

Formalizing Relations in Type Theory

Farida Kachapova

Department of Mathematical Sciences, Auckland University of Technology, New Zealand

Article history

Received: 02-03-2022

Revised: 21-07-2022

Accepted: 01-09-2022

Email: farida.kachapova@aut.ac.nz

Abstract: Type theory plays an important role in the foundations of mathematics as a framework for formalizing mathematics and a base for proof assistants providing semi-automatic proof checking and construction. Derivation of each theorem in type theory results in a formal term encapsulating the whole proof process. This study uses a variant of type theory, namely the Calculus of Constructions with Definitions, to formalize the standard theory of binary relations. This includes basic operations on relations, criteria for special properties of relations, invariance of these properties under the basic operations, equivalence relation, well-ordering, and transfinite induction. Definitions and proofs are presented as flag-style derivations.

Keywords: Type Theory, Calculus of Constructions, Binary Relation, Transfinite Induction, Flag-Style Derivation

1. Introduction

Type theories were developed as alternatives to set theory for the foundation of mathematics. Important type theories were introduced by A. Church and P. Martin-Lof; they are typed λ -calculus (see, for example, Barendregt (2012)) and intuitionistic type theory (see, for example, Granstrom (2011)). There are several higher-order variants of typed λ -calculus, such as Calculus of Constructions (CoC) and Calculus of Inductive Constructions (CIC) (Bertot and Casteran, 2013). These variants make formal bases of proof assistants, which are computer tools for formalizing and developing mathematics. In particular, the well-known proof assistant Coq (Coq Development Team, 2021) is based on the CIC.

This study uses the variant λD of CoC developed by Nederpelt and Geuvers (2014); λD is called the Calculus of Constructions with Definitions. λD is chosen because of its following useful properties described in their book.

- In λD , as in other variants of CoC, proofs are expressed as formal terms and thus are incorporated into the system.
- In λD type checking is decidable and therefore proof checking is decidable. So the correctness of a proof can be checked by an algorithm.
- λD is strongly normalizing, which implies the logical consistency of this theory, even with classical logic (when no extra axioms are added).

The theory λD is weaker than CIC because λD does not have inductive types. This does not limit its capability for formalizing mathematics because in λD one can use an

axiomatic approach and higher-order logic to express the objects that CIC defines with inductive types.

In these formalizations, the author aims to keep the language and theorems as close as possible to the ones of standard mathematics. Definitions and proofs in this study use the flag-style derivation described in Nederpelt and Geuvers (2014). Long formal derivations are moved from the main text to Appendices for better readability.

2. Type Theory λD

Nederpelt and Geuvers (2014) developed a formal theory λD and formalized some parts of logic and mathematics in it. The main features of λD are briefly described below.

2.1. Type Theory λD

The language of λD described in Nederpelt and Geuvers (2014) has an infinite set of variables, V , and an infinite set of constants, C ; these two sets are disjoint. There are also special symbols \square and $*$.

Definition 2.1

Expressions of the language are defined recursively as follows.

- (1) Each variable is an expression.
- (2) Each constant is an expression.
- (3) The constant $*$ is an expression.
- (4) The constant \square is an expression.
- (5) (Application) If A and B are expressions, then AB is an expression.

- (6) (Abstraction) If A and B are expressions and x is a variable, then $\lambda x: A.B$ is an expression.
- (7) (Dependent Product) If A and B are expressions and x is a variable, then $\Pi x: A.B$ is an expression.
- (8) If A_1, A_2, \dots, A_n are expressions and c is a constant, then $c(A_1, A_2, \dots, A_n)$ is an expression.

An expression $A \rightarrow B$ is introduced as a particular type of Dependent Product from (7) when x is not a free variable in B .

Definition 2.2

- (1) A **statement** is of the form $M: N$, where M and N are expressions.
- (2) A **declaration** is of form $x: N$, where x is a variable and N is an expression.
- (3) A **descriptive definition** is of the form:

$$\bar{x} : \bar{A} \triangleright c(\bar{x}) := M : N,$$

where, \bar{x} is a list x_1, x_2, \dots, x_n of variables, \bar{A} is a list A_1, A_2, \dots, A_n of expressions, c is a constant, and M and N are expressions.

- (4) A **primitive definition** is of the form:

$$\bar{x} : \bar{A} \triangleright c(\bar{x}) := \perp : N,$$

where, \bar{x} , \bar{A} , and c are described the same way as in (3), and N is an expression. The symbol \perp denotes the non-existing definiens. Primitive definitions are used for introducing axioms where no proof terms are needed.

- (5) A **definition** is a descriptive definition or a primitive definition.
- (6) A **judgment** is of the form:

$$\Delta; \Gamma \vdash M : N,$$

where, M and N are expressions of the language, Δ is an environment (a properly constructed sequence of definitions) and Γ is a context (a properly constructed sequence of declarations).

For brevity, most definitions use implicit variables by omitting the previously declared variables \bar{x} in $c(\bar{x})$ in (3) and (4).

The following informally explains the meaning of expressions.

- (1) If an expression M appears in a derived statement of the form $M: *$, then M is interpreted as a **type**, which represents a set or a proposition.

Note: There is only one type $*$ in λD . But informally $*_p$ is often used for propositions and $*_s$ for sets to make proofs more readable.

- (2) If an expression M appears in a derived statement of the form $M: N$, where N is a type, then M is interpreted as an object at the lowest level. When N is interpreted as a set, then M is regarded as an element of this set. When N is interpreted as a proposition, then M is regarded as a proof (or a proof term) of this proposition.
- (3) The symbol \square represents the highest level.
- (4) **Sort** is $*$ or. Letters s, s_1, s_2, \dots are used as variables for sorts.
- (5) If an expression M appears in a statement of the form $M: \square$, then M is called a **kind**.

λD contains the derivation rule:

$$\phi; \phi \vdash *: \square,$$

It is an axiom (the only axiom) of λD , because it has an empty environment and an empty context.

Further details of the language and derivation rules of the theory λD can be found in Nederpelt and Geuvers (2014). Judgments are formally derived in λD using the derivation rules.

2.2. Flag Format of Derivations

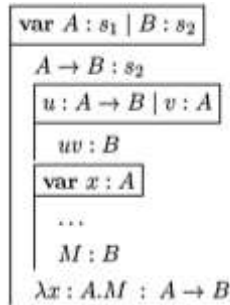
The flag-style deduction was introduced by Jaskowski and Fitch; it is described in detail by Nederpelt and Kamareddine (2004), and Nederpelt and Geuvers (2014). In short, a derivation in the flag format is a linear deduction. Each "flag" (a rectangular box) contains a declaration that introduces a variable or an assumption; a collection of already introduced variables and assumptions makes the current context. The scope of the variable or assumption is established by the "flag pole". In the scope, one constructs definitions and proof terms for proving statements/theorems in λD . Each new flag extends the context and at the end of each flag pole, the context is reduced by the corresponding declaration. For brevity, several declarations can be combined in one flag.

2.3. Logic in λD

The rules of intuitionistic logic are derived from the theory λD as shown in Nederpelt and Geuvers (2014). These are briefly described below by showing the introduction and elimination rules for logical connectives and quantifiers.

Implication

The logical implication $A \Rightarrow B$ is identified with the arrow type $A \rightarrow B$. The rules for implication follow the following general rules for the arrow type (here they are written in the flag format):

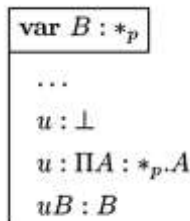


Here x is not a free variable in B .

In λD arrows are right-associative, that is $A \rightarrow B \rightarrow C$ is a shorthand for $A \rightarrow (B \rightarrow C)$.

Falsity and Negation

Falsity \perp is introduced in λD by: $\perp := \Pi A : *_{\mathcal{P}}.A : *_{\mathcal{P}}$.
 From this definition one gets a rule for falsity:



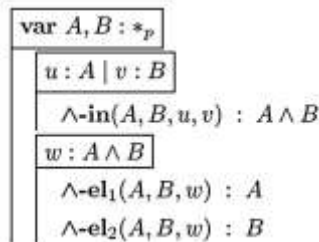
The rule states that falsity implies any proposition.
 As usual, negation is defined by: $\neg A := A \rightarrow \perp$.

Other logical connectives and quantifiers are also defined using second-order encoding. Here only a list of their derived rules and names of the corresponding terms are provided, without details of their construction. The exact values of the terms can be found in Nederpelt and Geuvers (2014).

Some of our flag derivations contain the proof terms that will be re-used in other proofs; such proof terms are written in bold font, e.g., $\mathbf{\Lambda-in}$ in the first derived rule for conjunction as follows.

