∀(s1 :α -> bool) (s2 :α -> bool). s1 ⊂ s2 ⇔ s1 ⊆ s2 ∧ ¬(s2 ⊆ s1)
∀(p :num -> bool). (∃(n :num). p n) ⇔ p ($LEAST p) ∧ ∀(n :num). n < $LEAST p ⇒ ¬p n
∀(f :β -> γ) (g :α -> β) (l :α list). MAP f (MAP g l) = MAP (f ∘ g) l
(x :α) ∈ RDOM ((R :α -> β -> bool) \\ (k :α)) ⇔ x ∈ RDOM R ∧ x ≠ k
WF (R :α -> α -> bool) ⇒ antisymmetric R
∀(P :α option -> bool). (∀(a :α). P (SOME a)) ∧ P (NONE :α option) ⇒ ∀(x :α option). P x
PROD_SET (∅ :num -> bool) = (1 :num)
(p :bool) ∧ (q :bool) ⇒ p
∀(R :α -> α -> bool) (P :α -> bool). (∀(x :α). (∀(y :α). R y x ⇒ P y) ⇒ P x) ⇒ ∀(x :α). WFP R x ⇒ P x
∀(t :bool). t ∧ T ⇔ t
∀(s :α -> bool) (t :α -> bool) (x :α). x ∈ s DIFF t ⇔ x ∈ s ∧ x ∉ t
∀(s :α -> bool) (t :α -> bool) (x :α). (s DELETE x) ∩ t = s ∩ t DELETE x
∀(a :α -> bool) (b :α -> bool) (c :β -> bool) (d :β -> bool). a ⊆ b ∧ c ⊆ d ⇒ a × c ⊆ b × d
∀(s :α -> bool) (n :num). s HAS_SIZE n ⇒ CARD s = n
∀(f :α -> β) (x :α option) (y :β). OPTION_MAP f x = SOME y ⇔ ∃(z :α). x = SOME z ∧ y = f z
∀(x :num -> α -> bool) (y :num -> α -> bool). mk_rec x = mk_rec y ⇒ ZRECSPACE x ∧ ZRECSPACE y ⇒ x = y
((p :bool) ⇔ (q :bool) ∨ (r :bool)) ⇔ (p ∨ ¬q) ∧ (p ∨ ¬r) ∧ (q ∨ r ∨ ¬p)
∀(s :α -> bool). (∃(f :num -> α). BIJ f 𝕌(:num) s) ⇔ BIJ (enumerate s) 𝕌(:num) s
LENGTH (TAKE (n :num) (xs :α list)) = if n ≤ LENGTH xs then n else LENGTH xs
(∀(x :α) (y :α) (z :α). (R :α -> α -> bool) x y ∧ R y z ⇒ R x z) ⇒ ∀(x :α list) (y :α list) (z :α list). LIST_REL R x y ∧ LIST_REL R y z ⇒ LIST_REL R x z
∀(h1 :α) (h2 :α). h1 = h2 ⇒ ∀(l1 :α list) (l2 :α list). l1 = l2 ⇒ h1::l1 = h2::l2
($SUBSET :(α -> bool) -> (α -> bool) -> bool)✡ = ($SUBSET :(α -> bool) -> (α -> bool) -> bool)
(∀(x :α). x ∈ (s :α -> bool)) ⇔ s = 𝕌(:α)
∀(n :num). count (SUC n) = n INSERT count n
∀(xs :α list) (x :α) (y :α) (ys :α list). LUPDATE x (LENGTH xs) (xs ++ y::ys) = xs ++ x::ys
∀(f :α -> β) (s :β -> bool) (t :β -> bool). PREIMAGE f (s ∪ t) = PREIMAGE f s ∪ PREIMAGE f t
∀(l :α list) (m :num). m = LENGTH l ⇒ TAKE m l = l
∀(s :α -> bool). s ⊆ s
(f :δ -> γ) ∘ UNCURRY (g :α -> β -> δ) = UNCURRY (($o f :(β -> δ) -> β -> γ) ∘ g)
∀(t :bool). F ⇒ t ⇔ T
∀(s :num -> bool). s ≠ (∅ :num -> bool) ∧ FINITE s ⇒ MIN_SET s ≤ MAX_SET s
∀(s :α -> bool). SURJ (λ(x :α). x) s s
∀(x :α -> bool). BIGUNION {x} = x
∀(f :β -> γ) (g :α -> β) (s :α -> bool). IMAGE (f ∘ g) s = IMAGE f (IMAGE g s)
∀(R1 :α -> β -> bool) (R2 :α -> β -> bool). R1 $\subseteq_r$ R2 ∧ R2 $\subseteq_r$ R1 ⇒ R1 = R2
$! (UNCURRY (λ(x :α). (P :α -> β -> bool) x)) ⇔ ∀(x :α). $! (P x)
∀(x :α) (y :α) (s :α -> bool). (y INSERT s) x ⇔ x = y ∨ x ∈ s
∀(l :α list) (x :α). BUTLASTN (LENGTH l) (x::l) = [x]
SING ((x :α) INSERT (s :α -> bool)) ⇔ s = (∅ :α -> bool) ∨ s = {x}

¬LLEX (R :α -> α -> bool) (l :α list) ([] :α list)
∀(P :α -> β -> bool) (Q :α -> β -> bool) (l1 :α list) (l2 :β list). (∀(x :α # β). MEM x (ZIP (l1,l2)) ∧ UNCURRY P x ⇒ UNCURRY Q x) ∧ LIST_REL P l1 l2 ⇒ LIST_REL Q l1 l2
∀(P :α -> bool) (l :α list). EXISTS P (REVERSE l) ⇔ EXISTS P l
(l1 :α list) ++ (l2 :α list) = (h :α)::(t :α list) ⇔ l1 = ([] :α list) ∧ l2 = h::t ∨ ∃(lt :α list). l1 = h::lt ∧ t = lt ++ l2
(([] :α list) ≼ (l :α list) ⇔ T) ∧ ((h :α)::(t :α list) ≼ ([] :α list) ⇔ F) ∧ ((h1 :α)::(t1 :α list) ≼ (h2 :α)::(t2 :α list) ⇔ h1 = h2 ∧ t1 ≼ t2)
¬((A :bool) ∨ (B :bool)) ⇒ F ⇔ ¬A ⇒ ¬B ⇒ F
∀(f :α -> bool) (l :α list). FILTER f (FILTER f l) = FILTER f l
∀(P :α -> bool) (l :α list). EXISTS P l ⇔ FOLDR $\/ F (MAP P l)
∀(R :α -> α -> bool). irreflexive R ∧ transitive R ⇒ antisymmetric R
(∀(s :α -> bool) (t :α -> bool). s ⊆ s ∪ t) ∧ ∀(s :α -> bool) (t :α -> bool). s ⊆ t ∪ s
countable 𝕌(:num)
∀(a :α list) (b :α list) (c :α list). a ++ b ≼ c ⇒ a ≼ c
∀(r1 :α -> β option) (r2 :α -> β option) (rows1 :(α -> β option) list) (rows2 :(α -> β option) list) (rows3 :(α -> β option) list) (v :α). (IS_SOME (r2 v) ⇒ IS_SOME (r1 v)) ⇒ PMATCH v (rows1 ++ r1::rows2 ++ r2::rows3) = PMATCH v (rows1 ++ r1::rows2 ++ rows3)
INVOL (relinv :(α -> α -> bool) -> α -> α -> bool)
∀(p :α -> β) (g :α -> bool) (i :β). (∀(x1 :α) (x2 :α). g x1 ∧ g x2 ∧ p x1 = p x2 ⇒ x1 = x2) ⇒ ∀(x :α). PMATCH_ROW_COND p g i x ⇒ (@(y :α). PMATCH_ROW_COND p g i y) = x
∀(f :num -> α) (s :num -> bool). countable (IMAGE f s)
∀(f :α -> β -> α) (e :α) (l :β list). LENGTH (SCANL f e l) = SUC (LENGTH l)
∀(x :α) (s :α -> bool). REST s x ⇔ x ∈ s ∧ x ≠ CHOICE s
(∀(x :α) (y :α). (R1 :α -> α -> bool) x y ⇒ (R2 :α -> α -> bool) x y) ∧ (∀(x :β) (y :β). (R3 :β -> β -> bool) x y ⇒ (R4 :β -> β -> bool) x y) ⇒ (R1 LEX R3) (x :α # β) (y :α # β) ⇒ (R2 LEX R4) x y
∀(s :α -> bool). FINITE s ⇒ ∀(t :α -> bool). t ⊂ s ⇒ FINITE t
∀(r :α -> bool) (s :α -> bool) (t :α -> bool). r ⊆ s ⇒ r DIFF s ∩ t = r DIFF t
∀(R :β -> β -> bool) (f :α -> β) (x :α) (y :α). inv_image R f x y ⇔ R (f x) (f y)
COMPL ((s :α -> bool) ∪ (t :α -> bool)) = COMPL s ∩ COMPL t
∀(s :α -> bool) (x :α) (y :α). (s DELETE y) x ⇔ x ∈ s ∧ x ≠ y
∀(y :β) (s :α -> bool) (f :α -> β). y ∈ IMAGE f s ⇔ ∃(x :α). y = f x ∧ x ∈ s
(x :α) ∈ RDOM (R :α -> β -> bool) ⇔ ∃(y :β). R x y
∀(R :α -> α -> bool) (x :α) (y :α). R^{*} x y ⇔ x = y ∨ R^{+} x y
∀(l1 :α list) (l2 :α list) (P :α -> bool) (P' :α -> bool). l1 = l2 ∧ (∀(x :α). MEM x l2 ⇒ (P x ⇔ P' x)) ⇒ (EVERY P l1 ⇔ EVERY P' l2)
∀(m :num) (n :num). m < n ⇒ EL m (COUNT_LIST n) = m
∀(c :bool) (x :bool). (if c then T else x) ⇔ ¬c ⇒ x
TAKE (0 :num) (l :α list) = ([] :α list)
∀(f :α -> β -> β) (e :β) (x :α). FOLDR f e [x] = f x e
∀(l :α list). DROP (LENGTH l) l = ([] :α list)
∀(e :α) (s1 :α -> bool) (s2 :α -> bool). s1 ⊆ s2 ⇒ s1 ⊆ e INSERT s2
∀(P :α list -> bool). P ([] :α list) ∧ (∀(t :α list). P t ⇒ ∀(h :α). P (h::t)) ⇒ ∀(l :α list). P l

¬T ⇔ F	
(R :α -> α -> bool)$^+$ (x :α) (z :α) ⇔ R x z ∨ ∃(y :α). R$^+$ x y ∧ R y z	
∀(s :α -> bool) (t :α -> bool) (u :α -> bool). s ⊂ t ∧ t ⊆ u ⇒ s ⊂ u	
RC ($PSUBSET :(α -> bool) -> (α -> bool) -> bool) = ($SUBSET :(α -> bool) -> (α -> bool) -> bool)	
∀(x :bool) (x' :bool) (y :bool) (y' :bool). (¬y ⇒ x ⇒ x') ∧ (¬x' ⇒ y ⇒ y') ⇒ x ∨ y ⇒ x' ∨ y'	
∀(e :α) (s :α -> bool). POW (e INSERT s) = IMAGE ($INSERT e) (POW s) ∪ POW s	
∀(s :α -> bool). s DIFF s = (∅ :α -> bool)	
∀(x :bool) (x' :bool) (y :bool) (y' :bool). (x' ⇒ x) ∧ (x' ⇒ y ⇒ y') ⇒ (x ⇒ y) ⇒ x' ⇒ y'	
∀(x :α) (l :α list). REVERSE (SNOC x l) = x::REVERSE l	
(∀(x :α). (P :α -> bool) x ⇒ (P' :α -> bool) x) ⇒ OPTION_ALL P (opt :α option) ⇒ OPTION_ALL P' opt	
∀(x :α) (P :α -> bool). x ∈ (λ(x :α). P x) ⇔ P x	
(tri (n :num) = (0 :num) ⇔ n = (0 :num)) ∧ ((0 :num) = tri n ⇔ n = (0 :num))	
REVERSE ([] :β list) = ([] :β list) ∧ ∀(x :α) (l :α list). REVERSE (x::l) = SNOC x (REVERSE l)	
∀(l1 :α list) (l2 :α list). ¬NULL l1 ⇒ ELL (LENGTH l2) (l1 ++ l2) = LAST l1	
∀(x :α) (l :α list). LENGTH (SNOC x l) = SUC (LENGTH l)	
(∀(f :α -> β -> γ). MAP2 f ([] :α list) ([] :β list) = ([] :γ list)) ∧ ∀(f :δ -> ε -> ζ) (h1 :δ) (t1 :δ list) (h2 :ε) (t2 :ε list). MAP2 f (h1::t1) (h2::t2) = f h1 h2::MAP2 f t1 t2	
∀(s :α -> bool) (t :β -> bool) (f :α -> β) (x :α). f ∈ FUNSET s t ∧ x ∈ s ⇒ f x ∈ t	
∀(s :α -> bool) (t :α -> bool). s ⊆ t ⇔ s ∪ t = t	
RC (R :α -> α -> bool) = R $\cup_r$ ($= :α -> α -> bool)	
∀(f :α -> β -> γ). UNCURRY f = (λ(x :α # β). f (FST x) (SND x))	
∀(l :bool list). OR_EL l ⇔ FOLDR $\/ F l	
∀(l1 :α list) (l2 :α list). LENGTH l1 = LENGTH l2 ∧ (∀(x :num). x < LENGTH l1 ⇒ EL x l1 = EL x l2) ⇒ l1 = l2	
∀(x :α) (s :α -> bool). x ∈ s ⇒ x INSERT s = s	
∀(P :num -> bool) (Q :num -> bool). FINITE P ∧ (P = (∅ :num -> bool) ⇒ Q (0 :num)) ∧ (∀(x :num). (∀(y :num). y ∈ P ⇒ y ≤ x) ∧ x ∈ P ⇒ Q x) ⇒ Q (MAX_SET P)	
∀(e :α) (l :α list). MEM (LAST (e::l)) (e::l)	
∀(R :α -> α -> bool). $R^{✡\,✡}$ = $R^{✡}$	
¬((p :bool) ⇒ (q :bool)) ⇒ ¬q	
∀(n :num) (ls :α list). n < LENGTH ls ⇒ LUPDATE (EL n ls) n ls = ls	
∀(y :bool) (x :bool). ASM_MARKER y x ⇔ x	
∀(x :num) (l :num list). SUM (SNOC x l) = SUM l + x	
∀(R :α -> α -> bool). StrongOrder R ⇒ STRORD (RC R) = R	
∀(l1 :α list) (l2 :α list) (l3 :α list). (∀(n :num). n < LENGTH l1 ∧ (R :α -> α -> bool) (EL n l1) (EL n l2) ∧ R (EL n l2) (EL n l3) ⇒ R (EL n l1) (EL n l3)) ∧ LIST_REL R l1 l2 ∧ LIST_REL R l2 l3 ⇒ LIST_REL R l1 l3	
∀(P :α -> β) (v :γ). P (PMATCH v ([] :(γ -> α option) list)) = P (ARB :α)	
∀(s :α -> bool) (t :α -> bool). INFINITE s ∧ FINITE t ⇒ ∃(x :α). x ∈ s ∧ x ∉ t	
∀(v :α) (rows :(α -> β option) list) (n :num). n < LENGTH rows ∧ IS_SOME (EL n rows v) ⇒ PMATCH v rows = PMATCH v (TAKE (SUC n) rows)	

∀(f :α -> β) (ls :α list). ALL_DISTINCT (MAP f ls) ⇒ ALL_DISTINCT ls
(∀(x :α). LAST [x] = x) ∧ ∀(h1 :β) (h2 :β) (t :β list). LAST (h1::h2::t) = LAST (h2::t)
∀(s :α -> bool). FINITE (REST s) ⇔ FINITE s
∀(x :bool) (x' :bool) (y :bool) (y' :bool). (¬y ⇒ x' ⇒ x) ∧ (¬x' ⇒ y' ⇒ y) ⇒ x' ∨ y' ⇒ x ∨ y
PMATCH_ROW_COND (pat :α -> β) (guard :α -> bool) (inp :β) (v :α) ⇔ inp = pat v ∧ guard v
∀(l :α list). NULL l ⇔ FOLDR (λ(x :α) (l' :bool). F) T l
(0 :num) < (n :num) ⇒ TAKE n ((x :α)::(xs :α list)) = x::TAKE (n − (1 :num)) xs
COUNT_LIST (0 :num) = ([] :num list) ∧ ∀(n :num). COUNT_LIST (SUC n) = SNOC n (COUNT_LIST n)
∀(x :α) (l :α list). ([] :α list) ≠ SNOC x l
IDEM (RTC :(α -> α -> bool) -> α -> α -> bool)
∀(v_old :α) (v_new :β). PMATCH v_old ([] :(α -> γ option) list) = PMATCH v_new ([] :(β -> γ option) list)
OPTION_GUARD (b :bool) = if b then SOME () else (NONE :unit option)
(∀(opt :α option). (P :α option -> bool) opt) ⇔ P (NONE :α option) ∧ ∀(x :α). P (SOME x)
∀(x :α) (y :α) (s :α -> bool). x INSERT y INSERT s = y INSERT x INSERT s
∀(f :α -> β) (s :α -> bool) (t :β -> bool). BIJ f s t ∧ FINITE s ⇒ FINITE t
∀(s :α -> bool). (∅ :α -> bool) DIFF s = (∅ :α -> bool)
∀(M :α list) (N :α list) (f :α -> num) (f' :α -> num). M = N ∧ (∀(x :α). MEM x N ⇒ f x = f' x) ⇒ list_size f M = list_size f' N
∀(m :num) (l :α list). m ≤ LENGTH l ⇒ ∀(n :num). n ≤ m ⇒ TAKE n (TAKE m l) = TAKE n l
∀(s :α -> bool) (t :α -> bool). s ⊆ t ⇒ ∀(f :α -> β). IMAGE f s ⊆ IMAGE f t
∀(s :α -> bool) (t :β -> bool). (∃(f :α -> β). SURJ f s t) ⇒ ∃(g :β -> α). INJ g t s
∀(f :num -> α) (n :num). LENGTH (GENLIST f n) = n
IDEM (TC :(α -> α -> bool) -> α -> α -> bool)
∀(f :α -> α -> α). ASSOC f ⇒ ∀(e :α). LEFT_ID f e ⇒ ∀(l :α list list). FOLDR f e (FLAT l) = FOLDR f e (MAP (FOLDR f e) l)
∀(l :α list). IS_SUFFIX l l
∀(h1 :α) (h2 :α). h1 ≠ h2 ⇒ ∀(l1 :α list) (l2 :α list). h1::l1 ≠ h2::l2
∀(P :β -> bool) (f :α -> β) (l :α list). EVERY P (MAP f l) ⇔ EVERY (λ(x :α). P (f x)) l
∀(l1 :α list) (l2 :α list) (n :num) (x :α). LENGTH l1 ≤ n ⇒ LUPDATE x n (l1 ++ l2) = l1 ++ LUPDATE x (n − LENGTH l1) l2
{x | (P :α -> bool) x} = P
∀(x :α) (s :α -> bool) (t :α -> bool). x INSERT s ⊆ t ⇔ x ∈ t ∧ s ⊆ t
∀(t :bool). t ⇒ T
(∀(x :α) (y :β). (R1 :α -> β -> bool) x y ⇒ (R2 :β -> α -> bool) y x) ⇒ ∀(x :α list) (y :β list). LIST_REL R1 x y ⇒ LIST_REL R2 y x
∀(l :α list). ¬NULL l ⇒ ELL (0 :num) l = LAST l
∀(v :α) (rows :(α -> β option) list) (i :num). i < LENGTH rows ⇒ (EL i (STRONGEST_REDUNDANT_ROWS_INFO v rows) ⇔ EVERY (λ(r :α -> β option). r v = (NONE :β option)) (TAKE i rows) ⇒ EL i rows v = (NONE :β option))
UNCURRY (f :α -> β -> γ) = UNCURRY (g :α -> β -> γ) ⇔ f = g

PMATCH_ROW_COND_NOT_EX_OR_EQ (i :α) (r :α -> β option) ([] :(α -> β option) list) ⇔ r i ≠ (NONE :β option) ⇒ F
∀(s :α -> bool) (t :α -> bool) (u :α -> bool). s ⊆ t ∩ u ⇔ s ⊆ t ∧ s ⊆ u
∀(P :α -> bool) (v :β) (rows :(β -> α option) list). P (PMATCH v rows) ⇔ PMATCH_EXPAND_PRED P v ([] :(β -> α option) list) rows
nsnd (0 :num) = (0 :num)
∀(l :α list) (m :num). m = LENGTH l ⇒ DROP m l = ([] :α list)
∀(x :α) (s :α -> bool). x ∈ s ∧ (∀(y :α). y ∈ s ⇒ x = y) ⇔ s = {x}
(∃(p :α # β). (P :α -> β -> bool) (FST p) (SND p)) ⇔ ∃(p1 :α) (p2 :β). P p1 p2
([] :α list) ∈ common_prefixes (s :α list -> bool)
∀(P :β -> bool) (f :α -> β) (l :α list). EXISTS P (MAP f l) ⇔ EXISTS (λ(x :α). P (f x)) l
∀(x :α). (NONE :α option) ≠ SOME x
∀(f :α -> β) (R :α -> α -> bool) (y :α) (z :α). R y z ⇒ RESTRICT f R z y = f y
∀(x :α) (y :α). SOME x = SOME y ⇔ x = y
∀(s :α -> bool). DISJOINT (∅ :α -> bool) s ∧ DISJOINT s (∅ :α -> bool)
(∀(s :α -> bool) (t :α -> bool). s ∪ (t DIFF s) = s ∪ t) ∧ ∀(s :α -> bool) (t :α -> bool). t DIFF s ∪ s = t ∪ s
DATATYPE ((option :α option -> (α -> α option) -> β) (NONE :α option) (SOME :α -> α option))
∀(s :α -> bool) (t :α -> bool). s ⊆ t ∧ t ⊆ s ⇒ s = t
∀(s :α -> bool) (t :α -> bool) (u :α -> bool). s ∪ t ∩ u = (s ∪ t) ∩ (s ∪ u)
∀(l :α list). (0 :num) < LENGTH l ⇔ ¬NULL l
∀(P :α -> bool) (Q :α -> bool). (∀(x :α). P x ⇒ Q x) ⇒ ∀(l :α list). EVERY P l ⇒ EVERY Q l
∀(p :num -> bool). (∃(n :num). p n) ⇒ p ($LEAST p) ∧ ∀(n :num). n < $LEAST p ⇒ ¬p n
∀(s :α -> bool) (t :α -> bool) (u :α -> bool). (DISJOINT (s ∪ t) u ⇔ DISJOINT s u ∧ DISJOINT t u) ∧ (DISJOINT u (s ∪ t) ⇔ DISJOINT s u ∧ DISJOINT t u)
∀(xs :α list) (h :α) (i :num). (P :α -> bool) h ∧ EVERY P xs ⇒ EVERY P (LUPDATE h i xs)
∀(P :(α -> bool) -> bool). (BIGUNION P = (∅ :α -> bool) ⇔ P = (∅ :(α -> bool) -> bool) ∨ P = {(∅ :α -> bool)}) ∧ ((∅ :α -> bool) = BIGUNION P ⇔ P = (∅ :(α -> bool) -> bool) ∨ P = {(∅ :α -> bool)})
(x :α -> bool) ∩ COMPL x = (∅ :α -> bool) ∧ COMPL x ∩ x = (∅ :α -> bool)
TAKE (n :num) (l :α list) = ([] :α list) ⇔ n = (0 :num) ∨ l = ([] :α list)
APPLY_REDUNDANT_ROWS_INFO ([] :bool list) ([] :α list) = ([] :α list) ∧ (∀(is :bool list) (x :β) (xs :β list). APPLY_REDUNDANT_ROWS_INFO (T::is) (x::xs) = APPLY_REDUNDANT_ROWS_INFO is xs) ∧ ∀(is :bool list) (x :γ) (xs :γ list). APPLY_REDUNDANT_ROWS_INFO (F::is) (x::xs) = x::APPLY_REDUNDANT_ROWS_INFO is xs
∀(s :α -> bool) (t :α -> bool) (u :α -> bool). s ∩ (t ∪ u) = s ∩ t ∪ s ∩ u
∀(n :num) (f :num -> α) (m :num). n < m ⇒ EL n (MAP f (COUNT_LIST m)) = f n
∀(x :α + β). ¬ISR x ⇔ ISL x
∀(R :α -> α -> bool) (x :α) (y :α). R x y ⇒ R^= x y
∀(l :α list) (n :num). LENGTH l ≤ n ⇒ TAKE n l = l
∀(R :α -> β -> bool). $R^{T\,T}$ = R
∀(R :α -> α -> bool). $R^{+\,+}$ = R^{+}

