INSTITUTE OF MATHEMATICS FOR INDUSTRY,
KYUSHU UNIVERSITY

LOGIC AND COMPUTATION PROJECT

Coq Modules for Relational Calculus

(Ver.0.1)

Hisaharu TANAKA
Saga University

Toshiaki MATSUSHIMA
Kyushu University

Shuichi INOKUCHI
Fukuoka Institute of Technology

Yoshihiro MIZOGUCHI
Kyushu University

May 3, 2016

Contents

Chapter 1

Library Basic_Notations

1.1

> - "   - Dedekind   -"
>   .
>
> - ,
>   - 1.4   , 1.5 - 1.6
>   - 2.1 - 2.3   , 2.4 - 2.5   , 2.6   , 2.7   , 2.8
>   - 4.2 - 4.3   , 4.4 - 4.5   , 4.6   4.6.1, 4.7   , 4.8
>   4.8.1, 4.9
>   - 5.1   , 5.2 - 5.3
>
>   .
>
> - 3   ,   , Coq
>   ,   ,
>   .

1.2

- A, B eqType   , A B (Rel A B)   , $A \to B \to$ Prop
  .   (Rel A B) $A \to B$.

- $\alpha : A \to B$ $\alpha^\sharp : B \to A$ (inverse α)   , Coq   (α #)   .

4

- 2 $\alpha : A \rightharpoonup B, \beta : B \rightharpoonup C$ $\alpha \cdot \beta (\text{or } \alpha\beta) : A \rightharpoonup C$ (composite α β) , $(\alpha \quad \beta)$.

- $\alpha \rhd \beta : A \rightharpoonup C$ (residual α β)   , $(\alpha \quad \beta)$.

- $\mathrm{id}_A : A \rightharpoonup A$ (identity A)   , (Id A)   .

- $\phi_{AB} : A \rightharpoonup B$ (empty A B)   , (A B)   .

- $\nabla_{AB} : A \rightharpoonup B$ (universal A B)   , (A B)   .

- 2 $\alpha : A \rightharpoonup B, \beta : A \rightharpoonup B$ $\alpha \sqcup \beta : A \rightharpoonup B$ (cup α β)   , $(\alpha \quad \beta)$.

- $\alpha \sqcap \beta : A \rightharpoonup B$ (cap α β)   , $(\alpha \quad \beta)$.

- $\alpha \Rightarrow \beta : A \rightharpoonup B$ (rpc α β)   , $(\alpha$ >> $\beta)$.

- $\alpha : A \rightharpoonup B$ $\alpha^- : A \rightharpoonup B$ (complement α)   , Coq $(\alpha$ ^) .

- 2 $\alpha : A \rightharpoonup B, \beta : A \rightharpoonup B$ $\alpha - \beta : A \rightharpoonup B$ (difference α β)   , $(\alpha$ -- $\beta)$.

- (cupP)   (capP)   , P , $\{f(\alpha) \mid P(\alpha)\}$, .

- , 1 $I = \{*\}$ i   .

- ,   ,   ,   .

- , $\alpha \rhd \beta$ $(\alpha \cdot \beta^-)^-$,"   , $(\alpha \cdot \beta^-)^-$ "   ,   .

1.1   .

Axiom $prop\_extensionality\_ok$: $prop\_extensionality$.

Definition $Rel\ (A\ B : eqType) := A \rightarrow B \rightarrow$ Prop.

Module Type $Relation$.

Parameter $inverse$: $(\forall\ A\ B : eqType,\ Rel\ A\ B \rightarrow Rel\ B\ A)$.
Notation "a #" := $(inverse\ \_\ \_\ a)$ (at level 20).
Parameter $composite$: $(\forall\ A\ B\ C : eqType,\ Rel\ A\ B \rightarrow Rel\ B\ C \rightarrow Rel\ A\ C)$.
Notation "a ' ' b" := $(composite\ \_\ \_\ \_\ a\ b)$ (at level 50).
Parameter $residual$: $(\forall\ A\ B\ C : eqType,\ Rel\ A\ B \rightarrow Rel\ B\ C \rightarrow Rel\ A\ C)$.
Notation "a ' ' b" := $(residual\ \_\ \_\ \_\ a\ b)$ (at level 50).

		Coq	Notation
	$\alpha^{\sharp}$	(inverse α)	(α #)
	$\alpha \cdot \beta$	(composite α β)	(α β)
	$\alpha \rhd \beta$	(residual α β)	(α β)
	id_A	(identity A)	(Id A)
	ϕ_{AB}	(empty A B)	(A B)
	∇_{AB}	(universal A B)	(A B)
	$\alpha \sqcup \beta$	(cup α β)	(α β)
	$\alpha \sqcap \beta$	(cap α β)	(α β)
	$\alpha \Rightarrow \beta$	(rpc α β)	(α >> β)
	α^{-}	(complement α)	(α ^)
	$\alpha - \beta$	(difference α β)	(α -- β)
	$\sqcup_{P(\alpha)} f(\alpha)$	(cupP P f)	(_{P} f)
	$\sqcap_{P(\alpha)} f(\alpha)$	(capP P f)	(_{P} f)

Table 1.1:

Parameter *identity* : ($\forall$ A : *eqType*, *Rel A A*).
Notation "'Id'" := *identity*.
Parameter *empty* : ($\forall$ A B : *eqType*, *Rel A B*).
Notation "'   '" := *empty*.
Parameter *universal* : ($\forall$ A B : *eqType*, *Rel A B*).
Notation "'   '" := *universal*.

Parameter *include* : ($\forall$ A B : *eqType*, *Rel A B* $\to$ *Rel A B* $\to$ Prop).
Notation "a '   ' b" := (*include* _ _ a b) (at level 50).

Parameter *cupP* : ($\forall$ A B C D : *eqType*, (*Rel C D* $\to$ Prop) $\to$ (*Rel C D* $\to$ *Rel A B*) $\to$ *Rel A B*).
Notation "'   _{' p '}' f" := (*cupP* _ _ _ _ p f) (at level 50).
Parameter *capP* : ($\forall$ A B C D : *eqType*, (*Rel C D* $\to$ Prop) $\to$ (*Rel C D* $\to$ *Rel A B*) $\to$ *Rel A B*).
Notation "'   _{' p '}' f" := (*capP* _ _ _ _ p f) (at level 50).

Definition *cup* {A B : *eqType*} (*alpha* beta : *Rel A B*)
 :=   _{fun *gamma* : *Rel A B* $\Rightarrow$ *gamma* = *alpha* $\lor$ *gamma* = beta} *id*.
Notation "a '   ' b" := (*cup* a b) (at level 50).
Definition *cap* {A B : *eqType*} (*alpha* beta : *Rel A B*)
 :=   _{fun *gamma* : *Rel A B* $\Rightarrow$ *gamma* = *alpha* $\lor$ *gamma* = beta} *id*.
Notation "a '   ' b" := (*cap* a b) (at level 50).
Parameter *rpc* : ($\forall$ A B : *eqType*, *Rel A B* $\to$ *Rel A B* $\to$ *Rel A B*).
Notation "a '»' b" := (*rpc* _ _ a b) (at level 50).
Definition *complement* {A B : *eqType*} (*alpha* : *Rel A B*) := *alpha* » A B.

Notation "a '^'" := (*complement a*) (at level 20).
Definition *difference* {*A B* : *eqType*} (*alpha* beta : *Rel A B*) := *alpha*   beta ^.
Notation "a − b" := (*difference a b*) (at level 50).

Notation "'i'" := *unit_eqType*.

1.3

$\alpha : A \rightarrow B$,   total_r,   univalent_r,   function_r, surjective_r,   injective_r,   bijection_r   .

- total_r : $id_A \sqsubseteq \alpha \cdot \alpha^\sharp$

- univalent_r : $\alpha^\sharp \cdot \alpha \sqsubseteq id_B$

- function_r : $id_A \sqsubseteq \alpha \cdot \alpha^\sharp \wedge \alpha^\sharp \cdot \alpha \sqsubseteq id_B$

- surjection_r : $id_A \sqsubseteq \alpha \cdot \alpha^\sharp \wedge \alpha^\sharp \cdot \alpha = id_B$

- injection_r : $id_A = \alpha \cdot \alpha^\sharp \wedge \alpha^\sharp \cdot \alpha \sqsubseteq id_B$

- bijection_r : $id_A = \alpha \cdot \alpha^\sharp \wedge \alpha^\sharp \cdot \alpha = id_B$

Definition *total_r* {*A B* : *eqType*} (*alpha* : *Rel A B*) := (*Id A*)   (*alpha*   *alpha* #).
Definition *univalent_r* {*A B* : *eqType*} (*alpha* : *Rel A B*) := (*alpha* #   *alpha*)   (*Id B*).
Definition *function_r* {*A B* : *eqType*} (*alpha* : *Rel A B*)
 := (*total_r alpha*) ∧ (*univalent_r alpha*).
Definition *surjection_r* {*A B* : *eqType*} (*alpha* : *Rel A B*)
 := (*function_r alpha*) ∧ (*total_r* (*alpha* #)).
Definition *injection_r* {*A B* : *eqType*} (*alpha* : *Rel A B*)
 := (*function_r alpha*) ∧ (*univalent_r* (*alpha* #)).
Definition *bijection_r* {*A B* : *eqType*} (*alpha* : *Rel A B*)
 := (*function_r alpha*) ∧ (*total_r* (*alpha* #)) ∧ (*univalent_r* (*alpha* #)).

1.4

                              ,   ,
                    .

1.4.1 Dedekind

Axiom 1 (comp_id_l, comp_id_r) *Let $\alpha : A \rightharpoonup B$. Then,*

$$id_A \cdot \alpha = \alpha \cdot id_B = \alpha.$$

Definition *axiom1a* := $\forall$ *(A B : eqType)(alpha : Rel A B), Id A   alpha = alpha.*
Axiom *comp_id_l* : *axiom1a.*
Definition *axiom1b* := $\forall$ *(A B : eqType)(alpha : Rel A B), alpha   Id B = alpha.*
Axiom *comp_id_r* : *axiom1b.*

Axiom 2 (comp_assoc) *Let $\alpha : A \rightharpoonup B$, $\beta : B \rightharpoonup C$, and $\gamma : C \rightharpoonup D$. Then,*

$$(\alpha \cdot \beta) \cdot \gamma = \alpha \cdot (\beta \cdot \gamma).$$

Definition *axiom2* :=
 $\forall$ *(A B C D : eqType)(alpha : Rel A B)*(beta : *Rel B C)(gamma : Rel C D)*,
 (alpha   beta*)   gamma = alpha*   (beta   *gamma).*
Axiom *comp_assoc* : *axiom2.*

Axiom 3 (inc_refl) *Let $\alpha : A \rightharpoonup B$. Then,*

$$\alpha \sqsubseteq \alpha.$$

Definition *axiom3* := $\forall$ *(A B : eqType)(alpha : Rel A B), alpha   alpha.*
Axiom *inc_refl* : *axiom3.*

Axiom 4 (inc_trans) *Let $\alpha, \beta, \gamma : A \rightharpoonup B$. Then,*

$$\alpha \sqsubseteq \beta \wedge \beta \sqsubseteq \gamma \Rightarrow \alpha \sqsubseteq \gamma.$$

Definition *axiom4* := $\forall$ *(A B : eqType)(alpha* beta gamma : *Rel A B)*,
 alpha   beta $\rightarrow$ beta   *gamma* $\rightarrow$ *alpha*   *gamma.*
Axiom *inc_trans* : *axiom4.*

Axiom 5 (inc_antisym) *Let $\alpha, \beta : A \rightharpoonup B$. Then,*

$$\alpha \sqsubseteq \beta \wedge \beta \sqsubseteq \alpha \Rightarrow \alpha = \beta.$$

Definition *axiom5* := $\forall$ *(A B : eqType)(alpha* beta : *Rel A B)*,
 alpha   beta $\rightarrow$ beta   *alpha* $\rightarrow$ *alpha* = beta.
Axiom *inc_antisym* : *axiom5.*

Axiom 6 (inc_empty_alpha) *Let $\alpha : A \to B$. Then,*

$$\phi_{AB} \sqsubseteq \alpha.$$

Definition *axiom6* := $\forall$ (*A B* : *eqType*)(*alpha* : *Rel A B*),   *A B*   *alpha*.
Axiom *inc_empty_alpha* : *axiom6*.

Axiom 7 (inc_alpha_universal) *Let $\alpha : A \to B$. Then,*

$$\alpha \sqsubseteq \nabla_{AB}.$$

Definition *axiom7* := $\forall$ (*A B* : *eqType*)(*alpha* : *Rel A B*), *alpha*   *A B*.
Axiom *inc_alpha_universal* : *axiom7*.

Axiom 8 (inc_capP, inc_cap)

    1. **inc_capP :** *Let $\alpha : A \to B$, $f : (C \to D) \to (A \to B)$ and P : predicate. Then,*

$$\alpha \sqsubseteq (\sqcap_{P(\beta)} f(\beta)) \Leftrightarrow \forall \beta : C \to D, P(\beta) \Rightarrow \alpha \sqsubseteq f(\beta).$$

    2. **inc_cap :** *Let $\alpha, \beta, \gamma : A \to B$. Then,*

$$\alpha \sqsubseteq (\beta \sqcap \gamma) \Leftrightarrow \alpha \sqsubseteq \beta \wedge \alpha \sqsubseteq \gamma.$$

Definition *axiom8a* :=
 $\forall$ (*A B C D* : *eqType*)
 (*alpha* : *Rel A B*)(*f* : *Rel C D $\to$ Rel A B*)(*P* : *Rel C D $\to$* Prop),
 alpha   (_{*P*} *f*) $\leftrightarrow$ $\forall$ beta : *Rel C D*, *P* beta $\to$ *alpha*   *f* beta.
Axiom *inc_capP* : *axiom8a*.
Definition *axiom8b* := $\forall$ (*A B* : *eqType*)(*alpha* beta *gamma* : *Rel A B*),
 alpha   (beta   *gamma*) $\leftrightarrow$ (*alpha*   beta) $\wedge$ (*alpha*   *gamma*).
Lemma *inc_cap* : *axiom8b*.
Proof.
move $\Rightarrow$ *A B alpha* beta *gamma*.
rewrite *inc_capP*.
split; move $\Rightarrow$ *H*.
split; apply *H*.
by [left].
by [right].
move $\Rightarrow$ delta *H0*.
case *H0* $\Rightarrow$ *H1*; rewrite *H1*; apply *H*.
Qed.

Axiom 9 (inc_cupP, inc_cup)

    1. **inc_cupP :** *Let $\alpha : A \rightharpoonup B$, $f : (C \rightharpoonup D) \to (A \rightharpoonup B)$ and P : predicate. Then,*

$$(\sqcup_{P(\beta)} f(\beta)) \sqsubseteq \alpha \Leftrightarrow \forall \beta : C \rightharpoonup D, P(\beta) \Rightarrow f(\beta) \sqsubseteq \alpha.$$

    2. **inc_cup :** *Let $\alpha, \beta, \gamma : A \rightharpoonup B$. Then,*

$$(\beta \sqcup \gamma) \sqsubseteq \alpha \Leftrightarrow \beta \sqsubseteq \alpha \wedge \gamma \sqsubseteq \alpha.$$

Definition *axiom9a* :=
 $\forall$ (*A B C D* : *eqType*)
 (*alpha* : *Rel A B*)(*f* : *Rel C D* → *Rel A B*)(*P* : *Rel C D* → Prop),
 (_{*P*} *f*)   *alpha* ↔ $\forall$ beta : *Rel C D*, *P* beta → *f* beta   *alpha*.
Axiom *inc_cupP* : *axiom9a*.
Definition *axiom9b* := $\forall$ (*A B* : *eqType*)(*alpha* beta *gamma* : *Rel A B*),
 (beta   *gamma*)   *alpha* ↔ (beta   *alpha*) ∧ (*gamma*   *alpha*).
Lemma *inc_cup* : *axiom9b*.
Proof.
move ⇒ *A B alpha* beta *gamma*.
rewrite *inc_cupP*.
split; move ⇒ *H*.
split; apply *H*.
by [left].
by [right].
move ⇒ delta *H0*.
case *H0* ⇒ *H1*; rewrite *H1*; apply *H*.
Qed.

Axiom 10 (inc_rpc) *Let $\alpha, \beta, \gamma : A \rightharpoonup B$. Then,*

$$\alpha \sqsubseteq (\beta \Rightarrow \gamma) \Leftrightarrow (\alpha \sqcap \beta) \sqsubseteq \gamma.$$

Definition *axiom10* := $\forall$ (*A B* : *eqType*)(*alpha* beta *gamma* : *Rel A B*),
 alpha   (beta » *gamma*) ↔ (*alpha*   beta)   *gamma*.
Axiom *inc_rpc* : *axiom10*.

Axiom 11 (inv_invol) *Let $\alpha : A \rightharpoonup B$. Then,*

$$(\alpha^\sharp)^\sharp = \alpha.$$

Definition *axiom11* := $\forall$ (*A B* : *eqType*)(*alpha* : *Rel A B*), (*alpha* #) # = *alpha*.
Axiom *inv_invol* : *axiom11*.

Axiom 12 (comp_inv) *Let $\alpha : A \rightharpoonup B$ and $\beta : B \rightharpoonup C$. Then,*

$$(\alpha \cdot \beta)^\sharp = \beta^\sharp \cdot \alpha^\sharp.$$

Definition *axiom12* := ∀ (*A B C* : *eqType*)(*alpha* : *Rel A B*)(beta : *Rel B C*),
(*alpha* beta) # = (beta # *alpha* #).
Axiom *comp_inv* : *axiom12*.

Axiom 13 (inc_inv) *Let $\alpha, \beta : A \rightharpoonup B$. Then,*

$$\alpha \sqsubseteq \beta \Rightarrow \alpha^\sharp \sqsubseteq \beta^\sharp.$$

Definition *axiom13* :=
∀ (*A B* : *eqType*)(*alpha* beta : *Rel A B*), *alpha* beta → *alpha* # beta #.
Axiom *inc_inv* : *axiom13*.

Axiom 14 (dedekind) *Let $\alpha : A \rightharpoonup B$, $\beta : B \rightharpoonup C$, and $\gamma : A \rightharpoonup C$. Then,*

$$(\alpha \cdot \beta) \sqcap \gamma \sqsubseteq (\alpha \sqcap (\gamma \cdot \beta^\sharp)) \cdot (\beta \sqcap (\alpha^\sharp \cdot \gamma)).$$

Definition *axiom14* :=
∀ (*A B C* : *eqType*)(*alpha* : *Rel A B*)(beta : *Rel B C*)(*gamma* : *Rel A C*),
((*alpha* beta) *gamma*)
((*alpha* (*gamma* beta #)) (beta (*alpha* # *gamma*))).
Axiom *dedekind* : *axiom14*.

Axiom 15 (inc_residual) *Let $\alpha : A \rightharpoonup B$, $\beta : B \rightharpoonup C$, and $\gamma : A \rightharpoonup C$. Then,*

$$\gamma \sqsubseteq (\alpha \rhd \beta) \Leftrightarrow \alpha^\sharp \cdot \gamma \sqsubseteq \beta.$$

Definition *axiom15* :=
∀ (*A B C* : *eqType*)(*alpha* : *Rel A B*)(beta : *Rel B C*)(*gamma* : *Rel A C*),
gamma (*alpha* beta) ↔ (*alpha* # *gamma*) beta.
Axiom *inc_residual* : *axiom15*.

1.4.2

Dedekind , " " , Schröder

, Bool .

, Schröder .

Axiom 16 (complement_classic) *Let* $\alpha : A \rightharpoonup B$*. Then,*

$$\alpha \sqcup \alpha^- = \nabla_{AB}$$

Definition *axiom16* := $\forall$ $(A\ B : eqType)(alpha : Rel\ A\ B)$,
 alpha *alpha* ^ = *A B*.
Axiom *complement_classic* : *axiom16*.

1.4.3

1 *I*

$$\phi_{II} \neq id_I \wedge id_I = \nabla_{II} \wedge \forall X, \nabla_{XI} \cdot \nabla_{IX} = \nabla_{XX}$$

, Rel 2 , .

Axiom 17 (unit_universal)

$$\nabla_{AI} \cdot \nabla_{IA} = \nabla_{AA}$$

Definition *axiom17* := $\forall$ $(A : eqType)$, *A i* *i A* = *A A*.
Axiom *unit_universal* : *axiom17*.

1.4.4

" " , ( )

.

Axiom 18 (axiom_of_choice) *Let* $\alpha : A \rightharpoonup B$ *be a total relation. Then,*

$$\exists \beta : A \to B, \beta \sqsubseteq \alpha.$$

Definition *axiom18* := $\forall$ $(A\ B : eqType)(alpha : Rel\ A\ B)$,
 total_r alpha $\to$ $\exists$ beta : *Rel A B*, *function_r* beta $\wedge$ beta *alpha*.
Axiom *axiom_of_choice* : *axiom18*.

1.4.5

,

.

Axiom 19 (rationality) *Let* $\alpha : A \to B$. *Then,*

$$\exists R, \exists f : R \to A, \exists g : R \to B, \alpha = f^\sharp \cdot g \wedge f \cdot f^\sharp \sqcap g \cdot g^\sharp = id_R.$$

Definition *axiom19* := $\forall$ $(A\ B : eqType)(alpha : Rel\ A\ B)$,
 $\exists$ $(R : eqType)(f : Rel\ R\ A)(g : Rel\ R\ B)$,
 function_r f $\wedge$ *function_r g* $\wedge$ *alpha* = *f #   g* $\wedge$ $((f$ f #)) $(g$ g #)) = *Id R*.
Axiom *rationality* : *axiom19*.

1.4.6

,   .

Axiom 20 (pair_of_inclusions) $\exists j : A \to A + B, \exists k : B \to A + B,$

$$j \cdot j^\sharp = id_A \wedge k \cdot k^\sharp = id_B \wedge j \cdot k^\sharp = \phi_{AB} \wedge j^\sharp \cdot j \sqcup k^\sharp \cdot k = id_{A+B}.$$

Definition *axiom20* :=
 $\forall$ $(A\ B : eqType)$, $\exists$ $(j : Rel\ A\ (sum\_eqType\ A\ B))(k : Rel\ B\ (sum\_eqType\ A\ B))$,
 j   j # = *Id A* $\wedge$ *k   k #* = *Id B* $\wedge$ *j   k #* =   *A B* $\wedge$
 $(j$ # $j)$ $(k$ # $k)$ = *Id* $(sum\_eqType\ A\ B)$.
Axiom *pair_of_inclusions* : *axiom20*.

,
            (Axiom 19)   , Axiom 19
     "   "   ," `prod_eqType A B`
         "   .

Axiom 21 (pair_of_projections) $\exists p : A \times B \to A, \exists q : A \times B \to B,$

$$p^\sharp \cdot q = \nabla_{AB} \wedge p \cdot p^\sharp \sqcap q \cdot q^\sharp = id_{A \times B}.$$

Definition *axiom21* :=
 $\forall$ $(A\ B : eqType)$, $\exists$ $(p : Rel\ (prod\_eqType\ A\ B)\ A)(q : Rel\ (prod\_eqType\ A\ B)\ B)$,
 p #   q =   *A B* $\wedge$ $(p$ *p* #) $(q$ *q* #) = *Id* $(prod\_eqType\ A\ B)$ $\wedge$ *univalent_r p*
$\wedge$ *univalent_r q*.
Axiom *pair_of_projections* : *axiom21*.

End *Relation*.

Chapter 2

Library Basic_Notations_Set

Require Export *Basic_Notations*.
Require Import *Logic.FunctionalExtensionality*.
Require Import *Logic.Classical_Prop*.
Require Import *Logic.IndefiniteDescription*.
Require Import *Logic.ProofIrrelevance*.

2.1

<center>` '                                           '`</center>
<center>`.                    Basic_Notations                      .`</center>

Module *Rel_Set* <: *Relation*.

Definition *inverse* {*A B* : *eqType*} (*alpha* : *Rel A B*) : *Rel B A*
 := (fun (*b* : *B*)(*a* : *A*) ⇒ *alpha a b*).
Notation "a #" := (*inverse a*) (at level 20).
Definition *composite* {*A B C* : *eqType*} (*alpha* : *Rel A B*) (beta : *Rel B C*) : *Rel A C*
 := (fun (*a* : *A*)(*c* : *C*) ⇒ ∃ *b* : *B*, *alpha a b* ∧ beta *b c*).
Notation "a ' ' b" := (*composite a b*) (at level 50).
Definition *residual* {*A B C* : *eqType*} (*alpha* : *Rel A B*) (beta : *Rel B C*) : *Rel A C*
 := (fun (*a* : *A*)(*c* : *C*) ⇒ ∀ *b* : *B*, *alpha a b* → beta *b c*).
Notation "a ' ' b" := (*residual a b*) (at level 50).

Definition *identity* (*A* : *eqType*) : *Rel A A* := (fun *a a0* : *A* ⇒ *a* = *a0*).
Notation "'Id'" := *identity*.
Definition *empty* (*A B* : *eqType*) : *Rel A B* := (fun (*a* : *A*)(*b* : *B*) ⇒ *False*).
Notation "' '" := *empty*.
Definition *universal* (*A B* : *eqType*) : *Rel A B* := (fun (*a* : *A*)(*b* : *B*) ⇒ *True*).
Notation "' '" := *universal*.

Definition *include* {*A B* : *eqType*} (*alpha* beta : *Rel A B*) : Prop
 := (∀ (*a* : *A*)(*b* : *B*), *alpha a b* → beta *a b*).

<center>14</center>

Notation "a ' ' b" := (*include a b*) (at level 50).

Definition *cupP* {*A B C D* : *eqType*} (*P* : *Rel C D* → Prop) (*f* : *Rel C D* → *Rel A B*)
: *Rel A B*
 := (fun (*a* : *A*)(*b* : *B*) ⇒ ∃ *alpha* : *Rel C D*, *P alpha* ∧ (*f alpha*) *a b*).
Notation "' _{' p '}' f" := (*cupP p f*) (at level 50).
Definition *capP* {*A B C D* : *eqType*} (*P* : *Rel C D* → Prop) (*f* : *Rel C D* → *Rel A B*) :
Rel A B
 := (fun (*a* : *A*)(*b* : *B*) ⇒ ∀ *alpha* : *Rel C D*, *P alpha* → (*f alpha*) *a b*).
Notation "' _{' p '}' f" := (*capP p f*) (at level 50).

Definition *cup* {*A B* : *eqType*} (*alpha* beta : *Rel A B*)
 := _{fun *gamma* : *Rel A B* ⇒ *gamma* = *alpha* ∨ *gamma* = beta} *id*.
Notation "a ' ' b" := (*cup a b*) (at level 50).
Definition *cap* {*A B* : *eqType*} (*alpha* beta : *Rel A B*)
 := _{fun *gamma* : *Rel A B* ⇒ *gamma* = *alpha* ∨ *gamma* = beta} *id*.
Notation "a ' ' b" := (*cap a b*) (at level 50).
Definition *rpc* {*A B* : *eqType*} (*alpha* beta : *Rel A B*) : *Rel A B*
 := (fun (*a* : *A*)(*b* : *B*) ⇒ *alpha a b* → beta *a b*).
Notation "a '»' b" := (*rpc a b*) (at level 50).
Definition *complement* {*A B* : *eqType*} (*alpha* : *Rel A B*) := *alpha* » *A B*.
Notation "a '^'" := (*complement a*) (at level 20).
Definition *difference* {*A B* : *eqType*} (*alpha* beta : *Rel A B*) := *alpha* beta ^.
Notation "a – b" := (*difference a b*) (at level 50).

Notation "'i'" := *unit_eqType*.

2.2

Definition *total_r* {*A B* : *eqType*} (*alpha* : *Rel A B*) := (*Id A*) (*alpha* *alpha* #).
Definition *univalent_r* {*A B* : *eqType*} (*alpha* : *Rel A B*) := (*alpha* # *alpha*) (*Id B*).
Definition *function_r* {*A B* : *eqType*} (*alpha* : *Rel A B*)
 := (*total_r alpha*) ∧ (*univalent_r alpha*).
Definition *surjection_r* {*A B* : *eqType*} (*alpha* : *Rel A B*)
 := (*function_r alpha*) ∧ (*total_r* (*alpha* #)).
Definition *injection_r* {*A B* : *eqType*} (*alpha* : *Rel A B*)
 := (*function_r alpha*) ∧ (*univalent_r* (*alpha* #)).
Definition *bijection_r* {*A B* : *eqType*} (*alpha* : *Rel A B*)
 := (*function_r alpha*) ∧ (*total_r* (*alpha* #)) ∧ (*univalent_r* (*alpha* #)).

2.3

, ,
.

15

2.3.1 Dedekind

Lemma 1 (comp_id_l, comp_id_r) *Let $\alpha : A \rightarrow B$. Then,*

$$id_A \cdot \alpha = \alpha \cdot id_B = \alpha.$$

Definition *axiom1a* := $\forall$ *(A B : eqType)(alpha : Rel A B), Id A   alpha = alpha.*
Lemma *comp_id_l* : *axiom1a.*
Proof.
move $\Rightarrow$ *A B alpha.*
apply *functional_extensionality.*
move $\Rightarrow$ *a.*
apply *functional_extensionality.*
move $\Rightarrow$ *b.*
apply *prop_extensionality_ok.*
split.
elim $\Rightarrow$ *a0.*
elim $\Rightarrow$ *H H0.*
rewrite *H.*
apply *H0.*
move $\Rightarrow$ *H.*
$\exists$ *a.*
split.
by $\|$.
apply *H.*
Qed.

Definition *axiom1b* := $\forall$ *(A B : eqType)(alpha : Rel A B), alpha   Id B = alpha.*
Lemma *comp_id_r* : *axiom1b.*
Proof.
move $\Rightarrow$ *A B alpha.*
apply *functional_extensionality.*
move $\Rightarrow$ *a.*
apply *functional_extensionality.*
move $\Rightarrow$ *b.*
apply *prop_extensionality_ok.*
split.
elim $\Rightarrow$ *b0.*
elim $\Rightarrow$ *H H0.*
rewrite -*H0.*
apply *H.*
move $\Rightarrow$ *H.*
$\exists$ *b.*

```
split.
apply H.
by ||.
Qed.
```

> **Lemma 2 (comp_assoc)** *Let $\alpha : A \rightarrow B$, $\beta : B \rightarrow C$, and $\gamma : C \rightarrow D$. Then,*
>
> $$(\alpha \cdot \beta) \cdot \gamma = \alpha \cdot (\beta \cdot \gamma).$$

```
Definition axiom2 :=
 ∀ (A B C D : eqType)(alpha : Rel A B)(beta : Rel B C)(gamma : Rel C D),
 (alpha    beta)    gamma = alpha    (beta    gamma).
Lemma comp_assoc : axiom2.
Proof.
move ⇒ A B C D alpha beta gamma.
apply functional_extensionality.
move ⇒ a.
apply functional_extensionality.
move ⇒ d.
apply prop_extensionality_ok.
split.
elim ⇒ c.
elim ⇒ H H0.
elim H ⇒ b H1.
∃ b.
split.
apply H1.
∃ c.
split.
apply H1.
apply H0.
elim ⇒ b.
elim ⇒ H.
elim ⇒ c H0.
∃ c.
split.
∃ b.
split.
apply H.
apply H0.
apply H0.
Qed.
```

Lemma 3 (inc_refl) *Let $\alpha : A \rightarrow B$. Then,*

$$\alpha \sqsubseteq \alpha.$$

Definition *axiom3* := $\forall$ $(A\ B : eqType)(alpha : Rel\ A\ B),\ alpha \qquad alpha.$
Lemma *inc_refl* : *axiom3*.
Proof.
by [rewrite $/axiom3/include$].
Qed.

Lemma 4 (inc_trans) *Let $\alpha, \beta, \gamma : A \rightarrow B$. Then,*

$$\alpha \sqsubseteq \beta \wedge \beta \sqsubseteq \gamma \Rightarrow \alpha \sqsubseteq \gamma.$$

Definition *axiom4* := $\forall$ $(A\ B : eqType)(alpha$ beta $gamma : Rel\ A\ B),$
 $alpha \qquad beta \rightarrow beta \qquad gamma \rightarrow alpha \qquad gamma.$
Lemma *inc_trans* : *axiom4*.
Proof.
move $\Rightarrow$ *A B alpha* beta *gamma H H0 a b H1*.
apply $(H0\ \_\ \_ (H\ \_\ \_ H1))$.
Qed.

Lemma 5 (inc_antisym) *Let $\alpha, \beta : A \rightarrow B$. Then,*

$$\alpha \sqsubseteq \beta \wedge \beta \sqsubseteq \alpha \Rightarrow \alpha = \beta.$$

Definition *axiom5* := $\forall$ $(A\ B : eqType)(alpha$ beta $: Rel\ A\ B),$
 $alpha \qquad beta \rightarrow beta \qquad alpha \rightarrow alpha =$ beta.
Lemma *inc_antisym* : *axiom5*.
Proof.
move $\Rightarrow$ *A B alpha* beta *H H0*.
apply *functional_extensionality*.
move $\Rightarrow$ *a*.
apply *functional_extensionality*.
move $\Rightarrow$ *b*.
apply *prop_extensionality_ok*.
split.
apply *H*.
apply *H0*.
Qed.

Lemma 6 (inc_empty_alpha) *Let $\alpha : A \rightharpoonup B$. Then,*

$$\phi_{AB} \sqsubseteq \alpha.$$

Definition *axiom6* := $\forall$ $(A\ B : eqType)(alpha : Rel\ A\ B)$, $\quad A\ B \quad alpha$.
Lemma *inc_empty_alpha* : *axiom6*.
Proof.
move $\Rightarrow$ *A B alpha a b*.
apply *False_ind*.
Qed.

Lemma 7 (inc_alpha_universal) *Let $\alpha : A \rightharpoonup B$. Then,*

$$\alpha \sqsubseteq \nabla_{AB}.$$

Definition *axiom7* := $\forall$ $(A\ B : eqType)(alpha : Rel\ A\ B)$, $alpha \quad A\ B$.
Lemma *inc_alpha_universal* : *axiom7*.
Proof.
move $\Rightarrow$ *A B alpha a b H*.
apply *I*.
Qed.

Lemma 8 (inc_capP, inc_cap)

    1. **inc_capP :** *Let $\alpha : A \rightharpoonup B$, $f : (C \rightharpoonup D) \rightarrow (A \rightharpoonup B)$ and P : predicate. Then,*

$$\alpha \sqsubseteq (\sqcap_{P(\beta)} f(\beta)) \Leftrightarrow \forall \beta : C \rightharpoonup D, P(\beta) \Rightarrow \alpha \sqsubseteq f(\beta).$$

    2. **inc_cap :** *Let $\alpha, \beta, \gamma : A \rightharpoonup B$. Then,*

$$\alpha \sqsubseteq (\beta \sqcap \gamma) \Leftrightarrow \alpha \sqsubseteq \beta \wedge \alpha \sqsubseteq \gamma.$$

Definition *axiom8a* :=
 $\forall$ $(A\ B\ C\ D : eqType)$
 $(alpha : Rel\ A\ B)(f : Rel\ C\ D \rightarrow Rel\ A\ B)(P : Rel\ C\ D \rightarrow$ Prop$)$,
 alpha $(\quad \_\{P\}\ f) \leftrightarrow \forall$ beta $: Rel\ C\ D, P$ beta $\rightarrow alpha \quad f$ beta.
Lemma *inc_capP* : *axiom8a*.
Proof.
move $\Rightarrow$ *A B C D alpha f P*.
split; move $\Rightarrow$ *H*.
move $\Rightarrow$ beta *H0 a b H1*.
apply $(H\ \_\ \_\ H1\ \_\ H0)$.
move $\Rightarrow$ *a b H0* beta *H1*.

apply (H _ $H1$ _ _ $H0$).
Qed.
Definition $axiom8b := \forall$ (A B : $eqType$)($alpha$ beta $gamma$: Rel A B),
 $alpha$ (beta $gamma$) $\leftrightarrow$ ($alpha$ beta) $\wedge$ ($alpha$ $gamma$).
Lemma inc_cap : $axiom8b$.
Proof.
move $\Rightarrow$ A B $alpha$ beta $gamma$.
rewrite inc_capP.
split; move $\Rightarrow$ H.
split; apply H.
by $\lfloor$left$\rfloor$.
by $\lfloor$right$\rfloor$.
move $\Rightarrow$ delta $H0$.
case $H0$ $\Rightarrow$ $H1$; rewrite $H1$; apply H.
Qed.

Lemma 9 (inc_cupP, inc_cup)

  1. **inc_cupP :** *Let $\alpha : A \rightarrow B$, $f : (C \rightarrow D) \rightarrow (A \rightarrow B)$ and P : predicate. Then,*

$$(\sqcup_{P(\beta)} f(\beta)) \sqsubseteq \alpha \Leftrightarrow \forall \beta : C \rightarrow D, P(\beta) \Rightarrow f(\beta) \sqsubseteq \alpha.$$

  2. **inc_cup :** *Let $\alpha, \beta, \gamma : A \rightarrow B$. Then,*

$$(\beta \sqcup \gamma) \sqsubseteq \alpha \Leftrightarrow \beta \sqsubseteq \alpha \wedge \gamma \sqsubseteq \alpha.$$

Definition $axiom9a :=$
 $\forall$ (A B C D : $eqType$)
 ($alpha$: Rel A B)(f : Rel C D $\rightarrow$ Rel A B)(P : Rel C D $\rightarrow$ Prop),
 (_{P} f) $alpha$ $\leftrightarrow$ $\forall$ beta : Rel C D, P beta $\rightarrow$ f beta $alpha$.
Lemma inc_cupP : $axiom9a$.
Proof.
move $\Rightarrow$ A B C D $alpha$ f P.
split.
move $\Rightarrow$ H beta $H0$ a b $H1$.
apply H.
$\exists$ beta.
split.
apply $H0$.
apply $H1$.
move $\Rightarrow$ H a b.
elim $\Rightarrow$ beta.
elim $\Rightarrow$ $H0$ $H1$.

```
apply (H beta H0 _ _ H1).
Qed.
Definition axiom9b := ∀ (A B : eqType)(alpha beta gamma : Rel A B),
 (beta      gamma)    alpha ↔ (beta      alpha) ∧ (gamma      alpha).
Lemma inc_cup : axiom9b.
Proof.
move ⇒ A B alpha beta gamma.
rewrite inc_cupP.
split; move ⇒ H.
split; apply H.
by [left].
by [right].
move ⇒ delta H0.
case H0 ⇒ H1; rewrite H1; apply H.
Qed.
```

> **Lemma 10 (inc_rpc)** Let $\alpha, \beta, \gamma : A \to B$. Then,
>
> $$\alpha \sqsubseteq (\beta \Rightarrow \gamma) \Leftrightarrow (\alpha \sqcap \beta) \sqsubseteq \gamma.$$

```
Definition axiom10 := ∀ (A B : eqType)(alpha beta gamma : Rel A B),
 alpha      (beta » gamma) ↔ (alpha      beta)      gamma.
Lemma inc_rpc : axiom10.
Proof.
move ⇒ A B alpha beta gamma.
split; move ⇒ H.
move ⇒ a b H0.
apply H.
apply H0.
by [left].
apply H0.
by [right].
move ⇒ a b H0 H1.
apply H.
move ⇒ delta.
case ⇒ H2; rewrite H2.
apply H0.
apply H1.
Qed.
```

Lemma 11 (inv_invol) *Let $\alpha : A \rightarrow B$. Then,*

$$(\alpha^\sharp)^\sharp = \alpha.$$

Definition *axiom11* := $\forall$ (*A B* : *eqType*)(*alpha* : *Rel A B*), (*alpha* #) # = *alpha*.
Lemma *inv_invol* : *axiom11*.
Proof.
by [move $\Rightarrow$ *A B alpha*].
Qed.

Lemma 12 (comp_inv) *Let $\alpha : A \rightarrow B$ and $\beta : B \rightarrow C$. Then,*

$$(\alpha \cdot \beta)^\sharp = \beta^\sharp \cdot \alpha^\sharp.$$

Definition *axiom12* := $\forall$ (*A B C* : *eqType*)(*alpha* : *Rel A B*)(beta : *Rel B C*),
 (*alpha*   beta) # = (beta #   *alpha* #).
Lemma *comp_inv* : *axiom12*.
Proof.
move $\Rightarrow$ *A B C alpha* beta.
apply *functional_extensionality*.
move $\Rightarrow$ *c*.
apply *functional_extensionality*.
move $\Rightarrow$ *a*.
apply *prop_extensionality_ok*.
split; elim $\Rightarrow$ *b*.
elim $\Rightarrow$ *H H0*.
$\exists$ *b*.
split.
apply *H0*.
apply *H*.
elim $\Rightarrow$ *H H0*.
$\exists$ *b*.
split.
apply *H0*.
apply *H*.
Qed.

Lemma 13 (inc_inv) *Let $\alpha, \beta : A \rightarrow B$. Then,*

$$\alpha \sqsubseteq \beta \Rightarrow \alpha^\sharp \sqsubseteq \beta^\sharp.$$

Definition *axiom13* :=
 $\forall$ (*A B* : *eqType*)(*alpha* beta : *Rel A B*), *alpha*   beta $\rightarrow$ *alpha* #   beta #.

Lemma *inc_inv* : *axiom13*.
Proof.
move ⇒ *A B alpha* beta *H b a H0*.
apply (*H* _ _ *H0*).
Qed.

Lemma 14 (dedekind) *Let $\alpha : A \to B$, $\beta : B \to C$, and $\gamma : A \to C$. Then,*

$$(\alpha \cdot \beta) \sqcap \gamma \sqsubseteq (\alpha \sqcap (\gamma \cdot \beta^\sharp)) \cdot (\beta \sqcap (\alpha^\sharp \cdot \gamma)).$$

Definition *axiom14* :=
 ∀ (*A B C* : *eqType*)(*alpha* : *Rel A B*)(beta : *Rel B C*)(*gamma* : *Rel A C*),
 ((*alpha*   beta)   *gamma*)
   ((*alpha*   (*gamma*   beta #))   (beta   (*alpha* #   *gamma*))).
Lemma *dedekind* : *axiom14*.
Proof.
move ⇒ *A B C alpha* beta *gamma a c H*.
assert (∃ *b* : *B*, *alpha a b* ∧ beta *b c*).
apply *H*.
by [left].
elim *H0* ⇒ *b*.
elim ⇒ *H1 H2*.
∃ *b*.
repeat split.
move ⇒ delta *H3*.
case *H3* ⇒ *H4*; rewrite *H4*.
apply *H1*.
unfold *id*.
∃ *c*.
split.
apply *H*.
by [right].
apply *H2*.
move ⇒ delta *H3*.
case *H3* ⇒ *H4*; rewrite *H4*.
apply *H2*.
unfold *id*.
∃ *a*.
split.
apply *H1*.
apply *H*.
by [right].
Qed.

Lemma 15 (inc_residual) *Let $\alpha : A \rightharpoonup B$, $\beta : B \rightharpoonup C$, and $\gamma : A \rightharpoonup C$. Then,*

$$\gamma \sqsubseteq (\alpha \rhd \beta) \Leftrightarrow \alpha^\sharp \cdot \gamma \sqsubseteq \beta.$$

Definition *axiom15* :=
 $\forall\,(A\ B\ C : eqType)(alpha : Rel\ A\ B)(\mathtt{beta} : Rel\ B\ C)(gamma : Rel\ A\ C)$,
 $gamma \quad (alpha \quad \mathtt{beta}) \leftrightarrow (alpha \,\#\quad gamma) \quad \mathtt{beta}$.
Lemma *inc_residual* : *axiom15*.
Proof.
move $\Rightarrow$ *A B C alpha* beta *gamma*.
split; move $\Rightarrow$ *H*.
move $\Rightarrow$ *b c*.
elim $\Rightarrow$ *a H0*.
apply $(H\ a)$.
apply *H0*.
apply *H0*.
move $\Rightarrow$ *a c H0 b H1*.
apply *H*.
$\exists$ *a*.
split.
apply *H1*.
apply *H0*.
Qed.

2.3.2

Dedekind   ,   "   "   ,   Schröder
       , Bool   .
   ,   Schröder   .

Lemma 16 (complement_classic) *Let $\alpha : A \rightharpoonup B$. Then,*

$$\alpha \sqcup \alpha^- = \nabla_{AB}$$

Definition *axiom16* := $\forall\,(A\ B : eqType)(alpha : Rel\ A\ B)$,
 $alpha \quad alpha\ \hat{}\ = \quad A\ B$.
Lemma *complement_classic* : *axiom16*.
Proof.
move $\Rightarrow$ *A B alpha*.
apply *functional_extensionality*.
move $\Rightarrow$ *a*.
apply *functional_extensionality*.

move ⇒ *b*.
apply *prop_extensionality_ok*.
split; move ⇒ *H*.
apply *I*.
case (*classic* (*alpha a b*)) ⇒ *H0*.
∃ *alpha*.
split.
by [left].
apply *H0*.
∃ (fun (*a0* : *A*) (*b0* : *B*) ⇒ *alpha a0 b0* → *False*).
split.
by [right].
apply *H0*.
Qed.

2.3.3

1   *I*

$$\phi_{II} \neq id_I \wedge id_I = \nabla_{II} \wedge \forall X, \nabla_{XI} \cdot \nabla_{IX} = \nabla_{XX}$$

, Rel   2   ,   .

Lemma 17 (unit_universal)

$$\nabla_{AI} \cdot \nabla_{IA} = \nabla_{AA}$$

Definition *axiom17* := ∀ (*A* : *eqType*),   *A i*   *i A* =   *A A*.
Lemma *unit_universal* : *axiom17*.
Proof.
move ⇒ *A*.
apply *functional_extensionality*.
move ⇒ *a*.
apply *functional_extensionality*.
move ⇒ *a0*.
apply *prop_extensionality_ok*.
split; move ⇒ *H*.
apply *I*.
∃ *tt*.
by [].
Qed.

2.3.4

        ``   ''   ,   (   )

               .   .

Lemma 18 (axiom_of_choice) *Let $\alpha : A \rightharpoonup B$ be a total relation. Then,*

$$\exists \beta : A \to B, \beta \sqsubseteq \alpha.$$

`Definition` *axiom18* := $\forall$ (*A B* : *eqType*)(*alpha* : *Rel A B*),
 total_r alpha $\to$ $\exists$ `beta` : *Rel A B*, *function_r* `beta` $\land$ `beta`   *alpha*.
`Lemma` *axiom_of_choice* : *axiom18*.
`Proof`.
`move` $\Rightarrow$ *A B alpha*.
`rewrite` /*function_r*/*total_r*/*univalent_r*/*identity*/*include*/*composite*/*inverse*.
`move` $\Rightarrow$ *H*.
`assert` ($\forall$ *a* : *A*, {*b* : *B* | *alpha a b*}).
`move` $\Rightarrow$ *a*.
`apply` *constructive_indefinite_description*.
`move` : (*H a a* (*Logic.eq_refl a*)).
`elim` $\Rightarrow$ *b H0*.
$\exists$ *b*.
`apply` *H0*.
$\exists$ (`fun` (*a* : *A*)(*b* : *B*) $\Rightarrow$ *b* = *sval* (*X a*)).
`repeat` `split`.
`move` $\Rightarrow$ *a a0 H0*.
$\exists$ (*sval* (*X a*)).
`by` [`rewrite` *H0*].
`move` $\Rightarrow$ *b b0*.
`elim` $\Rightarrow$ *a*.
`elim` $\Rightarrow$ *H0 H1*.
`by` [`rewrite` *H0 H1*].
`move` $\Rightarrow$ *a b H0*.
`rewrite` *H0*.
`apply` *proj2_sig*.
`Qed`.

2.3.5

(Logic.IndefiniteDescription)

(Logic.ProofIrrelevance)   ,   .

Axiom   , Standard Library

.   .

Lemma 19 (rationality) *Let* $\alpha : A \rightarrow B$. *Then,*

$$\exists R, \exists f : R \rightarrow A, \exists g : R \rightarrow B, \alpha = f^{\sharp} \cdot g \wedge f \cdot f^{\sharp} \sqcap g \cdot g^{\sharp} = id_R.$$

,   .   bool   ,

sval (proj1_sig)   .

Lemma *is_true_inv0* : $\forall$ *P* : Prop, $\exists$ *b* : *bool*, *P* $\leftrightarrow$ *is_true b*.

Proof.

move $\Rightarrow$ *P*.

case (*classic P*); move $\Rightarrow$ *H*.

$\exists$ *true*.

split; move $\Rightarrow$ *H0*.

by $\|$.

apply *H*.

$\exists$ *false*.

split; move $\Rightarrow$ *H0*.

apply *False_ind*.

apply (*H H0*).

discriminate *H0*.

Qed.

Definition *is_true_inv* : Prop $\rightarrow$ *bool*.

move $\Rightarrow$ *P*.

move : (*is_true_inv0 P*) $\Rightarrow$ *H*.

apply *constructive_indefinite_description* in *H*.

apply *H*.

Defined.

Lemma *is_true_id* : $\forall$ *P* : Prop, *is_true* (*is_true_inv P*) $\leftrightarrow$ *P*.

Proof.

move $\Rightarrow$ *P*.

unfold *is_true_inv*.

move : (*constructive_indefinite_description* (fun *b* : *bool* $\Rightarrow$ *P* $\leftrightarrow$ *is_true b*) (*is_true_inv0 P*)) $\Rightarrow$ *x0*.

apply (@*sig_ind bool* (fun *b* $\Rightarrow$ (*P* $\leftrightarrow$ *is_true b*)) (fun *y* $\Rightarrow$ *is_true* (let (*x*, _) := *y* in *x*) $\leftrightarrow$ *P*)).

```
move ⇒ x H.
apply iff_sym.
apply H.
Qed.
Lemma sval_inv : ∀ (A : Type)(P : A → Prop)(x : sig P)(a : A), a = sval x → ∃ (H : P a),
x = exist P a H.
Proof.
move ⇒ A P x a H0.
rewrite H0.
∃ (proj2_sig x).
apply (@sig_ind A P (fun y ⇒ y = exist P (sval y) (proj2_sig y))).
move ⇒ a0 H.
by [simpl].
Qed.
Lemma sval_injective : ∀ (A : Type)(P : A → Prop)(x y : sig P), sval x = sval y → x = y.
Proof.
move ⇒ A P x y H.
move : (sval_inv A P y (sval x) H).
elim ⇒ H0 H1.
rewrite H1.
assert (H0 = proj2_sig x).
apply proof_irrelevance.
rewrite H2.
apply (@sig_ind A P (fun y ⇒ y = exist P (sval y) (proj2_sig y))).
move ⇒ a0 H3.
by [simpl].
Qed.
```

```
Definition axiom19 := ∀ (A B : eqType)(alpha : Rel A B),
 ∃ (R : eqType)(f : Rel R A)(g : Rel R B),
 function_r f ∧ function_r g ∧ alpha = f #    g ∧ ((f    f #)    (g    g #)) = Id R.
Lemma rationality : axiom19.
Proof.
move ⇒ A B alpha.
rewrite /function_r/total_r/univalent_r/cap/capP/identity/composite/inverse/include.
∃ (sig_eqType (fun x : prod_eqType A B ⇒ is_true_inv (alpha (fst x) (snd x)))).
∃ (fun x a ⇒ a = (fst (sval x))).
∃ (fun x b ⇒ b = (snd (sval x))).
simpl.
repeat split.
move ⇒ x x0 H.
∃ (fst (sval x)).
```

```
repeat split.
by [rewrite H].
move ⇒ a a0.
elim ⇒ x.
elim ⇒ H H0.
by [rewrite H H0].
move ⇒ x x0 H.
∃ (snd (sval x)).
repeat split.
by [rewrite H].
move ⇒ b b0.
elim ⇒ x.
elim ⇒ H H0.
by [rewrite H H0].
apply functional_extensionality.
move ⇒ a.
apply functional_extensionality.
move ⇒ b.
apply prop_extensionality_ok.
split; move ⇒ H.
assert (is_true (is_true_inv (alpha (fst (a,b)) (snd (a,b))))).
simpl.
apply is_true_id.
apply H.
∃ (exist (fun x ⇒ (is_true (is_true_inv (alpha (fst x) (snd x))))) (a,b) H0).
by [simpl].
elim H ⇒ x.
elim ⇒ H0 H1.
rewrite H0 H1.
apply is_true_id.
apply (@sig_ind (A × B) (fun x ⇒ is_true (is_true_inv (alpha (fst x) (snd x)))) (fun x
⇒ is_true (is_true_inv (alpha (fst (sval x)) (snd (sval x)))))).
simpl.
by [move ⇒ x0].
apply functional_extensionality.
move ⇒ y.
apply functional_extensionality.
move ⇒ y0.
apply prop_extensionality_ok.
split; move ⇒ H.
apply sval_injective.
move : (H (fun a c : {x : A × B | is_true (is_true_inv (alpha (fst x) (snd x)))} ⇒ ∃ b :
```

A, $b = fst$ ($sval\ a$) $\wedge\ b = fst$ ($sval\ c$)) ($or\_introl\ Logic.eq\_refl$)).
move : (H (**fun** $a\ c$: $\{x : A \times B \mid is\_true$ (is_true_inv ($alpha$ ($fst\ x$) ($snd\ x$)))$\} \Rightarrow \exists\ b$:
B, $b = snd$ ($sval\ a$) $\wedge\ b = snd$ ($sval\ c$)) ($or\_intror\ Logic.eq\_refl$)).
unfold id.
clear H.
elim $\Rightarrow b$.
elim $\Rightarrow H\ H0$.
elim $\Rightarrow a$.
elim $\Rightarrow H1\ H2$.
rewrite ($surjective\_pairing$ ($sval\ y0$)) $-H0$ $-H2$ H $H1$.
apply $surjective\_pairing$.
rewrite H.
move $\Rightarrow$ **beta** $H0$.
case $H0 \Rightarrow H1$; **rewrite** $H1$; **unfold** id.
$\exists$ (fst ($sval\ y0$)).
repeat split.
$\exists$ (snd ($sval\ y0$)).
repeat split.
Qed.