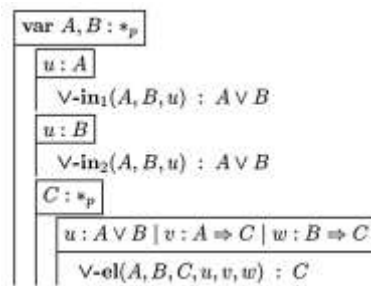
Conjunction

These are derived rules for conjunction \wedge :



Disjunction

These are derived rules for disjunction \vee :



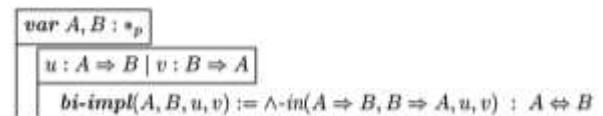
Bi-Implication

Bi-implication \Leftrightarrow has the standard definition:

$$(A \Leftrightarrow B) := (A \Rightarrow B) \wedge (B \Rightarrow A).$$

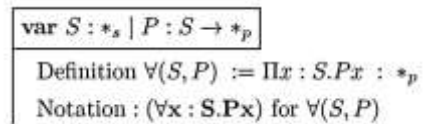
Lemma 2.3.

This lemma will be often used to prove bi-implication $A \Leftrightarrow B$.



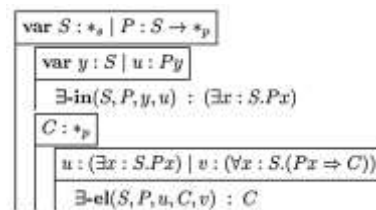
Universal Quantifier

The universal quantifier \forall is defined through the dependent product:



Existential Quantifier

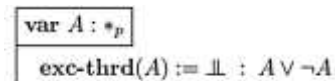
These are derived rules for the existential quantifier \exists :



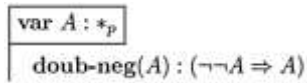
Here x is not a free variable in C .

Classical Logic

This study uses mostly intuitionistic logic. But sometimes classical logic is needed; in these cases, the following Axiom of Excluded Third is added:

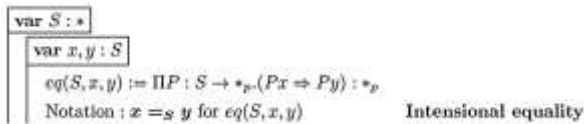


This axiom implies the Double Negation theorem:



3. Intensional Equality in λD

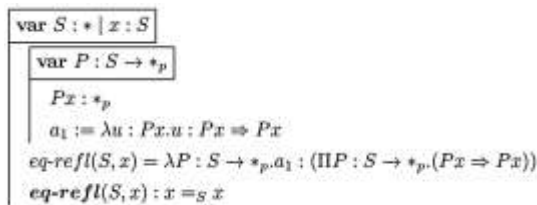
This section introduces intensional equality for elements of any type; it is called just equality. The next section will introduce extensional equality and the axiom of extensionality relating to the two types of equality.



3.1. Properties of Equality

Reflexivity

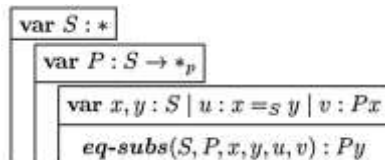
The following diagram proves the reflexivity property of equality in λD :



Proof terms are constructed similarly for the following properties of Substitutivity, Congruence, Symmetry, and Transitivity (see Nederpelt and Geuvers (2014)).

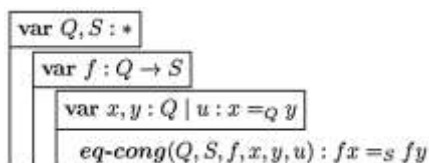
Substitutivity

Substitutivity means that equality is consistent with predicates of corresponding types.



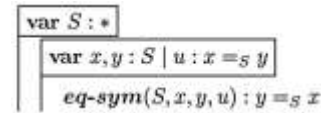
Congruence

Congruence means that equality is consistent with functions of corresponding types.



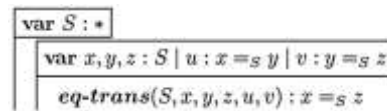
Symmetry

The following diagram expresses the symmetry property of equality in λD .



Transitivity

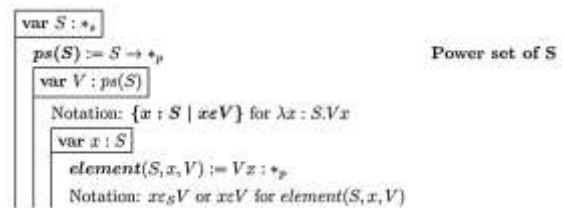
The following diagram expresses the transitivity property of equality in λD .



4. Relations in Type Theory

4.1. Sets in λD

Below are some definitions from Nederpelt and Geuvers (2014) relating to sets, in particular, subsets of type S :

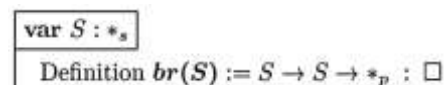


Thus, a subset V of S is regarded as a predicate on S and $x \in V$ means x satisfies the predicate V .

4.2. Defining Binary Relations in λD

Binary relations are introduced in Nederpelt and Geuvers (2014), together with the properties of reflexivity, symmetry, antisymmetry, and transitivity, and definitions of equivalence relation and partial order. These are used here as a starting point for formalizing the theory of binary relations in λD .

A relation on S is a binary predicate on S , which is regarded in λD as a composition of unary predicates. The type $br(S)$ of all binary relations on S is introduced below, for brevity:



In the rest of the article, binary relations are called just relations. The equality of relations and operations on relations are defined similarly to the set equality and set operations.

Next, the extensional equality of relations is defined vs the intentional equality introduced in the previous section.

var $S : *_s$	
var $R, Q : br(S)$	
Definition $\subseteq (S, R, Q) := (\forall x, y : S. (Rxy \Rightarrow Qxy)) : *_p$	Extensional equality
Notation $R \subseteq Q$ for $\subseteq (S, R, Q)$	
Definition $Ex\text{-}eq(S, R, Q) := R \subseteq Q \wedge Q \subseteq R : *_p$	
Notation $R = Q$ for $Ex\text{-}eq(S, R, Q)$	

The following axiom of extensionality for relations is added to the theory λD .

var $S : *_s$	
var $R, Q : br(S)$	
u : $R = Q$	
ext-axiom(S, R, Q, u) := $\perp \vdash R =_{br(S)} Q$ Extensionality Axiom	

The axiom is introduced in the last line by a primitive definition with the symbol \perp replacing a non-existing proof term. The Extensionality Axiom states that the two types of equality are the same for binary relations. So the symbol $=$ will be used for both without elaborating on details of applying the axiom of extensionality, when converting one type of equality to the other.

4.3. Operations on Binary Relations

The flag format is used to introduce the identity relation id_S on type S and converse R^{-1} of a relation R :

var $S : *_s$	
Definition $id_S := \lambda x, y : S. (x =_S y) : br(S)$	Identity relation
var $R : br(S)$	
Definition $conv(S, R) := \lambda x, y : S. (Ryx) : br(S)$	Converse relation
Notation R^{-1} for $conv(S, R)$	

Next, the operations of union \cup , intersection \cap , and composition \circ of relations are introduced:

var $S : *_s$	
var $R, Q : br(S)$	
Definition $\cup (S, R, Q) := \lambda x, y : S. (Rxy \vee Qxy) : br(S)$	Union
Notation $R \cup Q$ for $\cup (S, R, Q)$	
Definition $\cap (S, R, Q) := \lambda x, y : S. (Rxy \wedge Qxy) : br(S)$	Intersection
Notation $R \cap Q$ for $\cap (S, R, Q)$	
Definition $\circ (S, R, Q) := \lambda x, y : S. (\exists z : S. (Rxz \wedge Qzy)) : br(S)$	Composition
Notation $R \circ Q$ for $\circ (S, R, Q)$	

4.4. Properties of Operations

The following two technical lemmas will be used in some future proofs.

Lemma 4.1.

This lemma gives a shortcut for constructing an element of a composite relation.

var $S : *_s \mid R, Q : br(S) \mid x, y, z : S$	
u : $Rxy \mid v : Qyz$	
a := $\wedge\text{-in} (Rxy, Qyz, u, v) : Rxy \wedge Qyz$	
prod-term (S, R, Q, x, y, z, u, v) := $\exists\text{-in} (S, \lambda t. Rxt \wedge Qtz, y, a) : (R \circ Q)xz$	

Lemma 4.2.

This lemma gives a shortcut for proving the equality of two relations:

var $S : *_s \mid R, Q : br(S)$	
u : $R \subseteq Q \mid v : Q \subseteq R$	
rel-equal(S, R, Q, u, v) := $\wedge\text{-in} (R \subseteq Q, Q \subseteq R, u, v) : R = Q$	

Theorem 4.3.