∀(x :α) (f :α -> num -> α). PRIM_REC x f (0 :num) = x ∧ ∀(m :num). PRIM_REC x f (SUC m) = f (PRIM_REC x f m) m
∀(l1 :α list) (l2 :α list) (l :α list). IS_SUFFIX l2 l ⇒ IS_SUFFIX (l1 ++ l2) l
∀(x :α) (f :β -> α -> bool) (s :β -> bool). x ∈ BIGINTER (IMAGE f s) ⇔ ∀(y :β). y ∈ s ⇒ x ∈ f y
∀(x :α). x INSERT 𝕌(:α) = 𝕌(:α)
OPTION_MAP2 (f :α -> β -> γ) (o1 :α option) (o2 :β option) = SOME (v :γ) ⇔ ∃(x1 :α) (x2 :β). o1 = SOME x1 ∧ o2 = SOME x2 ∧ v = f x1 x2
(∀(s :α). (P :α -> bool) s ⇔ (Q :α -> bool) s) ⇒ ((∃(s :α). P s) ⇔ ∃(s :α). Q s)
reflexive (R :α -> α -> bool)^=
∀(f :α -> β) (ls :α list) (a :β list). FOLDR (λ(x :α) (y :β list). f x::y) a ls = MAP f ls ++ a
INJ (f :α -> β) (s :α -> bool) (t :β -> bool) ⇔ (∀(x :α). x ∈ s ⇒ f x ∈ t) ∧ ∀(x :α) (y :α). x ∈ s ∧ y ∈ s ⇒ (f x = f y ⇔ x = y)
∀(s :α -> bool). IMAGE (λ(x :α). x) s = s
(R :α -> α -> bool)$^{✡}$ (x :α) x
∀(l :α list) (l' :α list) (b :β) (b' :β) (f :β -> α -> β) (f' :β -> α -> β). l = l' ∧ b = b' ∧ (∀(x :α) (a :β). MEM x l' ⇒ f a x = f' a x) ⇒ FOLDL f b l = FOLDL f' b' l'
∀(x :α) (f :α -> α) (g1 :num -> α) (g2 :num -> α) (m1 :num) (m2 :num). SIMP_REC_REL g1 x f m1 ∧ SIMP_REC_REL g2 x f m2 ⇒ ∀(n :num). n < m1 ∧ n < m2 ⇒ g1 n = g2 n
∀(P :α -> bool) (l :α list list). FILTER P (FLAT l) = FLAT (MAP (FILTER P) l)
∀(s :α list) (h :α) (t :α list). IS_SUFFIX s (h::t) ⇒ IS_SUFFIX s t
∀(P :α -> bool) (s :(α -> bool) -> bool). (∀(x :α). x ∈ BIGUNION s ⇒ P x) ⇔ ∀(t :α -> bool) (x :α). t ∈ s ∧ x ∈ t ⇒ P x
∀(t :bool). F ∧ t ⇔ F
∀(P :α -> bool) (l :α list). FILTER P l ≠ l ⇔ ∃(x :α). MEM x l ∧ ¬P x
(x :α list) ≠ ([] :α list) ⇔ (0 :num) < LENGTH x
∀(y :α) (x :α) (l :α list). MEM y (SNOC x l) ⇔ y = x ∨ MEM y l
SUM_SET (∅ :num -> bool) = (0 :num)
MAP (λ(x :α). x) (l :α list) = l ∧ MAP (I :α -> α) l = l
∀(s :α -> bool) (t :α -> bool). s ⊆ t ∧ countable t ⇒ countable s
{x | F} = (∅ :α -> bool)
∀(f :α -> β) (s :α -> bool). s ⊆ PREIMAGE f (IMAGE f s)
∀(x :α) (s :α -> bool) (t :α -> bool). DISJOINT t (x INSERT s) ⇔ DISJOINT t s ∧ x ∉ t
∀(x :α) (s :α -> bool) (t :α -> bool). (x INSERT s) ∪ t = if x ∈ t then s ∪ t else x INSERT s ∪ t
∀(R :α -> α -> bool). (∀(x :α). R$^{✡}$ x x) ∧ ∀(x :α) (y :α) (z :α). R$^{✡}$ x y ∧ R y z ⇒ R$^{✡}$ x z
∀(s :num -> bool) (t :num -> bool). FINITE t ∧ s ⊆ t ⇒ SUM_SET s ≤ SUM_SET t
∀(x :α list) (l :α list list). FLAT (SNOC x l) = FLAT l ++ x
∀(f :α -> β) (P :α -> bool) (Q :β -> bool). f ∈ FUNSET P Q ⇔ ∀(x :α). x ∈ P ⇒ f x ∈ Q
∀(v :α) (rows1 :(α -> β option) list) (rows2 :(α -> β option) list) (rows3 :(α -> β option) list). PMATCH_EQUIV_ROWS v rows1 rows2 ⇒ PMATCH_EQUIV_ROWS v rows2 rows3 ⇒ PMATCH_EQUIV_ROWS v rows1 rows3
∀(l1 :α list) (l2 :β list) (f :α -> β -> bool). LIST_REL f l1 l2 ⇔ LENGTH l1 = LENGTH l2 ∧ EVERY (UNCURRY f) (ZIP (l1,l2))
(∀(s :α + β). (P :α + β -> bool) s) ⇔ (∀(x :α). P (INL x :α + β)) ∧ ∀(y :β). P (INR y :α + β)

∀(P :α # β -> γ). (λ(p :α # β). P p) = (λ((p1 :α),(p2 :β)). P (p1,p2))
∀(n :num). count (n + (1 :num)) = n INSERT count n
∀(P :α -> bool) (l :α list). FILTER P l = FOLDR (λ(x :α) (l' :α list). if P x then x::l' else l') ([] :α list) l
∀(l :(α # β) list). LENGTH (UNZIP_FST l) = LENGTH l
∀(R :α -> α -> bool). WeakOrder R ⇒ ∀(y :α) (z :α). y = z ⇔ R y z ∧ R z y
∀(opt :α option). OPTION_BIND opt (SOME :α -> α option) = opt
((p :bool) ⇔ (q :bool) ∧ (r :bool)) ⇔ (p ∨ ¬q ∨ ¬r) ∧ (q ∨ ¬p) ∧ (r ∨ ¬p)
LIST_REL (λ(a :α) (b :β). (P :α -> β -> bool) a b ∧ (Q :α -> β -> bool) a b) (l1 :α list) (l2 :β list) ⇔ LIST_REL (λ(a :α) (b :β). P a b) l1 l2 ∧ LIST_REL (λ(a :α) (b :β). Q a b) l1 l2
∀(n :num) (m :num) (l :α list). TAKE (n + m) l = TAKE n l ++ TAKE m (DROP n l)
MAX (0 :num) (x :num) = x ∧ MAX x (0 :num) = x ∧ MAX (NUMERAL x) (NUMERAL (y :num)) = NUMERAL (if x < y then y else x)
∀(R :α -> α -> bool). transitive R ⇒ R^+ = R
(OPTION_GUARD (b :bool) = SOME () ⇔ b) ∧ (OPTION_GUARD b = (NONE :unit option) ⇔ ¬b)
∀(f :α -> β) (s :β -> bool). PREIMAGE f s = s ∘ f
∀(l1 :α list) (l2 :α list) (P :α -> bool) (P' :α -> bool). l1 = l2 ∧ (∀(x :α). MEM x l2 ⇒ (P x ⇔ P' x)) ⇒ (EXISTS P l1 ⇔ EXISTS P' l2)
flip (UNCURRY (f :α -> β -> δ -> γ)) (x :δ) = UNCURRY (flip ((flip :(β -> δ -> γ) -> δ -> β -> γ) ∘ f) x)
∀(P1 :α -> bool) (P2 :α -> bool) (l :α list). EVERY P1 l ⇒ EVERY P1 (FILTER P2 l)
∀(p :α -> β) (g :α -> bool) (r :α -> γ) (v :β) (rows :(β -> γ option) list). (∀(x :α). r x = (ARB :γ)) ⇒ PMATCH v (SNOC (PMATCH_ROW p g r) rows) = PMATCH v rows
∀(x :α) (l :α list). MEM x l ⇔ EXISTS ($= x) l
¬(¬(A :bool) ∨ (B :bool)) ⇒ F ⇔ A ⇒ ¬B ⇒ F
∀(R :α -> α -> bool). SC (RC R) = RC (SC R) ∧ RC (RC R) = RC R ∧ $(RC\ R)^+$ = RC R^+
∀(s :α -> bool). (∅ :α -> bool) ⊆ s
∀(l :α list). NULL l ⇔ LENGTH l = (0 :num)
∀(P :α # β -> bool). (∀(p_1 :α) (p_2 :β). P (p_1,p_2)) ⇒ ∀(p :α # β). P p
WF (R :α -> α -> bool) ⇒ R^+ (x :α) (y :α) ⇒ x ≠ y
∀(f :α -> β -> α) (e :α) (g :γ -> β) (l :γ list). FOLDL f e (MAP g l) = FOLDL (λ(x :α) (y :γ). f x (g y)) e l
∀(f :α -> β) (s :α -> bool) (t :β -> bool). INJ f s t ⇒ IMAGE f s ⊆ t
∀(opt :α option). opt = (NONE :α option) ∨ ∃(x :α). opt = SOME x
(∀(s :α -> bool). (∅ :α -> bool) ∩ s = (∅ :α -> bool)) ∧ ∀(s :α -> bool). s ∩ (∅ :α -> bool) = (∅ :α -> bool)
OPTION_MAP (I :α -> α) (x :α option) = x ∧ OPTION_MAP (λ(x :α). x) x = x
(pair_CASE (p :β # γ) (f :β -> γ -> α) :α) = (v :α) ⇔ ∃(x :β) (y :γ). p = (x,y) ∧ f x y = v
WF (R :α -> α -> bool) ⇒ irreflexive R
∀(s :α -> bool) (t :α -> bool). FINITE s ∧ t ⊆ s ⇒ FINITE t
∀(f :α -> β) (R :α -> α -> bool) (P :α -> β -> bool) (M :(α -> β) -> α -> β) (x :α). f = WFREC R M ∧ WF R ∧ INDUCTIVE_INVARIANT R P M ⇒ P x (f x)
∀(R :α -> α -> bool). symmetric (RC R) ⇔ symmetric R

∀(n :num) (x :α) (y :α). MEM y (REPLICATE n x) ⇔ x = y ∧ (0 :num) < n
∀(P :α -> bool) (ls :α list). NULL (FILTER P ls) ⇔ ∀(x :α). MEM x ls ⇒ ¬P x
SING (∅ :α -> bool) ⇔ F
¬((p :bool) ∨ (q :bool)) ⇒ ¬q
(([(f :β -> α)] <*> [(x :β)]) :α list) = [f x]
∀(f :α -> β) (s :β -> bool) (x :α). x ∈ PREIMAGE f s ⇔ f x ∈ s
∀(f :α -> β -> α) (e :α) (x :β). FOLDL f e [x] = f e x
∀(P :α -> bool) (Q :β -> bool) (x :α). (x INSERT P) × Q = {x} × Q ∪ P × Q
(n :num) = LENGTH (l1 :α list) ⇒ ZIP (l1,COUNT_LIST n) = GENLIST (λ(n :num). (EL n l1,n)) (LENGTH l1)
(∀(x :α). MEM x (ls :α list) ⇒ (R :α -> α -> bool) x x) ⇒ LIST_REL R ls ls
LAST ((h :α)::(t :α list)) = if t = ([] :α list) then h else LAST t
∀(s :α -> bool). s ⊆ (∅ :α -> bool) ⇔ s = (∅ :α -> bool)
SET_TO_LIST {(x :α)} = [x]
∀(s :α -> bool) (t :α -> bool) (x :α). (s DIFF t) x ⇔ x ∈ s ∧ x ∉ t
∀(n :num) (k :num). SUM (REPLICATE n k) = n * k
∀(P :α -> bool). INFINITE P ⇒ ∃(x :α). x ∈ P
(n :num) ≤ tri n
(∀(x :α) (y :α). (R :α -> α -> bool) x y ⇒ R ((f :α -> α) x) (f y)) ⇒ ∀(x :α) (y :α). R✡ x y ⇒ R✡ (f x) (f y)
∀(l :α list) (f :α -> β). LENGTH (MAP f l) = LENGTH l
INJ (f :α -> β) (s :α -> bool) (t :β -> bool) ⇒ ∀(x :α) (y :β). LINV_OPT f s y = SOME x ⇔ y = f x ∧ x ∈ s ∧ y ∈ t
MAP (λ((x :α),(y :β),(z :γ)). x) (funs :(α # β # γ) list) = MAP (FST :α # β # γ -> α) funs
∀(l :(α # β) list). LENGTH (UNZIP_SND l) = LENGTH l
Order (R :α -> α -> bool) ⇔ WeakOrder (RC R)
∀(l :α list) (l' :α list) (b :β) (b' :β) (f :α -> β -> β) (f' :α -> β -> β). l = l' ∧ b = b' ∧ (∀(x :α) (a :β). MEM x l' ⇒ f x a = f' x a) ⇒ FOLDR f b l = FOLDR f' b' l'
∀(f :α -> α) (s :α -> bool). (x :α) ∈ schroeder_close f s ⇔ ∃(n :num). x ∈ FUNPOW (IMAGE f) n s
∀(R :α -> α -> bool) (s :α -> bool) (t :α -> bool). t ∈ partition R s ⇒ t ⊆ s
∀(M :(α -> β) -> α -> β) (R :α -> α -> bool) (f :α -> β). f = WFREC R M ⇒ WF R ⇒ ∀(x :α). f x = M (RESTRICT f R x) x
∀(n :num). ¬(n < (0 :num))
∀(S :α -> bool). FINITE S ⇒ ∀(t :β -> bool) (f :α -> β). BIJ f S t ∧ FINITE t ⇒ CARD S = CARD t
((x :α),(y :β)) ∈ {(x,y) | (P :α -> β -> bool) x y} ⇔ P x y
∀(ls :α list) (f :β -> α # α -> β) (e :β). FOLDL f e (ZIP (ls,ls)) = FOLDL (λ(x :β) (y :α). f x (y,y)) e ls
∀(t :bool). ¬¬t ⇔ t
∀(s :α -> bool) (t :α -> bool) (x :α). (x INSERT s) DIFF t = if x ∈ t then s DIFF t else x INSERT s DIFF t
∀(set :α -> bool) (e :α -> bool). POW set e ⇔ e ⊆ set
(∀(x :α) (y :α). (R :α -> α -> bool) x y ⇒ (R' :α -> α -> bool) x y) ⇒ R^= (x :α) (y :α) ⇒ R'^= x y

nfst (0 :num) = (0 :num)
∀(s :α -> bool). s ∪ s = s
(∀(l :α list). TAKE (0 :num) l = ([] :α list)) ∧ ∀(n :num) (x :α) (l :α list). TAKE (SUC n) (x::l) = x::TAKE n l
∀(l :α list). NULL l ⇔ l = ([] :α list)
∀(x :α -> bool) (y :β -> bool). FUNSET x y = DFUNSET x (K y :α -> β -> bool)
∀(A :bool). ¬A ∧ A ⇔ F
∀(x :α) (y :β). (INL x :α + β) ≠ (INR y :α + β)
∀(ls :α list) (n :num). DROP n ls = ([] :α list) ⇔ LENGTH ls ≤ n
DIV2 (BIT1 (x :num)) = x
∀(R :α -> α -> bool) (P :α -> α -> bool). (∀(x :α). P x x) ∧ (∀(x :α) (y :α) (z :α). R x y ∧ P y z ⇒ P x z) ⇒ ∀(x :α) (y :α). R* x y ⇒ P x y
∀(s :α -> bool) (x :α). x INSERT s = {x} ∪ s
BIJ (f :α -> β) 𝕌(:α) 𝕌(:β) ⇒ ∀(x :α) (y :α). f x = f y ⇔ x = y
∀(l :α list) (f :α -> β). MAP f (TL l) = TL (MAP f l)
∀(s :α -> bool) (f :α -> β -> β) (b :β). FINITE s ⇒ ITSET f s b = if s = (∅ :α -> bool) then b else ITSET f (REST s) (f (CHOICE s) b)
∀(x :α) (l :α list). ALL_DISTINCT (SNOC x l) ⇔ ¬MEM x l ∧ ALL_DISTINCT l
∀(p :bool) (q :bool) (m :bool). (stmarker m ∨ p ⇔ p ∨ stmarker m) ∧ (p ∨ q ∨ stmarker m ⇔ (p ∨ q) ∨ stmarker m) ∧ ((p ∨ stmarker m) ∨ q ⇔ (p ∨ q) ∨ stmarker m)
∀(x :α). {x} ≠ (∅ :α -> bool)
S (f :α # β -> δ -> γ) (UNCURRY (g :α -> β -> δ)) = UNCURRY (S ((S :(β -> δ -> γ) -> (β -> δ) -> β -> γ) ∘ ($o f :(β -> α # β) -> β -> δ -> γ) ∘ ($, :α -> β -> α # β)) g)
∀(s :α -> bool). s DIFF (∅ :α -> bool) = s
∀(x :α) (s :α -> bool). x ∈ s ⇔ x INSERT s = s
∀(f :α -> β) (P :α -> bool) (Q :α -> β -> bool). DFUNSET P Q f ⇔ ∀(x :α). x ∈ P ⇒ f x ∈ Q x
∀(R :α -> α -> bool) (P :α -> α -> bool). (∀(x :α). P x x) ∧ (∀(x :α) (y :α) (z :α). R x y ∧ R* y z ∧ P y z ⇒ P x z) ⇒ ∀(x :α) (y :α). R* x y ⇒ P x y
∀(f :α -> β). (∀(s :β -> bool). SURJ f (∅ :α -> bool) s ⇔ s = (∅ :β -> bool)) ∧ ∀(s :α -> bool). SURJ f s (∅ :β -> bool) ⇔ s = (∅ :α -> bool)
∀(P :α -> bool) (f :β -> α) (l :β list). EVERY P (MAP f l) ⇔ EVERY (P ∘ f) l
∀(n1 :num) (n2 :num). (ind_type$INJN n1 :num -> α -> bool) = (ind_type$INJN n2 :num -> α -> bool) ⇔ n1 = n2
∀(x :α list) (y :α list) (z :α list). y ≼ x ∧ z ≼ y ⇒ z ≼ x
∀(P :α -> bool) (x :α) (l :α list). EVERY P (SNOC x l) ⇔ EVERY P l ∧ P x
(∀(x :α) (y :α). (R :α -> α -> bool) x y ⇒ (f :α -> β) x = f y) ⇒ ∀(x :α) (y :α). SC R x y ⇒ f x = f y
∀(R :α -> α -> bool) (f :β -> α). transitive R ⇒ transitive (inv_image R f)
∀(n :num) (l :α list). n < LENGTH l ⇒ SEG (1 :num) n l = [EL n l]
(∀(a :α list) (c :α list). (d :α list) ≠ a ++ [(b :α)] ++ c) ⇔ ¬MEM b d
∀(a :α list) (b :α list) (c :α list). a ++ b ≼ a ++ c ⇔ b ≼ c
countable 𝕌(:α # β) ⇔ countable 𝕌(:α) ∧ countable 𝕌(:β)
∀(l1 :α list) (l2 :α list) (e :α). FRONT (l1 ++ e::l2) = l1 ++ FRONT (e::l2)

∀(v :α) (rows_old :(α -> β option) list) (rows_new :(α -> β option) list) (r :α -> β option). PMATCH v rows_old = PMATCH v rows_new ⇒ PMATCH v (r::rows_old) = PMATCH v (r::rows_new)
∀(R :α -> α -> bool) (Q :α -> bool) (a :α). Q a ∧ (∀(y :α) (z :α). Q y ∧ R y z ⇒ Q z) ⇒ ∀(z :α). $R^{\star}$ a z ⇒ Q z
∀(n :num) (l :α list). LENGTH (DROP n l) = LENGTH l − n
(∀(l1 :α list) (l2 :α list). ([] :α list) = l1 ++ l2 ⇔ l1 = ([] :α list) ∧ l2 = ([] :α list)) ∧ ∀(l1 :α list) (l2 :α list). l1 ++ l2 = ([] :α list) ⇔ l1 = ([] :α list) ∧ l2 = ([] :α list)
∀(P :α option -> bool). P (NONE :α option) ∧ (∀(a :α). P (SOME a)) ⇒ ∀(x :α option). P x
∀(R :α -> α -> bool). WF R ⇒ ∀(P :α -> bool). (∀(x :α). (∀(y :α). R y x ⇒ P y) ⇒ P x) ⇒ ∀(x :α). P x
∀(s :α -> bool). INFINITE s ⇒ INFINITE (REST s)
transitive (R1 :α -> α -> bool) ∧ transitive (R2 :α -> α -> bool) ⇒ transitive (R1 $\cap_r$ R2)
∀(l :α list) (m :num) (x :α). MEM x (TAKE m l) ⇒ MEM x l
∀(x :α) (s :α -> bool). x ∉ s ⇒ ∀(t :α -> bool). s ⊆ x INSERT t ⇔ s ⊆ t
∀(m :num) (n :num). m ≤ n ⇒ count m ⊆ count n
∀(R :α -> α -> bool) (f :β -> α). reflexive R ⇒ reflexive (inv_image R f)
∀(R :α -> α -> bool) (P :α -> α -> bool). (∀(x :α) (y :α). R x y ⇒ P x y) ∧ (∀(x :α) (y :α) (z :α). P x y ∧ P y z ⇒ P x z) ⇒ ∀(u :α) (v :α). R^+ u v ⇒ P u v
LIST_BIND [(x :β)] (f :β -> α list) = f x
(∀(x :α) (y :α). (R :α -> α -> bool) x y ⇒ (Q :α -> α -> bool) x y) ⇒ R^+ (x :α) (y :α) ⇒ Q^+ x y
∀(X :α -> bool) (P :(α -> bool) -> bool). X ⊆ BIGINTER P ⇔ ∀(Y :α -> bool). Y ∈ P ⇒ X ⊆ Y
∀(P :α -> bool) (Q :β -> bool) (x :α # β). x ∈ P × Q ⇔ FST x ∈ P ∧ SND x ∈ Q
∀(f :α -> β) (x :α) (s :α -> bool) (t :β -> bool). INJ f (x INSERT s) t ⇔ INJ f s t ∧ f x ∈ t ∧ ∀(y :α). y ∈ s ∧ f x = f y ⇒ x = y
FINITE (s :α -> bool) ∧ FINITE (t :α -> bool) ⇒ CARD (s ∪ t) ≤ CARD s + CARD t
∀(l :α list) (f :α -> β). l ≠ ([] :α list) ⇒ LAST (MAP f l) = f (LAST l)
∀(m :num). m < SUC m ∧ m < SUC (SUC m)
∀(s :α -> bool). ¬(s ⊂ s)
∀(l1 :α list) (n :num). LENGTH l1 ≤ n ⇒ ∀(l2 :α list). EL n (l1 ++ l2) = EL (n − LENGTH l1) l2
LINV_OPT (f :α -> β) (s :α -> bool) (y :β) = SOME (x :α) ⇒ x ∈ s ∧ f x = y
∀(P :α -> bool) (l :α list). EVERY P (REVERSE l) ⇔ EVERY P l
∀(x :α) (s :α -> bool) (t :α -> bool). DISJOINT (x INSERT s) t ⇔ DISJOINT s t ∧ x ∉ t
∀(l :num list). SUM (REVERSE l) = SUM l
𝕌(:α # β) = 𝕌(:α) × 𝕌(:β)
∀(ll :α list). ll = ([] :α list) ∨ ∃(x :α) (l :α list). ll = SNOC x l
∀(P :α + β -> bool). (∃(s :α + β). P s) ⇔ (∃(x :α). P (INL x :α + β)) ∨ ∃(y :β). P (INR y :α + β)
∀(R :α -> α -> bool). reflexive R ⇒ reflexive R^+
∀(p :bool) (q :bool) (m :bool). (p ∨ stmarker m ⇔ stmarker m ∨ p) ∧ ((stmarker m ∨ p) ∨ q ⇔ stmarker m ∨ p ∨ q) ∧ (p ∨ stmarker m ∨ q ⇔ stmarker m ∨ p ∨ q)
∀(l1 :α list) (l2 :α list) (f :α -> β) (f' :α -> β). l1 = l2 ∧ (∀(x :α). MEM x l2 ⇒ f x = f' x) ⇒ MAP f l1 = MAP f' l2
(∀(x :α) (y :α). (R :α -> α -> bool) x y ⇒ (Q :β -> β -> bool) ((f :α -> β) x) (f y)) ∧ reflexive Q ∧ transitive Q ⇒ ∀(x :α) (y :α). $R^{\star}$ x y ⇒ Q (f x) (f y)

