neg\ evalF\ M\ phi\ A)$   
 $\langle proof \rangle$

**lemma**  $evalF\text{-}equiv[rule\text{-}format]: !f\ g. (equalOn\ (freeVarsF\ A)\ f\ g) \longrightarrow (evalF\ M$   
 $f\ A = evalF\ M\ g\ A)$   
 $\langle proof \rangle$

**lemma**  $evalF\text{-}subF\text{-}eq: !phi\ theta. evalF\ M\ phi\ (subF\ theta\ A) = evalF\ M\ (phi\ o$   
 $theta)\ A$

```

    <proof>

lemma o-id'[simp]: f o (% x. x) = f
<proof>

lemma evalF-instance: evalF M phi (instanceF u A) = evalF M (vblcase (phi u)
phi) A
    <proof>

lemma s[simp]: FConj signs formula1 formula2 ≠ formula1
    <proof>

lemma s'[simp]: FConj signs formula1 formula2 ≠ formula2
    <proof>

lemma instanceF-E: instanceF g formula ≠ FAll signs formula
    <proof>

end

```

## 4 Sequents

```

theory Sequents
imports Formula
begin

types sequent = formula list

definition
  evalS :: [model,vbl => object,formula list] => bool where
  evalS M phi fs ←→ (? f : set fs . evalF M phi f = True)

lemma evalS-nil[simp]: evalS M phi [] = False
    <proof>

lemma evalS-cons[simp]: evalS M phi (A # Gamma) = (evalF M phi A | evalS
M phi Gamma)
    <proof>

lemma evalS-append: evalS M phi (Gamma @ Delta) = (evalS M phi Gamma |
evalS M phi Delta)
    <proof>

lemma evalS-equiv[rule-format]: (equalOn (freeVarsFL Gamma) f g) --> (evalS
M f Gamma = evalS M g Gamma)
    <proof>

```

**definition**

*modelAssigns* :: [model] => (vbl => object) set **where**  
*modelAssigns* M = { phi . range phi <= objects M }

**lemma** *modelAssignsI*: range f <= objects M ==> f : *modelAssigns* M  
 ⟨proof⟩

**lemma** *modelAssignsD*: f : *modelAssigns* M ==> range f <= objects M  
 ⟨proof⟩

**definition**

*validS* :: formula list => bool **where**  
*validS* fs <=> (! M . ! phi : *modelAssigns* M . evalS M phi fs = True)

## 4.1 Rules

**types** rule = sequent \* (sequent set)

**definition**

*concR* :: rule => sequent **where**  
*concR* = (%(conc,prems). conc)

**definition**

*premsR* :: rule => sequent set **where**  
*premsR* = (%(conc,prems). prems)

**definition**

*mapRule* :: (formula => formula) => rule => rule **where**  
*mapRule* = (%f (conc,prems) . (map f conc,(map f) ‘ prems))

**lemma** *mapRuleI*: [ A = map f a; B = (map f) ‘ b ] ==> (A,B) = *mapRule* f  
 (a,b)  
 ⟨proof⟩

## 4.2 Deductions

**inductive-set**

*deductions* :: rule set => formula list set  
**for** *rules* :: rule set

**where**

*inferI*: [ (conc,prems) : rules;  
           prems : Pow(deductions(rules))  
 ] ==> conc : deductions(rules)

**monos** Pow-mono

**lemma** *mono-deductions*: [ A <= B ] ==> deductions(A) <= deductions(B)  
 ⟨proof⟩

### 4.3 Basic Rule sets

**definition**

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

**definition**

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

**definition**

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

**definition**

$Alls = \{ z. ? A x \quad Gamma. z = (FAll Pos A \# Gamma, \{instanceF x A \# Gamma\}) \ \& \ x \sim: freeVarsFL (FAll Pos A \# Gamma) \}$

**definition**

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

**definition**

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

**definition**

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

**definition**

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

**definition**

$Perms = \{ z. ? Gamma Gamma' \quad . z = (Gamma, \{Gamma'\}) \ \& \ Gamma <\sim\sim> Gamma' \}$

**definition**

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

**lemma** *AxiomI*:  $[[ Axioms <= A ]] ==> [FAtom Pos p vs, FAtom Neg p vs] : deductions(A)$   
 ⟨proof⟩

**lemma** *DAxiomsI*:  $[[ DAxioms <= A ]] ==> [FAtom Neg p vs, FAtom Pos p vs] : deductions(A)$   
 ⟨proof⟩

**lemma** *DisjI*:  $[[ A0 \# A1 \# Gamma : deductions(A); Disjs <= A ]] ==> (FConj Neg A0 A1 \# Gamma) : deductions(A)$   
 ⟨proof⟩

**lemma** *ConjI*:  $[[ (A0 \# Gamma) : deductions(A); (A1 \# Delta) : deductions(A); Conjs <= A ]] ==> FConj Pos A0 A1 \# Gamma @ Delta : deductions(A)$   
 ⟨proof⟩

**lemma** *AllI*:  $[[ instanceF w A \# Gamma : deductions(R); w \sim: freeVarsFL (FAll Pos A \# Gamma); Alls <= R ]] ==> (FAll Pos A \# Gamma) : deductions(R)$   
 ⟨proof⟩

**lemma** *ExI*:  $[[ \text{instanceF } w \ A \# \Gamma : \text{deductions}(R); \text{Exs} \leq R ]] \implies (\text{Fall Neg } A \# \Gamma) : \text{deductions}(R)$   
 ⟨proof⟩

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

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

**lemma** *PermI*:  $[[ \Gamma' : \text{deductions } R; \Gamma \sim \Gamma'; \text{Perms} \leq R ]] \implies \Gamma : \text{deductions}(R)$   
 ⟨proof⟩

#### 4.4 Derived Rules

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

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

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

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

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

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

**lemma** *weaken-append[rule-format]*:  $\text{Contrs} \leq R \implies \text{Perms} \leq R \implies !A. A @ \Gamma @ \Gamma : \text{deductions}(R) \dashrightarrow A @ \Gamma : \text{deductions}(R)$   
 ⟨proof⟩