2.3.6

,   .

Lemma 20 (pair_of_inclusions) $\exists j : A \to A + B, \exists k : B \to A + B,$

$$j \cdot j^{\sharp} = id_A \wedge k \cdot k^{\sharp} = id_B \wedge j \cdot k^{\sharp} = \phi_{AB} \wedge j^{\sharp} \cdot j \sqcup k^{\sharp} \cdot k = id_{A+B}.$$

Definition $axiom20$:=
$\forall$ ($A\ B$: $eqType$), $\exists$ (j : $Rel\ A$ ($sum\_eqType\ A\ B$))(k : $Rel\ B$ ($sum\_eqType\ A\ B$)),
$j\quad\quad j\ \# = Id\ A \wedge k\quad\quad k\ \# = Id\ B \wedge j\quad\quad k\ \# =\quad\quad A\ B \wedge$
($j\ \#\quad\quad j$)$\quad\quad(k\ \#\quad\quad k) = Id$ ($sum\_eqType\ A\ B$).
Lemma $pair\_of\_inclusions$: $axiom20$.
Proof.
move $\Rightarrow A\ B$.
$\exists$ (**fun** (a : A)(x : $sum\_eqType\ A\ B$) $\Rightarrow x = inl\ a$).
$\exists$ (**fun** (b : B)(x : $sum\_eqType\ A\ B$) $\Rightarrow x = inr\ b$).
repeat split.
apply $functional\_extensionality$.
move $\Rightarrow a$.
apply $functional\_extensionality$.
move $\Rightarrow a0$.

```
apply prop_extensionality_ok.
split; move ⇒ H.
elim H ⇒ x.
elim ⇒ H0 H1.
rewrite H0 in H1.
by [injection H1].
∃ (inl a).
repeat split.
by [rewrite H].
apply functional_extensionality.
move ⇒ b.
apply functional_extensionality.
move ⇒ b0.
apply prop_extensionality_ok.
split; move ⇒ H.
elim H ⇒ x.
elim ⇒ H0 H1.
rewrite H0 in H1.
by [injection H1].
∃ (inr b).
repeat split.
by [rewrite H].
apply functional_extensionality.
move ⇒ a.
apply functional_extensionality.
move ⇒ b.
apply prop_extensionality_ok.
split; move ⇒ H.
elim H ⇒ x.
elim ⇒ H0 H1.
rewrite H0 in H1.
discriminate H1.
apply False_ind.
apply H.
apply functional_extensionality.
move ⇒ x.
apply functional_extensionality.
move ⇒ x0.
apply prop_extensionality_ok.
split.
elim ⇒ alpha.
elim ⇒ H0 H1.
```

```
case H0 ⇒ H2; rewrite H2 in H1.
elim H1 ⇒ a.
elim ⇒ H3 H4.
by [rewrite H3 H4].
elim H1 ⇒ b.
elim ⇒ H3 H4.
by [rewrite H3 H4].
assert ((∃ a : A, x = inl a) ∨ (∃ b : B, x = inr b)).
move : x.
apply sum_ind.
move ⇒ a.
left.
by [∃ a].
move ⇒ b.
right.
by [∃ b].
case H.
elim ⇒ a H0 H1.
∃ (fun x x0 ⇒ ∃ a0 : A, (x = inl a0 ∧ x0 = inl a0)).
split.
by [left].
∃ a.
by [rewrite -H1 H0].
elim ⇒ b H0 H1.
∃ (fun x x0 ⇒ ∃ b0 : B, (x = inr b0 ∧ x0 = inr b0)).
split.
by [right].
∃ b.
by [rewrite -H1 H0].
Qed.
```

> , .
>   (Axiom 19)   , Axiom 19
>   "   "   , "   prod_eqType A B
>   "   .
>
> **Lemma 21 (pair_of_projections)** $\exists p : A \times B \to A, \exists q : A \times B \to B,$
>
> $$p^\sharp \cdot q = \nabla_{AB} \wedge p \cdot p^\sharp \sqcap q \cdot q^\sharp = id_{A \times B}.$$

```
Definition axiom21 :=
 ∀ (A B : eqType), ∃ (p : Rel (prod_eqType A B) A)(q : Rel (prod_eqType A B) B),
 p #    q =    A B ∧ (p    p #)   (q    q #) = Id (prod_eqType A B) ∧ univalent_r p
```

$\wedge$ *univalent_r q*.
Lemma *pair_of_projections* : *axiom21*.
Proof.
move $\Rightarrow$ *A B*.
$\exists$ (fun (*x* : *prod_eqType A B*)(*a* : *A*) $\Rightarrow$ *a* = (*fst x*)).
$\exists$ (fun (*x* : *prod_eqType A B*)(*b* : *B*) $\Rightarrow$ *b* = (*snd x*)).
split.
apply *functional_extensionality*.
move $\Rightarrow$ *a*.
apply *functional_extensionality*.
move $\Rightarrow$ *b*.
apply *prop_extensionality_ok*.
split; move $\Rightarrow$ *H*.
apply *I*.
$\exists$ (*a*,*b*).
by [simpl].
split.
apply *functional_extensionality*.
move $\Rightarrow$ *x*.
apply *functional_extensionality*.
move $\Rightarrow$ *x0*.
apply *prop_extensionality_ok*.
split; move $\Rightarrow$ *H*.
move : (*H* (fun *a c* : *prod_eqType A B* $\Rightarrow$ $\exists$ *b* : *A*, *b* = *fst a* $\wedge$ *b* = *fst c*) (*or_introl*
Logic.eq_refl)).
move : (*H* (fun *a c* : *prod_eqType A B* $\Rightarrow$ $\exists$ *b* : *B*, *b* = *snd a* $\wedge$ *b* = *snd c*) (*or_intror*
Logic.eq_refl)).
unfold *id*.
clear *H*.
elim $\Rightarrow$ *b*.
elim $\Rightarrow$ *H H0*.
elim $\Rightarrow$ *a*.
elim $\Rightarrow$ *H1 H2*.
rewrite (*surjective_pairing x0*) -*H0* -*H2 H H1*.
apply *surjective_pairing*.
rewrite *H*.
move $\Rightarrow$ *alpha H0*.
case *H0* $\Rightarrow$ *H1*; rewrite *H1*; unfold *id*.
$\exists$ (*fst x0*).
repeat split.
$\exists$ (*snd x0*).
repeat split.

```
split.
move ⇒ a a0.
elim ⇒ x.
elim ⇒ H H0.
by [rewrite H H0].
move ⇒ b b0.
elim ⇒ x.
elim ⇒ H H0.
by [rewrite H H0].
Qed.

End Rel_Set.
```

Chapter 3

Library Basic_Lemmas

3.1

3.1.1   ,

Lemma 22 (cap_l) *Let* $\alpha, \beta : A \rightharpoonup B$. *Then,*

$$\alpha \sqcap \beta \sqsubseteq \alpha.$$

Lemma *cap_l* {*A B* : *eqType*} {*alpha* beta : *Rel A B*}: (*alpha*   beta)   *alpha*.
Proof.
assert ((*alpha*   beta)   (*alpha*   beta)).
apply *inc_refl*.
apply *inc_cap* in *H*.
apply *H*.
Qed.

Lemma 23 (cap_r) *Let* $\alpha, \beta : A \rightharpoonup B$. *Then,*

$$\alpha \sqcap \beta \sqsubseteq \beta.$$

Lemma *cap_r* {*A B* : *eqType*} {*alpha* beta : *Rel A B*}: (*alpha*   beta)   beta.
Proof.
assert ((*alpha*   beta)   (*alpha*   beta)).
apply *inc_refl*.

```
apply inc_cap in H.
apply H.
Qed.
```

Lemma 24 (cup_l) *Let $\alpha, \beta : A \rightarrow B$. Then,*

$$\alpha \sqsubseteq \alpha \sqcup \beta.$$

```
Lemma cup_l {A B : eqType} {alpha beta : Rel A B}: alpha    (alpha    beta).
Proof.
assert ((alpha    beta)    (alpha    beta)).
apply inc_refl.
apply inc_cup in H.
apply H.
Qed.
```

Lemma 25 (cup_r) *Let $\alpha, \beta : A \rightarrow B$. Then,*

$$\beta \sqsubseteq \alpha \sqcup \beta.$$

```
Lemma cup_r {A B : eqType} {alpha beta : Rel A B}: beta    (alpha    beta).
Proof.
assert ((alpha    beta)    (alpha    beta)).
apply inc_refl.
apply inc_cup in H.
apply H.
Qed.
```

Lemma 26 (inc_def1) *Let $\alpha, \beta : A \rightarrow B$. Then,*

$$\alpha = \alpha \sqcap \beta \Leftrightarrow \alpha \sqsubseteq \beta.$$

```
Lemma inc_def1 {A B : eqType} {alpha beta : Rel A B}:
 alpha = alpha    beta ↔ alpha    beta.
Proof.
split; move ⇒ H.
assert (alpha    (alpha    beta)).
rewrite -H.
apply inc_refl.
apply inc_cap in H0.
apply H0.
apply inc_antisym.
apply inc_cap.
```

```
split.
apply inc_refl.
apply H.
apply cap_l.
Qed.
```

Lemma 27 (inc_def2) *Let $\alpha, \beta : A \rightarrow B$. Then,*

$$\beta = \alpha \sqcup \beta \Leftrightarrow \alpha \sqsubseteq \beta.$$

```
Lemma inc_def2 {A B : eqType} {alpha beta : Rel A B}:
 beta = alpha    beta ↔ alpha    beta.
Proof.
split; move ⇒ H.
assert ((alpha    beta)    beta).
rewrite -H.
apply inc_refl.
apply inc_cup in H0.
apply H0.
apply inc_antisym.
assert ((alpha    beta)    (alpha    beta)).
apply inc_refl.
apply cup_r.
apply inc_cup.
split.
apply H.
apply inc_refl.
Qed.
```

Lemma 28 (cap_assoc) *Let $\alpha, \beta, \gamma : A \rightarrow B$. Then,*

$$(\alpha \sqcap \beta) \sqcap \gamma = \alpha \sqcap (\beta \sqcap \gamma).$$

```
Lemma cap_assoc {A B : eqType} {alpha beta gamma : Rel A B}:
 (alpha    beta)    gamma = alpha    (beta    gamma).
Proof.
apply inc_antisym.
rewrite inc_cap.
split.
apply (inc_trans _ _ _ (alpha    beta)).
apply cap_l.
apply cap_l.
rewrite inc_cap.
```

```
split.
apply (inc_trans _ _ _ (alpha ⊔ beta)).
apply cap_l.
apply cap_r.
apply cap_r.
rewrite inc_cap.
split.
rewrite inc_cap.
split.
apply cap_l.
apply (inc_trans _ _ _ (beta ⊓ gamma)).
apply cap_r.
apply cap_l.
apply (inc_trans _ _ _ (beta ⊓ gamma)).
apply cap_r.
apply cap_r.
Qed.
```

Lemma 29 (cup_assoc) *Let $\alpha, \beta, \gamma : A \rightarrow B$. Then,*

$$(\alpha \sqcup \beta) \sqcup \gamma = \alpha \sqcup (\beta \sqcup \gamma).$$

```
Lemma cup_assoc {A B : eqType} {alpha beta gamma : Rel A B}:
 (alpha ⊔ beta) ⊔ gamma = alpha ⊔ (beta ⊔ gamma).
Proof.
apply inc_antisym.
rewrite inc_cup.
split.
rewrite inc_cup.
split.
apply cup_l.
apply (inc_trans _ _ _ (beta ⊔ gamma)).
apply cup_l.
apply cup_r.
apply (inc_trans _ _ _ (beta ⊔ gamma)).
apply cup_r.
apply cup_r.
rewrite inc_cup.
split.
apply (inc_trans _ _ _ (alpha ⊔ beta)).
apply cup_l.
apply cup_l.
rewrite inc_cup.
```

```
split.
apply (inc_trans _ _ _ (alpha    beta)).
apply cup_r.
apply cup_l.
apply cup_r.
Qed.
```

Lemma 30 (cap_comm) *Let* $\alpha, \beta : A \rightharpoonup B$. *Then,*

$$\alpha \sqcap \beta = \beta \sqcap \alpha.$$

```
Lemma cap_comm {A B : eqType} {alpha beta : Rel A B}: alpha    beta = beta    alpha.
Proof.
apply inc_antisym.
rewrite inc_cap.
split.
apply cap_r.
apply cap_l.
rewrite inc_cap.
split.
apply cap_r.
apply cap_l.
Qed.
```

Lemma 31 (cup_comm) *Let* $\alpha, \beta : A \rightharpoonup B$. *Then,*

$$\alpha \sqcup \beta = \beta \sqcup \alpha.$$

```
Lemma cup_comm {A B : eqType} {alpha beta : Rel A B}: alpha    beta = beta    alpha.
Proof.
apply inc_antisym.
rewrite inc_cup.
split.
apply cup_r.
apply cup_l.
rewrite inc_cup.
split.
apply cup_r.
apply cup_l.
Qed.
```

Lemma 32 (cup_cap_abs) *Let $\alpha, \beta : A \rightarrow B$. Then,*

$$\alpha \sqcup (\alpha \sqcap \beta) = \alpha.$$

Lemma *cup_cap_abs* $\{A\ B : eqType\}$ $\{alpha\ \texttt{beta} : Rel\ A\ B\}$:
 alpha (*alpha* *beta*) = *alpha*.
Proof.
move : $(@cap\_l$ _ _ *alpha* beta$) \Rightarrow H$.
apply *inc_def2* in H.
by [rewrite *cup_comm* -H].
Qed.

Lemma 33 (cap_cup_abs) *Let $\alpha, \beta : A \rightarrow B$. Then,*

$$\alpha \sqcap (\alpha \sqcup \beta) = \alpha.$$

Lemma *cap_cup_abs* $\{A\ B : eqType\}$ $\{alpha\ \texttt{beta} : Rel\ A\ B\}$:
 alpha (*alpha* *beta*) = *alpha*.
Proof.
move : $(@cup\_l$ _ _ *alpha* beta$) \Rightarrow H$.
apply *inc_def1* in H.
by [rewrite -H].
Qed.

Lemma 34 (cap_idem) *Let $\alpha : A \rightarrow B$. Then,*

$$\alpha \sqcap \alpha = \alpha.$$

Lemma *cap_idem* $\{A\ B : eqType\}$ $\{alpha : Rel\ A\ B\}$: *alpha* *alpha* = *alpha*.
Proof.
apply *inc_antisym*.
apply *cap_l*.
apply *inc_cap*.
split; apply *inc_refl*.
Qed.

Lemma 35 (cup_idem) *Let $\alpha : A \rightarrow B$. Then,*

$$\alpha \sqcup \alpha = \alpha.$$

Lemma *cup_idem* $\{A\ B : eqType\}$ $\{alpha : Rel\ A\ B\}$: *alpha* *alpha* = *alpha*.
Proof.
apply *inc_antisym*.

```
apply inc_cup.
split; apply inc_refl.
apply cup_l.
Qed.
```

Lemma 36 (cap_inc_compat) *Let* $\alpha, \alpha', \beta, \beta' : A \rightarrow B$. *Then,*

$$\alpha \sqsubseteq \alpha' \wedge \beta \sqsubseteq \beta' \Rightarrow \alpha \sqcap \beta \sqsubseteq \alpha' \sqcap \beta'.$$

```
Lemma cap_inc_compat {A B : eqType} {alpha alpha' beta beta' : Rel A B}:
 alpha     alpha' → beta     beta' → (alpha     beta)     (alpha'     beta').
Proof.
move ⇒ H H0.
rewrite -inc_def1.
apply inc_def1 in H.
apply inc_def1 in H0.
rewrite cap_assoc -(@cap_assoc _ _ beta).
rewrite (@cap_comm _ _ beta).
rewrite cap_assoc -(@cap_assoc _ _ alpha).
by [rewrite -H -H0].
Qed.
```

Lemma 37 (cap_inc_compat_l) *Let* $\alpha, \beta, \beta' : A \rightarrow B$. *Then,*

$$\beta \sqsubseteq \beta' \Rightarrow \alpha \sqcap \beta \sqsubseteq \alpha \sqcap \beta'.$$

```
Lemma cap_inc_compat_l {A B : eqType} {alpha beta beta' : Rel A B}:
 beta     beta' → (alpha     beta)     (alpha     beta').
Proof.
move ⇒ H.
apply (@cap_inc_compat _ _ _ _ _ (@inc_refl _ _ alpha) H).
Qed.
```

Lemma 38 (cap_inc_compat_r) *Let* $\alpha, \alpha', \beta : A \rightarrow B$. *Then,*

$$\alpha \sqsubseteq \alpha' \Rightarrow \alpha \sqcap \beta \sqsubseteq \alpha' \sqcap \beta.$$

```
Lemma cap_inc_compat_r {A B : eqType} {alpha alpha' beta : Rel A B}:
 alpha     alpha' → (alpha     beta)     (alpha'     beta).
Proof.
move ⇒ H.
apply (@cap_inc_compat _ _ _ _ _ _ H (@inc_refl _ _ beta)).
Qed.
```

Lemma 39 (cup_inc_compat) *Let* $\alpha, \alpha', \beta, \beta' : A \rightarrow B$. *Then,*

$$\alpha \sqsubseteq \alpha' \wedge \beta \sqsubseteq \beta' \Rightarrow \alpha \sqcup \beta \sqsubseteq \alpha' \sqcup \beta'.$$

```
Lemma cup_inc_compat {A B : eqType} {alpha alpha' beta beta' : Rel A B}:
 alpha    alpha' → beta    beta' → (alpha    beta)    (alpha'    beta').
Proof.
move ⇒ H H0.
rewrite -inc_def2.
apply inc_def2 in H.
apply inc_def2 in H0.
rewrite cup_assoc -(@cup_assoc _ _ beta).
rewrite (@cup_comm _ _ beta).
rewrite cup_assoc -(@cup_assoc _ _ alpha).
by [rewrite -H -H0].
Qed.
```

Lemma 40 (cup_inc_compat_l) *Let* $\alpha, \beta, \beta' : A \rightarrow B$. *Then,*

$$\beta \sqsubseteq \beta' \Rightarrow \alpha \sqcup \beta \sqsubseteq \alpha \sqcup \beta'.$$

```
Lemma cup_inc_compat_l {A B : eqType} {alpha beta beta' : Rel A B}:
 beta    beta' → (alpha    beta)    (alpha    beta').
Proof.
move ⇒ H.
apply (@cup_inc_compat _ _ _ _ _ _ (@inc_refl _ _ alpha) H).
Qed.
```

Lemma 41 (cup_inc_compat_r) *Let* $\alpha, \alpha', \beta : A \rightarrow B$. *Then,*

$$\alpha \sqsubseteq \alpha' \Rightarrow \alpha \sqcup \beta \sqsubseteq \alpha' \sqcup \beta.$$

```
Lemma cup_inc_compat_r {A B : eqType} {alpha alpha' beta : Rel A B}:
 alpha    alpha' → (alpha    beta)    (alpha'    beta).
Proof.
move ⇒ H.
apply (@cup_inc_compat _ _ _ _ _ _ H (@inc_refl _ _ beta)).
Qed.
```

Lemma 42 (cap_empty) *Let* $\alpha : A \rightarrow B$. *Then,*

$$\alpha \sqcap \phi_{AB} = \phi_{AB}.$$

Lemma *cap_empty* {*A B* : *eqType*} {*alpha* : *Rel A B*}: *alpha*   *A B* =   *A B*.
Proof.
apply *inc_antisym*.
apply *cap_r*.
apply *inc_empty_alpha*.
Qed.

Lemma 43 (cup_empty) *Let* $\alpha : A \to B$. *Then,*

$$\alpha \sqcup \phi_{AB} = \alpha.$$

Lemma *cup_empty* {*A B* : *eqType*} {*alpha* : *Rel A B*}: *alpha*   *A B* = *alpha*.
Proof.
apply *inc_antisym*.
apply *inc_cup*.
split.
apply *inc_refl*.
apply *inc_empty_alpha*.
apply *cup_l*.
Qed.

Lemma 44 (cap_universal) *Let* $\alpha : A \to B$. *Then,*

$$\alpha \sqcap \nabla_{AB} = \alpha.$$

Lemma *cap_universal* {*A B* : *eqType*} {*alpha* : *Rel A B*}: *alpha*   *A B* = *alpha*.
Proof.
apply *inc_antisym*.
apply *cap_l*.
apply *inc_cap*.
split.
apply *inc_refl*.
apply *inc_alpha_universal*.
Qed.

Lemma 45 (cup_universal) *Let* $\alpha : A \to B$. *Then,*

$$\alpha \sqcup \nabla_{AB} = \nabla_{AB}.$$

Lemma *cup_universal* {*A B* : *eqType*} {*alpha* : *Rel A B*}: *alpha*   *A B* =   *A B*.
Proof.
apply *inc_antisym*.
apply *inc_cup*.

```
split.
apply inc_alpha_universal.
apply inc_refl.
apply cup_r.
Qed.
```

Lemma 46 (inc_lower) *Let $\alpha, \beta : A \rightarrow B$. Then,*

$$\alpha = \beta \Leftrightarrow (\forall \gamma : A \rightarrow B, \gamma \sqsubseteq \alpha \Leftrightarrow \gamma \sqsubseteq \beta).$$

```
Lemma inc_lower {A B : eqType} {alpha beta : Rel A B}:
  alpha = beta ↔ (∀ gamma : Rel A B, gamma    alpha ↔ gamma    beta).
Proof.
split; move ⇒ H.
move ⇒ gamma.
by [rewrite H].
apply inc_antisym.
rewrite -H.
apply inc_refl.
rewrite H.
apply inc_refl.
Qed.
```

Lemma 47 (inc_upper) *Let $\alpha, \beta : A \rightarrow B$. Then,*

$$\alpha = \beta \Leftrightarrow (\forall \gamma : A \rightarrow B, \alpha \sqsubseteq \gamma \Leftrightarrow \beta \sqsubseteq \gamma).$$

```
Lemma inc_upper {A B : eqType} {alpha beta : Rel A B}:
  alpha = beta ↔ (∀ gamma : Rel A B, alpha    gamma ↔ beta    gamma).
Proof.
split; move ⇒ H.
move ⇒ gamma.
by [rewrite H].
apply inc_antisym.
rewrite H.
apply inc_refl.
rewrite -H.
apply inc_refl.
Qed.
```

3.1.2

Lemma 48 (cap_cup_distr_l) *Let* $\alpha, \beta, \gamma : A \rightarrow B$. *Then,*

$$\alpha \sqcap (\beta \sqcup \gamma) = (\alpha \sqcap \beta) \sqcup (\alpha \sqcap \gamma).$$

Lemma *cap_cup_distr_l* $\{A\ B : eqType\}$ $\{alpha$ beta *gamma* $:\ Rel\ A\ B\}$:
 alpha   (beta   *gamma*) = (*alpha*   beta)   (*alpha*   *gamma*).
Proof.
apply *inc_upper*.
move $\Rightarrow$ delta.
split; move $\Rightarrow H$.
rewrite *cap_comm* (@*cap_comm* _ _ _ *gamma*).
apply *inc_cup*.
rewrite -*inc_rpc* -*inc_rpc*.
apply *inc_cup*.
rewrite *inc_rpc* *cap_comm*.
apply H.
rewrite *cap_comm* -*inc_rpc*.
apply *inc_cup*.
rewrite *inc_rpc* *inc_rpc*.
apply *inc_cup*.
rewrite *cap_comm* (@*cap_comm* _ _ *gamma*).
apply H.
Qed.

Lemma 49 (cap_cup_distr_r) *Let* $\alpha, \beta, \gamma : A \rightarrow B$. *Then,*

$$(\alpha \sqcup \beta) \sqcap \gamma = (\alpha \sqcap \gamma) \sqcup (\beta \sqcap \gamma).$$

Lemma *cap_cup_distr_r* $\{A\ B : eqType\}$ $\{alpha$ beta *gamma* $:\ Rel\ A\ B\}$:
 (*alpha*   beta)   *gamma* = (*alpha*   *gamma*)   (beta   *gamma*).
Proof.
rewrite (@*cap_comm* _ _ (*alpha*   beta)) (@*cap_comm* _ _ *alpha*) (@*cap_comm* _ _ beta).
apply *cap_cup_distr_l*.
Qed.

Lemma 50 (cup_cap_distr_l) *Let* $\alpha, \beta, \gamma : A \rightarrow B$. *Then,*

$$\alpha \sqcup (\beta \sqcap \gamma) = (\alpha \sqcup \beta) \sqcap (\alpha \sqcup \gamma).$$

Lemma *cup_cap_distr_l* $\{A\ B : eqType\}$ $\{alpha$ beta *gamma* $:\ Rel\ A\ B\}$:
 alpha   (beta   *gamma*) = (*alpha*   beta)   (*alpha*   *gamma*).

Proof.
rewrite *cap_cup_distr_l*.
rewrite (@*cap_comm* _ _ (*alpha*   beta)) *cap_cup_abs* (@*cap_comm* _ _ (*alpha*   beta)).
rewrite *cap_cup_distr_l*.
rewrite -*cup_assoc* (@*cap_comm* _ _ *gamma*) *cup_cap_abs*.
by [rewrite *cap_comm*].
Qed.

Lemma 51 (cup_cap_distr_r) *Let* $\alpha, \beta, \gamma : A \to B$*. Then,*

$$(\alpha \sqcap \beta) \sqcup \gamma = (\alpha \sqcup \gamma) \sqcap (\beta \sqcup \gamma).$$

Lemma *cup_cap_distr_r* {*A B* : *eqType*} {*alpha* beta *gamma* : *Rel A B*}:
 (*alpha*   beta)   *gamma* = (*alpha*   *gamma*)   (beta   *gamma*).
Proof.
rewrite (@*cup_comm* _ _ (*alpha*   beta)) (@*cup_comm* _ _ *alpha*) (@*cup_comm* _ _ beta).
apply *cup_cap_distr_l*.
Qed.

Lemma 52 (cap_cup_unique) *Let* $\alpha, \beta, \gamma : A \to B$*. Then,*

$$\alpha \sqcap \beta = \alpha \sqcap \gamma \wedge \alpha \sqcup \beta = \alpha \sqcup \gamma \Rightarrow \beta = \gamma.$$

Lemma *cap_cup_unique* {*A B* : *eqType*} {*alpha* beta *gamma* : *Rel A B*}:
 alpha   beta = *alpha*   *gamma* → *alpha*   beta = *alpha*   *gamma* → beta = *gamma*.
Proof.
move ⇒ *H H0*.
rewrite -(@*cap_cup_abs* _ _ beta *alpha*) *cup_comm H0*.
rewrite *cap_cup_distr_l*.
rewrite *cap_comm H*.
rewrite -*cap_cup_distr_r*.
rewrite *H0 cap_comm cup_comm*.
apply *cap_cup_abs*.
Qed.

3.1.3

$$\alpha : A \to B \quad , \qquad \beta : A \to B$$

$$\beta \sqsubseteq \alpha \Rightarrow \beta = \phi_{AB} \vee \beta = \alpha$$

, α (atomic)   .

Definition *atomic* $\{A\ B : eqType\}$ (*alpha* : *Rel A B*):=
  alpha $\neq$ *A B* $\wedge$ ($\forall$ beta : *Rel A B*, beta   *alpha* $\to$ beta =   *A B* $\vee$ beta = *alpha*).

Lemma 53 (atomic_cap_empty) *Let* $\alpha, \beta : A \to B$ *are atomic and* $\alpha \neq \beta$. *Then,*

$$\alpha \sqcap \beta = \phi_{AB}.$$

Lemma *atomic_cap_empty* $\{A\ B : eqType\}$ $\{alpha\ \text{beta} : Rel\ A\ B\}$:
  atomic alpha $\to$ *atomic* beta $\to$ *alpha* $\neq$ beta $\to$ *alpha*   beta =   *A B*.
Proof.
move $\Rightarrow$ *H H0*.
apply *or_to_imply*.
case (*classic* (*alpha*   beta =   *A B*)); move $\Rightarrow$ *H1*.
right.
apply *H1*.
left.
move $\Rightarrow$ *H2*.
apply *H2*.
apply *inc_antisym*.
apply *inc_def1*.
elim *H* $\Rightarrow$ *H3 H4*.
case (*H4* (*alpha*   beta) (*cap_l*)); move $\Rightarrow$ *H5*.
apply *False_ind*.
apply (*H1 H5*).
by [rewrite *H5*].
apply *inc_def1*.
elim *H0* $\Rightarrow$ *H3 H4*.
case (*H4* (*alpha*   beta) (*cap_r*)); move $\Rightarrow$ *H5*.
apply *False_ind*.
apply (*H1 H5*).
by [rewrite *cap_comm H5*].
Qed.

Lemma 54 (atomic_cup) *Let* $\alpha, \beta, \gamma : A \to B$ *and* α *is atomic. Then,*

$$\alpha \sqsubseteq \beta \sqcup \gamma \Rightarrow \alpha \sqsubseteq \beta \vee \alpha \sqsubseteq \gamma.$$

Lemma *atomic_cup* $\{A\ B : eqType\}$ $\{alpha\ \text{beta gamma} : Rel\ A\ B\}$:
  atomic alpha $\to$ *alpha*   (beta   *gamma*) $\to$ *alpha*   beta $\vee$ *alpha*   *gamma*.
Proof.
move $\Rightarrow$ *H H0*.
apply *inc_def1* in *H0*.
rewrite *cap_cup_distr_l* in *H0*.

```
elim H ⇒ H1 H2.
rewrite H0 in H1.
assert (alpha    beta ≠    A B ∨ alpha    gamma ≠    A B).
apply not_and_or.
elim ⇒ H3 H4.
rewrite H3 H4 in H1.
apply H1.
by [rewrite cup_empty].
case H3; move ⇒ H4.
left.
apply inc_def1.
case (H2 (alpha    beta) (cap_l)); move ⇒ H5.
apply False_ind.
apply (H4 H5).
by [rewrite H5].
right.
apply inc_def1.
case (H2 (alpha    gamma) (cap_l)); move ⇒ H5.
apply False_ind.
apply (H4 H5).
by [rewrite H5].
Qed.
```

3.2 Heyting

Lemma 55 (rpc_universal) *Let $\alpha : A \rightarrow B$. Then,*

$$(\alpha \Rightarrow \alpha) = \nabla_{AB}.$$

```
Lemma rpc_universal {A B : eqType} {alpha : Rel A B}: (alpha » alpha) =    A B.
Proof.
apply inc_lower.
move ⇒ gamma.
split; move ⇒ H.
apply inc_alpha_universal.
apply inc_rpc.
apply cap_r.
Qed.
```

Lemma 56 (rpc_r) *Let $\alpha, \beta : A \rightarrow B$. Then,*

$$(\alpha \Rightarrow \beta) \sqcap \beta = \beta.$$

Lemma *rpc_r* {*A B* : *eqType*} {*alpha* beta : *Rel A B*}: (*alpha* » beta)   beta = beta.
Proof.
assert (beta   (*alpha* » beta)).
apply *inc_rpc*.
apply *cap_l*.
apply *inc_def1* in *H*.
by [rewrite *cap_comm* -*H*].
Qed.

Lemma 57 (inc_def3) *Let $\alpha, \beta : A \rightarrow B$. Then,*

$$(\alpha \Rightarrow \beta) = \nabla_{AB} \Leftrightarrow \alpha \sqsubseteq \beta.$$

Lemma *inc_def3* {*A B* : *eqType*} {*alpha* beta : *Rel A B*}:
 (*alpha* » beta) =   *A B* ↔ *alpha*   beta.
Proof.
split; move ⇒ *H*.
rewrite -(@*rpc_universal* _ _ *alpha*) in *H*.
assert ((*alpha* » *alpha*)   (*alpha* » beta)).
rewrite *H*.
apply *inc_refl*.
apply *inc_rpc* in *H0*.
rewrite *rpc_r* in *H0*.
apply *H0*.
apply *inc_antisym*.
apply *inc_alpha_universal*.
rewrite -(@*rpc_universal* _ _ *alpha*).
apply *inc_rpc*.
rewrite *rpc_r*.
apply *H*.
Qed.

Lemma 58 (rpc_l) *Let $\alpha, \beta : A \rightarrow B$. Then,*

$$\alpha \sqcap (\alpha \Rightarrow \beta) = \alpha \sqcap \beta.$$

Lemma *rpc_l* {*A B* : *eqType*} {*alpha* beta : *Rel A B*}:
 alpha   (*alpha* » beta) = *alpha*   beta.
Proof.

49

apply *inc_lower*.
move ⇒ *gamma*.
split; move ⇒ *H*.
apply *inc_cap*.
apply *inc_cap* in *H*.
split.
apply *H*.
elim *H* ⇒ *H0 H1*.
apply *inc_rpc* in *H1*.
rewrite -(@*cap_idem* _ _ *gamma*).
apply (*inc_trans* _ _ _ (*gamma*   *alpha*)).
apply *cap_inc_compat*.
apply *inc_refl*.
apply *H0*.
apply *H1*.
apply *inc_cap*.
apply *inc_cap* in *H*.
split.
apply *H*.
apply *inc_rpc*.
apply (*inc_trans* _ _ _ *gamma*).
apply *cap_l*.
apply *H*.
Qed.

Lemma 59 (rpc_inc_compat) *Let* $\alpha, \alpha', \beta, \beta' : A \rightarrow B$. *Then,*

$$\alpha' \sqsubseteq \alpha \wedge \beta \sqsubseteq \beta' \Rightarrow (\alpha \Rightarrow \beta) \sqsubseteq (\alpha' \Rightarrow \beta').$$

Lemma *rpc_inc_compat* {*A B* : *eqType*} {*alpha alpha'* beta *beta'* : *Rel A B*}:
 alpha'   *alpha* → beta   *beta'* → (*alpha* » beta)   (*alpha'* » *beta'*).
Proof.
move ⇒ *H H0*.
apply *inc_rpc*.
apply (@*inc_trans* _ _ _ ((*alpha* » beta)   *alpha*)).
apply (@*cap_inc_compat_l* _ _ _ _ _ *H*).
rewrite *cap_comm rpc_l*.
apply (@*inc_trans* _ _ _ beta).
apply *cap_r*.
apply *H0*.
Qed.

Lemma 60 (rpc_inc_compat_l) *Let* $\alpha, \beta, \beta' : A \rightarrow B$. *Then,*

$$\beta \sqsubseteq \beta' \Rightarrow (\alpha \Rightarrow \beta) \sqsubseteq (\alpha \Rightarrow \beta').$$

Lemma *rpc_inc_compat_l* {*A B : eqType*} {*alpha* beta *beta' : Rel A B*}:
 beta   *beta'* → (*alpha* » beta)   (*alpha* » *beta'*).
Proof.
move ⇒ *H*.
apply (@*rpc_inc_compat* _ _ _ _ _ _ (@*inc_refl* _ _ *alpha*) *H*).
Qed.

Lemma 61 (rpc_inc_compat_r) *Let* $\alpha, \alpha', \beta : A \rightarrow B$. *Then,*

$$\alpha' \sqsubseteq \alpha \Rightarrow (\alpha \Rightarrow \beta) \sqsubseteq (\alpha' \Rightarrow \beta).$$

Lemma *rpc_inc_compat_r* {*A B : eqType*} {*alpha alpha'* beta : *Rel A B*}:
 alpha'   *alpha* → (*alpha* » beta)   (*alpha'* » beta).
Proof.
move ⇒ *H*.
apply (@*rpc_inc_compat* _ _ _ _ _ _ *H* (@*inc_refl* _ _ beta)).
Qed.

Lemma 62 (rpc_universal_alpha) *Let* $\alpha : A \rightarrow B$. *Then,*

$$\nabla_{AB} \Rightarrow \alpha = \alpha.$$

Lemma *rpc_universal_alpha* {*A B : eqType*} {*alpha : Rel A B*}:   *A B » alpha = alpha*.
Proof.
apply *inc_lower*.
move ⇒ *gamma*.
split; move ⇒ *H*.
apply *inc_rpc* in *H*.
rewrite *cap_universal* in *H*.
apply *H*.
apply *inc_rpc*.
rewrite *cap_universal*.
apply *H*.
Qed.

Lemma 63 (rpc_lemma1) *Let* $\alpha, \beta, \gamma : A \rightarrow B$. *Then,*

$$(\alpha \Rightarrow \beta) \sqsubseteq ((\alpha \sqcap \gamma) \Rightarrow (\beta \sqcap \gamma)).$$

Lemma *rpc_lemma1* {*A B* : *eqType*} {*alpha* beta *gamma* : *Rel A B*}:
 (*alpha* » beta)   ((*alpha*   *gamma*) » (beta   *gamma*)).
Proof.
apply *inc_rpc*.
rewrite -*cap_assoc* (@*cap_comm* _ _ _ *alpha*).
rewrite *rpc_l*.
apply *cap_inc_compat_r*.
apply *cap_r*.
Qed.

Lemma 64 (rpc_lemma2) *Let* $\alpha, \beta, \gamma : A \to B$. *Then,*

$$(\alpha \Rightarrow \beta) \sqcap (\alpha \Rightarrow \gamma) = (\alpha \Rightarrow (\beta \sqcap \gamma)).$$

Lemma *rpc_lemma2* {*A B* : *eqType*} {*alpha* beta *gamma* : *Rel A B*}:
 (*alpha* » beta)   (*alpha* » *gamma*) = *alpha* » (beta   *gamma*).
Proof.
apply *inc_lower*.
move ⇒ delta.
split; move ⇒ *H*.
rewrite *inc_rpc*.
apply *inc_cap* in *H*.
apply *inc_cap*.
rewrite -*inc_rpc* -*inc_rpc*.
apply *H*.
apply *inc_cap*.
rewrite *inc_rpc* *inc_rpc*.
apply *inc_cap*.
rewrite -*inc_rpc*.
apply *H*.
Qed.

Lemma 65 (rpc_lemma3) *Let* $\alpha, \beta, \gamma : A \to B$. *Then,*

$$(\alpha \Rightarrow \beta) \sqcap (\beta \Rightarrow \gamma) \sqsubseteq ((\alpha \sqcup \beta) \Rightarrow (\beta \sqcap \gamma)).$$

Lemma *rpc_lemma3* {*A B* : *eqType*} {*alpha* beta *gamma* : *Rel A B*}:
 ((*alpha* » beta)   (beta » *gamma*))   ((*alpha*   beta) » (beta   *gamma*)).
Proof.
apply *inc_rpc*.
rewrite *cap_cup_distr_l*.
rewrite *cap_comm* -*cap_assoc* *rpc_l*.
rewrite (@*cap_assoc* _ _ _ _ beta) (@*cap_comm* _ _ (beta » *gamma*)) -*cap_assoc* *rpc_r*.

```
rewrite cap_assoc rpc_l.
apply inc_cup.
split.
apply cap_r.
apply inc_refl.
Qed.
```

Lemma 66 (rpc_lemma4) *Let* $\alpha, \beta, \gamma : A \to B$. *Then,*

$$(\alpha \Rightarrow \beta) \sqcap (\beta \Rightarrow \gamma) \sqsubseteq (\alpha \Rightarrow \gamma).$$

```
Lemma rpc_lemma4 {A B : eqType} {alpha beta gamma : Rel A B}:
 ((alpha » beta)    (beta » gamma))    (alpha » gamma).
Proof.
apply (@inc_trans _ _ _ ((alpha    beta) » (beta    gamma))).
apply rpc_lemma3.
apply rpc_inc_compat.
apply cup_l.
apply cap_r.
Qed.
```

Lemma 67 (rpc_lemma5) *Let* $\alpha, \beta, \gamma : A \to B$. *Then,*

$$\alpha \Rightarrow (\beta \Rightarrow \gamma) = (\alpha \sqcap \beta) \Rightarrow \gamma.$$

```
Lemma rpc_lemma5 {A B : eqType} {alpha beta gamma : Rel A B}:
 alpha » (beta » gamma) = (alpha    beta) » gamma.
Proof.
apply inc_lower.
move ⇒ delta.
split; move ⇒ H.
apply inc_rpc.
rewrite -cap_assoc.
rewrite -inc_rpc -inc_rpc.
apply H.
rewrite inc_rpc inc_rpc.
rewrite cap_assoc.
apply inc_rpc.
apply H.
Qed.
```

Lemma 68 (rpc_lemma6) *Let* $\alpha, \beta, \gamma : A \rightarrow B$. *Then,*

$$\alpha \Rightarrow (\beta \Rightarrow \gamma) \sqsubseteq (\alpha \Rightarrow \beta) \Rightarrow (\alpha \Rightarrow \gamma).$$

Lemma *rpc_lemma6* {*A B* : *eqType*} {*alpha* beta *gamma* : *Rel A B*}:
 (*alpha* » (beta » *gamma*))   ((*alpha* » beta) » (*alpha* » *gamma*)).
Proof.
rewrite *inc_rpc inc_rpc*.
rewrite *cap_assoc* (@*cap_comm* _ _ _ *alpha*).
rewrite *rpc_l*.
rewrite -*cap_assoc* (@*cap_comm* _ _ _ *alpha*).
rewrite *rpc_l*.
rewrite *cap_assoc* (@*cap_comm* _ _ _ beta).
rewrite *rpc_l*.
rewrite -*cap_assoc*.
apply *cap_r*.
Qed.

Lemma 69 (rpc_lemma7) *Let* $\alpha, \beta, \gamma, \delta : A \rightarrow B$ *and* $\beta \sqsubseteq \alpha \sqsubseteq \gamma$. *Then,*

$$(\alpha \sqcap \delta = \beta) \wedge (\alpha \sqcup \delta = \gamma) \Leftrightarrow (\gamma \sqsubseteq \alpha \sqcup (\alpha \Rightarrow \beta)) \wedge (\delta = \gamma \sqcap (\alpha \Rightarrow \beta)).$$

Lemma *rpc_lemma7* {*A B* : *eqType*} {*alpha* beta *gamma* delta : *Rel A B*}:
beta   *alpha* → *alpha*   *gamma* → (*alpha*   delta = beta ∧ *alpha*   delta = *gamma*
 ↔ *gamma*   (*alpha*   (*alpha* » beta)) ∧ delta = *gamma*   (*alpha* » beta)).
Proof.
move ⇒ *H H0*.
split; elim; move ⇒ *H1 H2*; split.
rewrite -*H2*.
apply *cup_inc_compat_l*.
apply *inc_rpc*.
rewrite *cap_comm H1*.
apply *inc_refl*.
rewrite -*H2*.
rewrite *cap_cup_distr_r rpc_l*.
assert (delta   (*alpha* » beta)).
apply *inc_rpc*.
rewrite *cap_comm H1*.
apply *inc_refl*.
apply *inc_def1* in *H3*.
rewrite -*H3* -*H1*.
rewrite -*cap_assoc cap_idem*.

by [rewrite *cap_comm cup_comm cup_cap_abs*].
rewrite *H2*.
rewrite (@*cap_comm* _ _ *gamma*) -*cap_assoc rpc_l*.
apply *inc_antisym*.
apply (@*inc_trans* _ _ _ (beta   gamma)).
apply *cap_inc_compat_r*.
apply *cap_r*.
apply *cap_l*.
move : (@*inc_trans* _ _ _ _ _ *H H0*) ⇒ *H3*.
apply *inc_def1* in *H*.
apply *inc_def1* in *H3*.
rewrite *cap_comm* in *H*.
rewrite -*H* -*H3*.
apply *inc_refl*.
rewrite *H2*.
rewrite *cup_cap_distr_l*.
apply *inc_def2* in *H0*.
rewrite -*H0*.
apply *inc_def1* in *H1*.
by [rewrite -*H1*].
Qed.

3.3

Lemma 70 (complement_universal)

$$\nabla_{AB}{}^- = \phi_{AB}.$$

Lemma *complement_universal* {*A B : eqType*}:   *A B* ˆ =   *A B*.
Proof.
apply *rpc_universal_alpha*.
Qed.

Lemma 71 (complement_alpha_universal) *Let* $\alpha : A \rightarrowtail B$. *Then,*

$$\alpha^- = \nabla_{AB} \Leftrightarrow \alpha = \phi_{AB}.$$

Lemma *complement_alpha_universal* {*A B : eqType*} {*alpha : Rel A B*}:
 alpha ˆ =   *A B* ↔ *alpha* =   *A B*.
Proof.
split; move ⇒ *H*.
apply *inc_antisym*.

```
rewrite -(@cap_universal _ _ alpha) cap_comm.
apply inc_rpc.
rewrite -H.
apply inc_refl.
apply inc_empty_alpha.
apply inc_antisym.
apply inc_alpha_universal.
apply inc_rpc.
rewrite cap_comm cap_universal.
rewrite H.
apply inc_refl.
Qed.
```

Lemma 72 (complement_empty)

$$\phi_{AB}{}^{-} = \nabla_{AB}.$$

```
Lemma complement_empty {A B : eqType}:    A B ^ =    A B.
Proof.
by [apply complement_alpha_universal].
Qed.
```

Lemma 73 (complement_invol_inc) *Let* $\alpha : A \rightharpoonup B$. *Then,*

$$\alpha \sqsubseteq (\alpha^{-})^{-}.$$

```
Lemma complement_invol_inc {A B : eqType} {alpha : Rel A B}: alpha    (alpha ^) ^.
Proof.
apply inc_rpc.
rewrite cap_comm.
apply inc_rpc.
apply inc_refl.
Qed.
```

Lemma 74 (cap_complement_empty) *Let* $\alpha : A \rightharpoonup B$. *Then,*

$$\alpha \sqcap \alpha^{-} = \phi_{AB}.$$

```
Lemma cap_complement_empty {A B : eqType} {alpha : Rel A B}:
 alpha    alpha ^ =    A B.
Proof.
apply inc_antisym.
rewrite cap_comm.
```

apply *inc_rpc*.
apply *inc_refl*.
apply *inc_empty_alpha*.
Qed.

Lemma 75 (complement_invol) *Let* $\alpha : A \rightarrow B$. *Then,*

$$(\alpha^-)^- = \alpha.$$

Lemma *complement_invol* $\{A\ B : eqType\}$ $\{alpha : Rel\ A\ B\}$: $(alpha\ \hat{}\ )\ \hat{}\ = alpha$.
Proof.
rewrite -(@*cap_universal* _ _ (($alpha\ \hat{}\ )\ \hat{}\ $)).
rewrite -(@*complement_classic* _ _ *alpha*).
rewrite *cap_cup_distr_l*.
rewrite (@*cap_comm* _ _ _ ($alpha\ \hat{}\ $)) *cap_complement_empty*.
rewrite *cup_empty cap_comm*.
apply *Logic.eq_sym*.
apply *inc_def1*.
apply *complement_invol_inc*.
Qed.

Lemma 76 (complement_move) *Let* $\alpha, \beta : A \rightarrow B$. *Then,*

$$\alpha = \beta^- \Leftrightarrow \alpha^- = \beta.$$

Lemma *complement_move* $\{A\ B : eqType\}$ $\{alpha$ beta $: Rel\ A\ B\}$:
 $alpha =$ beta $\hat{}\ \leftrightarrow alpha\ \hat{}\ =$ beta.
Proof.
split; move $\Rightarrow H$.
by [rewrite H *complement_invol*].
by [rewrite -H *complement_invol*].
Qed.

Lemma 77 (contraposition) *Let* $\alpha, \beta : A \rightarrow B$. *Then,*

$$(\alpha \Rightarrow \beta) = (\beta^- \Rightarrow \alpha^-).$$

Lemma *contraposition* $\{A\ B : eqType\}$ $\{alpha$ beta $: Rel\ A\ B\}$:
 $alpha \gg$ beta $=$ beta $\hat{}\ \gg alpha\ \hat{}\ $.
Proof.
apply *inc_antisym*.
apply *inc_rpc*.
apply *rpc_lemma4*.

```
replace (alpha » beta) with ((alpha ^) ^ » (beta ^) ^).
apply inc_rpc.
apply rpc_lemma4.
by [rewrite complement_invol complement_invol].
Qed.
```

Lemma 78 (de_morgan1) *Let $\alpha, \beta : A \rightharpoonup B$. Then,*

$$(\alpha \sqcup \beta)^- = \alpha^- \sqcap \beta^-.$$

```
Lemma de_morgan1 {A B : eqType} {alpha beta : Rel A B}:
 (alpha    beta) ^ = alpha ^    beta ^.
Proof.
apply inc_lower.
move ⇒ gamma.
split; move ⇒ H.
apply inc_cap.
rewrite inc_rpc inc_rpc.
apply inc_cup.
rewrite -cap_cup_distr_l.
apply inc_rpc.
apply H.
apply inc_rpc.
rewrite cap_cup_distr_l.
apply inc_cup.
rewrite -inc_rpc -inc_rpc.
apply inc_cap.
apply H.
Qed.
```

Lemma 79 (de_morgan2) *Let $\alpha, \beta : A \rightharpoonup B$. Then,*

$$(\alpha \sqcap \beta)^- = \alpha^- \sqcup \beta^-.$$

```
Lemma de_morgan2 {A B : eqType} {alpha beta : Rel A B}:
 (alpha    beta) ^ = alpha ^    beta ^.
Proof.
by [rewrite -complement_move de_morgan1 complement_invol complement_invol].
Qed.
```

> **Lemma 80 (cup_to_rpc)** *Let $\alpha, \beta : A \to B$. Then,*
>
> $$\alpha^- \sqcup \beta = (\alpha \Rightarrow \beta).$$

Lemma *cup_to_rpc* $\{A\ B : eqType\}$ $\{alpha$ `beta` $: Rel\ A\ B\}$:
 alpha ^   `beta` $=$ *alpha* » `beta`.
Proof.
apply *inc_antisym*.
apply *inc_rpc*.
rewrite *cap_cup_distr_r cap_comm*.
rewrite *cap_complement_empty cup_comm cup_empty*.
apply *cap_l*.
rewrite -(@*cap_universal* _ _ (*alpha* » `beta`)) *cap_comm*.
rewrite -(@*complement_classic* _ _ *alpha*).
rewrite *cap_cup_distr_r cup_comm*.
apply *cup_inc_compat*.
apply *cap_l*.
rewrite *rpc_l*.
apply *cap_r*.
Qed.

> **Lemma 81 (beta_contradiction)** *Let $\alpha, \beta : A \to B$. Then,*
>
> $$(\alpha \Rightarrow \beta) \sqcap (\alpha \Rightarrow \beta^-) = \alpha^-.$$

Lemma *beta_contradiction* $\{A\ B : eqType\}$ $\{alpha$ `beta` $: Rel\ A\ B\}$:
 (*alpha* » `beta`)   (*alpha* » `beta` ^) $=$ *alpha*^.
Proof.
rewrite -*cup_to_rpc* -*cup_to_rpc*.
rewrite -*cup_cap_distr_l*.
by [rewrite *cap_complement_empty cup_empty*].
Qed.

3.4   Bool

> **Lemma 82 (bool_lemma1)** *Let $\alpha, \beta : A \to B$. Then,*
>
> $$\alpha \sqsubseteq \beta \Leftrightarrow \nabla_{AB} = \alpha^- \sqcup \beta.$$

Lemma *bool_lemma1* $\{A\ B : eqType\}$ $\{alpha$ `beta` $: Rel\ A\ B\}$:
 alpha   `beta` $\leftrightarrow$ $A\ B =$ *alpha* ^   `beta`.