∀(s :α -> bool) (t :α -> bool) (u :α -> bool). s ⊆ t ∧ t ⊂ u ⇒ s ⊂ u
∀(p :α # β) (q :α # β). p = q ⇔ FST p = FST q ∧ SND p = SND q
∀(n :num). EL n (l :α list) = if n = (0 :num) then HD l else EL (PRE n) (TL l)
∀(s :α -> bool) (t :α -> bool) (x :α). x ∈ s ∪ t ⇔ x ∈ s ∨ x ∈ t
∀(l1 :num list) (l2 :num list). SUM (l1 ++ l2) = SUM l1 + SUM l2
∀(M :α + β) (M' :α + β) (f :α -> γ) (f1 :β -> γ). M = M' ∧ (∀(x :α). M' = (INL x :α + β) ⇒ f x = (f' :α -> γ) x) ∧ (∀(y :β). M' = (INR y :α + β) ⇒ f1 y = (f1' :β -> γ) y) ⇒ (sum_CASE M f f1 :γ) = (sum_CASE M' f' f1' :γ)
REVERSE (l :α list) = ([] :α list) ⇔ l = ([] :α list)
∀(l :α list). FOLDR (CONS :α -> α list -> α list) ([] :α list) l = l
∀(f :α -> β) (x :α) (s :α -> bool). x ∉ s ⇒ IMAGE f (s DELETE x) = IMAGE f s
∀(c :bool) (x :bool). (if c then F else x) ⇔ ¬c ∧ x
OPTION_CHOICE (m1 :α option) (NONE :α option) = m1
(∀(x :α) (y :α). (R :α -> α -> bool) x y ⇒ (f :α -> β) x = f y) ⇒ ∀(x :α) (y :α). RC R x y ⇒ f x = f y
∀(c :bool) (l :α list). EXISTS (λ(x :α). c) l ⇔ l ≠ ([] :α list) ∧ c
∀(x :α list) (y :α list). x ≼ y ∧ LENGTH x = LENGTH y ⇔ x = y
((SOME (f :β -> α) <*> SOME (x :β)) :α option) = SOME (f x)
∀(l :α list). LENGTH l = (0 :num) ⇔ l = ([] :α list)
(R :α -> α -> bool) equiv_on (s :α -> bool) ⇒ ∀(t1 :α -> bool) (t2 :α -> bool). t1 ∈ partition R s ∧ t2 ∈ partition R s ∧ t1 ≠ t2 ⇒ DISJOINT t1 t2
∀(L :α list). LENGTH L = LEN L (0 :num)
∀(R :α -> α -> bool) (x :α) (z :α). R^+ x z ⇒ R x z ∨ ∃(y :α). R x y ∧ R^+ y z
∀(l1 :α list) (l2 :α list). REVERSE (l1 ++ l2) = REVERSE l2 ++ REVERSE l1
total (R1 :α -> α -> bool) ∧ total (R2 :β -> β -> bool) ⇒ total (R1 LEX R2)
(∀(y :α) (x :α). (INL x :α + β) = (INL y :α + β) ⇔ x = y) ∧ ∀(y :β) (x :β). (INR x :α + β) = (INR y :α + β) ⇔ x = y
∀(s :α -> bool). POW s ≠ (∅ :(α -> bool) -> bool)
∀(l :α list). l ≠ ([] :α list) ⇒ EL (PRE (LENGTH l)) l = LAST l
∀(x :α -> bool) (y :α -> bool). x ∪ y DIFF x = y DIFF x ∧ x ∪ y DIFF y = x DIFF y
∀(P :α -> bool) (Q :β -> bool) (x :α # β). (P × Q) x ⇔ FST x ∈ P ∧ SND x ∈ Q
∀(n :num). SUC n ≠ n
∀(x :α). x ∉ (∅ :α -> bool)
(∀(m :num). m < (n :num) ⇒ (f1 :num -> α) m = (f2 :num -> α) m) ⇒ GENLIST f1 n = GENLIST f2 n
∀(f :α -> β) (l1 :α list) (l2 :α list). INJ f (set l1 ∪ set l2) 𝕌(:β) ⇒ (MAP f l1 = MAP f l2 ⇔ l1 = l2)
∀(R :α -> α -> bool) (x :α). (∀(y :α). R y x ⇒ WFP R y) ⇒ WFP R x
∀(l2 :α list) (l1 :α list). LASTN (LENGTH l2) (l1 ++ l2) = l2
∀(l :α list). (0 :num) < LENGTH l ⇒ LENGTH (TL l) = LENGTH l − (1 :num)
∀(s :α -> bool) (t :β -> bool) (f :α -> β). BIJ f s t ⇒ ∀(e :α). e ∈ s ⇒ BIJ f (s DELETE e) (t DELETE f e)
∀(n :num) (x :α). LENGTH (REPLICATE n x) = n
IMAGE (I :α -> α) (s :α -> bool) = s

antisymmetric ($RSUBSET :(α -> β -> bool) -> (α -> β -> bool) -> bool)
∀(l :α list). LASTN (LENGTH l) l = l
∀(R :α -> α -> bool). equivalence R ⇔ ∀(x :α) (y :α). R x y ⇔ R x = R y
∀(l :α list) (x :α). ELL (LENGTH l) (SNOC x l) = if NULL l then x else HD l
∀(l :α list) (m :num) (P :α -> bool). EXISTS P (DROP m l) ⇒ EXISTS P l
∀(m :num) (n :num). (λ(x :num) (y :num). y = SUC x)$^{+}$ m (SUC n) ⇔ (λ(x :num) (y :num). y = SUC x)* m n
∀(P :γ -> δ) (M :α # β) (N :α -> β -> γ). P (let ((x :α),(y :β)) = M in N x y) = (let ((x :α),(y :β)) = M in P (N x y))
∀(R :α -> α -> bool). rcdiamond R ⇔ diamond (RC R)
∀(s :α -> bool). ¬(𝕌(:α) ⊂ s)
∀(l :(α # β) list). ZIP (UNZIP l) = l
∀(P :α -> bool). BIGINTER {P} = P
∀(M :'a1 # 'a2) (N :'a1 -> 'a2 -> β -> γ) (b :β). (let ((x :'a1),(y :'a2)) = M in N x y) b = (let ((x :'a1),(y :'a2)) = M in N x y b)
(∀(x :α). x ∈ (s :α -> bool) ⇒ (INR x :β + α) ∈ (t :β + α -> bool)) ⇒ INJ (INR :α -> β + α) s t
∀(n :num) (m :num). n < m ⇒ tri n < tri m
∀(m :num) (n :num). SUC m < n ⇒ m < n
transitive (R :α -> α -> bool)^=
∀(s :α -> bool) (t :α -> bool). s = t ⇒ s ⊆ t ∧ t ⊆ s
∀(l :α list) (n :num). n < LENGTH (FRONT l) ∧ ¬NULL l ⇒ EL n (FRONT l) = EL n l
(s0 :α -> bool) ⊆ (s :α -> bool) ∧ INJ (f :α -> β) s (t :β -> bool) ⇒ BIJ f s0 (IMAGE f s0)
∀(n :num). LENGTH (l1 :α list) < n ⇒ TAKE n (l1 ++ (l2 :α list)) = l1 ++ TAKE (n − LENGTH l1) l2
∀(x :α + β). ISL x ⇒ (INL (OUTL x) :α + β) = x
∀(s :num -> bool). FINITE s ⇒ ∀(e :num). SUM_SET (s DELETE e) = if e ∈ s then SUM_SET s − e else SUM_SET s
∀(f :α -> β). IMAGE f (∅ :α -> bool) = (∅ :β -> bool)
∀(m :num) (n :num). n ≤ n ⊗ m
∀(l :α list) (x :α). MEM x (TL l) ⇒ MEM x l
∀(R :α -> α -> bool) (s :α -> bool) (t :α -> bool). pairwise R t ∧ s ⊆ t ⇒ pairwise R s
¬¬(p :bool) ⇒ p
∀(s :α -> bool). (∅ :α -> bool) ∈ POW s
∀(l :α list). ¬NULL l ⇒ HD l::TL l = l
∀(x :α) (y :α list). FRONT (x::y) ≼ x::y
$! (UNCURRY (f :α -> β -> bool)) ⇔ $! (($! :(β -> bool) -> bool) ∘ f)
∀(n :num) (l :α list). n ≤ LENGTH l ⇒ ∀(x :α). LASTN n (x::l) = LASTN n l
∀(x :α). SOME x ≠ (NONE :α option)
∀(R :α -> α -> bool). symmetric (SC R)
∀(l1 :α list) (l1' :α list) (l2 :β list) (l2' :β list) (P :α -> β -> bool) (P' :α -> β -> bool). l1 = l1' ∧ l2 = l2' ∧ (∀(x :α) (y :β). MEM x l1' ∧ MEM y l2' ⇒ (P x y ⇔ P' x y)) ⇒ (LIST_REL P l1 l2 ⇔ LIST_REL P' l1' l2')
∀(p :α # β) (f :α -> γ) (g :β -> δ). SND ((f ## g) p) = g (SND p)

ind_type$ISO (f :α -> γ) (f' :γ -> α) ∧ ind_type$ISO (g :β -> δ) (g' :δ -> β) ⇒ ind_type$ISO (λ(h :α -> β) (a' :γ). g (h (f' a'))) (λ(h :γ -> δ) (a :α). g' (h (f a)))
∀(R :α -> α -> bool). symmetric R ⇒ R^T = R
∀(l1 :α list) (n :num). LENGTH l1 ≤ n ⇒ ∀(l2 :α list). DROP n (l1 ++ l2) = DROP (n − LENGTH l1) l2
∀(n :num) (l1 :α list). n ≤ LENGTH l1 ⇒ ∀(l2 :α list). TAKE n (l1 ++ l2) = TAKE n l1
∀(l1 :α list) (l2 :α list). LENGTH (l1 ++ l2) = LENGTH l1 + LENGTH l2
∀(s :α -> bool) (t :α -> bool). s ⊆ t ⇔ ∀(x :α). s x ⇒ t x
∀(P :α -> bool) (l :α list). EVERY P l ⇔ FOLDR $/\ T (MAP P l)
∀(l1 :α list) (l2 :α list). DROP (LENGTH l1) (l1 ++ l2) = l2
∀(f :'z -> 'z). INVOL f ⇔ ∀(x :'z). f (f x) = x
∀(R :α -> α -> bool). antisymmetric R^T ⇔ antisymmetric R
∀(P :α -> bool). P × (∅ :β -> bool) = (∅ :α # β -> bool) ∧ (∅ :γ -> bool) × P = (∅ :γ # α -> bool)
∀(P :α -> bool) (l :α list). EVERY P l ⇔ FOLDR (λ(x :α) (l' :bool). P x ∧ l') T l
¬NULL (l2 :α list) ∧ (n :num) = LENGTH (l1 :α list) ⇒ EL n (l1 ++ l2) = HD l2
(r1 :α) = (r2 :α) ⇒ (Abbrev ((v :α) = r1) ⇔ Abbrev (v = r2))
MAP2 (f :β -> γ -> α) (x :β list) ([] :γ list) = ([] :α list)
∀(c :α list) (a :α list). a ≼ a ++ c
∀(P :α -> bool) (Q :α -> bool) (l :α list). (EXISTS (λ(x :α). P x ∨ Q x) l ⇔ EXISTS P l) ∨ EXISTS Q l
∀(x :α option). ¬IS_SOME x ⇔ x = (NONE :α option)
OPTION_MAP2 (f :α -> β -> γ) (o1 :α option) (o2 :β option) = (NONE :γ option) ⇔ o1 = (NONE :α option) ∨ o2 = (NONE :β option)
∀(l1 :α list) (l2 :α list). REVERSE l1 = l2 ⇔ l1 = REVERSE l2
∀(s :num -> bool). countable s
∀(P :α -> bool) (l :α list). EXISTS P l ⇔ ∃(e :α). MEM e l ∧ P e
∀(x :α) (l :α list). MEM x l ⇔ FOLDR $\/ F (MAP ($= x) l)
symmetric (R :α -> α -> bool) ⇔ $R^T \subseteq_r$ R
OPTION_IGNORE_BIND (NONE :β option) (m :α option) = (NONE :α option) ∧ OPTION_IGNORE_BIND (SOME (v :γ)) m = m
∀(s :α -> bool) (t :α -> bool). s ∩ t = t ∩ s
∀(p :bool) (q :bool) (m :bool). (p ∧ stmarker m ⇔ stmarker m ∧ p) ∧ ((stmarker m ∧ p) ∧ q ⇔ stmarker m ∧ p ∧ q) ∧ (p ∧ stmarker m ∧ q ⇔ stmarker m ∧ p ∧ q)
∀(x :α) (l :α list). SNOC x l = REVERSE (x::REVERSE l)
SURJ (f :α -> β) (s :α -> bool) (IMAGE f s)
PMATCH_ROW (p :β -> γ) (g :β -> bool) (r :β -> α) (i :γ) = SOME (y :α) ⇒ ∃(x :β). PMATCH_ROW_COND p g i x ∧ y = r x
∀(x :α) (sos :(α -> bool) -> bool). BIGUNION sos x ⇔ ∃(s :α -> bool). x ∈ s ∧ s ∈ sos
(∀(p :α # β). (P :α # β -> bool) p) ⇔ ∀(p_1 :α) (p_2 :β). P (p_1,p_2)
∀(f :α -> β) (s :β -> bool) (t :β -> bool). PREIMAGE f (s ∩ t) = PREIMAGE f s ∩ PREIMAGE f t
∀(s :α -> bool). (∃(x :α). x ∈ s) ⇔ s ≠ (∅ :α -> bool)
(diag (A :α -> bool))^T = diag A
∀(f :α -> bool) (n :num) (x :α). EVERY f (REPLICATE n x) ⇔ n = (0 :num) ∨ f x

{x | x = (y :α)} = {y}
transitive ($PSUBSET :(α -> bool) -> (α -> bool) -> bool)
∀(f :α -> β -> γ). CURRY (UNCURRY f) = f
∀(e :α) (l1 :α list) (l2 :α list). MEM e (l1 ++ l2) ⇔ MEM e l1 ∨ MEM e l2
∀(x :α list) (y :α list). x ≼ y ∧ y ≼ x ⇒ x = y
∀(P :α -> bool) (L :α list) (M :α list). FILTER P (L ++ M) = FILTER P L ++ FILTER P M
(∀(x :α) (y :α). (R :α -> α -> bool) x y ⇒ (Q :α -> α -> bool) x y) ⇒ SC R (x :α) (y :α) ⇒ SC Q x y
∀(P :α -> bool) (l :α list). ¬EXISTS P l ⇔ EVERY ($¬ ∘ P) l
∀(x :bool) (x' :bool) (y :bool) (y' :bool). (x ⇒ x') ∧ (x' ⇒ y' ⇒ y) ⇒ (x' ⇒ y') ⇒ x ⇒ y
∀(l :α list) (e :α) (y :α). MEM y (FRONT (e::l)) ⇒ MEM y (e::l)
(∀(l1 :α list) (l2 :α list). l1 ++ l2 = l1 ⇔ l2 = ([] :α list)) ∧ (∀(l1 :α list) (l2 :α list). l1 ++ l2 = l2 ⇔ l1 = ([] :α list)) ∧ (∀(l1 :α list) (l2 :α list). l1 = l1 ++ l2 ⇔ l2 = ([] :α list)) ∧ ∀(l1 :α list) (l2 :α list). l2 = l1 ++ l2 ⇔ l1 = ([] :α list)
∀(x :α). ALL_DISTINCT [x]
∀(s :α -> bool). FINITE s ⇒ ∀(x :α). CARD (x INSERT s) = if x ∈ s then CARD s else SUC (CARD s)
∀(m :num) (n :num). m ≤ n ⊗ m
∀(l :α list) (l1 :α list) (l2 :α list). l1 ≼ l ∧ l2 ≼ l ⇒ l1 ≼ l2 ∨ l2 ≼ l1
∀(A :bool). A ⇒ ¬A ⇒ F
∀(f :α -> β) (l :α list). set (MAP f l) = IMAGE f (set l)
∀(x :α) (s :α -> bool) (t :α -> bool). (x INSERT s) ∩ t = if x ∈ t then x INSERT s ∩ t else s ∩ t
∀(ls :α list) (n :num). n < LENGTH ls ⇒ TAKE (n + (1 :num)) ls = SNOC (EL n ls) (TAKE n ls)
OPTION_MAP2 (f :β -> γ -> α) (SOME (x :β)) (SOME (y :γ)) = SOME (f x y) ∧ OPTION_MAP2 f (SOME x) (NONE :γ option) = (NONE :α option) ∧ OPTION_MAP2 f (NONE :β option) (SOME y) = (NONE :α option) ∧ OPTION_MAP2 f (NONE :β option) (NONE :γ option) = (NONE :α option)
∀(R1 :α -> α -> bool) (R2 :β -> β -> bool) (v1 :α # β) (v2 :α # β) (R1' :α -> α -> bool) (R2' :β -> β -> bool) (v1' :α # β) (v2' :α # β). v1 = v1' ∧ v2 = v2' ∧ (∀(a :α) (b :β) (c :α) (d :β). v1' = (a,b) ∧ v2' = (c,d) ⇒ (R1 a c ⇔ R1' a c)) ∧ (∀(a :α) (b :β) (c :α) (d :β). v1' = (a,b) ∧ v2' = (c,d) ∧ a = c ⇒ (R2 b d ⇔ R2' b d)) ⇒ ((R1 LEX R2) v1 v2 ⇔ (R1' LEX R2') v1' v2')
∀(f :α -> β) (R :α -> α -> bool) (D :α -> bool) (P :α -> β -> bool) (M :(α -> β) -> α -> β) (x :α). f = WFREC R M ∧ WF R ∧ INDUCTIVE_INVARIANT_ON R D P M ∧ D x ⇒ P x (f x)
∀(x :bool) (y :α -> bool). (K x :α -> bool) ⊆ y ⇔ ¬x ∨ 𝕌(:α) ⊆ y
(OPTREL (R :α -> β -> bool) (SOME (x :α)) (NONE :β option) ⇔ F) ∧ (OPTREL R (NONE :α option) (SOME (y :β)) ⇔ F) ∧ (OPTREL R (NONE :α option) (NONE :β option) ⇔ T) ∧ (OPTREL R (SOME x) (SOME y) ⇔ R x y)
∀(s :α -> bool). SING s ⇒ FINITE s
∀(s :α + β) (s' :α + β) (P :α -> bool) (P' :α -> bool) (Q :β -> bool) (Q' :β -> bool). s = s' ∧ (∀(a :α). s' = (INL a :α + β) ⇒ (P a ⇔ P' a)) ∧ (∀(b :β). s' = (INR b :α + β) ⇒ (Q b ⇔ Q' b)) ⇒ (SUM_ALL P Q s ⇔ SUM_ALL P' Q' s')
∀(f :α -> β) (M :(α -> bool) -> bool). IMAGE f (BIGUNION M) = BIGUNION (IMAGE (IMAGE f) M)
∀(s :α -> bool) (t :α -> bool) (x :α). s DIFF (x INSERT t) = s DELETE x DIFF t

∀(v :α) (p :β -> α) (g :β -> bool). (∀(x :β). ¬g x) ⇒ (PMATCH_ROW_COND_EX v p g ⇔ F)
(∀(x :α) (y :α). (f :α -> β) x = f y ⇔ x = y) ⇒ (IMAGE f (s1 :α -> bool) = IMAGE f (s2 :α -> bool) ⇔ s1 = s2)
∀(R :α -> α -> bool). transitive R^T ⇔ transitive R
∀(t :bool). T ∨ t ⇔ T
∀(t :bool). t ∧ F ⇔ F
∀(t :bool). T ∧ t ⇔ t
∀(f :α -> β # bool). (∀(x :α). ¬SND (f x)) ⇒ GSPEC f = (∅ :β -> bool)
∀(R :α -> α -> bool). equivalence R ⇒ R^T = R
(∀(x :α) (y :α). (R :α -> α -> bool) x y ⇒ R ((f :α -> α) x) (f y)) ⇒ ∀(x :α) (y :α). RC R x y ⇒ RC R (f x) (f y)
SND (p :β # α) = (y :α) ⇔ ∃(x :β). p = (x,y)
∀(P :num -> bool). P (0 :num) ∧ (∀(n :num). P n ⇒ P (SUC n)) ⇒ ∀(n :num). P n
(∀(x :α). (P :α -> bool) x ⇒ (Q :α option -> bool) (SOME x)) ∧ ((∀(x :α). ¬P x) ⇒ Q (NONE :α option)) ⇒ Q ($some P)
∀(l :α list). l ≠ ([] :α list) ⇒ ELL (PRE (LENGTH l)) l = HD l
∀(n1 :num) (n2 :num) (x :α). n1 < n2 ⇒ EL n1 (REPLICATE n2 x) = x
∀(n :num) (l1 :α list) (l2 :α list). n < LENGTH l1 ⇒ EL n (l1 ++ l2) = EL n l1
∀(f :α -> β) (t :β -> bool) (sp :α -> bool). PREIMAGE f (COMPL t) ∩ sp = sp DIFF PREIMAGE f t
∀(x :α) (s :α -> bool). x ∈ s ⇒ ∀(f :α -> β). f x ∈ IMAGE f s
∀(l1 :α list) (h :α) (l2 :α list) (l3 :α list). l1 ++ h::l2 ++ l3 = l1 ++ h::(l2 ++ l3)
∀(n :num). count n = if n = (0 :num) then (∅ :num -> bool) else (let (p :num) = PRE n in p INSERT count p)
∀(x :α) (s :α -> bool). x ∈ COMPL s ⇔ x ∉ s
∀(l :α list) (n :num). LENGTH l ≤ n ⇒ DROP n l = ([] :α list)
∀(P :α -> bool) (B :(α -> bool) -> bool). BIGINTER (P INSERT B) = P ∩ BIGINTER B
∀(xs :α list) (ys :β list) (P :α -> β -> bool) (x :α). LIST_REL P xs ys ∧ MEM x xs ⇒ ∃(y :β). MEM y ys ∧ P x y
longest_prefix (∅ :α list -> bool) = ([] :α list)
(∀(x :α) (y :α). (R :α -> α -> bool) x y ⇒ R ((f :α -> α) x) (f y)) ⇒ ∀(x :α) (y :α). SC R x y ⇒ SC R (f x) (f y)
∀(s :α -> bool) (t :α -> bool). countable s ∧ t ⊆ s ⇒ countable t
∀(ls :num list). SUM ls = (0 :num) ⇔ ∀(x :num). MEM x ls ⇒ x = (0 :num)
∀(s :α -> bool) (t :α -> bool). DISJOINT s t ⇔ DISJOINT t s
∀(n :num). n ≤ LENGTH (l1 :α list) ⇒ TAKE n (l1 ++ (l2 :α list)) = TAKE n l1
∀(xs :α list) (k :num) (n :num). TAKE k (DROP n xs) = DROP n (TAKE (k + n) xs)
∀(P1 :α -> bool) (P2 :α -> bool) (l :α list). EVERY P1 (FILTER P2 l) ⇔ EVERY (λ(x :α). P2 x ⇒ P1 x) l
∀(n :num) (l :α list). n < LENGTH l ⇒ EL n l = HD (SEG (1 :num) n l)
∀(x :α option). (option_CASE x x (SOME :α -> α option) :α option) = x
∀(R :α -> α -> bool). WeakOrder R ⇒ Order R
∀(s :α -> bool) (t :α -> bool) (u :α -> bool). DISJOINT (s ∪ t) u ⇔ DISJOINT s u ∧ DISJOINT t u

