ИМПЛЕМЕНТАЦИЯ ОСНОВАНИЙ МАТЕМАТИКИ В СИСТЕМЕ COQ

Дунаев Георгий

2019

Содержание

1 Введение

Разнообразные теории типов и теории множеств – это различные подходы к строгой записи математических теорем. В обоих случаях, доказательство некторого факта есть сведение сложных утверждений с помощью правил и аксиом к более более простым, принятым без доказательств.

The Coq Proof Assistant – это современный обширный инструмент формальной верификации доказательств("пруфассистант"). Он позволяет проверять корректность математических доказательств. Используемый язык доказательств называется Gallina. Базируется Coq на системе типов CIC "Calculus of Inductive Constructions"(правила вывода здесь: `https://coq.inria.fr/refman/language/cic.html#the-terms`).

Поскольку программный код пруфассистанта велик и не представляется возможным проверить его вручную, возникает вопрос о доверии полученным результатам. Для обеспечения возможности проверки полученных результатов используется "принцип Де Брюйна": минималистичная версия программы-тайпчекера отделено от остального массива кода, обеспечивающего взаимодействиет с пользователем.

Исчисление индуктивных конструкций в свою очередь является расширением системы типов CoC("Calculus of Constructions"/"Исчисление конструкций"). Формализм исчисления конструкций – это разработка Тьерри Кокана и Жерара Юэ. Это не единственная система, базирующаяся на CIC, следует также отметить инструмент формальной верификации Agda, написанной на Haskell.

Математические утверждения, кодируются некоторым типом. Доказательствами утверждений - термами да данного типа. Выбор типов или множеств для формализации утверждений обусловлен практическими соображениями. Недостатками множеств является малая модульность, достоинством – возможность однородно, без стратификации, работать как с множествами, так и с множествами множеств и так далее (пример: нормальная подгруппа одновременно является и подмножеством группы и элементом факторгруппы). В теории типов CIC – ситуация обратная: модульность нативно обеспечивается типами и модулями, но чтобы работать с совокупностями, необходимо использовать некоторую библиотеку.

В обоих случаях программный комплексы были написанны на языке Галлина.

Под имплементацией оснований математики в данной работе понимаются две вещи. Первая – это построение логического каркаса для первопорядковых теорий. Вторая – перевод доказательств из языка теории множеств ZFC в язык теории типов.

Исходные коды в последней редакции можно найти в репозиториях автора данной работы:

- Пропозициональная логика:
  `https://github.com/georgydunaev/VerifiedMathFoundations/blob/development/PropLang.v`,
- Первопорядковая логика:
  `https://github.com/georgydunaev/VerifiedMathFoundations/blob/development/OneFile.v`,
- Теория множеств:
  `https://github.com/georgydunaev/Jech` .

2 Постановка задачи

Цель данной работы – разработка и исследование инструментов для перевода математических утверждений в программы. Философское обоснование – развитие идей логицизма, которые сводят математику к логике, за счёт сведения логических выкладок на бумаге к программам на Coq. Актуальность данной работы обусловлена необходимостью в формализации математики, вызванна широким спектром причин: как наличием критических к отказу электроники задач(банковское дело, врачебное дело, авиация, атомная промышленность), так и тем, что в современной математике накоплен значительный объём доказательств, статус которых не принимается однозначно в математическом сообществе (теорема о 4-х красках, abc-гипотеза).

Обычно, в курсах математической логики для доказательства различных фактов про теории первого порядка используются теоретико-множественные основания математики.

В первой части работы доказана корректность классического и интуиционисткого исчислений высказываний относительно различных семантик и полнота классического исчисления высказываний относительно булевой семантики.

Во второй части работы исследуются методы реализации тех же фактов про первопорядковые теории с помощью теоретико-типовых оснований математики. Во второй части работы записаны теоретико-множественные определения и их интерпретация в системе Coq.

Поскольку индуктивные определения – одни из базовых конструкций Coq, интерпретация будет именно содержательная, интуитивная, а не строгое вложение теоретико-множественных результатов в построенную модель теории множеств, так как тогда пришлось бы разрабатывать внутри модели теорию ординалов и теоремы о рекурсии.

Модели одного в другом можно найти в работе Вернера “Sets in Types, Types in Sets” (`https://link.springer.com/chapter/10.1007/BFb0014566`).

Один из вариантов вложения логики высказываний в Coq можно найти в работе Флориса Ван Доорна.(`https://arxiv.org/abs/1503.08744`)

Во третей части, продолжена работа Вернера (https://github.com/coq-contribs/zfc), где построены конструкции теории множеств, упрощены доказательства, реализовано доказательство аксиомы выбора в модели за счёт принятия невычислимой теоретико-типовой аксиомы выбора. При этом, конструкции теории множеств соответствуют определениям из книги Томаша Йеха “Теория множеств”(2003), а не представленным в работе Вернера. При этом, для облегчения перевода теорем, были доказаны аксиомы теории множеств, для потенциального использования их в дальнейшем.

Последующее соединение этих всех частей в дальнейшем позволит доказывать теоремы, верифицируя ровно те же выкладки, котороые делает математик работающий в первопорядковой теории множеств, что позволит уменьшить число ошибок в написанных таким образом математических текстах.

3 Аксиомы, расширяющие Coq

Выводимость в языке Галлина может быть расширена дополнительными пользовательсткими аксиомами. Неудачный выбор аксиом может привести к доказательству лжи, что по принципу ex falso quodlibet докажет все утверждения. Существует несколько аксиом, которые можно безопасно добавить использовать. Они присутствуют в стандартной библиотеке The Coq Proof Assistant.

1. Аксиома конструктивно-неопределённого описания(constructive indefinite description). Эта существенно неконструктивная аксиома позволяет считать построенным объект с заданным свойством, если доказано только его существование и истинность свойства, без явного построения этого объекта.

```
Axiom constructive_indefinite_description : forall (A : Type) (P : A -> Prop), (exists x : A, P x) -> {x
    ↪ : A | P x}.

Definition ex2sig {A : Type} {P : A -> Prop}
 : (exists x : A, P x) -> {x : A | P x}
 := constructive_indefinite_description P.
```

2. Закон исключённого третьего

```
Axiom classic : forall P:Prop, P \/ ~P.
```

3. Предыдущая аксиома является теоремой, если принять теоретико-типовую аксиому выбора. Доказательство этого факта может быть найдено в стандартной библиотеке: /theories/Logic/Diaconescu.v.

```
Definition FunctionalChoice_on :=
  forall R:A->B->Prop,
    (forall x : A, exists y : B, R x y) ->
    (exists f : A->B, forall x : A, R x (f x)).

Notation FunctionalChoice :=
  (forall A B : Type, FunctionalChoice_on A B).

Axiom axFC : FunctionalChoice.
```

4. Существует также другой вариант этой аксиомы. Она содержит такое свойство: если отношение функционально и сохраняет некоторое отношение эквивалентности T, тогда функция выбора корректно опредлена на классах эквивалентности.

```
(** AC_fun_setoid = functional form of the (so-called extensional) axiom of choice from setoids **)
Definition SetoidFunctionalChoice_on :=
  forall R : A -> A -> Prop,
  forall T : A -> B -> Prop,
  Equivalence R ->
  (forall x x' y, R x x' -> T x y -> T x' y) ->
  (forall x, exists y, T x y) ->
  exists f : A -> B, forall x : A, T x (f x) /\ (forall x' : A, R x x' -> f x = f x').
Notation SetoidFunctionalChoice :=
```

```
  (forall A B: Type, SetoidFunctionalChoice_on A B).

Axiom (axSFC:SetoidFunctionalChoice).
```

Как будет показано далее, принятия этой аксиомы достаточно для доказательства аксиомы выбора для множеств в модели ZFC.