59

```
Proof.
split; move ⇒ H.
apply inc_antisym.
rewrite -(@complement_classic _ _ alpha) cup_comm.
apply cup_inc_compat_l.
apply H.
apply inc_alpha_universal.
apply inc_def3.
rewrite H.
apply (Logic.eq_sym cup_to_rpc).
Qed.
```

Lemma 83 (bool_lemma2) *Let* $\alpha, \beta : A \rightharpoonup B$. *Then,*

$$\alpha \sqsubseteq \beta \Leftrightarrow \alpha \sqcap \beta^{-} = \phi_{AB}.$$

```
Lemma bool_lemma2 {A B : eqType} {alpha beta : Rel A B}:
 alpha    beta ↔ alpha    beta ˆ =    A B.
Proof.
split; move ⇒ H.
rewrite -(@cap_universal _ _ (alpha    beta ˆ)).
apply bool_lemma1 in H.
rewrite H.
rewrite cap_cup_distr_l.
rewrite (@cap_comm _ _ alpha) cap_assoc cap_complement_empty cap_empty.
rewrite cap_comm -cap_assoc cap_complement_empty cap_comm cap_empty.
by [rewrite cup_empty].
rewrite -(@cap_universal _ _ alpha).
rewrite -(@complement_classic _ _ beta).
rewrite cap_cup_distr_l.
rewrite H cup_empty.
apply cap_r.
Qed.
```

Lemma 84 (bool_lemma3) *Let* $\alpha, \beta, \gamma : A \rightharpoonup B$. *Then,*

$$\alpha \sqsubseteq \beta \sqcup \gamma \Leftrightarrow \alpha \sqcap \beta^{-} \sqsubseteq \gamma.$$

```
Lemma bool_lemma3 {A B : eqType} {alpha beta gamma : Rel A B}:
 alpha    (beta    gamma) ↔ (alpha    beta ˆ)    gamma.
Proof.
split; move ⇒ H.
apply (@inc_trans _ _ _ ((beta    gamma)    beta ˆ)).
```

```
apply cap_inc_compat_r.
apply H.
rewrite cap_cup_distr_r.
rewrite cap_complement_empty cup_comm cup_empty.
apply cap_l.
apply (@inc_trans _ _ _ (beta     (alpha     beta ^))).
rewrite cup_cap_distr_l.
rewrite complement_classic cap_universal.
apply cup_r.
apply cup_inc_compat_l.
apply H.
Qed.
```

Lemma 85 (bool_lemma4) *Let* $\alpha, \beta, \gamma : A \rightharpoonup B$. *Then,*

$$\alpha \sqsubseteq \beta \sqcup \gamma \Leftrightarrow \beta^- \sqsubseteq \alpha^- \sqcup \gamma.$$

```
Lemma bool_lemma4 {A B : eqType} {alpha beta gamma : Rel A B}:
 alpha     (beta     gamma) ↔ beta ^     (alpha ^     gamma).
Proof.
rewrite bool_lemma3.
rewrite cap_comm.
apply iff_sym.
replace (beta ^     alpha) with (beta ^     (alpha ^) ^).
apply bool_lemma3.
by [rewrite complement_invol].
Qed.
```

Lemma 86 (bool_lemma5) *Let* $\alpha, \beta, \gamma : A \rightharpoonup B$. *Then,*

$$\alpha \sqsubseteq \beta \sqcup \gamma \Leftrightarrow \nabla_{AB} = \alpha^- \sqcup \beta \sqcup \gamma.$$

```
Lemma bool_lemma5 {A B : eqType} {alpha beta gamma : Rel A B}:
 alpha     (beta     gamma) ↔     A B = (alpha ^     beta)     gamma.
Proof.
rewrite bool_lemma1.
by [rewrite cup_assoc].
Qed.

End main.
```

Chapter 4

Library Relation_Properties

4.1

Lemma 87 (RelAB_unique)

$$\phi_{AB} = \nabla_{AB} \Leftrightarrow \forall \alpha, \beta : A \rightharpoonup B, \alpha = \beta.$$

Lemma *RelAB_unique* {*A B* : *eqType*}:
    A B =   *A B* ↔ (∀ *alpha* beta : *Rel A B*, *alpha* = beta).
Proof.
split; move ⇒ *H*.
move ⇒ *alpha* beta.
replace beta with (*A B*).
apply *inc_antisym*.
rewrite *H*.
apply *inc_alpha_universal*.
apply *inc_empty_alpha*.
apply *inc_antisym*.
apply *inc_empty_alpha*.
rewrite *H*.

```
apply inc_alpha_universal.
apply H.
Qed.
```

Lemma 88 (either_empty)

$$\phi_{AB} = \nabla_{AB} \Leftrightarrow A = \emptyset \vee B = \emptyset.$$

```
Lemma either_empty {A B : eqType}:    A B =    A B ↔ (A → False) ∨ (B → False).
Proof.
rewrite RelAB_unique.
split; move ⇒ H.
case (classic (∃ _ : A, True)).
elim ⇒ a H0.
right.
move ⇒ b.
remember (fun (_ : A) (_ : B) ⇒ True) as T.
remember (fun (_ : A) (_ : B) ⇒ False) as F.
move : (H T F) ⇒ H1.
assert (T a b = F a b).
by [rewrite H1].
rewrite HeqT HeqF in H2.
rewrite -H2.
apply I.
move ⇒ H0.
left.
move ⇒ a.
apply H0.
∃ a.
apply I.
move ⇒ alpha beta.
assert (A → B → False).
move ⇒ a b.
case H; move ⇒ H0.
apply (H0 a).
apply (H0 b).
apply functional_extensionality.
move ⇒ a.
apply functional_extensionality.
move ⇒ b.
apply False_ind.
apply (H0 a b).
Qed.
```

Lemma 89 (unit_empty_not_universal)

$$\phi_{II} \neq \nabla_{II}.$$

Lemma *unit_empty_not_universal* :   *i i $\neq$*   *i i.*
Proof.
move $\Rightarrow$ *H.*
apply *either_empty* in *H.*
case *H*; move $\Rightarrow$ *H0.*
apply (*H0 tt*).
apply (*H0 tt*).
Qed.

Lemma 90 (unit_empty_or_universal) *Let $\alpha : I \rightarrow I$. Then,*

$$\alpha = \phi_{II} \vee \alpha = \nabla_{II}.$$

Lemma *unit_empty_or_universal* {*alpha : Rel i i*}: *alpha =*   *i i* $\vee$ *alpha =*   *i i.*
Proof.
assert ($\forall$ beta : *Rel i i*, beta = (fun (_ _ : *i*) $\Rightarrow$ *True*) $\vee$ beta = (fun (_ _ : *i*) $\Rightarrow$ *False*)).
move $\Rightarrow$ beta.
case (*classic* (beta *tt tt*)); move $\Rightarrow$ *H.*
left.
apply *functional_extensionality.*
induction *x.*
apply *functional_extensionality.*
induction *x.*
apply *prop_extensionality_ok.*
split; move $\Rightarrow$ *H0.*
apply *I.*
apply *H.*
right.
apply *functional_extensionality.*
induction *x.*
apply *functional_extensionality.*
induction *x.*
apply *prop_extensionality_ok.*
split.
apply *H.*
apply *False_ind.*
assert ((fun _ _ : *i* $\Rightarrow$ *True*) $\neq$ (fun _ _ : *i* $\Rightarrow$ *False*)).
move $\Rightarrow$ *H0.*
remember (fun _ _ : *i* $\Rightarrow$ *True*) as *T.*

remember (`fun` _ _ : $i \Rightarrow$ *False*) `as` *F*.
`assert` (*T tt tt = F tt tt*).
`by` [`rewrite` *H0*].
`rewrite` *HeqT HeqF* `in` *H1*.
`rewrite` -*H1*.
`apply` *I*.
`case` (*H* (*i i*)); `move` $\Rightarrow$ *H1*.
`case` (*H* (*i i*)); `move` $\Rightarrow$ *H2*.
`apply` *False_ind*.
`apply` *unit_empty_not_universal*.
`by` [`rewrite` *H1 H2*].
`case` (*H alpha*); `move` $\Rightarrow$ *H3*.
`left`.
`by` [`rewrite` *H3 H1*].
`right`.
`by` [`rewrite` *H3 H2*].
`case` (*H* (*i i*)); `move` $\Rightarrow$ *H2*.
`case` (*H alpha*); `move` $\Rightarrow$ *H3*.
`right`.
`by` [`rewrite` *H3 H2*].
`left`.
`by` [`rewrite` *H3 H1*].
`apply` *False_ind*.
`apply` *unit_empty_not_universal*.
`by` [`rewrite` *H1 H2*].
`Qed`.

Lemma 91 (unit_identity_is_universal)

$$id_I = \nabla_{II}.$$

`Lemma` *unit_identity_is_universal* : *Id i =   i i*.
`Proof`.
`case` (@*unit_empty_or_universal* (*Id i*)); `move` $\Rightarrow$ *H*.
`apply` *False_ind*.
`assert` (*Id i   (i i #   i i*)).
`rewrite` *H*.
`apply` *inc_empty_alpha*.
`apply` *inc_residual* `in` *H0*.
`rewrite` *inv_invol comp_id_r* `in` *H0*.
`apply` *unit_empty_not_universal*.
`apply` *inc_antisym*.
`apply` *inc_empty_alpha*.

```
apply H0.
apply H.
Qed.
```

Lemma 92 (unit_identity_not_empty)

$$id_I \neq \phi_{II}.$$

```
Lemma unit_identity_not_empty : Id i ≠    i i.
Proof.
move ⇒ H.
apply unit_empty_not_universal.
rewrite -H.
apply unit_identity_is_universal.
Qed.
```

Lemma 93 (cupP_False) *Let* $f : (C \rightharpoonup D) \to (A \rightharpoonup B)$ *and* $P(\alpha) :=$ *"False". Then,*

$$\sqcup_{P(\alpha)} f(\alpha) = \phi_{AB}.$$

```
Lemma cupP_False {A B C D : eqType} {f : Rel C D → Rel A B}:
   _{fun _ : Rel C D ⇒ False} f =    A B.
Proof.
apply inc_antisym.
apply inc_cupP.
move ⇒ beta.
apply False_ind.
apply inc_empty_alpha.
Qed.
```

Lemma 94 (capP_False) *Let* $f : (C \rightharpoonup D) \to (A \rightharpoonup B)$ *and* $P(\alpha) :=$ *"False". Then,*

$$\sqcap_{P(\alpha)} f(\alpha) = \nabla_{AB}.$$

```
Lemma capP_False {A B C D : eqType} {f : Rel C D → Rel A B}:
   _{fun _ : Rel C D ⇒ False} f =    A B.
Proof.
apply inc_antisym.
apply inc_alpha_universal.
apply inc_capP.
move ⇒ beta.
apply False_ind.
Qed.
```

Lemma 95 (cupP_eq) *Let $f, g : (C \rightarrow D) \rightarrow (A \rightarrow B)$ and P : predicate. Then,*

$$(\forall \alpha : C \rightarrow D, P(\alpha) \Rightarrow f(\alpha) = g(\alpha)) \Rightarrow \sqcup_{P(\alpha)} f(\alpha) = \sqcup_{P(\alpha)} g(\alpha).$$

Lemma *cupP_eq* {*A B C D : eqType*}
 {*f g : Rel C D → Rel A B*} {*P : Rel C D →* Prop}:
 ($\forall$ *alpha : Rel C D, P alpha → f alpha = g alpha*) → ⎵{*P*} *f* = ⎵{*P*} *g*.
Proof.
move ⇒ *H*.
apply *inc_antisym*.
apply *inc_cupP*.
move ⇒ beta *H0*.
rewrite (*H _ H0*).
move : beta *H0*.
apply *inc_cupP*.
apply *inc_refl*.
apply *inc_cupP*.
move ⇒ beta *H0*.
rewrite -(*H _ H0*).
move : beta *H0*.
apply *inc_cupP*.
apply *inc_refl*.
Qed.

Lemma 96 (capP_eq) *Let $f, g : (C \rightarrow D) \rightarrow (A \rightarrow B)$ and P : predicate. Then,*

$$(\forall \alpha : C \rightarrow D, P(\alpha) \Rightarrow f(\alpha) = g(\alpha)) \Rightarrow \sqcap_{P(\alpha)} f(\alpha) = \sqcap_{P(\alpha)} g(\alpha).$$

Lemma *capP_eq* {*A B C D : eqType*}
 {*f g : Rel C D → Rel A B*} {*P : Rel C D →* Prop}:
 ($\forall$ *alpha : Rel C D, P alpha → f alpha = g alpha*) → ⎵{*P*} *f* = ⎵{*P*} *g*.
Proof.
move ⇒ *H*.
apply *inc_antisym*.
apply *inc_capP*.
move ⇒ beta *H0*.
rewrite -(*H _ H0*).
move : beta *H0*.
apply *inc_capP*.
apply *inc_refl*.
apply *inc_capP*.
move ⇒ beta *H0*.

```
rewrite (H _ H0).
move : beta H0.
apply inc_capP.
apply inc_refl.
Qed.
```

Lemma 97 (cap_cupP_distr_l) *Let $\alpha : A \rightarrow B$, $f : (C \rightarrow D) \rightarrow (A \rightarrow B)$ and $P :$ predicate. Then,*

$$\alpha \sqcap (\sqcup_{P(\beta)} f(\beta)) = \sqcup_{P(\beta)}(\alpha \sqcap f(\beta)).$$

```
Lemma cap_cupP_distr_l {A B C D : eqType}
 {alpha : Rel A B} {f : Rel C D → Rel A B} {P : Rel C D → Prop}:
 alpha    ( _{P} f) =    _{P} (fun beta : Rel C D ⇒ alpha    f beta).
Proof.
apply inc_upper.
move ⇒ gamma.
split; move ⇒ H.
apply inc_cupP.
move ⇒ beta H0.
apply (@inc_trans _ _ _ (alpha       _{P} f)).
apply cap_inc_compat_l.
move : H0.
apply inc_cupP.
apply inc_refl.
apply H.
assert (∀ beta : Rel C D, P beta → (alpha    f beta)    gamma).
apply inc_cupP.
apply H.
assert (∀ beta : Rel C D, P beta → f beta    (alpha » gamma)).
move ⇒ beta H1.
rewrite inc_rpc cap_comm.
apply (H0 _ H1).
rewrite cap_comm -inc_rpc.
apply inc_cupP.
apply H1.
Qed.
```

Lemma 98 (cap_cupP_distr_r) *Let $\beta : A \rightarrow B$, $f : (C \rightarrow D) \rightarrow (A \rightarrow B)$ and $P :$ predicate. Then,*

$$(\sqcup_{P(\alpha)} f(\alpha)) \sqcap \beta = \sqcup_{P(\alpha)}(f(\alpha) \sqcap \beta).$$

```
Lemma cap_cupP_distr_r {A B C D : eqType}
```

{beta : *Rel A B*} {*f* : *Rel C D* → *Rel A B*} {*P* : *Rel C D* → Prop}:
(⊓_{*P*} *f*) ⊓ beta = ⊓_{*P*} (fun *alpha* : *Rel C D* ⇒ *f alpha* ⊓ beta).
Proof.
rewrite *cap_comm*.
replace (fun *alpha* : *Rel C D* ⇒ *f alpha* ⊓ beta) with (fun *alpha* : *Rel C D* ⇒ beta ⊓ *f alpha*).
apply *cap_cupP_distr_l*.
apply *functional_extensionality*.
move ⇒ *x*.
by [rewrite *cap_comm*].
Qed.

Lemma 99 (cup_capP_distr_l) *Let* $\alpha : A \rightharpoonup B$, $f : (C \rightharpoonup D) \to (A \rightharpoonup B)$ *and* P : *predicate. Then,*
$$\alpha \sqcup (\sqcap_{P(\beta)} f(\beta)) = \sqcap_{P(\beta)} (\alpha \sqcup f(\beta)).$$

Lemma *cup_capP_distr_l* {*A B C D* : *eqType*}
 {*alpha* : *Rel A B*} {*f* : *Rel C D* → *Rel A B*} {*P* : *Rel C D* → Prop}:
 alpha ⊔ (⊓_{*P*} *f*) = ⊓_{*P*} (fun beta : *Rel C D* ⇒ *alpha* ⊔ *f* beta).
Proof.
apply *inc_lower*.
move ⇒ *gamma*.
split; move ⇒ *H*.
apply *inc_capP*.
move ⇒ beta *H0*.
apply (@*inc_trans* _ _ _ (*alpha* ⊔ ⊓_{*P*} *f*)).
apply *H*.
apply *cup_inc_compat_l*.
move : *H0*.
apply *inc_capP*.
apply *inc_refl*.
rewrite *bool_lemma3*.
assert (∀ beta : *Rel C D*, *P* beta → *gamma* ⊆ (*alpha* ⊔ *f* beta)).
apply *inc_capP*.
apply *H*.
apply *inc_capP*.
move ⇒ beta *H1*.
rewrite -*bool_lemma3*.
apply (*H0* _ *H1*).
Qed.

Lemma 100 (cup_capP_distr_r) *Let* $\beta : A \rightarrow B$, $f : (C \rightarrow D) \rightarrow (A \rightarrow B)$ *and* $P :$ *predicate. Then,*

$$(\sqcap_{P(\alpha)} f(\alpha)) \sqcup \beta = \sqcap_{P(\alpha)}(f(\alpha) \sqcup \beta).$$

Lemma *cup_capP_distr_r* {*A B C D : eqType*}
 {beta : *Rel A B*} {*f : Rel C D* → *Rel A B*} {*P : Rel C D* → Prop}:
 (_{*P*} *f*)   beta =   _{*P*} (fun *alpha : Rel C D* ⇒ *f alpha*   beta).
Proof.
rewrite *cup_comm*.
replace (fun *alpha : Rel C D* ⇒ *f alpha*   beta) with (fun *alpha : Rel C D* ⇒ beta
f alpha).
apply *cup_capP_distr_l*.
apply *functional_extensionality*.
move ⇒ *x*.
by [rewrite *cup_comm*].
Qed.

Lemma 101 (de_morgan3) *Let* $f : (C \rightarrow D) \rightarrow (A \rightarrow B)$ *and* $P :$ *predicate. Then,*

$$(\sqcup_{P(\alpha)} f(\alpha))^{-} = (\sqcap_{P(\alpha)} f(\alpha)^{-}).$$

Lemma *de_morgan3*
 {*A B C D : eqType*} {*f : Rel C D* → *Rel A B*} {*P : Rel C D* → Prop}:
 (_{*P*} *f*) ^ =   _{*P*} (fun *alpha : Rel C D* ⇒ *f alpha* ^).
Proof.
apply *inc_lower*.
move ⇒ *gamma*.
rewrite *inc_capP*.
split; move ⇒ *H*.
move ⇒ beta *H0*.
rewrite *bool_lemma1 -de_morgan2 complement_move complement_universal*.
apply *bool_lemma2* in *H*.
apply *inc_antisym*.
apply *inc_empty_alpha*.
rewrite -*H complement_invol*.
apply *cap_inc_compat_l*.
move : *H0*.
apply *inc_cupP*.
apply *inc_refl*.
rewrite *bool_lemma2 complement_invol*.
rewrite *cap_cupP_distr_l*.
apply *inc_antisym*.

70

```
apply inc_cupP.
move ⇒ beta H0.
rewrite -inc_rpc.
apply (H _ H0).
apply inc_empty_alpha.
Qed.
```

Lemma 102 (de_morgan4) *Let* $f : (C \rightarrow D) \rightarrow (A \rightarrow B)$ *and* P : *predicate. Then,*

$$(\sqcap_{P(\alpha)} f(\alpha))^- = (\sqcup_{P(\alpha)} f(\alpha)^-).$$

```
Lemma de_morgan4
  {A B C D : eqType} {f : Rel C D → Rel A B} {P : Rel C D → Prop}:
  ( _{P} f) ^ =  _{P} (fun alpha : Rel C D ⇒ f alpha ^).
Proof.
rewrite -complement_move de_morgan3.
replace (fun alpha : Rel C D ⇒ (f alpha ^) ^) with f.
by [].
apply functional_extensionality.
move ⇒ x.
by [rewrite complement_invol].
Qed.
```

Lemma 103 (cup_to_cupP) *Let* $f : (C \rightarrow D) \rightarrow (A \rightarrow B)$ *and* P : *predicate. Then,*

$$f(\alpha) \sqcup f(\beta) = \sqcup_{\gamma = \alpha \vee \gamma = \beta} f(\gamma).$$

```
Lemma cup_to_cupP
  {A B C D : eqType} {alpha beta : Rel C D} {f : Rel C D → Rel A B}:
  (f alpha    f beta) =  _{fun gamma : Rel C D ⇒ gamma = alpha ∨ gamma = beta}
f.
Proof.
apply inc_upper.
move ⇒ delta.
split; move ⇒ H.
apply inc_cupP.
apply inc_cup in H.
move ⇒ gamma H0.
case H0 ⇒ H1.
rewrite H1.
apply H.
rewrite H1.
apply H.
```

apply *inc_cup*.
assert ($\forall$ *gamma* : *Rel C D*, *gamma* = *alpha* $\lor$ *gamma* = beta $\to$ *f gamma*   delta).
apply *inc_cupP*.
apply *H*.
split.
apply (*H0 alpha*).
by [left].
apply (*H0* beta).
by [right].
Qed.

Lemma 104 (cap_to_capP) *Let* $f : (C \to D) \to (A \to B)$ *and* P : *predicate. Then,*

$$f(\alpha) \sqcap f(\beta) = \sqcap_{\gamma = \alpha \lor \gamma = \beta} f(\gamma).$$

Lemma *cap_to_capP*
 {*A B C D* : *eqType*} {*alpha* beta : *Rel C D*} {*f* : *Rel C D* $\to$ *Rel A B*}:
 (*f alpha*   *f* beta) =   _{fun *gamma* : *Rel C D* $\Rightarrow$ *gamma* = *alpha* $\lor$ *gamma* = beta}
f.
Proof.
apply *inc_lower*.
move $\Rightarrow$ delta.
split; move $\Rightarrow$ *H*.
apply *inc_capP*.
apply *inc_cap* in *H*.
move $\Rightarrow$ *gamma H0*.
case *H0* $\Rightarrow$ *H1*.
rewrite *H1*.
apply *H*.
rewrite *H1*.
apply *H*.
apply *inc_cap*.
assert ($\forall$ *gamma* : *Rel C D*, *gamma* = *alpha* $\lor$ *gamma* = beta $\to$ delta   *f gamma*).
apply *inc_capP*.
apply *H*.
split.
apply (*H0 alpha*).
by [left].
apply (*H0* beta).
by [right].
Qed.

4.2 comp_inc_compat

Lemma 105 (comp_inc_compat_ab_ab') *Let $\alpha : A \rightharpoonup B$ and $\beta, \beta' : B \rightharpoonup C$. Then,*

$$\beta \sqsubseteq \beta' \Rightarrow \alpha \cdot \beta \sqsubseteq \alpha \cdot \beta'.$$

Lemma *comp_inc_compat_ab_ab'*
 {*A B C : eqType*} {*alpha : Rel A B*} {beta *beta' : Rel B C*}:
 beta ⊑ *beta'* → (*alpha* ⋅ beta) ⊑ (*alpha* ⋅ *beta'*).
Proof.
move ⇒ *H*.
replace (*alpha* ⋅ beta) with ((*alpha* #) # ⋅ beta).
apply *inc_residual*.
apply (@*inc_trans* _ _ _ *beta'*).
apply *H*.
apply *inc_residual*.
rewrite *inv_invol*.
apply *inc_refl*.
by [rewrite *inv_invol*].
Qed.

Lemma 106 (comp_inc_compat_ab_a'b) *Let $\alpha, \alpha' : A \rightharpoonup B$ and $\beta : B \rightharpoonup C$. Then,*

$$\alpha \sqsubseteq \alpha' \Rightarrow \alpha \cdot \beta \sqsubseteq \alpha' \cdot \beta.$$

Lemma *comp_inc_compat_ab_a'b*
 {*A B C : eqType*} {*alpha alpha' : Rel A B*} {beta *: Rel B C*}:
 alpha ⊑ *alpha'* → (*alpha* ⋅ beta) ⊑ (*alpha'* ⋅ beta).
Proof.
move ⇒ *H*.
rewrite -(@*inv_invol* _ _ (*alpha* ⋅ beta)).
rewrite -(@*inv_invol* _ _ (*alpha'* ⋅ beta)).
apply *inc_inv*.
rewrite *comp_inv comp_inv*.
apply *comp_inc_compat_ab_ab'*.
apply *inc_inv*.
apply *H*.
Qed.

Lemma 107 (comp_inc_compat) *Let $\alpha, \alpha' : A \rightharpoonup B$ and $\beta, \beta' : B \rightharpoonup C$. Then,*

$$\alpha \sqsubseteq \alpha' \wedge \beta \sqsubseteq \beta' \Rightarrow \alpha \cdot \beta \sqsubseteq \alpha' \cdot \beta'.$$

Lemma *comp_inc_compat*
 {*A B C : eqType*} {*alpha alpha' : Rel A B*} {beta *beta' : Rel B C*}:
 alpha   alpha' → beta   *beta'* → (*alpha*   beta)   (*alpha'*   *beta'*).
Proof.
move ⇒ *H H0*.
apply (@*inc_trans* _ _ _ (*alpha'*   beta)).
apply (@*comp_inc_compat_ab_a'b* _ _ _ _ _ _ *H*).
apply (@*comp_inc_compat_ab_ab'* _ _ _ _ _ _ *H0*).
Qed.

Lemma 108 (comp_inc_compat_ab_a) *Let $\alpha : A \rightharpoonup B$ and $\beta : B \rightharpoonup B$. Then,*

$$\beta \sqsubseteq id_B \Rightarrow \alpha \cdot \beta \sqsubseteq \beta.$$

Lemma *comp_inc_compat_ab_a* {*A B : eqType*} {*alpha : Rel A B*} {beta *: Rel B B*}:
 beta   *Id B* → (*alpha*   beta)   *alpha*.
Proof.
move ⇒ *H*.
move : (@*comp_inc_compat_ab_ab'* _ _ _ *alpha* _ _ *H*) ⇒ *H0*.
rewrite *comp_id_r* in *H0*.
apply *H0*.
Qed.

Lemma 109 (comp_inc_compat_a_ab) *Let $\alpha : A \rightharpoonup B$ and $\beta : B \rightharpoonup B$. Then,*

$$id_B \sqsubseteq \beta \Rightarrow \beta \sqsubseteq \alpha \cdot \beta.$$

Lemma *comp_inc_compat_a_ab* {*A B : eqType*} {*alpha : Rel A B*} {beta *: Rel B B*}:
 Id B   beta → *alpha*   (*alpha*   beta).
Proof.
move ⇒ *H*.
move : (@*comp_inc_compat_ab_ab'* _ _ _ *alpha* _ _ *H*) ⇒ *H0*.
rewrite *comp_id_r* in *H0*.
apply *H0*.
Qed.

Lemma 110 (comp_inc_compat_ab_b) *Let $\alpha : A \rightharpoonup A$ and $\beta : A \rightharpoonup B$. Then,*

$$\alpha \sqsubseteq id_A \Rightarrow \alpha \cdot \beta \sqsubseteq \beta.$$

Lemma *comp_inc_compat_ab_b* {*A B* : *eqType*} {*alpha* : *Rel A A*} {beta : *Rel A B*}:
 alpha ⊑ *Id A* → (*alpha* ⋅ beta) ⊑ beta.
Proof.
move ⇒ *H*.
move : (@*comp_inc_compat_ab_a'b* _ _ _ _ _ beta *H*) ⇒ *H0*.
rewrite *comp_id_l* in *H0*.
apply *H0*.
Qed.

Lemma 111 (comp_inc_compat_b_ab) *Let $\alpha : A \rightharpoonup A$ and $\beta : A \rightharpoonup B$. Then,*

$$id_A \sqsubseteq \alpha \Rightarrow \beta \sqsubseteq \alpha \cdot \beta.$$

Lemma *comp_inc_compat_b_ab* {*A B* : *eqType*} {*alpha* : *Rel A A*} {beta : *Rel A B*}:
 Id A ⊑ *alpha* → beta ⊑ (*alpha* ⋅ beta).
Proof.
move ⇒ *H*.
move : (@*comp_inc_compat_ab_a'b* _ _ _ _ _ beta *H*) ⇒ *H0*.
rewrite *comp_id_l* in *H0*.
apply *H0*.
Qed.

4.3

Lemma 112 (inv_move) *Let $\alpha : A \rightharpoonup B$ and $\beta : B \rightharpoonup A$. Then,*

$$\alpha = \beta^\sharp \Leftrightarrow \alpha^\sharp = \beta.$$

Lemma *inv_move* {*A B* : *eqType*} {*alpha* : *Rel A B*} {beta : *Rel B A*}:
 alpha = beta # ↔ *alpha* # = beta.
Proof.
split; move ⇒ *H*.
by [rewrite *H inv_invol*].
by [rewrite -*H inv_invol*].
Qed.

Lemma 113 (comp_inv_inv) *Let $\alpha : A \to B$ and $\beta : B \to C$. Then,*

$$\alpha \cdot \beta = (\beta^\sharp \cdot \alpha^\sharp)^\sharp.$$

Lemma *comp_inv_inv* {*A B C : eqType*} {*alpha : Rel A B*} {beta : *Rel B C*}:
 alpha   beta = (beta #   *alpha* #) #.
Proof.
apply *inv_move*.
apply *comp_inv*.
Qed.

Lemma 114 (inv_inc_move) *Let $\alpha : A \to B$ and $\beta : B \to A$. Then,*

$$\alpha \sqsubseteq \beta^\sharp \Leftrightarrow \alpha^\sharp \sqsubseteq \beta.$$

Lemma *inv_inc_move* {*A B : eqType*} {*alpha : Rel A B*} {beta : *Rel B A*}:
 alpha   beta # $\leftrightarrow$ *alpha* #   beta.
Proof.
split; move $\Rightarrow$ *H*.
rewrite -(@*inv_invol* _ _ beta).
apply *inc_inv*.
apply *H*.
rewrite -(@*inv_invol* _ _ *alpha*).
apply *inc_inv*.
apply *H*.
Qed.

Lemma 115 (inv_invol2) *Let $\alpha, \beta : A \to B$. Then,*

$$\alpha^\sharp = \beta^\sharp \Rightarrow \alpha = \beta.$$

Lemma *inv_invol2* {*A B : eqType*} {*alpha* beta : *Rel A B*}:
 alpha # = beta # $\to$ *alpha* = beta.
Proof.
move $\Rightarrow$ *H*.
rewrite -(@*inv_invol* _ _ *alpha*) -(@*inv_invol* _ _ beta).
apply f_equal.
apply *H*.
Qed.

Lemma 116 (inv_inc_invol) *Let* $\alpha, \beta : A \rightharpoonup B$. *Then,*

$$\alpha^\sharp \sqsubseteq \beta^\sharp \Rightarrow \alpha \sqsubseteq \beta.$$

Lemma *inv_inc_invol* $\{A\ B : eqType\}$ $\{alpha\ \text{beta} : Rel\ A\ B\}$:
  $alpha\ \#\quad \text{beta}\ \# \rightarrow alpha\quad \text{beta}$.
Proof.
move $\Rightarrow H$.
rewrite -(@*inv_invol* _ _ *alpha*) -(@*inv_invol* _ _ beta).
apply *inc_inv*.
apply H.
Qed.

Lemma 117 (inv_cupP_distr, inv_cup_distr) *Let* $f : (C \rightharpoonup D) \rightarrow (A \rightharpoonup B)$ *and* $P : predicate$. *Then,*

$$(\sqcup_{P(\alpha)} f(\alpha))^\sharp = (\sqcup_{P(\alpha)} f(\alpha)^\sharp).$$

Lemma *inv_cupP_distr* $\{A\ B\ C\ D : eqType\}$ $\{f : Rel\ C\ D \rightarrow Rel\ A\ B\}$ $\{P : Rel\ C\ D \rightarrow$
Prop$\}$:
  $(\quad _-\{P\}\ f)\ \# = (\quad _-\{P\}$ (fun *alpha* : *Rel C D* $\Rightarrow f$ *alpha* $\#$)).
Proof.
apply *inc_antisym*.
rewrite -*inv_inc_move*.
apply *inc_cupP*.
assert ($\forall$ beta : *Rel C D*, *P* beta $\rightarrow f$ beta $\#\qquad _-\{P\}$ (fun *alpha* : *Rel C D* $\Rightarrow f$
alpha $\#$)).
apply *inc_cupP*.
apply *inc_refl*.
move $\Rightarrow$ beta *H0*.
rewrite *inv_inc_move*.
apply (H _ *H0*).
apply *inc_cupP*.
move $\Rightarrow$ beta *H0*.
apply *inc_inv*.
move : *H0*.
apply *inc_cupP*.
apply *inc_refl*.
Qed.

Lemma *inv_cup_distr* $\{A\ B : eqType\}$ $\{alpha\ \text{beta} : Rel\ A\ B\}$:
  $(alpha\quad \text{beta})\ \# = alpha\ \#\quad \text{beta}\ \#$.
Proof.
by [rewrite *cup_to_cupP* -*inv_cupP_distr* -*cup_to_cupP*].

```
Qed.
```

> **Lemma 118 (inv_capP_distr, inv_cap_distr)** *Let $f : (C \rightharpoonup D) \to (A \rightharpoonup B)$ and P : predicate. Then,*
> $$(\sqcap_{P(\alpha)} f(\alpha))^{\sharp} = (\sqcap_{P(\alpha)} f(\alpha)^{\sharp}).$$

```
Lemma inv_capP_distr {A B C D : eqType} {f : Rel C D → Rel A B} {P : Rel C D →
Prop}:
 (    _{P} f) # = (    _{P} (fun alpha : Rel C D ⇒ f alpha #)).
Proof.
apply inc_antisym.
apply inc_capP.
move ⇒ beta H.
apply inc_inv.
move : H.
apply inc_capP.
apply inc_refl.
rewrite inv_inc_move.
apply inc_capP.
assert (∀ beta : Rel C D, P beta →    _{P} (fun alpha : Rel C D ⇒ f alpha #)      f
beta #).
apply inc_capP.
apply inc_refl.
move ⇒ beta H0.
rewrite -inv_inc_move.
apply (H _ H0).
Qed.

Lemma inv_cap_distr {A B : eqType} {alpha beta : Rel A B}:
 (alpha     beta) # = alpha #     beta #.
Proof.
by [rewrite cap_to_capP -inv_capP_distr -cap_to_capP].
Qed.
```

> **Lemma 119 (rpc_inv_distr)** *Let $\alpha, \beta : A \rightharpoonup B$. Then,*
> $$(\alpha \Rightarrow \beta)^{\sharp} = \alpha^{\sharp} \Rightarrow \beta^{\sharp}.$$

```
Lemma rpc_inv_distr {A B : eqType} {alpha beta : Rel A B}:
 (alpha » beta) # = alpha # » beta #.
Proof.
apply inc_lower.
move ⇒ gamma.
```

split; move $\Rightarrow$ *H*.
apply *inc_rpc*.
rewrite *inv_inc_move inv_cap_distr inv_invol*.
rewrite *-inc_rpc -inv_inc_move*.
apply *H*.
rewrite *inv_inc_move inc_rpc*.
rewrite *-(@inv_invol _ _ alpha) -inv_cap_distr -inv_inc_move*.
apply *inc_rpc*.
apply *H*.
Qed.

Lemma 120 (inv_empty)

$$\phi_{AB}{}^\sharp = \phi_{BA}.$$

Lemma *inv_empty* $\{A\ B : eqType\}$: $A\ B\ \#\ =\ B\ A.$
Proof.
apply *inc_antisym*.
rewrite *-inv_inc_move*.
apply *inc_empty_alpha*.
apply *inc_empty_alpha*.
Qed.

Lemma 121 (inv_universal)

$$\nabla_{AB}{}^\sharp = \nabla_{BA}.$$

Lemma *inv_universal* $\{A\ B : eqType\}$: $A\ B\ \#\ =\ B\ A.$
Proof.
apply *inc_antisym*.
apply *inc_alpha_universal*.
rewrite *inv_inc_move*.
apply *inc_alpha_universal*.
Qed.

Lemma 122 (inv_id)

$$id_A{}^\sharp = id_A.$$

Lemma *inv_id* $\{A : eqType\}$: $(Id\ A)\ \#\ =\ Id\ A.$
Proof.
replace $(Id\ A\ \#)$ with $((Id\ A\ \#)\ \#\ Id\ A\ \#).$
by [rewrite *-comp_inv comp_id_l inv_invol*].
by [rewrite *inv_invol comp_id_l*].
Qed.

Lemma 123 (inv_complement) *Let* $\alpha : A \rightarrow B$. *Then,*

$$(\alpha^-)^\sharp = (\alpha^\sharp)^-.$$

Lemma *inv_complement* $\{A\ B : eqType\}$ $\{alpha : Rel\ A\ B\}$: $(alpha\ \hat{})\ \# = (alpha\ \#)\ \hat{}$.
Proof.
apply *inc_antisym*.
apply *inc_rpc*.
rewrite -*inv_cap_distr*.
rewrite *cap_comm* -*inv_inc_move inv_empty*.
rewrite *cap_complement_empty*.
apply *inc_refl*.
rewrite *inv_inc_move*.
apply *inc_rpc*.
replace $(((alpha\ \#)\ \hat{})\ \#\quad alpha)$ with $(((alpha\ \#)\ \hat{})\ \#\quad (alpha\ \#)\ \#)$.
rewrite -*inv_cap_distr*.
rewrite *cap_comm* -*inv_inc_move inv_empty*.
rewrite *cap_complement_empty*.
apply *inc_refl*.
by [rewrite *inv_invol*].
Qed.

Lemma 124 (inv_difference_distr) *Let* $\alpha, \beta : A \rightarrow B$. *Then,*

$$(\alpha - \beta)^\sharp = \alpha^\sharp - \beta^\sharp.$$

Lemma *inv_difference_distr* $\{A\ B : eqType\}$ $\{alpha$ beta $: Rel\ A\ B\}$:
$(alpha - $ beta$)\ \# = alpha\ \# - $ beta $\#$.
Proof.
rewrite *inv_cap_distr*.
by [rewrite *inv_complement*].
Qed.

4.4

Lemma 125 (comp_cupP_distr_l, comp_cup_distr_l) *Let* $\alpha : A \rightarrow B$, $f : (D \rightarrow E) \rightarrow (B \rightarrow C)$ *and* $P : predicate$. *Then,*

$$\alpha \cdot (\sqcup_{P(\beta)} f(\beta)) = \sqcup_{P(\beta)} (\alpha \cdot f(\beta)).$$

Lemma *comp_cupP_distr_l* $\{A\ B\ C\ D\ E : eqType\}$

{*alpha* : *Rel A B*} {*f* : *Rel D E* → *Rel B C*} {*P* : *Rel D E* → Prop}:
alpha   (_{*P*} *f*) =   _{*P*} (fun beta : *Rel D E* ⇒ (*alpha*   *f* beta)).
Proof.
apply *inc_upper*.
move ⇒ *gamma*.
split; move ⇒ *H*.
rewrite -(@*inv_invol* _ _ *alpha*) in *H*.
apply *inc_residual* in *H*.
apply *inc_cupP*.
assert (∀ beta : *Rel D E*, *P* beta → *f* beta   (*alpha* #   *gamma*)).
apply *inc_cupP*.
apply *H*.
move ⇒ beta *H1*.
rewrite -(@*inv_invol* _ _ *alpha*).
apply *inc_residual*.
apply (*H0* _ *H1*).
rewrite -(@*inv_invol* _ _ *alpha*).
apply *inc_residual*.
apply *inc_cupP*.
assert (∀ beta : *Rel D E*, *P* beta → (*alpha*   *f* beta)   *gamma*).
apply *inc_cupP*.
apply *H*.
move ⇒ beta *H1*.
apply *inc_residual*.
rewrite *inv_invol*.
apply (*H0* _ *H1*).
Qed.

Lemma *comp_cup_distr_l*
 {*A B C* : *eqType*} {*alpha* : *Rel A B*} {beta *gamma* : *Rel B C*}:
 alpha   (beta   *gamma*) = (*alpha*   beta)   (*alpha*   *gamma*).
Proof.
by [rewrite *cup_to_cupP* -*comp_cupP_distr_l* -*cup_to_cupP*].
Qed.

Lemma 126 (comp_cupP_distr_r, comp_cup_distr_r) *Let* $f : (D \to E) \to (A \to B)$, $\beta : B \to C$ *and* P : *predicate. Then,*

$$(\sqcup_{P(\alpha)} f(\alpha)) \cdot \beta = \sqcup_{P(\alpha)}(f(\alpha) \cdot \beta).$$

Lemma *comp_cupP_distr_r* {*A B C D E* : *eqType*}
 {beta : *Rel B C*} {*f* : *Rel D E* → *Rel A B*} {*P* : *Rel D E* → Prop}:
 (_{*P*} *f*)   beta =   _{*P*} (fun *alpha* : *Rel D E* ⇒ (*f alpha*   beta)).

Proof.
replace (fun *alpha* : *Rel D E* ⇒ *f* *alpha*   beta) with (fun *alpha* : *Rel D E* ⇒ (beta #
    f *alpha* #) #).
rewrite -*inv_cupP_distr*.
rewrite -*comp_cupP_distr_l*.
rewrite -*inv_cupP_distr*.
rewrite *comp_inv*.
by [rewrite *inv_invol* *inv_invol*].
apply *functional_extensionality*.
move ⇒ *x*.
rewrite *comp_inv*.
by [rewrite *inv_invol* *inv_invol*].
Qed.

Lemma *comp_cup_distr_r*
  {*A B C* : *eqType*} {*alpha* beta : *Rel A B*} {*gamma* : *Rel B C*}:
  (*alpha*   beta)   *gamma* = (*alpha*   *gamma*)   (beta   *gamma*).
Proof.
by [rewrite (@*cup_to_cupP* _ _ _ _ _ _ *id*) *comp_cupP_distr_r* -*cup_to_cupP*].
Qed.

Lemma 127 (comp_capP_distr) *Let* $\alpha : A \rightharpoonup B$, $\gamma : C \rightharpoonup D$, $f : (E \rightharpoonup F) \rightarrow (B \rightharpoonup C)$ *and* P : *predicate. Then,*

$$\alpha \cdot (\sqcap_{P(\beta)} f(\beta)) \cdot \gamma \sqsubseteq \sqcap_{P(\beta)}(\alpha \cdot f(\beta) \cdot \gamma).$$

Lemma *comp_capP_distr* {*A B C D E F* : *eqType*}
  {*alpha* : *Rel A B*} {*gamma* : *Rel C D*}
  {*f* : *Rel E F* → *Rel B C*} {*P* : *Rel E F* → Prop}:
  (*alpha*   (_{*P*} *f*))   *gamma*
      _{*P*} (fun beta : *Rel E F* ⇒ ((*alpha*   *f* beta)   *gamma*)).
Proof.
apply *inc_capP*.
move ⇒ beta *H*.
apply *comp_inc_compat_ab_a'b*.
apply *comp_inc_compat_ab_ab'*.
move : *H*.
apply *inc_capP*.
apply *inc_refl*.
Qed.

Lemma 128 (comp_capP_distr_l, comp_cap_distr_l) *Let* $\alpha : A \rightarrow B$, $f : (D \rightarrow E) \rightarrow (B \rightarrow C)$ *and* $P :$ *predicate. Then,*

$$\alpha \cdot (\sqcap_{P(\beta)} f(\beta)) \sqsubseteq \sqcap_{P(\beta)} (\alpha \cdot f(\beta)).$$

Lemma *comp_capP_distr_l* {*A B C D E* : *eqType*}
 {*alpha* : *Rel A B*} {*f* : *Rel D E* → *Rel B C*} {*P* : *Rel D E* → Prop}:
 (*alpha*   (_{*P*} *f*))   _{*P*} (fun beta : *Rel D E* ⇒ (*alpha*   *f* beta)).
Proof.
move : (@*comp_capP_distr* _ _ _ _ _ _ *alpha* (*Id C*) *f P*) ⇒ *H*.
rewrite *comp_id_r* in *H*.
replace (fun beta : *Rel D E* ⇒ (*alpha*   *f* beta)   *Id C*) with (fun beta : *Rel D E* ⇒ (*alpha*   *f* beta)) in *H*.
apply *H*.
apply *functional_extensionality*.
move ⇒ *x*.
by [rewrite *comp_id_r*].
Qed.

Lemma *comp_cap_distr_l*
 {*A B C* : *eqType*} {*alpha* : *Rel A B*} {beta *gamma* : *Rel B C*}:
 (*alpha*   (beta   *gamma*))   ((*alpha*   beta)   (*alpha*   *gamma*)).
Proof.
rewrite *cap_to_capP* (@*cap_to_capP* _ _ _ _ _ _ *id*).
apply *comp_capP_distr_l*.
Qed.

Lemma 129 (comp_capP_distr_r, comp_cap_distr_r) *Let* $\beta : B \rightarrow C$, $f : (D \rightarrow E) \rightarrow (A \rightarrow B)$ *and* $P :$ *predicate. Then,*

$$(\sqcap_{P(\alpha)} f(\alpha)) \cdot \beta \sqsubseteq \sqcap_{P(\alpha)} (f(\alpha) \cdot \beta).$$

Lemma *comp_capP_distr_r*
 {*A B C D E* : *eqType*} {beta : *Rel B C*} {*f* : *Rel D E* → *Rel A B*} {*P* : *Rel D E* → Prop}:
 ((_{*P*} *f*)   beta)   _{*P*} (fun *alpha* : *Rel D E* ⇒ (*f alpha*   beta)).
Proof.
move : (@*comp_capP_distr* _ _ _ _ _ _ (*Id A*) beta *f P*) ⇒ *H*.
rewrite *comp_id_l* in *H*.
replace (fun *alpha* : *Rel D E* ⇒ (*Id A*   *f alpha*)   beta) with (fun *alpha* : *Rel D E* ⇒ *f alpha*   beta) in *H*.
apply *H*.
apply *functional_extensionality*.

move $\Rightarrow x$.
by [rewrite *comp_id_l*].
Qed.

Lemma *comp_cap_distr_r*
  {*A B C* : *eqType*} {*alpha* beta : *Rel A B*} {*gamma* : *Rel B C*}:
  ((*alpha*   beta)   *gamma*)   ((*alpha*   *gamma*)   (beta   *gamma*))).
Proof.
rewrite (@*cap_to_capP* _ _ _ _ _ _ *id*) (@*cap_to_capP* _ _ _ _ _ _ (fun $x \Rightarrow x$ *gamma*)).
apply *comp_capP_distr_r*.
Qed.

Lemma 130 (comp_empty_l, comp_empty_r) *Let* $\alpha : A \rightarrow B$, $\beta : B \rightarrow C$. *Then,*

$$\alpha \cdot \phi_{BC} = \phi_{AB} \cdot \beta = \phi_{AC}.$$

Lemma *comp_empty_r* {*A B C* : *eqType*} {*alpha* : *Rel A B*}: *alpha*   *B C* =   *A C*.
Proof.
apply *inc_antisym*.
rewrite -(@*inv_invol* _ _ *alpha*).
apply *inc_residual*.
apply *inc_empty_alpha*.
apply *inc_empty_alpha*.
Qed.

Lemma *comp_empty_l* {*A B C* : *eqType*} {beta : *Rel B C*}:   *A B*   beta =   *A C*.
Proof.
rewrite -(@*inv_invol* _ _ (*A B*   beta)).
rewrite -*inv_move comp_inv inv_empty inv_empty*.
apply *comp_empty_r*.
Qed.

Lemma 131 (comp_either_empty) *Let* $\alpha : A \rightarrow B$, $\beta : B \rightarrow C$. *Then,*

$$\alpha = \phi_{AB} \vee \beta = \phi_{BC} \Rightarrow \alpha \cdot \beta = \phi_{AC}.$$

Lemma *comp_either_empty* {*A B C* : *eqType*} {*alpha* : *Rel A B*} {beta : *Rel B C*}:
  alpha =   *A B* $\vee$ beta =   *B C* $\rightarrow$ *alpha*   beta =   *A C*.
Proof.
case; move $\Rightarrow H$.
rewrite H.
apply *comp_empty_l*.
rewrite H.
apply *comp_empty_r*.

Qed.

Lemma 132 (comp_neither_empty) *Let $\alpha : A \rightharpoonup B$, $\beta : B \rightharpoonup C$. Then,*

$$\alpha \cdot \beta \neq \phi_{AC} \Rightarrow \alpha \neq \phi_{AB} \wedge \beta \neq \phi_{BC}.$$

Lemma *comp_neither_empty* {*A B C : eqType*} {*alpha : Rel A B*} {beta : *Rel B C*}:
  alpha   beta ≠   *A C* → *alpha* ≠   *A B* ∧ beta ≠   *B C*.
Proof.
move ⇒ *H*.
split; move ⇒ *H0*.
apply *H*.
rewrite *H0*.
apply *comp_empty_l*.
apply *H*.
rewrite *H0*.
apply *comp_empty_r*.
Qed.

4.5 Tarski

Lemma 133 (lemma_for_tarski1) *Let $\alpha : A \rightharpoonup B$ and $\alpha \neq \phi_{AB}$. Then,*

$$\nabla_{IA} \cdot \alpha \cdot \nabla_{BI} = id_I.$$

Lemma *lemma_for_tarski1* {*A B : eqType*} {*alpha : Rel A B*}:
  alpha ≠   *A B* → ((*i A*   *alpha*)   *B i*) = *Id i*.
Proof.
move ⇒ *H*.
assert (((*i A*   *alpha*)   *B i*) ≠   *i i*).
move ⇒ *H0*.
apply *H*.
apply *inc_antisym*.
apply (@*inc_trans* _ _ _ ((*A i*   ((*i A*   *alpha*)   *B i*))   *i B*)).
rewrite *comp_assoc comp_assoc unit_universal*.
rewrite -*comp_assoc* -*comp_assoc unit_universal*.
apply (@*inc_trans* _ _ _ ((Id *A*   *alpha*)   Id *B*)).
rewrite *comp_id_l comp_id_r*.
apply *inc_refl*.
apply *comp_inc_compat*.
apply *comp_inc_compat_ab_a'b*.
apply *inc_alpha_universal*.

85

```
apply inc_alpha_universal.
rewrite H0 comp_empty_r comp_empty_l.
apply inc_refl.
apply inc_empty_alpha.
case (@unit_empty_or_universal ((  i A    alpha)      B i)); move ⇒ H1.
apply False_ind.
apply (H0 H1).
rewrite unit_identity_is_universal.
apply H1.
Qed.
```

Lemma 134 (lemma_for_tarski2)

$$\nabla_{AI} \cdot \nabla_{IB} = \nabla_{AB}.$$

```
Lemma lemma_for_tarski2 {A B : eqType}:    A i      i B =    A B.
Proof.
apply inc_antisym.
apply inc_alpha_universal.
apply (@inc_trans _ _ _ (   A A      A B)).
apply (@inc_trans _ _ _ (Id A      A B)).
rewrite comp_id_l.
apply inc_refl.
apply comp_inc_compat_ab_a'b.
apply inc_alpha_universal.
rewrite -(@unit_universal A) comp_assoc.
apply comp_inc_compat_ab_ab'.
apply inc_alpha_universal.
Qed.
```

Lemma 135 (tarski) *Let $\alpha : A \to B$ and $\alpha \neq \phi_{AB}$. Then,*

$$\nabla_{AA} \cdot \alpha \cdot \nabla_{BB} = \nabla_{AB}.$$

```
Lemma tarski {A B : eqType} {alpha : Rel A B}:
  alpha ≠    A B → ((   A A    alpha)      B B) =    A B.
Proof.
move ⇒ H.
rewrite -(@unit_universal A) -(@unit_universal B).
move : (@lemma_for_tarski1 _ _ alpha H) ⇒ H0.
rewrite -comp_assoc (@comp_assoc _ _ _ _ (   A i)) (@comp_assoc _ _ _ _ (   A i)).
rewrite H0 comp_id_r.
apply lemma_for_tarski2.
```

Qed.

Lemma 136 (comp_universal1) *Let $B \neq \emptyset$. Then,*

$$\nabla_{AB} \cdot \nabla_{BC} = \nabla_{AC}.$$

Lemma *comp_universal* {*A B C : eqType*} : *B →* ∇ *A B* · ∇ *B C =* ∇ *A C*.
Proof.
move ⇒ *b*.
replace (∇ *A B*) with (∇ *A B* · ∇ *B B*).
rewrite -(@*lemma_for_tarski2 A B*) -(@*lemma_for_tarski2 B C*).
rewrite (@*comp_assoc* _ _ _ _ (∇ *A i*)) (@*comp_assoc* _ _ _ _ (∇ *A i*)) -(@*comp_assoc* _
_ _ _ _ (∇ *B i*)).
rewrite *lemma_for_tarski1*.
rewrite *comp_id_l*.
apply *lemma_for_tarski2*.
apply *not_eq_sym*.
move ⇒ *H*.
apply *either_empty* in *H*.
case *H*; move ⇒ *H0*.
apply (*H0 b*).
apply (*H0 b*).
apply *inc_antisym*.
apply *inc_alpha_universal*.
apply (@*inc_trans* _ _ _ (∇ *A B* · Id *B*)).
rewrite *comp_id_r*.
apply *inc_refl*.
apply *comp_inc_compat_ab_ab'*.
apply *inc_alpha_universal*.
Qed.

Lemma 137 (comp_universal2)

$$\nabla_{IA}{}^{\sharp} \cdot \nabla_{IB} = \nabla_{AB}.$$

Lemma *comp_universal2* {*A B : eqType*}: ∇ *i A* # · ∇ *i B =* ∇ *A B*.
Proof.
rewrite *inv_universal*.
apply *lemma_for_tarski2*.
Qed.

Lemma 138 (empty_equivalence1, empty_equivalence2, empty_equivalence3)

$$A = \emptyset \Leftrightarrow \nabla_{IA} = \phi_{IA} \Leftrightarrow \nabla_{AA} = \phi_{AA} \Leftrightarrow id_A = \phi_{AA}.$$

Lemma *empty_equivalence1* $\{A : eqType\}$: $(A \to False) \leftrightarrow \quad i\ A = \quad i\ A$.
Proof.
move : $(@either\_empty\ i\ A) \Rightarrow H$.
split; move $\Rightarrow H0$.
apply *Logic.eq_sym*.
apply H.
right.
apply $H0$.
apply *Logic.eq_sym* in $H0$.
apply H in $H0$.
case $H0$.
move $\Rightarrow H1\ H2$.
apply $H1$.
apply tt.
by $\|$.
Qed.

Lemma *empty_equivalence2* $\{A : eqType\}$: $(A \to False) \leftrightarrow \quad A\ A = \quad A\ A$.
Proof.
move : $(@either\_empty\ A\ A) \Rightarrow H$.
split; move $\Rightarrow H0$.
apply *Logic.eq_sym*.
apply H.
left.
apply $H0$.
apply *Logic.eq_sym* in $H0$.
apply H in $H0$.
case $H0$.
by $\|$.
by $\|$.
Qed.

Lemma *empty_equivalence3* $\{A : eqType\}$: $(A \to False) \leftrightarrow Id\ A = \quad A\ A$.
Proof.
split; move $\Rightarrow H$.
assert ($\quad A\ A = \quad A\ A$).
apply *empty_equivalence2*.
apply H.
apply *RelAB_unique*.
apply *Logic.eq_sym*.

apply *H0*.

assert (*A A* =   *A A*).

by [rewrite -(@*comp_id_r* _ _ (*A A*)) *H comp_empty_r*].

apply *either_empty* in *H0*.

case *H0*.

by [].

by [].

Qed.

End *main*.

Chapter 5

Library Functions_Mappings

Require Import *Basic_Notations_Set*.
Require Import *Basic_Lemmas*.
Require Import *Relation_Properties*.
Require Import *Logic.FunctionalExtensionality*.

Module *main* (*def* : *Relation*).
Import *def*.
Module *Basic_Lemmas* := *Basic_Lemmas.main def*.
Module *Relation_Properties* := *Relation_Properties.main def*.
Import *Basic_Lemmas Relation_Properties*.

5.1   ,   ,

Lemma 139 (id_function) $id_A : A \rightharpoonup A$ *is a function.*

Lemma *id_function* {A : *eqType*}: *function_r* (*Id A*).
Proof.
rewrite /*function_r*/*total_r*/*univalent_r*.
rewrite *inv_id comp_id_l*.
split.
apply *inc_refl*.
apply *inc_refl*.
Qed.