For relations R, P , and Q on type S the following hold:

- 1) $(R^{-1})^{-1} = R$
- 2) $(R \circ Q)^{-1} = R^{-1} \circ Q^{-1}$
- 3) $(R \cap Q)^{-1} = R^{-1} \cap Q^{-1}$
- 4) $(R \cup Q)^{-1} = R^{-1} \cup Q^{-1}$
- 5) $R \circ (P \cup Q) = R \circ P \cup R \circ Q$
- 6) $(P \cup Q) \circ R = P \circ R \cup Q \circ R$
- 7) $R \circ (P \cap Q) \subseteq R \circ P \cap R \circ Q$
- 8) $(P \cap Q) \circ R \subseteq P \circ R \cap Q \circ R$
- 9) $(R \circ Q) \circ Q = R \circ (P \circ Q)$

The formal proof is in Appendix A. The proof of part 2) has the form:

var $S : *_s \mid R, Q : br(S)$	
...	
conv-prod(S, R, Q) := ... : $(R \circ Q)^{-1} = Q^{-1} \circ R^{-1}$	

Its proof term $conv\text{-}prod (S, R, Q)$ will be re-used later in the paper.

5. Properties of Binary Relations

The properties of reflexivity, symmetry, antisymmetry, transitivity and the relations of equivalence and partial order are defined in Nederpelt and Geuvers (2014) as follows.

var $S : *_s \mid R : br(S)$	
Definition $refl(S, R) := \forall x : S. (Rxx) : *_p$	Reflexivity
Definition $sym(S, R) := \forall x, y : S. (Rxy \Rightarrow Ryx) : *_p$	
Definition $antisym(S, R) := \forall x, y : S. (Rxy \wedge Ryx \Rightarrow x =_S y) : *_p$	Antisymmetry
Definition $trans(S, R) := \forall x, y, z : S. (Rxy \wedge Ryx \Rightarrow Rxz) : *_p$	
Definition $equiv\text{-}relation(S, R) := refl(S, R) \wedge sym(S, R) \wedge trans(S, R) : *_p$	Equivalence relation
Definition $part\text{-}ord(S, R) := refl(S, R) \wedge antisym(S, R) \wedge trans(S, R) : *_p$	

Theorem 5.1.

Suppose R is a relation on type S . Then the following hold.

- 1) **Criterion of reflexivity.** R is reflexive $\Leftrightarrow id_S \subseteq R$.

- 2) **First criterion of symmetry.** R is symmetric $\Leftrightarrow R^{-1} \subseteq R$.
- 3) **Second criterion of symmetry.** R is symmetric $\Leftrightarrow R^{-1} = R$.
- 4) **Criterion of antisymmetry.** R is antisymmetric $\Leftrightarrow R^{-1} \cap R \subseteq id_S$.
- 5) **Criterion of transitivity.** R is transitive $\Leftrightarrow R \circ R \subseteq R$.

The formal proof is in Appendix B. The proof of part 3) has the form:

```

var S : *s | R : br(S)
...
sym-criterion(S, R) := ... : sym(S, R) ⇔ R-1 = R
    
```

Its proof term *sym-criterion* (S, R) will be re-used later in the paper.

Theorem 5.2.

Relation R on S is reflexive, symmetric, and antisymmetric $\Rightarrow R = id_S$.

Proof. The formal proof is in the following flag diagram.

```

var S : *s | R : br(S)
  u1 : refl(S, R) | u2 : sym(S, R) | u3 : antisym(S, R)
    var x, y : S | v : Rxy
      a1 = u2xyv : Ryx
      a2 = u3xyva1 : x =S y
    a3 := λx, y : S. λv : Rxy. a2 : (R ⊆ idS)
      var x, y : S | v : (idS)xy
        v : x =S y
        Notation P := λz : S. Rzx : S → *p
        a4 = u1x : Rxx
        a4 : Px
        a5 := eq-subst(S, P, x, y, v, a4) : Py
        a5 : Rxy
      a6 := λx, y : S. λv : (idS)xy. a5 : (idS ⊆ R)
      a7 := rel-equal(S, R, idS, a3, a6) : R = idS
    
```

□

Theorem 5.3. Invariance under converse operation.

Suppose R is a relation on type S . Then the following hold:

- 1) R is reflexive $\Rightarrow R^{-1}$ is reflexive
- 2) R is symmetric $\Rightarrow R^{-1}$ is symmetric
- 3) R is antisymmetric $\Rightarrow R^{-1}$ is antisymmetric
- 4) R is transitive $\Rightarrow R^{-1}$ is transitive

Proof. 1)

```

var S : *s | R : br(S)
  u : refl(S, R)
    var x : S
      ux : Rxx
      ux : R-1xx
    a := λx : S. ux : refl(S, R-1)
    
```

2)

```

var S : *s | R : br(S)
  u : sym(S, R)
    var x, y : S | v : R-1xy
      v : Ryx
      uyx : (Ryx ⇒ Rxy)
      a1 := uyxv : Rxy
      a1 : R-1yx
    a2 := λx, y : S. λv : R-1xy. a1 : sym(S, R-1)
    
```

3)

```

var S : *s | R : br(S)
  u : antisym(S, R)
    var x, y : S | v : R-1xy | w : R-1yx
      v : Ryx
      w : Rxy
      uxy : (Rxy ⇒ Ryx ⇒ x = y)
      a1 := uxyvw : x = y
    a2 := λx, y : S. λv : R-1xy. λw : R-1yx. a1 : antisym(S, R-1)
    
```

4)

```

var S : *s | R : br(S)
  u : trans(S, R)
    var x, y, z : S | v : R-1xy | w : R-1yz
      w : Rzy
      v : Ryx
      uzyx : (Rzy ⇒ Ryx ⇒ Rzx)
      a1 := uzyxvw : Rzx
      a1 : R-1xz
    a2 := λx, y, z : S. λv : R-1xy. λw : R-1yz. a1 : trans(S, R-1)
    
```

□

Theorem 5.4. Invariance under intersection.

Suppose R and Q are relations on type S . Then the following hold.

- 1) R and Q are reflexive $\Rightarrow R \cap Q$ is reflexive.
- 2) R and Q are symmetric $\Rightarrow R \cap Q$ is symmetric.
- 3) R or Q is antisymmetric $\Rightarrow R \cap Q$ is antisymmetric.
- 4) R and Q are transitive $\Rightarrow R \cap Q$ is transitive.

Proof. 1)

```

var S : *s | R, Q : br(S)
u : refl(S, R) | v : refl(S, Q)
  var x : S
    a1 := ux : Rxx
    a2 := vx : Qxx
    a3 := ∧-in (Rxx, Qxx, a1, a2) : (R ∩ Q)xx
    a4 := λx : S.a3 : refl(S, R ∩ Q)
    
```

2)

```

var S : *s | R, Q : br(S)
u : sym(S, R) | v : sym(S, Q)
  var x, y : S | w : (R ∩ Q)xy
    w : Rxy ∧ Qxy
    a1 := ∧-el1(Rxy, Qxy, w) : Rxy
    a2 := ∧-el2(Rxy, Qxy, w) : Qxy
    a3 := uxya1 : Ryx
    a4 := vxya2 : Qyx
    a5 := ∧-in (Ryx, Qyx, a3, a4) : (R ∩ Q)yx
    a6 := λx, y : S.λw : (R ∩ Q)xy.a5 : sym(S, R ∩ Q)
    
```

3)

```

var S : *s | R, Q : br(S)
Notation A := antisym(S, R) : *p
Notation B := antisym(S, Q) : *p
Notation C := antisym(S, R ∩ Q) : *p
u : A ∨ B
  v : A
    var x, y : S | w1 : (R ∩ Q)xy | w2 : (R ∩ Q)yx
      w1 : Rxy ∧ Qxy
      a1 := ∧-el1(Rxy, Qxy, w1) : Rxy
      w2 : Ryx ∧ Qyx
      a2 := ∧-el1(Ryx, Qyx, w2) : Ryx
      vxy : (Rxy ⇒ Ryx ⇒ x = y)
      a3 := vxya1a2 : x = y
      a4 := λv : A.λx, y : S.λw1 : (R ∩ Q)xy.λw2 : (R ∩ Q)yx.a3
        : (A ⇒ C)
    v : B
      var x, y : S | w1 : (R ∩ Q)xy | w2 : (R ∩ Q)yx
        w1 : Rxy ∧ Qxy
        a5 := ∧-el2(Rxy, Qxy, w1) : Qxy
        w2 : Ryx ∧ Qyx
        a6 := ∧-el2(Ryx, Qyx, w2) : Qyx
        vxy : (Qxy ⇒ Qyx ⇒ x = y)
        a7 := vxya5a6 : x = y
        a8 := λv : B.λx, y : S.λw1 : (R ∩ Q)xy.λw2 : (R ∩ Q)yx.a7
          : (B ⇒ C)
      a9 := v-el(A, B, C, u, a4, a8) : C
      a9 : antisym(S, R ∩ Q)
    
```

4)

```

var S : *s | R, Q : br(S)
u1 : trans(S, R) | u2 : trans(S, Q)
  var x, y, z : S | v : (R ∩ Q)xy | w : (R ∩ Q)yz
    v : Rxy ∧ Qxy
    a1 := ∧-el1(Rxy, Qxy, v) : Rxy
    a2 := ∧-el2(Rxy, Qxy, v) : Qxy
    w : Ryz ∧ Qyz
    a3 := ∧-el1(Ryz, Qyz, w) : Ryz
    a4 := ∧-el2(Ryz, Qyz, w) : Qyz
    a5 := u1xyza1a3 : Rxz
    a6 := u2xyza2a4 : Qxz
    a7 := ∧-in (Rxz, Qxz, a5, a6) : (R ∩ Q)xz
    a8 := λx, y, z : S.λv : (R ∩ Q)xy.λw : (R ∩ Q)yz.a7
      : trans(S, R ∩ Q)
    
```

□

Theorem 5.5. Invariance under union.