4 Модель логики высказываний в Coq

Здесь и далее по тексту, надпись “определим”, или “докажем”, – есть сокращение от “формально определим терм некоторого тип в система Coq” или, соответственно, “формально выведем терм соответствующего типа в системе Coq”, подразумевающее присвоение новому идетификатору терма из языка Gallina. Если это возможно сделать, то тип называется *населённым*.

В данном разделе будут верифицированы теоремы о логике высказываний. Сначала определим пропозициональные формулы, их выводимость в логике высказываний. Будет имплементирована выводимость пропозициональной формулы из контекста, доказаны некоторые теормы про неё: определим различные семантики логики высказываний и докажем их корректность. При написании кода для этой главы были использованые определения и доказательства из главы 2 [1] и главы 3 из [12].

4.1 Начало программной реализации

В начале подгрузим все необходимые библиотеки.

```
Require Import Bool.
Require Import Coq.Lists.List.
Require Import Coq.Structures.Equalities.

Require Import Logic.Classical_Prop. (* For classical semantic *)
Require Import PeanoNat. (* for indices for contexts in completeness *)
Require Import Relations. (* for transitivity relation in Kripke semantics*)
```

Пусть задан тип PropVars.t пропозициональных переменных с разрешимым равенством. Атомарные формулы в исчислении высказываний – это пропозициональные переменные. Обозначим как A произвольную прозициональную переменную, тогда формулы порождаются следующей грамматикой.

$$F ::= A|\bot|(F \wedge F)|(F \vee F)|(F \to F).$$

Язык высказываний определим в Coq как индуктивный тип с конструкторами лжи, импликаций, конъюнкций и дизъюнкций. Нотация формул аналогичная Coq-овской для элементов типа Prop. Символ “:>” используется для возможности автоматической трактовки термов из типа атомарных суждений как термов из типа формул. Таким образом, если A:PropVars.t, то A:Fo – нотация для (Atom A):Fo.

```
(** Language of the propositional logic. **)
Inductive Fo :=
|Atom (p:PropVars.t) :> Fo
|Bot :Fo
|Conj:Fo->Fo->Fo
|Disj:Fo->Fo->Fo
|Impl:Fo->Fo->Fo
.

Notation " x --> y ":=(Impl x y) (at level 80, right associativity).
Notation " x -/\ y ":=(Conj x y) (at level 80).
Notation " x -\/ y ":=(Disj x y) (at level 80).
Notation " -. x " :=(Impl x Bot) (at level 80).
Definition Neg (A:Fo):Fo := Impl A Bot.
Definition Top:Fo := Neg Bot.
```

4.2 Операции на контекстах

Обозначение 4.1. Будем называть Type-предикатами на типе A(или предикатом) термы типа A->Type.

Определение 4.1. Предикат P *истинный* на формуле A, когда тип P(A) населён.

Определение 4.2. Контекст – это множество пропозициональных формул. Задаем в Coq контексты как Type-предикатами на типе Fo. Истинность последнего соответсвует принадлежности формулы контексту.

Порядок формул в контексте и количество их повторений не играют никакой роли в выводимости для пропозициональной логики, поэтому вместо моделирования контекста как списка или вектора(которые должны быть конечны), моделируется сам предикат $E(\phi, \Gamma)$:=“формула ϕ есть в контексте Γ”. Введём операции на таких предикатах, продолжив их с их аналогов для контекстов.

Определение 4.3. Здесь "empctx" – предикат для пустого контекста, т.е. функция, которая любую формулу преобразует в пустой тип.

```
Inductive empctx : Fo -> Type :=. (* empty context *)
```

Определение 4.4. "add2ctx A l" – предикат для расширения контекста l формулой A.

```
Definition add2ctx (A:Fo) (l:Fo->Type) : Fo->Type := fun f=> sum (A=f) (l f). (* add head *)
```

Определение 4.5. "cnctctx" – предикат для конкатенированных контекстов.

```
Definition cnctctx (l1 l2:Fo->Type) : Fo->Type := fun f=> sum (l1 f) (l2 f). (* concat *)
```

Определение 4.6. "addempeqv" – теорема о том, что конкатенирование с пустым конеткстом не изменяет предиката этого контекста.

```
Lemma addempeqv (il:Fo->Type) : forall (f:Fo), (cnctctx il empctx) f -> il f.
Proof. intros f q. destruct q. exact i. destruct e. Defined.
```

В тех случаях, когда для задания предиката $E(\varphi, \Gamma)$ удобно использовать список, введём функцию lf2ft, переводящую списки формул в Type-предикаты на формулах. (Ниже sum – это дизъюнктивное объединение типов.)

Определение 4.7. Тип (InL a l) населён, если в списке l есть элемент a.

```
Definition InL { A : Type } :=
fix InL (a : A) (l : list A) {struct l} : Type :=
 match l with
 | Datatypes.nil => False
 | b :: m => (sum (b = a) (InL a m))
 end.

Definition listFo := list Fo.

Theorem lf2ft : listFo -> (Fo -> Type).
Proof. intros lf f. exact (InL f lf). Defined.

Coercion lf2ft : listFo >-> Funclass.

Check (nil:listFo) : Fo -> Type.
```

4.3 Аксиоматизация логики высказываний

Классическая логика высказываний аксиоматизируется исчислением высказываний, которое задаётся аксиомами из таблицы 1 и правилом Modus Ponens. Эта же таблица без последней аксиомы задаёт интуиционисткое исчисление высказываний, аксиоматизирующее интуиционисткую логику высказываний.