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

**lemma** *ConjI'*: [| (*A0*#*Gamma*) : deductions(*A*); (*A1*#*Gamma*) : deductions(*A*);  
*Contrs* <= *A*; *Conjs* <= *A*; *Perms* <= *A* |] ==> *FConj Pos A0 A1#Gamma* :  
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

- these lemmas can be used to replace complicated permutation reasoning above
- essentially if *x* is a deduction, and set *x* subset set *y*, then *y* is a deduction

### definition

*inDed* :: formula list => bool **where**  
*inDed* *xs* <=> *xs* : deductions *CutFreePC*

**lemma** *perm*: ! *xs* *ys*. *xs* <~> *ys* --> (*inDed* *xs* = *inDed* *ys*)

⟨*proof*⟩

**lemma** *contr*: ! *x* *xs*. *inDed* (*x*#*x*#*xs*) --> *inDed* (*x*#*xs*)

⟨*proof*⟩

**lemma** *weak*: ! *x* *xs*. *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*  $x \implies \text{set } x \leq \text{set } y \implies \text{inDed } y$   
 ⟨*proof*⟩

**end**

**theory** *Tree* **imports** *Main* **begin**

## 4.7 Tree

**inductive-set**

*tree* :: [ $'a \implies 'a \text{ set}, 'a$ ]  $\implies (\text{nat} * 'a) \text{ set}$   
**for** *subs* ::  $'a \implies 'a \text{ set}$  **and** *gamma* ::  $'a$

**where**

*tree0*:  $(0, \text{gamma}) : \text{tree } \text{subs } \text{gamma}$

| *tree1*: [|  $(n, \text{delta}) : \text{tree } \text{subs } \text{gamma}; \text{sigma} : \text{subs}(\text{delta})$  |]  
 $\implies (\text{Suc } n, \text{sigma}) : \text{tree } \text{subs } \text{gamma}$

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

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

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

**lemma** *tree1Eq* [*rule-format*]:

$\forall Y. (\text{Suc } n, Y) \in \text{tree } \text{subs } \text{gamma} = (\exists \text{sigma} \in \text{subs } \text{gamma} . (n, Y) \in \text{tree } \text{subs } \text{sigma})$   
 ⟨*proof*⟩

**definition**

*incLevel* ::  $\text{nat} * 'a \implies \text{nat} * 'a$  **where**  
*incLevel* =  $(\% (n, a). (\text{Suc } n, a))$

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

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

**definition**

*fans* :: [ $'a \implies 'a \text{ set}$ ]  $\implies \text{bool}$  **where**  
*fans* *subs*  $\longleftrightarrow (!x. \text{finite } (\text{subs } x))$

**lemma** *fansD*: *fans* *subs*  $\implies \text{finite } (\text{subs } A)$   
 ⟨*proof*⟩

**lemma fansI:**  $(!A. \text{finite } (\text{subs } A)) \implies \text{fans } \text{subs}$   
 $\langle \text{proof} \rangle$

## 4.8 Terminal

### definition

$\text{terminal} :: ['a \Rightarrow 'a \text{ set}, 'a] \Rightarrow \text{bool}$  **where**  
 $\text{terminal } \text{subs } \text{delta} \iff \text{subs}(\text{delta}) = \{\}$

**lemma terminalD:**  $\text{terminal } \text{subs } \text{Gamma} \implies x \sim: \text{subs } \text{Gamma}$   
 $\langle \text{proof} \rangle$

**lemma terminalI:**  $x \in \text{subs } \text{Gamma} \implies \sim \text{terminal } \text{subs } \text{Gamma}$   
 $\langle \text{proof} \rangle$

## 4.9 Inherited

### definition

$\text{inherited} :: ['a \Rightarrow 'a \text{ set}, (\text{nat} * 'a) \text{ set} \Rightarrow \text{bool}] \Rightarrow \text{bool}$  **where**  
 $\text{inherited } \text{subs } P \iff (!A B. (P A \& P B) = P (A \text{ Un } B))$   
 $\quad \& (!A. P A = P (\text{incLevel } ' A))$   
 $\quad \& (!n \text{ Gamma } A. \sim(\text{terminal } \text{subs } \text{Gamma}) \dashrightarrow P A = P (\text{insert}$   
 $(n, \text{Gamma}) A))$   
 $\quad \& (P \{\})$

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

**lemma inheritedUn[rule-format]:**  $\text{inherited } \text{subs } P \dashrightarrow P A \dashrightarrow P B \dashrightarrow P$   
 $(A \text{ Un } B)$   
 $\langle \text{proof} \rangle$

**lemma inheritedIncLevel[rule-format]:**  $\text{inherited } \text{subs } P \dashrightarrow P A \dashrightarrow P (\text{incLevel}$   
 $' A)$   
 $\langle \text{proof} \rangle$

**lemma inheritedEmpty[rule-format]:**  $\text{inherited } \text{subs } P \dashrightarrow P \{\}$   
 $\langle \text{proof} \rangle$

**lemma inheritedInsert[rule-format]:**

$\text{inherited } \text{subs } P \dashrightarrow \sim(\text{terminal } \text{subs } \text{Gamma}) \dashrightarrow P A \dashrightarrow P (\text{insert}$   
 $(n, \text{Gamma}) A)$   
 $\langle \text{proof} \rangle$

**lemma inheritedI[rule-format]:**  $[| \forall A B. (P A \& P B) = P (A \text{ Un } B);$   
 $\forall A. P A = P (\text{incLevel } ' A);$

$\forall n \text{ Gamma } A . \sim(\text{terminal subs Gamma}) \dashrightarrow P A = P (\text{insert } (n, \text{Gamma}) A)$ ;  
 $P \{ \} [] \implies \text{inherited subs } P$   
 ⟨proof⟩

**lemma** *inheritedUnEq*[rule-format, symmetric]: *inherited subs*  $P \dashrightarrow (P A \& P B) = P (A \text{ Un } B)$   
 ⟨proof⟩