Suppose R and Q are relations on type S . Then the following hold.

- 1) R or Q is reflexive $\Rightarrow R \cup Q$ is reflexive.
- 2) R and Q are symmetric $\Rightarrow R \cup Q$ is symmetric.

Proof. 1)

```

var S : *s | R, Q : br(S)
u : refl(S, R) | x : S
  ux : Rxx
  a1 := v-in1(Rxx, Qxx, ux) : (R ∪ Q)xx
  a2 := v-in2(Rxx, Qxx, ux) : (Q ∪ R)xx
  a3 := λu : refl(S, R).λx : S.a1
    : (refl(S, R) ⇒ refl(S, R ∪ Q))
  a4(R, Q) := λu : refl(S, R).λx : S.a2
    : (refl(S, R) ⇒ refl(S, Q ∪ R))
  a5 := a4(Q, R) : (refl(S, Q) ⇒ refl(S, R ∪ Q))
  u : refl(S, R) ∨ refl(S, Q)
  a7 := v-el(refl(S, R), refl(S, Q), refl(S, R ∪ Q), u, a3, a5)
    : refl(S, R ∪ Q)
    
```

2)

```

var S : *s | R, Q : br(S)
u1 : sym(S, R) | u2 : sym(S, Q)
  var x, y : S | v : (R ∪ Q)xy
    v : Rxy ∨ Qxy
    w : Rxy
    a1 := u1xyw : Ryx
    
```

```

    a2 := v-in1(Ryx, Qyx, a1) : (R ∪ Q)yx
    a3 := λw : Rxy.a2 : (Rxy ⇒ (R ∪ Q)yx)
    w : Qxy
    a4 := a2xyw : Qyx
    a5 := v-in2(Ryx, Qyx, a4) : (R ∪ Q)yx
    a6 := λw : Qxy.a5 : (Qxy ⇒ (R ∪ Q)yx)
    a7 := v-el(Rxy, Qxy, (R ∪ Q)yx, v, a3, a6) : (R ∪ Q)yx
    a8 := λx, y : S.λv : (R ∪ Q)xy.a7 : sym(S, R ∪ Q)
    
```

□

Theorem 5.6. Invariance under composition.

Suppose R and Q are relations on type S . Then the following hold.

- 1) $R \circ R^{-1}$ is always symmetric.
- 2) R and Q are reflexive $\Rightarrow R \circ Q$ is reflexive.
- 3) Suppose R and Q are symmetric. Then $R \circ Q$ is symmetric $\Leftrightarrow R \circ Q = Q \circ R$.

Proof. 1)

```

var S : *s | R : br(S)
var x, y : S | u : (R ∘ R-1)xy
Notation P := λz : S.Rxz ∧ R-1zy : S → *p
u : (∃z : S.Pz)
var z : S | v : Pz
v : Rxz ∧ R-1zy
a1 := ∧-el1(Rxz, R-1zy, v) : Rxz
a2 := ∧-el2(Rxz, R-1zy, v) : R-1zy
a3 := Ryz
a4 := R-1zy
a5 := prod-term(S, R, R-1, y, z, x, a2, a1) : (R ∘ R-1)yx
a6 := λz : S.λv : Pz.a5 : (∀z : S.(Pz ⇒ (R ∘ R-1)yx))
a7 := ∃-el(S, P, u, (R ∘ R-1)yx, a6) : (R ∘ R-1)yx
a8 := λx, y : S.λu : (R ∘ R-1)xy.a7 : sym(S, R ∘ R-1)
    
```

2)

```

var S : *s | R, Q : br(S)
u : refl(S, R) | v : refl(S, Q)
var x : S
ux : Rxx
vx : Qxx
a1 := prod-term(S, R, Q, x, x, ux, vx) : (R ∘ Q)xx
a2 := λx : S.a1 : refl(S, R ∘ Q)
    
```

3) The derivation below uses the proof term *sym-criterion* (S, R) from Theorem 5.1.3) for the second criterion of symmetry and the proof term *conv-prod* from Theorem 4.3.2).

```

var S : *s
var R : br(S)
a1 := sym-criterion(S, R) : sym(S, R) ⇔ (R-1 = R)
a2(R) := ∧-el1(sym(S, R) ⇒ (R-1 = R), (R-1 = R) ⇒ sym(S, R), a1) : sym(S, R) ⇒ (R-1 = R)
a3(R) := ∧-el2(sym(S, R) ⇒ (R-1 = R), (R-1 = R) ⇒ sym(S, R), a1) : (R-1 = R) ⇒ sym(S, R)
var R, Q : br(S) | u : sym(S, R) | v : sym(S, Q)
a4 := a2(R)u : (R-1 = R)
a5 := a2(Q)v : (Q-1 = Q)
a6 := conv-prod(S, R, Q) : (R ∘ Q)-1 = Q-1 ∘ R-1
Notation P1 := λK : br(S).(R ∘ Q)-1 = K ∘ R-1 : br(S) → *p
Notation P2 := λK : br(S).(R ∘ Q)-1 = Q ∘ K : br(S) → *p
a6 := P1(Q-1)
a7 := eq-subs(br(S), P1, Q-1, Q, a5, a6) : (R ∘ Q)-1 = Q ∘ R-1
a7 := P2(R-1)
a8 := eq-subs(br(S), P2, R-1, R, a4, a7) : (R ∘ Q)-1 = Q ∘ R
Notation A := sym(S, R ∘ Q) : *p
Notation B := (R ∘ Q = Q ∘ R) : *p
w : A
a9 := a2(R ∘ Q)w : (R ∘ Q)-1 = R ∘ Q
a10 := eq-sym(br(S), (R ∘ Q)-1, R ∘ Q, a9) : R ∘ Q = (R ∘ Q)-1
a11 := eq-trans(br(S), R ∘ Q, (R ∘ Q)-1, Q ∘ R, a10, a8) : R ∘ Q = Q ∘ R
a12 := λw : A.a11 : A ⇒ B
w : B
w : (R ∘ Q = Q ∘ R)
a13 := eq-sym(br(S), R ∘ Q, Q ∘ R, w) : Q ∘ R = R ∘ Q
a14 := eq-trans(br(S), (R ∘ Q)-1, Q ∘ R, R ∘ Q, a8, a13) : (R ∘ Q)-1 = R ∘ Q
a15 := a3(R ∘ Q)a14 : sym(S, R ∘ Q)
a16 := λw : B.a15 : B ⇒ A
a17 := bi-impl(A, B, a12, a16) : (sym(S, R ∘ Q) ⇔ R ∘ Q = Q ∘ R)
    
```

□

6. Special Binary Relations

6.1. Equivalence Relation and Partition

Theorem 6.1. Invariance of equivalence relation under converse operation and intersection.

Suppose R and Q are equivalence relations on type S . Then the following hold.

- 1) R^{-1} is an equivalence relation on S .
- 2) $R \cap Q$ is an equivalence relation on S .

Proof

- 1) Can easily be derived from Theorem 5.3.1), 2), 4) using intuitionistic logic.
- 2) Can easily be derived from Theorem 5.4. 1), 2), 4) using intuitionistic logic.

The formal proofs are skipped. □

Next is a formalization of the fact that there is a correspondence between equivalence relations on S and partitions of S . Equivalence classes are introduced in Nederpelt and Geuvers (2014) as follows.

```

var S : *s | R : br(S) | u : equiv-rel(S, R)
  var x : S
  class(S, R, u, x) := {y : S | Rxy} : ps(S)
  Notation [x]R for class(S, R, u, x)
    
```

Next, a partition of type S is defined:

```

var S : *s | R : S → ps(S)
  partition(S, R) := (∀x : S. x ∈ Rx)
  ∧ ∀x, y, z : S. (z ∈ Rx ⇒ z ∈ Ry ⇒ Rx = Ry)
    
```

As usual, one can regard a partition R as a collection Rx ($x \in S$) of subsets of S . From this point of view, the above diagram expresses the standard two facts for a partition:

- (1) any element of S belongs to one of the subsets from the collection (namely Rx);
- (2) if intersection of two subsets Rx and Ry is non-empty, then they coincide.

