cheat sheet ssrbool.v (Coq v8.16)

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`	`e □ x = x`
`right_id e op`	`x □ e = x`
`left_zero z op`	`z □ x = z`
`right_zero z op`	`x □ z = z`
`self_inverse e op`	`x □ x = e`
`idempotent op`	`x □ x = x`
`commutative op`	`x □ y = y □ x`
`associative op`	`x □ (y □ z) = (x □ y) □ z`
`right_commutative op`	`(x □ y) □ z = (x □ z) □ y`
`left_commutative op`	`x □ (y □ z) = y □ (x □ z)`
`left_distributive op add`	`(x + y) * z = (x * z) + (y * z)`
`right_distributive op add`	`x * (y + z) = (x * y) + (x * z)`
`left_loop inv op`	`cancel (op x) (op (inv x))`

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)   Generalized to [&& b1 , b2 , ... , bn & c ]
Notation "a || b" :=  (orb a b)   Generalized to [|| 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

Variant 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.

Variant alt_spec (P : Prop) (b : bool) : bool -> Type :=
  | AltTrue of     P : alt_spec P b true
  | AltFalse of ~~ b : alt_spec P b 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)).
```