(s :α -> bool) ⊆ (t :α -> bool) ⇒ s ∪ (t DIFF s) = t ∧ t DIFF s ∪ s = t
(∀(s :(α -> bool) -> bool) (t :α -> bool). DISJOINT (BIGUNION s) t ⇔ ∀(s' :α -> bool). s' ∈ s ⇒ DISJOINT s' t) ∧ ∀(s :(α -> bool) -> bool) (t :α -> bool). DISJOINT t (BIGUNION s) ⇔ ∀(s' :α -> bool). s' ∈ s ⇒ DISJOINT t s'
(∀(x :α) (y :β). (R1 :α -> β -> bool) x y ⇒ (R2 :α -> β -> bool) x y) ⇒ LIST_REL R1 (l1 :α list) (l2 :β list) ⇒ LIST_REL R2 l1 l2
∀(R :α -> α -> bool) (x :α) (y :α) (z :α). R^= x y ∧ R^= y z ⇒ R^= x z
∀(x :α) (l :α list). SNOC x l = FOLDR (CONS :α -> α list -> α list) [x] l
∀(n :num) (x :α). REVERSE (REPLICATE n x) = REPLICATE n x
(y :α) ∈ {x | (P :α -> bool) x} ⇔ P y
∀(P :α -> bool) (x :α) (l :α list). EXISTS P (SNOC x l) ⇔ P x ∨ EXISTS P l
∀(v :α) (rows :(α -> β option) list). PMATCH_EQUIV_ROWS v rows rows
∀(I :num -> bool) (J :num -> bool) (n :α). I ≠ (∅ :num -> bool) ∧ J ≠ (∅ :num -> bool) ∧ I ⊆ J ⇒ MIN_SET J ≤ MIN_SET I
(∀(x :α) (y :α). (P :α -> bool) x ∧ (R :α -> α -> bool) x y ⇒ P y) ⇒ ∀(x :α) (y :α). P x ∧ RC R x y ⇒ P y
∀(P :α -> bool) (l :α list). ¬EVERY P l ⇔ EXISTS ($¬ ∘ P) l
∀(m :num) (l :α list) (n :num). n ≤ m ⇒ TAKE n (TAKE m l) = TAKE n l
∀(f :α -> β -> β) (e :β) (g :γ -> α) (l :γ list). FOLDR f e (MAP g l) = FOLDR (λ(x :γ) (y :β). f (g x) y) e l
∀(P :α -> β -> bool) (l1 :α list) (l2 :β list) (v1 :α) (v2 :β) (n :num). P v1 v2 ∧ LIST_REL P l1 l2 ⇒ LIST_REL P (LUPDATE v1 n l1) (LUPDATE v2 n l2)
∀(P :α -> β -> bool) (xs :α list) (ys :β list) (n :num). LIST_REL P xs ys ⇒ LIST_REL P (TAKE n xs) (TAKE n ys)
∀(l :α list) (m :num) (P :α -> bool). EXISTS P (TAKE m l) ⇒ EXISTS P l
RC ($SUBSET :(α -> bool) -> (α -> bool) -> bool) = ($SUBSET :(α -> bool) -> (α -> bool) -> bool)
transitive (R1 :α -> α -> bool) ∧ transitive (R2 :β -> β -> bool) ⇒ transitive (R1 LEX R2)
∀(P :α -> bool) (Q :α -> bool) (l :α list). EVERY (λ(x :α). P x ∧ Q x) l ⇔ EVERY P l ∧ EVERY Q l
∀(l1 :α list) (l2 :α list). l1 ≠ l2 ⇒ ∀(h1 :α) (h2 :α). h1::l1 ≠ h2::l2
∀(y :α) (l :α list). MEM y l ⇔ FOLDR (λ(x :α) (l' :bool). y = x ∨ l') F l
∀(x :α) (y :α). {x} = {y} ⇔ x = y
∀(P :unit -> bool). P () ⇒ ∀(x :unit). P x
∀(s :α -> bool). FINITE s ⇒ ∀(t :α -> bool). FINITE (s DIFF t)
∀(m :num) (n :num) (l :α list). m + n = LENGTH l ⇒ BUTLASTN m l ++ DROP n l = l
∀(l1 :α list) (l2 :β list). LENGTH l1 = LENGTH l2 ⇒ LENGTH (ZIP (l1,l2)) = LENGTH l1 ∧ LENGTH (ZIP (l1,l2)) = LENGTH l2
∀(x :α) (xs :α list). LENGTH (FRONT (x::xs)) = LENGTH xs
∀(s :α -> bool). COMPL s ∩ s = (∅ :α -> bool) ∧ COMPL s ∪ s = 𝕌(:α)
UNZIP ([] :(α # β) list) = (([] :α list),([] :β list)) ∧ UNZIP (((x :α),(y :β))::(t :(α # β) list)) = (let ((L1 :α list),(L2 :β list)) = UNZIP t in (x::L1,y::L2))
∀(s :α -> bool) (t :α -> bool) (x :α). DISJOINT (s DELETE x) t ⇔ DISJOINT (t DELETE x) s
∀(f :α -> β) (g :γ -> α) (s :γ -> bool). IMAGE f (IMAGE g s) = IMAGE (f ∘ g) s
∀(P :α -> bool) (Q :β -> bool) (x :β). P × (x INSERT Q) = P × {x} ∪ P × Q

∀(f :α -> β) (n :num) (l :α list). MAP f (TAKE n l) = TAKE n (MAP f l)
∀(P :α -> bool) (f :β -> α) (s :β -> bool). (∃(y :α). y ∈ IMAGE f s ∧ P y) ⇔ ∃(x :β). x ∈ s ∧ P (f x)
∀(x :α) (s :α -> bool). x ∈ REST s ⇔ x ∈ s ∧ x ≠ CHOICE s
REPLICATE (x :num) (y :α) = ([] :α list) ⇔ x = (0 :num)
∀(n :num). n < SUC n
∀(x :α) (y :α) (s :α -> bool). s DELETE x DELETE y = s DELETE y DELETE x
((p :bool) ⇔ ¬(q :bool)) ⇔ (p ∨ q) ∧ (¬q ∨ ¬p)
NULL ((l1 :α list) ++ (l2 :α list)) ⇔ NULL l1 ∧ NULL l2
IS_SOME (OPTION_MAP (f :α -> β) (x :α option)) ⇔ IS_SOME x
∀(M :α option) (M' :α option) (v :β) (f :α -> β). M = M' ∧ (M' = (NONE :α option) ⇒ v = (v' :β)) ∧ (∀(x :α). M' = SOME x ⇒ f x = (f' :α -> β) x) ⇒ (option_CASE M v f :β) = (option_CASE M' v' f' :β)
∀(f :α -> β) (s :α -> bool) (f' :α -> β) (s' :α -> bool). s = s' ∧ (∀(x :α). x ∈ s' ⇒ f x = f' x) ⇒ IMAGE f s = IMAGE f' s'
∀(x :α -> bool) (P :(α -> bool) -> bool). x ∈ P ⇒ x ⊆ BIGUNION P
∀(f :α -> β -> γ) (x :α) (y :β). UNCURRY f (x,y) = f x y
∀(sp :α -> bool) (s :(α -> bool) -> bool). (∀(t :α -> bool). t ∈ s ⇒ t ⊆ sp) ∧ s ≠ (∅ :(α -> bool) -> bool) ⇒ BIGINTER s ⊆ sp
(∀(s :α -> bool). (∅ :α -> bool) ∪ s = s) ∧ ∀(s :α -> bool). s ∪ (∅ :α -> bool) = s
∀(ls :α list). ls ≠ ([] :α list) ⇒ TAKE (PRE (LENGTH ls)) ls = FRONT ls
∀(f :α -> β -> α) (e :α) (l1 :β list) (l2 :β list). FOLDL f e (l1 ++ l2) = FOLDL f (FOLDL f e l1) l2
∀(R :α -> α -> bool). WF R ⇔ ∀(x :α). WFP R x
∀(l :α list list list). FLAT (FLAT l) = FLAT (MAP (FLAT :α list list -> α list) l)
∀(P :α -> bool) (Q :α -> bool) (l :α list). (∀(x :α). MEM x l ∧ P x ⇒ Q x) ∧ EVERY P l ⇒ EVERY Q l
(x1 :num) ⊗ (y1 :num) = (x2 :num) ⊗ (y2 :num) ⇔ x1 = x2 ∧ y1 = y2
(x :α) ∈ RDOM ((R1 :α -> β -> bool) $\cup_r$ (R2 :α -> β -> bool)) ⇔ x ∈ RDOM R1 ∨ x ∈ RDOM R2
∀(y :α) (x :β) (P :α -> bool). P y ∧ x = (f :α -> β) y ⇒ x ∈ {f x | P x}
((if (P :bool) then (X :α option) else (NONE :α option)) = (NONE :α option) ⇔ P ⇒ IS_NONE X) ∧ ((if P then (NONE :α option) else X) = (NONE :α option) ⇔ IS_SOME X ⇒ P) ∧ ((if P then X else (NONE :α option)) = SOME (x :α) ⇔ P ∧ X = SOME x) ∧ ((if P then (NONE :α option) else X) = SOME x ⇔ ¬P ∧ X = SOME x)
(∀(x :α). x ∈ (s :α -> bool) ⇒ (INL x :α + β) ∈ (t :α + β -> bool)) ⇒ INJ (INL :α -> α + β) s t
∀(m :num) (n :num). m < n ⇒ SUC m < SUC n
PMATCH_ROW (p :β -> γ) (g :β -> bool) (r :β -> α) (i :γ) ≠ (NONE :α option) ⇔ PMATCH_ROW_COND_EX i p g
OWHILE (G :α -> bool) (f :α -> α) (s :α) = (NONE :α option) ⇔ ∀(n :num). G (FUNPOW f n s)
∀(f :α -> α) (P :α -> bool) (l :α list). (∀(x :α). P (f x) ⇔ P x) ⇒ MAP f (FILTER P l) = FILTER P (MAP f l)
∀(s :α -> bool). COMPL (COMPL s) = s
(y :α) ∈ RRANGE (R :β -> α -> bool) ⇔ ∃(x :β). R x y

LIST_REL (P :α -> β -> bool) (REPLICATE (n :num) (x :α)) (REPLICATE n (y :β)) ⇔ n > (0 :num) ⇒ P x y
∀(s :α -> bool) (t :γ -> bool) (u :β -> bool). (∃(f :α -> γ). BIJ f s t) ∧ (∃(g :γ -> β). BIJ g t u) ⇒ ∃(h :α -> β). BIJ h s u
∀(R :α -> α -> bool). reflexive R ⇔ ($= :α -> α -> bool) $\subseteq_r$ R
DATATYPE ((sum :(α -> α + β) -> (β -> α + β) -> γ) (INL :α -> α + β) (INR :β -> α + β))
GENLIST (f :num -> α) (SUC (n :num)) = f (0 :num)::GENLIST (f ∘ SUC) n
∀(n :num) (l1 :α list). n ≤ LENGTH l1 ⇒ ∀(l2 :α list). DROP n (l1 ++ l2) = DROP n l1 ++ l2
∀(f :α -> β) (x :α) (l :α list). MAP f (SNOC x l) = SNOC (f x) (MAP f l)
INVOL $¬
∀(f :α -> num) (x :α) (y :α). measure f x y ⇔ f x < f y
∀(f :α -> β) (g :α -> β). f (EXT_POINT f g) = g (EXT_POINT f g) ⇔ f = g
∀(t :bool). t ∨ F ⇔ t
∀(l :α list). ¬NULL l ⇔ ∃(e :α). MEM e l
∀(f :α -> β -> β) (e :β) (P :α -> bool) (l :α list). FOLDR f e (FILTER P l) = FOLDR (λ(x :α) (y :β). if P x then f x y else y) e l
(R :α -> α -> bool) equiv_on (s :α -> bool) ⇒ ∀(t :α -> bool). t ∈ partition R s ⇒ ∀(x :α) (y :α). x ∈ t ∧ y ∈ t ⇒ R x y
∀(v :α) (rs1 :(α -> β option) list) (rs2 :(α -> β option) list) (i :num). i < LENGTH rs1 ⇒ (PMATCH_ROW_REDUNDANT v (rs1 ++ rs2) i ⇔ PMATCH_ROW_REDUNDANT v rs1 i)
∀(l1 :α list) (l2 :α list). IS_SUFFIX l1 l2 ⇔ REVERSE l2 ≼ REVERSE l1
∀(f :α # β -> γ). UNCURRY (CURRY f) = f
EL (NUMERAL (BIT1 (n :num))) ((l :α)::(ls :α list)) = EL (PRE (NUMERAL (BIT1 n))) ls ∧ EL (NUMERAL (BIT2 n)) (l::ls) = EL (NUMERAL (BIT1 n)) ls
∀(a1 :α list) (a0 :α). a0::a1 ≠ ([] :α list)
∀(x :num) (y :num). x ⊗ y = (0 :num) ⇔ x = (0 :num) ∧ y = (0 :num)
∀(R :α -> α -> bool). (∀(x :α) (y :α). R x y ⇒ R^+ x y) ∧ ∀(x :α) (y :α) (z :α). R^+ x y ∧ R^+ y z ⇒ R^+ x z
∀(v_old :α) (v_new :β) (rows_old :(α -> γ option) list) (rows_new :(β -> γ option) list) (r_old :α -> γ option) (r_new :β -> γ option). r_old v_old = r_new v_new ⇒ PMATCH v_old rows_old = PMATCH v_new rows_new ⇒ PMATCH v_old (r_old::rows_old) = PMATCH v_new (r_new::rows_new)
∀(s :α -> bool). FINITE s ⇒ ∀(t :α -> bool). t ⊆ s ⇒ FINITE t
∀(f :α -> β) (s :α -> bool) (t :β -> bool). INJ f s t ⇒ INJ f s (IMAGE f s)
∀(x :α) (y :α) (s :α -> bool). x ∈ y INSERT s ⇔ x = y ∨ x ∈ s
∀(v :α) (rows1 :(α -> β option) list) (rows2 :(α -> β option) list). PMATCH_EQUIV_ROWS v rows1 rows2 ⇔ PMATCH_EQUIV_ROWS v rows2 rows1
(antisymmetric (R1 :α -> α -> bool) ⇒ antisymmetric (R1 $\cap_r$ (R2 :α -> α -> bool))) ∧ (antisymmetric R2 ⇒ antisymmetric (R1 $\cap_r$ R2))
∀(s :α -> bool) (M :α -> num). (∃(x :α). x ∈ s) ⇔ ∃(x :α). x ∈ s ∧ ∀(y :α). y ∈ s ⇒ M x ≤ M y
∀(f :α -> β) (s :α -> bool) (t :α -> bool). IMAGE f (s ∩ t) ⊆ IMAGE f s ∩ IMAGE f t
∀(s :α -> bool) (t :α -> bool). s ∩ t = COMPL (COMPL s ∪ COMPL t)
∀(R :α -> α -> bool) (P :α -> α -> bool). WF R ∧ (∀(x :α) (y :α). P x y ⇒ R x y) ⇒ WF P

∀(f1 :num -> α -> bool) (f1' :num -> α -> bool) (f2 :num -> α -> bool) (f2' :num -> α -> bool). ind_type$INJP f1 f2 = ind_type$INJP f1' f2' ⇔ f1 = f1' ∧ f2 = f2'
∀(R :α -> α -> bool) (x :α) (y :α). R✡ x y ⇔ x = y ∨ ∃(u :α). R x u ∧ R✡ u y
∀(s :α -> bool) (t :α -> bool). s ∪ t = t ∪ s
∀(x :α) (y :α list) (a :α) (b :α list). SNOC x y = SNOC a b ⇔ x = a ∧ y = b
∀(n :num). TAKE n ([] :α list) = ([] :α list)
DROP (n :num) (REPLICATE (m :num) (a :α)) = REPLICATE (m − n) a
((a :α),(b :β)) ∈ {(y,(x :β)) | (P :α -> bool) y} ⇔ P a ∧ b = x
∀(l :α list list). FLAT (REVERSE l) = REVERSE (FLAT (MAP (REVERSE :α list -> α list) l))
∀(n :num) (m :num) (l :α list). n + m ≤ LENGTH l ⇒ LASTN n (BUTLASTN m l) = BUTLASTN m (LASTN (n + m) l)
∀(P :α -> bool) (Q :α -> bool) (l :α list). FILTER P (FILTER Q l) = FILTER (λ(x :α). P x ∧ Q x) l
∀(x :α option). (option_CASE x (NONE :α option) (SOME :α -> α option) :α option) = x
FINITE (∅ :α -> bool)
∀(s :α -> bool) (e :α) (s2 :α -> bool). s DELETE e ⊆ s2 ⇔ s ⊆ e INSERT s2
𝕌(:α -> bool) = POW 𝕌(:α)
NULL ([] :α list) ∧ ∀(h :α) (t :α list). ¬NULL (h::t)
∀(R :α -> α -> bool) (x :α). R^= x x
∀(v :α). IS_REDUNDANT_ROWS_INFO v ([] :(α -> β option) list) T ([] :bool list)
∀(f :α -> β) (s :(β -> bool) -> bool). PREIMAGE f (BIGUNION s) = BIGUNION (IMAGE (PREIMAGE f) s)
∀(f :'z -> 'z). IDEM f ⇔ ∀(x :'z). f (f x) = f x
∀(x :α) (s :α -> bool). x ∉ s DELETE x
(OPTION_MAP (f :β -> α) (x :β option) = (NONE :α option) ⇔ x = (NONE :β option)) ∧ ((NONE :α option) = OPTION_MAP f x ⇔ x = (NONE :β option))
∀(x :α) (y :α list) (z :α list). z ≼ SNOC x y ⇔ z ≼ y ∨ z = SNOC x y
∀(c :num) (i :α) (r :num -> num -> α -> bool). ind_type$ZCONSTR c i r ≠ (ind_type$ZBOT :num -> α -> bool)
∀(f :α -> β -> γ). ∃(fn :α # β -> γ). ∀(x :α) (y :β). fn (x,y) = f x y
(set (l :α list) = (∅ :α -> bool) ⇔ l = ([] :α list)) ∧ ((∅ :α -> bool) = set l ⇔ l = ([] :α list))
¬((p :bool) ⇒ (q :bool)) ⇒ p
∀(f :α -> β) (g :β -> γ) (s :γ -> bool). PREIMAGE f (PREIMAGE g s) = PREIMAGE (g ∘ f) s
∀(x :α) (n :num) (ys :α list). LENGTH (LUPDATE x n ys) = LENGTH ys
(∀(s :α -> bool). 𝕌(:α) ∪ s = 𝕌(:α)) ∧ ∀(s :α -> bool). s ∪ 𝕌(:α) = 𝕌(:α)
∀(M :β # γ) (M' :β # γ) (f :β -> γ -> α). M = M' ∧ (∀(x :β) (y :γ). M' = (x,y) ⇒ f x y = (f' :β -> γ -> α) x y) ⇒ (pair_CASE M f :α) = (pair_CASE M' f' :α)
∀(R :α -> α -> bool). (RC R)^{T} = RC R^{T}
∀(n :num). SUC n ≠ (0 :num)
∀(n :num) (l :α list). n ≤ LENGTH l ⇒ BUTLASTN n (LASTN n l) = ([] :α list)
∀(s :α -> bool) (t :α -> bool). s ∪ t = (∅ :α -> bool) ⇔ s = (∅ :α -> bool) ∧ t = (∅ :α -> bool)
∀(s :num -> bool). (∃(n :num). n ∈ s) ⇔ ∃(n :num). n ∈ s ∧ ∀(m :num). m ∈ s ⇒ n ≤ m
((∃(opt :α option). (P :α option -> bool) opt) ⇔ P (NONE :α option)) ∨ ∃(x :α). P (SOME x)
∀(P :α -> bool) (l1 :α list) (l2 :α list). EVERY P (l1 ++ l2) ⇔ EVERY P l1 ∧ EVERY P l2

∀(v :α) (rs :(α -> β option) list). (EVERY (λ(r :α -> β option). r v = (NONE :β option)) rs ⇒ F) ⇒ PMATCH_IS_EXHAUSTIVE v rs
(s1 :α -> bool) ⊆ (s2 :α -> bool) ⇒ REL_RESTRICT (R :α -> α -> bool) s1 $\subseteq_r$ REL_RESTRICT R s2
nsnd (n :num) ≤ n
∀(R :α -> α -> bool) (x :α) (y :α) (z :α). R x y ∧ R^{*} y z ⇒ R^{+} x z
∀(s :α -> bool). ¬(s ⊂ (∅ :α -> bool))
∀(s :α -> bool) (t :α -> bool). s DIFF t DIFF t = s DIFF t
∀(c :bool) (x :bool). (if c then x else F) ⇔ c ∧ x
(R :α -> β -> bool) $\circ_r$ ($= :α -> α -> bool) = R
∀(p :α # β) (f :α -> γ) (g :β -> δ). FST ((f ## g) p) = f (FST p)
∀(n :num). count n DELETE n = count n
∀(s :α -> bool) (t :α -> bool). s ∪ (s DIFF t) = s
PMATCH_ROW (p :β -> γ) (g :β -> bool) (r :β -> α) (v :γ) ≠ (NONE :α option) ⇒ PMATCH v (PMATCH_ROW p g r::(rs :(γ -> α option) list)) = r (@(x :β). PMATCH_ROW_COND p g v x)
∀(f :α -> β) (g :γ -> δ) (x :α) (y :γ). (f ## g) (x,y) = (f x,g y)
∀(n :num). ALL_DISTINCT (COUNT_LIST n)
MAP (f :α -> β) (l :α list) = [(x :β)] ⇔ ∃(x0 :α). l = [x0] ∧ x = f x0
∀(y :α -> β -> bool) (z :α -> β -> bool). y = z ⇔ y $\subseteq_r$ z ∧ z $\subseteq_r$ y
LIST_BIND ((l1 :β list) ++ (l2 :β list)) (f :β -> α list) = LIST_BIND l1 f ++ LIST_BIND l2 f
IDEM (SC :(α -> α -> bool) -> α -> α -> bool)
∀(n :num) (l2 :α list). n ≤ LENGTH l2 ⇒ ∀(l1 :α list). LASTN n (l1 ++ l2) = LASTN n l2
∀(x :α) (y :α). x ∈ {y} ⇔ x = y
∀(t1 :α) (t2 :α). (if T then t1 else t2) = t1
∀(o1 :α option) (o2 :α option) (f1 :α -> β option) (f2 :α -> β option). o1 = o2 ∧ (∀(x :α). o2 = SOME x ⇒ f1 x = f2 x) ⇒ OPTION_BIND o1 f1 = OPTION_BIND o2 f2
∀(L :num list). SUM L = SUM_ACC L (0 :num)
∀(R :α -> α -> bool). reflexive R^{T} ⇔ reflexive R
(∀(x :α) (y :β) (z :α). (if (P :bool) then (INR x :β + α) else (INL y :β + α)) = (INR z :β + α) ⇔ P ∧ z = x) ∧ (∀(x :γ) (y :δ) (z :δ). (if P then (INR x :δ + γ) else (INL y :δ + γ)) = (INL z :δ + γ) ⇔ ¬P ∧ z = y) ∧ (∀(x :ε) (y :ζ) (z :ε). (if P then (INL x :ε + ζ) else (INR y :ε + ζ)) = (INL z :ε + ζ) ⇔ P ∧ z = x) ∧ ∀(x :η) (y :θ) (z :θ). (if P then (INL x :η + θ) else (INR y :η + θ)) = (INR z :η + θ) ⇔ ¬P ∧ z = y
∀(h :α) (l1 :α list) (l2 :α list). LAST (l1 ++ h::l2) = LAST (h::l2)
trichotomous (STRORD (R :α -> α -> bool)) ⇔ trichotomous R
∀(l :α list) (x :α) (y :α) (i :num). MEM x (LUPDATE y i l) ⇔ i < LENGTH l ∧ x = y ∨ ∃(j :num). j < LENGTH l ∧ i ≠ j ∧ EL j l = x
numeral$iiSUC ZERO = BIT2 ZERO ∧ numeral$iiSUC (BIT1 (n :num)) = BIT1 (SUC n) ∧ numeral$iiSUC (BIT2 n) = BIT2 (SUC n)
¬((A :bool) ∨ (B :bool)) ⇒ F ⇔ (A ⇒ F) ⇒ ¬B ⇒ F
∀(R :α -> α -> bool) (f :β -> α). total R ⇒ total (inv_image R f)
numeral$texp_help (n :num) (0 :num) = (2 :num) ** (n + (1 :num))
BIGUNION (∅ :(α -> bool) -> bool) = (∅ :α -> bool)