Lemma 140 (unit_function) $\nabla_{AI} : A \rightharpoonup I$ *is a function.*

Lemma *unit_function* {A : *eqType*}: *function_r* (*A i*).
Proof.
rewrite /*function_r*/*total_r*/*univalent_r*.

```
rewrite inv_universal lemma_for_tarski2 unit_identity_is_universal.
split.
apply inc_alpha_universal.
apply inc_alpha_universal.
Qed.
```

Lemma 141 (total_comp) *Let $\alpha : A \rightharpoonup B$ and $\beta : B \rightharpoonup C$ be total relations, then $\alpha \cdot \beta$ is also a total relation.*

```
Lemma total_comp {A B C : eqType} {alpha : Rel A B} {beta : Rel B C}:
 total_r alpha → total_r beta → total_r (alpha ⋅ beta).
Proof.
rewrite /total_r.
move ⇒ H H0.
rewrite comp_inv comp_assoc -(@comp_assoc _ _ _ _ beta).
apply (@inc_trans _ _ _ _ _ H).
apply comp_inc_compat_ab_ab'.
apply comp_inc_compat_b_ab.
apply H0.
Qed.
```

Lemma 142 (univalent_comp) *Let $\alpha : A \rightharpoonup B$ and $\beta : B \rightharpoonup C$ be univalent relations, then $\alpha \cdot \beta$ is also a univalent relation.*

```
Lemma univalent_comp {A B C : eqType} {alpha : Rel A B} {beta : Rel B C}:
 univalent_r alpha → univalent_r beta → univalent_r (alpha ⋅ beta).
Proof.
rewrite /univalent_r.
move ⇒ H H0.
rewrite comp_inv comp_assoc -(@comp_assoc _ _ _ _ (alpha #)).
apply (fun H' ⇒ @inc_trans _ _ _ _ _ H' H0).
apply comp_inc_compat_ab_ab'.
apply comp_inc_compat_ab_b.
apply H.
Qed.
```

Lemma 143 (function_comp) *Let $\alpha : A \rightarrow B$ and $\beta : B \rightarrow C$ be functions, then $\alpha \cdot \beta$ is also a function.*

```
Lemma function_comp {A B C : eqType} {alpha : Rel A B} {beta : Rel B C}:
 function_r alpha → function_r beta → function_r (alpha ⋅ beta).
Proof.
elim ⇒ H H0.
```

```
elim ⇒ H1 H2.
split.
apply (total_comp H H1).
apply (univalent_comp H0 H2).
Qed.
```

Lemma 144 (total_comp2) *Let $\alpha : A \to B$, $\beta : B \to C$ and $\alpha \cdot \beta$ be a total relation, then α is also a total relation.*

```
Lemma total_comp2 {A B C : eqType} {alpha : Rel A B} {beta : Rel B C}:
 total_r (alpha    beta) → total_r alpha.
Proof.
move ⇒ H.
apply inc_def1 in H.
rewrite comp_inv cap_comm comp_assoc in H.
rewrite /total_r.
rewrite H.
apply (@inc_trans _ _ _ _ _ (@dedekind _ _ _ _ _ _)).
apply comp_inc_compat.
apply cap_l.
rewrite comp_id_r.
apply cap_r.
Qed.
```

Lemma 145 (univalent_comp2) *Let $\alpha : A \to B$, $\beta : B \to C$, $\alpha \cdot \beta$ be a univalent relation and $\alpha^\sharp$ be a total relation, then β is a univalent relation.*

```
Lemma univalent_comp2 {A B C : eqType} {alpha : Rel A B} {beta : Rel B C}:
 univalent_r (alpha    beta) → total_r (alpha #) → univalent_r beta.
Proof.
move ⇒ H H0.
apply (fun H' ⇒ @inc_trans _ _ _ _ _ H' H).
rewrite comp_inv comp_assoc -(@comp_assoc _ _ _ _ _ alpha).
apply comp_inc_compat_ab_ab'.
rewrite /total_r in H0.
rewrite inv_invol in H0.
apply (comp_inc_compat_b_ab H0).
Qed.
```

Lemma 146 (total_inc) *Let $\alpha : A \to B$ be a total relation and $\alpha \sqsubseteq \beta$, then β is also a total relation.*

```
Lemma total_inc {A B : eqType} {alpha beta : Rel A B}:
```

$total\_r$ $alpha$ $\to$ $alpha$ $\qquad$ beta $\to total\_r$ beta.
Proof.
move $\Rightarrow$ *H H0*.
apply (@*inc_trans* _ _ _ _ _ *H*).
apply *comp_inc_compat*.
apply *H0*.
apply (@*inc_inv* _ _ _ _ *H0*).
Qed.

> **Lemma 147 (univalent_inc)** *Let* $\alpha : A \to B$ *be a univalent relation and* $\beta \sqsubseteq \alpha$, *then* β *is also a univalent relation.*

Lemma *univalent_inc* $\{A\ B : eqType\}$ $\{alpha$ beta $: Rel\ A\ B\}$:
 $univalent\_r$ $alpha$ $\to$ beta $\qquad$ $alpha$ $\to univalent\_r$ beta.
Proof.
move $\Rightarrow$ *H H0*.
apply (fun *H'* $\Rightarrow$ @*inc_trans* _ _ _ _ _ *H' H*).
apply *comp_inc_compat*.
apply (@*inc_inv* _ _ _ _ *H0*).
apply *H0*.
Qed.

> **Lemma 148 (function_inc)** *Let* $\alpha, \beta : A \to B$ *be functions and* $\alpha \sqsubseteq \beta$. *Then,*
>
> $$\alpha = \beta.$$

Lemma *function_inc* $\{A\ B : eqType\}$ $\{alpha$ beta $: Rel\ A\ B\}$:
 $function\_r$ $alpha$ $\to function\_r$ beta $\to alpha$ $\qquad$ beta $\to alpha =$ beta.
Proof.
move $\Rightarrow$ *H H0 H1*.
apply *inc_antisym*.
apply *H1*.
apply (@*inc_trans* _ _ _ (($alpha$ $\qquad$ $alpha$ #) $\qquad$ beta)).
apply *comp_inc_compat_b_ab*.
apply *H*.
move : (@*inc_inv* _ _ _ _ *H1*) $\Rightarrow$ *H2*.
apply (@*inc_trans* _ _ _ (($alpha$ $\qquad$ beta #) $\qquad$ beta)).
apply *comp_inc_compat_ab_a'b*.
apply *comp_inc_compat_ab_ab'*.
apply *H2*.
rewrite *comp_assoc*.
apply *comp_inc_compat_ab_a*.
apply *H0*.

Qed.

Lemma 149 (total_universal) *If ∇_{IB} be a total relation, then*

$$\nabla_{AB} \cdot \nabla_{BC} = \nabla_{AC}.$$

Lemma *total_universal* $\{A\ B\ C : eqType\}$:
 total_r ($i\ B$) $\to$ $A\ B$ $B\ C =$ $A\ C$.
Proof.
move $\Rightarrow H$.
rewrite -(@*lemma_for_tarski2* $A\ B$) -(@*lemma_for_tarski2* $B\ C$).
rewrite *comp_assoc* -(@*comp_assoc* _ _ _ _ ($i\ B$)).
replace ($i\ B$ $B\ i$) with ($Id\ i$).
rewrite *comp_id_l*.
apply *lemma_for_tarski2*.
apply *inc_antisym*.
rewrite /*total_r* in H.
rewrite *inv_universal* in H.
apply H.
rewrite *unit_identity_is_universal*.
apply *inc_alpha_universal*.
Qed.

Lemma 150 (function_rel_inv_rel) *Let $\alpha : A \to B$ be function. Then,*

$$\alpha \cdot \alpha^{\sharp} \cdot \alpha = \alpha.$$

Lemma *function_rel_inv_rel* $\{A\ B : eqType\}$ $\{alpha : Rel\ A\ B\}$:
 function_r alpha $\to$ ($alpha$ $alpha\ \#$) $alpha = alpha$.
Proof.
move $\Rightarrow H$.
apply *inc_antisym*.
rewrite *comp_assoc*.
apply *comp_inc_compat_ab_a*.
apply H.
apply *comp_inc_compat_b_ab*.
apply H.
Qed.

Lemma 151 (function_capP_distr) *Let* $f : A \rightarrow B, g : D \rightarrow C$ *be functions,* $\theta :$ $(E \rightarrow F) \rightarrow (B \rightarrow C)$ *and* $P :$ *predicate. Then,*

$$f \cdot (\sqcap_{P(\theta)} \theta(\alpha)) \cdot g^{\sharp} = \sqcap_{P(\alpha)} (f \cdot \theta(\alpha) \cdot g^{\sharp}).$$

Lemma *function_capP_distr* {*A B C D E F : eqType*}
 {*f : Rel A B*} {*g : Rel D C*} {*theta : Rel E F → Rel B C*} {*P : Rel E F →* Prop}:
 function_r f → function_r g →
 (*f* (_{*P*} *theta*)) *g* # =
 _{*P*} (fun *alpha : Rel E F* ⇒ (*f theta alpha*) *g* #).
Proof.
elim ⇒ *H H0*.
elim ⇒ *H1 H2*.
apply *inc_antisym*.
apply *comp_capP_distr*.
apply (@*inc_trans* _ _ _ (((*f f* #) _{*P*} (fun *alpha : Rel E F* ⇒ (*f theta alpha*)
 g #)) (*g g* #))).
apply (@*inc_trans* _ _ _ ((*f f* #) (_{*P*} (fun *alpha : Rel E F* ⇒ (*f theta alpha*)
 g #)))).
apply (*comp_inc_compat_b_ab H*).
apply (*comp_inc_compat_a_ab H1*).
rewrite (@*comp_assoc* _ _ _ _ _ (*f* #)) *comp_assoc* -(@*comp_assoc* _ _ _ _ _ *g*) -*comp_assoc*.
apply *comp_inc_compat_ab_a'b*.
apply *comp_inc_compat_ab_ab'*.
apply (@*inc_trans* _ _ _ (_{*P*} (fun *alpha : Rel E F* ⇒ (*f* # ((*f theta alpha*) *g*
 #)) *g*))).
apply *comp_capP_distr*.
replace (fun *alpha : Rel E F* ⇒ (*f* # ((*f theta alpha*) *g* #)) *g*) with (fun *alpha*
 : Rel E F ⇒ ((*f* # *f*) *theta alpha*) (*g* # *g*)).
apply *inc_capP*.
move ⇒ beta *H3*.
apply (@*inc_trans* _ _ _ ((*f* # *f*) *theta* beta)).
apply (@*inc_trans* _ _ _ (((*f* # *f*) *theta* beta) (*g* # *g*))).
move : beta *H3*.
apply *inc_capP*.
apply *inc_refl*.
apply (*comp_inc_compat_ab_a H2*).
apply (*comp_inc_compat_ab_b H0*).
apply *functional_extensionality*.
move ⇒ *x*.
by [rewrite *comp_assoc comp_assoc comp_assoc comp_assoc comp_assoc*].
Qed.

Lemma 152 (function_cap_distr, function_cap_distr_l, function_cap_distr_r)
Let $f : A \to B, g : D \to C$ be functions and $\alpha, \beta : B \to C$. Then,

$$f \cdot (\alpha \sqcap \beta) \cdot g^{\sharp} = (f \cdot \alpha \cdot g^{\sharp}) \sqcap (f \cdot \beta \cdot g^{\sharp}).$$

Lemma *function_cap_distr*
 {A B C D : eqType} {f : Rel A B} {alpha beta *: Rel B C} {g : Rel D C}*:
 function_r f → function_r g →
 (f   (alpha   beta))   g # = ((f   alpha)   g #)   ((f   beta)   g #).
Proof.
rewrite (@*cap_to_capP* _ _ _ _ _ _ *id*) (@*cap_to_capP* _ _ _ _ _ _ (fun *x* ⇒ (*f   x*)   *g*
#)).
apply *function_capP_distr*.
Qed.

Lemma *function_cap_distr_l*
 {A B C : eqType} {f : Rel A B} {alpha beta *: Rel B C}*:
 function_r f →
 f   (alpha   beta) = (f   alpha)   (f   beta).
Proof.
move : (@*id_function C*) ⇒ *H*.
move ⇒ *H0*.
apply (@*function_cap_distr* _ _ _ _ *f alpha* beta) in *H*.
rewrite *inv_id comp_id_r comp_id_r comp_id_r* in *H*.
apply *H*.
apply *H0*.
Qed.

Lemma *function_cap_distr_r*
 {B C D : eqType} {alpha beta *: Rel B C} {g : Rel D C}*:
 function_r g →
 (alpha   beta)   g # = (alpha   g #)   (beta   g #).
Proof.
move : (@*id_function B*) ⇒ *H*.
move ⇒ *H0*.
apply (@*function_cap_distr* _ _ _ _ _ *alpha* beta *g*) in *H*.
rewrite *comp_id_l comp_id_l comp_id_l* in *H*.
apply *H*.
apply *H0*.
Qed.

> **Lemma 153 (function_move1)** *Let $\alpha : A \to B$ be a function, $\beta : B \to C$ and $\gamma : A \to C$. Then,*
> $$\gamma \sqsubseteq \alpha \cdot \beta \Leftrightarrow \alpha^{\sharp} \cdot \gamma \sqsubseteq \beta.$$

Lemma *function_move1* $\{A \; B \; C : eqType\}$ $\{alpha : Rel \; A \; B\}$ {beta $: Rel \; B \; C\}$ $\{gamma : Rel \; A \; C\}$:
 function_r alpha $\to$ (*gamma*   (*alpha*   beta) $\leftrightarrow$ (*alpha* #   *gamma*)   beta).
Proof.
move $\Rightarrow H$.
split; move $\Rightarrow H0$.
apply (@*inc_trans* _ _ _ ((*alpha* #   *alpha*)   beta)).
rewrite *comp_assoc*.
apply (*comp_inc_compat_ab_ab' H0*).
apply *comp_inc_compat_ab_b*.
apply H.
apply (@*inc_trans* _ _ _ ((*alpha*   *alpha* #)   *gamma*)).
apply *comp_inc_compat_b_ab*.
apply H.
rewrite *comp_assoc*.
apply (*comp_inc_compat_ab_ab' H0*).
Qed.

> **Lemma 154 (function_move2)** *Let $\beta : B \to C$ be a function, $\alpha : A \to B$ and $\gamma : A \to C$. Then,*
> $$\alpha \cdot \beta \sqsubseteq \gamma \Leftrightarrow \alpha \sqsubseteq \gamma \cdot \beta^{\sharp}.$$

Lemma *function_move2* $\{A \; B \; C : eqType\}$ $\{alpha : Rel \; A \; B\}$ {beta $: Rel \; B \; C\}$ $\{gamma : Rel \; A \; C\}$:
 function_r beta $\to$ ((*alpha*   beta)   *gamma* $\leftrightarrow$ *alpha*   (*gamma*   beta #)).
Proof.
move $\Rightarrow H$.
split; move $\Rightarrow H0$.
apply (@*inc_trans* _ _ _ ((*alpha*   beta)   beta #)).
rewrite *comp_assoc*.
apply *comp_inc_compat_a_ab*.
apply H.
apply (*comp_inc_compat_ab_a'b H0*).
apply (@*inc_trans* _ _ _ ((*gamma*   beta #)   beta)).
apply (*comp_inc_compat_ab_a'b H0*).
rewrite *comp_assoc*.
apply *comp_inc_compat_ab_a*.
apply H.

`Qed`.

Lemma 155 (function_rpc_distr) *Let* $f : A \rightarrow B, g : D \rightarrow C$ *be functions and* $\alpha, \beta : B \rightarrow C$. *Then,*

$$f \cdot (\alpha \Rightarrow \beta) \cdot g^{\sharp} = (f \cdot \alpha \cdot g^{\sharp}) \Rightarrow (f \cdot \beta \cdot g^{\sharp}).$$

`Lemma` *function_rpc_distr*
 $\{A\ B\ C\ D : eqType\}$ $\{f : Rel\ A\ B\}$ $\{alpha$ `beta` $: Rel\ B\ C\}$ $\{g : Rel\ D\ C\}$:
 function_r f $\rightarrow$ *function_r g* $\rightarrow$
 $(f \quad (alpha » $ `beta`$))\quad g\ \# = ((f \quad alpha)\quad g\ \#) » ((f \quad$ `beta`$)\quad g\ \#)$.
`Proof`.
`move` $\Rightarrow$ *H H0*.
`apply` *inc_lower*.
`move` $\Rightarrow$ *gamma*.
`split`; `move` $\Rightarrow$ *H1*.
`apply` *inc_rpc*.
`apply` (*function_move2 H0*).
`apply` (*function_move1 H*).
`apply` (@*inc_trans* _ _ _ (((*f #* \quad *gamma*)\quad *g*)\quad ((*f #* \quad ((*f* \quad *alpha*)\quad *g #*))\quad *g*))).
`rewrite` -*comp_assoc*.
`apply` (`fun` *H'* $\Rightarrow$ @*inc_trans* _ _ _ _ _ *H'* (@*comp_cap_distr_r* _ _ _ _ _ _)).
`apply` *comp_inc_compat_ab_a'b*.
`apply` *comp_cap_distr_l*.
`apply` (*function_move2 H0*) `in` *H1*.
`apply` (*function_move1 H*) `in` *H1*.
`rewrite` -*inc_rpc comp_assoc*.
`apply` (@*inc_trans* _ _ _ _ _ *H1*).
`apply` *rpc_inc_compat_r*.
`rewrite` *comp_assoc comp_assoc comp_assoc* -*comp_assoc*.
`apply` (@*inc_trans* _ _ _ (*alpha* \quad (*g #* \quad *g*))).
`apply` *comp_inc_compat_ab_b*.
`apply` *H*.
`apply` *comp_inc_compat_ab_a*.
`apply` *H0*.
`apply` (*function_move2 H0*).
`apply` (*function_move1 H*).
`apply` *inc_rpc*.
`apply` (@*inc_trans* _ _ _ _ _ (@*dedekind* _ _ _ _ _ _)).
`apply` (@*inc_trans* _ _ _ (*f #* \quad ((*gamma* \quad *g*)\quad ((*f #*) # \quad *alpha*)))).
`apply` *comp_inc_compat_ab_a'b*.
`apply` *cap_l*.
`rewrite` *inv_invol*.

```
apply (@inc_trans _ _ _ ((f #    (gamma    ((f    alpha)    g #)))    g)).
rewrite comp_assoc.
apply comp_inc_compat_ab_ab'.
apply (@inc_trans _ _ _ _ _ (@dedekind _ _ _ _ _ _)).
apply comp_inc_compat_ab_ab'.
apply cap_l.
apply (function_move2 H0).
apply (function_move1 H).
rewrite -inc_rpc -comp_assoc.
apply H1.
Qed.
```

> **Lemma 156 (function_inv_rel1, function_inv_rel2)** *Let $f : A \to B$ be a function. Then,*
> $$f^\sharp \cdot f = id_B \sqcap f^\sharp \cdot \nabla_{AA} \cdot f = id_B \sqcap \nabla_{BA} \cdot f.$$

```
Lemma function_inv_rel1 {A B : eqType} {f : Rel A B}:
 function_r f → f #    f = Id B    ((f #        A A)    f).
Proof.
move ⇒ H.
apply inc_antisym.
apply inc_cap.
split.
apply H.
apply comp_inc_compat_ab_a'b.
apply comp_inc_compat_a_ab.
apply inc_alpha_universal.
apply (@inc_trans _ _ _ (Id B    (    B A    f))).
apply cap_inc_compat_l.
apply comp_inc_compat_ab_a'b.
apply inc_alpha_universal.
rewrite cap_comm.
apply (@inc_trans _ _ _ _ _ (@dedekind _ _ _ _ _ _)).
rewrite comp_id_l comp_id_r cap_comm inv_universal.
rewrite cap_universal cap_universal.
apply inc_refl.
Qed.

Lemma function_inv_rel2 {A B : eqType} {f : Rel A B}:
 function_r f → f #    f = Id B    (    B A    f).
Proof.
move ⇒ H.
apply inc_antisym.
```

```
rewrite (@function_inv_rel1 _ _ _ H).
apply cap_inc_compat_l.
apply comp_inc_compat_ab_a'b.
apply inc_alpha_universal.
rewrite cap_comm.
apply (@inc_trans _ _ _ _ _ (@dedekind _ _ _ _ _ _)).
rewrite comp_id_l comp_id_r cap_comm inv_universal.
rewrite cap_universal cap_universal.
apply inc_refl.
Qed.
```

Lemma 157 (function_dedekind1, function_dedekind2) *Let* $f : A \rightarrow B$ *be a function,* $\mu : C \rightarrow A$ *and* $\rho : C \rightarrow B$. *Then,*

$$(\mu \sqcap \rho \cdot f^{\sharp}) \cdot f = \mu \cdot f \sqcap \rho \wedge \rho \cdot f^{\sharp} \cdot f = \nabla_{CA} \cdot f \sqcap \rho.$$

```
Lemma function_dedekind1
 {A B C : eqType} {f : Rel A B} {mu : Rel C A} {rho : Rel C B}:
 function_r f → (mu     (rho     f #))     f = (mu     f)     rho.
Proof.
move ⇒ H.
apply inc_antisym.
apply (@inc_trans _ _ _ _ _ (comp_cap_distr_r)).
apply cap_inc_compat_l.
rewrite comp_assoc.
apply comp_inc_compat_ab_a.
apply H.
apply (@inc_trans _ _ _ _ _ (@dedekind _ _ _ _ _ _)).
apply comp_inc_compat_ab_ab'.
apply cap_l.
Qed.

Lemma function_dedekind2 {A B C : eqType} {f : Rel A B} {rho : Rel C B}:
 function_r f → (rho     f #)     f = (    C A     f)     rho.
Proof.
move ⇒ H.
move : (@function_dedekind1 _ _ _ f (    C A) rho H) ⇒ H0.
rewrite cap_comm cap_universal in H0.
apply H0.
Qed.
```

Lemma 158 (square_diagram) *In below figure,*

$$f \cdot x = g \cdot y \Leftrightarrow f^\sharp \cdot g \sqsubseteq x \cdot y^\sharp.$$

Lemma *square_diagram* $\{X\ A\ B\ D : eqType\}$
 $\{f : Rel\ X\ A\}\ \{g : Rel\ X\ B\}\ \{x : Rel\ A\ D\}\ \{y : Rel\ B\ D\}$:
 $function\_r\ f \rightarrow function\_r\ g \rightarrow function\_r\ x \rightarrow function\_r\ y \rightarrow$
 $(f \quad x = g \quad y \leftrightarrow (f\ \# \quad g) \quad (x \quad y\ \#))$.
Proof.
move $\Rightarrow$ *H H0 H1 H2*.
split; move $\Rightarrow$ *H3*.
rewrite -$(function\_move1\ H)$ -$comp\_assoc$ -$(function\_move2\ H2)\ H3$.
apply *inc_refl*.
apply *Logic.eq_sym*.
apply *function_inc*.
apply $(function\_comp\ H0\ H2)$.
apply $(function\_comp\ H\ H1)$.
rewrite $(function\_move2\ H2)\ comp\_assoc\ (function\_move1\ H)$.
apply *H3*.
Qed.

5.2 ,

Lemma 159 (surjection_comp) *Let $\alpha : A \rightarrow B$ and $\beta : B \rightarrow C$ be surjections, then $\alpha \cdot \beta$ is also a surjection.*

Lemma *surjection_comp* $\{A\ B\ C : eqType\}\ \{alpha : Rel\ A\ B\}\ \{$beta$: Rel\ B\ C\}$:
 $surjection_r\ alpha \rightarrow surjection_r\ $beta$ \rightarrow surjection_r\ (alpha \quad $beta$)$.
Proof.
rewrite $/surjection\_r$.
elim $\Rightarrow$ *H H0*.
elim $\Rightarrow$ *H1 H2*.
split.
apply $(function\_comp\ H\ H1)$.
rewrite *comp_inv*.
apply $(total\_comp\ H2\ H0)$.

Qed.

Lemma 160 (injection_comp) *Let $\alpha : A \rightharpoonup B$ and $\beta : B \rightharpoonup C$ be injections, then $\alpha \cdot \beta$ is also an injection.*

Lemma *injection_comp* $\{A\ B\ C : eqType\}$ $\{alpha : Rel\ A\ B\}$ {beta : *Rel B C*}:
 injection_r alpha → *injection_r* beta → *injection_r* (*alpha*   beta).
Proof.
rewrite /*injection_r*.
elim ⇒ *H H0*.
elim ⇒ *H1 H2*.
split.
apply (*function_comp H H1*).
rewrite *comp_inv*.
apply (*univalent_comp H2 H0*).
Qed.

Lemma 161 (bijection_comp) *Let $\alpha : A \rightharpoonup B$ and $\beta : B \rightharpoonup C$ be bijections, then $\alpha \cdot \beta$ is also a bijection.*

Lemma *bijection_comp* $\{A\ B\ C : eqType\}$ $\{alpha : Rel\ A\ B\}$ {beta : *Rel B C*}:
 bijection_r alpha → *bijection_r* beta → *bijection_r* (*alpha*   beta).
Proof.
rewrite /*bijection_r*.
elim ⇒ *H*.
elim ⇒ *H0 H1*.
elim ⇒ *H2*.
elim ⇒ *H3 H4*.
split.
apply (*function_comp H H2*).
rewrite *comp_inv*.
split.
apply (*total_comp H3 H0*).
apply (*univalent_comp H4 H1*).
Qed.

Lemma 162 (surjection_unique1) *Let $e : A \twoheadrightarrow B$ be a surjection, $f : A \rightarrow C$ be a function and $e \cdot e^{\sharp} \sqsubseteq f \cdot f^{\sharp}$, then there exists a unique function $g : B \rightarrow C$ s.t. $f = eg$.*

Lemma *surjection_unique1* $\{A\ B\ C : eqType\}$ $\{e : Rel\ A\ B\}$ $\{f : Rel\ A\ C\}$:
 surjection_r e → *function_r f* → (*e*   *e* #)   (*f*   *f* #) →
 (∃! *g : Rel B C, function_r g* ∧ *f = e*   *g*).
Proof.

```
rewrite /surjection_r/function_r/total_r/univalent_r.
elim.
elim ⇒ H H0 H1.
elim ⇒ H2 H3 H4.
∃ (e #   f).
repeat split.
rewrite comp_inv comp_assoc -(@comp_assoc _ _ _ _ f).
apply (@inc_trans _ _ _ _ _ H1).
apply comp_inc_compat_ab_ab'.
apply comp_inc_compat_b_ab.
apply H2.
rewrite comp_inv inv_invol comp_assoc -(@comp_assoc _ _ _ _ e).
apply (@inc_trans _ _ _ (f #   ((f   f #)   f))).
apply comp_inc_compat_ab_ab'.
apply (comp_inc_compat_ab_a'b H4).
rewrite comp_assoc -comp_assoc.
apply (fun H' ⇒ @inc_trans _ _ _ _ _ H' H3).
apply (comp_inc_compat_ab_a H3).
apply function_inc.
split.
apply H2.
apply H3.
split.
rewrite /total_r.
rewrite comp_inv comp_inv inv_invol.
rewrite -(@comp_assoc _ _ _ _ e) (@comp_assoc _ _ _ _ _ e) (@comp_assoc _ _ _ _ _ f)
-(@comp_assoc _ _ _ _ f).
apply (@inc_trans _ _ _ _ _ H).
apply comp_inc_compat_a_ab.
apply (@inc_trans _ _ _ _ _ H2).
apply (comp_inc_compat_a_ab H).
rewrite /univalent_r.
rewrite comp_inv comp_inv inv_invol.
rewrite (@comp_assoc _ _ _ _ _ e) -(@comp_assoc _ _ _ _ e) comp_assoc -(@comp_assoc
_ _ _ _ _ _ f).
apply (@inc_trans _ _ _ (f #   (((f   f #)   (f   f #))   f))).
apply comp_inc_compat_ab_ab'.
apply comp_inc_compat_ab_a'b.
apply comp_inc_compat.
apply H4.
apply H4.
rewrite comp_assoc (@comp_assoc _ _ _ _ _ _ f) -(@comp_assoc _ _ _ _ (f #)) -(@comp_assoc
```

```
_ _ _ _ (f #)) (@comp_assoc _ _ _ _ _ (f #)) -(@comp_assoc _ _ _ _ (f #)).
apply (fun H' ⇒ @inc_trans _ _ _ _ _ H' H3).
apply comp_inc_compat_ab_a.
apply (fun H' ⇒ @inc_trans _ _ _ _ _ H' H3).
apply (comp_inc_compat_ab_a H3).
rewrite -comp_assoc.
apply (comp_inc_compat_b_ab H).
move ⇒ g.
elim.
elim ⇒ H5 H6 H7.
replace g with (e #    (e    g)).
apply f_equal.
apply H7.
rewrite -comp_assoc.
apply inc_antisym.
apply (comp_inc_compat_ab_b H0).
rewrite inv_invol in H1.
apply (comp_inc_compat_b_ab H1).
Qed.
```

Lemma 163 (surjection_unique2) *Let $e : A \twoheadrightarrow B$ be a surjection, $f : A \to C$ be a function and $e \cdot e^\sharp = f \cdot f^\sharp$, then function $e^\sharp f$ is an injection.*

```
Lemma surjection_unique2 {A B C : eqType} {e : Rel A B} {f : Rel A C}:
  surjection_r e → function_r f → (e    e #) = (f    f #) → injection_r (e #    f).
Proof.
rewrite /surjection_r/injection_r/function_r/total_r/univalent_r.
elim.
elim ⇒ H H0 H1.
elim ⇒ H2 H3 H4.
repeat split.
rewrite comp_inv comp_assoc -(@comp_assoc _ _ _ _ f).
apply (@inc_trans _ _ _ _ _ H1).
apply comp_inc_compat_ab_ab'.
apply comp_inc_compat_b_ab.
apply H2.
rewrite comp_inv inv_invol comp_assoc -(@comp_assoc _ _ _ _ e).
rewrite H4.
rewrite comp_assoc -comp_assoc.
apply (fun H' ⇒ @inc_trans _ _ _ _ _ H' H3).
apply (comp_inc_compat_ab_a H3).
rewrite inv_invol comp_inv inv_invol comp_assoc -(@comp_assoc _ _ _ _ f).
rewrite -H4.
```

```
rewrite comp_assoc -comp_assoc.
apply (fun H' ⇒ @inc_trans _ _ _ _ _ H' H0).
apply comp_inc_compat_ab_a.
apply H0.
Qed.
```

Lemma 164 (injection_unique1) *Let $m : B \rightarrowtail A$ be an injection, $f : C \to A$ be a function and $f^\sharp \cdot f \sqsubseteq m^\sharp \cdot m$, then there exists a unique function $g : C \to B$ s.t. $f = gm$.*

```
Lemma injection_unique1 {A B C : eqType} {m : Rel B A} {f : Rel C A}:
  injection_r m → function_r f → (f #    f)    (m #     m) →
  (∃! g : Rel C B, function_r g ∧ f = g     m).
Proof.
rewrite /injection_r/function_r/total_r/univalent_r.
elim.
elim ⇒ H H0 H1.
elim ⇒ H2 H3 H4.
∃ (f     m #).
repeat split.
rewrite comp_inv inv_invol comp_assoc -(@comp_assoc _ _ _ _ _ m).
apply (@inc_trans _ _ _ (f     ((f #    f)    f #))).
rewrite comp_assoc -comp_assoc.
apply (@inc_trans _ _ _ _ _ H2).
apply (comp_inc_compat_a_ab H2).
apply comp_inc_compat_ab_ab'.
apply (comp_inc_compat_ab_a'b H4).
rewrite comp_inv comp_assoc -(@comp_assoc _ _ _ _ _ f).
apply (fun H' ⇒ @inc_trans _ _ _ _ _ H' H1).
apply comp_inc_compat_ab_ab'.
apply (comp_inc_compat_ab_b H3).
rewrite comp_assoc.
apply Logic.eq_sym.
apply function_inc.
split.
rewrite /total_r.
rewrite comp_inv comp_inv inv_invol.
apply (@inc_trans _ _ _ _ _ H2).
apply comp_inc_compat.
apply (@inc_trans _ _ _ (f     (f #    f))).
rewrite -comp_assoc.
apply (comp_inc_compat_b_ab H2).
apply (comp_inc_compat_ab_ab' H4).
apply (@inc_trans _ _ _ ((f #    f)    f #)).
```

```
rewrite comp_assoc.
apply (comp_inc_compat_a_ab H2).
apply (comp_inc_compat_ab_a'b H4).
rewrite /univalent_r.
rewrite comp_inv comp_inv inv_invol comp_assoc -(@comp_assoc _ _ _ _ _ f).
apply (fun H' ⇒ @inc_trans _ _ _ _ _ H' H0).
apply comp_inc_compat_ab_a.
apply (fun H' ⇒ @inc_trans _ _ _ _ _ H' H3).
apply (comp_inc_compat_ab_a H0).
split.
apply H2.
apply H3.
apply (comp_inc_compat_ab_a H0).
move ⇒ g.
elim.
elim ⇒ H5 H6 H7.
rewrite H7 comp_assoc.
apply inc_antisym.
rewrite inv_invol in H1.
apply (comp_inc_compat_ab_a H1).
apply (comp_inc_compat_a_ab H).
Qed.
```

Lemma 165 (injection_unique2) *Let $m : B \rightarrowtail A$ be an injection, $f : C \rightarrow A$ be a function and $f^\sharp \cdot f = m^\sharp \cdot m$, then function $f \cdot m^\sharp$ is a surjection.*

```
Lemma injection_unique2 {A B C : eqType} {m : Rel B A} {f : Rel C A}:
 injection_r m → function_r f → (f #   f) = (m #   m) → surjection_r (f   m #).
Proof.
rewrite /surjection_r/injection_r/function_r/total_r/univalent_r.
elim.
elim ⇒ H H0 H1.
elim ⇒ H2 H3 H4.
repeat split.
rewrite comp_inv inv_invol comp_assoc -(@comp_assoc _ _ _ _ _ m).
apply (@inc_trans _ _ _ (f   ((f #   f)   f #))).
rewrite comp_assoc -comp_assoc.
apply (@inc_trans _ _ _ _ _ H2).
apply (comp_inc_compat_a_ab H2).
apply comp_inc_compat_ab_ab'.
rewrite H4.
apply inc_refl.
rewrite comp_inv comp_assoc -(@comp_assoc _ _ _ _ _ f).
```

apply (fun *H'* ⇒ @*inc_trans _ _ _ _ _ H' H1*).
apply *comp_inc_compat_ab_ab'*.
apply (*comp_inc_compat_ab_b H3*).
rewrite *inv_invol comp_inv inv_invol comp_assoc* -(@*comp_assoc _ _ _ _ _ f*).
apply (@*inc_trans _ _ _ _ _ H*).
apply *comp_inc_compat_ab_ab'*.
rewrite *H4 comp_assoc*.
apply (*comp_inc_compat_a_ab H*).
Qed.

> **Lemma 166 (bijection_inv)** *Let* $\alpha : A \rightharpoonup B$, $\beta : B \rightharpoonup A$, $\alpha \cdot \beta = id_A$ *and* $\beta \cdot \alpha = id_B$,
> *then* α *and* β *are bijections and* $\beta = \alpha^{\sharp}$.

Lemma *bijection_inv* {*A B* : *eqType*} {*alpha* : *Rel A B*} {beta : *Rel B A*}:
  alpha   beta = *Id A* → beta   *alpha* = *Id B* → *bijection_r alpha* ∧ *bijection_r* beta ∧
beta = *alpha* #.
Proof.
move ⇒ *H H0*.
move : (@*id_function A*) ⇒ *H1*.
move : (@*id_function B*) ⇒ *H2*.
assert (*bijection_r alpha* ∧ *bijection_r* beta).
assert (*total_r alpha* ∧ *total_r* (*alpha* #) ∧ *total_r* beta ∧ *total_r* (beta #)).
repeat split.
apply (@*total_comp2 _ _ _ _* beta).
rewrite *H*.
apply *H1*.
apply (@*total_comp2 _ _ _ _* (beta #)).
rewrite -*comp_inv H0 inv_id*.
apply *H2*.
apply (@*total_comp2 _ _ _ _ alpha*).
rewrite *H0*.
apply *H2*.
apply (@*total_comp2 _ _ _ _* (*alpha* #)).
rewrite -*comp_inv H inv_id*.
apply *H1*.
repeat split.
apply *H3*.
apply (@*univalent_comp2 _ _ _* beta).
rewrite *H0*.
apply *H2*.
apply *H3*.
apply *H3*.
apply (@*univalent_comp2 _ _ _* (beta #)).

rewrite -*comp_inv H inv_id*.
apply *H1*.
rewrite *inv_invol*.
apply *H3*.
apply *H3*.
apply (@*univalent_comp2* _ _ _ *alpha*).
rewrite *H*.
apply *H1*.
apply *H3*.
apply *H3*.
apply (@*univalent_comp2* _ _ _ (*alpha* #)).
rewrite -*comp_inv H0 inv_id*.
apply *H2*.
rewrite *inv_invol*.
apply *H3*.
split.
apply *H3*.
split.
apply *H3*.
rewrite -(@*comp_id_r* _ _ `beta`) -(@*comp_id_l* _ _ (*alpha* #)).
rewrite -*H0 comp_assoc*.
apply f_equal.
apply *inc_antisym*.
apply *H3*.
rewrite *comp_inv_inv* -*inv_inc_move inv_id*.
apply *H3*.
Qed.

Lemma 167 (bijection_inv_corollary) *Let $\alpha : A \rightarrow B$ be a bijection, then $\alpha^\sharp$ is also a bijection.*

Lemma *bijection_inv_corollary* {*A B* : *eqType*} {*alpha* : *Rel A B*}:
 bijection_r alpha → *bijection_r* (*alpha* #).
Proof.
move : (@*bijection_inv* _ _ *alpha* (*alpha* #)) ⇒ *H*.
move ⇒ *H0*.
rewrite /*bijection_r*/*function_r*/*total_r*/*univalent_r* in *H0*.
rewrite *inv_invol* in *H0*.
apply *H*.
apply *inc_antisym*.
apply *H0*.
apply *H0*.

```
apply inc_antisym.
apply H0.
apply H0.
Qed.
```

5.3

Lemma 168 (rationality_corollary1) *Let $u : A \to A$ and $u \sqsubseteq id_A$. Then,*

$$\exists R, \exists j : R \rightarrowtail A, u = j^\sharp \cdot j.$$

```
Lemma rationality_corollary1 {A : eqType} {u : Rel A A}:
  u    Id A → ∃ (R : eqType)(j : Rel R A), injection_r j ∧ u = j #    j.
Proof.
move : (rationality _ _ u).
elim ⇒ R.
elim ⇒ f.
elim ⇒ g.
elim ⇒ H.
elim ⇒ H0.
elim ⇒ H1 H2 H3.
∃ R.
∃ f.
assert (g = f).
apply (function_inc H0 H).
apply (@inc_trans _ _ _ ((f    f #)    g)).
apply comp_inc_compat_b_ab.
apply H.
rewrite comp_assoc -H1.
apply (comp_inc_compat_ab_a H3).
rewrite H4 in H1.
rewrite H4 cap_idem in H2.
split.
split.
apply H.
rewrite /univalent_r.
rewrite inv_invol H2.
apply inc_refl.
apply H1.
Qed.
```

Lemma 169 (rationality_corollary2) *Let $f : A \to B$ be a function. Then,*

$$\exists e : A \twoheadrightarrow R, \exists m : R \rightarrowtail B, f = e \cdot m.$$

Lemma *rationality_corollary2* $\{A\ B : eqType\}$ $\{f : Rel\ A\ B\}$:
 function_r f $\to \exists\ (R : eqType)(e : Rel\ A\ R)(m : Rel\ R\ B), surjection\_r\ e \land injection\_r$
m.
Proof.
elim $\Rightarrow$ *H H0*.
move : $(@rationality\_corollary1$ _ $(f\ \#\quad f)$ *H0*).
elim $\Rightarrow$ *R*.
elim $\Rightarrow$ *m*.
elim $\Rightarrow$ *H1 H2*.
$\exists$ *R*.
$\exists$ $(f\quad m\ \#)$.
$\exists$ *m*.
split.
apply $(injection\_unique2\ H1\ (conj\ H\ H0)\ H2)$.
apply *H1*.
Qed.

Lemma 170 (axiom_of_subobjects) *Let $u : A \to A$ and $u \sqsubseteq id_A$. Then,*

$$\exists R, \exists j : R \to A, j^{\sharp} \cdot j = u \land j \cdot j^{\sharp} = id_R.$$

Lemma *axiom_of_subobjects* $\{A : eqType\}$ $\{u : Rel\ A\ A\}$:
 u $\quad$ *Id A* $\to \exists\ (R : eqType)(j : Rel\ R\ A), j\ \#\quad j = u \land j\quad j\ \# = Id\ R$.
Proof.
move $\Rightarrow$ *H*.
elim $(rationality\_corollary1\ H) \Rightarrow R$.
elim $\Rightarrow$ *j H0*.
$\exists$ *R*.
$\exists$ *j*.
split.
apply *Logic.eq_sym*.
apply *H0*.
apply *inc_antisym*.
replace $(j\quad j\ \#)$ with $((j\ \#)\ \#\quad j\ \#)$.
apply *H0*.
by $[$rewrite *inv_invol*$]$.
apply *H0*.
Qed.

End *main*.

Chapter 6

Library Tactics

Require Import *Basic_Notations*.
Require Import *Basic_Lemmas*.
Require Import *Relation_Properties*.

Module *main* (*def* : *Relation*).
Import *def*.
Module *Basic_Lemmas* := *Basic_Lemmas.main def*.
Module *Relation_Properties* := *Relation_Properties.main def*.
Import *Basic_Lemmas Relation_Properties*.

6.1   Tactic

$$
\alpha = \beta \qquad\qquad , \qquad\qquad \alpha \sqsubseteq \beta \wedge \beta \sqsubseteq \alpha \qquad\qquad .
$$

Lemma *inc_antisym_eq* {*A B* : *eqType*} {*alpha* beta : *Rel A B*}:
 alpha = beta ↔ *alpha*   beta ∧ beta   *alpha*.
Proof.
split; move ⇒ *H*.
rewrite *H*.
split; apply *inc_refl*.
apply *inc_antisym*; apply *H*.
Qed.

6.2 Tactic

- Rel_simpl_rewrite ...

- Rel_simpl_intro ...   , inc_antisym_eq

- Rel_simpl_comp_inc ... comp_inc_compat

- Rel_simpl ...   ,   3 tactics

- Rel_trans ... Rel_simpl   inc_trans   ,

Ltac *Rel_simpl_rewrite* :=
 rewrite /*bijection_r*/*surjection_r*/*injection_r*;
 rewrite /*function_r*/*total_r*/*univalent_r*.
Ltac *Rel_simpl_intro* :=
 Rel_simpl_rewrite;
 repeat match goal with
         | [_ : _ ⊢ (_ ∧ _) → _] ⇒ elim
         | [_ : _ ⊢ _ → _] ⇒ intro
         | [_ : _ ⊢ _ ∧ _] ⇒ split
         | [_ : _ ⊢ _ ↔ _] ⇒ split
         | [_ : _ ⊢ _ = _] ⇒ rewrite *inc_antisym_eq*
         | [H : _ = _ ⊢ _] ⇒ rewrite *inc_antisym_eq* in H
       end.
Ltac *Rel_simpl_comp_inc* :=
 repeat match goal with
         | [H : ?g   Id _ ⊢ (?f   ?g)   ?f] ⇒ apply (*comp_inc_compat_ab_a H*)
         | [H : ?g   Id _ ⊢ (?g   ?f)   ?f] ⇒ apply (*comp_inc_compat_ab_b H*)
         | [H : Id _   ?g ⊢ ?f   (?f   ?g)] ⇒ apply (*comp_inc_compat_a_ab H*)
         | [H : Id _   ?g ⊢ ?f   (?g   ?f)] ⇒ apply (*comp_inc_compat_b_ab H*)
         | [_ : _ ⊢ _] ⇒ repeat rewrite -*comp_assoc*; (apply *comp_inc_compat_ab_a'b*
|| apply *comp_inc_compat_b_ab* || apply *comp_inc_compat_ab_b*)
         | [_ : _ ⊢ _] ⇒ repeat rewrite *comp_assoc*; (apply *comp_inc_compat_ab_ab'*
|| apply *comp_inc_compat_a_ab* || apply *comp_inc_compat_ab_a*)
         | [_ : _ ⊢ (_   _)   (_   _)] ⇒ apply *comp_inc_compat*
       end.
Ltac *Rel_simpl* :=
 Rel_simpl_intro;
 repeat match goal with
         | [f : *Rel* _ _, H : _   _ ⊢ _] ⇒ rewrite (@*inv_invol* _ _ f) in H
       end;

```
repeat match goal with
```
        | [_ : _ ⊢ ?f   ?f] ⇒ apply *inc_refl*
        | [H : ?P ⊢ ?P] ⇒ apply H
        | [H : ?f   ?g, H0 : ?g   ?h ⊢ ?f   ?h] ⇒ apply (@*inc_trans* _ _ _ _ _ H

H0)

        | [_ : _ ⊢ (_ #)   (_ #)] ⇒ apply *inc_inv*
        | [A : eqType, B : eqType, C : eqType ⊢ _] ⇒ rewrite (@*comp_inv* A B C)
        | [f : Rel _ _ ⊢ _] ⇒ rewrite (@*inv_invol* _ _ f)
        | [H : (Id _)   _ ⊢ (Id _)   _] ⇒ apply (@*inc_trans* _ _ _ _ _ H)
        | [H : _   (Id _) ⊢ _   (Id _)] ⇒ apply (fun H' ⇒ (@*inc_trans* _ _ _ _ _ H'

H))

        | [_ : _ ⊢ _] ⇒ *Rel_simpl_comp_inc*
```
    end.
```
Ltac *Rel_trans* f :=
 apply (@*inc_trans* _ _ _ f); *Rel_simpl*.

6.3

⎛‾‾⎞

`Functions_Mappings.v`   ,   tactic   .

⎝_____⎠

Lemma *total_comp* {A B C : eqType} {*alpha* : Rel A B} {beta : Rel B C}:
 total_r alpha → *total_r* beta → *total_r* (*alpha*   beta).
Proof.
Rel_simpl.
Qed.

Lemma *univalent_comp* {A B C : eqType} {*alpha* : Rel A B} {beta : Rel B C}:
 univalent_r alpha → *univalent_r* beta → *univalent_r* (*alpha*   beta).
Proof.
Rel_simpl.
Qed.

Lemma *function_comp* {A B C : eqType} {*alpha* : Rel A B} {beta : Rel B C}:
 function_r alpha → *function_r* beta → *function_r* (*alpha*   beta).
Proof.
Rel_simpl.
Qed.

Lemma *univalent_comp2* {A B C : eqType} {*alpha* : Rel A B} {beta : Rel B C}:
 univalent_r (*alpha*   beta) → *total_r* (*alpha* #) → *univalent_r* beta.
Proof.
Rel_simpl.

114

Qed.

Lemma *total_inc* {*A B* : *eqType*} {*alpha* beta : *Rel A B*}:
 total_r alpha → *alpha*   beta → *total_r* beta.
Proof.
Rel_simpl.
Qed.

Lemma *univalent_inc* {*A B* : *eqType*} {*alpha* beta : *Rel A B*}:
 univalent_r alpha → beta   *alpha* → *univalent_r* beta.
Proof.
Rel_simpl.
Qed.

Lemma *function_inc* {*A B* : *eqType*} {*alpha* beta : *Rel A B*}:
 function_r alpha → *function_r* beta → *alpha*   beta → *alpha* = beta.
Proof.
Rel_simpl.
Rel_trans ((*alpha*   *alpha* #)   beta).
Qed.

Lemma *function_move1* {*A B C* : *eqType*} {*alpha* : *Rel A B*} {beta : *Rel B C*} {*gamma* :
Rel A C}:
 function_r alpha → (*gamma*   (*alpha*   beta) ↔ (*alpha* #   *gamma*)   beta).
Proof.
Rel_simpl.
Rel_trans ((*alpha* #   *alpha*)   beta).
Rel_trans ((*alpha*   *alpha* #)   *gamma*).
Qed.

Lemma *function_move2* {*A B C* : *eqType*} {*alpha* : *Rel A B*} {beta : *Rel B C*} {*gamma* :
Rel A C}:
 function_r beta → ((*alpha*   beta)   *gamma* ↔ *alpha*   (*gamma*   beta #)).
Proof.
Rel_simpl.
Rel_trans ((*alpha*   beta)   beta #).
Rel_trans ((*gamma*   beta #)   beta).
Qed.

Lemma *surjection_comp* {*A B C* : *eqType*} {*alpha* : *Rel A B*} {beta : *Rel B C*}:
 surjection_r alpha → *surjection_r* beta → *surjection_r* (*alpha*   beta).
Proof.
Rel_simpl.
Qed.

Lemma *injection_comp* {*A B C* : *eqType*} {*alpha* : *Rel A B*} {beta : *Rel B C*}:
 injection_r alpha → *injection_r* beta → *injection_r* (*alpha*   beta).
Proof.

Rel_simpl.
Qed.

Lemma *bijection_comp* {*A B C : eqType*} {*alpha : Rel A B*} {beta *: Rel B C*}:
  bijection_r alpha → *bijection_r* beta → *bijection_r* (*alpha*   beta).
Proof.
Rel_simpl.
Qed.

Lemma *bijection_inv_corollary* {*A B : eqType*} {*alpha : Rel A B*}:
  bijection_r alpha → *bijection_r* (*alpha #*).
Proof.
Rel_simpl.
Qed.

End *main.*

116

Chapter 7

Library Dedekind

Require Import *Basic_Notations_Set*.
Require Import *Basic_Lemmas*.
Require Import *Relation_Properties*.
Require Import *Functions_Mappings*.

Module *main* (*def* : *Relation*).
Import *def*.
Module *Basic_Lemmas* := *Basic_Lemmas.main def*.
Module *Relation_Properties* := *Relation_Properties.main def*.
Module *Functions_Mappings* := *Functions_Mappings.main def*.
Import *Basic_Lemmas Relation_Properties Functions_Mappings*.

7.1 Dedekind formula

Lemma 171 (dedekind1) *Let* $\alpha : A \rightharpoonup B$, $\beta : B \rightharpoonup C$ *and* $\gamma : A \rightharpoonup C$. *Then*

$$\alpha \cdot \beta \sqcap \gamma \sqsubseteq \alpha \cdot (\beta \sqcap \alpha^{\sharp} \cdot \gamma).$$

Lemma *dedekind1*
 {*A B C* : *eqType*} {*alpha* : *Rel A B*} {beta : *Rel B C*} {*gamma* : *Rel A C*}:
 ((*alpha* beta) gamma) (*alpha* (beta (*alpha* # gamma))).
Proof.
apply (@*inc_trans* _ _ _ _ _ (@*dedekind* _ _ _ _ _ _)).
apply *comp_inc_compat_ab_a'b*.
apply *cap_l*.
Qed.

117

Lemma 172 (dedekind2) *Let $\alpha : A \rightharpoonup B$, $\beta : B \rightharpoonup C$ and $\gamma : A \rightharpoonup C$. Then*

$$\alpha \cdot \beta \sqcap \gamma \sqsubseteq (\alpha \sqcap \gamma \cdot \beta^{\sharp}) \cdot \beta.$$

Lemma *dedekind2*
  {*A B C : eqType*} {*alpha : Rel A B*} {beta : *Rel B C*} {*gamma : Rel A C*}:
  ((*alpha* beta) *gamma*) ((*alpha* (*gamma* beta #)) beta).
Proof.
apply (@*inc_trans* _ _ _ _ _ (@*dedekind* _ _ _ _ _ _)).
apply *comp_inc_compat_ab_ab'*.
apply *cap_l*.
Qed.

Lemma 173 (relation_rel_inv_rel) *Let $\alpha : A \rightharpoonup B$. Then*

$$\alpha \sqsubseteq \alpha \cdot \alpha^{\sharp} \cdot \alpha.$$

Lemma *relation_rel_inv_rel* {*A B : eqType*} {*alpha : Rel A B*}:
  alpha ((*alpha* *alpha* #) *alpha*).
Proof.
move : (@*dedekind1* _ _ _ *alpha* (*Id B*) *alpha*) ⇒ *H*.
rewrite *comp_id_r* *cap_idem* in *H*.
apply (@*inc_trans* _ _ _ _ _ *H*).
rewrite *comp_assoc*.
apply *comp_inc_compat_ab_ab'*.
apply *cap_r*.
Qed.