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

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

**lemmas** *inheritedUnD* = *iffD1*[OF *inheritedUnEq*]

**lemmas** *inheritedInsertD* = *inheritedInsertEq*[THEN *iffD1*]

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

**lemma** *inheritedUNEQ*[rule-format]:  
 $\text{finite } A \dashrightarrow \text{inherited subs } P \dashrightarrow (!x:A. P (B x)) = P (UN a:A. B a)$   
 ⟨proof⟩

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

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

**lemma** *inheritedPropagateEq*[rule-format]: **assumes**  $a$ : *inherited subs*  $P$   
**and**  $b$ : *fans subs*  
**and**  $c$ :  $\sim(\text{terminal subs delta})$   
**shows**  $P(\text{tree subs delta}) = (!\sigma:\text{subs delta}. P(\text{tree subs sigma}))$   
 ⟨proof⟩

**lemma** *inheritedPropagate*:  
 $[[\sim P(\text{tree subs delta}); \text{inherited subs } P; \text{fans subs}; \sim(\text{terminal subs delta})]]$   
 $\implies \exists \sigma \in \text{subs delta} . \sim P(\text{tree subs sigma})$   
 ⟨proof⟩

**lemma** *inheritedViaSub*:  $[[\text{inherited subs } P; \text{fans subs}; P(\text{tree subs delta}); \sigma \in \text{subs delta} ]]$   
 $\implies P(\text{tree subs sigma})$   
 ⟨proof⟩

**lemma** *inheritedJoin*[rule-format]:

*(inherited subs P & inherited subs Q) --> inherited subs (%x. P x & Q x)*  
 <proof>

**lemma inheritedJoinI**[rule-format]:  $[[ \textit{inherited subs } P; \textit{inherited subs } Q; R = ( \% x . P x \ \& \ Q x ) ] ] \implies \textit{inherited subs } R$   
 <proof>

## 4.10 bounded, boundedBy

### definition

*boundedBy* ::  $\textit{nat} \implies (\textit{nat} * 'a) \textit{set} \implies \textit{bool}$  **where**  
*boundedBy*  $N \ A \longleftrightarrow (\forall (n, \textit{delta}) \in A. n < N)$

### definition

*bounded* ::  $(\textit{nat} * 'a) \textit{set} \implies \textit{bool}$  **where**  
*bounded*  $A \longleftrightarrow (\exists N . \textit{boundedBy } N \ A)$

**lemma boundedByEmpty**[simp]: *boundedBy*  $N \ \{\}$   
 <proof>

**lemma boundedByInsert**: *boundedBy*  $N \ (\textit{insert } (n, \textit{delta}) \ B) = (n < N \ \& \ \textit{boundedBy } N \ B)$   
 <proof>

**lemma boundedByUn**: *boundedBy*  $N \ (A \ \textit{Un} \ B) = (\textit{boundedBy } N \ A \ \& \ \textit{boundedBy } N \ B)$   
 <proof>

**lemma boundedByIncLevel'**: *boundedBy*  $(\textit{Suc } N) \ (\textit{incLevel } ' \ A) = \textit{boundedBy } N \ A$   
 <proof>

**lemma boundedByAdd1**: *boundedBy*  $N \ B \implies \textit{boundedBy } (N+M) \ B$   
 <proof>

**lemma boundedByAdd2**: *boundedBy*  $M \ B \implies \textit{boundedBy } (N+M) \ B$   
 <proof>

**lemma boundedByMono**: *boundedBy*  $m \ B \implies m < M \implies \textit{boundedBy } M \ B$   
 <proof>

**lemmas boundedByMonoD** = *boundedByMono*

**lemma boundedBy0**: *boundedBy*  $0 \ A = (A = \{\})$   
 <proof>

**lemma boundedBySuc'**: *boundedBy*  $N \ A \implies \textit{boundedBy } (\textit{Suc } N) \ A$   
 <proof>

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

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

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

#### 4.11 Inherited Properties- bounded

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

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

**lemma** *boundedIncLevel*:  $\text{bounded (incLevel } \text{' } A) = (\text{bounded } A)$   
 ⟨proof⟩

**lemma** *boundedInsert*:  $\text{bounded (insert } a \ B) = (\text{bounded } B)$   
 ⟨proof⟩

**lemma** *inheritedBounded*:  $\text{inherited subs bounded}$   
 ⟨proof⟩

#### 4.12 founded

##### definition

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

**lemma** *foundedD*:  $\text{founded subs } P \text{ (tree subs delta)} \implies \text{terminal subs delta} \implies P \ \text{delta}$   
 ⟨proof⟩

**lemma** *foundedMono*:  $[[ \text{founded subs } P \ A; \forall x. P \ x \longrightarrow Q \ x ]] \implies \text{founded subs } Q \ A$   
 ⟨proof⟩

**lemma** *foundedSubs*:  $\text{founded subs } P \text{ (tree subs Gamma)} \implies \text{sigma} \in \text{subs } \text{Gamma} \implies \text{founded subs } P \text{ (tree subs sigma)}$   
 ⟨proof⟩

### 4.13 Inherited Properties- founded

**lemma** *foundedInsert*[rule-format]:  $\sim$  terminal subs delta  $\implies$  founded subs  $P$   
(insert (n,delta) B) = (founded subs  $P$  B)  
{proof}

**lemma** *foundedUn*: (founded subs  $P$  (A Un B)) = (founded subs  $P$  A & 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

**lemmas** *finiteInsert* = finite-insert

**lemma** *finiteUn*: finite (A Un B) = (finite A & finite B)  
{proof}

**lemma** *finiteIncLevel*: finite (incLevel ' A) = finite A  
{proof}

**lemma** *finiteEmpty*: finite {} {proof}

**lemma** *inheritedFinite*: inherited subs (%A. finite A)  
{proof}

### 4.15 path: follows a failing inherited property through tree

#### definition

*failingSub* :: [ $'a \implies 'a$  set, (nat \*  $'a$ ) set  $\implies$  bool,  $'a$ ]  $\implies$   $'a$  where  
*failingSub* subs  $P$  gamma = (SOME sigma. (sigma:subs gamma &  $\sim P$ (tree subs sigma)))