Таблица 1: Схемы аксиом классической логики высказываний

название	утверждение
impI	$A \to (B \to A)$
impE	$(A \to (B \to C)) \to ((A \to B) \to (A \to C))$
andE1	$A \wedge B \to A$
andE2	$A \wedge B \to B$
andI	$A \to (B \to (A \wedge B))$
orI1	$A \to A \vee B$
orI2	$B \to A \vee B$
orE	$(A \to C) \to ((B \to C) \to (A \vee B \to C))$
negI	$(A \to B) \to ((A \to \neg B) \to \neg A)$
negE	$A \to (\neg A \to B)$
LEM	$A \vee \neg A$

Определение 4.8. Присвоим предикату выделяющему среди корректных формул аксиомы интуиционисткой логики идентификатор PROCAI, а предикату выделяющему среди корректных формул аксиомы классической логике – идентификатор PROCA.
Индуктивно зададим их в языке Галлина:

```
(* Proposional calculus' axioms (Intuitionistic) *)
Inductive PROCAI : Fo -> Type :=
| Ha1 : forall A B, PROCAI (A-->(B-->A))
| Ha2 : forall A B C, PROCAI ((A-->(B-->C))-->((A-->B)-->(A-->C)))
| Ha3 : forall A B, PROCAI ((A-/\ B)--> A)
| Ha4 : forall A B, PROCAI ((A-/\ B)--> B)
| Ha5 : forall A B, PROCAI (A-->(B-->(A-/\B)))
| Ha6 : forall A B, PROCAI (A-->(A-\/ B))
| Ha7 : forall A B, PROCAI (B-->(A-\/ B))
| Ha8 : forall A B C, PROCAI ((A-->C)-->((B-->C)-->((A-\/ B)-->C)))
| Ha9 : forall A B, PROCAI (-.A --> (A --> B) )
| Ha10 : forall A B, PROCAI ((A-->B)-->(A -->-.B)-->-.A)
.

(* Proposional calculus' axioms (Classical) *)
Inductive PROCA : Fo -> Type :=
| Intui :> forall f, PROCAI f -> PROCA f
| Ha11 : forall A, PROCA (A -\/ -.A)
.
```

Далее определим предикат выводимости из контекста в исчислении высказваний, для фиксированного набора аксиом axs и некоторого контекста ctx. (На самом деле, axs – предикат на типе формул, истинный на аксиомах. Аналогичное замечание верно и для ctx.)

```
Section PR.
 Context (axs:Fo -> Type).
 Context (ctx:Fo -> Type).
 Inductive PR : Fo -> Type :=
 | hyp (A : Fo) : ctx A -> PR A
 | Hax :> forall (A : Fo), (axs A) -> PR A
 | MP (A B: Fo) : (PR A)->(PR (A-->B))->(PR B)
 .
End PR.
```

Легко показать, что формула $A \to A$ выводима в обоих исчислениях, в силу того, что она выводима интуиционистки, а всякая интуиционистки выводимая формула классически выводима. (Утверждение теоремы “subcalc”.)

```
Definition AtoA_I {ctx} (A:Fo) : PR PROCAI ctx (A-->A).
Proof.
apply MP with (A-->(A-->A)).
apply Hax, Ha1.
apply MP with (A-->((A-->A)-->A)).
apply Hax, Ha1.
apply Hax, Ha2.
Defined.

Theorem subcalc {ctx} {B} : (PR PROCAI ctx B) -> (PR PROCA ctx B).
Proof.
intro m.
induction m.
+ apply hyp. assumption.
+ apply Hax. apply Intui. assumption.
+ eapply MP. exact IHm1. exact IHm2.
Defined.

Definition AtoA {ctx} (A:Fo) : PR PROCA ctx (A-->A)
 := subcalc (AtoA_I A).
```

Существуют различные семантики логики высказываний, на данный момент дадим им названия, а

определения и смысл – раскроем в следующих параграфах: “семантика двойного отрицания”, “булева семантика”, “классическая семантика”, “семантка Крипке”.

4.4 Семантика двойного отрицания

Определение 4.9. Функция foI_dn реализует замену атомов в формуле на высказывания (объекты типа Prop), начинающиеся с двойного отрицания.

```
(*
   Double negation semantics for the classical
   propositional logic("CPRoL"). (definitions)
*)
Section foI_dn. (* Entails for double negation. *)
 Context (val:PropVars.t->Prop).
 Fixpoint foI_dn (f:Fo) : Prop :=
 match f with
  | Atom p => (((val p)->False)->False)
  | Bot => False
  | f1 -/\ f2 => foI_dn f1 /\ foI_dn f2
  | f1 -\/ f2 => (((foI_dn f1 \/ foI_dn f2)->False)->False)
  | f1 --> f2 => (foI_dn f1) -> (foI_dn f2)
 end.
End foI_dn.
```

4.4.1 Корректность

Теорема 4.2. (о корректности семантики двойного отрицания) Всякая выводимая из аксиом PROCA формула переведится с помощью foI_dn в истинное высказывание, т.е. в населённый тип типа Prop.

```
(* Soundness of the double-negation semantics *)
Theorem sou_dn f (H:PR PROCA empctx f) :
   forall (val:PropVars.t->Prop), foI_dn val f.
Proof. intro val.
 induction H;firstorder.
 + induction a;firstorder.
   * induction p; firstorder.
       simpl. intros.
       induction C ;firstorder.
Defined.
```

4.5 Булева семантика

Сопоставим каждой пропозициональной переменной некоторый элемент булевого типа. Продолжим данную оценку на все формулы.

```
Section foI_bo.
 Context (val:PropVars.t->bool).
 Fixpoint foI_bo (f:Fo) : bool :=
 match f with
  | Atom p => (val p)
  | Bot => false
  | f1 -/\ f2 => andb (foI_bo f1) (foI_bo f2)
  | f1 -\/ f2 => orb (foI_bo f1) (foI_bo f2)
  | f1 --> f2 => implb (foI_bo f1) (foI_bo f2)
 end.
End foI_bo.
```

4.5.1 Корректность

Теорема 4.3. (о корректности булевой семантики) Всякая выводимая из аксиом PROCA формула переведится с помощью foI_bo в истинное высказывание, т.е. в терм true типа bool.

```
Theorem sou_bo f (H:PR PROCA empctx f) :
   forall (val:PropVars.t->bool), (foI_bo val f)=true.
Proof. intro val.
induction H.
+ destruct c.
+ induction a.
  * induction p; simpl; destruct (foI_bo val A), (foI_bo val B);
    try destruct (foI_bo val C); firstorder.
  * simpl; destruct (foI_bo val A); firstorder.
+ simpl in * |- *; destruct (foI_bo val A), (foI_bo val B); firstorder.
Defined.
```

4.5.2 Полнота

В этой главе докажем теорему о полноте булевой семантики: “Всякая тавтология выводима из пустого контекста в классическом исчислении высказываний”.
Рассмотренное доказательство для этой теоремы неконструктивное, для него используются следующие аксиомы:

```
(* Completeness *)
Axiom classicT : forall (P : Prop), {P} + {~ P}.
Axiom classicType : forall (P : Type), sum P (P->False).
Axiom NNPP_Type : forall p : Type, ((p->False)->False) -> p.
```