(1) implies that each subset from the collection is non-empty and that the union of all subsets from the collection is S .

Theorem 6.2.

Any equivalence relation R on type S is a partition of S and vice versa.

Proof. The type of partitions of S is $S \rightarrow ps(S)$, which is $S \rightarrow S \rightarrow *_p$, and it is the same as the type $br(S)$ of relations on S . The proof consists of two steps.

Step 1. Any equivalence relation is a partition.

```

var S : *s | R : S → S → *p
  u : equiv-rel(S, R)
  a1 := ∧-el1(refl(S, R), sym(S, R), ∧-el1(refl(S, R) ∧ sym(S, R),
    trans(S, R), u)) : refl(S, R)
  var x : S
  a2 := u1x : Rx
  a2 := (x ∈ Rx)
  a3 := λx : S. a2 : (∀x : S. x ∈ Rx)
    
```

This proves the first part of the definition of *partition* (S, R), and the second part was proven in Nederpelt and Geuvers (2014), pg. 291.

Step 2. Any partition is an equivalence relation.

```

var S : *s | R : S → S → *p
  u : partition(S, R)
  Notation A := ∀x : S. (x ∈ Rx)
  Notation B := ∀x, y, z : S. (z ∈ Rx ⇒ z ∈ Ry ⇒ Rx = Ry)
  u : A ∧ B
  a1 := ∧-el1(A, B, u) : A
  a2 := ∧-el2(A, B, u) : B
  var x : S
  a3 := a1x : x ∈ Rx
  a3 := Rx
  a4 := λx : S. a3 : refl(S, R)
  var x, y : S | v : Rxy
  a5 := a1y : (y ∈ Ry)
  v : (y ∈ Rx)
  a6 := a2xyv a5 : Rx = Ry
  a7 := a1x : (x ∈ Rx)
  a8 := eq-subst(ps(S), λZ : ps(S). x ∈ Z, Rx, Ry, a6, a7) : (x ∈ Ry)
  a8 := Ryx
  a9 := λx, y : S. λv : Rxy. a8 : sym(S, R)
  var x, y, z : S | v : Rxy | w : Ryz
  v : y ∈ Rx
  a10 := a9yzw : Rz
  a10 := (y ∈ Rz)
  a11 := a2zxy a10 v : Rz = Rx
  a12 := a1z : (z ∈ Rz)
  a13 := eq-subst(ps(S), λZ : ps(S). z ∈ Z, Rz, Rx, a11, a12) : z ∈ Rx
  a13 := Rxz
  a14 := λx, y, z : S. λv : Rxy. λw : Ryz. a13 : trans(S, R)
  a15 := ∧-in(refl(S, R) ∧ sym(S, R), trans(S, R), ∧-in(refl(S, R),
    sym(S, R), a4, a9), a14) : equiv-rel(S, R)
    
```

□

6.2. Partial Order

Theorem 6.3. Invariance of partial order under converse operation and intersection.

Suppose R and Q are partial orders on type S . Then the following hold.

- 1) R^{-1} is a partial order on S .
- 2) $R \cap Q$ is a partial order on S .

Proof.

- 1) can easily be derived from Theorem 5.3.1), 3), 4) using intuitionistic logic.
- 2) can easily be derived from Theorem 5.4. 1), 3), 4) using intuitionistic logic.

The formal proofs are skipped. □

Example 6.4

\subseteq is a partial order on the power set $ps(S)$ of type S .

Proof.

This is the formal proof.

```

var S : *s
    Notation R := λX, Y : ps(S). X ⊆ Y : br(ps(S))
    Notation A := refl(ps(S), R)
    Notation B := antisym(ps(S), R)
    Notation C := trans(ps(S), R)
    var X : ps(S)
        a1 := λx : S. λu : (xεX). u : X ⊆ X
    a2 := λX : ps(S). a1 : A
    var X, Y : ps(S) | u : X ⊆ Y | v : Y ⊆ X
        a3 := ∧-in(X ⊆ Y, Y ⊆ X, u, v) : X = Y
    a4 := λX, Y : ps(S). λu : X ⊆ Y. λv : Y ⊆ X. a3 : B
    var X, Y, Z : ps(S) | u : X ⊆ Y | v : Y ⊆ Z
        var x : S | w : xεX
            a5 := uxw : (xεY)
            a6 := vxw : (xεZ)
        a7 := λx : S. λw : (xεX). a6 : X ⊆ Z
    a8 := λX, Y, Z : ps(S). λu : X ⊆ Y. λv : Y ⊆ Z. a7 : C
    a9 := ∧-in(A ∧ B, C, ∧-in(A, B, a2, a4), a8) : A ∧ B ∧ C
    a9 : part-ord(ps(S), R)
    
```

□

6.3. Well-Ordering and Transfinite Induction

Notation \leq will be used for a partial order. The following diagram defines the strict order $<$, the least element of a partially ordered set, and the well-ordering of type S .

```

var S : *s | ≤ : br(S) | u : part-ord(S, ≤)
    Definition < := λx, y : S. (x ≤ y ∧ ¬(x = y))
    var X : ps(S) | x : S
        Definition least(S, ≤, X, x) := xεX ∧ ∀y : S. (yεX ⇒ x ≤ y)
    Definition well-ord(S, ≤) := part-ord(S, ≤)
        ∧ ∀X : ps(S). [∃x : S. xεX ⇒ ∃x : S. least(S, ≤, X, x)]
    
```

Theorem 6.5. Transfinite Induction.

Suppose \leq is a well-ordering of type S . Then for any predicate P on S :

$$\forall x : S. [(\forall y : S. (y < x \Rightarrow Py) \Rightarrow Px] \Rightarrow \forall x : S. Px.$$

Proof

Here is the formal proof.

```

var S : *s | ≤ : br(S) | u1 : well-ord(S, ≤) | P : S → *p
    u2 := ∀x : S. [∀y : S. (y < x ⇒ Py) ⇒ Px]
    Notation A := part-ord(S, ≤)
    Notation B := [∀X : ps(S). (∃x : S. xεX
        ⇒ ∃x : S. least(S, ≤, X, x))]
    u1 : A ∧ B
    a1 := ∧-el1(A, B, u1) : A
    a2 := ∧-el2(A, B, u1) : B
    a3 := ∧-el1(refl(S, ≤) ∧ antisym(S, ≤), trans(S, ≤), a1)
        : refl(S, ≤) ∧ antisym(S, ≤)
    a4 := ∧-el2(refl(S, ≤), antisym(S, ≤), a3) : antisym(S, ≤)
    Notation X := λx : S. ¬Px : ps(S)
    v1 := (∃x : S. xεX)
    a5 := a2Xv1 : [∃x : S. least(S, ≤, X, x)]
    var x : S | v2 : least(S, ≤, X, x)
        a6 := ∧-el1(xεX, ∀y : S. (yεX ⇒ x ≤ y), v2) : xεX
        a6 : ¬Px
        a7 := ∧-el2(xεX, ∀y : S. (yεX ⇒ x ≤ y), v2)
            : [∀y : S. (yεX ⇒ x ≤ y)]
        var y : S | w1 : y < x
            a8 := ∧-el1(y < x, ¬(x = y), w1) : y < x
            a9 := ∧-el2(y < x, ¬(x = y), w1) : ¬(x = y)
            w2 := ¬Py
                w2 : yεX
                a10 := a7yw2 : x ≤ y
                a11 := a4xya10a8 : x = y
                a12 := a9a11 : ⊥
            a13 := λw2 : ¬Py. a12 : ¬¬Py
            a14 := doub-neg(Py)a13 : Py
        a15 := λy : S. λw1 : y < x. a14 : [∀y : S. (y < x ⇒ Py)]
        a16 := u2xa15 : Px
        a17 := a6a16 : ⊥
    a18 := λx : S. λv2 : least(S, ≤, X, x). a17
        : [∀x : S. (least(S, ≤, X, x) ⇒ ⊥)]
    a19 := ∃-el(S, λx : S. least(S, ≤, X, x), a5, ⊥, a18) : ⊥
    a20 := λv1 : (∃x : S. xεX). a19 : ¬(∃x : S. xεX)
    var x : S
        w : ¬Px
        w : xεX
        a21 := ∃-in(S, λz : S. zεX, x, w) : (∃z : S. zεX)
        a22 := a20a21 : ⊥
        a23 := λw : ¬Px. a22 : ¬¬Px
        a24 := doub-neg(Px)a23 : Px
    a25 := λx : S. a24 : (∀x : S. Px)
    
```

Here the Double Negation theorem is used (twice) with the proof term *doub-neg*. This is the only place in this study where classical (not intuitionistic) logic is used.