∀(P :α -> β -> bool). (∃(x :α) (y :β). P x y) ⇔ ∃((x :α),(y :β)). P x y
∀(s :α -> bool) (t :α -> bool) (u :α -> bool). DISJOINT s t ∧ u ⊆ t ⇒ DISJOINT s u
PMATCH_ROW_COND_EX (i :α) (p :β -> α) (g :β -> bool) ⇔ ∃(x :β). i = p x ∧ g x
FILTER (P :α -> bool) ([] :α list) = ([] :α list) ∧ (∀(h :α). P h ⇒ FILTER P (h::(l :α list)) = h::FILTER P l) ∧ ∀(h :α). ¬P h ⇒ FILTER P (h::l) = FILTER P l
∀(c :bool) (x :bool). (if c then x else T) ⇔ c ⇒ x
∀(s :α -> bool) (t :α -> bool) (x :α). x ∈ s ∩ t ⇔ x ∈ s ∧ x ∈ t
IDEM (STRORD :(α -> α -> bool) -> α -> α -> bool)
(∀(x :α) (y :β). (P :α -> β -> bool) x y ⇒ (Q :α -> β -> bool) x y) ⇒ OPTREL P (x :α option) (y :β option) ⇒ OPTREL Q x y
(∀(x :num). pair_to_num (num_to_pair x) = x) ∧ ∀(x :num) (y :num). num_to_pair (pair_to_num (x,y)) = (x,y)
∀(n :num). (0 :num) < n ⇔ count n ≠ (∅ :num -> bool)
∀(ls :α list) (n :num). n < LENGTH ls ⇒ DROP n ls = EL n ls::DROP (n + (1 :num)) ls
(2 :num) * tri (n :num) = n * (n + (1 :num))
(x :α) ∈ BIGINTER (B :(α -> bool) -> bool) ⇔ ∀(P :α -> bool). P ∈ B ⇒ x ∈ P
∀(x :α) (s :α -> bool). s DELETE x ⊆ s
∀(s :α -> bool). FINITE s ⇒ ∀(x :α). x ∈ s ⇔ MEM x (SET_TO_LIST s)
∀(s1 :α -> bool) (s2 :α -> bool) (x :α). s1 ⊆ s2 ⇒ s1 DELETE x ⊆ s2 DELETE x
∀(s :α -> bool). s = (∅ :α -> bool) ∨ ∃(x :α) (t :α -> bool). s = x INSERT t ∧ x ∉ t
∀(s1 :α -> bool) (s2 :α -> bool). s1 ⊆ s2 ⇒ POW s1 ⊆ POW s2
MAX_SET (∅ :num -> bool) = (0 :num) ∧ MAX_SET {(e :num)} = e
OPTION_MAP (f :α -> β) (x :α option) = (option_CASE x (NONE :β option) ((SOME :β -> β option) ∘ f) :β option)
∀(R :α -> α -> bool) (x :α) (y :α). R^{*} x y ⇒ RC R x y ∨ R^{+} x y
∀(R :α -> α -> bool). (∀(x :α). R^{*} x x) ∧ ∀(x :α) (y :α) (z :α). R x y ∧ R^{*} y z ⇒ R^{*} x z
∀(x :α) (y :β). (INR y :α + β) ≠ (INL x :α + β)
∀(x :α) (s :α -> bool) (t :α -> bool). (x INSERT s) ∪ t = x INSERT s ∪ t
∀(n :num) (l :α list). n < LENGTH l ⇒ DROP n l = EL n l::DROP (SUC n) l
∀(s :α -> bool). FINITE s ⇔ s HAS_SIZE CARD s
∀(n :num) (l :α list). n ≤ LENGTH l ⇒ LENGTH (LASTN n l) = n
∀(opt :α option). IS_SOME opt ⇔ ∃(x :α). opt = SOME x
∀(x :α) (sos :(α -> bool) -> bool). x ∈ BIGUNION sos ⇔ ∃(s :α -> bool). x ∈ s ∧ s ∈ sos
∀(P :α -> bool) (Q :α -> bool). {x | P x ∨ Q x} = {x | P x} ∪ {x | Q x}
([] :α list) ∈ (s :α list -> bool) ⇒ longest_prefix s = ([] :α list)
∀(R :α -> α -> bool). SC (SC R) = SC R
∀(a :α -> bool) (b :β -> bool) (c :β -> bool). FUNSET a (b ∩ c) = FUNSET a b ∩ FUNSET a c
INFINITE 𝕌(:α) ⇔ ∀(s :α -> bool). FINITE s ⇒ ∃(x :α). x ∉ s
∀(l :α list list). LENGTH (FLAT l) = SUM (MAP (LENGTH :α list -> num) l)
∀(n :num) (m :num) (l :α list). m ≤ n ⇒ TAKE m l ++ TAKE (n − m) (DROP m l) = TAKE n l
∀(x :α) (l :α list). ¬NULL (SNOC x l)
INJ (f :α -> β) (s :α -> bool) (t :β -> bool) ⇒ FINITE s ⇒ CARD (IMAGE f s) = CARD s
∀(P :α list -> bool). P ([] :α list) ∧ (∀(l :α list). P l ⇒ ∀(a :α). P (a::l)) ⇒ ∀(l :α list). P l

∀(s :α -> bool) (t :α -> bool). s DIFF t ⊆ s
∀(l :α list). BUTLASTN (LENGTH l) l = ([] :α list)
∀(P :α -> β -> bool) (l1 :α list) (l2 :β list). LIST_REL P l1 l2 ⇒ LENGTH l1 = LENGTH l2
∀(f :α -> β) (s :β -> bool). PREIMAGE f (COMPL s) = COMPL (PREIMAGE f s)
symmetric (R :α -> α -> bool)^=
∀(P :α -> bool) (l :α list) (m :num). EVERY P l ⇒ EVERY P (DROP m l)
∀(s :α -> bool) (t :β -> bool). (∃(f :α -> β). BIJ f s t) ⇔ ∃(g :β -> α). BIJ g t s
∀(f :α -> β) (s :α -> bool) (t :β -> bool). BIJ f s t ⇒ t = IMAGE f s
∀(l1 :α list) (x :α) (l2 :α list). l1 ++ SNOC x l2 = SNOC x (l1 ++ l2)
CURRY (f :α # β -> γ) = CURRY (g :α # β -> γ) ⇔ f = g
∀(n :num) (l :α list). n ≤ LENGTH l ⇒ TAKE n l = BUTLASTN (LENGTH l − n) l
∀(x :α). 𝕌(:α) x
∀(m :num) (n :num). TAKE n (TAKE m (l :α list)) = TAKE (MIN n m) l
∀(f :α -> β) (s :α -> bool) (t :α -> bool). IMAGE f (s ∪ t) = IMAGE f s ∪ IMAGE f t
(∀(x :α) (y :α). (R :α -> α -> bool) x y ⇒ (f :α -> β) x = f y) ⇒ ∀(x :α) (y :α). R^{+} x y ⇒ f x = f y
∀(f :α -> β) (s :β -> bool) (t :β -> bool). PREIMAGE f (s DIFF t) = PREIMAGE f s DIFF PREIMAGE f t
∀(A :α -> bool) (B :α -> bool) (x :α). A ∪ B DELETE x = A DELETE x ∪ (B DELETE x)
(∀(l :α list). BUTLASTN (0 :num) l = l) ∧ ∀(n :num) (x :β) (l :β list). BUTLASTN (SUC n) (SNOC x l) = BUTLASTN n l
∀(R :α -> α -> bool) (x :α) (y :α). R x y ⇒ R$^{✡}$ x y
∀(n :num) (l :α list). n ≤ LENGTH l ⇒ TAKE n (REVERSE l) = REVERSE (LASTN n l)
∀(xs :α option list) (n :num) (f :α -> β) (h :α). LUPDATE (SOME (f h)) n (MAP (OPTION_MAP f) xs) = MAP (OPTION_MAP f) (LUPDATE (SOME h) n xs)
∀(x :α) (s :α -> bool). x ∈ s ⇒ x INSERT s DELETE x = s
∀(R :α -> α -> bool). reflexive R ⇒ RC R = R
(OPTION_BIND (p :α option) (f :α -> β option) = (NONE :β option) ⇔ p = (NONE :α option) ∨ ∃(x :α). p = SOME x ∧ f x = (NONE :β option)) ∧ (OPTION_BIND p f = SOME (y :β) ⇔ ∃(x :α). p = SOME x ∧ f x = SOME y)
∀(l1 :α list) (l2 :α list). set l1 ∪ set l2 = set (l1 ++ l2)
∀(R :α -> α -> bool) (x :α) (y :α). R$^{✡}$ x y ⇔ ∃(u :α). R$^{✡}$ x u ∧ R$^{✡}$ u y
MIN (0 :num) (x :num) = (0 :num) ∧ MIN x (0 :num) = (0 :num) ∧ MIN (NUMERAL x) (NUMERAL (y :num)) = NUMERAL (if x < y then x else y)
∀(c :bool) (x :bool) (x' :bool) (y :bool) (y' :bool). (x' ⇒ x) ∧ (y' ⇒ y) ⇒ (if c then x' else y') ⇒ if c then x else y
∀(P :α -> bool) (s :α -> bool) (t :α -> bool). (∀(x :α). x ∈ s ∪ t ⇒ P x) ⇔ (∀(x :α). x ∈ s ⇒ P x) ∧ ∀(x :α). x ∈ t ⇒ P x
PMATCH_ROW (p :β -> γ) (g :β -> bool) (r :β -> α) (i :γ) = (NONE :α option) ⇔ ∀(x :β). ¬PMATCH_ROW_COND p g i x
∀(f1 :α -> bool) (f2 :β -> α) (l :β list). FILTER f1 (MAP f2 l) = MAP f2 (FILTER (f1 ∘ f2) l)
∀(s :α -> bool) (t :α -> bool) (u :α -> bool). s ⊆ t ∧ t ⊆ u ⇒ s ⊆ u
(∀(R :α -> β -> bool) (x :α) (y :β option). OPTREL R (SOME x) y ⇔ ∃(z :β). y = SOME z ∧ R x z) ∧ ∀(R :γ -> δ -> bool) (x :γ option) (y :δ). OPTREL R x (SOME y) ⇔ ∃(z :γ). x = SOME z ∧ R z y

∀(m :num) (n :num). m < SUC n ⇒ m = n ∨ m < n
tri (n :num) = n * (n + (1 :num)) DIV (2 :num)
∀(f :α -> β -> β) (e :β) (x :α) (l :α list). FOLDR f e (SNOC x l) = FOLDR f (f x e) l
∀(l1 :α list) (l2 :β list). LENGTH l1 = LENGTH l2 ⇒ (ZIP (l1,l2) = ([] :(α # β) list) ⇔ l1 = ([] :α list) ∧ l2 = ([] :β list))
{x | x ∈ (y :α -> bool)} = y
∀(n :num). n ≠ (0 :num) ⇒ COUNT_LIST n = (0 :num)::MAP SUC (COUNT_LIST (n − (1 :num)))
∀(R :α -> α -> bool) (x :α) (y :α). WF R ⇒ R x y ⇒ x ≠ y
∀(s :α -> bool) (t :α -> bool) (u :α -> bool). s ∪ (t ∪ u) = s ∪ t ∪ u
∀(m :num) (n :num). m < SUC n ⇔ m = n ∨ m < n
∀(s :α -> bool) (t :α -> bool). s ⊂ t ⇔ ∃(x :α). x ∉ s ∧ x INSERT s ⊆ t
∀(x :α) (s :α -> bool) (t :α -> bool). s ⊆ t DELETE x ⇔ x ∉ s ∧ s ⊆ t
(∀(l :α list). DROP (0 :num) l = l) ∧ ∀(n :num) (x :α) (l :α list). DROP (SUC n) (x::l) = DROP n l
∀(n :num) (f :num -> α). (0 :num) < n ⇒ HD (GENLIST f n) = f (0 :num)
∀(x :α list) (y :β list). LIST_REL (R :α -> β -> bool) x y ⇒ LENGTH x = LENGTH y
(l1 :α list) ++ (l2 :α list) = [(e :α)] ⇔ l1 = [e] ∧ l2 = ([] :α list) ∨ l1 = ([] :α list) ∧ l2 = [e]
∀(b :α -> β) (s :α -> bool) (t :β -> bool) (x :α) (y :β). INJ b s t ∧ x ∉ s ∧ y ∉ t ⇒ INJ b⦇x ↦ y⦈ (x INSERT s) (y INSERT t)
∀(n :num) (x :α) (l :α list). ELL (SUC n) (SNOC x l) = ELL n l
∀(n :num) (m :num) (l :α list). n + m ≤ LENGTH l ⇒ ∀(x :α). SEG n m (SNOC x l) = SEG n m l
∀(s :α -> bool) (f :α -> β). (IMAGE f s = (∅ :β -> bool) ⇔ s = (∅ :α -> bool)) ∧ ((∅ :β -> bool) = IMAGE f s ⇔ s = (∅ :α -> bool))
∀(L :(α # β) list). UNZIP L = (MAP (FST :α # β -> α) L,MAP (SND :α # β -> β) L)
∀(p :α # β) (p' :α # β) (P :α -> bool) (P' :α -> bool) (Q :β -> bool) (Q' :β -> bool). p = p' ∧ (∀(x :α) (y :β). p' = (x,y) ⇒ (P x ⇔ P' x)) ∧ (∀(x :α) (y :β). p' = (x,y) ⇒ (Q y ⇔ Q' y)) ⇒ (PROD_ALL P Q p ⇔ PROD_ALL P' Q' p')
SET_TO_LIST (∅ :α -> bool) = ([] :α list)
∀(f :α -> β -> bool) (s :α -> bool) (t :γ -> bool). BIGUNION (IMAGE f s) × t = BIGUNION (IMAGE (λ(n :α). f n × t) s)
∀(s :α -> bool) (t :α -> bool) (u :α -> bool) (v :α -> bool). DISJOINT s t ∧ u ⊆ s ∧ v ⊆ t ⇒ DISJOINT u v
FST (p :α # β) = (x :α) ⇔ ∃(y :β). p = (x,y)
∀(x :α list) (y :α list). x ≼ y ⇒ LENGTH x ≤ LENGTH y
∀(x :α). one_CASE () x = x
∀(A :α -> bool) (B :β -> bool) (C :α -> bool) (D :β -> bool). A × B ∩ (C × D) = A ∩ C × (B ∩ D)
∀(x :α) (y :α). {y} x ⇔ x = y
WeakOrder ($RSUBSET :(α -> β -> bool) -> (α -> β -> bool) -> bool)
INFINITE 𝕌(:α) ⇔ ∃(f :α -> α). (∀(x :α) (y :α). f x = f y ⇒ x = y) ∧ ∃(y :α). ∀(x :α). f x ≠ y
∀(R :α -> α -> bool) (x :α) (z :α). R^+ x z ⇒ R x z ∨ ∃(y :α). R^+ x y ∧ R y z
∀(R :α -> α -> bool) (RTC' :α -> α -> bool). (∀(x :α). RTC' x x) ∧ (∀(x :α) (y :α) (z :α). R x y ∧ RTC' y z ⇒ RTC' x z) ⇒ ∀(a0 :α) (a1 :α). $R^{✡}$ a0 a1 ⇒ RTC' a0 a1
∀(x :α) (s :α -> bool). x INSERT s ≠ (∅ :α -> bool)

∀(f :α -> β). PREIMAGE f 𝕌(:β) = 𝕌(:α)
∀(P :α -> bool) (l :α list). EXISTS P l ⇔ FOLDR (λ(x :α) (l' :bool). P x ∨ l') F l
IDEM (RC :(α -> α -> bool) -> α -> α -> bool)
∀(X :α -> bool) (P :(α -> bool) -> bool). BIGUNION P ⊆ X ⇔ ∀(Y :α -> bool). Y ∈ P ⇒ Y ⊆ X
∀(f :α -> β) (x :α). IMAGE f {x} = {f x}
(($\mathbb{U}_r$:α -> β -> bool) $\subseteq_r$ (R :α -> β -> bool) ⇔ R = ($\mathbb{U}_r$:α -> β -> bool)) ∧ R $\subseteq_r$ ($\mathbb{U}_r$:α -> β -> bool)
∀(x :α + β). ISL x ∨ ISR x
transitive (R :α -> α -> bool) ⇔ R $\circ_r$ R $\subseteq_r$ R
¬F ⇔ T
¬((p :bool) ∨ (q :bool)) ⇒ ¬p
WF $<
∀(f :α -> β) (g :β -> γ) (s :α -> bool) (t :β -> bool) (u :γ -> bool). BIJ f s t ∧ BIJ g t u ⇒ BIJ (g ∘ f) s u
∀(s :α -> bool) (t :α -> bool). INFINITE s ∧ FINITE t ⇒ s DIFF t ≠ (∅ :α -> bool)
∀(f :α -> β) (v :γ) (p :β -> γ) (g :β -> bool) (p' :α -> γ) (g' :α -> bool). ¬PMATCH_ROW_COND_EX v p g ⇒ (∀(x :α). p' x = p (f x)) ⇒ (PMATCH_ROW_COND_EX v p' g' ⇔ PMATCH_ROW_COND_EX v p' (λ(x :α). g' x ∧ ¬g (f x)))
∀(f :α -> β) (l :α list list). MAP f (FLAT l) = FLAT (MAP (MAP f) l)
∀(m :num) (n :num). SUC m = SUC n ⇒ m = n
∀(s :α -> bool) (x :α). x ∉ s ⇒ s DELETE x = s
∀(l1 :α list) (x :α) (l2 :α list). SNOC x l1 ++ l2 = l1 ++ x::l2
pairwise (R :α -> α -> bool) ((s1 :α -> bool) ∪ (s2 :α -> bool)) ⇔ pairwise R s1 ∧ pairwise R s2 ∧ ∀(x :α) (y :α). x ∈ s1 ∧ y ∈ s2 ⇒ R x y ∧ R y x
∀(s :α -> bool) (t :α -> bool). s ⊆ t ⇔ s ∩ t = s
∀(l :α list). REVERSE l = FOLDR (SNOC :α -> α list -> α list) ([] :α list) l
∀(x :α # β). ∃(q :α) (r :β). x = (q,r)
∀(P :α -> bool) (Q :α -> bool). BIGINTER {P; Q} = P ∩ Q
∀(m :α) (n :α). (λ(x :α) (y :α). y = (f :α -> α) x)* (f m) n ⇔ (λ(x :α) (y :α). y = f x)$^{+}$ m n
∀(xs :α list) (n :num) (k :num). DROP n (TAKE k xs) = TAKE (k − n) (DROP n xs)
∀(n :num) (l2 :α list). n < LENGTH l2 ⇒ ∀(l1 :α list). ELL n (l1 ++ l2) = ELL n l2
($\emptyset_r$:α -> α -> bool) $\subseteq_r$ (R :α -> α -> bool) ∧ (R $\subseteq_r$ ($\emptyset_r$:α -> α -> bool) ⇔ R = ($\emptyset_r$:α -> α -> bool))
∀(P :α -> bool) (ls :α list). EXISTS ($¬ ∘ P) ls ⇒ LENGTH (FILTER P ls) < LENGTH ls
LIST_BIND (l :α list) (λ(x :α). [x]) = l
(some(x :bool). ¬x) = SOME F
∀(x :α) (s :α -> bool) (t :α -> bool). s ⊆ x INSERT t ⇔ s DELETE x ⊆ t
∀(P :α + β -> bool). (∀(x :α). P (INL x :α + β)) ∧ (∀(y :β). P (INR y :α + β)) ⇒ ∀(s :α + β). P s
∀(P :α -> bool) (a :α) (s :α -> bool). (∀(x :α). x ∈ a INSERT s ⇒ P x) ⇔ P a ∧ ∀(x :α). x ∈ s ⇒ P x
(∀(x :α). (P :α -> bool) x ⇒ (Q :α -> bool) x) ⇒ EVERY P (l :α list) ⇒ EVERY Q l
(LIST_REL (R :α -> β -> bool) ([] :α list) (y :β list) ⇔ y = ([] :β list)) ∧ (LIST_REL R (x :α list) ([] :β list) ⇔ x = ([] :α list))

∀(l :bool list). OR_EL l ⇔ FOLDL $\/ F l
∀(R :α -> α -> bool). antisymmetric (RC R) ⇔ antisymmetric R
(∀(x :α) (y :α). (R :α -> α -> bool) x y ⇒ (f :α -> β) x = f y) ⇒ ∀(x :α) (y :α). R^* x y ⇒ f x = f y
∀(x :α) (s :α -> bool). PREIMAGE (K x :β -> α) s = if x ∈ s then 𝕌(:β) else (∅ :β -> bool)
∀(xs :α list) (i :num) (n :num) (x :α). oEL n (LUPDATE x i xs) = if i ≠ n then oEL n xs else if i < LENGTH xs then SOME x else (NONE :α option)
∀(R :α -> α -> bool). reflexive R^*
(list_CASE (x :α list) (v :β) (f :α -> α list -> β) :β) = (v' :β) ⇔ x = ([] :α list) ∧ v = v' ∨ ∃(a :α) (l :α list). x = a::l ∧ f a l = v'
∀(n :num) (l :α list). n ≤ LENGTH l ⇒ LASTN n (REVERSE l) = REVERSE (TAKE n l)
∀(s :α -> bool) (t :α -> bool). s ⊆ t ∧ t ⊆ s ⇔ s = t
(p :bool) ∧ (q :bool) ⇒ q
∀(P :α -> bool) (ls :α list). dropWhile P ls = ([] :α list) ⇔ EVERY P ls
∀(p :α -> bool) (q :α -> bool). p ∩ q ∪ COMPL p ∩ q = q
((p :bool) ⇔ (q :bool) ⇒ (r :bool)) ⇔ (p ∨ q) ∧ (p ∨ ¬r) ∧ (¬q ∨ r ∨ ¬p)
(pair_CASE ((x :β),(y :γ)) (f :β -> γ -> α) :α) = f x y
∀(s1 :(α -> bool) -> bool) (s2 :(α -> bool) -> bool). BIGUNION (s1 ∪ s2) = BIGUNION s1 ∪ BIGUNION s2
FINITE (s :α -> bool) ∧ SURJ (f :α -> β) s (t :β -> bool) ⇒ FINITE t
OPTION_MAP (f :γ -> β) (OPTION_MAP (g :α -> γ) (x :α option)) = OPTION_MAP (f ∘ g) x
∀(f :num -> α) (n :num). set (GENLIST f n) = IMAGE f (count n)
∀(m :num) (n :num). m = n ∨ m < n ⇒ m < SUC n
REVERSE (l :α list) = [(e :α)] ⇔ l = [e]
∀(n :num). SUM_SET {n} = n
∀(f1 :num -> num -> α -> bool) (f2 :num -> num -> α -> bool). ind_type$INJF f1 = ind_type$INJF f2 ⇔ f1 = f2
LIST_REL (R :α -> β -> bool) ((h :α)::(t :α list)) (xs :β list) ⇔ ∃(h' :β) (t' :β list). xs = h'::t' ∧ R h h' ∧ LIST_REL R t t'
∀(n :num) (l :α list). n ≤ LENGTH l ⇒ ∀(x :α). DROP n (SNOC x l) = SNOC x (DROP n l)
∀(x :α) (y :β). FST (x,y) = x
is_measure_maximal (m :α -> num) {(x :α)} (y :α) ⇔ y = x
∀(P :α -> bool) (l :α list). EXISTS P l ⇔ ¬EVERY (λ(x :α). ¬P x) l
∀(x :α option). IS_NONE x ⇔ x = (NONE :α option)
∀(x :bool) (x' :bool) (y :bool) (y' :bool). (y ⇒ x ⇒ x') ∧ (x' ⇒ y ⇒ y') ⇒ x ∧ y ⇒ x' ∧ y'
(R :α -> α -> bool) equiv_on (s :α -> bool) ⇒ (∅ :α -> bool) ∉ partition R s
∀(R :α -> α -> bool) (x :α) (y :α). R x y ⇒ RC R x y
∀(a :α list) (b :α list) (c :α list). a ≼ b ++ c ⇒ a ≼ b ∨ b ≼ a
∀(n :num). set (COUNT_LIST n) = count n
∀(x :α option option) (y :α). OPTION_JOIN x = SOME y ⇔ x = SOME (SOME y)
¬(A :bool) ⇒ F ⇔ A
∀(s :α -> bool). REST s ⊆ s
∀(R :α -> α -> bool) (f :β -> α). symmetric R ⇒ symmetric (inv_image R f)
∀(R :α -> α -> bool). RC (RC R) = RC R