Они позволяют неконструктивно построить функции fu и fuT, отображающие всякий насленный тип из Prop или, соответственно, Type в элемент true типа bool, а пустые типы – в элемент false. Эти функции и их свойства определены в следующем фрагменте кода.

```
Definition fu : Prop -> bool := fun P =>
if (classicT P) then true else false.

Definition fuT : Type -> bool := fun P =>
if (classicType P) then true else false.

Lemma fuPtrue P : (fu P = true)->P.
Proof.
intro K. unfold fu in K.
destruct (classicT P). exact p. inversion K.
Defined.

Lemma fuPfalse P : (fu P = false)->~P.
Proof.
intro K. unfold fu in K.
destruct (classicT P). inversion K. exact n.
Defined.

Lemma truePfu (P:Prop) : P->(fu P = true).
Proof.
intro K.
unfold fu.
destruct (classicT P). reflexivity.
destruct (n K).
Defined.

Lemma falsePfu (P:Prop) : ~P->(fu P = false).
Proof.
intro K.
unfold fu.
destruct (classicT P).
destruct (K p).
reflexivity.
Defined.

Lemma fuTPtrue P : (fuT P = true)->P.
```

```
Proof.
intro K. unfold fuT in K.
destruct (classicType P). exact p. inversion K.
Defined.

Lemma fuTPfalse P : (fuT P = false)->(P->False).
Proof.
intro K. unfold fuT in K.
destruct (classicType P). inversion K. exact f.
Defined.

Lemma trueTPfu (P:Type) : P->(fuT P = true).
Proof.
intro K.
unfold fuT.
destruct (classicType P). reflexivity.
destruct (f K).
Defined.

Lemma falseTPfu (P:Type) : (P->False)->(fuT P = false).
Proof.
intro K.
unfold fuT.
destruct (classicType P).
destruct (K p).
reflexivity.
Defined.
```

Определение 4.10. Контекст G – непротиворечивый, если неверно, что из него выводится некоторая формула и её отрицание.

```
(* Consistent *)
Definition consi G :=
 (sigT (fun A=> prod (PR PROCA G A) (PR PROCA G (Neg A)))->False).
```

Определение 4.11. Контекст G – полный, если он непротиворечивый и для каждой формулы A из него выводся или A, или $\neg A$.

```
Definition complete G := prod (consi G)
 (forall A:Fo, sum (PR PROCA G A) (PR PROCA G (Neg A))).
```

Далее будем считать, что между формулами и натуральными числами есть биекция nomer, примем это без доказательства.

```
Section natnum.
Context (nomer:nat->Fo).
Context (remon:Fo->nat).
Context (nomer_sect: forall x:nat, remon (nomer x) = x).
Context (nomer_retr: forall x:Fo, nomer (remon x) = x).
```

Перейдём непосредственно к доказательству теоремы о полноте через лемму Линденбаума. Предположим, что есть некоторое непротиворечивое множество формул Γ.

Определение 4.12. Все формулы пронумерованы, поэтому пусть семейство Γ_n (в коде обозначено как "fam n") задаётся такими соотношениями: $\Gamma_0 = \Gamma$, а Γ_{n+1} – это пополнение Γ_n формулой номер n, если такое пополнение непротиворечиво, а если противоречиво, то $\Gamma_n + 1 = \Gamma_n$.

```
Section W.
Context (Gamma:Fo->Type).
Context (CG:consi Gamma).
Fixpoint fam (n:nat) : Fo->Type :=
match n with
| 0 => Gamma
| S n =>
        let G:= nomer n in
```

```
        let extctx := (add2ctx G (fam n)) in
         if (fu(consi extctx)) then extctx else (fam n)
end.
```

Определение 4.13. Множество $\Delta = \bigcup_{n\in\mathbb{N}} \Gamma_n$ – это объединение всего рассматриваемого семейства множеств.

```
Definition Delta : Fo->Type := fun f=> sigT (fun n=> fam n f).
```

Лемма 4.4. Семейство Γ_n при возрастании n монотонно неубывает относительно отношения включения.

```
Lemma fam_mon f n : fam n f -> fam (S n) f.
Proof.
intro H.
simpl.
destruct (fu (consi (add2ctx (nomer n) (fam n)))).
2 : exact H.
right. exact H.
Defined.
```

Определение 4.14. small – функция, которая по доказательству формулы A из контекста Δ находит некоторый номер n, такой что из Γ_n также выводится A.

```
(* Here we find some n, such that Γ_n can prove A *)
Section finite_argument.
Definition small (A : Fo) (p : PR PROCA Delta A) : nat.
Proof.
induction p.
+ destruct c as [x fxA]. exact x.
+ exact 0.
+ exact (max IHp1 IHp2).
Defined.
```

Далее рассмотрим секцию, независимую от текущего доказательства, взятую из [14]. В ней серия лемм, показывающая в итоге, что для всякого всякое монотонного неубывающего семейства Γ_n верно, что $\Gamma_n \subseteq \Gamma_{\max(m,n)}$ и применим этот факт в теореме indstep1.

```
Section upward_inhabited_family.
Context (fam : nat -> Type).
Context (fam_mon : forall n, fam n -> fam (S n)).
Lemma fam_gt n k (hb : fam n) : fam (n + k).
Proof. now rewrite Nat.add_comm; induction k; auto; apply fam_mon. Qed.
Lemma fam_leq n m (hl : n <= m) (hb : fam n) : fam m.
Proof. now rewrite <- (Nat.sub_add _ _ hl), Nat.add_comm; apply fam_gt. Qed.
Lemma mxinh m n (hb : fam n) : fam (max n m).
Proof. exact (fam_leq _ _ (Nat.le_max_l _ _) hb). Qed.
End upward_inhabited_family.

Lemma indstep1 m n (A : Fo) (c : fam n A) : fam (max n m) A.
Proof.
apply (mxinh (fun k=>fam k A) (fam_mon A)).
assumption.
Defined.
```

Лемма 4.5. $\forall n \forall m \max(m,n) = \max(n,m)$.

```
Theorem max_sym n : forall m, (max m n) = (max n m).
Proof.
induction n.
+ intros. simpl. induction m; trivial.
+ intros. simpl. induction m; try trivial.
simpl. apply f_equal. apply IHn.
Defined.
```

Лемма 4.6. lemJ0 – формулы выводимые из контекста номер Q, выводимы из контекста номер $\max(Q, R)$.

```
Lemma lemJ0 A Q R: PR PROCA (fam Q) A -> PR PROCA (fam (max Q R)) A.
Proof.
intro x. induction x.
+ apply hyp. apply indstep1. exact c.
+ apply Hax. exact a.
+ simpl in *|-*.
  eapply MP.
  - apply IHx1.
  - apply IHx2.
Defined.
```

Здесь покажем, что определённая выше функция small удовлетворяет её спецификации.

```
Definition it_works (A : Fo) (p : PR PROCA Delta A) :
PR PROCA (fam (small A p)) A .
Proof.
induction p.
+ apply hyp.
  simpl.
  destruct c. assumption.
+ apply Hax. exact a.
+ simpl in *|-*.
  eapply MP.
  - apply lemJ0. exact IHp1.
  - rewrite max_sym. apply lemJ0. exact IHp2.
Defined.
End finite_argument.
```

Покажем индукцией, что на каждом шаге расширения Γ получилось непротиворечивое множество.

```
Lemma consi_fam n : consi (fam n).
Proof.
induction n.
+ simpl. exact CG.
+ simpl.
  pose (t:= fu(consi (add2ctx (nomer n) (fam n)))).
induction (fu (consi (add2ctx (nomer n) (fam n)))) eqn:h.
  2 : { exact IHn. }
apply fuPtrue.
exact h.
Defined.
```