7. Conclusion

Starting with the definitions from Nederpelt and Geuvers (2014) of binary relations and properties of reflexivity, symmetry, antisymmetry, and transitivity, this study formalizes in the theory λD (the Calculus of Constructions with Definitions) criteria for these properties and proves their invariance under operations of union, intersection, composition, and taking converse. The author provides a formal definition of partition and formally proves correspondence between equivalence relations and partitions. The author derives formal proof that \subseteq is a partial order on the power set. Finally, the author formally proves the principle of transfinite induction for a type with well-ordering.

The results can be transferred to the proof assistants that are based on the Calculus of Constructions. Since binary relation is an abstract concept used in many areas of mathematics, the results can be useful for further formalizations of mathematics in λD . Next direction of research is formalization of parts of probability theory in λD that was outlined in Kachapova (2018).

Acknowledgment

The author thanks the Editor in Chief and Reviewer.

Ethics

This is a mathematical article; no ethical issues can arise after its publication.

References

- Barendregt, H. (2012). The Lambda Calculus, its Syntax and Semantics. vol 40 (Studies in Logic, Mathematical Logic and Foundations). <https://www.collegepublications.co.uk/logic/mlf/?00021>
- Bertot, Y., & Castéran, P. (2013). *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Springer Science & Business Media. <https://link.springer.com/book/10.1007/978-3-662-07964-5>
- Coq Development Team. (2021). *The coq proof assistant, reference manual*, 2021. Coq Development Team. <https://coq.inria.fr/distrib/current/refman/>
- Granstrom, J. G. (2011). *Treatise on intuitionistic type theory* (Vol. 22). Springer Science & Business Media. <https://link.springer.com/book/10.1007/978-94-007-1736-7>
- Kachapova, F. (2018). Formalizing Probability Concepts in a Type Theory. *Journal of Mathematics and Statistics*. <https://thescipub.com/abstract/10.3844/jmssp.2018.209.218>
- Nederpelt, R. P., & Kamareddine, F. D. (2004). Logical reasoning: A first course. ISBN: 9780954300678.

Nederpelt, R., & Geuvers, H. (2014). *Type theory and formal proof*. Cambridge University Press. ISBN: 978-1-107-03650-5.

APPENDIX

Appendix A. Proof of Theorem 4.3

Proof. 1)

```

var S : *κ | R : br(S)
var x, y : S
  u : (R-1)-1xy
  u : R-1yx
  u : Rxy
  a1 := λu : (R-1)-1xy.u : (R-1)-1xy ⇒ Rxy
  u : Rxy
  u : R-1yx
  u : (R-1)-1xy
  a2 := λu : Rxy.u : Rxy ⇒ (R-1)-1xy
  a3 := λx, y : S.a1 : (R-1)-1 ⊆ R
  a4 := λx, y : S.a2 : R ⊆ (R-1)-1
  a5 := rel-equal(R-1)-1, R, a3, a4 : (R-1)-1 = R
    
```

2)

```

var S : *κ | R, Q : br(S)
Notation A := (R ∘ Q)-1 : br(S)
Notation B := Q-1 ∘ R-1 : br(S)
var x, y : S
  Notation P1 := λz : S.Ryz ∧ Qzx : S → *ρ
  Notation P2 := λz : S.Q-1xz ∧ R-1zy : S → *ρ
  u : Axy
  u : (R ∘ Q)yx
  u : (∃z : S.P1z)
  z : S | v : P1z
  v : Ryz ∧ Qzx
  a1 := λ-el1(Ryz, Qzx, v) : Ryz
  a2 := λ-el2(Ryz, Qzx, v) : Qzx
  a1 : R-1zy
  a2 : Q-1xz
  a3 := prod-term(S, Q-1, R-1, x, z, y, a2, a1)
  : (Q-1 ∘ R-1)xy
  a5 : Bxy
  a4 := λz : S.λv : P1z.a3 : (∀z : S.(P1z ⇒ Bxy))
  a5 := ∃-el(S, P1, u, Bxy, a4) : Bxy
  a6 := λx, y : S.λu : Axy.a5 : A ⊆ B
    
```

$\alpha_5 := \exists\text{-el}(S, P_1, u, Bxy, a_4) : Bxy$
 $a_6 := \lambda x, y : S. \lambda u : Axy. \alpha_5 : A \subseteq B$
var $x, y : S \mid u : Bxy$
 $u : (\exists z : S. P_2 z)$
 $z : S \mid v : P_2 z$
 $v : Q^{-1}xz \wedge R^{-1}zy$
 $a_7 := \wedge\text{-el}_1(Q^{-1}xz, R^{-1}zy, v) : Q^{-1}xz$
 $a_8 := \wedge\text{-el}_2(Q^{-1}xz, R^{-1}zy, v) : R^{-1}zy$
 $a_7 : Qzx$
 $a_8 : Ryz$
 $a_9 := \text{prod-term}(S, R, Q, y, z, x, a_8, a_7) : (R \circ Q)yx$
 $a_9 : (R \circ Q)^{-1}xy$
 $a_9 : Axy$
 $a_{10} := \lambda z : S. \lambda v : P_2 z. a_9 : (\forall z : S. (P_2 z \Rightarrow Axy))$
 $a_{11} := \exists\text{-el}(S, P_2, u, Axy, a_{10}) : Axy$
 $a_{12} := \lambda x, y : S. \lambda u : Bxy. a_{11} : B \subseteq A$
conv-prod(S, R, Q) := *rel-equal*(A, B, a_6, a_{12})
 $: (R \circ Q)^{-1} = Q^{-1} \circ R^{-1}$

3)

var $S : *_s \mid R, Q : br(S)$
 Notation $A := (R \cap Q)^{-1} : br(S)$
 Notation $B := R^{-1} \cap Q^{-1} : br(S)$
var $x, y : S \mid u : Axy$
 $u : (R \cap Q)^{-1}xy$
 $u : (R \cap Q)yx$
 $u : Ryx \wedge Qyx$
 $a_1 := \wedge\text{-el}_1(Ryx, Qyx, v) : Ryx$
 $a_2 := \wedge\text{-el}_2(Ryx, Qyx, v) : Qyx$
 $a_1 : R^{-1}xy$
 $a_2 : Q^{-1}xy$
 $a_3 := \wedge\text{-in}(R^{-1}xy, Q^{-1}xy, a_1, a_2) : Bxy$
 $a_4 := \lambda x, y : S. \lambda u : Axy. a_3 : A \subseteq B$
var $x, y : S \mid u : Bxy$
 $u : R^{-1}xy \wedge Q^{-1}xy$
 $a_5 := \wedge\text{-el}_1(R^{-1}xy, Q^{-1}xy, v) : R^{-1}xy$
 $a_6 := \wedge\text{-el}_2(R^{-1}xy, Q^{-1}xy, v) : Q^{-1}xy$
 $a_5 : Ryx$
 $a_6 : Qyx$
 $a_7 := \wedge\text{-in}(Ryx, Qyx, a_5, a_6) : (R \cap Q)yx$
 $a_7 : (R \cap Q)^{-1}xy$
 $a_7 : Axy$
 $a_8 := \lambda x, y : S. \lambda u : Bxy. a_7 : B \subseteq A$
 $a_9 := \text{rel-equal}(A, B, a_4, a_8) : (R \cap Q)^{-1} = R^{-1} \cap Q^{-1}$

4)

var $S : *_s \mid R, Q : br(S)$
 Notation $A := (R \cup Q)^{-1} : br(S)$
 Notation $B := R^{-1} \cup Q^{-1} : br(S)$
var $x, y : S \mid u : Axy$
 $u : (R \cup Q)yx$
 $u : Ryx \vee Qyx$
 $v : Ryx$
 $v : R^{-1}xy$
 $a_1 := \vee\text{-in}_1(R^{-1}xy, Q^{-1}xy, v) : Bxy$
 $a_2 := \lambda v : Ryx. a_1 : Ryx \Rightarrow Bxy$
 $v : Qyx$
 $v : Q^{-1}xy$
 $a_3 := \vee\text{-in}_2(R^{-1}xy, Q^{-1}xy, v) : Bxy$
 $a_4 := \lambda v : Qyx. a_3 : Qyx \Rightarrow Bxy$
 $a_5 := \vee\text{-el}(Ryx, Qyx, Bxy, u, a_2, a_4) : Bxy$
 $a_6 := \lambda x, y : S. \lambda u : Axy. a_5 : A \subseteq B$
var $x, y : S \mid u : Bxy$
 $u : R^{-1}xy \vee Q^{-1}xy$
 $v : R^{-1}xy$
 $v : Ryx$
 $a_7 := \vee\text{-in}_1(Ryx, Qyx, v) : Ryx \vee Qyx$
 $a_7 : (R \cup Q)^{-1}xy$
 $a_7 : Axy$
 $a_8 := \lambda v : R^{-1}xy. a_7 : R^{-1}xy \Rightarrow Axy$
 $v : Q^{-1}xy$
 $v : Qyx$
 $a_9 := \vee\text{-in}_2(Ryx, Qyx, v) : Ryx \vee Qyx$
 $a_9 : (R \cup Q)^{-1}xy$
 $a_9 : Axy$
 $a_{10} := \lambda v : Q^{-1}xy. a_9 : Q^{-1}xy \Rightarrow Axy$
 $a_{11} := \vee\text{-el}(R^{-1}xy, Q^{-1}xy, Axy, u, a_8, a_{10}) : Axy$
 $a_{12} := \lambda x, y : S. \lambda u : Bxy. a_{11} : B \subseteq A$
 $a_{13} := \text{rel-equal}(A, B, a_6, a_{12}) : (R \cup Q)^{-1} = R^{-1} \cup Q^{-1}$