7.2 Dedekind formula

Lemma 174 (dedekind_universal1) *Let $\alpha : B \rightharpoonup C$. Then*

$$\nabla_{AC} \cdot \alpha^{\sharp} \cdot \alpha = \nabla_{AB} \cdot \alpha.$$

Lemma *dedekind_universal1* {*A B C : eqType*} {*alpha : Rel B C*}:
  (*A C*   *alpha* #)   *alpha* =   *A B*   *alpha*.
Proof.
apply *inc_antisym*.
apply *comp_inc_compat_ab_a'b*.
apply *inc_alpha_universal*.
apply (@*inc_trans* _ _ _ (*A B*   ((*alpha*   *alpha* #)   *alpha*))).

```
apply comp_inc_compat_ab_ab'.
apply relation_rel_inv_rel.
rewrite -comp_assoc -comp_assoc.
apply comp_inc_compat_ab_a'b.
apply comp_inc_compat_ab_a'b.
apply inc_alpha_universal.
Qed.
```

Lemma 175 (dedekind_universal2a, dedekind_universal2b, dedekind_universal2c) *Let $\alpha : A \to B$ and $\beta : C \to B$. Then*

$$\nabla_{IC} \cdot \beta \sqsubseteq \nabla_{IA} \cdot \alpha \Leftrightarrow \nabla_{CC} \cdot \beta \sqsubseteq \nabla_{CA} \cdot \alpha \Leftrightarrow \beta \sqsubseteq \beta \cdot \alpha^{\sharp} \cdot \alpha.$$

```
Lemma dedekind_universal2a {A B C : eqType} {alpha : Rel A B} {beta : Rel C B}:
 (   i C    beta) (   i A    alpha) → (   C C    beta) (   C A    alpha).
Proof.
move ⇒ H.
rewrite -unit_universal -(@lemma_for_tarski2  C  A).
rewrite comp_assoc comp_assoc.
apply (comp_inc_compat_ab_ab'  H).
Qed.

Lemma dedekind_universal2b {A B C : eqType} {alpha : Rel A B} {beta : Rel C B}:
 (   C C    beta) (   C A    alpha) → beta    ((beta    alpha #)    alpha).
Proof.
move ⇒ H.
apply (@inc_trans _ _ _ (beta    (   C C    beta))).
apply inc_cap.
split.
apply inc_refl.
apply comp_inc_compat_b_ab.
apply inc_alpha_universal.
apply (@inc_trans _ _ _ (beta    (   C A    alpha))).
apply (cap_inc_compat_l H).
rewrite cap_comm.
apply (@inc_trans _ _ _ _ _ (dedekind2)).
apply comp_inc_compat_ab_a'b.
apply cap_r.
Qed.

Lemma dedekind_universal2c {A B C : eqType} {alpha : Rel A B} {beta : Rel C B}:
 beta    ((beta    alpha #)    alpha) → (   i C    beta) (   i A    alpha).
Proof.
move ⇒ H.
```

```
apply (@inc_trans _ _ _ (   i C    ((beta    alpha #)    alpha))).
apply (comp_inc_compat_ab_ab' H).
rewrite -comp_assoc.
apply comp_inc_compat_ab_a'b.
apply inc_alpha_universal.
Qed.
```

Lemma 176 (dedekind_universal3a, dedekind_universal3b) *Let* $\alpha : A \to B$ *and* $\beta : A \to C$. *Then*

$$\beta \cdot \nabla_{CI} \sqsubseteq \alpha \cdot \nabla_{BI} \Leftrightarrow \beta \cdot \nabla_{CC} \sqsubseteq \alpha \cdot \nabla_{BC} \Leftrightarrow \beta \sqsubseteq \alpha \cdot \alpha^{\sharp} \cdot \beta.$$

```
Lemma dedekind_universal3a {A B C : eqType} {alpha : Rel A B} {beta : Rel A C}:
 (beta    C i)   (alpha    B i) ↔ (beta    C C)   (alpha    B C).
Proof.
split; move ⇒ H.
apply inv_inc_invol.
rewrite comp_inv comp_inv inv_universal inv_universal.
apply dedekind_universal2a.
apply inv_inc_invol.
rewrite comp_inv comp_inv inv_invol inv_invol inv_universal inv_universal.
apply H.
apply inv_inc_invol.
rewrite comp_inv comp_inv inv_universal inv_universal.
apply dedekind_universal2c.
apply dedekind_universal2b.
apply inv_inc_invol.
rewrite comp_inv comp_inv inv_invol inv_invol inv_universal inv_universal.
apply H.
Qed.

Lemma dedekind_universal3b {A B C : eqType} {alpha : Rel A B} {beta : Rel A C}:
 (beta    C i)   (alpha    B i) ↔ beta    ((alpha    alpha #)    beta).
Proof.
split; move ⇒ H.
apply inv_inc_invol.
rewrite comp_inv comp_inv -comp_assoc.
apply dedekind_universal2b.
apply dedekind_universal2a.
apply inv_inc_invol.
rewrite comp_inv comp_inv inv_invol inv_invol inv_universal inv_universal.
apply H.
apply inv_inc_invol.
```

```
rewrite comp_inv comp_inv inv_universal inv_universal.
apply dedekind_universal2c.
rewrite -comp_inv -comp_inv -comp_assoc.
apply inc_inv.
apply H.
Qed.
```

Lemma 177 (universal_total) *Let* $\alpha : A \to B$. *Then*

$$\alpha \cdot \nabla_{BI} = \nabla_{AI} \Leftrightarrow \text{``}\alpha \text{ is total''}.$$

```
Lemma universal_total {A B : eqType} {alpha : Rel A B}:
  alpha       B i =    A i ↔ total_r alpha.
Proof.
move : (@dedekind_universal3b _ _ _ alpha (Id A)) ⇒ H.
rewrite comp_id_l comp_id_r in H.
rewrite /total_r.
rewrite -H.
split; move ⇒ H0.
rewrite H0.
apply inc_refl.
apply inc_antisym.
apply inc_alpha_universal.
apply H0.
Qed.
```

7.3 Dedekind formula

Lemma 178 (dedekind_id1) *Let* $\alpha : A \to A$. *Then*

$$\alpha \sqsubseteq id_A \Rightarrow \alpha^\sharp = \alpha.$$

```
Lemma dedekind_id1 {A : eqType} {alpha : Rel A A}: alpha    Id A → alpha # = alpha.
Proof.
move ⇒ H.
assert (alpha #    alpha).
move : (@dedekind1 _ _ _ (alpha #) (Id A) (Id A)) ⇒ H0.
rewrite comp_id_r comp_id_r inv_invol in H0.
replace (alpha #    Id A) with (alpha #) in H0.
replace (Id A    alpha) with alpha in H0.
apply (@inc_trans _ _ _ (alpha #    alpha)).
apply H0.
```

```
apply comp_inc_compat_ab_b.
rewrite -inv_inc_move.
rewrite inv_id.
apply H.
rewrite cap_comm.
apply inc_def1.
apply H.
apply inc_def1.
rewrite -inv_inc_move.
rewrite inv_id.
apply H.
apply inc_antisym.
apply H0.
apply inv_inc_move.
apply H0.
Qed.
```

Lemma 179 (dedekind_id2) *Let $\alpha : A \rightarrow A$. Then*

$$\alpha \sqsubseteq id_A \Rightarrow \alpha \cdot \alpha = \alpha.$$

```
Lemma dedekind_id2 {A : eqType} {alpha : Rel A A}:
 alpha     Id A → alpha     alpha = alpha.
Proof.
move ⇒ H.
apply inc_antisym.
apply (comp_inc_compat_ab_a H).
move : (dedekind_id1 H) ⇒ H0.
apply (@inc_trans _ _ _ ((alpha     Id A)     Id A)).
rewrite comp_id_r.
apply inc_cap.
split.
apply inc_refl.
apply H.
apply (@inc_trans _ _ _ _ _ (dedekind1)).
apply comp_inc_compat_ab_ab'.
rewrite H0 comp_id_r.
apply cap_r.
Qed.
```

Lemma 180 (dedekind_id3) *Let $\alpha, \beta : A \to A$. Then*

$$\alpha \sqsubseteq id_A \wedge \beta \sqsubseteq id_A \Rightarrow \alpha \cdot \beta = \alpha \sqcap \beta.$$

```
Lemma dedekind_id3 {A : eqType} {alpha beta : Rel A A}:
 alpha      Id A → beta      Id A → alpha      beta = alpha      beta.
Proof.
move ⇒ H H0.
apply inc_antisym.
apply inc_cap.
split.
apply (comp_inc_compat_ab_a H0).
apply (comp_inc_compat_ab_b H).
replace (alpha      beta) with ((alpha      beta)      (alpha      beta)).
apply comp_inc_compat.
apply cap_l.
apply cap_r.
apply dedekind_id2.
apply (fun H' ⇒ @inc_trans _ _ _ _ _ H' H).
apply cap_l.
Qed.
```

Lemma 181 (dedekind_id4) *Let $\alpha, \beta : A \to A$. Then*

$$\alpha \sqsubseteq id_A \wedge \beta \sqsubseteq id_A \Rightarrow (\alpha \rhd \beta) \sqcap id_A = (\alpha \Rightarrow \beta) \sqcap id_A.$$

```
Lemma dedekind_id4 {A : eqType} {alpha beta : Rel A A}:
 alpha      Id A → beta      Id A → (alpha      beta)      Id A = (alpha » beta)      Id A.
Proof.
move ⇒ H H0.
apply inc_lower.
move ⇒ gamma.
rewrite inc_cap inc_cap.
split; elim ⇒ H1 H2.
split.
rewrite inc_rpc cap_comm.
rewrite -(@dedekind_id3 _ _ _ H H2).
rewrite -(@dedekind_id1 _ _ H).
apply inc_residual.
apply H1.
apply H2.
split.
```

```
rewrite inc_residual (@dedekind_id1 _ _ H) (@dedekind_id3 _ _ _ H H2).
rewrite cap_comm -inc_rpc.
apply H1.
apply H2.
Qed.

End main.
```

Chapter 8

Library Conjugate

Require Import *Basic_Notations*.
Require Import *Basic_Lemmas*.
Require Import *Relation_Properties*.
Require Import *Functions_Mappings*.
Require Import *Dedekind*.

Module *main* (*def* : *Relation*).
Import *def*.
Module *Basic_Lemmas* := *Basic_Lemmas.main def*.
Module *Relation_Properties* := *Relation_Properties.main def*.
Module *Functions_Mappings* := *Functions_Mappings.main def*.
Module *Dedekind* := *Dedekind.main def*.
Import *Basic_Lemmas Relation_Properties Functions_Mappings Dedekind*.

8.1

$$
\begin{array}{l}
\quad P \qquad\qquad \alpha : A \rightharpoonup B \qquad Q \qquad\qquad \beta : A' \rightharpoonup B' \qquad \alpha = \\
\phi(\beta), \beta = \psi(\alpha) \qquad ,1 \quad 1\ (\qquad) \qquad\qquad\qquad , \\[2mm]
\qquad\qquad\qquad \dfrac{\alpha : A \rightharpoonup B\ \{P\}}{\beta : A' \rightharpoonup B'\ \{Q\}}\ \dfrac{\alpha = \phi(\beta)}{\beta = \psi(\alpha)} \\[3mm]
\qquad\qquad . \qquad , \mathrm{Coq} \qquad\qquad\qquad\qquad\qquad .
\end{array}
$$

Definition *conjugate*
 (*A B C D* : *eqType*) (*P* : *Rel A B* → Prop) (*Q* : *Rel C D* → Prop)
 (*phi* : *Rel C D* → *Rel A B*) (*psi* : *Rel A B* → *Rel C D*):=
 ($\forall$ *alpha* : *Rel A B*, *P alpha* → *Q* (*psi alpha*) ∧ *phi* (*psi alpha*) = *alpha*)
 ∧ ($\forall$ beta : *Rel C D*, *Q* beta → *P* (*phi* beta) ∧ *psi* (*phi* beta) = beta).

125

$$\boxed{\quad ,\qquad\qquad\qquad P\qquad Q\qquad\qquad\qquad ,\qquad \texttt{True\_r}\qquad .}$$

Definition *True_r* $\{A\ B : eqType\} :=$ `fun` $\_$: *Rel A B* $\Rightarrow$ *True*.

8.2

Lemma 182 (inv_conjugate) *Inverse relation ($^\sharp$) makes conjugate. That is,*

$$\frac{\alpha : A \rightharpoonup B \quad \alpha = \beta^\sharp}{\beta : B \rightharpoonup A \quad \beta = \alpha^\sharp}.$$

Lemma *inv_conjugate* $\{A\ B : eqType\}$:
 conjugate A B B A True_r True_r (@*inverse* $\_$ $\_$) (@*inverse* $\_$ $\_$).
Proof.
`split.`
`move` $\Rightarrow$ *alpha H*.
`split.`
`by` $\|.$
`apply` *inv_invol*.
`move` $\Rightarrow$ `beta` *H*.
`split.`
`by` $\|.$
`apply` *inv_invol*.
Qed.

Lemma 183 (injection_conjugate) *Let $j : C \rightarrowtail B$ be an injection. Then,*

$$\frac{f : A \rightarrow B\ \{f^\sharp \cdot f \sqsubseteq j^\sharp \cdot j\} \quad f = h \cdot j}{h : A \rightarrow C \qquad\qquad h = f \cdot j^\sharp}$$

Lemma *injection_conjugate* $\{A\ B\ C : eqType\}$ $\{j : Rel\ C\ B\}$:
 injection_r j $\rightarrow$
 conjugate A B A C (`fun` *f* : *Rel A B* $\Rightarrow$ ((*f* # *f*) (*j* # *j*)) $\wedge$ *function_r f*)
 (`fun` *h* : *Rel A C* $\Rightarrow$ *function_r h*) (`fun` *h* : *Rel A C* $\Rightarrow$ *h* *j*)
 (`fun` *f* : *Rel A B* $\Rightarrow$ *f* *j* #).
Proof.
`elim.`
`elim` $\Rightarrow$ *H H0 H1*.
`split.`
`move` $\Rightarrow$ *alpha*.

```
elim ⇒ H2.
elim ⇒ H3 H4.
assert (function_r (alpha    j #)).
split.
apply (@inc_trans _ _ _ _ _ H3).
rewrite comp_inv inv_invol comp_assoc -(@comp_assoc _ _ _ _ _ j).
apply (@inc_trans _ _ _ (alpha    ((alpha #    alpha)    alpha #))).
rewrite comp_assoc -comp_assoc.
apply (comp_inc_compat_a_ab H3).
apply comp_inc_compat_ab_ab'.
apply (comp_inc_compat_ab_a'b H2).
apply (fun H' ⇒ @inc_trans _ _ _ _ _ H' H1).
rewrite comp_inv inv_invol comp_assoc -(@comp_assoc _ _ _ _ _ alpha).
apply comp_inc_compat_ab_ab'.
apply comp_inc_compat_ab_b.
apply (@inc_trans _ _ _ _ _ H2).
apply H0.
split.
apply H5.
apply function_inc.
apply function_comp.
apply H5.
split.
apply H.
apply H0.
split.
apply H3.
apply H4.
rewrite comp_assoc.
apply comp_inc_compat_ab_a.
apply H0.
move ⇒ beta.
elim ⇒ H2 H3.
assert (function_r (beta    j)).
split.
apply (@inc_trans _ _ _ _ _ H2).
rewrite comp_inv comp_assoc -(@comp_assoc _ _ _ _ j).
apply comp_inc_compat_ab_ab'.
apply (comp_inc_compat_b_ab H).
apply (fun H' ⇒ @inc_trans _ _ _ _ _ H' H0).
rewrite comp_inv comp_assoc -(@comp_assoc _ _ _ _ _ beta).
apply comp_inc_compat_ab_ab'.
```

```
apply (comp_inc_compat_ab_b H3).
split.
split.
rewrite comp_inv comp_assoc -(@comp_assoc _ _ _ _ _ beta).
apply comp_inc_compat_ab_ab'.
apply (comp_inc_compat_ab_b H3).
apply H4.
rewrite comp_assoc.
replace (j       j #) with (Id C).
apply comp_id_r.
apply inc_antisym.
apply H.
rewrite /univalent_r in H1.
rewrite inv_invol in H1.
apply H1.
Qed.
```

> **Lemma 184 (injection_conjugate_corollary1, injection_conjugate_corollary2)**
> Let $j : C \rightarrowtail B$ be an injection and $f : A \to B$ be a function. Then,
>
> $$f^\sharp \cdot f \sqsubseteq j^\sharp \cdot j \Leftrightarrow (\exists! h : A \to C, f = h \cdot j) \Leftrightarrow (\exists h' : A \to C, f \sqsubseteq h' \cdot j).$$

```
Lemma injection_conjugate_corollary1 {A B C : eqType} {f : Rel A B} {j : Rel C B}:
 injection_r j → function_r f →
 ((f #     f)     (j #     j) ↔ ∃! h : Rel A C, function_r h ∧ f = h     j).
Proof.
move ⇒ H H0.
move : (@injection_conjugate A _ _ _ H).
elim ⇒ H1 H2.
split; move ⇒ H3.
∃ (f     j #).
split.
move : (H1 f (conj H3 H0)).
elim ⇒ H4 H5.
split.
apply H4.
by [rewrite H5].
move ⇒ h.
elim ⇒ H4 H5.
rewrite H5 comp_assoc.
replace (j     j #) with (Id C).
apply comp_id_r.
rewrite /injection_r/function_r/univalent_r in H.
```

```
rewrite inv_invol in H.
apply inc_antisym.
apply H.
apply H.
elim H3 ⇒ h.
elim.
elim ⇒ H4 H5 H6.
rewrite H5 comp_inv comp_assoc -(@comp_assoc _ _ _ _ _ h).
apply comp_inc_compat_ab_ab'.
apply comp_inc_compat_ab_b.
apply H4.
Qed.
```

Lemma *injection_conjugate_corollary2* $\{A\ B\ C : eqType\}\ \{f : Rel\ A\ B\}\ \{j : Rel\ C\ B\}$:
  injection_r j → *function_r f* →
  $((f\ \#\quad f)\quad (j\ \#\quad j)) \leftrightarrow \exists\ h' : Rel\ A\ C, f\quad (h'\quad j))$.

```
Proof.
move ⇒ H  H0.
split; move ⇒ H1.
apply (injection_conjugate_corollary1 H H0) in H1.
elim H1 ⇒ h.
elim.
elim ⇒ H2 H3 H4.
∃ h.
rewrite H3.
apply inc_refl.
elim H1 ⇒ h' H2.
replace (f #    f) with (f #    (f    (h'    j))).
apply (@inc_trans _ _ _ ((f #    f)    (j #    j))).
rewrite comp_assoc cap_comm -(@comp_assoc _ _ _ _ f).
apply comp_inc_compat_ab_ab'.
apply (@inc_trans _ _ _ _ _ (dedekind2)).
apply comp_inc_compat_ab_a'b.
apply cap_r.
apply comp_inc_compat_ab_b.
apply H0.
apply f_equal.
apply inc_def1 in H2.
by [rewrite -H2].
Qed.
```

> **Lemma 185 (surjection_conjugate)** *Let $e : A \twoheadrightarrow C$ be a surjection. Then,*
>
> $$\frac{f : A \to B \ \{e \cdot e^\sharp \sqsubseteq f \cdot f^\sharp\}}{h : C \to B} \quad \frac{f = e \cdot h}{h = e^\sharp \cdot f}$$

Lemma *surjection_conjugate* $\{A\ B\ C : eqType\}$ $\{e : Rel\ A\ C\}$:
 surjection_r e $\to$
 conjugate A B C B (**fun** *f* : *Rel A B* $\Rightarrow$ (($e \quad e \#$) $\quad$ ($f \quad f \#$)) $\wedge$ *function_r f*)
 (**fun** *h* : *Rel C B* $\Rightarrow$ *function_r h*) (**fun** *h* : *Rel C B* $\Rightarrow$ $e \quad h$) (**fun** *f* : *Rel A B* $\Rightarrow$ $e \#$
 f).
Proof.
elim.
elim $\Rightarrow$ *H H0 H1*.
split.
move $\Rightarrow$ *alpha*.
elim $\Rightarrow$ *H2*.
elim $\Rightarrow$ *H3 H4*.
assert (*function_r* ($e \# \quad alpha$)).
split.
apply (@*inc_trans* _ _ _ _ _ *H1*).
rewrite *comp_inv inv_invol comp_assoc* -(@*comp_assoc* _ _ _ _ *alpha*).
apply *comp_inc_compat_ab_ab'*.
apply (*comp_inc_compat_b_ab H3*).
apply (**fun** *H'* $\Rightarrow$ @*inc_trans* _ _ _ _ _ *H' H4*).
rewrite *comp_inv inv_invol comp_assoc* -(@*comp_assoc* _ _ _ _ *e*).
apply (@*inc_trans* _ _ _ (*alpha* # $\quad$ ((*alpha* $\quad$ *alpha* #) $\quad$ *alpha*))).
apply *comp_inc_compat_ab_ab'*.
apply (*comp_inc_compat_ab_a'b H2*).
rewrite *comp_assoc* -*comp_assoc*.
apply (*comp_inc_compat_ab_a H4*).
split.
apply *H5*.
apply *Logic.eq_sym*.
apply *function_inc*.
split.
apply *H3*.
apply *H4*.
apply *function_comp*.
split.
apply *H*.
apply *H0*.
apply *H5*.

```
rewrite -comp_assoc.
apply comp_inc_compat_b_ab.
apply H.
move ⇒ beta.
elim ⇒ H2 H3.
assert (function_r (e    beta)).
split.
apply (@inc_trans _ _ _ _ _ H).
rewrite comp_inv comp_assoc -(@comp_assoc _ _ _ _ beta).
apply comp_inc_compat_ab_ab'.
apply (comp_inc_compat_b_ab H2).
apply (fun H' ⇒ @inc_trans _ _ _ _ _ H' H3).
rewrite comp_inv comp_assoc -(@comp_assoc _ _ _ _ _ e).
apply comp_inc_compat_ab_ab'.
apply (comp_inc_compat_ab_b H0).
split.
split.
rewrite comp_inv comp_assoc -(@comp_assoc _ _ _ _ beta).
apply comp_inc_compat_ab_ab'.
apply (comp_inc_compat_b_ab H2).
apply H4.
rewrite -comp_assoc.
replace (e #    e) with (Id C).
apply comp_id_l.
apply inc_antisym.
rewrite /total_r in H1.
rewrite inv_invol in H1.
apply H1.
apply H0.
Qed.
```

Lemma 186 (surjection_conjugate_corollary) *Let $e : A \twoheadrightarrow C$ be a surjection and $f : A \to B$ be a function. Then,*

$$e \cdot e^\sharp \sqsubseteq f \cdot f^\sharp \Leftrightarrow (\exists! h : C \to B, f = e \cdot h).$$

```
Lemma surjection_conjugate_corollary {A B C : eqType} {f : Rel A B} {e : Rel A C}:
  surjection_r e → function_r f →
  ((e    e #)    (f    f #) ↔ ∃! h : Rel C B, function_r h ∧ f = e    h).
Proof.
move ⇒ H H0.
move : (@surjection_conjugate _ B _ _ H).
elim ⇒ H1 H2.
```

```
split; move ⇒ H3.
∃ (e #     f).
split.
move : (H1 f (conj H3 H0)).
elim ⇒ H4 H5.
split.
apply H4.
by [rewrite H5].
move ⇒ h.
elim ⇒ H4 H5.
rewrite H5 -comp_assoc.
replace (e #     e) with (Id C).
apply comp_id_l.
rewrite /surjection_r/function_r/total_r in H.
rewrite inv_invol in H.
apply inc_antisym.
apply H.
apply H.
elim H3 ⇒ h.
elim.
elim ⇒ H4 H5 H6.
rewrite H5 comp_inv comp_assoc -(@comp_assoc _ _ _ _ h).
apply comp_inc_compat_ab_ab'.
apply comp_inc_compat_b_ab.
apply H4.
Qed.
```

Lemma 187 (subid_conjugate) *Subidentity* $u \sqsubseteq id_A$ *corresponds* $\rho : I \rightarrow A$. *That is,*

$$\frac{\rho : I \rightarrow A}{u : A \rightarrow A \ \{u \sqsubseteq id_A\}} \quad \frac{\rho = \nabla_{IA} \cdot u}{u = id_A \sqcap \nabla_{AI} \cdot \rho}.$$

```
Lemma subid_conjugate {A : eqType}:
 conjugate i A A A True_r (fun u : Rel A A ⇒ u      Id A)
 (fun u : Rel A A ⇒     i A     u) (fun rho : Rel i A ⇒ Id A     (     A i     rho)).
Proof.
split.
move ⇒ alpha H.
split.
apply cap_l.
apply inc_antisym.
apply (@inc_trans _ _ _ (     i A     (     A i     alpha))).
apply comp_inc_compat_ab_ab'.
```

```
apply cap_r.
rewrite -comp_assoc.
apply comp_inc_compat_ab_b.
rewrite unit_identity_is_universal.
apply inc_alpha_universal.
rewrite -(@inv_universal i A).
apply (fun H' ⇒ @inc_trans _ _ _ _ _ H' (dedekind1)).
rewrite comp_id_r cap_comm cap_universal.
apply inc_refl.
move ⇒ beta H.
split.
by ||.
apply inc_antisym.
rewrite cap_comm -comp_assoc lemma_for_tarski2.
apply (@inc_trans _ _ _ _ _ (dedekind2)).
rewrite comp_id_l cap_comm cap_universal.
apply comp_inc_compat_ab_b.
rewrite -inv_inc_move inv_id.
apply H.
apply inc_cap.
split.
apply H.
rewrite -comp_assoc.
apply comp_inc_compat_b_ab.
rewrite lemma_for_tarski2.
apply inc_alpha_universal.
Qed.
```

Lemma 188 (subid_conjugate_corollary1) *Let $u, v : A \rightarrow A$ and $u, v \sqsubseteq id_A$. Then,*

$$\nabla_{IA} \cdot u = \nabla_{IA} \cdot v \Rightarrow u = v.$$

```
Lemma subid_conjugate_corollary1 {A : eqType} {u v : Rel A A}:
  u    Id A → v    Id A →    i A    u =    i A    v → u = v.
Proof.
move ⇒ H H0 H1.
move : (@subid_conjugate A).
elim ⇒ H2 H3.
move : (H3 u H).
elim ⇒ H4 H5.
rewrite -H5.
move : (H3 v H0).
elim ⇒ H6 H7.
```

```
rewrite -H7.
apply f_equal.
apply f_equal.
apply H1.
Qed.
```

Lemma 189 (subid_conjugate_corollary2) *Let* $\rho, \rho' : I \rightarrow A$. *Then,*

$$id_A \sqcap \nabla_{AI} \cdot \rho = id_A \sqcap \nabla_{AI} \cdot \rho' \Rightarrow \rho = \rho'.$$

Lemma *subid_conjugate_corollary2* $\{A : eqType\}$ $\{rho\ rho' : Rel\ i\ A\}$:
 $Id\ A$ $(\ \ A\ i\ \ \ rho) = Id\ A$ $(\ \ A\ i\ \ \ rho') \rightarrow rho = rho'$.
Proof.
move $\Rightarrow H$.
move : $(@subid\_conjugate\ A)$.
elim $\Rightarrow$ *H0 H1*.
move : $(H0\ rho\ I)$.
elim $\Rightarrow$ *H2 H3*.
rewrite -*H3*.
move : $(H0\ rho'\ I)$.
elim $\Rightarrow$ *H4 H5*.
rewrite -*H5*.
apply f_equal.
apply *H*.
Qed.

End *main*.

Chapter 9

Library Domain

Require Import *Basic_Notations*.
Require Import *Basic_Lemmas*.
Require Import *Relation_Properties*.
Require Import *Functions_Mappings*.
Require Import *Dedekind*.
Require Import *Logic.FunctionalExtensionality*.

Module *main* (*def* : *Relation*).
Import *def*.
Module *Basic_Lemmas* := *Basic_Lemmas.main def*.
Module *Relation_Properties* := *Relation_Properties.main def*.
Module *Functions_Mappings* := *Functions_Mappings.main def*.
Module *Dedekind* := *Dedekind.main def*.
Import *Basic_Lemmas Relation_Properties Functions_Mappings Dedekind*.

9.1

$$\alpha : A \rightharpoonup B \qquad , \qquad (\quad) \lfloor \alpha \rfloor : A \rightharpoonup A \quad ,$$

$$\lfloor \alpha \rfloor = \alpha \cdot \alpha^{\sharp} \sqcap id_A$$

. , Coq .

Definition *domain* {*A B* : *eqType*} (*alpha* : *Rel A B*):= (*alpha*   *alpha* #)   *Id A*.

135

9.2

9.2.1

Lemma 190 (domain_another_def) *Let $\alpha : A \rightharpoonup B$. Then,*

$$\lfloor \alpha \rfloor = \alpha \cdot \nabla_{BA} \sqcap id_A.$$

Lemma *domain_another_def* {*A B : eqType*} {*alpha : Rel A B*}:
  domain alpha = (alpha   B A)   Id A.
Proof.
apply *inc_antisym*.
apply *cap_inc_compat_r*.
apply *comp_inc_compat_ab_ab'*.
apply *inc_alpha_universal*.
apply *inc_cap*.
split.
apply (@*inc_trans* _ _ _ _ _ (*dedekind1*)).
apply *comp_inc_compat_ab_ab'*.
rewrite *cap_comm comp_id_r cap_universal*.
apply *inc_refl*.
apply *cap_r*.
Qed.

Lemma 191 (domain_inv) *Let $\alpha : A \rightharpoonup B$. Then,*

$$\lfloor \alpha \rfloor^{\sharp} = \lfloor \alpha \rfloor.$$

Lemma *domain_inv* {*A B : eqType*} {*alpha : Rel A B*}:
  (*domain alpha*) # = *domain alpha*.
Proof.
apply *dedekind_id1*.
apply *cap_r*.
Qed.

Lemma 192 (domain_comp_alpha1, domain_comp_alpha2) *Let $\alpha : A \rightharpoonup B$. Then,*

$$\lfloor \alpha \rfloor \cdot \alpha = \alpha \wedge \alpha^{\sharp} \cdot \lfloor \alpha \rfloor = \alpha^{\sharp}.$$

Lemma *domain_comp_alpha1* {*A B : eqType*} {*alpha : Rel A B*}:
  (*domain alpha*)   *alpha = alpha*.
Proof.

```
apply inc_antisym.
apply comp_inc_compat_ab_b.
apply cap_r.
rewrite /domain.
rewrite cap_comm.
apply (fun H' ⇒ @inc_trans _ _ _ _ _ H' (dedekind2)).
rewrite comp_id_l cap_idem.
apply inc_refl.
Qed.

Lemma domain_comp_alpha2 {A B : eqType} {alpha : Rel A B}:
  alpha #    (domain alpha) = alpha #.
Proof.
rewrite -domain_inv -comp_inv.
apply f_equal.
apply domain_comp_alpha1.
Qed.
```

Lemma 193 (domain_inc_compat) *Let* $\alpha, \alpha' : A \rightharpoonup B$. *Then,*

$$\alpha \sqsubseteq \alpha' \Rightarrow \lfloor \alpha \rfloor \sqsubseteq \lfloor \alpha' \rfloor.$$

```
Lemma domain_inc_compat {A B : eqType} {alpha alpha' : Rel A B}:
  alpha    alpha' → domain alpha    domain alpha'.
Proof.
move ⇒ H.
apply cap_inc_compat_r.
apply comp_inc_compat.
apply H.
apply (@inc_inv _ _ _ _ H).
Qed.
```

Lemma 194 (domain_total) *Let* $\alpha : A \rightharpoonup B$. *Then,*

$$\text{``}\alpha \text{ is total''} \Leftrightarrow \lfloor \alpha \rfloor = id_A.$$

```
Lemma domain_total {A B : eqType} {alpha : Rel A B}:
  total_r alpha ↔ domain alpha = Id A.
Proof.
split; move ⇒ H.
rewrite /domain.
rewrite cap_comm.
apply Logic.eq_sym.
apply inc_def1.
```

137

```
apply H.
apply inc_def1.
rewrite /domain in H.
by [rewrite cap_comm H].
Qed.
```

Lemma 195 (domain_inc_id) *Let* $u : A \rightharpoonup A$*. Then,*

$$u \sqsubseteq id_A \Leftrightarrow \lfloor u \rfloor = u.$$

```
Lemma domain_inc_id {A : eqType} {u : Rel A A}: u     Id A ↔ domain u = u.
Proof.
split; move ⇒ H.
rewrite /domain.
rewrite (dedekind_id1 H) (dedekind_id2 H).
apply inc_def1 in H.
by [rewrite -H].
rewrite -H.
apply cap_r.
Qed.
```

9.2.2

Lemma 196 (comp_domain1, comp_domain2) *Let* $\alpha : A \rightharpoonup B$ *and* $\beta : B \rightharpoonup C$*. Then,*
$$\lfloor \alpha \cdot \beta \rfloor = \lfloor \alpha \cdot \lfloor \beta \rfloor \rfloor \sqsubseteq \lfloor \alpha \rfloor.$$

```
Lemma comp_domain1 {A B C : eqType} {alpha : Rel A B} {beta : Rel B C}:
 domain (alpha    beta)    domain alpha.
Proof.
rewrite /domain.
rewrite comp_inv.
apply (@inc_trans _ _ _ ((alpha    ((beta    (beta #    alpha #))    alpha #))    Id A)).
replace (((alpha    beta)    (beta #    alpha #))    Id A) with ((((alpha    beta)
(beta #    alpha #))    Id A)    Id A).
apply cap_inc_compat_r.
rewrite comp_assoc.
apply (@inc_trans _ _ _ _ _ (dedekind1)).
rewrite comp_id_r.
apply inc_refl.
by [rewrite cap_assoc cap_idem].
apply cap_inc_compat_r.
```

138

```
apply comp_inc_compat_ab_ab'.
apply cap_r.
Qed.
```

Lemma *comp_domain2* {*A B C* : *eqType*} {*alpha* : *Rel A B*} {beta : *Rel B C*}:
 domain (*alpha*   beta) = *domain* (*alpha*   *domain* beta).
```
Proof.
apply inc_antisym.
replace (domain (alpha    beta)) with (domain ((alpha    domain beta)    beta)).
apply comp_domain1.
by [rewrite comp_assoc domain_comp_alpha1].
apply (@inc_trans _ _ _ (domain (alpha    (beta    beta #)))).
apply domain_inc_compat.
apply comp_inc_compat_ab_ab'.
apply cap_l.
rewrite -comp_assoc.
apply comp_domain1.
Qed.
```

Lemma 197 (comp_domain3) *Let $\alpha : A \rightharpoonup B$ be a relation and $\beta : B \rightharpoonup C$ be a total relation. Then,*
$$\lfloor \alpha \cdot \beta \rfloor = \lfloor \alpha \rfloor.$$

Lemma *comp_domain3* {*A B C* : *eqType*} {*alpha* : *Rel A B*} {beta : *Rel B C*}:
 total_r beta → *domain* (*alpha*   beta) = *domain alpha*.
```
Proof.
move ⇒ H.
apply inc_antisym.
apply comp_domain1.
rewrite /domain.
rewrite comp_inv comp_assoc -(@comp_assoc _ _ _ _ beta).
apply cap_inc_compat_r.
apply comp_inc_compat_ab_ab'.
apply (comp_inc_compat_b_ab H).
Qed.
```

Lemma 198 (comp_domain4) *Let $\alpha : A \rightharpoonup B$ and $\beta : B \rightharpoonup C$. Then,*
$$\lfloor \alpha^{\sharp} \rfloor \sqsubseteq \lfloor \beta \rfloor \Rightarrow \lfloor \alpha \cdot \beta \rfloor = \lfloor \alpha \rfloor.$$

Lemma *comp_domain4* {*A B C* : *eqType*} {*alpha* : *Rel A B*} {beta : *Rel B C*}:
 domain (*alpha* #)   *domain* beta → *domain* (*alpha*   beta) = *domain alpha*.
```
Proof.
```

move $\Rightarrow$ *H*.
apply *inc_antisym*.
apply *comp_domain1*.
rewrite $/domain$.
rewrite -(@*domain_comp_alpha1* _ _ (*alpha* #)) *comp_inv comp_assoc* -(@*comp_assoc* _ _ _ _ beta).
apply *cap_inc_compat_r*.
apply *comp_inc_compat_ab_ab'*.
apply *comp_inc_compat_ab_a'b*.
apply (@*inc_trans* _ _ _ _ _ *H*).
apply *cap_l*.
Qed.

Lemma 199 (comp_domain5) *Let* $\alpha : A \rightharpoonup B$ *be a univalent relation and* $\beta : B \rightharpoonup C$. *Then,*
$$\lfloor \alpha^\sharp \rfloor \sqsubseteq \lfloor \beta \rfloor \Leftrightarrow \lfloor \alpha \cdot \beta \rfloor = \lfloor \alpha \rfloor.$$

Lemma *comp_domain5* {*A B C* : *eqType*} {*alpha* : *Rel A B*} {beta : *Rel B C*}:
 univalent_r alpha $\rightarrow$
 (*domain* (*alpha* #)   *domain* beta $\leftrightarrow$ *domain* (*alpha*   beta) = *domain alpha*).
Proof.
move $\Rightarrow$ *H*.
split; move $\Rightarrow$ *H0*.
apply (*comp_domain4 H0*).
rewrite $/domain$.
rewrite *inv_invol*.
apply *cap_inc_compat_r*.
replace (*alpha* #   *alpha*) with (*alpha* #   (*domain* (*alpha*   beta)   *alpha*)).
rewrite $/domain$.
rewrite *comp_inv*.
apply (@*inc_trans* _ _ _ (*alpha* #   (((*alpha*   beta)   (beta #   *alpha* #))   *alpha*))).
apply *comp_inc_compat_ab_ab'*.
apply *comp_inc_compat_ab_a'b*.
apply *cap_l*.
rewrite *comp_assoc comp_assoc comp_assoc* -*comp_assoc* -(@*comp_assoc* _ _ _ _ beta).
apply (@*inc_trans* _ _ _ _ _ (*comp_inc_compat_ab_b H*)).
apply (*comp_inc_compat_ab_a H*).
by [rewrite *H0 domain_comp_alpha1*].
Qed.

Lemma 200 (comp_domain6) *Let* $\alpha : A \rightharpoonup B$ *and* $\beta : B \rightharpoonup C$. *Then,*

$$\alpha \cdot \lfloor \beta \rfloor \sqsubseteq \lfloor \alpha \cdot \beta \rfloor \cdot \alpha.$$

Lemma *comp_domain6* {$A\ B\ C : eqType$} {$alpha : Rel\ A\ B$} {beta $:\ Rel\ B\ C$}:
 ($alpha$ ⋅ $domain$ beta) ⊑ ($domain$ ($alpha$ ⋅ beta) ⋅ $alpha$).
Proof.
apply (@inc_trans _ _ _ _ _ (@$comp\_cap\_distr\_l$ _ _ _ _ _ _)).
rewrite *cap_comm*.
replace ($alpha$ ∩ $Id\ B$) with ($Id\ A$ ∩ $alpha$).
apply (@inc_trans _ _ _ _ _ (*dedekind2*)).
rewrite *cap_comm -comp_assoc comp_assoc -comp_inv*.
apply *inc_refl*.
by [rewrite *comp_id_l comp_id_r*].
Qed.

Lemma 201 (comp_domain7) *Let* $\alpha : A \rightharpoonup B$ *be a univalent relation and* $\beta : B \rightharpoonup C$. *Then,*
$$\alpha \cdot \lfloor \beta \rfloor = \lfloor \alpha \cdot \beta \rfloor \cdot \alpha.$$

Lemma *comp_domain7* {$A\ B\ C : eqType$} {$alpha : Rel\ A\ B$} {beta $:\ Rel\ B\ C$}:
 univalent_r alpha → $alpha$ ⋅ $domain$ beta = $domain$ ($alpha$ ⋅ beta) ⋅ $alpha$.
Proof.
move ⇒ H.
apply *inc_antisym*.
apply *comp_domain6*.
apply (@inc_trans _ _ _ _ _ (@$comp\_cap\_distr\_r$ _ _ _ _ _ _)).
rewrite *comp_id_l comp_inv comp_assoc comp_assoc*.
apply (@inc_trans _ _ _ _ _ (*dedekind1*)).
apply *comp_inc_compat_ab_ab'*.
apply (fun $H' ⇒ cap\_inc\_compat\ H'\ H$).
rewrite *comp_assoc -comp_assoc*.
apply ($comp\_inc\_compat\_ab\_a\ H$).
Qed.

Lemma 202 (comp_domain8) *Let* $u : A \rightharpoonup A$, $\alpha : A \rightharpoonup B$ *and* $u \sqsubseteq id_A$. *Then,*

$$\lfloor u \cdot \alpha \rfloor = u \cdot \lfloor \alpha \rfloor.$$

Lemma *comp_domain8* {$A\ B : eqType$} {$u : Rel\ A\ A$} {$alpha : Rel\ A\ B$}:
 u ⊑ $Id\ A$ → $domain$ (u ⋅ $alpha$) = u ⋅ $domain\ alpha$.
Proof.

```
move ⇒ H.
apply inc_antisym.
rewrite -(@cap_idem _ _ (domain (u    alpha))).
rewrite (dedekind_id3 H).
apply cap_inc_compat.
apply (@inc_trans _ _ _ _ _ (comp_domain1)).
apply domain_inc_id in H.
rewrite H.
apply inc_refl.
apply domain_inc_compat.
apply (comp_inc_compat_ab_b H).
apply cap_r.
apply (@inc_trans _ _ _ _ _ (comp_domain6)).
apply (comp_inc_compat_ab_a H).
Qed.
```

9.2.3

> **Lemma 203 (cap_domain)** *Let* $\alpha, \alpha' : A \rightarrow B$. *Then,*
>
> $$\lfloor \alpha \sqcap \alpha' \rfloor = \alpha \cdot \alpha'^{\sharp} \sqcap id_A.$$

```
Lemma cap_domain {A B : eqType} {alpha alpha' : Rel A B}:
 domain (alpha    alpha') = (alpha    alpha' #)    Id A.
Proof.
apply inc_antisym.
apply cap_inc_compat_r.
apply comp_inc_compat.
apply cap_l.
apply inc_inv.
apply cap_r.
rewrite -(@cap_idem _ _ (Id A)) -cap_assoc.
apply cap_inc_compat_r.
apply (@inc_trans _ _ _ _ _ (@dedekind _ _ _ _ _ _)).
rewrite inv_invol comp_id_l comp_id_r -inv_cap_distr (@cap_comm _ _ alpha').
apply inc_refl.
Qed.
```

Lemma 204 (cupP_domain_distr, cup_domain_distr) *Let $f : (C \to D) \to (A \to B)$ and P : predicate. Then,*

$$\lfloor \sqcup_{P(\alpha)} f(\alpha) \rfloor = \sqcup_{P(\alpha)} \lfloor f(\alpha) \rfloor.$$

Lemma *cupP_domain_distr* $\{A \ B \ C \ D : eqType\}$ $\{f : Rel \ C \ D \to Rel \ A \ B\}$ $\{P : Rel \ C \ D$ $\to$ Prop$\}$:
 domain (⊔_$\{P\}$ *f*) = ⊔_$\{P\}$ (fun *alpha* : *Rel C D* $\Rightarrow$ *domain* (*f alpha*)).
Proof.
rewrite /*domain*.
rewrite *inv_cupP_distr comp_cupP_distr_l cap_cupP_distr_r*.
apply *cupP_eq*.
move $\Rightarrow$ *alpha H*.
rewrite -*cap_domain* -*cap_domain*.
apply f_equal.
rewrite *cap_idem*.
apply *inc_antisym*.
apply *cap_r*.
apply *inc_cap*.
split.
move : *alpha H*.
apply *inc_cupP*.
apply *inc_refl*.
apply *inc_refl*.
Qed.

Lemma *cup_domain_distr* $\{A \ B : eqType\}$ $\{alpha \ alpha' : Rel \ A \ B\}$:
 domain (*alpha* ⊔ *alpha'*) = *domain alpha* ⊔ *domain alpha'*.
Proof.
rewrite *cup_to_cupP* (@*cup_to_cupP* _ _ _ _ _ _ *id*).
apply *cupP_domain_distr*.
Qed.

Lemma 205 (domain_universal1) *Let $\alpha : A \to B$. Then,*

$$\lfloor \alpha \rfloor \cdot \nabla_{AC} = \alpha \cdot \nabla_{BC}.$$

Lemma *domain_universal1* $\{A \ B \ C : eqType\}$ $\{alpha : Rel \ A \ B\}$:
 domain alpha ⋅ ∇ *A C* = *alpha* ⋅ ∇ *B C*.
Proof.
apply *inc_antisym*.
apply (@*inc_trans* _ _ _ ((*alpha* ⋅ *alpha* #) ⋅ ∇ *A C*)).
apply *comp_inc_compat_ab_a'b*.

```
apply cap_l.
rewrite comp_assoc.
apply comp_inc_compat_ab_ab'.
apply inc_alpha_universal.
apply (@inc_trans _ _ _ ((domain alpha     alpha)        B C)).
rewrite domain_comp_alpha1.
apply inc_refl.
rewrite comp_assoc.
apply comp_inc_compat_ab_ab'.
apply inc_alpha_universal.
Qed.
```

Lemma 206 (domain_universal2) *Let $\alpha : A \rightharpoonup B$ and $\beta : B \rightharpoonup C$. Then,*

$$\alpha \cdot \lfloor \beta \rfloor = \alpha \sqcap \nabla_{AC} \cdot \beta^{\sharp}.$$

```
Lemma domain_universal2 {A B C : eqType} {alpha : Rel A B} {beta : Rel B C}:
 alpha    domain beta = alpha    (   A C    beta #).
Proof.
apply inc_antisym.
apply inc_cap.
split.
apply comp_inc_compat_ab_a.
apply cap_r.
apply (@inc_trans _ _ _ _ _ (comp_cap_distr_l)).
apply (@inc_trans _ _ _ _ _ (cap_l)).
rewrite -comp_assoc.
apply comp_inc_compat_ab_a'b.
apply inc_alpha_universal.
rewrite -inv_universal -comp_inv -domain_universal1.
rewrite comp_inv inv_universal domain_inv cap_comm.
apply (@inc_trans _ _ _ _ _ (dedekind2)).
apply comp_inc_compat_ab_a'b.
rewrite cap_comm cap_universal domain_inv.
apply comp_inc_compat_ab_a.
apply cap_r.
Qed.
```

Lemma 207 (domain_lemma1) *Let $\alpha, \beta : A \rightharpoonup B$ and β is univalent. Then,*

$$\alpha \sqsubseteq \beta \wedge \lfloor \alpha \rfloor = \lfloor \beta \rfloor \Rightarrow \alpha = \beta.$$

```
Lemma domain_lemma1 {A B : eqType} {alpha beta : Rel A B}:
```

  $univalent\_r$ beta $\rightarrow alpha$ beta $\rightarrow domain\ alpha = domain$ beta $\rightarrow alpha =$ beta.
Proof.
move $\Rightarrow$ *H H0 H1*.
apply *inc_antisym*.
apply *H0*.
rewrite -(@$domain\_comp\_alpha1$ _ _ beta) -*H1*.
apply (@inc_trans _ _ _ _ _ ($comp\_cap\_distr\_r$)).
apply (@inc_trans _ _ _ _ _ (cap_l)).
rewrite $comp\_assoc$.
apply $comp\_inc\_compat\_ab\_a$.
apply (fun *H'* $\Rightarrow$ @inc_trans _ _ _ _ _ *H' H*).
apply $comp\_inc\_compat\_ab\_a'b$.
apply (@inc_inv _ _ _ _ *H0*).
Qed.

Lemma 208 (domain_lemma2a, domain_lemma2b) *Let* $\alpha : A \rightarrow B$ *and* $\beta : A \rightarrow C$. *Then,*
$$\lfloor \alpha \rfloor \sqsubseteq \lfloor \beta \rfloor \Leftrightarrow \alpha \cdot \nabla_{BB} \sqsubseteq \beta \cdot \nabla_{CB} \Leftrightarrow \alpha \sqsubseteq \beta \cdot \beta^\sharp \cdot \alpha.$$

Lemma $domain\_lemma2a$ $\{A\ B\ C : eqType\}$ $\{alpha : Rel\ A\ B\}$ $\{$beta$: Rel\ A\ C\}$:
 $domain\ alpha$ $domain$ beta $\leftrightarrow$ ($alpha$ $B\ B$)   (beta $C\ B$).
Proof.
split; move $\Rightarrow$ *H*.
rewrite -(@$domain\_comp\_alpha1$ _ _ $alpha$) $comp\_assoc$.
apply (@inc_trans _ _ _ _ _ ($comp\_inc\_compat\_ab\_a'b$ *H*)).
apply (@inc_trans _ _ _ _ _ ($comp\_inc\_compat\_ab\_a'b$ (cap_l))).
rewrite $comp\_assoc$.
apply $comp\_inc\_compat\_ab\_ab'$.
apply $inc\_alpha\_universal$.
apply (@inc_trans _ _ _ ($domain$ ((beta   beta $\#$) $alpha$))).
apply $domain\_inc\_compat$.
apply (@inc_trans _ _ _ ($alpha$ (beta $C\ B$))).
apply (fun *H'* $\Rightarrow$ @inc_trans _ _ _ _ _ *H'* ($cap\_inc\_compat\_l$ *H*)).
replace ($alpha$ ($alpha$ $B\ B$)) with (($alpha$ $Id\ B$)   ($alpha$ $B\ B$)).
apply (fun *H'* $\Rightarrow$ @inc_trans _ _ _ _ _ *H'* ($comp\_cap\_distr\_l$)).
rewrite $cap\_universal$ $comp\_id\_r$.
apply inc_refl.
by [rewrite $comp\_id\_r$].
rewrite cap_comm $comp\_assoc$.
apply (@inc_trans _ _ _ _ _ ($dedekind1$)).
rewrite cap_comm $cap\_universal$.
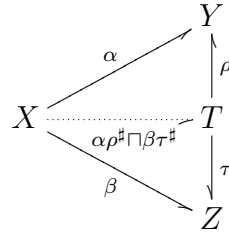apply inc_refl.

```
rewrite comp_assoc.
apply comp_domain1.
Qed.
Lemma domain_lemma2b {A B C : eqType} {alpha : Rel A B} {beta : Rel A C}:
 domain alpha    domain beta ↔ alpha    ((beta    beta #)    alpha).
Proof.
split; move ⇒ H.
apply domain_lemma2a in H.
apply (@inc_trans _ _ _ (alpha    (beta        C B))).
apply (fun H' ⇒ @inc_trans _ _ _ _ _ H' (cap_inc_compat_l H)).
replace (alpha    (alpha        B B)) with ((alpha    Id B)    (alpha        B B)).
apply (fun H' ⇒ @inc_trans _ _ _ _ _ H' (comp_cap_distr_l)).
rewrite cap_universal comp_id_r.
apply inc_refl.
by [rewrite comp_id_r].
rewrite cap_comm comp_assoc.
apply (@inc_trans _ _ _ _ _ (dedekind1)).
rewrite cap_comm cap_universal.
apply inc_refl.
apply domain_inc_compat in H.
apply (@inc_trans _ _ _ _ _ H).
rewrite comp_assoc.
apply comp_domain1.
Qed.
```

Lemma 209 (domain_corollary1) *In below figure,*

$$\text{``}\alpha \text{ and } \beta \text{ are total''} \wedge \alpha^\sharp \cdot \beta \sqsubseteq \rho^\sharp \cdot \tau \Rightarrow \text{``}\alpha \cdot \rho^\sharp \sqcap \beta \cdot \tau^\sharp \text{ is total''}.$$

```
Lemma domain_corollary1 {X Y Z T : eqType}
 {alpha : Rel X Y} {beta : Rel X Z} {rho : Rel T Y} {tau : Rel T Z}:
 total_r alpha → total_r beta → (alpha #    beta)    (rho #    tau) →
 total_r ((alpha    rho #)    (beta    tau #)).
Proof.
move ⇒ H H0 H1.
move : (comp_inc_compat H H0) ⇒ H2.
```

146

rewrite *comp_id_l* -*comp_assoc* (@*comp_assoc* _ _ _ _ *alpha*) in *H2*.
rewrite /*total_r*.
replace (*Id X*) with (((*alpha* ⋅ (*rho* # ⋅ *tau*)) ⊓ beta #) ⋅ *Id X*).
rewrite -*comp_assoc comp_assoc*.
apply (@*inc_trans* _ _ _ _ _ (@*dedekind* _ _ _ _ _ _)).
rewrite *comp_id_l comp_id_r comp_inv comp_inv inv_invol inv_invol*.
rewrite *inv_cap_distr comp_inv comp_inv inv_invol inv_invol* (@*cap_comm* _ _ (*tau* ⋅ beta #)).
apply *inc_refl*.
apply *Logic.eq_sym*.
rewrite *cap_comm*.
apply *inc_def1*.
apply (@*inc_trans* _ _ _ _ _ *H2*).
apply *comp_inc_compat_ab_a'b*.
apply (*comp_inc_compat_ab_ab' H1*).
Qed.