Теорема 4.7. Δ – непротиворечивое множество.

```
Theorem condel : consi Delta.
Proof.
unfold consi.
intros [A [p1 p2]].
assert (q1:=it_works _ p1).
assert (q2:=it_works _ p2).
eapply lemJ0 in q1.
eapply lemJ0 in q2.
rewrite max_sym in q2.
eapply consi_fam.
exists A.
split.
- exact q1.
- exact q2.
Defined.
```

Лемма 4.8. Δ включает в себя всякий контекст из семейства.

```
Lemma delta_mon A j : fam j A -> Delta A.
```

```
Proof. intro H.
unfold Delta. exists j. exact H.
Defined.
```

Лемма 4.9. Всё, что доказуемо из некоторого контекста из последовательности, доказуемо и из Δ.

```
Lemma pr_del_mon A j: PR PROCA (fam j) A -> PR PROCA Delta A.
Proof. intro H.
induction H.
+ apply hyp. eapply delta_mon. exact c.
+ apply Hax. exact a.
+ eapply MP. exact IHPR1. exact IHPR2.
Defined.
```

Теорема 4.10. Δ – полное множество.

```
Theorem comdel : complete Delta.
Proof.
unfold complete.
split.
+ exact condel.
+ intro A.
pose (i:=remon A).
pose (extctx := add2ctx A (fam i)).
destruct (classicT (consi extctx)).
- left.
  assert (R:fam (S i) A).
  { simpl. unfold i.
    rewrite nomer_retr.
    fold i. rewrite truePfu.
    2 : exact c.
    left. trivial. }
  apply hyp.
  eapply delta_mon. exact R.
- right. unfold consi,not in n.
  apply NNPP_Type in n.
  destruct n as [a [H1 H2]].
  apply Ded in H1.
  apply Ded in H2.
  eapply pr_del_mon.
  eapply MP.
  2 : eapply MP.
  3 : { apply Hax,Intui. eapply Ha10. }
  exact H2.
  exact H1.
Defined.
End W.
```

В следующей секции считываем оценку для переменных с полного непротиворечивого множества. Пусть Δ – полное множество. Определим оценку ν, для этого в качестве истинных переменных возьмём те и только те, истинность которых выводится из Δ.

```
(* p.49 *)
Section lemma2.
Context (Delta:Fo->Type) (CG:complete Delta).
Definition nu (p:PropVars.t) : bool
:= fuT (PR PROCA Delta p).
(*if (classicType (PR PROCA Delta p)) then true else false.*)
```

Индукцией по формуле, покажем, что истинность любой формулы при оценке ν влечёт её выводимость из Δ, а ложность – выводимость $\neg A$ из Δ.

```
Lemma lem_2_1 (A:Fo):
 (((foI_bo nu A)=true)->(PR PROCA Delta A)) *
 (((foI_bo nu A)=false)->(PR PROCA Delta (Neg A))).
```

```
Proof.
induction A.
+ simpl. split; intro w.
  - unfold nu in w.
    apply fuTPtrue in w.
    exact w.
  - unfold nu in w.
    destruct CG as [c s].
    destruct (s p) as [H|H].
    2 : { exact H. }
    exfalso.
    apply (fuTPfalse (PR PROCA Delta p)) in w.
    * destruct w.
    * exact H.
+ split;simpl;intros p.
  - inversion p.
  - unfold Neg. apply AtoA.
+ split;simpl;intros p.
  - rewrite andb_true_iff in p.
    destruct p as [p1 p2].
    destruct IHA1 as [IHA1 _]. apply IHA1 in p1.
    destruct IHA2 as [IHA2 _]. apply IHA2 in p2.
    eapply MP.
    2 : eapply MP.
    3 : eapply Hax, Intui, Ha5.
    exact p2.
    exact p1.
  - destruct IHA1 as [_ IHA1].
    destruct IHA2 as [_ IHA2].
    apply andb_false_elim in p.
    destruct p.
    * apply IHA1 in e.
      eapply MP. exact e.
      apply contrap.
      apply Hax, Intui, Ha3.
    * apply IHA2 in e.
      eapply MP. exact e.
      apply contrap.
      apply Hax, Intui, Ha4.
+ split;simpl;intros p.
  - apply orb_true_elim in p.
    destruct p.
    * destruct IHA1 as [IHA1 _].
      apply IHA1 in e.
      eapply MP. exact e.
      eapply Hax, Intui. constructor.
    * destruct IHA2 as [IHA2 _].
      apply IHA2 in e.
      eapply MP. exact e.
      eapply Hax, Intui. constructor.
  - apply orb_false_elim in p.
    destruct p as [p1 p2].
    destruct IHA1 as [_ IHA1].
    destruct IHA2 as [_ IHA2].
    apply IHA1 in p1.
    apply IHA2 in p2.
    eapply MP. exact p2.
    eapply MP. exact p1.
    eapply Hax, Intui.
    constructor.
+ split;simpl;intros p.
  - rewrite <- leb_implb in p.
    unfold leb in p.
    destruct (foI_bo nu A1).
    * destruct IHA1 as [IHA1 _].
      destruct IHA2 as [IHA2 _].
```

```
      apply IHA2 in p.
      eapply MP. exact p. apply Hax, Intui, Ha1.
    * destruct IHA1 as [_ IHA1].
      eapply MP.
      apply IHA1; reflexivity.
      apply Hax, Intui, Ha9.
  - destruct (foI_bo nu A1); simpl in *|-*.
    * destruct IHA2 as [_ IHA2].
       apply IHA2 in p.
      eapply MP. exact p. eapply contrap.
      apply Ded.
      eapply MP. 2 : { apply hyp. left. reflexivity. }
      destruct IHA1 as [IHA1 _].
      assert (IHA1:=IHA1 eq_refl).
      apply weak, IHA1.
    * inversion p.
Defined.
End lemma2.
```

Определение 4.15. Контекст ctx совместный, если существует оценка, при которой все формулы из ctx истинны.

```
Definition satisf ctx : Prop :=
 exists (v:PropVars.t->bool), forall g, ctx g -> foI_bo v g = true.
```

Лемма 4.11. Всякий полный контекст совместный.

```
Lemma compl ctx : (complete ctx) -> (satisf ctx).
Proof.
intro H.
exists (nu ctx).
intros g ctxg.
destruct (classic (foI_bo (nu ctx) g = true)).
+ assumption.
+ apply not_true_iff_false in H0.
exfalso.
destruct (lem_2_1 ctx H g) as [_ I2].
apply I2 in H0.
eapply hyp in ctxg.
destruct H as [c s].
unfold consi in c.
apply c.
exists g.
split.
- exact ctxg.
- exact H0.
Defined.
```

Лемма 4.12. Любое подмножество совместного множества формул – тоже совместное множество.

```
Lemma satsubctx ctx CTX
(i:forall x, ctx x -> CTX x):
 (satisf CTX) -> (satisf ctx).
Proof.
unfold satisf.
intros [v H].
exists v. intros g cg.
apply H, i, cg.
Defined.
```

Лемма 4.13. Полное множество, полученное пополнением контекста, содержит формулы этого контекста.

```
Lemma lem3 ctx : forall x : Fo, ctx x -> Delta ctx x.
```

```
Proof.
intros.
eapply (delta_mon ctx x 0).
simpl.
exact X.
Defined.
```