**lemma** *failingSubProps*: [[ inherited subs  $P$ ;  $\sim P$  (tree subs gamma);  $\sim$ (terminal subs gamma); fans subs ]]  
 $\implies$  *failingSub* subs  $P$  gamma  $\in$  subs gamma &  $\sim$ ( $P$  (tree subs (*failingSub* subs  $P$  gamma)))  
{proof}

**lemma** *failingSubFailsI*: [[ inherited subs  $P$ ;  $\sim P$  (tree subs gamma);  $\sim$ (terminal subs gamma); fans subs ]]  
 $\implies$   $\sim$ ( $P$  (tree subs (*failingSub* subs  $P$  gamma)))

*<proof>*

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

**lemma** *failingSubSubs: [| inherited subs P; ~ P (tree subs gamma); ~(terminal subs gamma); fans subs |]*

*==> failingSub subs P gamma ∈ subs gamma*

*<proof>*

**primrec** *path :: ['a ==> 'a set, 'a, (nat \* 'a) set ==> bool, nat] ==> 'a*

**where**

*path0: path subs gamma P 0 = gamma*

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

**lemma** *pathFailsP: [| inherited subs P; fans subs; ~ P (tree subs gamma) |]*

*==> ~ (P (tree subs (path subs gamma P n)))*

*<proof>*

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

**lemma** *pathTerminal[rule-format]: [| inherited subs P; fans subs; terminal subs gamma |]*

*==> terminal subs (path subs gamma P n)*

*<proof>*

**lemma** *pathStarts: path subs gamma P 0 = gamma*

*<proof>*

**lemma** *pathSubs: [| inherited subs P; fans subs; ~ P (tree subs gamma); ~ (terminal subs (path subs gamma P n)) |]*

*==> path subs gamma P (Suc n) ∈ subs (path subs gamma P n)*

*<proof>*

**lemma** *pathStops: terminal subs (path subs gamma P n) ==> path subs gamma P (Suc n) = path subs gamma P n*

*<proof>*

## 4.16 Branch

**definition**

*branch :: ['a ==> 'a set, 'a, nat ==> 'a] ==> bool* **where**

*branch subs Gamma f <math>\longleftrightarrow f 0 = Gamma*

*& (!n . terminal subs (f n) --> f (Suc n) = f n)*

*& (!n . ~ terminal subs (f n) --> f (Suc n) ∈ subs (f n))*

**lemma** *branch0: branch subs Gamma f ==> f 0 = Gamma*

*<proof>*

**lemma** *branchStops*: *branch subs Gamma f ==> terminal subs (f n) ==> f (Suc n) = f n*  
*<proof>*

**lemma** *branchSubs*: *branch subs Gamma f ==> ~ terminal subs (f n) ==> f (Suc n) ∈ subs (f n)*  
*<proof>*

**lemma** *branchI*:  $\llbracket (f\ 0 = \text{Gamma});$   
 $!n . \text{terminal subs } (f\ n) \dashrightarrow f\ (\text{Suc } n) = f\ n;$   
 $!n . \sim \text{terminal subs } (f\ n) \dashrightarrow f\ (\text{Suc } n) \in \text{subs } (f\ n) \rrbracket ==> \text{branch subs}$   
*Gamma f*  
*<proof>*

**lemma** *branchTerminalPropagates*: *branch subs Gamma f ==> terminal subs (f m) ==> terminal subs (f (m + n))*  
*<proof>*

**lemma** *branchTerminalMono*: *branch subs Gamma f ==> m < n ==> terminal subs (f m) ==> terminal subs (f n)*  
*<proof>*

**lemma** *branchPath*:  
 $\llbracket \text{inherited subs } P; \text{fans subs}; \sim P(\text{tree subs gamma}) \rrbracket$   
 $==> \text{branch subs gamma } (\text{path subs gamma } P)$   
*<proof>*

## 4.17 failing branch property: abstracts path defn

**lemma** *failingBranchExistence*:  $!!\text{subs}.$   
 $\llbracket \text{inherited subs } P; \text{fans subs}; \sim P(\text{tree subs gamma}) \rrbracket$   
 $==> \exists f . \text{branch subs gamma } f \ \& \ (\forall n . \sim P(\text{tree subs } (f\ n)))$   
*<proof>*

### definition

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

**lemma** *infBranchI*:  $\llbracket (f\ 0 = \text{Gamma}); !n . f\ (\text{Suc } n) \in \text{subs } (f\ n) \rrbracket ==> \text{inf-Branch subs Gamma f}$   
*<proof>*

## 4.18 Tree induction principles

— we work hard to use nothing fancier than induction over naturals

**lemma** *boundedTreeInduction'*:  
 $\llbracket \text{fans subs};$

```

    ∀ delta. ~ terminal subs delta --> (∀ sigma ∈ subs delta. P sigma) --> P
delta ]]
    ==> ∀ Gamma. boundedBy m (tree subs Gamma) → founded subs P (tree subs
Gamma) → P Gamma
    <proof>

```

**lemma** *boundedTreeInduction*:

```

[[ fans subs;
   bounded (tree subs Gamma); founded subs P (tree subs Gamma);
   ∀ delta. ~ terminal subs delta --> (∀ sigma ∈ subs delta. P sigma) --> P delta
]] ==> P Gamma
    <proof>

```

**lemma** *boundedTreeInduction2'*:

```

[[ fans subs;
   ∀ delta. (∀ sigma ∈ subs delta. P sigma) --> P delta ]]
==> ∀ Gamma. boundedBy m (tree subs Gamma) → P Gamma
    <proof>

```

**lemma** *boundedTreeInduction2*:

```

[[ fans subs; boundedBy m (tree subs Gamma);
   ∀ delta. (∀ sigma ∈ subs delta. P sigma) --> P delta ]]
==> P Gamma
    <proof>

```

end

## 5 Completeness

```

theory Completeness
imports Tree Sequents
begin

```

### 5.1 pseq: type represents a processed sequent

```

types atom = (signs * predicate * vbl list)
        nform = (nat * formula)
        pseq = (atom list * nform list)

```

**definition**

```

sequent :: pseq => formula list where
sequent = (%(atoms,nforms) . map snd nforms @ map (% (z,p,vs) . FAtom z p
vs) atoms)