PRE (0 :num) = (0 :num) ∧ ∀(m :num). PRE (SUC m) = m
∀(l :α list). TAKE (LENGTH l) l = l
∀(f :α -> β) (x :α # β) (ls :α list). ALL_DISTINCT ls ⇒ (MEM x (ZIP (ls,MAP f ls)) ⇔ MEM (FST x) ls ∧ SND x = f (FST x))
LIST_BIND (l :α list list) (λ(x :α list). x) = FLAT l ∧ LIST_BIND l (I :α list -> α list) = FLAT l
∀(v_old :α) (v_new :β) (rows_old :(α -> γ option) list) (rows_new :(β -> γ option) list) (r_old :α -> γ option). r_old v_old = (NONE :γ option) ⇒ PMATCH v_old rows_old = PMATCH v_new rows_new ⇒ PMATCH v_old (r_old::rows_old) = PMATCH v_new rows_new
∀(x1 :α option) (x2 :α option) (y1 :β option) (y2 :β option) (f1 :α -> β -> γ) (f2 :α -> β -> γ). x1 = x2 ∧ y1 = y2 ∧ (∀(x :α) (y :β). x2 = SOME x ∧ y2 = SOME y ⇒ f1 x y = f2 x y) ⇒ OPTION_MAP2 f1 x1 y1 = OPTION_MAP2 f2 x2 y2
∀(l1 :α list) (l2 :α list). l1 = REVERSE l2 ⇔ l2 = REVERSE l1
∀(x :α) (y :β) (l :(α # β) list). MEM (x,y) l ⇒ ∃(z :α # β). x = FST z ∧ MEM z l
$? (UNCURRY (λ(x :α). (P :α -> β -> bool) x)) ⇔ ∃(x :α). $? (P x)
∀(s :α -> bool). s ⊆ 𝕌(:α)
∀(f :α -> num) (g :α -> num) (ls :α list). SUM (MAP (λ(x :α). f x + g x) ls) = SUM (MAP f ls) + SUM (MAP g ls)
(∀(p :α # β). (P :α -> β -> bool) (FST p) (SND p)) ⇔ ∀(p1 :α) (p2 :β). P p1 p2
∀(P :α -> bool) (L :α list) (x :α). MEM x (FILTER P L) ⇔ P x ∧ MEM x L
∀(j :num) (n :num) (h :α) (t :α list). (0 :num) < j ∧ n + j ≤ LENGTH t + (1 :num) ⇒ SEG n j (h::t) = SEG n (j − (1 :num)) t
∀(s :α -> bool) (t :α -> bool). s DIFF t = s ∩ COMPL t
($PSUBSET :(α -> bool) -> (α -> bool) -> bool)✡ = ($SUBSET :(α -> bool) -> (α -> bool) -> bool)
∀(h :α) (t :α list). TL (h::t) = t
∀(l :α list). l ≠ ([] :α list) ⇒ LENGTH (FRONT l) = PRE (LENGTH l)
∀(l :α list). l = ([] :α list) ∨ ∃(h :α) (t :α list). l = h::t
((p :bool) ⇔ ((q :bool) ⇔ (r :bool))) ⇔ (p ∨ q ∨ r) ∧ (p ∨ ¬r ∨ ¬q) ∧ (q ∨ ¬r ∨ ¬p) ∧ (r ∨ ¬q ∨ ¬p)
𝕌(:α) ≠ (∅ :α -> bool)
(((f :(α -> β) option) <*> SOME (x :α)) :β option) = ((SOME (λ(f :α -> β). f x) <*> f) :β option)
((a :α),(b :β)) ∈ {((x :α),y) | (P :β -> bool) y} ⇔ P b ∧ a = x
∀(x :α + β). ¬ISL x ⇔ ISR x
∀(n :num). LENGTH (FLAT (REPLICATE n (ls :α list))) = n * LENGTH ls
LIST_BIND (MAP (f :γ -> β) (l :γ list)) (g :β -> α list) = LIST_BIND l (g ∘ f)
∀(f :α -> β). (∀(x :α) (y :α). f x = f y ⇒ x = y) ⇒ ∀(s :α -> bool). INFINITE s ⇒ INFINITE (IMAGE f s)
∀(n :num) (f :num -> α). NULL (GENLIST f n) ⇔ n = (0 :num)
(∀(x :α) (y :α). (R :α -> α -> bool) x y ⇒ (Q :α -> α -> bool) x y) ⇒ RC R (x :α) (y :α) ⇒ RC Q x y
∀(P :α -> bool) (Q :α -> bool). P ⊆ Q ⇒ ∀(x :α). x ∈ P ⇒ x ∈ Q
∀(x :α) (s :α -> bool). x INSERT x INSERT s = x INSERT s
∀(l :α list). REVERSE (REVERSE l) = l

ind_type$ISO (λ(x :α). x) (λ(x :α). x)
∀(s :α -> bool) (t :α -> bool). s = t ⇔ s ⊆ t ∧ t ⊆ s
∀(R :α -> α -> bool) (P :α -> α -> bool). (∀(x :α). P x x) ∧ (∀(x :α) (y :α) (z :α). P x y ∧ R y z ⇒ P x z) ⇒ ∀(x :α) (y :α). R^{*} x y ⇒ P x y
EVERY (P :α -> bool) (FLAT (ls :α list list)) ⇔ EVERY (EVERY P) ls
(∀(l :α list). LASTN (0 :num) l = ([] :α list)) ∧ ∀(n :num) (x :β) (l :β list). LASTN (SUC n) (SNOC x l) = SNOC x (LASTN n l)
((∀(n :num). ¬(P :num -> bool) n) ⇒ (Q :num option -> bool) (NONE :num option)) ∧ (∀(n :num). P n ∧ (∀(m :num). m < n ⇒ ¬P m) ⇒ Q (SOME n)) ⇒ Q ($OLEAST P)
(∀(l :α list). l ++ ([] :α list) = l) ∧ ∀(l :β list). ([] :β list) ++ l = l
∀(c :bool) (l :α list). EVERY (λ(x :α). c) l ⇔ l = ([] :α list) ∨ c
∀(s :α -> bool) (t :α -> bool). s ≠ t ⇔ ∃(x :α). x ∈ t ⇔ x ∉ s
PMATCH (v :β) (rows :(β -> α option) list) = PMATCH v (SNOC (PMATCH_ROW (λ(_0 :β). _0) (λ(_0 :β). T) (λ(_0 :β). (ARB :α))) rows)
∀(x :α # β) (l :(α # β) list). UNZIP (SNOC x l) = (SNOC (FST x) (FST (UNZIP l)),SNOC (SND x) (SND (UNZIP l)))
∀(X :α). (P :bool) ⇔ Case X ⇒ P
(∀(l :α list). (P :α list -> bool) l) ⇔ P ([] :α list) ∧ ∀(h :α) (t :α list). P (h::t)
∀(f :α -> β). (∀(s :β -> bool). BIJ f (∅ :α -> bool) s ⇔ s = (∅ :β -> bool)) ∧ ∀(s :α -> bool). BIJ f s (∅ :β -> bool) ⇔ s = (∅ :α -> bool)
∀(P :α -> bool) (f :β -> α) (s :β -> bool). (∀(y :α). y ∈ IMAGE f s ⇒ P y) ⇔ ∀(x :β). x ∈ s ⇒ P (f x)
∀(s :α -> bool). INJ (λ(x :α). x) s s
∀(f :α -> β) (s :α -> bool) (t :β -> bool). INJ f s t ⇒ ∀(e :α). e ∈ s ⇒ INJ f (s DELETE e) (t DELETE f e)
∀(P :α -> bool) (l :α list). EVERY (λ(x :α). ¬P x) l ⇒ SPLITP P l = (l,([] :α list))
∀(x :α -> bool) (y :bool). x ⊆ (K y :α -> bool) ⇔ x ⊆ (∅ :α -> bool) ∨ y
∀(a1 :α) (a2 :α). ind_type$INJA a1 = ind_type$INJA a2 ⇔ a1 = a2
∀(R :α -> α -> bool). $(SC\ R)^{T}$ = SC R ∧ SC R^{T} = SC R
∀(s :α -> bool) (P :(α -> bool) -> bool). BIGUNION (s INSERT P) = s ∪ BIGUNION P
∀(t :bool). T ⇒ t ⇔ t
∀(B :α -> bool) (C :α -> α) (R :α -> α -> bool). WF R ∧ (∀(s :α). B s ⇒ R (C s) s) ⇒ ∀(P :α -> bool). (∀(s :α). (B s ⇒ P (C s)) ⇒ P s) ⇒ ∀(v :α). P v
WF ($\emptyset_r$:α -> α -> bool)
ZIP (([] :γ list),([] :δ list)) = ([] :(γ # δ) list) ∧ ∀(x1 :α) (l1 :α list) (x2 :β) (l2 :β list). ZIP (x1::l1,x2::l2) = (x1,x2)::ZIP (l1,l2)
∀(x :α) (L :α list list). MEM x (FLAT L) ⇔ ∃(l :α list). MEM l L ∧ MEM x l
GSPEC (f :β -> α # bool) = IMAGE ((FST :α # bool -> α) ∘ f) ((SND :α # bool -> bool) ∘ f)
∀(adjacent' :α list -> α -> α -> bool). (∀(a :α) (b :α) (t :α list). adjacent' (a::b::t) a b) ∧ (∀(a :α) (b :α) (h :α) (t :α list). adjacent' t a b ⇒ adjacent' (h::t) a b) ⇒ ∀(a0 :α list) (a1 :α) (a2 :α). adjacent a0 a1 a2 ⇒ adjacent' a0 a1 a2
∀(m :num) (n :num) (l :α list). n + m ≤ LENGTH l ⇒ BUTLASTN n (BUTLASTN m l) = BUTLASTN (n + m) l
trichotomous (RC (R :α -> α -> bool)) ⇔ trichotomous R

∀(n :num) (l :α list). n ≤ LENGTH l ⇒ BUTLASTN n (REVERSE l) = REVERSE (DROP n l)
∀(s :α -> bool) (t :α -> bool). s DIFF t = (∅ :α -> bool) ⇔ s ⊆ t
∀(s :α -> bool) (t :α -> bool). s ⊆ t ⇒ t ∩ s = s
∀(f :β -> γ) (g :α -> β). MAP (f ∘ g) = MAP f ∘ MAP g
(∀(x :unit). (P :unit -> bool) x) ⇔ P ()
∀(f :'z -> 'z). INVOL f ⇒ ∀(a :'z) (b :'z). f a = b ⇔ a = f b
OPTION_CHOICE (m1 :α option) (m2 :α option) = (NONE :α option) ⇔ m1 = (NONE :α option) ∧ m2 = (NONE :α option)
∀(l :α list) (n :num). LENGTH l = SUC n ⇔ ∃(h :α) (l' :α list). LENGTH l' = n ∧ l = h::l'
∀(R :α -> α -> bool) (x :α) (y :α). R^{*} x y ⇔ x = y ∨ ∃(u :α). R^{*} x u ∧ R u y
(∀(s :α -> bool) (t :α -> bool). s ∩ t ⊆ s) ∧ ∀(s :α -> bool) (t :α -> bool). t ∩ s ⊆ s
reflexive ($SUBSET :(α -> bool) -> (α -> bool) -> bool)
∀(xs :α list) (n :num) (y :α). oEL n xs = SOME y ⇔ n < LENGTH xs ∧ y = EL n xs
∀(s :β -> bool). FUNSET (∅ :α -> bool) s = 𝕌(:α -> β)
∀(l1 :α list list) (l2 :α list list). FLAT (l1 ++ l2) = FLAT l1 ++ FLAT l2
FLAT ([] :α list list) = ([] :α list) ∧ FLAT (([] :β list)::(t :β list list)) = FLAT t ∧ FLAT (((h :γ)::(t1 :γ list))::(t2 :γ list list)) = h::FLAT (t1::t2)
∀(P :α -> bool) (x :α) (l :α list). FILTER P (SNOC x l) = if P x then SNOC x (FILTER P l) else FILTER P l
∀(x :α). (∅ :α -> bool) ≠ {x}
(∃(l :α list). (P :α list -> bool) l) ⇔ P ([] :α list) ∨ ∃(h :α) (t :α list). P (h::t)
ind_type$ISO (f :α -> β) (g :β -> α) ⇒ (∀(P :α -> bool). (∀(x :α). P x) ⇔ ∀(x :β). P (g x)) ∧ (∀(P :α -> bool). (∃(x :α). P x) ⇔ ∃(x :β). P (g x)) ∧ ∀(a :α) (b :β). a = g b ⇔ f a = b
∀(f :α -> β -> β) (b :β). ITSET f (∅ :α -> bool) b = b
(∀(x :α) (xs :α list). FRONT (x::xs) = ([] :α list) ⇔ xs = ([] :α list)) ∧ (∀(x :α) (xs :α list). ([] :α list) = FRONT (x::xs) ⇔ xs = ([] :α list)) ∧ ∀(x :α) (xs :α list). NULL (FRONT (x::xs)) ⇔ NULL xs
(0 :num) < SUC (0 :num)
∀(e :α) (l :α list) (h :α). MEM e l ∧ e ≠ LAST (h::l) ⇒ MEM e (FRONT (h::l))
∀(f :α -> β -> α) (e :α) (P :β -> bool) (l :β list). FOLDL f e (FILTER P l) = FOLDL (λ(x :α) (y :β). if P y then f x y else x) e l
∀(x :α list). ([] :α list) ≼ x ∧ (x ≼ ([] :α list) ⇔ x = ([] :α list))
∀(x :α). FINITE {x}
∀(f :(α -> bool) -> bool) (g :(α -> bool) -> bool). f ⊆ g ⇒ BIGUNION f ⊆ BIGUNION g
∀(l1 :α list) (l2 :α list). REVERSE l1 = REVERSE l2 ⇔ l1 = l2
∀(a1 :α list) (a0 :α). ([] :α list) ≠ a0::a1
DROP (0 :num) (l :α list) = l
∀(l :α list) (n1 :num) (n2 :num). ALL_DISTINCT l ∧ n1 < LENGTH l ∧ n2 < LENGTH l ⇒ (EL n1 l = EL n2 l ⇔ n1 = n2)
∀(ls :α list list). set (FLAT ls) = BIGUNION (set (MAP (set :α list -> α -> bool) ls))
∀(s :α -> bool). FINITE s ⇒ ∀(x :α). MEM x (SET_TO_LIST s) ⇔ x ∈ s
∀(f :α -> γ) (g :β -> δ) (z :α + β). SUM_MAP f g z = (sum_CASE z ((INL :γ -> γ + δ) ∘ f) ((INR :δ -> γ + δ) ∘ g) :γ + δ)
(∅ :α -> bool) ≠ 𝕌(:α)

<table>
<tr><td>∀(ls :α list) (n :num). ALL_DISTINCT ls ⇒ ALL_DISTINCT (DROP n ls)</td></tr>
<tr><td>∀(s :α -> bool) (t :α -> bool). DISJOINT s t ⇔ ¬∃(x :α). x ∈ s ∧ x ∈ t</td></tr>
<tr><td>∀(P :α -> bool) (Q :α -> bool). {x | P x ∧ Q x} = {x | P x} ∩ {x | Q x}</td></tr>
<tr><td>∀(P :α -> bool) (l :α list). FILTER P l ≠ ([] :α list) ⇔ ∃(x :α). MEM x l ∧ P x</td></tr>
<tr><td>∀(n :num) (l :α list). TAKE n l ++ DROP n l = l</td></tr>
<tr><td>∀(x :α) (s :α -> bool). COMPL s x ⇔ x ∉ s</td></tr>
<tr><td>∀(l :α list). LENGTH (REVERSE l) = LENGTH l</td></tr>
<tr><td>MAP (f :num -> α) (COUNT_LIST (n :num)) = GENLIST f n</td></tr>
<tr><td>{(x,y) | (P :α -> β -> bool) x y} = UNCURRY P</td></tr>
<tr><td>∀(s :α -> bool) (x :α) (x' :α). (x ∈ s ⇔ x' ∈ s) ⇔ (x ∈ s DELETE x' ⇔ x' ∈ s DELETE x)</td></tr>
<tr><td>(∀(x :α). (P :α -> bool) x ⇒ (P' :α -> bool) x) ∧ (∀(y :β). (Q :β -> bool) y ⇒ (Q' :β -> bool) y) ⇒ SUM_ALL P Q (s :α + β) ⇒ SUM_ALL P' Q' s</td></tr>
<tr><td>∀(p :bool) (q :bool) (m :bool). (stmarker m ∧ p ⇔ p ∧ stmarker m) ∧ (p ∧ q ∧ stmarker m ⇔ (p ∧ q) ∧ stmarker m) ∧ ((p ∧ stmarker m) ∧ q ⇔ (p ∧ q) ∧ stmarker m)</td></tr>
<tr><td>(LEAST(n :num). n = (x :num)) = x ∧ (LEAST(n :num). x = n) = x</td></tr>
<tr><td>∀(f :α -> β -> α) (e :α) (l :β list). FOLDL f e (REVERSE l) = FOLDR (λ(x :β) (y :α). f y x) e l</td></tr>
<tr><td>(Case (X :α) ∧ (Y :bool) ⇔ Y) ∧ (Y ∧ Case X ⇔ Y) ∧ (Case X ⇒ Y ⇔ Y)</td></tr>
<tr><td>∀(s :α -> bool) (t :α -> bool) (u :α -> bool). s ∪ t ⊆ u ⇔ s ⊆ u ∧ t ⊆ u</td></tr>
<tr><td>∀(pl :(α # β) list). LENGTH (FST (UNZIP pl)) = LENGTH pl ∧ LENGTH (SND (UNZIP pl)) = LENGTH pl</td></tr>
<tr><td>∀(p :α -> bool). p (UNIV_POINT p) ⇔ ∀(x :α). p x</td></tr>
<tr><td>DATATYPE ((list :α list -> (α -> α list -> α list) -> bool) ([] :α list) (CONS :α -> α list -> α list))</td></tr>
<tr><td>LIST_BIND (LIST_BIND (l :γ list) (g :γ -> β list)) (f :β -> α list) = LIST_BIND l (flip (LIST_BIND :β list -> (β -> α list) -> α list) f ∘ g)</td></tr>
<tr><td>∀(l :bool list). AND_EL l ⇔ FOLDR $/\ T l</td></tr>
<tr><td>(x :α list) ≼ ([] :α list) ⇔ x = ([] :α list)</td></tr>
<tr><td>∀(m :num) (n :num). m ∈ count n ⇔ m < n</td></tr>
<tr><td>∀(l :α list) (i :num). MAP (f :α -> β) (DROP i l) = DROP i (MAP f l)</td></tr>
<tr><td>∀(s1 :α -> bool) (s2 :α -> bool) (s3 :α -> bool). s1 ⊆ s2 DIFF s3 ⇔ s1 ⊆ s2 ∧ DISJOINT s1 s3</td></tr>
<tr><td>∀(m :num) (n :num). SUC m = SUC n ⇔ m = n</td></tr>
<tr><td>∀(R :α -> α -> bool). transitive R^{+}</td></tr>
<tr><td>∀(n :num) (m :num) (l :α list). n + m ≤ LENGTH l ⇒ DROP n (DROP m l) = DROP (n + m) l</td></tr>
<tr><td>∀(s :α -> bool). s = (∅ :α -> bool) ⇔ DISJOINT s s</td></tr>
<tr><td>∀(x :α) (y :β). {x} × {y} = {(x,y)}</td></tr>
<tr><td>(∀(x :α). (R :α -> α -> bool) x x) ⇒ ∀(x :α option). OPTREL R x x</td></tr>
<tr><td>(0 :num) ⊗ (0 :num) = (0 :num)</td></tr>
<tr><td>∀(v :unit). v = ()</td></tr>
<tr><td>∀(ss :α + β). (∃(x :α). ss = (INL x :α + β)) ∨ ∃(y :β). ss = (INR y :α + β)</td></tr>
<tr><td>∀(R :α -> α -> bool). (∀(P :α -> bool). (∀(x :α). (∀(y :α). R y x ⇒ P y) ⇒ P x) ⇒ ∀(x :α). P x) ⇒ WF R</td></tr>
<tr><td>∀(x :α). {x} DELETE x = (∅ :α -> bool)</td></tr>
<tr><td>∀(s :α -> bool). INFINITE s ⇔ ∀(t :α -> bool). FINITE t ⇒ t ⊆ s ⇒ t ⊂ s</td></tr>
<tr><td>∀(R :α -> α -> bool). diamond R ⇒ diamond (RC R)</td></tr>
</table>