Lemma 210 (domain_corollary2) *In below figure,*

"*α and β are univalent*" ∧ *ρ · ρ♯ ⊓ τ · τ♯ = id_T* ⇒ "*α · ρ♯ ⊓ β · τ♯ is univalent*".
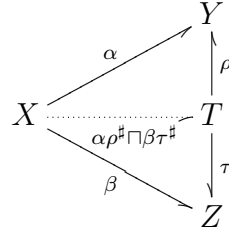
Lemma *domain_corollary2* {*X Y Z T* : *eqType*}
{*alpha* : *Rel X Y*} {beta : *Rel X Z*} {*rho* : *Rel T Y*} {*tau* : *Rel T Z*}:
univalent_r alpha → *univalent_r* beta → (*rho* ⋅ *rho* #) ⊓ (*tau* ⋅ *tau* #) = *Id T* →
univalent_r ((*alpha* ⋅ *rho* #) ⊓ (beta ⋅ *tau* #)).
Proof.
move ⇒ *H H0 H1*.
rewrite /*univalent_r*.
rewrite -*H1 inv_cap_distr*.
apply (@*inc_trans* _ _ _ _ _ (*comp_cap_distr_l*)).
apply *cap_inc_compat*.
apply (@*inc_trans* _ _ _ _ _ (*comp_cap_distr_r*)).
apply (@*inc_trans* _ _ _ _ _ (*cap_l*)).
rewrite *comp_inv inv_invol* -*comp_assoc* (@*comp_assoc* _ _ _ _ *rho*).
apply *comp_inc_compat_ab_a'b*.
apply (*comp_inc_compat_ab_a H*).

```
apply (@inc_trans _ _ _ _ _ (comp_cap_distr_r)).
apply (@inc_trans _ _ _ _ _ (cap_r)).
rewrite comp_inv inv_invol -comp_assoc (@comp_assoc _ _ _ _ tau).
apply comp_inc_compat_ab_a'b.
apply (comp_inc_compat_ab_a H0).
Qed.
```

9.2.4

$$\alpha : A \rightharpoonup B$$

$$\alpha \cdot \nabla_{BA} \cdot \alpha \sqsubseteq \alpha$$

, α (rectangular relation)   .

```
Definition rectangular {A B : eqType} (alpha : Rel A B):=
 ((alpha      B A)    alpha)    alpha.
```

Lemma 211 (rectangular_inv) *Let* $\alpha : A \rightharpoonup B$ *is a rectangular relation, then* $\alpha^{\sharp}$ *is also a rectangular relation.*

```
Lemma rectangular_inv {A B : eqType} {alpha : Rel A B}:
 rectangular alpha → rectangular (alpha #).
Proof.
move ⇒ H.
apply inv_inc_move.
rewrite comp_inv comp_inv inv_invol inv_universal -comp_assoc.
apply H.
Qed.
```

Lemma 212 (rectangular_capP, rectangular_cap) *Let* $f(\alpha)$ *is always a rectangular relation and* P : *predicate, then* $\sqcap_{P(\beta)} f(\beta)$ *is also a rectangular relation.*

```
Lemma rectangular_capP {A B C D : eqType} {f : Rel C D → Rel A B} {P : Rel C D →
Prop}:
 (∀ alpha : Rel C D, P alpha → rectangular (f alpha)) → rectangular (   _{P} f).
Proof.
move ⇒ H.
rewrite /rectangular.
apply (@inc_trans _ _ _ (   _{P} (fun alpha : Rel C D ⇒ (f alpha      B A)    f alpha))).
apply (@inc_trans _ _ _ _ _ (comp_capP_distr_l)).
apply inc_capP.
move ⇒ beta H0.
```

```
apply (@inc_trans _ _ _ ((( _{P} f)        B A)     f beta)).
move : beta H0.
apply inc_capP.
apply inc_refl.
apply comp_inc_compat_ab_a'b.
apply comp_inc_compat_ab_a'b.
move : H0.
apply inc_capP.
apply inc_refl.
apply inc_capP.
move ⇒ beta H0.
apply (fun H' ⇒ @inc_trans _ _ _ _ _ H' (H beta H0)).
move : beta H0.
apply inc_capP.
apply inc_refl.
Qed.
```

```
Lemma rectangular_cap {A B : eqType} {alpha beta : Rel A B}:
 rectangular alpha → rectangular beta → rectangular (alpha    beta).
Proof.
move ⇒ H H0.
rewrite (@cap_to_capP _ _ _ _ _ id).
apply rectangular_capP.
move ⇒ gamma.
case ⇒ H1; rewrite H1.
apply H.
apply H0.
Qed.
```

Lemma 213 (rectangular_comp) *Let $\alpha : A \rightarrow B$, $\beta : B \rightarrow C$ and α or β is a rectangular relation, then $\alpha \cdot \beta$ is also a rectangular relation.*

```
Lemma rectangular_comp {A B C : eqType} {alpha : Rel A B} {beta : Rel B C}:
 rectangular alpha ∨ rectangular beta → rectangular (alpha    beta).
Proof.
rewrite /rectangular.
case; move ⇒ H.
rewrite -comp_assoc.
apply comp_inc_compat_ab_a'b.
apply (fun H' ⇒ @inc_trans _ _ _ _ _ H' H).
apply comp_inc_compat_ab_a'b.
rewrite comp_assoc.
apply comp_inc_compat_ab_ab'.
```

```
apply inc_alpha_universal.
rewrite comp_assoc comp_assoc.
apply comp_inc_compat_ab_ab'.
apply (fun H' ⇒ @inc_trans _ _ _ _ _ H' H).
rewrite -comp_assoc -comp_assoc.
apply comp_inc_compat_ab_a'b.
rewrite comp_assoc.
apply comp_inc_compat_ab_ab'.
apply inc_alpha_universal.
Qed.
```

Lemma 214 (rectangular_unit) *Let $\alpha : A \rightarrow B$. Then,*

$$\text{``}\alpha \text{ is rectangular''} \Leftrightarrow \exists \mu : I \rightarrow A, \exists \rho : I \rightarrow B, \alpha = \rho^{\sharp} \cdot \mu.$$

```
Lemma rectangular_unit {A B : eqType} {alpha : Rel A B}:
 rectangular alpha ↔ ∃ (mu : Rel i A)(rho : Rel i B), alpha = mu #   rho.
Proof.
split; move ⇒ H.
∃ (  i B    alpha #).
∃ (  i A    alpha).
rewrite comp_inv inv_invol inv_universal.
rewrite -comp_assoc (@comp_assoc _ _ _ _ alpha) lemma_for_tarski2.
apply inc_antisym.
apply (@inc_trans _ _ _ _ _ (relation_rel_inv_rel)).
apply comp_inc_compat_ab_a'b.
apply comp_inc_compat_ab_ab'.
apply inc_alpha_universal.
apply H.
elim H ⇒ mu.
elim ⇒ rho H0.
rewrite H0.
rewrite /rectangular.
rewrite -comp_assoc.
apply comp_inc_compat_ab_a'b.
rewrite comp_assoc comp_assoc.
apply comp_inc_compat_ab_a.
rewrite unit_identity_is_universal.
apply inc_alpha_universal.
Qed.

End main.
```

Chapter 10

Library Residual

Require Import *Basic_Notations*.
Require Import *Basic_Lemmas*.
Require Import *Relation_Properties*.
Require Import *Functions_Mappings*.
Require Import *Dedekind*.
Require Import *Domain*.
Require Import *Logic.FunctionalExtensionality*.

Module *main* (*def* : *Relation*).
Import *def*.
Module *Basic_Lemmas* := *Basic_Lemmas.main def*.
Module *Relation_Properties* := *Relation_Properties.main def*.
Module *Functions_Mappings* := *Functions_Mappings.main def*.
Module *Dedekind* := *Dedekind.main def*.
Module *Domain* := *Domain.main def*.
Import *Basic_Lemmas Relation_Properties Functions_Mappings Dedekind Domain*.

10.1

10.1.1

Lemma 215 (double_residual) *Let $\alpha : A \rightharpoonup B$, $\beta : B \rightharpoonup C$ and $\gamma : C \rightharpoonup D$. Then*

$$\alpha \rhd (\beta \rhd \gamma) = (\alpha \cdot \beta) \rhd \gamma.$$

Lemma *double_residual*
 {*A B C D* : *eqType*} {*alpha* : *Rel A B*} {beta : *Rel B C*} {*gamma* : *Rel C D*}:
 alpha (beta *gamma*) = (*alpha* beta) *gamma*.
Proof.
apply *inc_lower*.

151

```
move ⇒ delta.
split; move ⇒ H.
apply inc_residual.
rewrite comp_inv comp_assoc.
rewrite -inc_residual -inc_residual.
apply H.
rewrite inc_residual inc_residual.
rewrite -comp_assoc -comp_inv.
apply inc_residual.
apply H.
Qed.
```

Lemma 216 (residual_to_complement) *Let $\alpha : A \rightarrow B$ and $\beta : B \rightarrow C$. Then*

$$\alpha \rhd \beta = (\alpha \cdot \beta^-)^-.$$

```
Lemma residual_to_complement {A B C : eqType} {alpha : Rel A B} {beta : Rel B C}:
  alpha    beta = (alpha    beta ^) ^.
Proof.
apply inc_lower.
move ⇒ gamma.
split; move ⇒ H.
rewrite bool_lemma2 complement_invol cap_comm.
apply inc_antisym.
apply (@inc_trans _ _ _ _ _ (dedekind1)).
replace (beta ^    (alpha #    gamma)) with (    B C).
rewrite comp_empty_r.
apply inc_refl.
apply Logic.eq_sym.
rewrite cap_comm.
apply bool_lemma2.
apply inc_residual.
apply H.
apply inc_empty_alpha.
apply inc_residual.
apply bool_lemma2.
apply inc_antisym.
apply (@inc_trans _ _ _ _ _ (dedekind1)).
rewrite inv_invol.
replace (gamma    (alpha    beta ^)) with (    A C).
rewrite comp_empty_r.
apply inc_refl.
apply Logic.eq_sym.
```

```
rewrite -(@complement_invol _ _ (alpha    beta ^)).
apply bool_lemma2.
apply H.
apply inc_empty_alpha.
Qed.
```

Lemma 217 (inv_residual_inc) *Let $\alpha : A \rightharpoonup B$ and $\beta : B \rightharpoonup C$. Then*

$$\alpha^{\sharp} \cdot (\alpha \rhd \beta) \sqsubseteq \beta.$$

```
Lemma inv_residual_inc {A B C : eqType} {alpha : Rel A B} {beta : Rel B C}:
 alpha #    (alpha    beta)    beta.
Proof.
apply inc_residual.
apply inc_refl.
Qed.
```

Lemma 218 (inc_residual_inv) *Let $\alpha : A \rightharpoonup B$ and $\gamma : A \rightharpoonup C$. Then*

$$\gamma \sqsubseteq \alpha \rhd \alpha^{\sharp} \cdot \gamma.$$

```
Lemma inc_residual_inv {A B C : eqType} {alpha : Rel A B} {gamma : Rel A C}:
 gamma    (alpha    (alpha #    gamma)).
Proof.
apply inc_residual.
apply inc_refl.
Qed.
```

Lemma 219 (id_inc_residual) *Let $\alpha : A \rightharpoonup B$. Then*

$$id_A \sqsubseteq \alpha \rhd \alpha^{\sharp}.$$

```
Lemma id_inc_residual {A B : eqType} {alpha : Rel A B}: Id A    (alpha    alpha #).
Proof.
apply inc_residual.
rewrite comp_id_r.
apply inc_refl.
Qed.
```

Lemma 220 (residual_universal) *Let $\alpha : A \rightharpoonup B$. Then*

$$\alpha \rhd \nabla_{BC} = \nabla_{AC}.$$

153

Lemma *residual_universal* {*A B C* : *eqType*} {*alpha* : *Rel A B*}: *alpha* ▷ *B C* = ▷ *A C*.
Proof.
apply *inc_antisym*.
apply *inc_alpha_universal*.
apply *inc_residual*.
apply *inc_alpha_universal*.
Qed.

10.1.2

> **Lemma 221 (residual_inc_compat)** *Let* $\alpha, \alpha' : A \to B$ *and* $\beta, \beta' : B \to C$. *Then*
>
> $$\alpha' \sqsubseteq \alpha \wedge \beta \sqsubseteq \beta' \Rightarrow \alpha \rhd \beta \sqsubseteq \alpha' \rhd \beta'.$$

Lemma *residual_inc_compat*
 {*A B C* : *eqType*} {*alpha alpha'* : *Rel A B*} {beta *beta'* : *Rel B C*}:
 alpha' ⊆ *alpha* → beta ⊆ *beta'* → (*alpha* ▷ beta) ⊆ (*alpha'* ▷ *beta'*).
Proof.
move ⇒ *H H0*.
apply *inc_residual*.
apply (fun *H'* ⇒ @*inc_trans* _ _ _ _ _ *H' H0*).
move : (@*inc_refl* _ _ (*alpha* ▷ beta)) ⇒ *H1*.
apply *inc_residual* in *H1*.
apply (fun *H'* ⇒ @*inc_trans* _ _ _ _ _ *H' H1*).
apply *comp_inc_compat_ab_a'b*.
apply *inc_inv*.
apply *H*.
Qed.

> **Lemma 222 (residual_inc_compat_l)** *Let* $\alpha : A \to B$ *and* $\beta, \beta' : B \to C$. *Then*
>
> $$\beta \sqsubseteq \beta' \Rightarrow \alpha \rhd \beta \sqsubseteq \alpha \rhd \beta'.$$

Lemma *residual_inc_compat_l*
 {*A B C* : *eqType*} {*alpha* : *Rel A B*} {beta *beta'* : *Rel B C*}:
 beta ⊆ *beta'* → (*alpha* ▷ beta) ⊆ (*alpha* ▷ *beta'*).
Proof.
move ⇒ *H*.
apply (@*residual_inc_compat* _ _ _ _ _ _ _ (@*inc_refl* _ _ _) *H*).
Qed.

> **Lemma 223 (residual_inc_compat_r)** *Let $\alpha, \alpha' : A \rightarrow B$ and $\beta : B \rightarrow C$. Then*
>
> $$\alpha' \sqsubseteq \alpha \Rightarrow \alpha \rhd \beta \sqsubseteq \alpha' \rhd \beta.$$

Lemma *residual_inc_compat_r*
 {*A B C : eqType*} {*alpha alpha' : Rel A B*} {beta : *Rel B C*}:
 alpha'   *alpha* → (*alpha*   beta)   (*alpha'*   beta).
Proof.
move ⇒ *H*.
apply (@*residual_inc_compat* _ _ _ _ _ _ _ *H* (@*inc_refl* _ _ _)).
Qed.

> **Lemma 224 (residual_capP_distr_l, residual_cap_distr_l)** *Let $\alpha : A \rightarrow B$, $f : (D \rightarrow E) \rightarrow (B \rightarrow C)$ and P : predicate. Then*
>
> $$\alpha \rhd (\sqcap_{P(\beta)} f(\beta)) = \sqcap_{P(\beta)}(\alpha \rhd f(\beta)).$$

Lemma *residual_capP_distr_l* {*A B C D E : eqType*}
 {*alpha : Rel A B*} {*f : Rel D E → Rel B C*} {*P : Rel D E →* Prop}:
 alpha   (_{*P*} *f*) =   _{*P*} (fun beta : *Rel D E* ⇒ *alpha*   *f* beta).
Proof.
apply *inc_lower*.
move ⇒ *gamma*.
split; move ⇒ *H*.
apply *inc_capP*.
move ⇒ beta *H0*.
apply *inc_residual*.
move : beta *H0*.
apply *inc_capP*.
apply *inc_residual*.
apply *H*.
apply *inc_residual*.
apply *inc_capP*.
move ⇒ beta *H0*.
apply *inc_residual*.
move : beta *H0*.
apply *inc_capP*.
apply *H*.
Qed.

Lemma *residual_cap_distr_l*
 {*A B C : eqType*} {*alpha : Rel A B*} {beta *gamma : Rel B C*}:
 alpha   (beta   *gamma*) = (*alpha*   beta)   (*alpha*   *gamma*).

Proof.
rewrite *cap_to_capP* (@*cap_to_capP* _ _ _ _ _ _ *id*).
apply *residual_capP_distr_l*.
Qed.

Lemma 225 (residual_cupP_distr_r, residual_cup_distr_r) *Let* $f : (D \rightharpoonup E) \rightarrow (A \rightharpoonup B)$, $\beta : B \rightharpoonup C$ *and* P : *predicate. Then*

$$(\sqcup_{P(\alpha)} f(\alpha)) \triangleright \beta = \sqcap_{P(\alpha)}(f(\alpha) \triangleright \beta).$$

Lemma *residual_cupP_distr_r* {*A B C D E* : *eqType*}
 {beta : *Rel B C*} {*f* : *Rel D E* → *Rel A B*} {*P* : *Rel D E* → Prop}:
 (_{*P*} *f*)   beta =   _{*P*} (fun *alpha* : *Rel D E* ⇒ *f alpha*   beta).
Proof.
apply *inc_lower*.
move ⇒ *gamma*.
split; move ⇒ *H*.
apply *inc_capP*.
move ⇒ *alpha H0*.
apply *inc_residual*.
move : *alpha H0*.
apply *inc_cupP*.
rewrite -*comp_cupP_distr_r* -*inv_cupP_distr*.
apply *inc_residual*.
apply *H*.
apply *inc_residual*.
rewrite *inv_cupP_distr comp_cupP_distr_r*.
apply *inc_cupP*.
move ⇒ *alpha H0*.
apply *inc_residual*.
move : *alpha H0*.
apply *inc_capP*.
apply *H*.
Qed.

Lemma *residual_cup_distr_r*
 {*A B C* : *eqType*} {*alpha* beta : *Rel A B*} {*gamma* : *Rel B C*}:
 (*alpha*   beta)   *gamma* = (*alpha*   *gamma*)   (beta   *gamma*).
Proof.
rewrite (@*cup_to_cupP* _ _ _ _ _ _ *id*) (@*cap_to_capP* _ _ _ _ _ _ (fun *x* ⇒ *x*   *gamma*)).
apply *residual_cupP_distr_r*.
Qed.

10.1.3

Lemma 226 (total_residual) *Let $\alpha : A \to B$ be a total relation and $\beta : B \to C$. Then*

$$\alpha \rhd \beta \sqsubseteq \alpha \cdot \beta.$$

Lemma *total_residual* {*A B C : eqType*} {*alpha : Rel A B*} {beta : *Rel B C*}:
 total_r alpha → (*alpha*   beta)   (*alpha*   beta).
Proof.
move ⇒ *H*.
apply (@*inc_trans* _ _ _ ((*alpha*   *alpha* #)   (*alpha*   beta))).
apply (*comp_inc_compat_b_ab H*).
rewrite *comp_assoc*.
apply *comp_inc_compat_ab_ab'*.
apply *inv_residual_inc*.
Qed.

Lemma 227 (univalent_residual) *Let $\alpha : A \to B$ be a univalent relation and $\beta : B \to C$. Then*

$$\alpha \cdot \beta \sqsubseteq \alpha \rhd \beta.$$

Lemma *univalent_residual* {*A B C : eqType*} {*alpha : Rel A B*} {beta : *Rel B C*}:
 univalent_r alpha → (*alpha*   beta)   (*alpha*   beta).
Proof.
move ⇒ *H*.
apply (@*inc_trans* _ _ _ _ _ (@*inc_residual_inv* _ _ _ *alpha* _)).
apply *residual_inc_compat_l*.
rewrite -*comp_assoc*.
apply (*comp_inc_compat_ab_b H*).
Qed.

Lemma 228 (function_residual1) *Let $\alpha : A \to B$ be a function and $\beta : B \to C$. Then*

$$\alpha \rhd \beta = \alpha \cdot \beta.$$

Lemma *function_residual1* {*A B C : eqType*} {*alpha : Rel A B*} {beta : *Rel B C*}:
 function_r alpha → *alpha*   beta = *alpha*   beta.
Proof.
elim ⇒ *H H0*.
apply *inc_antisym*.
apply (*total_residual H*).
apply (*univalent_residual H0*).
Qed.

Lemma 229 (residual_id) *Let* $\alpha : A \rightharpoonup B$. *Then*

$$id_A \rhd \alpha = \alpha.$$

Lemma *residual_id* $\{A\ B : eqType\}$ $\{alpha : Rel\ A\ B\}$:
 Id A   *alpha* $=$ *alpha*.
Proof.
move : (@*function_residual1* _ _ _ (*Id A*) *alpha* (@*id_function A*)) $\Rightarrow$ *H*.
rewrite *comp_id_l* in *H*.
apply *H*.
Qed.

Lemma 230 (universal_residual) *Let* $\alpha : A \rightharpoonup B$. *Then*

$$\nabla_{AA} \rhd \alpha \sqsubseteq \alpha.$$

Lemma *universal_residual* $\{A\ B : eqType\}$ $\{alpha : Rel\ A\ B\}$:
    A A   *alpha*   *alpha*.
Proof.
apply (@*inc_trans* _ _ _ (*Id A*   *alpha*)).
apply *residual_inc_compat_r*.
apply *inc_alpha_universal*.
rewrite *residual_id*.
apply *inc_refl*.
Qed.

Lemma 231 (function_residual2) *Let* $\alpha : A \rightharpoonup B$ *be a function,* $\beta : B \rightharpoonup C$ *and* $\gamma : C \rightharpoonup D$. *Then*

$$\alpha \cdot (\beta \rhd \gamma) = (\alpha \cdot \beta) \rhd \gamma.$$

Lemma *function_residual2*
 $\{A\ B\ C\ D : eqType\}$ $\{alpha : Rel\ A\ B\}$ $\{beta : Rel\ B\ C\}$ $\{gamma : Rel\ C\ D\}$:
 function_r alpha $\rightarrow$ *alpha*   (beta   *gamma*) $=$ (*alpha*   beta)   *gamma*.
Proof.
move $\Rightarrow$ *H*.
rewrite -(@*function_residual1* _ _ _ _ _ *H*).
apply *double_residual*.
Qed.

Lemma 232 (function_residual3) *Let $\alpha : A \rightharpoonup B$, $\beta : B \rightharpoonup C$ be relations and $\gamma : D \to C$ be a function. Then*

$$(\alpha \rhd \beta) \cdot \gamma^{\sharp} = \alpha \rhd (\beta \cdot \gamma^{\sharp}).$$

Lemma *function_residual3*
 {*A B C D : eqType*} {*alpha : Rel A B*} {beta : *Rel B C*} {*gamma : Rel D C*}:
 function_r gamma $\to$ (*alpha*   beta)   *gamma* # = *alpha*   (beta   *gamma* #).
Proof.
move $\Rightarrow H$.
apply *inc_lower*.
move $\Rightarrow$ delta.
split; move $\Rightarrow H0$.
apply *inc_residual*.
rewrite -(@*function_move2* _ _ _ _ _ _ *H*).
rewrite *comp_assoc*.
apply *inc_residual*.
rewrite (@*function_move2* _ _ _ _ _ _ *H*).
apply *H0*.
rewrite -(@*function_move2* _ _ _ _ _ _ *H*).
apply *inc_residual*.
rewrite -*comp_assoc*.
rewrite (@*function_move2* _ _ _ _ _ _ *H*).
apply *inc_residual*.
apply *H0*.
Qed.

Lemma 233 (function_residual4) *Let $\alpha : A \rightharpoonup B$, $\gamma : C \rightharpoonup D$ be relations and $\beta : B \to C$ be a function. Then*

$$\alpha \cdot \beta \rhd \gamma = \alpha \rhd \beta \cdot \gamma.$$

Lemma *function_residual4*
 {*A B C D : eqType*} {*alpha : Rel A B*} {beta : *Rel B C*} {*gamma : Rel C D*}:
 function_r beta $\to$ (*alpha*   beta)   *gamma* = *alpha*   (beta   *gamma*).
Proof.
move $\Rightarrow H$.
rewrite -*double_residual*.
by [rewrite (*function_residual1 H*)].
Qed.

10.2   Galois

Lemma 234 (galois) *Let $\alpha : A \rightarrow B$, $\beta : B \rightarrow C$ and $\gamma : A \rightarrow C$. Then*

$$\gamma \sqsubseteq \alpha \rhd \beta \Leftrightarrow \alpha \sqsubseteq \gamma \rhd \beta^{\sharp}.$$

Lemma *galois* $\{A\ B\ C : eqType\}$ $\{alpha : Rel\ A\ B\}$ {beta : *Rel B C*} $\{gamma : Rel\ A\ C\}$:
 gamma   (*alpha*   beta) ↔ *alpha*   (*gamma*   beta #).
Proof.
split; move ⇒ H.
apply *inc_residual*.
apply *inv_inc_move*.
rewrite *comp_inv inv_invol*.
apply *inc_residual*.
apply H.
apply *inc_residual*.
apply *inv_inc_invol*.
rewrite *comp_inv inv_invol*.
apply *inc_residual*.
apply H.
Qed.

Lemma 235 (galois_corollary1) *Let $\alpha : A \rightarrow B$ and $\beta : B \rightarrow C$. Then*

$$\alpha \sqsubseteq (\alpha \rhd \beta) \rhd \beta^{\sharp}.$$

Lemma *galois_corollary1* $\{A\ B\ C : eqType\}$ $\{alpha : Rel\ A\ B\}$ {beta : *Rel B C*}:
 alpha   ((*alpha*   beta)   beta #).
Proof.
rewrite -*galois*.
apply *inc_refl*.
Qed.

Lemma 236 (galois_corollary2) *Let $\alpha : A \rightarrow B$ and $\beta : B \rightarrow C$. Then*

$$((\alpha \rhd \beta) \rhd \beta^{\sharp}) \rhd \beta = \alpha \rhd \beta.$$

Lemma *galois_corollary2* $\{A\ B\ C : eqType\}$ $\{alpha : Rel\ A\ B\}$ {beta : *Rel B C*}:
 ((*alpha*   beta)   beta #)   beta = *alpha*   beta.
Proof.
apply *inc_antisym*.
apply *residual_inc_compat_r*.

apply *galois_corollary1*.
move : (@*galois_corollary1* _ _ _ (*alpha*   beta) (beta #)) ⇒ H.
rewrite *inv_invol* in H.
apply H.
Qed.

Lemma 237 (galois_corollary3) *Let* $\alpha : A \rightharpoonup B$ *and* $\beta : B \rightharpoonup C$. *Then*

$$\alpha = (\alpha \rhd \beta) \rhd \beta^\sharp \Leftrightarrow \exists \gamma : A \rightharpoonup C, \alpha = \gamma \rhd \beta^\sharp.$$

Lemma *galois_corollary3* {*A B C : eqType*} {*alpha : Rel A B*} {beta : *Rel B C*}:
 alpha = (*alpha*   beta)   beta # ↔ (∃ *gamma : Rel A C*, *alpha* = *gamma*   beta #).
Proof.
split; move ⇒ H.
∃ (*alpha*   beta).
apply H.
elim H ⇒ *gamma H0*.
rewrite *H0*.
move : (@*galois_corollary2* _ _ _ *gamma* (beta #)) ⇒ H1.
rewrite *inv_invol* in H1.
by [rewrite *H1*].
Qed.

10.3

,   ,   .
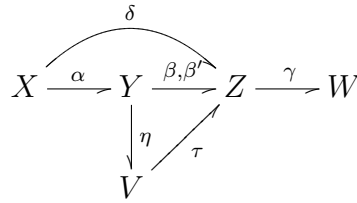
Lemma 238 (residual_property1)

$$(\alpha \rhd \beta) \cdot \gamma \sqsubseteq \alpha \rhd \beta \cdot \gamma.$$

Lemma *residual_property1*
 {*W X Y Z : eqType*} {*alpha : Rel X Y*} {beta : *Rel Y Z*} {*gamma : Rel Z W*}:
 ((*alpha*   beta)   *gamma*)   (*alpha*   (beta   *gamma*)).
Proof.

```
apply (@inc_trans _ _ _ (alpha     (alpha #     ((alpha     beta)     gamma)))).
apply inc_residual_inv.
apply residual_inc_compat_l.
rewrite -comp_assoc.
apply comp_inc_compat_ab_a'b.
apply inv_residual_inc.
Qed.
```

Lemma 239 (residual_property2)

$$(\alpha \rhd \beta) \cdot (\beta^{\sharp} \rhd \eta) \sqsubseteq \alpha \rhd \eta.$$

```
Lemma residual_property2
  {V X Y Z : eqType} {alpha : Rel X Y} {beta : Rel Y Z} {eta : Rel Y V}:
  ((alpha     beta)     (beta #     eta))     (alpha     eta).
Proof.
apply (@inc_trans _ _ _ _ _ (residual_property1)).
apply residual_inc_compat_l.
move : (@inv_residual_inc _ _ _ (beta #) eta).
by [rewrite inv_invol].
Qed.
```

Lemma 240 (residual_property3)

$$\alpha \rhd \beta \sqsubseteq \alpha \cdot \eta \rhd \eta^{\sharp} \cdot \beta.$$

```
Lemma residual_property3
  {V X Y Z : eqType} {alpha : Rel X Y} {beta : Rel Y Z} {eta : Rel Y V}:
  (alpha     beta)     ((alpha     eta)     (eta #     beta)).
Proof.
apply (@inc_trans _ _ _ _ _ (@inc_residual_inv _ _ _ (alpha     eta) (alpha     beta))).
apply residual_inc_compat_l.
rewrite comp_inv comp_assoc.
apply comp_inc_compat_ab_ab'.
apply inv_residual_inc.
Qed.
```

Lemma 241 (residual_property4a, residual_property4b)

$$(\alpha \rhd \beta) \cdot \gamma \sqsubseteq (\alpha \rhd \beta \cdot \gamma) \sqcap \nabla_{XZ} \cdot \gamma \sqsubseteq (\alpha \rhd \beta \cdot \gamma) \cdot \gamma^{\sharp} \cdot \gamma.$$

```
Lemma residual_property4a
  {W X Y Z : eqType} {alpha : Rel X Y} {beta : Rel Y Z} {gamma : Rel Z W}:
```

`((alpha` `beta)` *gamma)* `((alpha` `(beta` *gamma))* (*X Z* *gamma)).*
Proof.
`rewrite` -(@*cap_universal* _ _ (*alpha* `beta`)).
`apply` (@*inc_trans* _ _ _ _ _ (*comp_cap_distr_r*)).
`apply` *cap_inc_compat_r*.
`apply` *residual_property1*.
Qed.

Lemma *residual_property4b*
 {*W X Y Z* : *eqType*} {*alpha* : *Rel X Y*} {`beta` : *Rel Y Z*} {*gamma* : *Rel Z W*}:
 `((alpha` `(beta` *gamma))* (*X Z* *gamma))* `((alpha` `(beta` *gamma))*
(*gamma* # *gamma)).*
Proof.
`rewrite` *cap_comm*.
`apply` (@*inc_trans* _ _ _ _ _ (*dedekind2*)).
`rewrite` *cap_comm cap_universal comp_assoc*.
`apply` *inc_refl*.
Qed.

Lemma 242 (residual_property5) *Let τ be a univalent relation. Then,*

$$(\alpha \rhd \beta) \cdot \tau^{\sharp} = (\alpha \rhd \beta \cdot \tau^{\sharp}) \sqcap \nabla_{XZ} \cdot \tau^{\sharp}.$$

Lemma *residual_property5*
 {*V X Y Z* : *eqType*} {*alpha* : *Rel X Y*} {`beta` : *Rel Y Z*} {*tau* : *Rel V Z*}:
 univalent_r tau →
 (*alpha* `beta`) *tau* # = (*alpha* (`beta` *tau* #)) (*X Z* *tau* #).
Proof.
`move` ⇒ *H*.
`apply` *inc_antisym*.
`rewrite` -(@*cap_universal* _ _ (*alpha* `beta`)).
`apply` (@*inc_trans* _ _ _ _ _ (*comp_cap_distr_r*)).
`apply` *cap_inc_compat_r*.
`apply` *residual_property1*.
`rewrite` *cap_comm*.
`apply` (@*inc_trans* _ _ _ _ _ (*dedekind2*)).
`rewrite` *cap_comm cap_universal inv_invol*.
`apply` *comp_inc_compat_ab_a'b*.
`apply` (@*inc_trans* _ _ _ _ _ (*residual_property1*)).
`apply` *residual_inc_compat_l*.
`rewrite` *comp_assoc*.
`apply` (*comp_inc_compat_ab_a H*).
Qed.

Lemma 243 (residual_property6)

$$\alpha \triangleright (\gamma^\sharp \triangleright \beta^\sharp)^\sharp = (\gamma^\sharp \triangleright (\alpha \triangleright \beta)^\sharp)^\sharp.$$

Lemma *residual_property6*
 {*W X Y Z : eqType*} {*alpha : Rel X Y*} {beta *: Rel Y Z*} {*gamma : Rel Z W*}:
 alpha   (gamma #   beta #) # = (gamma #   (alpha   beta) #) #.
Proof.
apply *inc_lower*.
move ⇒ delta.
split; move ⇒ *H*.
apply *inv_inc_move*.
apply *inc_residual*.
apply *inv_inc_move*.
apply *inc_residual*.
rewrite -*comp_inv comp_assoc*.
apply *inv_inc_move*.
apply *inc_residual*.
apply *inv_inc_invol*.
rewrite *comp_inv inv_invol*.
apply *inc_residual*.
apply *H*.
apply *inc_residual*.
apply *inv_inc_move*.
apply *inc_residual*.
apply *inv_inc_move*.
rewrite *comp_inv inv_invol inv_invol comp_assoc*.
apply *inc_residual*.
apply *inv_inc_invol*.
rewrite *comp_inv*.
apply *inc_residual*.
apply *inv_inc_move*.
apply *H*.
Qed.

Lemma 244 (residual_property7a, residual_property7b)

$$\alpha \triangleright (\beta \Rightarrow \beta') \sqsubseteq (\alpha \cdot \beta \Rightarrow \alpha \cdot \beta') \sqsubseteq \alpha \triangleright (\beta \Rightarrow \alpha^\sharp \cdot \alpha \cdot \beta').$$

Lemma *residual_property7a* {*X Y Z : eqType*} {*alpha : Rel X Y*} {beta *beta' : Rel Y Z*}:
 (*alpha   (beta » beta'))   ((alpha   beta) » (alpha   beta'))*.
Proof.

apply *inc_rpc*.
rewrite *cap_comm*.
apply (@*inc_trans* _ _ _ _ _ (*dedekind1*)).
apply *comp_inc_compat_ab_ab'*.
rewrite *cap_comm*.
apply *inc_rpc*.
apply *inv_residual_inc*.
Qed.

Lemma *residual_property7b* {*X Y Z* : *eqType*} {*alpha* : *Rel X Y*} {beta *beta'* : *Rel Y Z*}:
 ((*alpha* beta) » (*alpha* *beta'*)) (*alpha* (beta » (*alpha* # (*alpha* *beta'*)))).
Proof.
rewrite *inc_residual inc_rpc*.
apply (@*inc_trans* _ _ _ _ _ (*dedekind1*)).
apply *comp_inc_compat_ab_ab'*.
rewrite *inv_invol* -*inc_rpc*.
apply *inc_refl*.
Qed.

Lemma 245 (residual_property8) *Let α be a univalent relation. Then,*

$$\alpha \rhd (\beta \Rightarrow \beta') = (\alpha \cdot \beta \Rightarrow \alpha \cdot \beta').$$

Lemma *residual_property8* {*X Y Z* : *eqType*} {*alpha* : *Rel X Y*} {beta *beta'* : *Rel Y Z*}:
 univalent_r alpha → *alpha* (beta » *beta'*) = (*alpha* beta) » (*alpha* *beta'*).
Proof.
move ⇒ *H*.
apply *inc_antisym*.
apply *residual_property7a*.
apply (@*inc_trans* _ _ _ _ _ *residual_property7b*).
apply *residual_inc_compat_l*.
apply *rpc_inc_compat_l*.
rewrite -*comp_assoc*.
apply (*comp_inc_compat_ab_b H*).
Qed.

Lemma 246 (residual_property9) *Let α be a univalent relation. Then,*

$$\alpha \rhd \beta = (\alpha \cdot \nabla_{YZ} \Rightarrow \alpha \cdot \beta).$$

Lemma *residual_property9* {*X Y Z* : *eqType*} {*alpha* : *Rel X Y*} {beta : *Rel Y Z*}:
 univalent_r alpha → *alpha* beta = (*alpha* *Y Z*) » (*alpha* beta).
Proof.
move ⇒ *H*.

by [rewrite -(*residual_property8 H*) *rpc_universal_alpha*].
Qed.

Lemma 247 (residual_property10) *Let α be a univalent relation. Then,*

$$\alpha \cdot \beta = \lfloor \alpha \rfloor \cdot (\alpha \rhd \beta).$$

Lemma *residual_property10* {*X Y Z* : *eqType*} {alpha : *Rel X Y*} {beta : *Rel Y Z*}:
  univalent_r alpha → *alpha*   beta = *domain alpha*   (*alpha*   *beta*).
Proof.
move ⇒ *H*.
apply *inc_antisym*.
replace (*alpha*   beta) with (*domain alpha*   (*alpha*   beta)).
apply *comp_inc_compat_ab_ab'*.
rewrite *inc_residual* -*comp_assoc*.
apply (*comp_inc_compat_ab_b H*).
by [rewrite -*comp_assoc domain_comp_alpha1*].
apply (@*inc_trans* _ _ _ ((*alpha*   *alpha* #)   (*alpha*   *beta*))).
apply *comp_inc_compat_ab_a'b*.
apply *cap_l*.
rewrite *comp_assoc*.
apply *comp_inc_compat_ab_ab'*.
apply *inv_residual_inc*.
Qed.

Lemma 248 (residual_property11)

$$(\alpha \cdot \beta \Rightarrow \delta) \sqsubseteq \alpha \rhd (\beta \Rightarrow \alpha^{\sharp} \cdot \delta).$$

Lemma *residual_property11*
  {*X Y Z* : *eqType*} {alpha : *Rel X Y*} {beta : *Rel Y Z*} {delta : *Rel X Z*}:
  ((*alpha*   beta) » delta)   (*alpha*   (beta » (*alpha* #   delta))).
Proof.
apply *inc_residual*.
apply *inc_rpc*.
apply (@*inc_trans* _ _ _ _ _ (*dedekind1*)).
rewrite *inv_invol*.
apply *comp_inc_compat_ab_ab'*.
apply *inc_rpc*.
apply *inc_refl*.
Qed.

Lemma 249 (residual_property12a, residual_property12b) *Let $u \sqsubseteq id_X$. Then,*

$$u \rhd \alpha = u \cdot \nabla_{XY} \Rightarrow \alpha = u \rhd u \cdot \alpha.$$

Lemma *residual_property12a* $\{X\ Y : eqType\}$ $\{u : Rel\ X\ X\}$ $\{alpha : Rel\ X\ Y\}$:
 u $Id\ X \to u$ $alpha = (u$ $X\ Y) \gg alpha$.
Proof.
move $\Rightarrow H$.
apply *inc_antisym*.
assert $(univalent\_r\ u)$.
apply (fun $H' \Rightarrow @inc\_trans$ _ _ _ _ _ $H'\ H$).
apply *comp_inc_compat_ab_b*.
rewrite -*inv_id*.
apply $(@inc\_inv$ _ _ _ _ $H)$.
rewrite $(residual\_property9\ H0)$.
apply *rpc_inc_compat_l*.
apply $(comp\_inc\_compat\_ab\_b\ H)$.
apply $(@inc\_trans$ _ _ _ _ _ $residual\_property11)$.
apply *residual_inc_compat_l*.
rewrite *rpc_universal_alpha*.
apply *comp_inc_compat_ab_b*.
rewrite -*inv_id*.
apply $(@inc\_inv$ _ _ _ _ $H)$.
Qed.

Lemma *residual_property12b* $\{X\ Y : eqType\}$ $\{u : Rel\ X\ X\}$ $\{alpha : Rel\ X\ Y\}$:
 u $Id\ X \to u$ $alpha = u$ $(u$ $alpha)$.
Proof.
move $\Rightarrow H$.
apply *inc_antisym*.
rewrite $(residual\_property12a\ H)$.
apply $(@inc\_trans$ _ _ _ _ _ $residual\_property11)$.
apply *residual_inc_compat_l*.
rewrite *rpc_universal_alpha*.
apply *comp_inc_compat_ab_a'b*.
rewrite $(dedekind\_id1\ H)$.
apply *inc_refl*.
apply *residual_inc_compat_l*.
apply $(comp\_inc\_compat\_ab\_b\ H)$.
Qed.

Lemma 250 (residual_property13)

$$(\alpha \cdot \nabla_{YZ} \sqcap \delta) \rhd \gamma = (\alpha \cdot \nabla_{YW} \Rightarrow (\delta \rhd \gamma)).$$

Lemma *residual_property13*
 {*W X Y Z : eqType*} {*alpha : Rel X Y*} {*gamma : Rel Z W*} {*delta : Rel X Z*}:
 ((*alpha*   *Y Z*)   delta)   *gamma* = (*alpha*   *Y W*) » (delta   *gamma*).
Proof.
apply *inc_antisym*.
rewrite *inc_rpc inc_residual*.
remember (((*alpha*   *Y Z*)   delta)   *gamma*) as *sigma1*.
apply (@*inc_trans* _ _ _ (((*alpha*   *Y Z*)   delta) #   *sigma1*)).
apply (@*inc_trans* _ _ _ (((*alpha*   *Y Z*)   delta) #   (*sigma1*   (*alpha*   *Y W*)))).
assert ((delta #   (*sigma1*   (*alpha*   *Y W*)))   (delta #   *sigma1*)).
apply *comp_inc_compat_ab_ab'*.
apply *cap_l*.
apply *inc_def1* in *H*.
rewrite *H*.
apply (@*inc_trans* _ _ _ _ _ (*dedekind2*)).
apply *comp_inc_compat_ab_a'b*.
rewrite (@*inv_cap_distr* _ _ _ delta) *cap_comm*.
apply *cap_inc_compat_r*.
rewrite *inv_cap_distr*.
apply (@*inc_trans* _ _ _ _ _ (*comp_cap_distr_l*)).
apply (@*inc_trans* _ _ _ _ _ (*cap_r*)).
rewrite *comp_inv comp_inv* -*comp_assoc* (@*inv_universal Y Z*).
apply *comp_inc_compat_ab_a'b*.
apply *inc_alpha_universal*.
apply *comp_inc_compat_ab_ab'*.
apply *cap_l*.
rewrite *Heqsigma1*.
apply *inc_residual*.
apply *inc_refl*.
rewrite *inc_residual*.
remember ((*alpha*   *Y W*) » (delta   *gamma*)) as *sigma2*.
apply (@*inc_trans* _ _ _ (delta #   ((*alpha*   *Y W*)   *sigma2*))).
apply (@*inc_trans* _ _ _ (((*alpha*   *Y Z*)   delta) #   ((*alpha*   *Y W*)   *sigma2*))).
assert ((((*alpha*   *Y Z*)   delta) #   *sigma2*)   (delta #   *sigma2*)).
apply *comp_inc_compat_ab_a'b*.
apply *inc_inv*.
apply *cap_r*.

```
apply inc_def1 in H.
rewrite H.
apply (@inc_trans _ _ _ _ _ (dedekind1)).
apply comp_inc_compat_ab_ab'.
rewrite cap_comm inv_invol.
apply cap_inc_compat_r.
apply (@inc_trans _ _ _ ((alpha      Y Z)    (delta #    sigma2))).
apply comp_inc_compat_ab_a'b.
apply cap_l.
rewrite comp_assoc.
apply comp_inc_compat_ab_ab'.
apply inc_alpha_universal.
apply comp_inc_compat_ab_a'b.
apply inc_inv.
apply cap_r.
rewrite Heqsigma2.
rewrite -inc_residual cap_comm -inc_rpc.
apply inc_refl.
Qed.
```

Lemma 251 (residual_property14) *Let* $\nabla_{XX} \cdot \alpha \sqsubseteq \alpha$. *Then,*

$$\nabla_{XX} \cdot (\alpha \triangleright \beta) \sqsubseteq \alpha \triangleright \beta.$$

```
Lemma residual_property14 {X Y Z : eqType} {alpha : Rel X Y} {beta : Rel Y Z}:
(   X X    alpha)   alpha → (   X X    (alpha    beta))   (alpha    beta).
Proof.
move ⇒ H.
apply (@inc_trans _ _ _ (   X X    (   X X    (alpha    beta)))).
apply comp_inc_compat_ab_ab'.
rewrite double_residual.
apply (residual_inc_compat_r H).
rewrite -inv_universal -inc_residual inv_universal.
apply inc_refl.
Qed.
```

Lemma 252 (residual_property15) *Let* $\beta \cdot \nabla_{ZZ} \sqsubseteq \beta$. *Then,*

$$(\alpha \triangleright \beta) \cdot \nabla_{ZZ} \sqsubseteq \alpha \triangleright \beta.$$

```
Lemma residual_property15 {X Y Z : eqType} {alpha : Rel X Y} {beta : Rel Y Z}:
(beta      Z Z)   beta → ((alpha    beta)      Z Z)   (alpha    beta).
Proof.
```

```
move ⇒ H.
apply (@inc_trans _ _ _ _ _ (residual_property1)).
apply (residual_inc_compat_l H).
Qed.
```

Lemma 253 (residual_property16)

$$id_X \sqsubseteq \alpha \triangleright \alpha^\sharp \wedge (\alpha \triangleright \alpha^\sharp) \cdot (\alpha \triangleright \alpha^\sharp) \sqsubseteq \alpha \triangleright \alpha^\sharp.$$

```
Lemma residual_property16 {X Y : eqType} {alpha : Rel X Y}:
 Id X     (alpha     alpha #) ∧
 ((alpha     alpha #)     (alpha     alpha #))     (alpha     alpha #).
Proof.
split.
rewrite inc_residual comp_id_r.
apply inc_refl.
move : (@residual_property2 _ _ _ _ alpha (alpha #) (alpha #)) ⇒ H.
rewrite inv_invol in H.
apply H.
Qed.
```

Lemma 254 (residual_property17) *Let $P(y) :=$ "$y : I \to Y$ is a function". Then,*

$$\sqcup_{P(y)} y^\sharp \cdot y = id_Y \Rightarrow \alpha \triangleright \beta = \sqcap_{P(y)}(\alpha \cdot y^\sharp \cdot \nabla_{IZ} \Rightarrow \alpha \cdot y^\sharp \cdot y \cdot \beta).$$

```
Lemma residual_property17 {X Y Z : eqType}
 {alpha : Rel X Y} {beta : Rel Y Z} {P : Rel i Y → Prop}:
 P = (fun y : Rel i Y ⇒ function_r y) →
   _{P} (fun y : Rel i Y ⇒ y #     y) = Id Y →
 alpha     beta =     _{P} (fun y : Rel i Y ⇒
 ((alpha     y #)          i Z) » ((alpha     y #)     (y     beta))).
Proof.
move ⇒ H H0.
replace (alpha     beta) with ((alpha     Id Y)     beta).
rewrite -H0 comp_cupP_distr_l residual_cupP_distr_r.
apply capP_eq.
move ⇒ y H1.
rewrite H in H1.
rewrite -comp_assoc (function_residual4 H1).
apply residual_property9.
rewrite /univalent_r.
rewrite unit_identity_is_universal.
apply inc_alpha_universal.
```

```
by [rewrite comp_id_r].
Qed.
```

10.4

10.4.1 max, sup, min, inf

$\xi : X \rightharpoonup X$ X $,$ $\rho : V \rightharpoonup X$ (max),
(sup), (min), (inf) $,$.

- $max(\rho, \xi) := \rho \sqcap (\rho \triangleright \xi)$

- $sup(\rho, \xi) := (\rho \triangleright \xi) \sqcap ((\rho \triangleright \xi) \triangleright \xi^{\sharp})$

- $min(\rho, \xi) := \rho \sqcap (\rho \triangleright \xi^{\sharp}) (= max(\rho, \xi^{\sharp}))$

- $inf(\rho, \xi) := (\rho \triangleright \xi^{\sharp}) \sqcap ((\rho \triangleright \xi^{\sharp}) \triangleright \xi) (= sup(\rho, \xi^{\sharp}))$

```
Definition max {V X : eqType} (rho : Rel V X) (xi : Rel X X)
 := rho     (rho     xi).
Definition sup {V X : eqType} (rho : Rel V X) (xi : Rel X X)
 := (rho     xi)     ((rho     xi)     xi #).
Definition min {V X : eqType} (rho : Rel V X) (xi : Rel X X)
 := rho     (rho     xi #).
Definition inf {V X : eqType} (rho : Rel V X) (xi : Rel X X)
 := (rho     xi #)     ((rho     xi #)     xi).
```

Lemma 255 (max_inc_sup) *Let* $\rho : V \rightharpoonup X$ *and* $\xi : X \rightharpoonup X$. *Then,*

$$max(\rho, \xi) \sqsubseteq sup(\rho, \xi).$$

```
Lemma max_inc_sup {V X : eqType} {rho : Rel V X} {xi : Rel X X}:
 max rho xi     sup rho xi.
Proof.
rewrite /max /sup.
rewrite cap_comm.
apply cap_inc_compat_l.
apply galois_corollary1.
Qed.
```

Lemma 256 (min_inc_inf) *Let $\rho : V \rightharpoonup X$ and $\xi : X \rightharpoonup X$. Then,*

$$min(\rho, \xi) \sqsubseteq inf(\rho, \xi).$$

Lemma *min_inc_inf* $\{V\ X : eqType\}$ $\{rho : Rel\ V\ X\}$ $\{xi : Rel\ X\ X\}$:
 min rho xi   inf rho xi.
Proof.
rewrite $/min/inf$.
rewrite *cap_comm*.
apply *cap_inc_compat_l*.
move : $(@galois\_corollary1\ \_\ \_\ \_\ rho\ (xi\ \#)) \Rightarrow H$.
rewrite *inv_invol* in H.
apply H.
Qed.

Lemma 257 (inf_to_sup) *Let $\rho : V \rightharpoonup X$ and $\xi : X \rightharpoonup X$. Then,*

$$inf(\rho, \xi) = sup(\rho \triangleright \xi^{\sharp}, \xi).$$

Lemma *inf_to_sup* $\{V\ X : eqType\}$ $\{rho : Rel\ V\ X\}$ $\{xi : Rel\ X\ X\}$:
 inf rho xi = sup (rho   xi $\#$) xi.
Proof.
rewrite $/sup/inf$.
rewrite *cap_comm*.
move : $(@galois\_corollary2\ \_\ \_\ \_\ rho\ (xi\ \#)) \Rightarrow H$.
rewrite *inv_invol* in H.
by [rewrite H].
Qed.

Lemma 258 (sup_to_inf) *Let $\rho : V \rightharpoonup X$ and $\xi : X \rightharpoonup X$. Then,*

$$sup(\rho, \xi) = inf(\rho \triangleright \xi, \xi).$$

Lemma *sup_to_inf* $\{V\ X : eqType\}$ $\{rho : Rel\ V\ X\}$ $\{xi : Rel\ X\ X\}$:
 sup rho xi = inf (rho   xi) xi.
Proof.
rewrite $/sup/inf$.
rewrite *cap_comm*.
by [rewrite *galois_corollary2*].
Qed.

172

> **Lemma 259 (residual_inc_sup1, residual_inc_sup2)** *Let $\rho : V \rightharpoonup X$ and $\xi : X \rightharpoonup X$. Then,*
> $$sup(\rho, \xi) \sqsubseteq \rho \rhd \xi \sqsubseteq sup(\rho, \xi) \rhd \xi.$$

Lemma $residual\_inc\_sup1$ $\{V\ X : eqType\}$ $\{rho : Rel\ V\ X\}$ $\{xi : Rel\ X\ X\}$:
 $sup\ rho\ xi\ \ (rho\ \ \ xi)$.
Proof.
apply cap_l.
Qed.

Lemma $residual\_inc\_sup2$ $\{V\ X : eqType\}$ $\{rho : Rel\ V\ X\}$ $\{xi : Rel\ X\ X\}$:
 $(rho\ \ \ xi)\ \ \ ((sup\ rho\ xi)\ \ \ xi)$.
Proof.
rewrite $galois$.
apply cap_r.
Qed.

> **Lemma 260 (max_inc_xi_cap)** *Let $\rho : V \rightharpoonup X$ and $\xi : X \rightharpoonup X$. Then,*
> $$(max(\rho, \xi))^{\sharp} \cdot max(\rho, \xi) \sqsubseteq \xi \sqcap \xi^{\sharp}.$$

Lemma $max\_inc\_xi\_cap$ $\{V\ X : eqType\}$ $\{rho : Rel\ V\ X\}$ $\{xi : Rel\ X\ X\}$:
 $(max\ rho\ xi\ \#\ \ \ max\ rho\ xi)\ \ \ (xi\ \ \ xi\ \#)$.
Proof.
rewrite $/max$.
rewrite inv_cap_distr.
apply $(@inc\_trans\ \_\ \_\ \_\ \_\ \_\ (comp\_cap\_distr\_r))$.
apply cap_inc_compat.
apply $inc\_residual$.
apply cap_r.
apply inv_inc_move.
rewrite $comp\_inv\ inv\_invol$.
apply $inc\_residual$.
apply $residual\_inc\_compat\_r$.
apply cap_l.
Qed.

> **Lemma 261 (sup_inc_xi_cap)** *Let $\rho : V \rightharpoonup X$ and $\xi : X \rightharpoonup X$. Then,*
> $$(sup(\rho, \xi))^{\sharp} \cdot sup(\rho, \xi) \sqsubseteq \xi \sqcap \xi^{\sharp}.$$

Lemma $sup\_inc\_xi\_cap$ $\{V\ X : eqType\}$ $\{rho : Rel\ V\ X\}$ $\{xi : Rel\ X\ X\}$:
 $(sup\ rho\ xi\ \#\ \ \ sup\ rho\ xi)\ \ \ (xi\ \ \ xi\ \#)$.

Proof.
move : (@*max_inc_xi_cap* _ _ (*rho*   *xi*) (*xi* #)).
rewrite /*max*/*sup*.
by [rewrite *inv_invol* (@*cap_comm* _ _ *xi*)].
Qed.

Lemma 262 (transitive_sup1) *Let* $\rho : V \to X$, $\xi : X \to X$ *and* $\xi \cdot \xi \sqsubseteq \xi$. *Then,*

$$sup(\rho, \xi) \cdot (\xi \sqcap \xi^{\sharp}) = sup(\rho, \xi).$$

Lemma *transitive_sup1* {*V X* : *eqType*} {*rho* : *Rel V X*} {*xi* : *Rel X X*}:
(*xi*   *xi*)   *xi* → *sup rho xi*   (*xi*   *xi* #) = *sup rho xi*.
Proof.
move ⇒ *H*.
apply *inc_antisym*.
rewrite /*sup*.
apply (@*inc_trans* _ _ _ _ _ (*comp_cap_distr_l*)).
apply *cap_inc_compat*.
apply (@*inc_trans* _ _ _ _ _ (*comp_cap_distr_r*)).
apply (@*inc_trans* _ _ _ _ _ (*cap_l*)).
apply (@*inc_trans* _ _ _ _ _ (*residual_property1*)).
apply (*residual_inc_compat_l H*).
apply (@*inc_trans* _ _ _ _ _ (*comp_cap_distr_r*)).
apply (@*inc_trans* _ _ _ _ _ (*cap_r*)).
apply (@*inc_trans* _ _ _ _ _ (*residual_property1*)).
apply *residual_inc_compat_l*.
rewrite -*comp_inv inv_inc_move inv_invol*.
apply *H*.
apply (@*inc_trans* _ _ _ _ _ (*relation_rel_inv_rel*)).
rewrite *comp_assoc*.
apply (*comp_inc_compat_ab_ab' sup_inc_xi_cap*).
Qed.