```

**definition**

```

pseq :: formula list => pseq where
pseq fs = ([],map (%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**  
 $SATAxiom\ fs \longleftrightarrow (? n\ vs . FAtom\ Pos\ n\ vs : set\ fs \ \&\ FAtom\ Neg\ n\ vs : set\ fs)$

## 5.3 subs: a CutFreePC justifiable backwards proof step

**definition**

$subsFAtom :: [atom\ list, (nat * formula)\ list, signs, predicate, vbl\ list] \Rightarrow pseq\ set$   
**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**  
 $subs = (\% pseq .$   
 $\quad if\ SATAxiom\ (sequent\ pseq)\ then$   
 $\quad \quad \{ \}$   
 $\quad else\ let\ (atms, nforms) = pseq$   
 $\quad \quad in\ case\ nforms\ of$   
 $\quad \quad \quad [] \Rightarrow \{ \}$   
 $\quad \quad \quad | nA \# nAs \Rightarrow let\ (n, A) = nA$   
 $\quad \quad \quad \quad in\ (case\ A\ of$   
 $\quad \quad \quad \quad \quad FAtom\ z\ P\ vs \Rightarrow subsFAtom\ atms\ nAs\ z\ P\ vs$   
 $\quad \quad \quad \quad \quad | FConj\ z\ A0\ A1 \Rightarrow subsFConj\ atms\ nAs\ z\ A0\ A1$   
 $\quad \quad \quad \quad \quad | FAll\ z\ A \Rightarrow subsFAll\ atms\ nAs\ n\ z\ A\ (freeVarsFL$   
 $\quad \quad \quad \quad \quad \quad (sequent\ pseq))))$

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

**definition**

*proofTree* :: (nat \* pseq) set => bool **where**  
*proofTree* A  $\longleftrightarrow$  bounded A & founded subs (SATAxiom o 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  $\longleftrightarrow$  nA : set (snd (f n))

### definition

*costBarrier* :: [nat \* formula, pseq] => nat **where**

*costBarrier* nA = (%(atms, nAs).  
 let barrier = takeWhile (%x. nA  $\sim$  x) nAs  
 in let costs = map (exp 3 o size o 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 = (%p args . ! i .  $\sim$ (EV (contains f (i, FAtom Pos p (map (X o 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 gamma)  
 <proof>

**lemma** fansSubs: fans subs

*<proof>*

**lemma** subs-def2:

!!gamma.

~ SATAxiom (sequent gamma) ==>

subs gamma = (case nforms gamma of

[] => {}

| nA#nAs => let (n,A) = nA

in (case A of

FAtom z P vs => subsFAtom (atoms gamma)

nAs z P vs

| FConj z A0 A1 => subsFConj (atoms gamma)

nAs z A0 A1

| FAll z A => subsFAll (atoms gamma) nAs n

z A (freeVarsFL (sequent gamma))))

*<proof>*

## 5.9 inherited: proofTree

**lemma** proofTree-def2: proofTree = (% x . bounded x & founded subs (SATAxiom o sequent) x)

*<proof>*

**lemma** inheritedProofTree: inherited subs proofTree

*<proof>*

**lemma** proofTreeI: [| bounded A; founded subs (SATAxiom o 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 o (Pair x)) = (% x. x)

*<proof>*

**lemma** sequent-pseq: sequent (pseq fs) = fs

*<proof>*

## 5.11 SATAxiom: proofTree

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

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

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

**lemma** *SATAxiomProofTree*[rule-format]: *SATAxiom* (sequent gamma)  $\dashrightarrow$  *proofTree* (tree subs gamma)  
(proof)

**lemma** *SATAxiomEq*: (*proofTree* (tree subs gamma) & *terminal subs gamma*) = *SATAxiom* (sequent gamma)  
(proof)

## 5.12 SATAxioms are deductions: - needed

**lemma** *SAT-deduction*: *SATAxiom*  $x \implies x$  : *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**  $\neg$  *SATAxiom* (sequent (ats, (n, f) # list))  $\dashrightarrow$   $\neg$  *terminal subs* (ats, (n, f) # list)

$\dashrightarrow$  ( $\forall$  sigma  $\in$  *subs* (ats, (n, f) # list). sequent sigma  $\in$  *deductions CutFreePC*)

$\dashrightarrow$  sequent (ats, (n, f) # list)  $\in$  *deductions CutFreePC*

(proof)

**lemma** *subsJustified*: !! gamma.  $\sim$  *terminal subs gamma*

$\implies$  ! sigma : *subs gamma* . sequent sigma : *deductions* (*CutFreePC*)

$\implies$  sequent gamma : *deductions* (*CutFreePC*)

(proof)

## 5.14 proofTrees are deductions: instance of boundedTreeInduction

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

**lemma** *proofTreeDeductionD*[*rule-format*]: *proofTree*(*tree subs gamma*)  $\implies$  *sequent gamma : deductions* (*CutFreePC*)  
 ⟨*proof*⟩

### 5.15 contains, considers:

**lemma** *contains-def2*: *contains f iA n* = (*iA : set* (*nforms* (*f n*)))  
 ⟨*proof*⟩

**lemma** *considers-def2*: *considers f iA n* = ( ? *nAs . nforms* (*f n*) = *iA#nAs*)  
 ⟨*proof*⟩

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

### 5.16 path: nforms = [] implies

**lemma** *nformsNoContains*: [] *branch subs gamma f; !n . ~proofTree* (*tree subs* (*f n*)); *nforms* (*f n*) = []  $\implies$  ~ *contains f iA n*  
 ⟨*proof*⟩

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