OPTION_MCOMP (g :γ -> δ option) (SOME :γ -> γ option) = g ∧ OPTION_MCOMP (SOME :β -> β option) (f :α -> β option) = f
∀(x :α) (y :β). SND (x,y) = y
∀(n :num) (l :α list). n ≤ LENGTH l ⇒ ∀(x :α). TAKE n (SNOC x l) = TAKE n l
∀(f :α -> β) (s :α -> bool) (t :β -> bool) (s0 :α -> bool) (t0 :β -> bool). INJ f s t ∧ s0 ⊆ s ∧ t ⊆ t0 ⇒ INJ f s0 t0
∀(n :num) (x :α). REPLICATE n x = GENLIST (K x :num -> α) n
(\$= :β -> β -> bool) $\circ_r$ (R :α -> β -> bool) = R
∀(R :α -> α -> bool). transitive R ⇒ transitive (RC R)
∀(t :bool). t ⇒ t ⇔ T
∀(v :α) (v' :α) (rows :(α -> β option) list) (rows' :(α -> β option) list) (r :α -> β option) (r' :α -> β option). v = v' ∧ r v' = r' v' ∧ PMATCH v' rows = PMATCH v' rows' ⇒ PMATCH v (r::rows) = PMATCH v' (r'::rows')
((A :α -> bool) ∩ (B :α -> bool) = A ⇔ A ⊆ B) ∧ (A ∩ B = B ⇔ B ⊆ A)
LENGTH (STRONGEST_REDUNDANT_ROWS_INFO (v :α) (rows :(α -> β option) list)) = LENGTH rows
(R1 :α -> β -> bool) $\subseteq_r$ (R2 :α -> β -> bool) ∧ (S1 :γ -> α -> bool) $\subseteq_r$ (S2 :γ -> α -> bool) ⇒ R1 $\circ_r$ S1 $\subseteq_r$ R2 $\circ_r$ S2
∀(opt1 :α option) (opt2 :α option) (f1 :α -> β) (f2 :α -> β). opt1 = opt2 ∧ (∀(x :α). opt2 = SOME x ⇒ f1 x = f2 x) ⇒ OPTION_MAP f1 opt1 = OPTION_MAP f2 opt2
∀(m :num) (n :num). m < n ⇒ m ≠ n
∀(x :α) (y :α). (R :α -> α -> bool)✡ x y ⇒ R^= x y
STRORD (R :α -> α -> bool) = R $\cap_r$ RCOMPL (\$= :α -> α -> bool)
∀(x :α) (s :α -> bool). s DELETE x DELETE x = s DELETE x
∀(R :α -> α -> bool). WeakOrder R ⇒ RC (STRORD R) = R
CARD (∅ :α -> bool) = (0 :num)
∀(m :num) (n :num). m = n ⇒ ¬(m < n)
∀(R1 :α -> α -> bool) (R2 :α -> α -> bool). irreflexive R2 ∧ R1 $\subseteq_r$ R2 ⇒ irreflexive R1
∀(s :α -> bool) (x :α). x ∈ s ⇒ (s DELETE x = (∅ :α -> bool) ⇔ s = {x})
∀(n :num) (m :num) (l :α list). DROP n (DROP m l) = DROP (n + m) l
∀(opt :α option) (opt' :α option) (P :α -> bool) (P' :α -> bool). opt = opt' ∧ (∀(x :α). opt' = SOME x ⇒ (P x ⇔ P' x)) ⇒ (OPTION_ALL P opt ⇔ OPTION_ALL P' opt')
𝕌(:bool) = {T; F}
((A :α -> bool) ∪ (B :α -> bool)) ∩ A = A ∧ (B ∪ A) ∩ A = A ∧ A ∩ (A ∪ B) = A ∧ A ∩ (B ∪ A) = A
∀(f' :α -> β -> γ) (f :α -> β -> γ) (M' :α # β) (M :α # β). M = M' ∧ (∀(x :α) (y :β). M' = (x,y) ⇒ f x y = f' x y) ⇒ UNCURRY f M = UNCURRY f' M'
∀(v :α) (rows1a :(α -> β option) list) (rows1b :(α -> β option) list) (rows2a :(α -> β option) list) (rows2b :(α -> β option) list). PMATCH_EQUIV_ROWS v rows1a rows1b ⇒ PMATCH_EQUIV_ROWS v rows2a rows2b ⇒ PMATCH_EQUIV_ROWS v (rows1a ++ rows2a) (rows1b ++ rows2b)
∀(t :bool). t ∨ T ⇔ T
∀(x :α) (y :α) (s :α -> bool). (x INSERT s) DELETE y = if x = y then s DELETE y else x INSERT s DELETE y

∀(R :α -> α -> bool) (Q :α -> bool) (b :α). Q b ∧ (∀(x :α) (y :α). R x y ∧ Q y ⇒ Q x) ⇒ ∀(x :α). R^{*} x b ⇒ Q x
∀(a :α list) (b :β list) (c :α list) (d :β list). LENGTH a = LENGTH b ∧ LENGTH c = LENGTH d ⇒ ZIP (a,b) ++ ZIP (c,d) = ZIP (a ++ c,b ++ d)
∀(x :α) (f :α -> num -> α). (∀(n :num). PRIM_REC_FUN x f (0 :num) n = x) ∧ ∀(m :num) (n :num). PRIM_REC_FUN x f (SUC m) n = f (PRIM_REC_FUN x f m (PRE n)) n
(∀(s :α). (P :α -> bool) s ⇔ (Q :α -> bool) s) ⇒ ((∀(s :α). P s) ⇔ ∀(s :α). Q s)
∀(x :α -> bool) (y :α -> bool) (z :α -> bool). x DIFF (y ∪ z) = x DIFF y DIFF z
(OLEAST(n :num). n = (x :num)) = SOME x ∧ (OLEAST(n :num). x = n) = SOME x ∧ (OLEAST(n :num). F) = (NONE :num option) ∧ (OLEAST(n :num). T) = SOME (0 :num)
∀(n :num). ¬(n < n)
∀(P :α -> bool) (l :α list). LENGTH (FILTER P l) ≤ LENGTH l
∀(P :num -> bool) (n :num). EVERY P (COUNT_LIST n) ⇔ ∀(m :num). m < n ⇒ P m
∀(m :num) (n :num). count n m ⇔ m < n
(∀(P :α -> bool). PREFIX P ([] :α list) = ([] :α list)) ∧ ∀(P :β -> bool) (x :β) (l :β list). PREFIX P (x::l) = if P x then x::PREFIX P l else ([] :β list)
(∀(x :α). LAST [x] = x) ∧ ∀(x :α) (y :α) (z :α list). LAST (x::y::z) = LAST (y::z)
∀(n :num) (l :α list). n < LENGTH l ⇒ ELL n l = EL (PRE (LENGTH l − n)) l
∀(x :α). DISJOINT {x} (∅ :α -> bool)
∀(n :num) (l :α list). n ≤ LENGTH l ⇒ LENGTH (BUTLASTN n l) = LENGTH l − n
(∀(x :α). (P :α -> bool) x ⇒ (P' :α -> bool) x) ∧ (∀(y :β). (Q :β -> bool) y ⇒ (Q' :β -> bool) y) ⇒ PROD_ALL P Q (p :α # β) ⇒ PROD_ALL P' Q' p
∀(l1 :α list) (l2 :α list) (l3 :α list). IS_SUFFIX l1 l2 ∧ IS_SUFFIX l2 l3 ⇒ IS_SUFFIX l1 l3
DATATYPE ((pair :(α -> β -> α # β) -> γ) ($, :α -> β -> α # β))
INJ (LINV_OPT (f :β -> α) (s :β -> bool)) (IMAGE f s) (IMAGE (SOME :β -> β option) s)
∀(s :α -> bool) (t :α -> bool). countable s ∨ countable t ⇒ countable (s ∩ t)
∀(f :α -> β) (s :α -> bool) (u :α -> bool) (t :β -> bool). s ⊆ u ∧ IMAGE f u ⊆ t ⇒ IMAGE f s ⊆ t
∀(R :α -> α -> bool). transitive R^{*}
∀(X :α -> bool) (Y :α -> bool) (P :(α -> bool) -> bool). Y ∈ P ∧ DISJOINT Y X ⇒ DISJOINT X (BIGINTER P) ∧ DISJOINT (BIGINTER P) X
(0 :num) = LENGTH (l :α list) ⇔ l = ([] :α list)
(option_CASE (x :α option) (e :β) (f :α -> β) :β) = if IS_SOME x then f (THE x) else e
∀(s :α -> bool). DISJOINT s s ⇔ s = (∅ :α -> bool)
∀(R :α -> α -> bool) (a :α) (b :α). R^{*} a b ⇔ ∀(Q :α -> bool). Q a ∧ (∀(y :α) (z :α). Q y ∧ R y z ⇒ Q z) ⇒ Q b
∀(R :α -> α -> bool) (x :α) (y :α). RC R x y ⇒ R^{*} x y
∀(m :num) (n :num). m < n ⇒ m < SUC n
BIGINTER (∅ :(α -> bool) -> bool) = 𝕌(:α)
(¬(A :bool) ⇒ F) ⇒ (A ⇒ F) ⇒ F
∀(f :α -> β # γ) (a :β -> bool) (b :γ -> bool). PREIMAGE f (a × b) = PREIMAGE ((FST :β # γ -> β) ∘ f) a ∩ PREIMAGE ((SND :β # γ -> γ) ∘ f) b
∀(f :α -> β) (x :α option). OPTION_MAP f x = (NONE :β option) ⇔ x = (NONE :α option)
∀(f :α -> β) (s :α -> bool) (t :β -> bool). BIJ f s t ⇔ (∀(x :α). x ∈ s ⇒ f x ∈ t) ∧ ∃(g :β -> α). (∀(x :β). x ∈ t ⇒ g x ∈ s) ∧ (∀(x :α). x ∈ s ⇒ g (f x) = x) ∧ ∀(x :β). x ∈ t ⇒ f (g x) = x

∀(l1 :α list) (l2 :α list) (n :num) (x :α). n < LENGTH l1 ⇒ LUPDATE x n (l1 ++ l2) = LUPDATE x n l1 ++ l2
(some(x :α). x = (y :α)) = SOME y ∧ (some(x :α). y = x) = SOME y
(∀(x :α). FRONT [x] = ([] :α list)) ∧ ∀(x :α) (y :α) (z :α list). FRONT (x::y::z) = x::FRONT (y::z)
ZRECSPACE (ind_type$ZBOT :num -> α -> bool) ∧ ∀(c :num) (i :α) (r :num -> num -> α -> bool). (∀(n :num). ZRECSPACE (r n)) ⇒ ZRECSPACE (ind_type$ZCONSTR c i r)
∀(m :num) (n :num). MEM m (COUNT_LIST n) ⇔ m < n
∀(P :α -> β -> bool). (∀(x :α) (y :β). P x y) ⇔ ∀((x :α),(y :β)). P x y
∀(P :α -> bool) (x :α). x ∈ P ⇔ P x
BIGINTER (B :(α -> bool) -> bool) (x :α) ⇔ ∀(P :α -> bool). P ∈ B ⇒ x ∈ P
LIST_REL (P :α -> β -> bool) (GENLIST (f :num -> α) (l :num)) (GENLIST (g :num -> β) l) ⇔ ∀(i :num). i < l ⇒ P (f i) (g i)
∀(f :α -> β) (s :α -> bool) (t :β -> bool). SURJ f s t ⇔ IMAGE f s = t
∀(f :α -> β) (g :num -> α) (n :num). MAP f (GENLIST g n) = GENLIST (f ∘ g) n
∀(f :α -> β) (l :α list) (a :α). MEM a l ⇒ MEM (f a) (MAP f l)
BIGUNION (IMAGE (f :β -> α -> bool) (set (ls :β list))) ⊆ (s :α -> bool) ⇔ ∀(x :β). MEM x ls ⇒ f x ⊆ s
∀(s :α -> bool) (t :α -> bool). t ⊆ s ⇒ s DIFF (s DIFF t) = t
∀(set :α -> bool) (e :α -> bool). e ∈ POW set ⇔ e ⊆ set
FINITE 𝕌(:α # β) ⇔ FINITE 𝕌(:α) ∧ FINITE 𝕌(:β)
∀(f :α -> β) (s :α -> bool) (t :β -> bool). FINITE s ∧ BIJ f s t ⇒ CARD s = CARD t
{x | T} = 𝕌(:α)
COMPL (∅ :α -> bool) = 𝕌(:α)
∀(s :α -> bool) (t :α -> bool). s = t ⇔ ∀(x :α). x ∈ s ⇔ x ∈ t
∀(s :α -> bool). (∃(f :num -> α). BIJ f 𝕌(:num) s) ⇒ countable s
∀(a0 :α) (a1 :α list) (a0' :α) (a1' :α list). a0::a1 = a0'::a1' ⇔ a0 = a0' ∧ a1 = a1'
∀(s1 :(α -> bool) -> bool) (s2 :(α -> bool) -> bool). BIGINTER (s1 ∪ s2) = BIGINTER s1 ∩ BIGINTER s2
∀(f :α -> β -> bool) (s :γ -> bool) (t :α -> bool). s × BIGUNION (IMAGE f t) = BIGUNION (IMAGE (λ(n :α). s × f n) t)
PMATCH_EQUIV_ROWS (v :α) (rows1 :(α -> β option) list) (rows2 :(α -> β option) list) ⇒ PMATCH v rows1 = PMATCH v rows2
∀(l :α list) (x :α). SEG (1 :num) (LENGTH l) (SNOC x l) = [x]
∀(s :α -> bool). FINITE s ⇒ countable s
∀(l :α list). FINITE (set l)
∀(f :α -> num) (x :α) (ls :α list). MEM x ls ⇒ f x ≤ SUM (MAP f ls)
∀(x :α) (s :α -> bool). x ∈ x INSERT s
∀(n :num) (l :α list). n < LENGTH l ⇒ BUTLASTN n (FRONT l) = FRONT (BUTLASTN n l)
∀(v :α) (p :β -> α) (g :β -> bool) (p' :γ -> α) (g' :γ -> bool) (RES :bool). PMATCH_ROW_COND_EX v p g ⇒ (∀(x :β). g x ⇒ ((∃(x' :γ). p' x' = p x ∧ g' x') ⇔ RES)) ⇒ (PMATCH_ROW_COND_EX v p' g' ⇔ RES)
∀(x :α). (∅ :α -> bool) DELETE x = (∅ :α -> bool)
∀(s :α -> bool) (t :α -> bool). s DIFF t ∩ s = s DIFF t

∀(n :num) (l1 :α list) (l2 :α list). TAKE n (l1 ++ l2) = TAKE n l1 ++ TAKE (n − LENGTH l1) l2
∀(s :α -> bool) (t :α -> bool) (u :α -> bool). s ∩ (t ∩ u) = s ∩ t ∩ u
∀(P :α list -> bool). (∀(l :α list). LENGTH l = (0 :num) ⇒ P l) ⇔ P ([] :α list)
ASSOC ($++ :α list -> α list -> α list)
PMATCH_ROW_REDUNDANT (v :α) ([] :(α -> β option) list) (i :num) ⇔ F
∀(ZRECSPACE' :(num -> α -> bool) -> bool). ZRECSPACE' (ind_type$ZBOT :num -> α -> bool) ∧ (∀(c :num) (i :α) (r :num -> num -> α -> bool). (∀(n :num). ZRECSPACE' (r n)) ⇒ ZRECSPACE' (ind_type$ZCONSTR c i r)) ⇒ ∀(a0 :num -> α -> bool). ZRECSPACE a0 ⇒ ZRECSPACE' a0
∀(l1 :α list) (l2 :β list). LENGTH l1 = LENGTH l2 ⇒ ∀(x1 :α) (x2 :β). ZIP (SNOC x1 l1,SNOC x2 l2) = SNOC (x1,x2) (ZIP (l1,l2))
¬SHORTLEX (R :α -> α -> bool) (l :α list) ([] :α list)
∀(e :α). ∃(fn :unit -> α). fn () = e
∀(l :α list). l ≠ ([] :α list) ⇒ FRONT l ++ [LAST l] = l
∀(l :α list) (n :num). n < LENGTH l ⇒ LAST (DROP n l) = LAST l
∀(n :num) (l :α list). n ≤ LENGTH l ⇒ ∀(x :α). BUTLASTN n (x::l) = x::BUTLASTN n l
∀(f :α -> β) (s :β -> bool) (t :β -> bool). DISJOINT s t ⇒ DISJOINT (PREIMAGE f s) (PREIMAGE f t)
(∃(ls :α list). (P :α list -> bool) ls) ⇔ ∃(n :num) (f :num -> α). P (GENLIST f n)
nub ([] :α list) = ([] :α list)
WF (R :α -> α -> bool)$^{+}$ ⇔ WF R
∀(n :num). countable (count n)
{x | (y :α) = x} = {y}
∀(x :α) (l :α list). SNOC x l = l ++ [x]
∀(s :α -> bool) (t :α -> bool). DISJOINT s t ⇔ ∀(x :α). x ∈ s ⇒ x ∉ t
GENLIST (f :num -> α) (0 :num) = ([] :α list) ∧ GENLIST f (NUMERAL (n :num)) = GENLIST_AUX f (NUMERAL n) ([] :α list)
∀(s :α) (x :α -> bool). x ⊂ {s} ⇔ x = (∅ :α -> bool)
∀(R :α -> α -> bool). symmetric R^{T} ⇔ symmetric R
∀(t1 :α) (t2 :α). (if F then t1 else t2) = t2
∀(l1 :α list) (l2 :α list). set (l1 ++ l2) = set l1 ∪ set l2
∀(l1 :α list) (l2 :α list). l2 ≼ l1 ⇔ ∃(l :α list). l1 = l2 ++ l
∀(R :α -> α -> bool). irreflexive R^{T} ⇔ irreflexive R
∀(s :α -> bool). CHOICE s ∉ REST s
∀(l1 :α list) (l2 :α list). l1 ++ l2 = FOLDL (λ(l' :α list) (x :α). SNOC x l') l1 l2
∀(L :α list) (n :num). LEN L n = LENGTH L + n
∀(x :bool) (x' :bool) (y :bool) (y' :bool). (x ⇒ y' ⇒ y) ∧ (¬y' ⇒ x ⇒ x') ⇒ (x' ⇒ y') ⇒ x ⇒ y
∀(P :α -> bool) (l :α list). EVERY P l ⇔ ∀(e :α). MEM e l ⇒ P e
∀(n1 :num) (n2 :num). count n1 = count n2 ⇔ n1 = n2
∀(n :num). (0 :num) < n ⇒ ∀(x :α) (l :α list). EL n (x::l) = EL (PRE n) l
∀(e :α) (l :α list). LIST_ELEM_COUNT e l > (0 :num) ⇔ MEM e l
∀(t :bool). t ∨ t ⇔ t

(∀(x :α) (y :α). (f :α -> β) x = f y ⇔ x = y) ⇒ (DISJOINT (IMAGE f (s1 :α -> bool)) (IMAGE f (s2 :α -> bool)) ⇔ DISJOINT s1 s2)
∀(x :bool) (x' :bool) (y :bool) (y' :bool). (y ⇒ x' ⇒ x) ∧ (x' ⇒ y' ⇒ y) ⇒ x' ∧ y' ⇒ x ∧ y
∀(n :num). DROP n ([] :α list) = ([] :α list)
∀(s :α -> bool). s ⊂ 𝕌(:α) ⇔ ∃(x :α). x ∉ s
(Q :α option -> bool) ($some (P :α -> bool)) ⇒ (∃(x :α). P x ∧ Q (SOME x)) ∨ (∀(x :α). ¬P x) ∧ Q (NONE :α option)
transitive ($SUBSET :(α -> bool) -> (α -> bool) -> bool)
∀(f :α -> β) (g :β -> γ) (s :α -> bool) (t :β -> bool) (u :γ -> bool). INJ f s t ∧ INJ g t u ⇒ INJ (g ∘ f) s u
∀(y :β) (s :α -> bool) (f :α -> β). IMAGE f s y ⇔ ∃(x :α). y = f x ∧ x ∈ s
∀(f :α -> β) (g :β -> γ) (s :α -> bool) (t :β -> bool) (u :γ -> bool). SURJ f s t ∧ SURJ g t u ⇒ SURJ (g ∘ f) s u
∀(f :'z -> 'z). INVOL f ⇒ ∀(a :'z) (b :'z). f a = f b ⇔ a = b
∀(l1 :α list) (l2 :α list) (l3 :α list). l1 ++ (l2 ++ l3) = l1 ++ l2 ++ l3
$OLEAST (P :num -> bool) = (NONE :num option) ⇔ ∀(n :num). ¬P n
∀(n :num) (l1 :α list) (l2 :α list). DROP n (l1 ++ l2) = DROP n l1 ++ DROP (n − LENGTH l1) l2
∀(ls :α list) (f :β -> α). EVERY (λ(x :α). ∃(y :β). x = f y) ls ⇒ ∃(l :β list). ls = MAP f l
∀(opt :α option). (∃(x :α). opt = SOME x) ∨ opt = (NONE :α option)
SURJ (f :α -> β) (s :α -> bool) (t :β -> bool) ⇒ ∃(g :β -> α). INJ g t s ∧ ∀(y :β). y ∈ t ⇒ f (g y) = y
∀(R :α -> α -> bool) (x :α) (y :α) (z :α). R^* x y ∧ R y z ⇒ R^+ x z
∀(f :α -> β) (s :α -> bool) (t :β -> bool). INJ f s t ∧ INFINITE s ⇒ INFINITE t
∀(x1 :α) (l1 :α list) (x2 :α) (l2 :α list). SNOC x1 l1 = SNOC x2 l2 ⇒ LENGTH l1 = LENGTH l2
∀(x :α) (l :α list). TL (SNOC x l) = if NULL l then ([] :α list) else SNOC x (TL l)
∀(P :α list -> bool) (n :num). (∀(l :α list). LENGTH l = SUC n ⇒ P l) ⇔ ∀(l :α list). LENGTH l = n ⇒ (λ(l :α list). ∀(x :α). P (x::l)) l
(x :α) ∈ RDOM (RRESTRICT (R :α -> β -> bool) (s :α -> bool)) ⇔ x ∈ RDOM R ∧ x ∈ s
∀(x :α) (l :α list). FRONT (SNOC x l) = l
∀(l :α list) (x :α). MEM x (REVERSE l) ⇔ MEM x l
∀(l :α list). NULL l ⇔ FOLDL (λ(x :bool) (l' :α). F) T l
∀(l :α list) (m :num) (x :α). MEM x (DROP m l) ⇒ MEM x l
∀(x :α + β). ISR x ⇒ (INR (OUTR x) :α + β) = x
(([(f :β -> α)] <*> (l :β list)) :α list) = MAP f l
∀(n :num) (x :α). SNOC x (REPLICATE n x) = REPLICATE (SUC n) x
∀(s :α -> bool) (x :α) (y :α). x INSERT s = {y} ⇔ x = y ∧ s ⊆ {y}
∀(n :num) (l1 :α list) (l2 :α list). l1 ≼ l2 ∧ n < LENGTH l1 ∧ n < LENGTH l2 ⇒ EL n l1 = EL n l2
∀(R :α -> α -> bool). reflexive (RC R)
(R1 :γ -> β -> bool) $\circ_r$ (R2 :δ -> γ -> bool) $\circ_r$ (R3 :α -> δ -> bool) = (R1 $\circ_r$ R2) $\circ_r$ R3
∀(n :num). (0 :num) < n ⇒ ∀(x :α) (l :α list). ELL n (SNOC x l) = ELL (PRE n) l
∀(l1 :α list) (l2 :α list). ALL_DISTINCT (l1 ++ l2) ⇔ ALL_DISTINCT l1 ∧ ALL_DISTINCT l2 ∧ ∀(e :α). MEM e l1 ⇒ ¬MEM e l2
∀(l :α list). l ≠ ([] :α list) ⇒ BUTLASTN (1 :num) l = FRONT l