5)

var $S : *_s \mid R, P, Q : br(S)$
 Notation $A := R \circ (P \cup Q) : br(S)$
 Notation $B := R \circ P \cup R \circ Q : br(S)$
var $x, y : S$
 Notation $P_0 := \lambda z : S. Rxz \wedge (P \cup Q)zy : S \rightarrow *_p$
 $u : Axy$
 $u : (\exists z : S. P_0 z)$
 $z : S \mid v : P_0 z$
 $v : Rxz \wedge (P \cup Q)zy$


```

a1 :=  $\wedge$ -el1(Rxz, (P  $\cup$  Q)zy, v) : Rxz
a2 :=  $\wedge$ -el2(Rxz, (P  $\cup$  Q)zy, v) : (P  $\cup$  Q)zy
a2 := Pzy  $\vee$  Qzy
w : Pzy
a3 := prod-term (S, R, P, x, z, y, a1, w) : (R  $\circ$  P)xy
a4 :=  $\vee$ -in1((R  $\circ$  P)xy, (R  $\circ$  Q)xy, a3) : Bxy
a5 :=  $\lambda$ w : Pzy.a4 : Pzy  $\Rightarrow$  Bxy
w : Qzy
a6 := prod-term (S, R, Q, x, z, y, a1, w) : (R  $\circ$  Q)xy
a7 :=  $\vee$ -in2((R  $\circ$  P)xy, (R  $\circ$  Q)xy, a6) : Bxy
a8 :=  $\lambda$ w : Qzy.a7 : Qzy  $\Rightarrow$  Bxy
a9 :=  $\vee$ -el (Pzy, Qzy, Bxy, a2, a5, a8) : Bxy
a10 :=  $\lambda$ z : S. $\lambda$ v : P0z.a9 : ( $\forall$ z : S.(P0z  $\Rightarrow$  Bxy))
a11 :=  $\exists$ -el (S, P0, u, Bxy, a10) : Bxy
a12 :=  $\lambda$ x, y : S. $\lambda$ u : Axy.a11 : A  $\subseteq$  B
var x, y : S
Notation P1 :=  $\lambda$ z : S.Rxz  $\wedge$  Pzy : S  $\rightarrow$  *p
Notation P2 :=  $\lambda$ z : S.Rxz  $\wedge$  Qzy : S  $\rightarrow$  *p
u : Bxy
u : (R  $\circ$  P)xy  $\vee$  (R  $\circ$  Q)xy
v : (R  $\circ$  P)xy
v : ( $\exists$ z : S.P1z)
z : S | w : P1z
w : Rxz  $\wedge$  Pzy
a13 :=  $\wedge$ -el1(Rxz, Pzy, w) : Rxz
a14 :=  $\wedge$ -el2(Rxz, Pzy, w) : Pzy
a15 :=  $\vee$ -in1(Pzy, Qzy, a14) : (P  $\cup$  Q)zy
a16 := prod-term (S, R, (P  $\cup$  Q), x, z, y, a13, a15) : Axy
a17 :=  $\lambda$ z : S. $\lambda$ w : P1z.a16 : ( $\forall$ z : S.(P1z  $\Rightarrow$  Axy))
a18 :=  $\exists$ -el (S, P1, v, Axy, a17) : Axy
a19 :=  $\lambda$ v : (R  $\circ$  P)xy.a18 : ((R  $\circ$  P)xy  $\Rightarrow$  Axy)
v : (R  $\circ$  Q)xy
v : ( $\exists$ z : S.P2z)
z : S | w : P2z
a20 :=  $\wedge$ -el1(Rxz, Qzy, w) : Rxz
a21 :=  $\wedge$ -el2(Rxz, Qzy, w) : Qzy
a22 :=  $\vee$ -in2(Pzy, Qzy, a21) : (P  $\cup$  Q)zy
a23 := prod-term (S, R, (P  $\cup$  Q), x, z, y, a20, a22) : Axy
a24 :=  $\lambda$ z : S. $\lambda$ w : P2z.a23 : ( $\forall$ z : S.(P2z  $\Rightarrow$  Axy))
a25 :=  $\exists$ -el (S, P2, v, Axy, a24) : Axy
a26 :=  $\lambda$ v : (R  $\circ$  Q)xy.a25 : ((R  $\circ$  Q)xy  $\Rightarrow$  Axy)
a27 :=  $\vee$ -el((R  $\circ$  P)xy, (R  $\circ$  Q)xy, Axy, u, a19, a26) : Axy
a28 :=  $\lambda$ x, y : S. $\lambda$ u : Bxy.a27 : B  $\subseteq$  A
a29 := rel-equal(A, B, u, a28) : R  $\circ$  (P  $\cup$  Q) = R  $\circ$  P  $\cup$  R  $\circ$  Q
    
```

6) is proven similarly to 5).

7)

```

var S : *s | R, P, Q : br(S)
Notation A := R  $\circ$  (P  $\cap$  Q) : br(S)
Notation B := R  $\circ$  P  $\cap$  R  $\circ$  Q : br(S)
var x, y : S
Notation P :=  $\lambda$ z : S.Rxz  $\wedge$  (P  $\cap$  Q)zy : *p
u : Axy
u : ( $\exists$ z : S.Pz)
var z : S | v : Pz
v : Rxz  $\wedge$  (P  $\cap$  Q)zy
a1 :=  $\wedge$ -el1(Rxz, (P  $\cap$  Q)zy, v) : Rxz
a2 :=  $\wedge$ -el2(Rxz, (P  $\cap$  Q)zy, v) : (P  $\cap$  Q)zy
a2 := Pzy  $\wedge$  Qzy
a3 :=  $\wedge$ -el1(Pzy, Qzy, a2) : Pzy
a4 :=  $\wedge$ -el2(Pzy, Qzy, a2) : Qzy
a5 := prod-term (S, R, P, x, z, y, a1, a3) : (R  $\circ$  P)xy
a6 := prod-term (S, R, Q, x, z, y, a1, a4) : (R  $\circ$  Q)xy
a7 :=  $\wedge$ -in ((R  $\circ$  P)xy, (R  $\circ$  Q)xy, a5, a6) : Bxy
a8 :=  $\lambda$ z : S. $\lambda$ v : Pz.a7 : ( $\forall$ z : S.(Pz  $\Rightarrow$  Bxy))
a9 :=  $\exists$ -el (S, P, u, Bxy, a8) : Bxy
a10 :=  $\lambda$ x, y : S. $\lambda$ u : Axy.a9 : R  $\circ$  (P  $\cap$  Q)  $\subseteq$  R  $\circ$  P  $\cap$  R  $\circ$  Q
    
```

8) is proven similarly to 7).