Лемма 4.14. Всякое непротиворечивое множество формул – совместно.

```
Lemma compl2 ctx : (consi ctx) -> (satisf ctx).
Proof.
intro H.
apply (satsubctx ctx (Delta ctx)).
exact (lem3 _).
apply compl.
apply comdel.
exact H.
Defined.
```

Определение 4.16. Множество формул противоречиво, если из него выводится некоторая формула и её отрицание.

```
Definition inconsi G :=
 {A : Fo & (PR PROCA G A * PR PROCA G (Neg A))%type}.
```

Лемма 4.15. Всякая невыполнимый контекст противоречив.

```
Lemma cimp ctx : (satisf ctx->False) -> inconsi (ctx).
Proof.
intro x.
unfold inconsi.
apply NNPP_Type.
intro K.
apply compl2 in K.
exact (x K).
Defined.
```

Лемма 4.16. Если расширение контекста формулой f противоречиво, то из исходного контекста выводится $\neg f$.

```
Lemma vse ctx f : inconsi (add2ctx f ctx) -> PR PROCA ctx (-.(f)).
Proof.
intro H.
destruct H as [A [p1 p2]].
apply Ded in p1.
apply Ded in p2.
eapply MP. exact p2.
eapply MP. exact p1.
apply Hax, Intui, Ha10.
Defined.
```

Далее вывод снятия двойного отрицание в классическом исчислении высказываний.

```
Lemma dne ctx f : PR PROCA ctx ((-.(-.f))-->f).
Proof.
apply Ded.
assert (Q:PR PROCA ctx (f -\/ (-.f))).
{ apply Hax, Ha11. }
eapply MP.
apply weak.
exact Q.
1 : eapply MP.
2 : eapply MP.
```

```
3 : { eapply Hax, Intui, Ha8. }
+ apply Ded.
  eapply MP. apply hyp. left. reflexivity.
  eapply MP.
  − apply weak. apply hyp. left. reflexivity.
  − apply Hax, Intui, Ha9.
+ apply AtoA.
Defined.

Lemma dnei ctx f : PR PROCA ctx (−.(−.f)) −> PR PROCA ctx f.
Proof.
intro H. eapply MP. exact H. apply dne.
Defined.
```

Теорема 4.17. Всякая тавтология выводима из пустого контекста в классическом исчисление высказываний.

```
 Theorem cmpl_bo f
(H: forall (val:PropVars.t−>bool), (foI_bo val f)=true)
 : PR PROCA empctx f.
Proof.
apply dnei.
apply vse.
apply cimp.
intro J.
unfold satisf in J.
destruct J as [val M].
assert (M:=M (−.f)).
assert (H:=H val).
assert (U:add2ctx (−.f) empctx (−.f)).
{ left. reflexivity. }
assert (M:=M U).
simpl in M.
destruct (foI_bo val f).
+ simpl in M. inversion M.
+ inversion H.
Defined.

End natnum.
```

4.6 Классическая семантика

Если используется неконструктивное классическое расшиерение языка Coq аксиомой исключённого третьего, то можно определить семантику исчисления PROCA, классическими высказываниями типа Prop.

```
Section foI_cl.
 Context (val:PropVars.t−>Prop).
 Fixpoint foI_cl (f:Fo) : Prop :=
 match f with
  | Atom p => (val p)
  | Bot => False
  | f1 −/\ f2 => (foI_cl f1) /\ (foI_cl f2)
  | f1 −\/ f2 => (foI_cl f1) \/ (foI_cl f2)
  | f1 −−> f2 => (foI_cl f1) −> (foI_cl f2)
 end.
End foI_cl.
```

4.7 Корректность

Теорема 4.18. (о корректности классической семантики) Всякая выводимая из аксиом PROCA формула переведится с помощью foI_cl в истинное высказывание, т.е. в населённый тип типа Prop.

```
Theorem sou_cl f (H:PR PROCA empctx f) :
```

```
   forall (val:PropVars.t->Prop), foI_cl val f.
Proof. intro val.
 induction H;firstorder.
 + induction a;firstorder.
   * induction p; firstorder.
   * simpl.
     destruct (classic (foI_cl val A)); firstorder.
Defined.
```

4.8 Семантика Крипке

Далее представлена семантика Крипке для интуиционистской логики. Определена функция foI_kr, сопоставляющая миру и формуле некоторой модели Крипке пропозицию, содержащую доказательства истинности этой формулы в данном мире. (Пустую, если формула ложная.)

```
Open Scope type_scope.
Section WR.
Variables (W:Set) (R:W->W->Prop) (R_transitive : transitive W R)
(R_reflexive : reflexive W R).
Variables (vf:PropVars.t -> W -> Prop)
(mvf: forall (x y : W)(p:PropVars.t), vf p x -> R x y -> vf p y).

Section foI_kr. (* Entails *)
Fixpoint foI_kr (x:W) (f:Fo) : Prop. (* := *)
Proof using W R R_transitive R_reflexive vf mvf.
exact (
match f with
  | Atom p => (vf p x)
  | Bot => False
  | f1 -/\ f2 => foI_kr x f1 /\ foI_kr x f2
  | f1 -\/ f2 => foI_kr x f1 \/ foI_kr x f2
  | f1 -~> f2 =>
(forall y:W, R x y -> ((foI_kr y f1) -> (foI_kr y f2)))
end
).
Defined.
End foI_kr.
```

Теорема 4.19. Отрицание формулы f истинно в некотором мире x тогда и только тогда, когда f ложна во всех мирах видимых из x.

```
Theorem utv1 x f: foI_kr x (f-~>Bot) <-> forall y, R x y -> not (foI_kr y f).
Proof.
simpl. unfold not. reflexivity.
Defined.
```

Теорема 4.20. Если формула истинна в мире x, то она истинна в любом следующем из него мире.

```
Theorem utv2 x y f : foI_kr x f -> R x y -> foI_kr y f.
Proof.
intros H1 H2.
induction f.
+ simpl in * |- *.
  apply mvf with x. apply H1. apply H2. (* , H2, H1 *)
+ exact H1.
+ simpl in * |- *.
  destruct H1 as [u1 u2].
  exact (conj (IHf1 u1) (IHf2 u2)).
+ simpl in * |- *.
  destruct H1 as [u1|u2].
  left. exact (IHf1 u1).
  right. exact (IHf2 u2).
+ simpl in * |- *.
```

```
  intros.
  apply H1.
  * apply (R_transitive x y y0 H2 H).
  * exact H0.
Defined.

(* Soundness of the Kripke semantics of IPro *)
Theorem sou_kr f (H:PR PROCAI empctx f) : forall x, foI_kr x f.
Proof.
induction H.
+ destruct c. (*simpl in i. destruct i.*)
+ induction a.
  * simpl. intros.
    simpl in * |- *.
    apply utv2 with (x:=y).
    - exact H0.
    - exact H1.
  * simpl. intros.
eapply (H0 y1 _ _ y1).
apply R_reflexive.
apply H2.
apply H3.
apply H4.
  * simpl. intros. destruct H0 as [LH0 RH0]. exact LH0.
  * simpl. intros. destruct H0 as [LH0 RH0]. exact RH0.
  * simpl. intros x y pxy yA z pyz zB. split.
    exact (utv2 y z A yA pyz).
   exact zB.
  * simpl. intros x y pxy H. left. exact H.
  * simpl. intros x y pxy H. right. exact H.
  * simpl. intros.
    destruct H4.
    - unshelve eapply H0. 2: exact H4. exact (R_transitive y y0 y1 H1 H3).
    - unshelve eapply H2. exact H3. exact H4.
  * simpl. intros. exfalso. eapply H0 with y0. exact H1. exact H2.
  * simpl. intros.
    apply (H2 y1 H3 H4 y1).
    apply R_reflexive.
    apply H0.
    eapply R_transitive. exact H1. exact H3. exact H4.
+ simpl in * |- *.
  intro x.
  unshelve apply (IHPR2 x).
  unshelve apply R_reflexive.
  unshelve apply IHPR1.
  Unshelve.
  exact (R_transitive y y0 y1 H1 H3).
  exact H4.
Defined.
End WR.
```