∀(s :α -> bool) (t :α -> bool). BIGUNION {s; t} = s ∪ t
∀(l2 :α list) (l1 :α list). ¬NULL l2 ⇒ EL (LENGTH l1) (l1 ++ l2) = HD l2
reflexive ((R1 :α -> α -> bool) LEX (R2 :β -> β -> bool)) ⇔ reflexive R1 ∨ reflexive R2
INJ (f :α -> β) (s :α -> bool) (IMAGE f s) ⇒ (countable (IMAGE f s) ⇔ countable s)
MAP (f :β -> α) (LIST_BIND (l :γ list) (g :γ -> β list)) = LIST_BIND l (MAP f ∘ g)
FUNSET 𝕌(:α) 𝕌(:β) = 𝕌(:α -> β)
∀(A :bool) (B :bool) (C :bool). A ∧ B ⇒ C ⇔ A ⇒ B ⇒ C
MAP (SND :β # α -> α) (FILTER (λ((x :β),(y :α)). y ≠ (z :α)) (ls :(β # α) list)) = FILTER (λ(y :α). z ≠ y) (MAP (SND :β # α -> α) ls)
REPLICATE (n :num) (a :α) ++ REPLICATE (m :num) a = REPLICATE (n + m) a
∀(b :bool). ¬b ⇒ (b ⇔ F)
∀(x :α) (y :β) (a :α) (b :β). (x,y) = (a,b) ⇔ x = a ∧ y = b
∀(f :α -> β) (n :num) (x :α). MAP f (REPLICATE n x) = REPLICATE n (f x)
∀(n :num) (l :α list). n < LENGTH l ⇒ SNOC (EL n l) (TAKE n l) = TAKE (SUC n) l
PMATCH_ROW_COND_EX (i :α) (p :β -> α) (g :β -> bool) ⇒ PMATCH_ROW p g (r :β -> γ) i = SOME (r (@(x :β). PMATCH_ROW_COND p g i x))
∀(R1 :α -> α -> bool) (R2 :α -> α -> bool) (x :α) (y :α). (R1 $\cap_r$ R2)$^{✡}$ x y ⇒ (R1$^{✡}$ $\cap_r$ R2$^{✡}$) x y
∀(P :α -> bool) (a :α) (s :α -> bool). (∃(x :α). x ∈ a INSERT s ∧ P x) ⇔ P a ∨ ∃(x :α). x ∈ s ∧ P x
∀(f :α -> γ) (g :β -> δ) (z :α + β). SUM_MAP f g z = if ISL z then (INL (f (OUTL z)) :γ + δ) else (INR (g (OUTR z)) :γ + δ)
∀(l :α list list). REVERSE (FLAT l) = FLAT (REVERSE (MAP (REVERSE :α list -> α list) l))
∀(f1 :α -> β) (f2 :α -> β) (l :α list). MAP f1 l = MAP f2 l ⇔ ∀(e :α). MEM e l ⇒ f1 e = f2 e
($= :α -> α -> bool)^{T} = ($= :α -> α -> bool)
∀(n :num) (l :α list). n < LENGTH l ⇒ ELL n (REVERSE l) = ELL (PRE (LENGTH l − n)) l
∀(x :α) (s :α -> bool). (∅ :α -> bool) ≠ x INSERT s
(¬SHORTLEX (R :α -> α -> bool) ([] :α list) ([] :α list) ∧ ¬SHORTLEX R ((h1 :α)::(t1 :α list)) ([] :α list)) ∧ SHORTLEX R ([] :α list) ((h2 :α)::(t2 :α list)) ∧ (SHORTLEX R (h1::t1) (h2::t2) ⇔ LENGTH t1 < LENGTH t2 ∨ LENGTH t1 = LENGTH t2 ∧ (R h1 h2 ∨ h1 = h2 ∧ SHORTLEX R t1 t2))
(∀(v :α). PMATCH_IS_EXHAUSTIVE v ([] :(α -> β option) list) ⇔ F) ∧ ∀(v :γ) (r :γ -> δ option) (rs :(γ -> δ option) list). PMATCH_IS_EXHAUSTIVE v (r::rs) ⇔ r v ≠ (NONE :δ option) ∨ PMATCH_IS_EXHAUSTIVE v rs
(∀(s :α -> bool). 𝕌(:α) ∩ s = s) ∧ ∀(s :α -> bool). s ∩ 𝕌(:α) = s
∀(f :α -> β) (s :α -> bool) (t :β -> bool). BIJ f s t ⇔ f ∈ FUNSET s t ∧ ∀(y :β). y ∈ t ⇒ ∃!(x :α). x ∈ s ∧ y = f x
∀(n :num) (l :α list). n ≤ LENGTH l ⇒ BUTLASTN n l = TAKE (LENGTH l − n) l
∀(l2 :α list) (l1 :α list). IS_SUFFIX (REVERSE l1) (REVERSE l2) ⇔ l2 ≼ l1
∀(s :α -> bool) (x :α) (y :α). x ∈ s DELETE y ⇔ x ∈ s ∧ x ≠ y
∀(ls :α list) (n :num). n ≤ LENGTH ls ⇒ REVERSE (DROP n ls) = REVERSE (LASTN (LENGTH ls − n) ls)
∀(f :α -> β). PREIMAGE f (∅ :β -> bool) = (∅ :α -> bool)
∀(f :α -> β) (l :α list). MAP f (REVERSE l) = REVERSE (MAP f l)
∀(s :α -> bool) (t :α -> bool) (x :α). (s ∪ t) x ⇔ x ∈ s ∨ x ∈ t

∀(n :num). LENGTH (COUNT_LIST n) = n
(∃(p :α # β). (P :α # β -> bool) p) ⇔ ∃(p_1 :α) (p_2 :β). P (p_1,p_2)
∀(l1 :α list) (l2 :α list). l1 ++ l2 = FOLDR (CONS :α -> α list -> α list) l2 l1
∀(s :α -> bool) (x :α). COMPL (x INSERT s) = COMPL s DELETE x
∀(t :bool). t ⇒ T ⇔ T
countable (∅ :α -> bool)
∀(R :α -> α -> bool). Order R ⇔ StrongOrder (STRORD R)
∀(x :bool) (x' :bool) (y :bool) (y' :bool). (x ⇒ y ⇒ y') ∧ (¬y' ⇒ x' ⇒ x) ⇒ (x ⇒ y) ⇒ x' ⇒ y'
(x :α list) ≼ (y :α)::(ys :α list) ⇔ x = ([] :α list) ∨ ∃(xs :α list). x = y::xs ∧ xs ≼ ys
∀(P :α -> bool) (l :α list) (m :num). EVERY P l ⇒ EVERY P (TAKE m l)
IS_SOME (OPTION_BIND (x :α option) (g :α -> β option)) ⇒ IS_SOME x
(∀(x :α) (y :α). (R :α -> α -> bool) x y ⇒ (Q :α -> α -> bool) x y) ⇒ R$^{✡}$ (x :α) (y :α) ⇒ Q$^{✡}$ x y
∀(x :α list). x ≼ x
count (0 :num) = (∅ :num -> bool)
(∀(x :α). (P :α -> bool) x ⇒ (Q :α -> bool) x) ⇒ EXISTS P (l :α list) ⇒ EXISTS Q l
∀(s :α -> bool) (t :α -> bool). s ⊆ t ⇒ s ∩ t = s
∀(R :α -> α -> bool) (x :α) (y :α). R x y ⇒ R^{+} x y
∀(xs :α list). FILTER (λ(x :α). T) xs = xs
∀(s :α -> bool). (λ(x :α). x) PERMUTES s
∀(l1 :α list) (n :num). LENGTH l1 ≤ n ⇒ ∀(l2 :α list). TAKE n (l1 ++ l2) = l1 ++ TAKE (n − LENGTH l1) l2
∀(s :α -> bool). INFINITE s ⇒ ∀(t :α -> bool). s ⊆ t ⇒ INFINITE t
(option_CASE (NONE :α option) :β -> (α -> β) -> β) = (λ(v :β) (f :α -> β). v) ∧ (option_CASE (SOME (x :α)) :β -> (α -> β) -> β) = (λ(v :β) (f :α -> β). f x)
∀(t :bool). t ⇒ F ⇔ ¬t
∀(f1 :α -> bool) (f2 :α -> bool) (l :α list). FILTER f1 (FILTER f2 l) = FILTER f2 (FILTER f1 l)
∀(f :α -> β) (P :α -> bool) (Q :β -> bool). FUNSET P Q f ⇔ ∀(x :α). x ∈ P ⇒ f x ∈ Q
𝕌(:α + β) = IMAGE (INL :α -> α + β) 𝕌(:α) ∪ IMAGE (INR :β -> α + β) 𝕌(:β)
∀(x :α) (s :α -> bool). s ⊆ x INSERT s
∀(l :α list) (x :α). l ≠ x::l ∧ x::l ≠ l
PROD_SET (∅ :num -> bool) = (1 :num) ∧ ∀(x :num) (s :num -> bool). FINITE s ⇒ PROD_SET (x INSERT s) = x * PROD_SET (s DELETE x)
∀(f :α -> β) (s :β -> bool) (x :α). PREIMAGE f s x ⇔ f x ∈ s
∀(f :α -> β -> β) (e :β) (l1 :α list) (l2 :α list). FOLDR f e (l1 ++ l2) = FOLDR f (FOLDR f e l2) l1
∀(n :num) (l :α list). n ≤ LENGTH l ⇒ DROP n l = LASTN (LENGTH l − n) l
∀(n :num) (l :α list). n ≤ LENGTH l ⇒ LENGTH (TAKE n l) = n
(0 :num) < (n :num) ⇒ DROP n ((x :α)::(xs :α list)) = DROP (n − (1 :num)) xs
∀(l :bool list). AND_EL l ⇔ FOLDL $/\ T l
∀(P :α -> bool) (l :α list). FILTER P l = ([] :α list) ⇔ EVERY (λ(x :α). ¬P x) l
∀(v :α) (rows :(α -> β option) list) (p :bool) (infos :bool list). FST (STRONGEST_REDUNDANT_ROWS_INFO_AUX v rows p infos) ⇔ p ∧ EVERY (λ(r :α -> β option). r v = (NONE :β option)) rows
∀(s :α -> bool). s ∩ s = s

∀(x :α -> bool) (y :α -> bool) (z :α -> bool). x DIFF y DIFF z = x DIFF z DIFF y
∀(c :bool) (x :bool) (x' :bool) (y :bool) (y' :bool). (c ⇒ x' ⇒ x) ∧ (¬c ⇒ y' ⇒ y) ⇒ (if c then x' else y') ⇒ if c then x else y
∀(x :α). SING {x}
((a :α),(b :α)) ∈ {(x,x) | (P :α -> bool) x} ⇔ P a ∧ a = b
∀(M :α list) (M' :α list) (v :β) (f :α -> α list -> β). M = M' ∧ (M' = ([] :α list) ⇒ v = (v' :β)) ∧ (∀(a0 :α) (a1 :α list). M' = a0::a1 ⇒ f a0 a1 = (f' :α -> α list -> β) a0 a1) ⇒ (list_CASE M v f :β) = (list_CASE M' v' f' :β)
∀(x :α) (l :α list). SNOC x l ≠ ([] :α list)
∀(g :α -> α -> α) (f :β -> α -> α). FCOMM g f ⇒ ∀(e :α). LEFT_ID g e ⇒ ∀(l :β list list). FOLDR f e (FLAT l) = FOLDR g e (MAP (FOLDR f e) l)
∀(m :num) (n :num). m < SUC n ⇒ m ≠ n ⇒ m < n
∀(s :α -> bool) (t :α -> bool) (u :α -> bool). s ⊂ t ∧ t ⊂ u ⇒ s ⊂ u
∀(s :α -> bool) (t :α -> bool). FINITE s ∧ FINITE t ⇒ CARD (s ∪ t) = CARD s + CARD t − CARD (s ∩ t)
∀(s :α -> bool). 𝕌(:α) ⊆ s ⇔ s = 𝕌(:α)
∀(v :α) (rows :(α -> β option) list) (p :bool) (infos :bool list). LENGTH (SND (STRONGEST_REDUNDANT_ROWS_INFO_AUX v rows p infos)) = LENGTH rows + LENGTH infos
LIST_REL (R :α -> β -> bool) (xs :α list) ((h :β)::(t :β list)) ⇔ ∃(h' :α) (t' :α list). xs = h'::t' ∧ R h' h ∧ LIST_REL R t' t
(∀(x :α) (y :α). (P :α -> bool) x ∧ (R :α -> α -> bool) x y ⇒ P y) ⇒ ∀(x :α) (y :α). P x ∧ R* x y ⇒ P y
((p :bool) ⇔ if (q :bool) then (r :bool) else (s :bool)) ⇔ (p ∨ q ∨ ¬s) ∧ (p ∨ ¬r ∨ ¬q) ∧ (p ∨ ¬r ∨ ¬s) ∧ (¬q ∨ r ∨ ¬p) ∧ (q ∨ s ∨ ¬p)
∀(s :α -> bool) (x :α). x ∈ s ⇔ ∃(t :α -> bool). s = x INSERT t ∧ x ∉ t
PMATCH_ROW_REDUNDANT (v :α) ((r :α -> β option)::(rs :(α -> β option) list)) (0 :num) ⇔ r v = (NONE :β option)
∀(s :α -> bool) (t :α -> bool). DISJOINT t (s DIFF t) ∧ DISJOINT (s DIFF t) t
∀(f :α -> β -> bool) (s :α -> bool) (y :β). y ∈ BIGUNION (IMAGE f s) ⇔ ∃(x :α). x ∈ s ∧ y ∈ f x
(∀(e :α). LIST_ELEM_COUNT e ([] :α list) = (0 :num)) ∧ (∀(e :β) (l1 :β list) (l2 :β list). LIST_ELEM_COUNT e (l1 ++ l2) = LIST_ELEM_COUNT e l1 + LIST_ELEM_COUNT e l2) ∧ (∀(e :γ) (h :γ) (l :γ list). h = e ⇒ LIST_ELEM_COUNT e (h::l) = SUC (LIST_ELEM_COUNT e l)) ∧ ∀(e :δ) (h :δ) (l :δ list). h ≠ e ⇒ LIST_ELEM_COUNT e (h::l) = LIST_ELEM_COUNT e l
∀(s :α -> bool) (t :α -> bool) (x :α). (s ∩ t) x ⇔ x ∈ s ∧ x ∈ t
∀(P :α -> bool) (l1 :α list) (l2 :α list). EXISTS P (l1 ++ l2) ⇔ EXISTS P l1 ∨ EXISTS P l2
∀(ls :α list) (f :α # α -> β). MAP f (ZIP (ls,ls)) = MAP (λ(x :α). f (x,x)) ls
MAP (f :α -> β) (l :α list) = (h :β)::(t :β list) ⇔ ∃(x0 :α) (t0 :α list). l = x0::t0 ∧ h = f x0 ∧ t = MAP f t0
∀(R :α -> α -> bool). symmetric R ⇒ SC R = R
∀(l1 :α list) (l2 :α list). SHORTLEX (R :α -> α -> bool) l1 l2 ⇒ LENGTH l1 ≤ LENGTH l2
∀(f :β -> α -> β) (e :β) (x :α) (l :α list). FOLDL f e (SNOC x l) = f (FOLDL f e l) x
∀(P :(α -> bool) -> bool). P (∅ :α -> bool) ∧ (∀(s :α -> bool). FINITE s ∧ P s ⇒ ∀(e :α). e ∉ s ⇒ P (e INSERT s)) ⇒ ∀(s :α -> bool). FINITE s ⇒ P s
∀(s :α -> bool). FINITE s ⇒ FINITE (REST s)

∀(s :α -> bool). FINITE s ⇒ ∃(f :num -> α) (b :num). BIJ f (count b) s
∀(P :α -> bool) (l1 :α list) (l2 :α list). EVERY P l1 ⇒ dropWhile P (l1 ++ l2) = dropWhile P l2
(OPTION_IGNORE_BIND (m1 :α option) (m2 :β option) = (NONE :β option) ⇔ m1 = (NONE :α option) ∨ m2 = (NONE :β option)) ∧ (OPTION_IGNORE_BIND m1 m2 = SOME (y :β) ⇔ ∃(x :α). m1 = SOME x ∧ m2 = SOME y)
∀(P :α -> bool) (l :α list). EVERY P l ⇔ ¬EXISTS (λ(x :α). ¬P x) l
∀(s :α -> bool). s = 𝕌(:α) ⇒ ∀(v :α). v ∈ s
∀(n :num). SUC (m :num) = n ⇒ m < n
((if (P :bool) then SOME (x :α) else (NONE :α option)) = (NONE :α option) ⇔ ¬P) ∧ ((if P then (NONE :α option) else SOME x) = (NONE :α option) ⇔ P) ∧ ((if P then SOME x else (NONE :α option)) = SOME (y :α) ⇔ P ∧ x = y) ∧ ((if P then (NONE :α option) else SOME x) = SOME y ⇔ ¬P ∧ x = y)
(some(x :bool). x) = SOME T
REDUNDANT_ROWS_INFOS_CONJ ([] :bool list) ([] :bool list) = ([] :bool list) ∧ REDUNDANT_ROWS_INFOS_CONJ ((i1 :bool)::(is1 :bool list)) ((i2 :bool)::(is2 :bool list)) = (i1 ∧ i2)::REDUNDANT_ROWS_INFOS_CONJ is1 is2
∀(l :num list list). SUM (FLAT l) = SUM (MAP SUM l)
∀(x :α recspace) (y :α recspace). dest_rec x = dest_rec y ⇔ x = y
((R1 :α -> α -> bool) LEX (R2 :β -> β -> bool)) ((a :α),(b :β)) ((c :α),(d :β)) ⇔ R1 a c ∨ a = c ∧ R2 b d
(∀(l :α list). ([] :α list) ≼ l ⇔ T) ∧ (∀(x :α) (l :α list). x::l ≼ ([] :α list) ⇔ F) ∧ ∀(x1 :α) (l1 :α list) (x2 :α) (l2 :α list). x2::l2 ≼ x1::l1 ⇔ x1 = x2 ∧ l2 ≼ l1
∀(n :num) (l :α list). n ≤ LENGTH l ⇒ DROP n (REVERSE l) = REVERSE (BUTLASTN n l)
∀(l :α list) (f :α -> β) (x :β). MEM x (MAP f l) ⇔ ∃(y :α). x = f y ∧ MEM y l
∀(f :α -> β) (s :β -> bool) (t :β -> bool). s ⊆ t ⇒ PREIMAGE f s ⊆ PREIMAGE f t
PROD_ALL (P :α -> bool) (Q :β -> bool) ((x :α),(y :β)) ⇔ P x ∧ Q y
∀(R :α -> α -> bool) (x :α) (y :α). R^+ x y ⇒ R^* x y
∀(l2 :α list) (l1 :α list). BUTLASTN (LENGTH l2) (l1 ++ l2) = l1
∀(l :α list). LENGTH l = FOLDR (λ(x :α) (l' :num). SUC l') (0 :num) l
∀(n :num) (l :α list). n ≤ LENGTH l ⇒ LASTN n l = DROP (LENGTH l − n) l
∀(t :bool). F ∨ t ⇔ t
∀(f :α -> β) (s :β -> bool). IMAGE f (PREIMAGE f s) ⊆ s
∀(s :α -> bool) (t :α -> bool). s ⊂ t ⇔ s ⊆ t ∧ ∃(y :α). y ∈ t ∧ y ∉ s
$(R$:α -> α -> bool$)^+$ (x :α) (z :α) ⇔ R x z ∨ ∃(y :α). R x y ∧ R^+ y z
∀(m :num) (n :num) (l :α list). m ≤ n ∧ n ≤ LENGTH l ⇒ BUTLASTN m (LASTN n l) = LASTN (n − m) (BUTLASTN m l)
∀(f :α -> β) (P :α -> bool) (Q :α -> β -> bool). f ∈ DFUNSET P Q ⇔ ∀(x :α). x ∈ P ⇒ f x ∈ Q x
(∀(x :α). MEM x ([] :α list) ⇔ F) ∧ ∀(x :α) (h :α) (t :α list). MEM x (h::t) ⇔ x = h ∨ MEM x t
∀(x :α) (s :α -> bool). x ∉ s ⇔ s DELETE x = s
((INL (a :α) :α + β) ∈ (A :α -> bool) ⊔ (B :β -> bool) ⇔ a ∈ A) ∧ ((INR (b :β) :α + β) ∈ A ⊔ B ⇔ b ∈ B)
WF (λ(x :num) (y :num). y = SUC x)
∀(L :num list) (n :num). SUM_ACC L n = SUM L + n
INFINITE 𝕌(:α) ⇔ ∀(s :α -> bool). FINITE s ⇒ s ⊂ 𝕌(:α)

∀(ls :α list list). FLAT ls = ([] :α list) ⇔ EVERY ($= ([] :α list)) ls
∀(f :α -> β). (∀(s :β -> bool). INJ f (∅ :α -> bool) s) ∧ ∀(s :α -> bool). INJ f s (∅ :β -> bool) ⇔ s = (∅ :α -> bool)
∀(l :α list) (f :α -> β). (MAP f l = ([] :β list) ⇔ l = ([] :α list)) ∧ (([] :β list) = MAP f l ⇔ l = ([] :α list))
∀(l :num list). SUM l = FOLDR $+ (0 :num) l
∀(R :α -> β -> bool) (l1 :α list) (l2 :β list) (n :num). LIST_REL R l1 l2 ⇒ LIST_REL R (DROP n l1) (DROP n l2)
(some(x :α). F) = (NONE :α option)
∀(f :α -> β) (l1 :α list) (l2 :α list). MAP f (l1 ++ l2) = MAP f l1 ++ MAP f l2
∀(g :α -> α -> α) (f :β -> α -> α). FCOMM g f ⇒ ∀(e :α). LEFT_ID g e ⇒ ∀(l1 :β list) (l2 :β list). FOLDR f e (l1 ++ l2) = g (FOLDR f e l1) (FOLDR f e l2)
∅ (x :α) ⇔ F
∀(R :α -> α -> bool). WeakLinearOrder R ⇔ WeakOrder R ∧ ∀(a :α) (b :α). R a b ∨ R b a
∀(R :α -> β -> bool) (R' :γ -> α -> bool). $(R \circ_r R')^T = R'^T \circ_r R^T$
∀(x :α) (l :α list). x::l = [x] ++ l
∀(f :α -> β) (l :α list). MAP f l = FOLDR (λ(x :α) (l' :β list). f x::l') ([] :β list) l
∀(x :α). x ∈ 𝕌(:α)
𝕌(:α + β) = 𝕌(:α) ⊔ 𝕌(:β)
∀(s :α -> bool) (t :α -> bool) (g :α -> bool). (s DIFF t) ∩ g = s ∩ g DIFF t
∀(Q :num -> bool) (P :num -> bool). (∃(n :num). P n) ∧ (∀(n :num). (∀(m :num). m < n ⇒ ¬P m) ∧ P n ⇒ Q n) ⇒ Q ($LEAST P)
∀(xs :α list). FILTER (λ(x :α). F) xs = ([] :α list)
∀(f :α -> β) (x :α) (s :α -> bool). IMAGE f (x INSERT s) = f x INSERT IMAGE f s
∀(l1 :α list) (l2 :α list). TAKE (LENGTH l1) (l1 ++ l2) = l1
∀(l :α list list). FLAT l = FOLDR ($++ :α list -> α list -> α list) ([] :α list) l
(¬LLEX (R :α -> α -> bool) ([] :α list) ([] :α list) ∧ ¬LLEX R ((h1 :α)::(t1 :α list)) ([] :α list)) ∧ LLEX R ([] :α list) ((h2 :α)::(t2 :α list)) ∧ (LLEX R (h1::t1) (h2::t2) ⇔ R h1 h2 ∨ h1 = h2 ∧ LLEX R t1 t2)
∀(l :α list). l ++ ([] :α list) = l
∀(s :α -> bool) (f :α -> β). f ∈ FUNSET s (∅ :β -> bool) ⇔ s = (∅ :α -> bool)
∀(b :bool). b ⇒ (b ⇔ T)
∀(l :α list) (x :α). ELL (0 :num) (SNOC x l) = x
∀(ls :α list). ls ≠ ([] :α list) ⇒ MAP (f :α -> β) (FRONT ls) = FRONT (MAP f ls)
∀(n :num). (0 :num) < SUC n
IMAGE (f :β -> α) (s :β -> bool) = {(z :α)} ⇔ s ≠ (∅ :β -> bool) ∧ ∀(x :β). x ∈ s ⇒ f x = z
∀(f :α -> β -> β) (e :β) (l :α list). FOLDR f e l = FOLDL (λ(x :β) (y :α). f y x) e (REVERSE l)
∀(l1 :α list) (l2 :α list) (a :α). IS_SUFFIX l1 l2 ⇒ IS_SUFFIX (a::l1) l2
∀(x :α) (P :α -> bool). x ∈ P ⇔ P x
∀(R :α -> α -> bool). StrongOrder R ⇒ Order R
∀(n :num) (l :α list). n < LENGTH l ⇒ MEM (EL n l) l
∀(x :α) (y :α) (P :α -> bool). x ∈ y INSERT P ⇔ x = y ∨ x ≠ y ∧ x ∈ P
OWHILE (G :α -> bool) (f :α -> α) (s :α) = SOME (s' :α) ⇒ ¬G s'

symmetric (R1 :α -> α -> bool) ∧ symmetric (R2 :β -> β -> bool) ⇒ symmetric (R1 LEX R2)
∀(P :α -> bool) (l :α list). PREFIX P l ≼ l
∀(s :α -> bool) (f :α -> β). (∃(t :β -> bool). INJ f s t) ⇒ BIJ f s (IMAGE f s)
∀(s :α -> bool). s DIFF 𝕌(:α) = (∅ :α -> bool)
∀(t :bool). t ∧ t ⇔ t
(∃(x :α). x ∈ (s :α -> bool)) ∧ (∀(x :α). x ∈ s ⇒ (P :α -> bool) x) ⇒ P (CHOICE s)