**lemma** *nformsStops*: !!*f*.

[] *branch subs gamma f; !n . ~proofTree* (*tree subs* (*f n*));  
*nforms* (*f n*) = [] []  
 $\implies$  *nforms* (*f* (*Suc n*)) = [] & *atoms* (*f* (*Suc n*)) = *atoms* (*f n*)  
 ⟨*proof*⟩

### 5.17 path: cases

**lemma** *terminalNFormCases*: !!*f*. *terminal subs* (*f n*) | (? *i A nAs . nforms* (*f n*) = (*i,A*)#*nAs*)  
 ⟨*proof*⟩

**lemma** *cases*[*elim-format*]: *terminal subs* (*f n*) | ( $\neg$  (*terminal subs* (*f n*)  $\wedge$  (? *i A nAs . nforms* (*f n*) = (*i,A*)#*nAs*)))  
 ⟨*proof*⟩

### 5.18 path: contains not terminal and propagate condition

**lemma** *containsNotTerminal*: [] *branch subs gamma f; !n . ~proofTree* (*tree subs* (*f n*)); *contains f iA n* []  $\implies$  ~ (*terminal subs* (*f n*))  
 ⟨*proof*⟩

**lemma** *containsPropagates*: !!*f*.

[] *branch subs gamma f; !n . ~proofTree* (*tree subs* (*f n*));  
*contains f iA n* []  
 $\implies$  *contains f iA* (*Suc n*) | *considers f iA n*  
 ⟨*proof*⟩

## 5.19 path: no consider lemmas

**lemma** *noConsidersD*:  $!!f. \sim \text{considers } f \text{ iA } n \implies n \text{ forms } (f \ n) = x \# xs \implies \text{iA} \sim = x$   
*<proof>*

**lemma** *considersD*:  $!!f. \text{considers } f \text{ iA } n \implies ? \ xs . n \text{ forms } (f \ n) = \text{iA} \# xs$   
*<proof>*

## 5.20 path: contains initially

**lemma** *contains-initially*:

*branch subs (pseq gamma) f  $\implies$  A : set gamma  $\implies$  (contains f (0,A) 0)*  
*<proof>*

**lemma** *contains-initialEVs*:

*branch subs (pseq gamma) f  $\implies$  A : set gamma  $\implies$  EV (contains f (0,A))*  
*<proof>*

## 5.21 termination: (for EV contains implies EV considers)

**lemmas** *r = wf-induct*[of measure *mSrFn*, *OF wf-measure*]

**lemmas** *r' = r*[*simplified measure-def inv-image-def less-than-def less-eq mem-Collect-eq*]

**lemma** *r''*:  $(\forall x. (\forall y. ((\text{mSrFn}::'a \implies \text{nat}) \ y) < ((\text{mSrFn}::'a \implies \text{nat}) \ x)) \longrightarrow P \ y) \longrightarrow P \ x) \implies P \ a$   
*<proof>*

**lemma** *terminationRule* [rule-format]:

$! \ n. P \ n \dashrightarrow (\sim(P \ (\text{Suc } n)) \mid (P \ (\text{Suc } n) \ \& \ \text{mSrFn} \ (\text{Suc } n) < (\text{mSrFn}::\text{nat} \implies \text{nat}) \ n)) \implies P \ m \dashrightarrow (? \ n . P \ n \ \& \ \sim(P \ (\text{Suc } n)))$   
(**is** -  $\implies$  ?*P m*)  
*<proof>*

## 5.22 costBarrier: lemmas

### 5.23 costBarrier: exp3 lemmas - bit specific...

**lemma** *exp3Min*:  $\text{exp } 3 \ a > 0$   
*<proof>*

**lemma** *exp1*:  $\text{exp } 3 \ (A) + \text{exp } 3 \ (B) < 3 * ((\text{exp } 3 \ A) * (\text{exp } 3 \ B))$   
*<proof>*

**lemma** *exp1'*:  $\text{exp } 3 \ (A) < 3 * ((\text{exp } 3 \ A) * (\text{exp } 3 \ B)) + C$   
*<proof>*

**lemma** *exp2*:  $\text{Suc } 0 < 3 * \text{exp } 3 \ (B)$   
*<proof>*

**lemma** *expSum*:  $\text{exp } x \ (a+b) = (\text{exp } x \ a) * (\text{exp } x \ b)$

*<proof>*

## 5.24 costBarrier: decreases whilst contains and unconsiders

**lemma** *costBarrierDecreases'*:

**notes** *ss = subs-def2 nforms-def Let-def subsFAtom-def subsFConj-def subsFAll-def costBarrier-def atoms-def exp3Min expSum*

**shows**  $\sim \text{SATAxiom}$  (sequent

$(a, (\text{num}, \text{fm}) \# \text{list}) \dashrightarrow iA \sim = (\text{num}, \text{fm}) \dashrightarrow \neg \text{proofTree} (\text{tree subs } (a, (\text{num}, \text{fm}) \# \text{list}) \dashrightarrow f\text{Sucn} : \text{subs } (a, (\text{num}, \text{fm}) \# \text{list}) \dashrightarrow iA \in \text{set list} \dashrightarrow \text{costBarrier } iA (f\text{Sucn}) < \text{costBarrier } iA (a, (\text{num}, \text{fm}) \# \text{list})$

*<proof>*

**lemma** *costBarrierDecreases*:

$[[ \text{branch subs gamma } f;$

$!n . \sim \text{proofTree} (\text{tree subs } (f n));$

$\text{contains } f iA n;$

$\sim (\text{considers } f iA) n$

$[[ \implies \text{costBarrier } iA (f (\text{Suc } n)) < \text{costBarrier } iA (f n)$

*<proof>*

## 5.25 path: EV contains implies EV considers

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

*<proof>*

**lemma** *containsConsiders*:  $[[ \text{branch subs gamma } f; !n . \sim \text{proofTree} (\text{tree subs } (f n));$

$\text{EV } (\text{contains } f iA) [[$

$\implies \text{EV } (\text{considers } f iA)$

*<proof>*

## 5.26 EV contains: common lemma

**lemma** *lemmaA*:

$[[ \text{branch subs gamma } f; !n . \sim \text{proofTree} (\text{tree subs } (f n));$

$\text{EV } (\text{contains } f (i, A)) [[$

$\implies ? n nAs . \sim \text{SATAxiom} (\text{sequent } (f n)) \ \& \ (n\text{forms } (f n) = (i, A) \# nAs \ \& \ f (\text{Suc } n) : \text{subs } (f n))$

*<proof>*