5 Модель логики предикатов в Coq

5.1 Логика предикатов

В данном разделе будут верифицированы теоремы о логике предикатов без равенства. Будет имплементирована выводимость из контекста, доказаны некоторые теормы про неё. При написании кода в этой главе были использованые определения и доказательства из главы 4 [1] и главы 6 из [12].Будет рассматриваться вариант исчисления предикатов, в котором индивидные переменные не разделяются на свободные и связанные. Также далее будем считать, что кванторы обладают наибольшим приоритетом при записи формул.

Таблица 2: Приоритет связок

Приоритет	Связка
0	$\forall$, $\exists$, $\neg$
1	$\wedge$
2	$\vee$
3	$\rightarrow$

Таблица 3: Схемы аксиом логики предикатов с равенством

назвывние	утверждение	условие
impI	$A \rightarrow (B \rightarrow A)$	-
impE	$(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$	-
andE1	$A \wedge B \rightarrow A$	-
andE2	$A \wedge B \rightarrow B$	-
andI	$A \rightarrow (B \rightarrow (A \wedge B))$	-
orI1	$A \rightarrow A \vee B$	-
orI2	$B \rightarrow A \vee B$	-
orE	$(A \rightarrow C) \rightarrow ((B \rightarrow C) \rightarrow (A \vee B \rightarrow C))$	-
negI	$(A \rightarrow B) \rightarrow ((A \rightarrow \neg B) \rightarrow \neg A)$	-
negE	$A \rightarrow (\neg A \rightarrow B)$	-
LEM	$A \vee \neg A$	-
$\forall$E	$\forall x A \rightarrow A(t/x)$	FFI(t, x, A)
$\exists$I	$A(t/x) \rightarrow \exists x A$	FFI(t, x, A)
=refl	$\forall w(w = w)$	-
=subs	$(x = y) \rightarrow (A(x/y) \rightarrow A)$	FFI(x, y, A)

Схема аксиом логики предикатов без равенства получается, если удалить последние две аксиомы. Здесь мы обозначили как FFI(t, x, A) утверждение "Терм t свободен для подстановки вместо переменной x в формуле A", и как $A(t/x)$ – результат корректной подстановки терма t вместо x.

Таблица 4: Правила вывода логики предикатов

название	правило	условие
MP	$\dfrac{A \rightarrow B \quad A}{B}$	-
$\forall$I	$\dfrac{A \rightarrow B}{A \rightarrow \forall x B}$	$x \notin \mathrm{FV}(A)$
$\exists$E	$\dfrac{A \rightarrow B}{\exists x A \rightarrow B}$	$x \notin \mathrm{FV}(B)$

В данной работе имплементируем эквивалентную аксиоматизацию логики предикатов, введя вместо правил Бернайса аксиомы Бернайса и добавив правило обобщения. Также обобщим выводимость в логике предикатов до выводимости из контекста.

Таблица 5: Правила вывода из контекста и аксиомы Бернайса

название	утверждение	условие
GEN	$\dfrac{\Gamma \vdash A}{\Gamma \vdash \forall x A}$	$x \notin \mathrm{FV}(\Gamma)$
$\forall$I	$\forall x(A \to B) \to (A \to \forall x B)$	$x \notin \mathrm{FV}(A)$
$\exists$E	$\forall x(A \to B) \to (\exists x A \to B)$	$x \notin \mathrm{FV}(B)$
MP	$\dfrac{\Gamma \vdash A \to B \quad \Gamma \vdash A}{\Gamma \vdash B}$	-

5.2 Язык первопорядковой теории

Определение 5.1. Сигнатура – это упорядоченная тройка разрешимых множеств с разрешимым равенством $\sigma = \langle \mathrm{Var}, \mathrm{Func}, \mathrm{Pred} \rangle$, где

1) Var – множество индивидных переменных,
2) Func – множество функциональных символов,
3) Pred – множество предикатных символов.

Пример 5.1. Язык теории множеств можно задать следующим образом:

- $\mathrm{Var} \stackrel{\text{def}}{:=} \{a, b, c, \ldots, x, y, z, a_0, b_0, c_0, \ldots\}$ – множество/тип индивидных переменных для множеств,
- $\mathrm{Func} \stackrel{\text{def}}{:=} \varnothing$ – пустое множество функциональные символов,
- $\mathrm{Pred} \stackrel{\text{def}}{:=} \{A_1^2, A_2^2\}$ – множество предикатных символов.

Для данных предикатных символов мы будем использовать инфиксную нотацию: $(x = y)$ для $A_1^2(x, y)$, $(x \in y)$ для $A_2^2(x, y)$.

Теперь рассмотрим теоретико-типовой подход: в данном случае Var, Func и Pred - это типы, причём на каждом типе из $\{\mathrm{Var}, \mathrm{Func}, \mathrm{Pred}\}$ должно быть определено своё разрешимое отношение графического равенства.

Для этого применим одна из сильных сторон Coq – гибкую систему модулей. Модули расширяют формальную систему CIC новыми правилами. Эти правила могут быть найдены в [10]. Теперь, пусть мы хотим задать модуль A модульного типа B. Чтобы это сделать, необходимо для каждого параметра из B дать определение в A, а для каждой аксиомы в B – теорему в A. Иными словами, A – модель теории B. Для рассматриваемого случая разрешимого равенства на типе, имя UsualDecidableTypeFull – это тип модуля. Таким модульным типом, например, обладают натуральные числа с отношением равенства, о чём свидетельствует компиляция без ошибок следующей программы на Coq. (Следующий модуль можно использовать как реализацию для переменных, функциональных символов и предикатных символов. Это позволяет доказывать что-либо не обязательно внутри модуля с параметрами SetVars, FuncSymb, PredSymb.)

```
Require Import Coq.Structures.Equalities.
Require Import Arith.PeanoNat.
Module nat_as_symb : UsualDecidableTypeFull.
Definition t:=nat.
Definition eq := @eq nat.
Definition eq_refl:=@eq_refl nat.
Definition eq_sym:=@eq_sym nat.
Definition eq_trans:=@eq_trans nat.
Definition eq_equiv:Equivalence eq := Nat.eq_equiv.
Definition eq_dec := Nat.eq_dec.
Definition eqb:=Nat.eqb.
Definition eqb_eq:=Nat.eqb_eq.
End nat_as_symb.
```

