cheat sheet ssrbool.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

```
bool_scope
Notation "~~ b":= (negb b)
Notation "b ==>c":= (implb b c).
Notation "b1 (+) b2":= (addb b1 b2).
Notation "a && b":=  (andb a b) (NB: generalization [ && b1 , b2 , .. , bn & c ])
Notation "a || b" :=  (orb a b) (NB: generalization [ || b1 , b2 , .. , bn |c ])
Notation "x \in A" :=  (in_mem x (mem A)).
Notation "x \notin A" :=  (~~ (x \in A)).
```

```
negbT      b = false -> ~~ b
negbTE     ~~ b -> b = false
negbK      involutive negb
contra     (c -> b) -> ~~ b -> ~~ c
contraNF   (c -> b) -> ~~ b -> c = false
contraFF   (c -> b) -> b = false -> c = false
ifP        if_spec (b = false) b (if b then vT else vF)
ifT        b -> (if b then vT else vF) = vT
ifF        b = false -> (if b then vT else vF) = vF
ifN        ~~ b -> (if b then vT else vF) = vF
boolP      alt_spec b1 b1 b1
negP       reflect (~ b1) (~~ b1)
negPn      reflect b1 (~~ ~~ b1)
andP       reflect (b1 /\ b2) (b1 && b2)
orP        reflect (b1 \/ b2) (b1 || b2)
nandP      reflect (~~ b1 \/ ~~ b2) (~~ (b1 && b2))
norP       reflect (~~ b1 /\ ~~ b2) (~~ (b1 || b2))
implyP     reflect (b1 -> b2) (b1 ==> b2)
andTb      left_id true andb
andbT      right_id true andb
andbb      idempotent andb
andbC      commutative andb
andbA      associative andb
orFb       left_id false orb
orbN       b || ~~ b = true
negb_and   ~~ (a && b) = ~~ a ||  ~~ b
negb_or    ~~ (a ||  b) = ~~ a && ~~ b
```

```
CoInductive if_spec (not_b : Prop) : bool -> A -> Set :=
  | IfSpecTrue  of       b : if_spec not_b true vT
  | IfSpecFalse of   not_b : if_spec not_b false vF.

Inductive reflect (P : Prop) : bool -> Set :=
  | ReflectT  of    P : reflect P true
  | ReflectF of ~ P : reflect P false.

CoInductive alt_spec : bool -> Type :=
  | AltTrue of     P : alt_spec true
  | AltFalse of ~~ b : alt_spec false.
```

```
Notation xpred0 := (fun _ => false).
Notation xpredT := (fun _ => true).
Notation xpredU := (fun (p1 p2 : pred _) x => p1 x || p2 x).
Notation xpredC := (fun (p : pred _) x =>~~ p x).
Notation "A =i B" :=  (eq_mem (mem A) (mem B)).
```