## 5.27 EV contains: FConj, FDisj, FAll

**lemma** *EV-disj*:  $(\text{EV } P \mid \text{EV } Q) = \text{EV } (\lambda n . P n \mid Q n)$

*<proof>*

**lemma** *evContainsConj*:  $[[ \text{EV } (\text{contains } f (i, \text{FConj Pos } A0 A1));$

$\text{branch subs gamma } f; !n . \sim \text{proofTree} (\text{tree subs } (f n))$

$[[ \implies \text{EV } (\text{contains } f (0, A0)) \mid \text{EV } (\text{contains } f (0, A1))$

*<proof>*

**lemma** *evContainsDisj*:  $\llbracket EV (\text{contains } f (i, FConj \text{ Neg } A0 \ A1));$   
*branch subs gamma f; !n . ~ proofTree (tree subs (f n))*  
 $\rrbracket \implies EV (\text{contains } f (0, A0)) \ \& \ EV (\text{contains } f (0, A1))$   
*<proof>*

**lemma** *evContainsAll*:  
 $\llbracket EV (\text{contains } f (i, FAll \text{ Pos } \text{body}));$  *branch subs gamma f; !n . ~ proofTree (tree*  
*subs (f n))*  
 $\rrbracket$   
 $\implies ? v . EV (\text{contains } f (0, \text{instanceF } v \ \text{body}))$   
*<proof>*

**lemma** *evContainsEx-instance*:  
 $\llbracket EV (\text{contains } f (i, FAll \text{ Neg } \text{body}));$  *branch subs gamma f; !n . ~ proofTree (tree*  
*subs (f n))*  
 $\rrbracket$   
 $\implies EV (\text{contains } f (0, \text{instanceF } (X \ i) \ \text{body}))$   
*<proof>*

**lemma** *evContainsEx-repeat*:  
 $\llbracket$  *branch subs gamma f; !n . ~ proofTree (tree subs (f n));*  
 $EV (\text{contains } f (i, FAll \text{ Neg } \text{body})) \rrbracket$   
 $\implies EV (\text{contains } f (Suc \ i, FAll \text{ Neg } \text{body}))$   
*<proof>*

## 5.28 EV contains: lemmas (temporal related)

**lemma** *lemma1*:  $\llbracket P \ A \ n \ ; \ !A \ n . P \ A \ n \ \dashrightarrow \ P \ A \ (Suc \ n) \rrbracket$   
 $\implies P \ A \ (n + m)$   
*<proof>*

**lemma** *lemma2*:  
 $\llbracket P \ A \ n \ ; \ P \ B \ m \ ; \ !A \ n . P \ A \ n \ \dashrightarrow \ P \ A \ (Suc \ n) \rrbracket$   
 $\implies ? n . P \ A \ n \ \& \ P \ B \ n$   
*<proof>*

## 5.29 EV contains: FAtoms

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

**lemma** *notTerminalNforms*:  $\neg \text{terminal } \text{subs } (f \ n) \implies \text{nforms } (f \ n) \neq \llbracket$   
*<proof>*

**lemma** *atomsPropagate*:  $\llbracket$  *branch subs gamma f*  $\rrbracket$   
 $\implies x : \text{set } (\text{atoms } (f \ n)) \ \dashrightarrow \ x : \text{set } (\text{atoms } (f \ (Suc \ n)))$   
*<proof>*

### 5.30 EV contains: FEx cases

**lemma** *evContainsEx0-allRepeats*:

$\llbracket \text{branch subs } \gamma f; !n . \sim \text{proofTree } (\text{tree subs } (f n));$   
 $\text{EV } (\text{contains } f (0, \text{FAll Neg } A)) \rrbracket$   
 $\implies \text{EV } (\text{contains } f (i, \text{FAll Neg } A))$   
 $\langle \text{proof} \rangle$

**lemma** *evContainsEx0-allInstances*:

$\llbracket \text{branch subs } \gamma f; !n . \sim \text{proofTree } (\text{tree subs } (f n));$   
 $\text{EV } (\text{contains } f (0, \text{FAll Neg } A)) \rrbracket$   
 $\implies \text{EV } (\text{contains } f (0, \text{instanceF } (X i) A))$   
 $\langle \text{proof} \rangle$

### 5.31 pseq: creates initial pseq

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

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

**lemma** *claim*:  $(A \mid B \mid C) = (\sim C \dashrightarrow \sim B \dashrightarrow A) \langle \text{proof} \rangle$

**lemma** *natPredCases*:  $(!n . P n) \mid (\sim P 0) \mid (? n . P n \ \& \ \sim P (Suc n))$   
 $\langle \text{proof} \rangle$

**lemma** *containsNotTerminal'*:

$\llbracket \text{branch subs } \gamma f; !n . \sim \text{proofTree } (\text{tree subs } (f n)); \text{contains } f i A n \rrbracket \implies$   
 $\sim (\text{terminal subs } (f n))$   
 $\langle \text{proof} \rangle$

**lemma** *notTerminalSucNotTerminal*:  $\llbracket \neg \text{terminal subs } (f (Suc n)); \text{branch subs } \gamma f \rrbracket \implies \neg \text{terminal subs } (f n)$   
 $\langle \text{proof} \rangle$

**lemma** *evContainsExSuc-containsEx*:

$\llbracket \text{branch subs } (\text{pseq fs}) f; !n . \sim \text{proofTree } (\text{tree subs } (f n));$   
 $\text{EV } (\text{contains } f (Suc i, \text{FAll Neg } \text{body})) \rrbracket$   
 $\implies \text{EV } (\text{contains } f (i, \text{FAll Neg } \text{body}))$   
 $\langle \text{proof} \rangle$

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

**lemma** *evContainsEx-containsEx0*:

$\llbracket \text{branch subs } (\text{pseq fs}) f; !n . \sim \text{proofTree } (\text{tree subs } (f n)) \rrbracket$   
 $\implies \text{EV } (\text{contains } f (i, \text{FAll Neg } A)) \dashrightarrow$   
 $\text{EV } (\text{contains } f (0, \text{FAll Neg } A))$   
 $\langle \text{proof} \rangle$

