cheat sheet ssrnat_doc.v (SSReflect v1.5)

ssrfun.v naming conventions

K	cancel
LR	move an op from the lhs of a rel to the rhs
RL	move an op from the rhs to the lhs

ssrfun.v notations

`f ^~ y`	`fun x => f x y`
`p .1`	`fst p`
`p .2`	`snd p`
`f =1 g`	`f x = g x`
`{morph f : x / aF x >-> rR x}`	`f (aF x) = rF (f x)`
`{morph f : x y / aOp x y >-> rOp x y}`	`f (aOp x y) = rOp (f x) (f y)`

ssrfun.v definitions

injective f	forall x1 x2, f x1 = f x2 -> x1 = x2
cancel f g	g (f x) = x
involutive f	cancel f f
left_injective op	injective (op^~ x)
right_injective op	injective (op y)
left_id e op	op e x = x
right_id e op	op x e = x
left_zero z op	op z x = z
right_commutative op	op (op x y) z = op (op x z) y
right_zero z op	op x z = z
left_commutative op	op x (op y z) = op y (op x z)
left_distributive op add	op (add x y) z = add (op x z) (op y z)
right_distributive op add	op x (add y z) = add (op x y) (op x z)
left_loop inv op	cancel (op x) (op (inv x))
self_inverse e op	op x x = e
commutative op	op x y = op y x
idempotent op	op x x = x
associative op	op x (op y z) = op (op x y) z

ssrbool.v naming conventions

A	associativity
AC	right commutativity
b	a boolean argument
C	commutativity/complement
D	predicate difference
E	elimination
F/f	boolean false
T/t	boolean truth
U	predicate union

ssrnat.v naming conventions

A(infix)	conjunction
B	subtraction
D	addition
p(prefix)	positive
S	successor
V(infix)	disjunction

```
nat_scope
Notation "n .+1":= (succn n).    Notation "n .*2":= (double n).   Notation "n `!":=(factorial n).
Notation "n .-1":= (predn n).    Notation "n ./2":= (half n).
Notation "m < n" := (m.+1 <= n).  Notation "m ^ n":=(expn m n).
```

```
add0n/addn0   left_id 0 addn/right_id 0 addn
add1n/addn1   1 + n = n.+1/n + 1 = n.+1
addn2         n + 2 = n.+2
addSn         m.+1 + n = (m + n).+1
addnS         m + n.+1 = (m + n).+1
addSnnS       m.+1 + n = m + n.+1
addnC         commutative addn
addnA         associative addn
addnCA        left_commutative addn
eqn_add2l     (p + m == p + n) = (m == n)
eqn_add2r     (m + p == n + p) = (m == n)
sub0n/subn0   left_zero 0 subn/right_id 0 subn
subnn         self_inverse 0 subn
subSS         m.+1 - n.+1 = m - n
subn1         n - 1 = n.-1
subnDl        (p + m) - (p + n) = m - n
subnDr        (m + p) - (n + p) = m - n
addKn         cancel (addn n) (subn^~ n)
addnK         cancel (addn^~ n) (subn^~ n)
subSnn        n.+1 -n = 1
subnDA        n - (m + p) = (n - m) - p
subnAC        right_commutative subn
ltnS          (m < n.+1) = (m <= n)
prednK        0 < n -> n.-1.+1 = n
leqNgt        (m <= n) = ~~ (n < m)
ltnNge        (m < n) = ~~ (n <= m)
ltnn          n < n = false
subSnn        n.+1 -n = 1
subnDA        n - (m + p) = (n - m) - p
leq_eqVlt     (m <= n) = (m == n) || (m < n)
ltn_neqAle    (m < n) = (m != n) && (m <=n)
ltn_add2l     (p + m < p + n) = (m < n)
leq_addr      n <= n + m
addn_gt0      (0 < m + n) = (0 < m) || (0 < n)
subn_gt0      (0 < n - m) = (m < n)
leq_subLR     (m - n <= p) = (m <= n + p)
ltn_sub2r     p < n -> m < n -> m - p < n - p
ltn_subRL     (n < p - m) = (m + n < p)
```

```
subnKC            m <= n -> m + (n - m) = n
subnK             m <= n -> (n - m) + m = n
addnBA            p <= n -> m + (n - p) = m + n - p
subnBA            p <= n -> m - (n - p) = m + p - n
subKn             m <= n -> n - (n - m) = m
leq_sub2r         m <= n -> m - p <= n - p
ltn_subRL         (n < p - m) = (m + n < p)
mul0n/muln0       left_zero 0 muln/right_zero 0 muln
mul1n/muln1       left_id 1 muln/right_id 1 muln
mulnC             commutative muln
mulnA             associative muln
mulSn             m.+1 * n = n + m * n
mulnS             m * n.+1 = m + m * n
mulnDl            left_distributive muln addn
mulnDr            right_distributive muln addn
mulnBl            left_distributive muln subn
mulnBr            right_distributive muln subn
mulnCA            left_commutative muln
muln_gt0          (0 < m * n) = (0 < m) && (0 < n)
leq_pmulr         n > 0 -> m <= m * n
leq_mul2l         (m * n1 <= m * n2) = (m == 0) || (n1 <= n2)
leq_pmul2r        0 < m -> (n1 * m <= n2 * m) = (n1 <= n2)
ltn_pmul2r        0 < m -> (n1 * m < n2 * m) = (n1 < n2)
leqP              leq_xor_gtn m n (m <= n) (n < m)
ltngtP            compare_nat m n (m < n) (n < m) (m == n)
expn0             m ^ 0 = 1
expn1             m ^ 1 = m
expnS             m ^ n.+1 = m * m ^ n
exp0n             0 < n -> 0 ^ n = 0
exp1n             1 ^ n =1
expnD             m ^ (n1 + n2) =m ^ n1 * m ^ n2
expn_gt0          (0 < m ^ n) = (0 < m) || (n == 0)
fact0             0`! = 1
factS             (n.+1)`! = n.+1 *n`!
mul2n/muln2       2 * m = m.*2/m * 2 = m.*2
odd_add           odd (m + n) = odd m (+) odd n
odd_double_half   odd n + n./2.*2 = n
```

```
CoInductive leq_xor_gtn m n : bool -> bool -> Set :=
  | LeqNotGtn of m <= n : leq_xor_gtn m n true false
  | GtnNotLeq of n < m  : leq_xor_gtn m n false true.
CoInductive compare_nat m n : bool -> bool -> bool -> Set :=
  | CompareNatLt of m < n : compare_nat m n true false false
  | CompareNatGt of m > n : compare_nat m n false true false
  | CompareNatEq of m = n : compare_nat m n false false true.
```