Lemma 263 (transitive_sup2) *Let* $\rho : V \to X$, $\xi : X \to X$ *and* $\xi \cdot \xi \sqsubseteq \xi$. *Then,*

$$sup(\rho, \xi) \cdot \xi = \lfloor sup(\rho, \xi) \rfloor \cdot (\rho \rhd \xi).$$

Lemma *transitive_sup2* {*V X* : *eqType*} {*rho* : *Rel V X*} {*xi* : *Rel X X*}:
(*xi*   *xi*)   *xi* → *sup rho xi*   *xi* = *domain* (*sup rho xi*)   (*rho*   *xi*).
Proof.
move ⇒ *H*.
apply *inc_antisym*.
replace (*sup rho xi*   *xi*) with (*domain* (*sup rho xi*)   (*sup rho xi*   *xi*)).

apply *comp_inc_compat_ab_ab'*.
apply (@*inc_trans* _ _ _ ((*rho*   *xi*)   *xi*)).
apply (*comp_inc_compat_ab_a'b cap_l*).
apply (@*inc_trans* _ _ _ _ _ (*residual_property1*) (*residual_inc_compat_l H*)).
by [rewrite -*comp_assoc domain_comp_alpha1*].
apply (@*inc_trans* _ _ _ (*domain* (*sup rho xi*)   (*sup rho xi*   *xi*))).
apply *comp_inc_compat_ab_ab'*.
apply *galois*.
apply *cap_r*.
rewrite /*domain*.
apply (@*inc_trans* _ _ _ _ _ (*comp_cap_distr_r*)).
apply (@*inc_trans* _ _ _ _ _ (*cap_l*)).
rewrite *comp_assoc*.
apply *comp_inc_compat_ab_ab'*.
apply *inc_residual*.
apply *inc_refl*.
Qed.

Lemma 264 (domain_sup_inc) *Let $\rho : V \rightharpoonup X$ and $\xi : X \rightharpoonup X$. Then,*

$$\lfloor sup(\rho, \xi) \rfloor \cdot \rho \sqsubseteq sup(\rho, \xi) \cdot \xi^{\sharp}.$$

Lemma *domain_sup_inc* {*V X* : *eqType*} {*rho* : *Rel V X*} {*xi* : *Rel X X*}:
(*domain* (*sup rho xi*)   *rho*)   (*sup rho xi*   *xi* #).
Proof.
apply (@*inc_trans* _ _ _ (*domain* (*sup rho xi*)   (*sup rho xi*   *xi* #))).
apply *comp_inc_compat_ab_ab'*.
rewrite -*galois*.
apply *cap_l*.
rewrite /*domain*.
apply (@*inc_trans* _ _ _ _ _ (*comp_cap_distr_r*)).
apply (@*inc_trans* _ _ _ _ _ (*cap_l*)).
rewrite *comp_assoc*.
apply *comp_inc_compat_ab_ab'*.
apply *inc_residual*.
apply *inc_refl*.
Qed.

Lemma 265 (sup_function) *Let $\rho : V \rightharpoonup X$, $\xi : X \rightharpoonup X$ be relations and $f : W \rightarrow V$ be a function. Then,*
$$f \cdot sup(\rho, \xi) = sup(f \cdot \rho, \xi).$$

Lemma *sup_function* {*V W X* : *eqType*} {*rho* : *Rel V X*} {*xi* : *Rel X X*} {*f* : *Rel W V*}:

175

function_r f → f   sup rho xi = sup (f   rho) xi.
Proof.
move ⇒ *H.*
rewrite /*sup.*
rewrite (*function_cap_distr_l H*).
by [rewrite (*function_residual2 H*) (*function_residual2 H*) (*function_residual2 H*)].
Qed.

Lemma 266 (max_univalent) *Let $\rho : V \rightarrow X$, $\xi : X \rightarrow X$ be relations and $\varphi : W \rightarrow V$ be a univalent relation. Then,*

$$\varphi \cdot max(\rho, \xi) = max(\varphi \cdot \rho, \xi).$$

Lemma *max_univalent* {*V W X : eqType*}
 {*rho : Rel V X*} {*xi : Rel X X*} {*phi : Rel W V*}:
 univalent_r phi → phi   max rho xi = max (phi   rho) xi.
Proof.
move ⇒ *H.*
rewrite /*max.*
apply *inc_antisym.*
apply (@*inc_trans* _ _ _ _ _ (*comp_cap_distr_l*)).
apply *cap_inc_compat_l.*
apply (@*inc_trans* _ _ _ _ _ (*univalent_residual H*)).
rewrite *double_residual.*
apply *inc_refl.*
apply (@*inc_trans* _ _ _ _ _ (*dedekind1*)).
apply *comp_inc_compat_ab_ab'.*
apply *cap_inc_compat_l.*
rewrite -*inc_residual double_residual.*
apply *inc_refl.*
Qed.

10.4.2

$$\alpha : X \rightarrow Y, \beta : Y \rightarrow Z \qquad , \qquad \alpha \lhd \beta := (\beta^\sharp \rhd \alpha^\sharp)^\sharp \qquad .$$

Definition *leftres* {*X Y Z : eqType*} (*alpha : Rel X Y*) (beta : *Rel Y Z*)
 := (beta #   *alpha #*) #.

Lemma 267 (inc_leftres) *Let* $\alpha : X \to Y$, $\beta : Y \to Z$ *and* $\delta : X \to Z$*. Then,*

$$\delta \sqsubseteq \alpha \lhd \beta \Leftrightarrow \delta \cdot \beta^{\sharp} \sqsubseteq \alpha.$$

Lemma *inc_leftres* {*X Y Z : eqType*}
 {*alpha : Rel X Y*} {beta : *Rel Y Z*} {delta : *Rel X Z*}:
 delta   *leftres alpha* beta $\leftrightarrow$ (delta   beta #)   *alpha*.
Proof.
rewrite /*leftres*.
by [rewrite *inv_inc_move inc_residual -comp_inv inv_inc_move inv_invol*].
Qed.

Lemma 268 (residual_leftres_assoc) *Let* $\alpha : X \to Y$, $\beta : Y \to Z$ *and* $\gamma : Z \to W$*. Then,*

$$(\alpha \rhd \beta) \lhd \gamma = \alpha \rhd (\beta \lhd \gamma).$$

Lemma *residual_leftres_assoc* {*W X Y Z : eqType*}
 {*alpha : Rel X Y*} {beta : *Rel Y Z*} {gamma : *Rel Z W*}:
 leftres (*alpha*   beta) *gamma* = *alpha*   *leftres* beta *gamma*.
Proof.
apply *inc_lower*.
move $\Rightarrow$ delta.
by [rewrite *inc_leftres inc_residual -comp_assoc -inc_leftres -inc_residual*].
Qed.

End *main*.

Chapter 11

Library Schroder

Require Import *Basic_Notations*.
Require Import *Basic_Lemmas*.
Require Import *Relation_Properties*.
Require Import *Functions_Mappings*.
Require Import *Dedekind*.
Require Import *Residual*.
Require Import *Logic.FunctionalExtensionality*.

Module *main* (*def* : *Relation*).
Import *def*.
Module *Basic_Lemmas* := *Basic_Lemmas.main def*.
Module *Relation_Properties* := *Relation_Properties.main def*.
Module *Functions_Mappings* := *Functions_Mappings.main def*.
Module *Dedekind* := *Dedekind.main def*.
Module *Residual* := *Residual.main def*.
Import *Basic_Lemmas Relation_Properties Functions_Mappings Dedekind Residual*.

11.1   Schröder

$$X \xrightarrow{\alpha} Y \underset{\beta,\beta',\beta_\lambda}{\xrightarrow{\hspace{2em}}} Z \xrightarrow{\gamma} W$$

with δ above and ρ on $I \to X$, τ on $V \to Z$.

Lemma 269 (schroder_equivalence1, schroder_equivalence2)

$$\alpha \cdot \beta \sqsubseteq \delta \Leftrightarrow \alpha^{\sharp} \cdot \delta^{-} \sqsubseteq \beta^{-} \Leftrightarrow \delta^{-} \cdot \beta^{\sharp} \sqsubseteq \alpha^{-}.$$

Lemma *schroder_equivalence1*
 $\{X\ Y\ Z : eqType\}$ $\{alpha : Rel\ X\ Y\}$ $\{$beta $: Rel\ Y\ Z\}$ $\{$delta $: Rel\ X\ Z\}$:
 $(alpha$ beta$)$ delta $\leftrightarrow (alpha\ \#$ delta ˆ$)$ beta ˆ.
Proof.
split; move $\Rightarrow H$.
rewrite *bool_lemma2 complement_invol*.
apply *inc_antisym*.
apply $(@inc\_trans$ _ _ _ _ _ $(dedekind1))$.
apply *bool_lemma2* in H.
rewrite *cap_comm inv_invol* H *comp_empty_r*.
apply *inc_refl*.
apply *inc_empty_alpha*.
rewrite *bool_lemma2*.
apply *inc_antisym*.
apply $(@inc\_trans$ _ _ _ _ _ $(dedekind1))$.
apply *bool_lemma2* in H.
rewrite *cap_comm* -$(@complement\_invol$ _ _ beta$)$ H *comp_empty_r*.
apply *inc_refl*.
apply *inc_empty_alpha*.
Qed.

Lemma *schroder_equivalence2*
 $\{X\ Y\ Z : eqType\}$ $\{alpha : Rel\ X\ Y\}$ $\{$beta $: Rel\ Y\ Z\}$ $\{$delta $: Rel\ X\ Z\}$:
 $(alpha$ beta$)$ delta $\leftrightarrow (delta$ ˆ beta $\#)$ alpha ˆ.
Proof.
split; move $\Rightarrow H$.
rewrite *bool_lemma2 complement_invol*.
apply *inc_antisym*.
apply $(@inc\_trans$ _ _ _ _ _ $(dedekind2))$.
apply *bool_lemma2* in H.
rewrite *cap_comm inv_invol* H *comp_empty_l*.
apply *inc_refl*.
apply *inc_empty_alpha*.
rewrite *bool_lemma2*.
apply *inc_antisym*.
apply $(@inc\_trans$ _ _ _ _ _ $(dedekind2))$.
apply *bool_lemma2* in H.
rewrite *cap_comm* -$(@complement\_invol$ _ _ alpha$)$ H *comp_empty_l*.
apply *inc_refl*.

```
apply inc_empty_alpha.
Qed.
```

Lemma 270 (function_inv_complement) *Let α and τ be functions. Then,*

$$(\alpha \cdot \beta \cdot \tau^{\sharp})^{-} = \alpha \cdot \beta^{-} \cdot \tau^{\sharp}.$$

```
Lemma function_inv_complement
 {V X Y Z : eqType} {alpha : Rel X Y} {beta : Rel Y Z} {tau : Rel V Z}:
 function_r alpha → function_r tau →
 ((alpha    beta)    tau #) ^ = (alpha    beta ^)    tau #.
Proof.
move ⇒ H H0.
apply inc_antisym.
rewrite bool_lemma1 complement_invol.
apply inc_antisym.
rewrite -comp_cup_distr_r -comp_cup_distr_l complement_classic.
apply (@inc_trans _ _ _ (((alpha    alpha #)    X V)    (tau    tau #))).
apply (@inc_trans _ _ _ ((alpha    alpha #)    X V)).
apply comp_inc_compat_b_ab.
apply H.
apply comp_inc_compat_a_ab.
apply H0.
rewrite -comp_assoc (@comp_assoc _ _ _ _ alpha) (@comp_assoc _ _ _ _ alpha).
apply comp_inc_compat_ab_a'b.
apply comp_inc_compat_ab_ab'.
apply inc_alpha_universal.
apply inc_alpha_universal.
rewrite bool_lemma2 complement_invol.
apply inc_antisym.
rewrite -(function_cap_distr H H0) cap_comm cap_complement_empty comp_empty_r comp_empty_l.
apply inc_refl.
apply inc_empty_alpha.
Qed.
```

Lemma 271 (schroder_univalent1) *Let α be a univalent relation and $\beta \sqsubseteq \beta'$. Then,*

$$\alpha \cdot (\beta' \sqcap \beta^{-}) = \alpha \cdot \beta' \sqcap (\alpha \cdot \beta)^{-}.$$

```
Lemma schroder_univalent1
 {X Y Z : eqType} {alpha : Rel X Y} {beta beta' : Rel Y Z}:
 univalent_r alpha → beta    beta' →
 alpha    (beta'    beta ^) = (alpha    beta')    (alpha    beta) ^.
```

Proof.
move ⇒ *H H0*.
apply (@*cap_cup_unique* _ _ (*alpha* ⊓ *beta*)).
replace ((*alpha* ⊓ *beta*) ⊔ (*alpha* ⊓ (*beta'* ⊔ *beta* ^))) with (⊤ X Z).
rewrite (@*cap_comm* _ _ (*alpha* ⊓ *beta'*)) -*cap_assoc*.
by [rewrite *cap_complement_empty cap_comm cap_empty*].
apply *inc_antisym*.
apply *inc_empty_alpha*.
apply (@*inc_trans* _ _ _ ((*alpha* ⊓ *beta*) ⊓ ((*alpha* ⊓ *beta'*) ⊔ (*alpha* ⊓ *beta* ^)))).
apply *cap_inc_compat_l*.
apply *comp_cap_distr_l*.
replace (⊤ X Z) with ((*alpha* ⊓ *beta*) ⊔ (*alpha* ⊓ *beta* ^)).
apply *cap_inc_compat_l*.
apply *cap_r*.
apply *inc_antisym*.
move : (@*univalent_residual* _ _ _ _ beta *H*) ⇒ *H1*.
rewrite -*inc_rpc*.
rewrite *residual_to_complement* in *H1*.
apply *H1*.
apply *inc_empty_alpha*.
apply *inc_def2* in *H0*.
rewrite -*comp_cup_distr_l cup_cap_distr_l*.
rewrite -*H0 complement_classic cap_universal*.
rewrite *cup_cap_distr_l* -*comp_cup_distr_l*.
by [rewrite -*H0 complement_classic cap_universal*].
Qed.

Lemma 272 (schroder_univalent2) *Let* α *be a univalent relation. Then,*

$$\alpha \cdot \beta^- = \alpha \cdot \nabla_{YZ} \sqcap (\alpha \cdot \beta)^-.$$

Lemma *schroder_univalent2* {*X Y Z* : *eqType*} {*alpha* : *Rel X Y*} {beta : *Rel Y Z*}:
 univalent_r alpha → *alpha* ⋅ beta ^ = (*alpha* ⋅ ∇ *Y Z*) ⊓ (*alpha* ⋅ beta) ^.
Proof.
move ⇒ *H*.
move : (@*schroder_univalent1* _ _ _ *alpha* beta (∇ *Y Z*) *H* (@*inc_alpha_universal* _ _ _))
 ⇒ *H0*.
rewrite *cap_comm cap_universal* in *H0*.
apply *H0*.
Qed.

Lemma 273 (schroder_univalent3) *Let α be a univalent relation. Then,*

$$(\alpha \cdot \beta)^- = (\alpha \cdot \nabla_{YZ})^- \sqcup \alpha \cdot \beta^-.$$

Lemma *schroder_univalent3* $\{X\ Y\ Z : eqType\}$ $\{alpha : Rel\ X\ Y\}$ $\{$beta $: Rel\ Y\ Z\}$:
 univalent_r alpha $\rightarrow$ (*alpha* beta) ^ = (*alpha* *Y Z*) ^ (*alpha* beta ^).
Proof.
move $\Rightarrow H$.
rewrite (*schroder_univalent2 H*).
rewrite *cup_cap_distr_l cup_comm complement_classic cap_comm cap_universal*.
apply *inc_def2*.
apply *rpc_inc_compat_r*.
apply *comp_inc_compat_ab_ab'*.
apply *inc_alpha_universal*.
Qed.

Lemma 274 (schroder_univalent4) *Let α be a univalent relation. Then,*

$$\alpha \rhd \beta = (\alpha \cdot \nabla_{YZ})^- \sqcup \alpha \cdot \beta.$$

Lemma *schroder_univalent4* $\{X\ Y\ Z : eqType\}$ $\{alpha : Rel\ X\ Y\}$ $\{$beta $: Rel\ Y\ Z\}$:
 univalent_r alpha $\rightarrow$ *alpha* beta = (*alpha* *Y Z*) ^ (*alpha* beta).
Proof.
move $\Rightarrow H$.
rewrite (*residual_property9 H*).
apply *Logic.eq_sym*.
apply *cup_to_rpc*.
Qed.

Lemma 275 (schroder_universal) *Let $\nabla_{XZ} \cdot \nabla_{ZW} = \nabla_{XW}$. Then,*

$$(\alpha \cdot \nabla_{YZ})^- \cdot \nabla_{ZW} = (\alpha \cdot \nabla_{YW})^-.$$

Lemma *schroder_universal* $\{W\ X\ Y\ Z : eqType\}$ $\{alpha : Rel\ X\ Y\}$:
(*X Z* *Z W*) = *X W* $\rightarrow$
(*alpha* *Y Z*) ^ *Z W* = (*alpha* *Y W*) ^.
Proof.
move $\Rightarrow H$.
apply (@*cap_cup_unique* _ _ (*alpha* *Y W*)).
rewrite *cap_complement_empty cap_comm*.
apply *inc_antisym*.
apply (@*inc_trans* _ _ _ _ _ (*dedekind2*)).
apply (@*inc_trans* _ _ _ (((*alpha* *Y Z*) ^ (*alpha* *Y Z*)) *Z W*)).

182

apply *comp_inc_compat_ab_a'b*.
apply *cap_inc_compat_l*.
rewrite *comp_assoc*.
apply *comp_inc_compat_ab_ab'*.
apply *inc_alpha_universal*.
rewrite *cap_comm cap_complement_empty comp_empty_l*.
apply *inc_refl*.
apply *inc_empty_alpha*.
rewrite *complement_classic*.
apply *inc_antisym*.
apply *inc_alpha_universal*.
rewrite -*H* -(@*complement_classic* _ _ (*alpha* Y Z)) *comp_cup_distr_r*.
apply *cup_inc_compat_r*.
rewrite *comp_assoc*.
apply *comp_inc_compat_ab_ab'*.
apply *inc_alpha_universal*.
Qed.

Lemma 276 (residual_inv)

$$(\alpha \rhd \beta)^{\sharp} = \beta^{-\sharp} \rhd \alpha^{-\sharp}.$$

Lemma *residual_inv* {*X Y Z* : *eqType*} {*alpha* : *Rel X Y*} {beta : *Rel Y Z*}:
(*alpha* beta) # = (beta ^) # (*alpha* ^) #.
Proof.
rewrite *residual_to_complement residual_to_complement*.
by [rewrite -*inv_complement complement_invol inv_complement comp_inv*].
Qed.

Lemma 277 (residual_cupP_distr_l, residual_cup_distr_l) *Let* α *be a univalent relation,* $f : (V \to W) \to (Y \to Z)$ *and* $\exists \beta, P(\beta)$. *Then,*

$$\alpha \rhd (\sqcup_{P(\beta)} f(\beta)) = \sqcup_{P(\beta)} (\alpha \rhd f(\beta)).$$

Lemma *residual_cupP_distr_l* {*V W X Y Z* : *eqType*}
{*alpha* : *Rel X Y*} {*f* : *Rel V W* → *Rel Y Z*} {*P* : *Rel V W* → Prop}:
univalent_r alpha → (∃ *beta'* : *Rel V W, P beta'*) →
alpha (_{*P*} *f*) = _{*P*} (fun beta : *Rel V W* ⇒ *alpha* *f* beta).
Proof.
move ⇒ *H*.
elim ⇒ *beta' H0*.
rewrite (*schroder_univalent4 H*) *comp_cupP_distr_l*.
replace (_{*P*} (fun beta : *Rel V W* ⇒ *alpha* *f* beta)) with (_{*P*} (fun beta :

Rel V W ⇒ (*alpha*   *Y Z*) ^   (*alpha*   *f* beta))).
`apply` (@*cap_cup_unique* _ _ (*alpha*   *Y Z*)).
`rewrite` *cap_cup_distr_l cap_cupP_distr_l cap_complement_empty cup_comm cup_empty.*
`rewrite` *cap_cupP_distr_l.*
`apply` *cupP_eq.*
`move` ⇒ *gamma H1.*
`by` [`rewrite` *cap_cup_distr_l cap_complement_empty cup_comm cup_empty*].
`rewrite` -*cup_assoc complement_classic cup_comm cup_universal.*
`rewrite` -(@*complement_invol* _ _ (*alpha*   *Y Z*)).
`apply` *bool_lemma1.*
`rewrite` *complement_invol.*
`apply` (@*inc_trans* _ _ _ ((*alpha*   *Y Z*) ^   (*alpha*   *f beta'*))).
`apply` *cup_l.*
`move` : *beta' H0.*
`apply` *inc_cupP.*
`apply` *inc_refl.*
`apply` *cupP_eq.*
`move` ⇒ *gamma H1.*
`by` [`rewrite` (*schroder_univalent4 H*)].
`Qed`.

`Lemma` *residual_cup_distr_l*
 {*X Y Z* : *eqType*} {*alpha* : *Rel X Y*} {beta *beta'* : *Rel Y Z*}:
 univalent_r alpha →
 alpha   (beta   *beta'*) = (*alpha*   beta)   (*alpha*   *beta'*).
`Proof`.
`move` ⇒ *H*.
`rewrite` *cup_to_cupP* (@*cup_to_cupP* _ _ _ _ _ _ *id*).
`apply` (*residual_cupP_distr_l H*).
∃ beta.
`by` [`left`].
`Qed`.

Lemma 278 (residual_capP_distr_r, residual_cap_distr_r) *Let* $f : (Y \rightarrow Z) \rightarrow (I \rightarrow X)$ *and* $\exists \alpha, P(\alpha)$. *Then,*

$$(\sqcap_{P(\alpha)} f(\alpha)^{\sharp}) \rhd \rho = \sqcup_{P(\alpha)} (f(\alpha)^{\sharp} \rhd \rho).$$

`Lemma` *residual_capP_distr_r*
 {*X Y Z* : *eqType*} {*rho* : *Rel i X*} {*f* : *Rel Y Z* → *Rel i X*} {*P* : *Rel Y Z* → `Prop`}:
 (∃ *alpha'* : *Rel Y Z, P alpha'*) →
 (_{*P*} (`fun` *alpha* : *Rel Y Z* ⇒ *f alpha* #))   *rho* =   _{*P*} (`fun` *alpha* : *Rel Y Z* ⇒ *f alpha* #   *rho*).

Proof.
elim ⇒ *alpha' H*.
rewrite *residual_to_complement*.
rewrite -(@*complement_invol* _ _ (_{P} (fun *alpha* : *Rel Y Z* ⇒ *f alpha* # *rho*))).
apply f_equal.
rewrite *de_morgan3*.
replace (fun *alpha* : *Rel Y Z* ⇒ (*f alpha* # *rho*) ˆ) with (fun *alpha* : *Rel Y Z* ⇒ *f alpha* # *rho* ˆ).
apply *inc_antisym*.
apply *comp_capP_distr_r*.
apply (@*inc_trans* _ _ _ _ _ (*relation_rel_inv_rel*)).
apply (@*inc_trans* _ _ _ (((_{P} (fun *alpha* : *Rel Y Z* ⇒ *f alpha* # *rho* ˆ)) (*f alpha'* # *rho* ˆ) #) (*f alpha'* # *rho* ˆ))).
apply *comp_inc_compat*.
apply *comp_inc_compat_ab_ab'*.
move : *alpha' H*.
apply *inc_capP*.
rewrite *inv_capP_distr*.
apply *inc_refl*.
move : *alpha' H*.
apply *inc_capP*.
apply *inc_refl*.
rewrite -*comp_assoc*.
apply *comp_inc_compat_ab_a'b*.
rewrite *comp_assoc*.
apply (@*inc_trans* _ _ _ _ _ (*comp_capP_distr_r*)).
apply *inc_capP*.
move ⇒ beta *H0*.
apply (@*inc_trans* _ _ _ ((*f* beta # *rho* ˆ) ((*f alpha'* # *rho* ˆ) # *f alpha'* #))).
move : beta *H0*.
apply *inc_capP*.
apply *inc_refl*.
rewrite *comp_assoc*.
apply *comp_inc_compat_ab_a*.
rewrite *unit_identity_is_universal*.
apply *inc_alpha_universal*.
apply *functional_extensionality*.
move ⇒ *x*.
by [rewrite *residual_to_complement complement_invol*].
Qed.

End *main*.

Chapter 12

Library Sum_Product

12.1

12.1.1   ,

> (Axiom 23)   ,
> $j : A \rightharpoonup A + B, k : B \rightharpoonup A + B$.

Definition *sum_r* (*A B* : *eqType*):
 $\{x : (Rel \ A \ (sum\_eqType \ A \ B)) \times (Rel \ B \ (sum\_eqType \ A \ B)) \mid$

$(fst\ x)\quad (fst\ x)\ \#\ =\ Id\ A \wedge (snd\ x)\quad (snd\ x)\ \#\ =\ Id\ B\ \wedge$
$(fst\ x)\quad (snd\ x)\ \#\ =\quad A\ B\ \wedge$
$((fst\ x)\ \#\quad (fst\ x))\quad ((snd\ x)\ \#\quad (snd\ x))\ =\ Id\ (sum\_eqType\ A\ B)\}.$
apply *constructive_indefinite_description*.
elim (@*pair_of_inclusions A B*) $\Rightarrow j$.
elim $\Rightarrow k\ H$.
$\exists\ (j,k)$.
simpl.
apply H.
Defined.
Definition *inl_r* $(A\ B : eqType) :=$ *fst* (*sval* (*sum_r A B*)).
Definition *inr_r* $(A\ B : eqType) :=$ *snd* (*sval* (*sum_r A B*)).

$,$ \hspace{8cm} (Axiom 23) $+\alpha$

$.$

Lemma *inl_id* $\{A\ B : eqType\}$: *inl_r A B*   *inl_r A B* $\#\ =\ Id\ A$.
Proof.
apply (*proj2_sig* (*sum_r A B*)).
Qed.
Lemma *inr_id* $\{A\ B : eqType\}$: *inr_r A B*   *inr_r A B* $\#\ =\ Id\ B$.
Proof.
apply (*proj2_sig* (*sum_r A B*)).
Qed.
Lemma *inl_inr_empty* $\{A\ B : eqType\}$: *inl_r A B*   *inr_r A B* $\#\ =\quad A\ B$.
Proof.
apply (*proj2_sig* (*sum_r A B*)).
Qed.
Lemma *inr_inl_empty* $\{A\ B : eqType\}$: *inr_r A B*   *inl_r A B* $\#\ =\quad B\ A$.
Proof.
apply *inv_invol2*.
rewrite *comp_inv inv_invol inv_empty*.
apply *inl_inr_empty*.
Qed.
Lemma *inl_inr_cup_id* $\{A\ B : eqType\}$:
$(inl\_r\ A\ B\ \#\quad inl\_r\ A\ B)\quad (inr\_r\ A\ B\ \#\quad inr\_r\ A\ B)\ =\ Id\ (sum\_eqType\ A\ B)$.
Proof.
apply (*proj2_sig* (*sum_r A B*)).
Qed.
Lemma *inl_function* $\{A\ B : eqType\}$: *function_r* (*inl_r A B*).
Proof.
move : (*proj2_sig* (*sum_r A B*)).

```
elim ⇒ H.
elim ⇒ H0.
elim ⇒ H1 H2.
split.
rewrite /total_r.
rewrite H.
apply inc_refl.
rewrite /univalent_r.
rewrite -H2.
apply cup_l.
Qed.
Lemma inr_function {A B : eqType}: function_r (inr_r A B).
Proof.
move : (proj2_sig (sum_r A B)).
elim ⇒ H.
elim ⇒ H0.
elim ⇒ H1 H2.
split.
rewrite /total_r.
rewrite H0.
apply inc_refl.
rewrite /univalent_r.
rewrite -H2.
apply cup_r.
Qed.
```

$$\alpha : A \rightharpoonup C \qquad \beta : B \rightharpoonup C \qquad\qquad \alpha \bot \beta : A + B \rightharpoonup C \quad , \alpha \bot \beta := j^\sharp \cdot \alpha \sqcup k^\sharp \cdot \beta$$

$$.$$

```
Definition Rel_sum {A B C : eqType} (alpha : Rel A C) (beta : Rel B C):=
 (inl_r A B #   alpha)   (inr_r A B #   beta).
```

12.1.2

Lemma 279 (sum_inc_compat) *Let* $\alpha, \alpha' : A \rightharpoonup C$ *and* $\beta, \beta' : B \rightharpoonup C$*. Then,*

$$\alpha \sqsubseteq \alpha' \wedge \beta \sqsubseteq \beta' \Rightarrow \alpha \bot \beta \sqsubseteq \alpha' \bot \beta'.$$

```
Lemma sum_inc_compat
 {A B C : eqType} {alpha alpha' : Rel A C} {beta beta' : Rel B C}:
 alpha    alpha' → beta    beta' → Rel_sum alpha beta    Rel_sum alpha' beta'.
Proof.
```

```
move ⇒ H H0.
apply cup_inc_compat.
apply (comp_inc_compat_ab_ab' H).
apply (comp_inc_compat_ab_ab' H0).
Qed.
```

Lemma 280 (sum_inc_compat_l) *Let* $\alpha : A \rightharpoonup C$ *and* $\beta, \beta' : B \rightharpoonup C$. *Then,*

$$\beta \sqsubseteq \beta' \Rightarrow \alpha \bot \beta \sqsubseteq \alpha \bot \beta'.$$

```
Lemma sum_inc_compat_l
 {A B C : eqType} {alpha : Rel A C} {beta beta' : Rel B C}:
 beta     beta' → Rel_sum alpha beta     Rel_sum alpha beta'.
Proof.
move ⇒ H.
apply (sum_inc_compat (@inc_refl _ _ alpha) H).
Qed.
```

Lemma 281 (sum_inc_compat_r) *Let* $\alpha, \alpha' : A \rightharpoonup C$ *and* $\beta : B \rightharpoonup C$. *Then,*

$$\alpha \sqsubseteq \alpha' \Rightarrow \alpha \bot \beta \sqsubseteq \alpha' \bot \beta.$$

```
Lemma sum_inc_compat_r
 {A B C : eqType} {alpha alpha' : Rel A C} {beta : Rel B C}:
 alpha     alpha' → Rel_sum alpha beta     Rel_sum alpha' beta.
Proof.
move ⇒ H.
apply (sum_inc_compat H (@inc_refl _ _ beta)).
Qed.
```

Lemma 282 (total_sum) *Let* $\alpha : A \rightharpoonup C$ *and* $\beta : B \rightharpoonup C$ *are total relations, then* $\alpha \bot \beta$ *is also a total relation.*

```
Lemma total_sum {A B C : eqType} {alpha : Rel A C} {beta : Rel B C}:
 total_r alpha → total_r beta → total_r (Rel_sum alpha beta).
Proof.
move ⇒ H H0.
rewrite /total_r/Rel_sum.
rewrite -inl_inr_cup_id inv_cup_distr comp_cup_distr_l comp_cup_distr_r comp_cup_distr_r.
rewrite comp_inv comp_inv inv_invol inv_invol.
apply cup_inc_compat.
apply (fun H' ⇒ @inc_trans _ _ _ _ _ H' (cup_l)).
rewrite comp_assoc -(@comp_assoc _ _ _ _ alpha).
```

189

```
apply comp_inc_compat_ab_ab'.
apply (comp_inc_compat_b_ab H).
apply (fun H' ⇒ @inc_trans _ _ _ _ _ H' (cup_r)).
rewrite comp_assoc -(@comp_assoc _ _ _ _ beta).
apply comp_inc_compat_ab_ab'.
apply (comp_inc_compat_b_ab H0).
Qed.
```

Lemma 283 (univalent_sum) *Let* $\alpha : A \rightharpoonup C$ *and* $\beta : B \rightharpoonup C$ *are univalent relations, then* $\alpha \bot \beta$ *is also a univalent relation.*

```
Lemma univalent_sum {A B C : eqType} {alpha : Rel A C} {beta : Rel B C}:
 univalent_r alpha → univalent_r beta → univalent_r (Rel_sum alpha beta).
Proof.
move ⇒ H H0.
rewrite /univalent_r/Rel_sum.
rewrite inv_cup_distr comp_cup_distr_l comp_cup_distr_r comp_cup_distr_r.
rewrite comp_inv comp_inv inv_invol inv_invol.
rewrite comp_assoc -(@comp_assoc _ _ _ _ (inl_r A B)) inl_id comp_id_l.
rewrite comp_assoc -(@comp_assoc _ _ _ _ (inr_r A B)) inr_inl_empty comp_empty_l
comp_empty_r cup_empty.
rewrite -cup_assoc comp_assoc -(@comp_assoc _ _ _ _ (inl_r A B)) inl_inr_empty comp_empty_l
comp_empty_r cup_empty.
rewrite comp_assoc -(@comp_assoc _ _ _ _ (inr_r A B)) inr_id comp_id_l.
apply inc_cup.
split.
apply H.
apply H0.
Qed.
```

Lemma 284 (function_sum) *Let* $\alpha : A \rightarrow C$ *and* $\beta : B \rightarrow C$ *are functions, then* $\alpha \bot \beta$ *is also a function.*

```
Lemma function_sum {A B C : eqType} {alpha : Rel A C} {beta : Rel B C}:
 function_r alpha → function_r beta → function_r (Rel_sum alpha beta).
Proof.
elim ⇒ H H0.
elim ⇒ H1 H2.
split.
apply (total_sum H H1).
apply (univalent_sum H0 H2).
Qed.
```

Lemma 285 (sum_conjugate) *Let* $\alpha : A \rightarrow C$, $\beta : B \rightarrow C$ *and* $\gamma : A + B \rightarrow C$ *be relations,* $j : A \rightarrow A + B$ *and* $k : B \rightarrow A + B$ *be inclusions. Then,*

$$j \cdot \gamma = \alpha \wedge k \cdot \gamma = \beta \Leftrightarrow \gamma = \alpha \bot \beta.$$

Lemma *sum_conjugate*
 $\{A\ B\ C : eqType\}\ \{alpha : Rel\ A\ C\}\ \{\text{beta} : Rel\ B\ C\}\ \{gamma : Rel\ (sum\_eqType\ A\ B)$
 $C\}$:
 $inl\_r\ A\ B \quad gamma = alpha \wedge inr\_r\ A\ B \quad gamma = \text{beta} \leftrightarrow$
 $gamma = Rel\_sum\ alpha\ \text{beta}$.
Proof.
split; move $\Rightarrow H$.
elim $H \Rightarrow$ H0 H1.
rewrite -$(@comp\_id\_l\ \_\ \_\ gamma)$.
rewrite -$inl\_inr\_cup\_id\ comp\_cup\_distr\_r\ comp\_assoc\ comp\_assoc$.
by [rewrite $H0\ H1$].
split.
rewrite $H\ comp\_cup\_distr\_l$ -$comp\_assoc$ -$comp\_assoc$.
rewrite $inl\_id\ inl\_inr\_empty\ comp\_id\_l\ comp\_empty\_l$.
by [rewrite cup_empty].
rewrite $H\ comp\_cup\_distr\_l$ -$comp\_assoc$ -$comp\_assoc$.
rewrite $inr\_id\ inr\_inl\_empty\ comp\_id\_l\ comp\_empty\_l$.
by [rewrite $cup\_comm\ cup\_empty$].
Qed.

Lemma 286 (sum_comp) *In below figure,*

$$(\alpha \bot \beta)^{\sharp} \cdot (\gamma \bot \delta) = \alpha^{\sharp} \cdot \gamma \sqcup \beta^{\sharp} \cdot \delta.$$
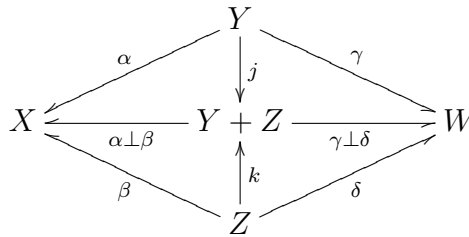
Lemma *sum_comp* $\{W\ X\ Y\ Z : eqType\}$
 $\{alpha : Rel\ Y\ X\}\ \{\text{beta} : Rel\ Z\ X\}\ \{gamma : Rel\ Y\ W\}\ \{\text{delta} : Rel\ Z\ W\}$:
 $(Rel\_sum\ alpha\ \text{beta})\ \# \quad Rel\_sum\ gamma\ \text{delta} =$
 $(alpha\ \# \quad gamma) \quad (\text{beta}\ \# \quad \text{delta})$.
Proof.
rewrite /Rel_sum.

```
rewrite inv_cup_distr comp_cup_distr_l comp_cup_distr_r comp_cup_distr_r.
rewrite comp_inv comp_inv inv_invol inv_invol.
apply f_equal2.
rewrite comp_assoc -(@comp_assoc _ _ _ _ (inl_r Y Z)) inl_id comp_id_l.
by [rewrite comp_assoc -(@comp_assoc _ _ _ _ (inr_r Y Z)) inr_inl_empty comp_empty_l
comp_empty_r cup_empty].
rewrite comp_assoc -(@comp_assoc _ _ _ _ (inl_r Y Z)) inl_inr_empty comp_empty_l
comp_empty_r cup_comm cup_empty.
by [rewrite comp_assoc -(@comp_assoc _ _ _ _ (inr_r Y Z)) inr_id comp_id_l].
Qed.
```

12.1.3

Lemma 287 (sum_cap_distr_l) *Let $\alpha : A \rightharpoonup C$ and $\beta, \beta' : B \rightharpoonup C$. Then,*

$$\alpha \bot (\beta \sqcap \beta') \sqsubseteq (\alpha \bot \beta) \sqcap (\alpha \bot \beta').$$

```
Lemma sum_cap_distr_l
 {A B C : eqType} {alpha : Rel A C} {beta beta' : Rel B C}:
 Rel_sum alpha (beta    beta')    (Rel_sum alpha beta    Rel_sum alpha beta').
Proof.
rewrite -cup_cap_distr_l.
apply cup_inc_compat_l.
apply comp_cap_distr_l.
Qed.
```

Lemma 288 (sum_cap_distr_r) *Let $\alpha, \alpha' : A \rightharpoonup C$ and $\beta : B \rightharpoonup C$. Then,*

$$(\alpha \sqcap \alpha') \bot \beta \sqsubseteq (\alpha \bot \beta) \sqcap (\alpha' \bot \beta).$$

```
Lemma sum_cap_distr_r
 {A B C : eqType} {alpha alpha' : Rel A C} {beta : Rel B C}:
 Rel_sum (alpha    alpha') beta    (Rel_sum alpha beta    Rel_sum alpha' beta).
Proof.
rewrite -cup_cap_distr_r.
apply cup_inc_compat_r.
apply comp_cap_distr_l.
Qed.
```

Lemma 289 (sum_cup_distr_l) *Let* $\alpha : A \rightharpoonup C$ *and* $\beta, \beta' : B \rightharpoonup C$. *Then,*

$$\alpha \bot (\beta \sqcup \beta') = (\alpha \bot \beta) \sqcup (\alpha \bot \beta').$$

Lemma *sum_cup_distr_l*
  $\{A\ B\ C : eqType\}$ $\{alpha : Rel\ A\ C\}$ {beta beta' : *Rel B C*}:
  Rel_sum alpha (beta   *beta'*) = *Rel_sum alpha* beta   *Rel_sum alpha beta'*.
Proof.
rewrite *-cup_assoc* (@*cup_comm* _ _ (*Rel_sum alpha* beta)) *-cup_assoc*.
by [rewrite *cup_idem cup_assoc -comp_cup_distr_l*].
Qed.

Lemma 290 (sum_cup_distr_r) *Let* $\alpha, \alpha' : A \rightharpoonup C$ *and* $\beta : B \rightharpoonup C$. *Then,*

$$(\alpha \sqcup \alpha') \bot \beta = (\alpha \bot \beta) \sqcup (\alpha' \bot \beta).$$

Lemma *sum_cup_distr_r*
  $\{A\ B\ C : eqType\}$ $\{alpha\ alpha' : Rel\ A\ C\}$ {beta : *Rel B C*}:
  Rel_sum (alpha   *alpha'*) beta = (*Rel_sum alpha* beta   *Rel_sum alpha'* beta).
Proof.
rewrite *cup_assoc* (@*cup_comm* _ _ (*inr_r A B #*   beta)) *cup_assoc*.
by [rewrite *cup_idem -cup_assoc -comp_cup_distr_l*].
Qed.

Lemma 291 (comp_sum_distr_r) *Let* $\alpha : A \rightharpoonup C$, $\beta : B \rightharpoonup C$ *and* $\gamma : C \rightharpoonup D$. *Then,*
$$(\alpha \bot \beta) \cdot \gamma = \alpha \cdot \gamma \bot \beta \cdot \gamma.$$

Lemma *comp_sum_distr_r*
  $\{A\ B\ C\ D : eqType\}$ $\{alpha : Rel\ A\ C\}$ {beta : *Rel B C*} $\{gamma : Rel\ C\ D\}$:
  (*Rel_sum alpha* beta)   *gamma = Rel_sum* (alpha   *gamma*) (beta   *gamma*).
Proof.
by [rewrite *comp_cup_distr_r comp_assoc comp_assoc*].
Qed.

12.2

12.2.1   ,

> (Axiom 24)   ,
> $p : A \times B \to A, k : A \times B \to B$.

Definition $prod\_r$ $(A \ B : eqType)$:
 $\{x : (Rel \ (prod\_eqType \ A \ B) \ A) \times (Rel \ (prod\_eqType \ A \ B) \ B) \ |$
 $(fst \ x) \ \# \quad (snd \ x) = \quad A \ B \ \wedge$
 $((fst \ x) \quad (fst \ x) \ \#) \quad ((snd \ x) \quad (snd \ x) \ \#) = Id \ (prod\_eqType \ A \ B) \ \wedge$
 $univalent\_r \ (fst \ x) \wedge univalent\_r \ (snd \ x)\}$.
apply $constructive\_indefinite\_description$.
elim $(@pair\_of\_projections \ A \ B) \Rightarrow p$.
elim $\Rightarrow q \ H$.
$\exists \ (p,q)$.
simpl.
apply H.
Defined.
Definition fst_r $(A \ B : eqType)$:= $fst \ (sval \ (prod\_r \ A \ B))$.
Definition snd_r $(A \ B : eqType)$:= $snd \ (sval \ (prod\_r \ A \ B))$.

>   ,   (Axiom 24) $+\alpha$
>   .

Lemma $fst\_snd\_universal$ $\{A \ B : eqType\}$: $fst\_r \ A \ B \ \# \quad snd\_r \ A \ B = \quad A \ B$.
Proof.
apply $(proj2\_sig \ (prod\_r \ A \ B))$.
Qed.
Lemma $snd\_fst\_universal$ $\{A \ B : eqType\}$: $snd\_r \ A \ B \ \# \quad fst\_r \ A \ B = \quad B \ A$.
Proof.
apply inv_invol2.
rewrite $comp\_inv \ inv\_invol \ inv\_universal$.
apply $fst\_snd\_universal$.
Qed.
Lemma $fst\_snd\_cap\_id$ $\{A \ B : eqType\}$:
 $(fst\_r \ A \ B \quad fst\_r \ A \ B \ \#) \quad (snd\_r \ A \ B \quad snd\_r \ A \ B \ \#) = Id \ (prod\_eqType \ A \ B)$.
Proof.
apply $(proj2\_sig \ (prod\_r \ A \ B))$.
Qed.
Lemma $fst\_function$ $\{A \ B : eqType\}$: $function\_r \ (fst\_r \ A \ B)$.
Proof.

```
move : (proj2_sig (prod_r A B)).
elim ⇒ H.
elim ⇒ H0 H1.
split.
rewrite /total_r.
rewrite -H0.
apply cap_l.
apply H1.
Qed.
Lemma snd_function {A B : eqType}: function_r (snd_r A B).
Proof.
move : (proj2_sig (prod_r A B)).
elim ⇒ H.
elim ⇒ H0 H1.
split.
rewrite /total_r.
rewrite -H0.
apply cap_r.
apply H1.
Qed.
```

$$\alpha : A \rightharpoonup B \qquad \beta : A \rightharpoonup C \qquad\qquad \alpha \top \beta : A \rightharpoonup B \times C \quad , \alpha \top \beta := \alpha \cdot p^\sharp \sqcap \beta \cdot q^\sharp$$

.

```
Definition Rel_prod {A B C : eqType} (alpha : Rel A B) (beta : Rel A C):=
 (alpha    fst_r B C #)    (beta    snd_r B C #).
```

12.2.2

Lemma 292 (prod_inc_compat) *Let* $\alpha, \alpha' : A \rightharpoonup B$ *and* $\beta, \beta' : A \rightharpoonup C$. *Then,*

$$\alpha \sqsubseteq \alpha' \land \beta \sqsubseteq \beta' \Rightarrow \alpha \top \beta \sqsubseteq \alpha' \top \beta'.$$

```
Lemma prod_inc_compat
 {A B C : eqType} {alpha alpha' : Rel A B} {beta beta' : Rel A C}:
 alpha    alpha' → beta    beta' → Rel_prod alpha beta    Rel_prod alpha' beta'.
Proof.
move ⇒ H H0.
apply cap_inc_compat.
apply (comp_inc_compat_ab_a'b H).
apply (comp_inc_compat_ab_a'b H0).
Qed.
```

Lemma 293 (prod_inc_compat_l) *Let $\alpha : A \rightharpoonup B$ and $\beta, \beta' : A \rightharpoonup C$. Then,*

$$\beta \sqsubseteq \beta' \Rightarrow \alpha \top \beta \sqsubseteq \alpha \top \beta'.$$

Lemma *prod_inc_compat_l*
 {*A B C : eqType*} {*alpha : Rel A B*} {beta *beta' : Rel A C*}:
 beta   *beta' → Rel_prod alpha* beta   *Rel_prod alpha beta'.*
Proof.
move $\Rightarrow$ *H.*
apply (*prod_inc_compat* (@*inc_refl* _ _ *alpha*) *H*).
Qed.

Lemma 294 (prod_inc_compat_r) *Let $\alpha, \alpha' : A \rightharpoonup B$ and $\beta : A \rightharpoonup C$. Then,*

$$\alpha \sqsubseteq \alpha' \Rightarrow \alpha \top \beta \sqsubseteq \alpha' \top \beta.$$

Lemma *prod_inc_compat_r*
 {*A B C : eqType*} {*alpha alpha' : Rel A B*} {beta *: Rel A C*}:
 alpha   *alpha' → Rel_prod alpha* beta   *Rel_prod alpha'* beta.
Proof.
move $\Rightarrow$ *H.*
apply (*prod_inc_compat H* (@*inc_refl* _ _ beta)).
Qed.

Lemma 295 (total_prod) *Let $\alpha : A \rightharpoonup B$ and $\beta : A \rightharpoonup C$ are total relations, then $\alpha \top \beta$ is also a total relation.*

Lemma *total_prod* {*A B C : eqType*} {*alpha : Rel A B*} {beta *: Rel A C*}:
 total_r alpha → total_r beta *→ total_r* (*Rel_prod alpha* beta).
Proof.
move $\Rightarrow$ *H H0.*
rewrite *domain_total cap_domain cap_comm.*
apply *Logic.eq_sym.*
apply *inc_def1.*
apply (@*inc_trans* _ _ _ _ _ *H*).
rewrite *comp_inv inv_invol comp_assoc.*
apply *comp_inc_compat_ab_ab'.*
apply (@*inc_trans* _ _ _ (*alpha #*   (beta   beta *#*))).
apply (*comp_inc_compat_a_ab H0*).
rewrite -*comp_assoc* -*comp_assoc fst_snd_universal.*
apply *comp_inc_compat_ab_a'b.*
apply *inc_alpha_universal.*
Qed.

Lemma 296 (univalent_prod) *Let $\alpha : A \to B$ and $\beta : A \to C$ are univalent relations, then $\alpha \top \beta$ is also a univalent relation.*

Lemma *univalent_prod* {*A B C : eqType*} {*alpha : Rel A B*} {beta : *Rel A C*}:
 univalent_r alpha → *univalent_r* beta → *univalent_r* (*Rel_prod alpha* beta).
Proof.
move ⇒ *H H0*.
rewrite /*univalent_r*/*Rel_prod*.
rewrite *inv_cap_distr comp_inv inv_invol comp_inv inv_invol*.
apply (@*inc_trans* _ _ _ _ _ (*comp_cap_distr_l*)).
rewrite -*fst_snd_cap_id*.
apply *cap_inc_compat*.
apply (@*inc_trans* _ _ _ _ _ (*comp_cap_distr_r*)).
apply (@*inc_trans* _ _ _ _ _ (*cap_l*)).
rewrite *comp_assoc* -(@*comp_assoc* _ _ _ _ _ *alpha*).
apply *comp_inc_compat_ab_ab'*.
apply (*comp_inc_compat_ab_b H*).
apply (@*inc_trans* _ _ _ _ _ (*comp_cap_distr_r*)).
apply (@*inc_trans* _ _ _ _ _ (*cap_r*)).
rewrite *comp_assoc* -(@*comp_assoc* _ _ _ _ _ beta).
apply *comp_inc_compat_ab_ab'*.
apply (*comp_inc_compat_ab_b H0*).
Qed.

Lemma 297 (function_prod) *Let $\alpha : A \to B$ and $\beta : A \to C$ are functions, then $\alpha \top \beta$ is also a function.*

Lemma *function_prod* {*A B C : eqType*} {*alpha : Rel A B*} {beta : *Rel A C*}:
 function_r alpha → *function_r* beta → *function_r* (*Rel_prod alpha* beta).
Proof.
elim ⇒ *H H0*.
elim ⇒ *H1 H2*.
split.
apply (*total_prod H H1*).
apply (*univalent_prod H0 H2*).
Qed.

Lemma 298 (prod_fst_surjection) *Let $p : B \times C \to B$ be a projection. Then,*

$$\text{``}p \text{ is a surjection''} \Leftrightarrow \forall D, \nabla_{BD} = \nabla_{BC} \cdot \nabla_{CD}.$$

Lemma *prod_fst_surjection* {*B C : eqType*}:

$surjection\_r\ (fst\_r\ B\ C) \leftrightarrow \forall\ D : eqType,\quad B\ D = \quad B\ C\qquad C\ D.$

```
Proof.
split; move ⇒ H.
move ⇒ D.
elim H ⇒ H0 H1.
apply inc_antisym.
apply (@inc_trans _ _ _ ((fst_r B C #    (fst_r B C #) #)        B D)).
apply (comp_inc_compat_b_ab H1).
rewrite inv_invol.
apply (@inc_trans _ _ _ (((fst_r B C #    snd_r B C)    (snd_r B C #    fst_r B C))
   B D)).
apply comp_inc_compat_ab_a'b.
rewrite comp_assoc -(@comp_assoc _ _ _ _ (snd_r B C)).
apply comp_inc_compat_ab_ab'.
apply comp_inc_compat_b_ab.
apply snd_function.
rewrite (@comp_assoc _ _ _ _ _ _ (    B D)).
apply comp_inc_compat.
apply inc_alpha_universal.
apply inc_alpha_universal.
apply inc_alpha_universal.
split.
apply fst_function.
rewrite /total_r.
rewrite -(@cap_universal _ _ (Id B)) (H B) -(@fst_snd_universal B C) cap_comm comp_assoc.
apply (@inc_trans _ _ _ _ _ (dedekind1)).
apply comp_inc_compat_ab_ab'.
rewrite comp_id_r.
apply cap_r.
Qed.
```

Lemma 299 (prod_snd_surjection) *Let $q : B \times C \to C$ be a projection. Then,*

$$\text{``q is a surjection''} \Leftrightarrow \forall D, \nabla_{CD} = \nabla_{CB} \cdot \nabla_{BD}.$$

```
Lemma prod_snd_surjection {B C : eqType}:
```
$surjection\_r\ (snd\_r\ B\ C) \leftrightarrow \forall\ D : eqType,\quad C\ D = \quad C\ B\qquad B\ D.$
```
Proof.
split; move ⇒ H.
move ⇒ D.
elim H ⇒ H0 H1.
apply inc_antisym.
apply (@inc_trans _ _ _ ((snd_r B C #    (snd_r B C #) #)        C D)).
```

apply ($comp\_inc\_compat\_b\_ab$ $H1$).
rewrite inv_invol.
apply (@inc_trans _ _ _ (((snd_r B C # fst_r B C) (fst_r B C # snd_r B C))
C D)).
apply $comp\_inc\_compat\_ab\_a'b$.
rewrite $comp\_assoc$ -(@$comp\_assoc$ _ _ _ _ (fst_r B C)).
apply $comp\_inc\_compat\_ab\_ab'$.
apply $comp\_inc\_compat\_b\_ab$.
apply $fst\_function$.
rewrite (@$comp\_assoc$ _ _ _ _ _ _ (C D)).
apply $comp\_inc\_compat$.
apply $inc\_alpha\_universal$.
apply $inc\_alpha\_universal$.
apply $inc\_alpha\_universal$.
split.
apply $snd\_function$.
rewrite /$total\_r$.
rewrite -(@$cap\_universal$ _ _ (Id C)) (H C) -(@$snd\_fst\_universal$ B C) cap_comm $comp\_assoc$.
apply (@inc_trans _ _ _ _ _ ($dedekind1$)).
apply $comp\_inc\_compat\_ab\_ab'$.
rewrite $comp\_id\_r$.
apply cap_r.
Qed.

Lemma 300 (prod_fst_domain1) *Let $p : B \times C \to B$ be a projection, $\alpha : A \to B$
and $\beta : A \to C$. Then,*
$$(\alpha \top \beta) \cdot p = \lfloor \beta \rfloor \cdot \alpha.$$

Lemma $prod\_fst\_domain1$ {A B C : $eqType$} {$alpha$: Rel A B} {beta : Rel A C}:
(Rel_prod $alpha$ beta) fst_r B C = $domain$ beta $alpha$.
Proof.
rewrite (@$comp\_inv\_inv$ A A) $domain\_inv$.
rewrite $domain\_universal2$ inv_cap_distr $comp\_inv$ inv_invol inv_invol $inv\_universal$.
rewrite -$snd\_fst\_universal$.
apply $inc\_antisym$.
apply (@inc_trans _ _ _ _ _ ($comp\_cap\_distr\_r$)).
rewrite $comp\_assoc$ $comp\_assoc$.
apply $cap\_inc\_compat\_r$.
apply $comp\_inc\_compat\_ab\_a$.
apply $fst\_function$.
rewrite cap_comm -$comp\_assoc$.
apply (@inc_trans _ _ _ _ _ ($dedekind2$)).

199

rewrite *cap_comm*.
apply *inc_refl*.
Qed.