**lemma** *evContainsExval*:

$\llbracket EV (\text{contains } f (i, \text{Fall Neg body})); \text{branch subs } (pseq fs) f; !n . \sim \text{proofTree } (tree\ \text{subs } (f\ n)) \rrbracket$   
 $\implies ! v . EV (\text{contains } f (0, \text{instanceF } v\ \text{body}))$   
 $\langle \text{proof} \rangle$

### 5.34 EV contains: atoms

**lemma** *atomsInSequentI*[*rule-format*]:  $(z, P, vs) : \text{set } (fst\ ps) \dashrightarrow$   
 $FAtom\ z\ P\ vs : \text{set } (sequent\ ps)$   
 $\langle \text{proof} \rangle$

**lemma** *evContainsAtom1*:

$\llbracket \text{branch subs } (pseq fs) f; !n . \sim \text{proofTree } (tree\ \text{subs } (f\ n));$   
 $EV (\text{contains } f (i, FAtom\ z\ P\ vs)) \rrbracket$   
 $\implies ? n . (z, P, vs) : \text{set } (fst\ (f\ n))$   
 $\langle \text{proof} \rangle$

**lemmas** *atomsPropagate''* = *atomsPropagate*[*rule-format*]

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

**lemma** *evContainsAtom*:

$\llbracket \text{branch subs } (pseq fs) f; !n . \sim \text{proofTree } (tree\ \text{subs } (f\ n));$   
 $EV (\text{contains } f (i, FAtom\ z\ P\ vs)) \rrbracket$   
 $\implies ? n . (! m . FAtom\ z\ P\ vs : \text{set } (sequent\ (f\ (n + m))))$   
 $\langle \text{proof} \rangle$

**lemma** *notEvContainsBothAtoms*:

$\llbracket \text{branch subs } (pseq fs) f; !n . \sim \text{proofTree } (tree\ \text{subs } (f\ n)) \rrbracket$   
 $\implies \sim EV (\text{contains } f (i, FAtom\ Pos\ p\ vs)) \mid$   
 $\sim EV (\text{contains } f (j, FAtom\ Neg\ p\ vs))$   
 $\langle \text{proof} \rangle$

### 5.35 counterModel: lemmas

**lemma** *counterModelInRepr*:  $(\text{counterM } f, \text{counterEvalP } f) : \text{model}$   
 $\langle \text{proof} \rangle$

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

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

**lemma** *map-X-map-counterAssign*:  $\text{map } X (\text{map } (inv\ obj) (\text{map } \text{counterAssign } xs)) = xs$   
 $\langle \text{proof} \rangle$

**lemma** *objectsCounterModel*:  $\text{objects } (\text{counterModel } f) = \{ z . ? i . z = obj\ i \}$

*<proof>*

**lemma** *inCounterM*: *counterAssign v : objects (counterModel f)*  
*<proof>*

**lemma** *counterAssign-eqI*[*rule-format*]: *x : objects (counterModel f) --> z = X*  
*(inv obj x) --> counterAssign z = x*  
*<proof>*

**lemma** *evalPCounterModel*: *M = counterModel f ==> evalP M = counterEvalP f*  
*<proof>*

### 5.36 counterModel: all path formula value false - step by step

**lemma** *path-evalF'*:

**notes** *ss = evalPCounterModel counterEvalP-def map-X-map-counterAssign map-map[symmetric]*  
**and** *ss1 = instanceF-def evalF-subF-eq comp-vblcase id-def[symmetric]*  
**shows**  $[[ \text{branch subs (pseq fs) } f ;$   
 $!n . \sim \text{proofTree (tree subs (f n))}$   
 $]] ==> (? i . EV (\text{contains } f (i,A))) \longrightarrow \sim(\text{evalF (counterModel f) counterAssign A})$   
*<proof>*

**lemmas** *path-evalF'' = mp[OF path-evalF']*

### 5.37 adequacy

**lemma** *counterAssignModelAssign*: *counterAssign : modelAssigns (counterModel gamma)*  
*<proof>*

**lemma** *branch-contains-initially*: *branch subs (pseq fs) f ==> x : set fs ==> contains f (0,x) 0*  
*<proof>*

**lemma** *path-evalF*:

$[[ \text{branch subs (pseq fs) } f ;$   
 $\forall n . \neg \text{proofTree (tree subs (f n))};$   
 $x \in \text{set fs}$   
 $]] ==> \neg \text{evalF (counterModel f) counterAssign } x$   
*<proof>*

**lemma** *validProofTree*:  $\sim \text{proofTree (tree subs (pseq fs))} ==> \sim(\text{validS fs})$   
*<proof>*

**lemma** *adequacy[simplified sequent-pseq]*: *validS fs ==> (sequent (pseq fs)) : deductions CutFreePC*

*<proof>*

**end**

## 6 Soundness

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

**lemma** *permutation-validS*:  $fs <\sim\sim> gs \longrightarrow (validS\ fs = validS\ gs)$

*<proof>*

**lemma** *modelAssigns-vblcase*:  $\phi \in modelAssigns\ M \implies x \in objects\ M \implies vblcase\ x\ \phi \in modelAssigns\ M$

*<proof>*

**lemma** *tmp*:  $(!x : A. P\ x \mid Q) \implies (!x : A. P\ x) \mid Q$  *<proof>*

**lemma** *soundnessFAll*:  $!!Gamma.$

$[[\ u \sim : freeVarsFL\ (FAll\ Pos\ A\ \# \ Gamma);$

$validS\ (instanceF\ u\ A\ \# \ Gamma)\ ]]$

$\implies validS\ (FAll\ Pos\ A\ \# \ Gamma)$

*<proof>*

**lemma** *soundnessFEx*:  $validS\ (instanceF\ x\ A\ \# \ Gamma) \implies validS\ (FAll\ Neg\ A\ \# \ Gamma)$

*<proof>*

**lemma** *soundnessFCut*:  $[[\ validS\ (C\ \# \ Gamma); validS\ (FNot\ C\ \# \ Delta)\ ]]$   
 $\implies validS\ (Gamma\ @\ Delta)$

*<proof>*

**lemma** *soundness*:  $fs : deductions(PC) \implies (validS\ fs)$

*<proof>*

**lemma** *completeness*:  $fs : deductions\ (PC) = validS\ fs$

*<proof>*

**end**