9)

```

var S : *s | R, P, Q : br(S)
Notation A := (R  $\circ$  P)  $\circ$  Q : br(S)
Notation B := R  $\circ$  (P  $\circ$  Q) : br(S)
var x, y : S
Notation P1(x, y) :=  $\lambda$ z : S.(R  $\circ$  P)xz  $\wedge$  Qzy : S  $\rightarrow$  *p
Notation P2(x, y) :=  $\lambda$ z : S.Rxz  $\wedge$  (P  $\circ$  Q)zy : S  $\rightarrow$  *p
Notation P3(x, y) :=  $\lambda$ z : S.Rxz  $\wedge$  Pzy : S  $\rightarrow$  *p
Notation P4(x, y) :=  $\lambda$ z : S.Pxz  $\wedge$  Qzy : S  $\rightarrow$  *p
var x, y : S | u : Axy
u : ( $\exists$ z : S.P1(x, y)z)
var z : S | v : P1(x, y)z
a1 :=  $\wedge$ -el1((R  $\circ$  P)xz, Qzy, v) : (R  $\circ$  P)xz
a2 :=  $\wedge$ -el2((R  $\circ$  P)xz, Qzy, v) : Qzy
a1 : ( $\exists$ z1 : S.P3(x, z)z1)
var z1 : S | w : P3(x, z)z1
w : Rxz1  $\wedge$  Pz1z
a3 :=  $\wedge$ -el1(Rxz1, Pz1z, w) : Rxz1
a4 :=  $\wedge$ -el2(Rxz1, Pz1z, w) : Pz1z
    
```

```

a5 := prod-term (S, P, Q, z1, z, y, a4, a2) : (P o Q)z1y
a6 := prod-term (S, R, (P o Q), x, z1, y, a3, a5) : Bxy
a7 := λz1 : S.λw : P3(x, z)z1.a6
      (∀z1 : S.(P3(x, z)z1 ⇒ Bxy))
a8 := ∃-el (S, P3(x, z), a1, Bxy, a7) : Bxy
a9 := λz : S.λv : P1(x, y)z.a8 : (∀z : S.(P1(x, y)z ⇒ Bxy))
a10 := ∃-el (S, P1(x, y), u, Bxy, a9) : Bxy
a11 := λx, y : S.λu : Axy.a10 : A ⊆ B
var x, y : S | u : Bxy
u : (∃z : S.P2(x, y)z)
var z : S | v : P2(x, y)z
a12 := ∧-el1 (Rxz, (P o Q)zy, v) : Rxz
a13 := ∧-el2 (Rxz, (P o Q)zy, v) : (P o Q)zy
a13 : (∃z1 : S.P4(z, y)z1)
var z1 : S | w : P4(z, y)z1
w : Pz1 ∧ Qz1y
a14 := ∧-el1 (Pz1, Qz1y, w) : Pz1
a15 := ∧-el2 (Pz1, Qz1y, w) : Qz1y
a16 := prod-term (S, R, P, x, z, z1, a12, a14)
      : (R o P)xz1
a17 := prod-term (S, R o P, Q, x, z1, y, a16, a15) : Axy
a18 := λz1 : S.λw : P4(z, y)z1.a17
      : (∀z1 : S.(P4(z, y)z1 ⇒ Axy))
a19 := ∃-el (S, P4(z, y), a13, Axy, a18) : Axy
a20 := λz : S.λv : P2(x, y)z.a19 : (∀z : S.(P2(x, y)z ⇒ Axy))
a21 := ∃-el (S, P2(x, y), u, Axy, a20) : Axy
a22 := λx, y : S.λu : Bxy.a21 : B ⊆ A
a23 := rel-equal(A, B, a11, a22) : (R o P) o Q = R o (P o Q)
    
```

□

Appendix B. Proof of Theorem 5.1

Proof. Each statement here is a bi-implication, so the proof term *bi-impl* from Lemma 2.3 is used.

1)

```

var S : *s | R : br(S)
Notation A := refl(S, R) : *p
Notation B := idS ⊆ R : *p
u : A
var x, y : S | v : (idS)xy
v : x =S y
Notation P := λz : S.Rxz : S → *p
ux : Px
a1 := eq-subst(S, P, x, y, v, ux) : Py
a1 : Rxy
    
```

```

a2 := λx, y : S.λv : (idS)xy.a1 : (idS ⊆ R)
a2 : B
a3 := λu : A.a2 : (A ⇒ B)
u : B
var x : S
a4 := eq-refl(S, x) : x =S x
a4 : (idS)xx
uxx : (idS)xx ⇒ Rxx
a5 := uxxa4 : Rxx
a6 := λx : S.a5 : (∀x : S.Rxx)
a6 : A
a7 := λu : B.a6 : (B ⇒ A)
a8 := bi-impl(A, B, a3, a7) : refl(S, R) ⇒ idS ⊆ R
    
```

2) and 3) are proven together as follows.

```

var S : *s | R : br(S)
Notation A := sym(S, R) : *p
Notation B := R-1 ⊆ R : *p
Notation C := R-1 = R : *p
u : A
var x, y : S | v : R-1xy
v : Ryx
uyx : (Ryx ⇒ Rxy)
a1 := uyxv : Rxy
a2 := λx, y : S.λu : R-1xy.a1 : (R-1 ⊆ R)
var x, y : S | v : Rxy
uxy : (Rxy ⇒ Ryx)
a3 := uxyv : Ryx
a3 : R-1xy
a4 := λx, y : S.λu : Rxy.a3 : (R ⊆ R-1)
a5 := rel-equal(S, R-1, R, a2, a4) : R-1 = R
a6 := λu : A.a2 : A ⇒ B
a7 := λu : A.a5 : A ⇒ C
u : B
var x, y : S | v : Rxy
v : R-1yx
uyx : (R-1yx ⇒ Ryx)
a8 := uyxv : Ryx
a9 := λx, y : S.λv : Rxy.a8 : sym(S, R)
a10 := λu : B.a8 : (B ⇒ A)
u : C
u : R-1 ⊆ R ∧ R ⊆ R-1
a11 := ∧-el1(R-1 ⊆ R, R ⊆ R-1, u) : R-1 ⊆ R
    
```

$a_{11} : B$
 $a_{12} := a_{10}a_{11} : A$
 $a_{13} := \lambda u : C.a_{12} : (C \Rightarrow A)$
 $a_{14} := bi-impl(A, B, a_6, a_{10}) : sym(S, R) \Leftrightarrow R^{-1} \subseteq R$
 $sym-criterion(S, R) := bi-impl(A, C, a_7, a_{13})$
 $sym(S, R) \Leftrightarrow R^{-1} = R$

4)

var $S : *_S | R : br(S)$
 Notation $A := antisym(S, R) : *_P$
 Notation $B := R \cap R^{-1} \subseteq id_S : *_P$
u : A
var $x, y : S | v : (R \cap R^{-1})xy$
 $v : R^{-1}xy \wedge Rxy$
 $a_1 := \wedge-el_1(R^{-1}xy, Rxy, v) : R^{-1}xy$
 $a_2 := \wedge-el_2(R^{-1}xy, Rxy, v) : Rxy$
 $a_1 : Ryx$
 $uxy : Rxy \Rightarrow Ryx \Rightarrow x = y$
 $a_3 := uxya_2a_1 : (x = y)$
 $a_3 : (id_S)xy$
 $a_4 := \lambda x, y : S.\lambda v : (R \cap R^{-1})xy.a_3 : (R \cap R^{-1} \subseteq id_S)$
 $a_4 : B$
 $a_5 := \lambda u : A.a_4 : (A \Rightarrow B)$
u : B
var $x, y : S | v : Rxy | w : Ryx$
 $w : R^{-1}xy$
 $a_6 := \wedge-in_1(R^{-1}xy, Rxy, w, v) : (R^{-1} \cap R)xy$
 $a_7 := uxya_6 : (id_S)xy$
 $a_7 : x = y$
 $a_8 := \lambda x, y : S.\lambda v : Rxy.\lambda w : Ryx.a_7 : antisym(S, R)$
 $a_8 : A$
 $a_9 := \lambda u : B.a_8 : (B \Rightarrow A)$
 $a_{10} := bi-impl(A, B, a_5, a_9)$
 $(antisym(S, R) \Leftrightarrow (R \cap R^{-1} \subseteq id_S))$

5)

var $S : *_S | R : br(S)$
 Notation $A := trans(S, R) : *_P$
 Notation $B := R \circ R \subseteq R : *_P$
u : A
var $x, y : S$
 Notation $P := \lambda z : S.Rxz \wedge Rzy : S \rightarrow *_P$
v : $(R \circ R)xy$
 $v : (\exists z : S.Pz)$
var $z : S | w : Pz$
 $w : Rxz \wedge Rzy$
 $a_1 := \wedge-el_1(Rxz, Rzy, w) : Rxz$
 $a_2 := \wedge-el_2(Rxz, Rzy, w) : Rzy$
 $a_3 := uxyz a_1 a_2 : Rxy$
 $a_4 := \lambda z : S.\lambda w : Pz.a_3 : (\forall z : S.(Pz \Rightarrow Rxy))$
 $a_5 := \exists-el(S, P, v, Rxy, a_4) : Rxy$
 $a_6 := \lambda x, y : S.\lambda v : (R \circ R)xy.a_5 : (R \circ R \subseteq R)$
 $a_6 : B$
 $a_7 := \lambda u : A.a_6 : (A \Rightarrow B)$
u : B
var $x, y, z : S | v : Rxy | w : Ryz$
 $a_8 := prod-term(S, R, R, x, y, z, v, w) : (R \circ R)xz$
 $a_9 := uxz : ((R \circ R)xz \Rightarrow Rxz)$
 $a_{10} := a_9 a_8 : Rxz$
 $a_{11} := \lambda x, y, z : S.\lambda v : Rxy.\lambda w : Ryz.a_{10} : trans(S, R)$
 $a_{11} : A$
 $a_{12} := \lambda u : B.a_{11} : (B \Rightarrow A)$
 $a_{13} := bi-impl(A, B, a_7, a_{12}) : (trans(S, R) \Leftrightarrow (R \circ R \subseteq R))$