Lemma 301 (prod_fst_domain2) *Let* $p : B \times C \to B$ *be a projection,* $\alpha : A \to B$
and $\beta : A \to C$. *Then,*
$$(\alpha \top \beta) \cdot p = \alpha \Leftrightarrow \lfloor \alpha \rfloor \sqsubseteq \lfloor \beta \rfloor.$$

Lemma *prod_fst_domain2* {*A B C* : *eqType*} {*alpha* : *Rel A B*} {beta : *Rel A C*}:
 (*Rel_prod alpha* beta)   *fst_r B C = alpha* ↔ *domain alpha*   *domain* beta.
Proof.
rewrite *prod_fst_domain1*.
split; move ⇒ *H*.
apply *domain_lemma2b*.
assert ((*domain* beta   *alpha*)   ((beta   beta #)   *alpha*)).
apply *comp_inc_compat_ab_a'b*.
apply *cap_l*.
rewrite *H* in *H0*.
apply *H0*.
apply *inc_antisym*.
apply *comp_inc_compat_ab_b*.
apply *cap_r*.
apply (@*inc_trans* _ _ _ (*domain alpha*   *alpha*)).
rewrite *domain_comp_alpha1*.
apply *inc_refl*.
apply (*comp_inc_compat_ab_a'b H*).
Qed.

Lemma 302 (prod_snd_domain1) *Let* $q : B \times C \to C$ *be a projection,* $\alpha : A \to B$
and $\beta : A \to C$. *Then,*
$$(\alpha \top \beta) \cdot q = \lfloor \alpha \rfloor \cdot \beta.$$

Lemma *prod_snd_domain1* {*A B C* : *eqType*} {*alpha* : *Rel A B*} {beta : *Rel A C*}:
 (*Rel_prod alpha* beta)   *snd_r B C = domain alpha*   beta.
Proof.
rewrite (@*comp_inv_inv A A*) *domain_inv*.
rewrite *domain_universal2 inv_cap_distr comp_inv inv_invol inv_invol inv_universal*.
rewrite -*fst_snd_universal*.
apply *inc_antisym*.
apply (@*inc_trans* _ _ _ _ _ (*comp_cap_distr_r*)).
rewrite *comp_assoc comp_assoc cap_comm*.
apply *cap_inc_compat_r*.
apply *comp_inc_compat_ab_a*.

apply *snd_function*.
rewrite *cap_comm* *-comp_assoc*.
apply *dedekind2*.
Qed.

Lemma 303 (prod_snd_domain2) *Let* $q : B \times C \to C$ *be a projection,* $\alpha : A \rightarrowtail B$ *and* $\beta : A \rightarrowtail C$. *Then,*
$$(\alpha \top \beta) \cdot q = \beta \Leftrightarrow \lfloor \beta \rfloor \sqsubseteq \lfloor \alpha \rfloor.$$

Lemma *prod_snd_domain2* {*A B C : eqType*} {*alpha : Rel A B*} {beta : *Rel A C*}:
 (*Rel_prod alpha* beta)   *snd_r B C* = beta $\leftrightarrow$ *domain* beta   *domain alpha*.
Proof.
rewrite *prod_snd_domain1*.
split; move $\Rightarrow$ *H*.
apply *domain_lemma2b*.
assert ((*domain alpha*   beta)   ((*alpha*   *alpha* #)   beta)).
apply *comp_inc_compat_ab_a'b*.
apply *cap_l*.
rewrite *H* in *H0*.
apply *H0*.
apply *inc_antisym*.
apply *comp_inc_compat_ab_b*.
apply *cap_r*.
apply (@*inc_trans* _ _ _ (*domain beta*   beta)).
rewrite *domain_comp_alpha1*.
apply *inc_refl*.
apply (*comp_inc_compat_ab_a'b H*).
Qed.

Lemma 304 (prod_to_cap) *Let* $\alpha : A \rightarrowtail B$ *and* $\beta : A \rightarrowtail C$. *Then,*
$$\lfloor \alpha \top \beta \rfloor = \lfloor \alpha \rfloor \sqcap \lfloor \beta \rfloor.$$

Lemma *prod_to_cap* {*A B C : eqType*} {*alpha : Rel A B*} {beta : *Rel A C*}:
 domain (*Rel_prod alpha* beta) = *domain alpha*   *domain* beta.
Proof.
replace (*domain* (*Rel_prod alpha* beta)) with (*domain* (*Rel_prod alpha* beta   *snd_r B C*)).
rewrite *prod_snd_domain1 comp_domain8*.
apply *dedekind_id3*.
apply *cap_r*.
apply *cap_r*.
apply *cap_r*.

```
apply comp_domain3.
apply snd_function.
Qed.
```

Lemma 305 (prod_conjugate1) *Let* $\alpha : A \to B$ *and* $\beta : A \to C$ *be functions,* $p : B \times C \to B$ *and* $q : B \times C \to C$ *be projections. Then,*

$$(\alpha \top \beta) \cdot p = \alpha \wedge (\alpha \top \beta) \cdot q = \beta.$$

```
Lemma prod_conjugate1 {A B C : eqType} {alpha : Rel A B} {beta : Rel A C}:
 function_r alpha → function_r beta →
 Rel_prod alpha beta     fst_r B C = alpha ∧ Rel_prod alpha beta     snd_r B C = beta.
Proof.
move ⇒ H H0.
split.
rewrite prod_fst_domain1.
elim H0 ⇒ H1 H2.
apply inc_def1 in H1.
rewrite /domain.
by [rewrite cap_comm -H1 comp_id_l].
rewrite prod_snd_domain1.
elim H ⇒ H1 H2.
apply inc_def1 in H1.
rewrite /domain.
by [rewrite cap_comm -H1 comp_id_l].
Qed.
```

Lemma 306 (prod_conjugate2) *Let* $\gamma : A \to B \times C$ *be a function,* $p : B \times C \to B$ *and* $q : B \times C \to C$ *be projections. Then,*

$$(\gamma \cdot p) \top (\gamma \cdot q) = \gamma.$$

```
Lemma prod_conjugate2 {A B C : eqType} {gamma : Rel A (prod_eqType B C)}:
 function_r gamma → Rel_prod (gamma     fst_r B C) (gamma     snd_r B C) = gamma.
Proof.
move ⇒ H.
rewrite /Rel_prod.
rewrite comp_assoc comp_assoc -(function_cap_distr_l H).
by [rewrite fst_snd_cap_id comp_id_r].
Qed.
```

Lemma 307 (diagonal_conjugate) *Let* $p : B \times C \to B$ *and* $q : B \times C \to C$ *be projections. Then,*

$$\frac{\alpha : A \to B \quad \alpha = p^\sharp \cdot u \cdot q}{u \sqsubseteq id_{A \times B} \quad u = \lfloor p \cdot \alpha \sqcap q \rfloor}.$$

Lemma *diagonal_conjugate* $\{A\ B : eqType\}$ $\{alpha : Rel\ A\ B\}$:
  conjugate A B (*prod_eqType A B*) (*prod_eqType A B*)
  True_r (**fun** $u \Rightarrow u$ *Id* (*prod_eqType A B*))
  (**fun** $u \Rightarrow$ (*fst_r A B* #   *u*)   *snd_r A B*)
  (**fun** *alpha* $\Rightarrow$ *domain* ((*fst_r A B*   *alpha*)   *snd_r A B*)).
Proof.
split.
move $\Rightarrow$ *alpha0 H*.
split.
apply *cap_r*.
rewrite *cap_domain*.
apply *inc_antisym*.
apply (@*inc_trans* _ _ _ ((*fst_r A B* #   ((*fst_r A B*   *alpha0*)   *snd_r A B* #))   *snd_r A B*)).
apply *comp_inc_compat_ab_a'b*.
apply *comp_inc_compat_ab_ab'*.
apply *cap_l*.
rewrite *comp_assoc comp_assoc* -*comp_assoc* -(@*comp_assoc* _ _ _ _ (*fst_r A B* #)).
apply (@*inc_trans* _ _ _ ((*fst_r A B* #   *fst_r A B*)   *alpha0*)).
apply *comp_inc_compat_ab_a*.
apply *snd_function*.
apply *comp_inc_compat_ab_b*.
apply *fst_function*.
apply (@*inc_trans* _ _ _ (*alpha0*   ((*fst_r A B* #   *Id* (*prod_eqType A B*))   *snd_r A B*))).
rewrite *comp_id_r fst_snd_universal cap_universal*.
apply *inc_refl*.
rewrite *cap_comm*.
apply (@*inc_trans* _ _ _ _ _ (*dedekind2*)).
apply *comp_inc_compat_ab_a'b*.
apply (@*inc_trans* _ _ _ _ _ (*dedekind1*)).
apply *comp_inc_compat_ab_ab'*.
rewrite *cap_comm inv_invol comp_assoc*.
apply *inc_refl*.
move $\Rightarrow$ *u H*.
split.
by [].

```
replace ((fst_r A B    ((fst_r A B #    u)    snd_r A B))    snd_r A B) with (u    snd_r A B).
apply domain_inc_id in H.
move : (@snd_function A B) ⇒ H0.
elim H0 ⇒ H1 H2.
by [rewrite (comp_domain3 H1) H].
rewrite comp_assoc -comp_assoc.
apply inc_antisym.
apply (@inc_trans _ _ _ ((u    snd_r A B)    snd_r A B)).
apply inc_cap.
split.
apply inc_refl.
apply (comp_inc_compat_ab_b H).
apply cap_inc_compat_r.
apply comp_inc_compat_b_ab.
apply fst_function.
apply (@inc_trans _ _ _ _ _ (dedekind2)).
apply comp_inc_compat_ab_b.
rewrite -fst_snd_cap_id.
apply cap_inc_compat_l.
apply comp_inc_compat_ab_ab'.
apply inc_inv.
apply (comp_inc_compat_ab_b H).
Qed.
```

12.2.3

1   ,   .

Lemma 308 (sharpness) *In below figure,*

$$\alpha \cdot \gamma^\sharp \sqcap \beta \cdot \delta^\sharp = (\alpha \cdot p^\sharp \sqcap \beta \cdot q^\sharp) \cdot (p \cdot \gamma^\sharp \sqcap q \cdot \delta^\sharp).$$
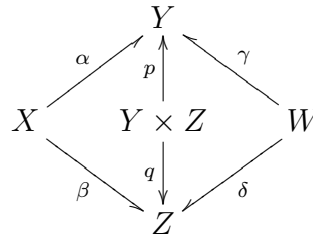

```
Lemma sharpness {W X Y Z : eqType}
 {alpha : Rel X Y} {beta : Rel X Z} {gamma : Rel W Y} {delta : Rel W Z}:
```

$(alpha \quad gamma \ \#) \quad (beta \quad delta \ \#) =$
$((alpha \quad fst\_r \ Y \ Z \ \#) \quad (beta \quad snd\_r \ Y \ Z \ \#))$
$\quad ((fst\_r \ Y \ Z \quad gamma \ \#) \quad (snd\_r \ Y \ Z \quad delta \ \#)).$

Proof.
apply $inc\_antisym$.
move : $(rationality \ \_ \ \_ \ alpha) \Rightarrow H.$
move : $(rationality \ \_ \ \_ \ beta) \Rightarrow H0.$
move : $(rationality \ \_ \ \_ \ (gamma \ \#)) \Rightarrow H1.$
move : $(rationality \ \_ \ \_ \ (delta \ \#)) \Rightarrow H2.$
elim $H \Rightarrow R.$
elim $\Rightarrow f0.$
elim $\Rightarrow g0 \ H3.$
elim $H0 \Rightarrow R0.$
elim $\Rightarrow f1.$
elim $\Rightarrow g1 \ H4.$
elim $H1 \Rightarrow R1.$
elim $\Rightarrow h0.$
elim $\Rightarrow k0 \ H5.$
elim $H2 \Rightarrow R2.$
elim $\Rightarrow h1.$
elim $\Rightarrow k1 \ H6.$
move : $(rationality \ \_ \ \_ \ (g0 \quad h0 \ \#)) \Rightarrow H7.$
move : $(rationality \ \_ \ \_ \ (g1 \quad h1 \ \#)) \Rightarrow H8.$
move : $(rationality \ \_ \ \_ \ ((alpha \quad gamma \ \#) \quad (beta \quad delta \ \#))) \Rightarrow H9.$
elim $H7 \Rightarrow R3.$
elim $\Rightarrow s0.$
elim $\Rightarrow t0 \ H10.$
elim $H8 \Rightarrow R4.$
elim $\Rightarrow s1.$
elim $\Rightarrow t1 \ H11.$
elim $H9 \Rightarrow R5.$
elim $\Rightarrow x.$
elim $\Rightarrow z \ H12.$
assert $(alpha \quad gamma \ \# = (f0 \ \# \quad (s0 \ \# \quad t0)) \quad k0).$
replace $alpha$ with $(f0 \ \# \quad g0).$
replace $(gamma \ \#)$ with $(h0 \ \# \quad k0).$
rewrite $-comp\_assoc$ $(@comp\_assoc \ \_ \ \_ \ \_ \ \_ \ (f0 \ \#)).$
apply $f\_equal2.$
apply f_equal.
apply $H10.$
by $\|.$
apply $Logic.eq\_sym.$

```
apply H5.
apply Logic.eq_sym.
apply H3.
assert (beta    delta # = (f1 #    (s1 #    t1))    k1).
replace beta with (f1 #    g1).
replace (delta #) with (h1 #    k1).
rewrite -comp_assoc (@comp_assoc _ _ _ _ (f1 #)).
apply f_equal2.
apply f_equal.
apply H11.
by [].
apply Logic.eq_sym.
apply H6.
apply Logic.eq_sym.
apply H4.
assert (t0    h0 = s0    g0).
apply function_inc.
apply function_comp.
apply H10.
apply H5.
apply function_comp.
apply H10.
apply H3.
apply (@inc_trans _ _ _ (s0    ((s0 #    t0)    h0))).
rewrite comp_assoc -comp_assoc.
apply comp_inc_compat_b_ab.
apply H10.
apply comp_inc_compat_ab_ab'.
replace (s0 #    t0) with (g0    h0 #).
rewrite comp_assoc.
apply comp_inc_compat_ab_a.
apply H5.
apply H10.
assert (t1    h1 = s1    g1).
apply function_inc.
apply function_comp.
apply H11.
apply H6.
apply function_comp.
apply H11.
apply H4.
apply (@inc_trans _ _ _ (s1    ((s1 #    t1)    h1))).
```

```
rewrite comp_assoc -comp_assoc.
apply comp_inc_compat_b_ab.
apply H11.
apply comp_inc_compat_ab_ab'.
replace (s1 #    t1) with (g1    h1 #).
rewrite comp_assoc.
apply comp_inc_compat_ab_a.
apply H6.
apply H11.
remember ((x    (s0    f0) #)    (z    (t0    k0) #)) as m0.
remember ((x    (s1    f1) #)    (z    (t1    k1) #)) as m1.
assert (total_r m0).
rewrite Heqm0.
apply domain_corollary1.
apply H12.
apply H12.
replace (x #    z) with ((alpha    gamma #)    (beta    delta #)).
apply (@inc_trans _ _ _ _ _ (cap_l)).
rewrite comp_inv H13 -comp_assoc comp_assoc.
apply inc_refl.
apply H12.
assert (total_r m1).
rewrite Heqm1.
apply domain_corollary1.
apply H12.
apply H12.
replace (x #    z) with ((alpha    gamma #)    (beta    delta #)).
apply (@inc_trans _ _ _ _ _ (cap_r)).
rewrite comp_inv H14 -comp_assoc comp_assoc.
apply inc_refl.
apply H12.
remember (m0    (s0    g0)) as n0.
remember (m1    (s1    g1)) as n1.
assert (total_r n0).
rewrite Heqn0.
apply (total_comp H17).
apply total_comp.
apply H10.
apply H3.
assert (total_r n1).
rewrite Heqn1.
apply (total_comp H18).
```

```
apply total_comp.
apply H11.
apply H4.
assert (total_r ((n0     fst_r Y Z #)     (n1     snd_r Y Z #))).
apply (domain_corollary1 H19 H20).
rewrite fst_snd_universal.
apply inc_alpha_universal.
assert ((x #     n0)     alpha).
replace alpha with (f0 #     g0).
rewrite Heqn0 Heqm0.
apply (@inc_trans _ _ _ (((x #     x)     f0 #)     ((s0 #     s0)     g0))).
rewrite comp_assoc comp_assoc.
apply comp_inc_compat_ab_ab'.
rewrite -comp_assoc -comp_assoc -comp_assoc -comp_assoc.
apply comp_inc_compat_ab_a'b.
apply comp_inc_compat_ab_a'b.
rewrite comp_assoc -comp_inv.
apply cap_l.
apply comp_inc_compat.
apply comp_inc_compat_ab_b.
apply H12.
apply comp_inc_compat_ab_b.
apply H10.
apply Logic.eq_sym.
apply H3.
assert ((x #     n1)     beta).
replace beta with (f1 #     g1).
rewrite Heqn1 Heqm1.
apply (@inc_trans _ _ _ (((x #     x)     f1 #)     ((s1 #     s1)     g1))).
rewrite comp_assoc comp_assoc.
apply comp_inc_compat_ab_ab'.
rewrite -comp_assoc -comp_assoc -comp_assoc -comp_assoc.
apply comp_inc_compat_ab_a'b.
apply comp_inc_compat_ab_a'b.
rewrite comp_assoc -comp_inv.
apply cap_l.
apply comp_inc_compat.
apply comp_inc_compat_ab_b.
apply H12.
apply comp_inc_compat_ab_b.
apply H11.
apply Logic.eq_sym.
```

```
apply H4.
assert ((n0 #    z)    gamma #).
replace (gamma #) with (h0 #     k0).
rewrite Heqn0 Heqm0 -H15 comp_inv comp_inv inv_cap_distr.
apply (@inc_trans _ _ _ ((h0 #    (t0 #     t0))    (k0    (z #    z)))).
rewrite -comp_assoc -comp_assoc -comp_assoc.
apply comp_inc_compat_ab_a'b.
rewrite comp_assoc comp_assoc comp_assoc comp_assoc.
apply comp_inc_compat_ab_ab'.
apply comp_inc_compat_ab_ab'.
rewrite -comp_assoc (@comp_inv _ _ _ z) inv_invol.
apply cap_r.
apply comp_inc_compat.
apply comp_inc_compat_ab_a.
apply H10.
apply comp_inc_compat_ab_a.
apply H12.
apply Logic.eq_sym.
apply H5.
assert ((n1 #    z)    delta #).
replace (delta #) with (h1 #     k1).
rewrite Heqn1 Heqm1 -H16 comp_inv comp_inv inv_cap_distr.
apply (@inc_trans _ _ _ ((h1 #    (t1 #    t1))    (k1    (z #    z)))).
rewrite -comp_assoc -comp_assoc -comp_assoc.
apply comp_inc_compat_ab_a'b.
rewrite comp_assoc comp_assoc comp_assoc comp_assoc.
apply comp_inc_compat_ab_ab'.
apply comp_inc_compat_ab_ab'.
rewrite -comp_assoc (@comp_inv _ _ _ z) inv_invol.
apply cap_r.
apply comp_inc_compat.
apply comp_inc_compat_ab_a.
apply H11.
apply comp_inc_compat_ab_a.
apply H12.
apply Logic.eq_sym.
apply H6.
replace ((alpha    gamma #)    (beta    delta #)) with (x #     z).
apply (@inc_trans _ _ _ ((x #    (((n0    fst_r Y Z #)    (n1    snd_r Y Z #))    (((n0
    fst_r Y Z #)    (n1    snd_r Y Z #))) #))    z)).
apply comp_inc_compat_ab_a'b.
apply (comp_inc_compat_a_ab H21).
```

rewrite -*comp_assoc comp_assoc*.
apply *comp_inc_compat*.
apply (@*inc_trans* _ _ _ _ _ (*comp_cap_distr_l*)).
apply *cap_inc_compat*.
rewrite -*comp_assoc*.
apply (*comp_inc_compat_ab_a'b H22*).
rewrite -*comp_assoc*.
apply (*comp_inc_compat_ab_a'b H23*).
rewrite *inv_cap_distr comp_inv comp_inv inv_invol inv_invol*.
apply (@*inc_trans* _ _ _ _ _ (*comp_cap_distr_r*)).
apply *cap_inc_compat*.
rewrite *comp_assoc*.
apply (*comp_inc_compat_ab_ab' H24*).
rewrite *comp_assoc*.
apply (*comp_inc_compat_ab_ab' H25*).
apply *Logic.eq_sym*.
apply *H12*.
apply (@*inc_trans* _ _ _ _ _ (*comp_cap_distr_l*)).
apply *cap_inc_compat*.
apply (@*inc_trans* _ _ _ _ _ (*comp_cap_distr_r*)).
apply (@*inc_trans* _ _ _ _ _ (*cap_l*)).
rewrite -*comp_assoc* (@*comp_assoc* _ _ _ _ *alpha*).
apply *comp_inc_compat_ab_a'b*.
apply *comp_inc_compat_ab_a*.
apply *fst_function*.
apply (@*inc_trans* _ _ _ _ _ (*comp_cap_distr_r*)).
apply (@*inc_trans* _ _ _ _ _ (*cap_r*)).
rewrite -*comp_assoc* (@*comp_assoc* _ _ _ _ beta).
apply *comp_inc_compat_ab_a'b*.
apply *comp_inc_compat_ab_a*.
apply *snd_function*.
Qed.

12.2.4

Lemma 309 (prod_cap_distr_l) *Let $\alpha : A \to B$ and $\beta, \beta' : A \to C$. Then,*

$$\alpha \top (\beta \sqcap \beta') = (\alpha \top \beta) \sqcap (\alpha \top \beta').$$

Lemma *prod_cap_distr_l* {*A B C* : *eqType*} {*alpha* : *Rel A B*} {beta beta' : *Rel A C*}:
 Rel_prod alpha (beta   beta') = *Rel_prod alpha* beta   *Rel_prod alpha beta'*.
Proof.

```
rewrite /Rel_prod.
rewrite -cap_assoc (@cap_comm _ _ _ (alpha    fst_r B C #)) -cap_assoc cap_idem
cap_assoc.
apply f_equal.
apply function_cap_distr_r.
apply snd_function.
Qed.
```

Lemma 310 (prod_cap_distr_r) *Let $\alpha, \alpha' : A \rightharpoonup B$ and $\beta : A \rightharpoonup C$. Then,*

$$(\alpha \sqcap \alpha')\top\beta = (\alpha\top\beta) \sqcap (\alpha'\top\beta).$$

```
Lemma prod_cap_distr_r {A B C : eqType} {alpha alpha' : Rel A B} {beta : Rel A C}:
 Rel_prod (alpha    alpha') beta = Rel_prod alpha beta    Rel_prod alpha' beta.
Proof.
rewrite /Rel_prod.
rewrite cap_assoc (@cap_comm _ _ (beta   snd_r B C #)) cap_assoc cap_idem -cap_assoc.
apply (@f_equal _ _ (fun x ⇒ @cap _ _ x (beta    snd_r B C #))).
apply function_cap_distr_r.
apply fst_function.
Qed.
```

Lemma 311 (prod_cup_distr_l) *Let $\alpha : A \rightharpoonup B$ and $\beta, \beta' : A \rightharpoonup C$. Then,*

$$\alpha\top(\beta \sqcup \beta') = (\alpha\top\beta) \sqcup (\alpha\top\beta').$$

```
Lemma prod_cup_distr_l {A B C : eqType} {alpha : Rel A B} {beta beta' : Rel A C}:
 Rel_prod alpha (beta    beta') = Rel_prod alpha beta    Rel_prod alpha beta'.
Proof.
by [rewrite -cap_cup_distr_l -comp_cup_distr_r].
Qed.
```

Lemma 312 (prod_cup_distr_r) *Let $\alpha, \alpha' : A \rightharpoonup B$ and $\beta : A \rightharpoonup C$. Then,*

$$(\alpha \sqcup \alpha')\top\beta = (\alpha\top\beta) \sqcup (\alpha'\top\beta).$$

```
Lemma prod_cup_distr_r {A B C : eqType} {alpha alpha' : Rel A B} {beta : Rel A C}:
 Rel_prod (alpha    alpha') beta = Rel_prod alpha beta    Rel_prod alpha' beta.
Proof.
by [rewrite -cap_cup_distr_r -comp_cup_distr_r].
Qed.
```

> **Lemma 313 (comp_prod_distr_l)** *Let $\alpha : A \rightharpoonup B$, $\beta : B \rightharpoonup C$ and $\gamma : B \rightharpoonup D$. Then,*
> $$\alpha \cdot (\beta \top \gamma) \sqsubseteq \alpha \cdot \beta \top \alpha \cdot \gamma.$$

Lemma *comp_prod_distr_l*
 $\{A\ B\ C\ D : eqType\}$ $\{alpha : Rel\ A\ B\}$ $\{$beta $: Rel\ B\ C\}$ $\{gamma : Rel\ B\ D\}$:
 alpha $\quad$ Rel_prod beta *gamma $\quad$ Rel_prod (alpha $\quad$* beta$)$ $(alpha \quad gamma)$.
Proof.
rewrite $/Rel\_prod$.
rewrite *comp_assoc comp_assoc*.
apply *comp_cap_distr_l*.
Qed.

> **Lemma 314 (function_prod_distr_l)** *Let $\alpha : A \rightharpoonup B$ be a function, $\beta : B \rightharpoonup C$ and $\gamma : B \rightharpoonup D$. Then,*
> $$\alpha \cdot (\beta \top \gamma) = \alpha \cdot \beta \top \alpha \cdot \gamma.$$

Lemma *function_prod_distr_l*
 $\{A\ B\ C\ D : eqType\}$ $\{alpha : Rel\ A\ B\}$ $\{$beta $: Rel\ B\ C\}$ $\{gamma : Rel\ B\ D\}$:
 function_r alpha $\rightarrow$ alpha $\quad$ Rel_prod beta *gamma = Rel_prod (alpha $\quad$* beta$)$ $(alpha$
gamma$)$.
Proof.
move $\Rightarrow H$.
rewrite $/Rel\_prod$.
rewrite *comp_assoc comp_assoc*.
apply $(function\_cap\_distr\_l\ H)$.
Qed.

> **Lemma 315 (comp_prod_universal)** *Let $\alpha : A \rightharpoonup B$, $\beta : B \rightharpoonup C$ and $\gamma : D \rightharpoonup E$. Then,*
> $$\alpha \cdot (\beta \top \nabla_{BD} \cdot \gamma) = \alpha \cdot \beta \top \nabla_{AD} \cdot \gamma.$$

Lemma *comp_prod_universal*
 $\{A\ B\ C\ D\ E : eqType\}$ $\{alpha : Rel\ A\ B\}$ $\{$beta $: Rel\ B\ C\}$ $\{gamma : Rel\ D\ E\}$:
 alpha $\quad$ Rel_prod beta $(\quad B\ D\quad gamma) = Rel\_prod\ (alpha\quad$ beta$)\ (\quad A\ D\quad gamma)$.
Proof.
apply *inc_antisym*.
apply $(@inc\_trans\ \_\ \_\ \_\ \_\ \_\ (comp\_prod\_distr\_l))$.
apply *prod_inc_compat_l*.
rewrite -*comp_assoc*.
apply *comp_inc_compat_ab_a'b*.
apply *inc_alpha_universal*.

```
rewrite /Rel_prod.
rewrite comp_assoc.
apply (@inc_trans _ _ _ _ _ (dedekind1)).
apply comp_inc_compat_ab_ab'.
apply cap_inc_compat_l.
rewrite comp_assoc comp_assoc -comp_assoc.
apply comp_inc_compat_ab_a'b.
apply inc_alpha_universal.
Qed.
```

Lemma 316 (fst_cap_snd_distr) *Let $u, v : A \times B \to A \times B$ and $u, v \sqsubseteq id_{A \times B}$, $p : B \times C \to B$ and $q : B \times C \to C$ be projections. Then,*

$$p^{\sharp} \cdot (u \sqcap v) \cdot q = p^{\sharp} \cdot u \cdot q \sqcap p^{\sharp} \cdot v \cdot q.$$

```
Lemma fst_cap_snd_distr
 {A B : eqType} {u v : Rel (prod_eqType A B) (prod_eqType A B)}:
 u     Id (prod_eqType A B) → v     Id (prod_eqType A B) →
 fst_r A B #     (u     v)     snd_r A B =
 ((fst_r A B #     u)     snd_r A B)     ((fst_r A B #     v)     snd_r A B).
Proof.
move ⇒ H H0.
apply inc_antisym.
apply (fun H' ⇒ @inc_trans _ _ _ _ _ H' (comp_cap_distr_r)).
apply comp_inc_compat_ab_a'b.
apply comp_cap_distr_l.
apply (@inc_trans _ _ _ _ _ (dedekind1)).
rewrite -(dedekind_id3 H H0) -(@comp_assoc _ _ _ _ _ u) (@comp_assoc _ _ _ _ (fst_r A
B #     u) v).
apply comp_inc_compat_ab_ab'.
rewrite cap_comm comp_assoc -comp_assoc.
apply (@inc_trans _ _ _ _ _ (dedekind2)).
apply comp_inc_compat_ab_b.
rewrite comp_inv comp_inv inv_invol -fst_snd_cap_id.
apply cap_inc_compat.
rewrite comp_assoc (dedekind_id1 H).
apply (comp_inc_compat_ab_b H).
rewrite -comp_assoc (dedekind_id1 H0).
apply (comp_inc_compat_ab_a H0).
Qed.

End main.
```

Chapter 13

Library Point_Axiom

Require Import *Basic_Notations_Set*.
Require Import *Basic_Lemmas*.
Require Import *Relation_Properties*.
Require Import *Functions_Mappings*.
Require Import *Dedekind*.
Require Import *Logic.IndefiniteDescription*.

Module *main* (*def* : *Relation*).
Import *def*.
Module *Basic_Lemmas* := *Basic_Lemmas.main def*.
Module *Relation_Properties* := *Relation_Properties.main def*.
Module *Functions_Mappings* := *Functions_Mappings.main def*.
Module *Dedekind* := *Dedekind.main def*.
Import *Basic_Lemmas Relation_Properties Functions_Mappings Dedekind*.

13.1   I-

13.1.1   I-

$$
\begin{array}{l}
\text{Dedekind} \qquad\qquad X \quad \text{I-} \quad x \quad , \qquad x : I \rightharpoonup X \qquad\qquad , \qquad x \dot{\in} X \\
\qquad\qquad . \qquad\qquad \rho : I \rightharpoonup X \quad \text{I-} \quad x : I \rightharpoonup X \qquad\qquad , \qquad x \dot{\in} \rho \quad x \sqsubseteq \rho \\
\qquad . \\
\qquad \text{I-} \qquad\quad x \dot{\in} X \quad x \dot{\in} \nabla_{IX} \qquad\qquad\qquad\qquad\qquad .
\end{array}
$$

Definition *point_inc* {*X* : *eqType*} (*x rho* : *Rel i X*) := *function_r x* ∧ *x*   *rho*.
Definition *point* {*X* : *eqType*} (*x* : *Rel i X*) := *point_inc x* (*i X*).

13.1.2   I-

Lemma 317 (point_property1) *Let $x, y \dot{\in} X$. Then,*

$$x = y \Leftrightarrow x \cdot y^{\sharp} = id_I.$$

Lemma *point_property1* $\{X : eqType\}$ $\{x\ y : Rel\ i\ X\}$:
  point $x \to point\ y \to (x = y \leftrightarrow x \qquad y \# = Id\ i)$.
Proof.
move $\Rightarrow H\ H0$.
split; move $\Rightarrow H1$.
apply *inc_antisym*.
rewrite *unit_identity_is_universal*.
apply *inc_alpha_universal*.
rewrite *H1*.
apply *H0*.
apply *Logic.eq_sym*.
apply *function_inc*.
apply *H0*.
apply *H*.
rewrite $-(@comp\_id\_l$ _ _ $y)$ *-H1 comp_assoc*.
apply *comp_inc_compat_ab_a*.
apply *H0*.
Qed.

Lemma 318 (point_property2a, point_property2b) *Let $\rho : I \to X$ be a total re-
lation. Then,*

$$\rho \cdot \rho^{\sharp} = \rho \cdot \nabla_{XI} = id_I.$$

Lemma *point_property2a* $\{X : eqType\}$ $\{rho : Rel\ i\ X\}$:
  total_r rho $\to$ *rho* $\qquad$ *rho* $\# = Id\ i$.
Proof.
move $\Rightarrow H$.
apply *inc_antisym*.
rewrite *unit_identity_is_universal*.
apply *inc_alpha_universal*.
apply *H*.
Qed.

Lemma *point_property2b* $\{X : eqType\}$ $\{rho : Rel\ i\ X\}$:
  total_r rho $\to$ *rho* $\qquad$ *rho* $\# = rho$ $\qquad$ $X\ i$.
Proof.
move $\Rightarrow H$.

```
apply inc_antisym.
apply comp_inc_compat_ab_ab'.
apply inc_alpha_universal.
rewrite (point_property2a H) unit_identity_is_universal.
apply inc_alpha_universal.
Qed.
```

Lemma 319 (point_property3) *Let* $\rho : I \rightarrow X$. *Then,*

$$\exists x \dot{\in} \rho \Rightarrow \text{``}\rho \text{ is total''} \wedge \rho \neq \phi_{IX}.$$

```
Lemma point_property3 {X : eqType} {rho : Rel i X}:
 (∃ x : Rel i X, point_inc x rho) → total_r rho ∧ rho ≠   i X.
Proof.
elim ⇒ x H.
assert (total_r rho).
elim H ⇒ H0 H1.
elim H0 ⇒ H2 H3.
apply (@inc_trans _ _ _ _ _ H2).
apply comp_inc_compat.
apply H1.
apply (@inc_inv _ _ _ _ H1).
split.
apply H0.
move ⇒ H1.
rewrite /total_r in H0.
rewrite H1 comp_empty_l in H0.
apply unit_identity_not_empty.
apply inc_antisym.
apply H0.
apply inc_empty_alpha.
Qed.
```

Lemma 320 (point_property4)

$$\exists x \dot{\in} X \Rightarrow \text{``}\nabla_{IX} \text{ is total''} \wedge \nabla_{IX} \neq \phi_{IX}.$$

```
Lemma point_property4 {X : eqType}:
 (∃ x : Rel i X, point x) → total_r (   i X) ∧ (   i X) ≠   i X.
Proof.
move ⇒ H.
apply (@point_property3 _ (   i X) H).
Qed.
```

13.2 I-

13.2.1

> " "   , I-   .
>
> **Lemma 321 (point_axiom)** *Let $\rho : I \rightarrow X$. Then,*
>
> $$\rho = \sqcup_{x \dot\in \rho} x.$$

Lemma *lemma_for_PA* $\{X : eqType\}$ $\{rho : Rel\ i\ X\}$:
  $(((rho = \quad i\ X) \rightarrow False) \rightarrow False) \rightarrow rho = \quad i\ X.$
Proof.
move $\Rightarrow H$.
case $(@unit\_empty\_or\_universal\ (rho \quad rho\ \#)) \Rightarrow H0$.
apply *inc_antisym*.
apply $(@inc\_trans\ \_\ \_\ \_\ \_\ \_\ (relation\_rel\_inv\_rel))$.
rewrite *H0 comp_empty_l*.
apply *inc_refl*.
apply *inc_empty_alpha*.
apply *False_ind*.
apply *H*.
move $\Rightarrow H1$.
rewrite *H1 comp_empty_l* in *H0*.
apply $(unit\_empty\_not\_universal\ H0)$.
Qed.

Lemma *point_axiom* $\{X : eqType\}$ $\{rho : Rel\ i\ X\}$:
  $rho = \quad \_\{$**fun** $x : Rel\ i\ X \Rightarrow point\_inc\ x\ rho\}\ id.$
Proof.
apply *inc_antisym*.
apply *bool_lemma2*.
assert $((\exists\ x : Rel\ i\ X,\ point\_inc\ x\ ((\ \_\{$**fun** $x : Rel\ i\ X \Rightarrow point\_inc\ x\ rho\}\ id)$
$(\ \_\{$**fun** $x : Rel\ i\ X \Rightarrow point\_inc\ x\ rho\}\ id)\ \hat{}\ )) \rightarrow False)$.
move $\Rightarrow H$.
move : $(point\_property3\ H) \Rightarrow H0$.
apply *H0*.
apply *cap_complement_empty*.
assert $((\exists\ x : Rel\ i\ X,\ point\_inc\ x\ (rho \quad (\ \_\{$**fun** $x : Rel\ i\ X \Rightarrow point\_inc\ x\ rho\}\ id)$
$\hat{}\ )) \rightarrow False)$.
move $\Rightarrow H0$.
apply *H*.
elim $H0 \Rightarrow x\ H1$.

$\exists\, x$.
split.
apply *H1*.
apply *inc_cap*.
split.
assert (*point_inc x rho*).
split.
apply *H1*.
elim *H1* $\Rightarrow$ *H2 H3*.
apply *inc_cap* in *H3*.
apply *H3*.
clear *H1*.
move : *x H2*.
apply *inc_cupP*.
apply *inc_refl*.
elim *H1* $\Rightarrow$ *H2 H3*.
apply *inc_cap* in *H3*.
apply *H3*.
apply *lemma_for_PA*.
move $\Rightarrow$ *H1*.
apply *H0*.
apply *axiom_of_choice*.
rewrite /*total_r*.
remember (*rho*   (_{fun *x* : *Rel i X* $\Rightarrow$ *point_inc x rho*} *id*) ˆ) *as rho'*.
case (@*unit_empty_or_universal* (*rho'*   *rho'* #)) $\Rightarrow$ *H2*.
apply *False_ind*.
apply *H1*.
apply *inc_antisym*.
apply (@*inc_trans* _ _ _ _ _ (*relation_rel_inv_rel*)).
rewrite *H2 comp_empty_l*.
apply *inc_refl*.
apply *inc_empty_alpha*.
rewrite *H2*.
apply *inc_alpha_universal*.
apply *inc_cupP*.
move $\Rightarrow$ beta *H*.
apply *H*.
Qed.

Lemma 322 (PA_corollary1)

$$\nabla_{IX} = \bigsqcup_{x \dot{\in} X} x.$$

Lemma *PA_corollary1* {*X* : *eqType*}:   *i X* =   _{*point*} *id*.
Proof.
apply *point_axiom*.
Qed.

Lemma 323 (PA_corollary2)

$$id_X = \sqcup_{x \dot\in X} x^\sharp \cdot x.$$

Lemma *PA_corollary2* {*X* : *eqType*}:
 Id X =   _{*point*} (fun *x* : *Rel i X* ⇒ *x* #   *x*).
Proof.
rewrite -(@*cap_universal* _ _ (*Id X*)) -*lemma_for_tarski2 PA_corollary1*.
rewrite *comp_cupP_distr_l cap_cupP_distr_l*.
apply *cupP_eq*.
move ⇒ *alpha H*.
apply *inc_antisym*.
rewrite *cap_comm*.
apply (@*inc_trans* _ _ _ _ _ (*dedekind2*)).
rewrite *comp_id_l cap_comm cap_universal*.
apply *inc_refl*.
apply *inc_cap*.
split.
apply *H*.
apply *comp_inc_compat_ab_a'b*.
apply *inc_alpha_universal*.
Qed.

Lemma 324 (PA_corollary3) *Let* $\alpha, \beta : X \rightarrow Y$. *Then,*

$$(\forall x \dot\in X, x \cdot \alpha = x \cdot \beta) \Rightarrow \alpha = \beta.$$

Lemma *PA_corollary3* {*X Y* : *eqType*} {*alpha* beta : *Rel X Y*}:
 (∀ *x* : *Rel i X*, *point x* → *x*   *alpha* = *x*   beta) → *alpha* = beta.
Proof.
move ⇒ *H*.
rewrite -(@*comp_id_l* _ _ *alpha*) -(@*comp_id_l* _ _ beta) *PA_corollary2*.
rewrite *comp_cupP_distr_r comp_cupP_distr_r*.
apply *cupP_eq*.
move ⇒ *gamma H0*.
by [rewrite *comp_assoc comp_assoc* (*H gamma H0*)].
Qed.

Lemma 325 (PA_corollary4) *Let $\alpha : X \rightarrow Y$. Then,*

$$\text{``}\alpha \text{ is total''} \Leftrightarrow \forall x \dot{\in} X, \text{``}x \cdot \alpha \text{ is total''}.$$

Lemma *PA_corollary4* {*X Y : eqType*} {*alpha : Rel X Y*}:
 total_r alpha $\leftrightarrow \forall$ *x : Rel i X*, *point x* $\rightarrow$ *total_r* (*x   alpha*).
Proof.
split; move $\Rightarrow$ *H*.
move $\Rightarrow$ *x H0*.
apply *total_comp*.
apply *H0*.
apply *H*.
rewrite /*total_r*.
rewrite *PA_corollary2*.
apply *inc_cupP*.
move $\Rightarrow$ *x H0*.
move : (*H x H0*) $\Rightarrow$ *H1*.
apply (@*inc_trans* _ _ _ ((*x* #   ((*x   alpha*)   (*x   alpha*) #))   *x*)).
apply *comp_inc_compat_ab_a'b*.
apply (*comp_inc_compat_a_ab H1*).
rewrite *comp_inv* -*comp_assoc* -*comp_assoc* -*comp_assoc*.
rewrite *comp_assoc* (@*comp_assoc* _ _ _ _ (*x* #   *x*)).
apply (@*inc_trans* _ _ _ ((*x* #   *x*)   (*alpha   alpha* #))).
apply *comp_inc_compat_ab_a*.
apply *H0*.
apply *comp_inc_compat_ab_b*.
apply *H0*.
Qed.

Lemma 326 (PA_corollary5) *Let $\alpha : X \rightarrow Y$. Then,*

$$\text{``}\alpha \text{ is univalent''} \Leftrightarrow \forall x \dot{\in} X, \text{``}x \cdot \alpha \text{ is univalent''}.$$

Lemma *PA_corollary5* {*X Y : eqType*} {*alpha : Rel X Y*}:
 univalent_r alpha $\leftrightarrow \forall$ *x : Rel i X*, *point x* $\rightarrow$ *univalent_r* (*x   alpha*).
Proof.
split; move $\Rightarrow$ *H*.
move $\Rightarrow$ *x H0*.
apply *univalent_comp*.
apply *H0*.
apply *H*.
rewrite /*univalent_r*.
rewrite -(@*comp_id_r* _ _ (*alpha* #)) *PA_corollary2*.

```
rewrite comp_cupP_distr_l comp_cupP_distr_r.
apply inc_cupP.
move ⇒ x H0.
move : (H x H0) ⇒ H1.
rewrite -comp_assoc -comp_inv comp_assoc.
apply H1.
Qed.
```

13.2.2

Lemma 327 (total_axiom) *Let $\rho : I \rightarrow X$. Then,*

$$\rho \neq \phi_{IX} \Rightarrow id_I = \rho \cdot \rho^{\sharp}.$$

```
Lemma total_axiom {X : eqType} {rho : Rel i X}:
  rho ≠    i X → Id i = rho    rho #.
Proof.
move ⇒ H.
case (@unit_empty_or_universal (rho    rho #)) ⇒ H0.
apply False_ind.
apply H.
apply inc_antisym.
apply (@inc_trans _ _ _ _ _ (relation_rel_inv_rel)).
rewrite H0 comp_empty_l.
apply inc_refl.
apply inc_empty_alpha.
by [rewrite H0 unit_identity_is_universal].
Qed.
```

Lemma 328 (Tot_corollary1) *Let $\rho : I \rightarrow X$ and $x \dot{\in} X$. Then,*

$$\rho \sqsubseteq x \Rightarrow \rho = \phi_{IX} \vee \rho = x.$$

```
Lemma Tot_corollary1 {X : eqType} {rho x : Rel i X}:
  point x → rho    x → rho =    i X ∨ rho = x.
Proof.
move ⇒ H H0.
case (@unit_empty_or_universal (rho    rho #)) ⇒ H1.
left.
apply inc_antisym.
apply (@inc_trans _ _ _ _ _ (relation_rel_inv_rel)).
rewrite H1 comp_empty_l.
```

```
apply inc_refl.
apply inc_empty_alpha.
right.
apply inc_antisym.
apply H0.
rewrite -(@comp_id_l _ _ x) unit_identity_is_universal -H1 comp_assoc.
apply (@inc_trans _ _ _ (rho    (x #    x))).
apply comp_inc_compat_ab_ab'.
apply comp_inc_compat_ab_a'b.
apply (@inc_inv _ _ _ _ H0).
apply comp_inc_compat_ab_a.
apply H.
Qed.
```

Lemma 329 (Tot_corollary2) *Let* $x, y \dot{\in} X$. *Then,*

$$x \neq y \Leftrightarrow x \cdot y^\sharp = \phi_{II}.$$

```
Lemma Tot_corollary2 {X : eqType} {x y : Rel i X}:
 point x → point y → (x ≠ y ↔ x    y # =    i i).
Proof.
move ⇒ H H0.
assert (x = y ↔ x    y # ≠    i i).
rewrite (point_property1 H H0).
split; move ⇒ H1.
rewrite H1.
apply unit_identity_not_empty.
case (@unit_empty_or_universal (x    y #)) ⇒ H2.
apply False_ind.
apply (H1 H2).
by [rewrite H2 unit_identity_is_universal].
rewrite H1.
split; move ⇒ H2.
apply (lemma_for_PA H2).
move ⇒ H3.
apply (H3 H2).
Qed.
```

Lemma 330 (Tot_corollary3) *Let* $f : (I \rightarrow X) \rightarrow (I \rightarrow Y)$. *Then,*

$$(\forall x \dot{\in} X, \text{``} f(x) \text{ is a function''}) \Rightarrow \text{``} \sqcup_{x \dot{\in} X} x^\sharp \cdot f(x) \text{ is a function''}.$$

Lemma *Tot_corollary3* {X Y : *eqType*} {f : *Rel i X* → *Rel i Y*}:

$(\forall\ x :\ Rel\ i\ X,\ point\ x \to function\_r\ (f\ x)) \to function\_r\ (\ \_\{point\}\ (\texttt{fun}\ x :\ Rel\ i\ X$
$\Rightarrow x\ \#\ \ \ f\ x)).$
Proof.
move $\Rightarrow H$.
assert $(\forall\ x :\ Rel\ i\ X,\ point\ x \to x\ \ \ (\ \ \_\{point\}\ (\texttt{fun}\ x0 :\ Rel\ i\ X \Rightarrow x0\ \#\ \ \ f\ x0)) =$
$f\ x).$
move $\Rightarrow x\ H0$.
assert $(x\ \ \ x\ \#\ =\ Id\ i).$
apply $inc\_antisym$.
rewrite $unit\_identity\_is\_universal$.
apply $inc\_alpha\_universal$.
apply $H0$.
rewrite $-(@comp\_id\_l\ \_\ \_\ (f\ x))$ -$H1$.
apply $inc\_antisym$.
rewrite $comp\_cupP\_distr\_l$.
apply inc_cupP.
move $\Rightarrow y\ H2$.
rewrite -$comp\_assoc$.
case $(@unit\_empty\_or\_universal\ (x\ \ \ y\ \#)) \Rightarrow H3$.
rewrite $H3\ comp\_empty\_l$.
apply inc_empty_alpha.
rewrite -$unit\_identity\_is\_universal$ in $H3$.
apply $(point\_property1\ H0\ H2)$ in $H3$.
rewrite $H3$.
apply inc_refl.
rewrite $comp\_assoc$.
apply $comp\_inc\_compat\_ab\_ab'$.
clear $H1$.
move : $x\ H0$.
apply inc_cupP.
apply inc_refl.
split.
rewrite $PA\_corollary4$.
move $\Rightarrow x\ H1$.
rewrite $(H0\ x\ H1)$.
apply $(H\ x\ H1)$.
rewrite $PA\_corollary5$.
move $\Rightarrow x\ H1$.
rewrite $(H0\ x\ H1)$.
apply $(H\ x\ H1)$.
Qed.

13.2.3

Lemma 331 (nonempty_axiom) *Let $\rho : I \rightarrow X$. Then,*

$$\rho \neq \phi_{IX} \Rightarrow \exists x \dot{\in} \rho.$$

Lemma *nonempty_axiom* $\{X : eqType\}$ $\{rho : Rel\ i\ X\}$:
 $rho \neq\quad i\ X \rightarrow \exists\ x : Rel\ i\ X,\ point\_inc\ x\ rho.$
Proof.
move : $(@axiom\_of\_choice\ \_\ \_\ rho) \Rightarrow H.$
move $\Rightarrow H0.$
apply $H.$
rewrite $/total\_r.$
rewrite $(total\_axiom\ H0).$
apply $inc\_refl.$
Qed.

Lemma 332 (axiom_of_subobjects2) *Let $\rho : I \rightarrow X$. Then,*

$$\exists S, \exists j : S \rightarrow X, \rho = \nabla_{IS} \cdot j \wedge j \cdot j^{\sharp} = id_S.$$

Lemma *axiom_of_subobjects2* $\{X : eqType\}$ $\{rho : Rel\ i\ X\}$:
 $\exists\ (S : eqType)(j : Rel\ S\ X),\ rho =\quad i\ S\quad j \wedge j\quad j\ \# = Id\ S.$
Proof.
elim $(@rationality\ \_\ \_\ rho) \Rightarrow R.$
elim $\Rightarrow f.$
elim $\Rightarrow g.$
elim $\Rightarrow H.$
elim $\Rightarrow H0.$
elim $\Rightarrow H1\ H2.$
$\exists\ R.$
$\exists\ g.$
split.
rewrite $H1.$
apply $inc\_antisym.$
apply $comp\_inc\_compat\_ab\_a'b.$
apply $inc\_alpha\_universal.$
apply $comp\_inc\_compat\_ab\_a'b.$
apply $(@inc\_trans\ \_\ \_\ \_\ (\quad i\ R\quad (f\quad f\ \#))).$
apply $comp\_inc\_compat\_a\_ab.$
apply $H.$
rewrite $-comp\_assoc.$
apply $comp\_inc\_compat\_ab\_b.$

```
rewrite unit_identity_is_universal.
apply inc_alpha_universal.
rewrite -H2 cap_comm inc_def1.
assert ((f #     g)     rho).
rewrite H1.
apply inc_refl.
apply (function_move1  H) in H3.
apply (@inc_trans _ _ _ ((f     rho)     (f     rho) #)).
apply comp_inc_compat.
apply H3.
apply (@inc_inv _ _ _ _ H3).
rewrite comp_inv comp_assoc -(@comp_assoc _ _ _ _ rho).
apply comp_inc_compat_ab_ab'.
apply comp_inc_compat_ab_b.
rewrite unit_identity_is_universal.
apply inc_alpha_universal.
Qed.
```

13.3

Lemma 333 (point_atomic) *Let $x \dot{\in} X$, then x is atomic.*

```
Lemma point_atomic {X : eqType} {x : Rel i X}: point x → atomic x.
Proof.
move ⇒ H.
split.
move : (@point_property3 X x) ⇒ H0.
apply H0.
∃ x.
split.
apply H.
apply inc_refl.
move ⇒ beta.
apply (Tot_corollary1  H).
Qed.
```

Lemma 334 (point_atomic2) *Let $x \dot{\in} X$ and $y \dot{\in} Y$, then $x^{\sharp} \cdot y$ is atomic.*

```
Lemma point_atomic2 {X  Y : eqType} {x : Rel i X} {y : Rel i Y}:
 point x → point y → atomic (x #     y).
Proof.
```

```
move ⇒ H H0.
split.
move ⇒ H1.
assert (Id i = (x    x #)    (y    y #)).
apply inc_antisym.
rewrite -(@comp_id_l _ _ (Id i)).
apply comp_inc_compat.
apply H.
apply H0.
rewrite unit_identity_is_universal.
apply inc_alpha_universal.
rewrite comp_assoc -(@comp_assoc _ _ _ _ (x #)) in H2.
rewrite H1 comp_empty_l comp_empty_r in H2.
apply (unit_identity_not_empty H2).
move ⇒ beta H1.
case (@unit_empty_or_universal ((   i X    beta)       Y i)) ⇒ H2.
left.
apply inc_antisym.
replace (   X Y) with ((   X i        i i)       i Y).
rewrite -H2 -comp_assoc -comp_assoc unit_universal.
rewrite comp_assoc unit_universal.
apply (@inc_trans _ _ _ (   X X    beta)).
apply comp_inc_compat_b_ab.
apply inc_alpha_universal.
apply comp_inc_compat_a_ab.
apply inc_alpha_universal.
by [rewrite comp_empty_r comp_empty_l].
apply inc_empty_alpha.
right.
apply inc_antisym.
apply H1.
assert (beta ≠    X Y).
move ⇒ H3.
rewrite H3 comp_empty_r comp_empty_l in H2.
apply (unit_empty_not_universal H2).
apply (@inc_trans _ _ _ (x #    (x    beta))).
apply comp_inc_compat_ab_ab'.
assert ((x    beta)    y).
apply (@inc_trans _ _ _ (x    (x #    y))).
apply (comp_inc_compat_ab_ab' H1).
rewrite -comp_assoc.
apply comp_inc_compat_ab_b.
```

```
rewrite unit_identity_is_universal.
apply inc_alpha_universal.
apply inc_def1 in H1.
rewrite H1 in H3.
assert (x #   ((x    beta)   y) ≠   X Y).
move ⇒ H5.
apply H3.
apply inc_antisym.
rewrite cap_comm.
apply (@inc_trans _ _ _ _ _ (dedekind1)).
rewrite cap_comm inv_invol H5.
apply inc_refl.
apply inc_empty_alpha.
case (Tot_corollary1 H0 H4) ⇒ H6.
rewrite H6 cap_comm cap_empty comp_empty_r in H5.
apply False_ind.
by [apply H5].
rewrite H6.
apply inc_refl.
rewrite -comp_assoc.
apply comp_inc_compat_ab_b.
apply H.
Qed.

End main.
```

Bibliography

[1] R. Affeldt and M. Hagiwara. Formalization of Shannon ' s Theorems in SSReflect-Coq. In 3rd Conference on Interactive Theorem Proving, LNCS 7406, 233–249, 2012.