5.3 Начало программной реализации

Теперь начнём описывать программную реализацию логики предикатов. Потребуются следующие библиотеки:

```
Require Import Bool.
Require Import Coq.Lists.List.
Require Import Coq.Vectors.Vector.
Require Import Coq.Structures.Equalities.
```

В начале единым образом обозначим предметную область Omega – как тип всех высказываний и операции на ней и значение констант.

```
(** 1. SOME NOTATIONS **)
Notation Omega := Prop.
Definition OFalse := False.
Definition OAnd := and.
Definition OOr := or.
Definition OImp := (fun x y:Omega => x->y).

(** 2. TRIVIAL LEMMA **)
Lemma my_andb_true_eq :
  forall a b:bool, a && b = true -> a = true /\ b = true.
Proof.
  destr_bool. auto.
Defined.
```

5.4 Леммы про булевы вектора

Теорема 5.2. Если дизъюнкция элементов булева вектора произвольной длины равна лжи, то каждый элемент вектора равен лжи.

В данном разделе описывается доказательство этих теорем в Coq. Они потребуются в дальнейшем, для разбора терма, свободного от параметра x, на подтермы, свободные от того же параметра.

Для начала определим сворачивание вектора произвольной длины.

Определение 5.2. Операция fold_left, или сворачивание вектора слева некоторой бинарной операцией *op* – это последовательное применение бинарной операции к вектору и выделенной ячейке таким образом, что результат операции сохраняется в ячейке. Иными словами,

fold_left op [x1 , ... , xn] i = (op ... (op (op i x1) x2) ... xn).

Элемент в выделенной ячейке на первом шаге алгоритма будем называть начальным.

```
(** 3. ALL THEN SOME (VECTOR) **)
Import VectorNotations.

Fixpoint ATS_B2 (n:nat) (l:t bool n) : fold_left orb true l = true.
Proof.
destruct l; simpl.
reflexivity.
apply ATS_B2.
Defined.

Fixpoint ATS_B0 b (n:nat) (l:t bool n) :
fold_left orb false (b :: l) = orb b (fold_left orb false l) .
Proof.
destruct l.
simpl. firstorder.
simpl.
destruct b.
simpl.
apply ATS_B2.
simpl.
reflexivity.
Defined.
```

Введём следующее, чисто техническое, определение для сокращения объёма выкладок.

Определение 5.3. ATS_G(h,n,l) – утверждение, что свёртка операцией "или" вектора l длины n с приписанныv элементом h равна "лжи".

```
Definition ATS_G h (n:nat) (l:Vector.t bool n) : Prop :=
 @eq bool (@fold_left bool bool orb false (S n) (cons bool h n l)) false.
```

Лемма 5.3. Вектор положительной длины есть некоторой вектор неотрицательной длины с приписанным элементом.

```
Lemma vp1 (n:nat) (l : t bool (S n)) : exists (q:bool) (m:t bool n), l = (q::m).
Proof.
apply (@caseS bool (fun n =>
fun (l : t bool (S n)) => exists (q : bool) (m : t bool n), l = q :: m)).
intros.
exists h.
exists t.
reflexivity.
Defined.
```

Теорема 5.4. Если свёртка элементов булева вектора операцией "или" равна лжи, то элемент номер p равен лжи.

```
Fixpoint all_then_someV (n:nat) (l:Vector.t bool n) {struct l} :
(Vector.fold_left orb false l ) = false ->
(forall p, (Vector.nth l p ) = false).
Proof.
intros.
destruct l.
{ inversion p. }
{ (*fold G in H.*)
  assert (vari : ATS_G h n l).
  { exact H. }
  clear H.
  revert h l vari.
  set (P := fun n p => forall (h : bool) (l : t bool n) (_ :ATS_G h n l),
    @eq bool (@nth bool (S n) (cons bool h n l) p) false).
  revert n p.
  fix loop 1.
  intros n;revert n.
  unshelve eapply (@Fin.rectS P).
  { unfold P.
    intros.
    simpl.
    unfold ATS_G in H.
    rewrite -> (ATS_B0 h n l) in H.
    apply orb_false_elim in H as [H _]; exact H.
  }
  { unfold P.
    intros.
    unfold ATS_G in H0.
    simpl in |- *.
    rewrite -> (ATS_B0 h (S n) l) in H0.
    apply orb_false_elim in H0 as [_ HH1].
    assert (Y0 := vp1 n l).
    destruct Y0 as [Y01 [Y02 Y03]].
    rewrite -> Y03.
    apply H.
    unfold ATS_G.
    rewrite <- Y03.
    exact HH1. }
}
Defined.
```

5.5 Предварительные леммы

В данном разделе доказаны вспомогательные теоремы.

```
Require Setoid.
Require Bool.Bool.
```

Теорема 5.5. *ap* – "любая функция на равных аргументах даёт равные значения".

```
(** 4. MISC **)
Definition ap {A B}{a0 a1:A} (f:A->B) (h:a0=a1):((f a0)=(f a1))
:= match h in (_ = y) return (f a0 = f y) with
   | eq_refl => eq_refl
   end.
```

Для произвольного типа A в стандартной библиотеке определён тип-контейнер option, который либо содержит тип A, либо ничего не содержит.

```
Inductive option (A : Type) : Type :=
    Some : A -> option A | None : option A
```

Теорема 5.6. Конструктор Some – инъективен, как доказано в теореме $SomeInj$.

```
Theorem SomeInj {foo} :forall (a b : foo), Some a = Some b -> a = b.
Proof.
  intros a b H.
  inversion H.
  reflexivity.
Defined.
```

Лемма 5.7. Равные высказывания эквивалентны.

```
Lemma EqualThenEquiv A B: A=B -> (A<->B).
Proof. intro H. rewrite H. exact (iff_refl B). Defined.
```

В следующих леммах докажем факты эквивалентности формул, полученных с помощью связок из эквивалетных формул.

Лемма 5.8. AND_EQV – "конъюнкции эквивалентных формул – эквивалентны".

```
(* Lemmas START *)
Lemma AND_EQV : forall A0 B0 A1 B1 : Prop,
 (A0 <-> A1) -> (B0 <-> B1) -> ((A0 /\ B0) <-> (A1 /\ B1)).
Proof.
intros A0 B0 A1 B1 H0 H1.
rewrite H0.
rewrite H1.
reflexivity.
Defined.
```

Лемма 5.9. OR_EQV – "дизъюнкции эквивалентных формул – эквивалентны".

```
Lemma OR_EQV : forall A0 B0 A1 B1 : Prop,
 (A0 <-> A1) -> (B0 <-> B1) -> ((A0 \/ B0) <-> (A1 \/ B1)).
Proof.
intros A0 B0 A1 B1 H0 H1.
rewrite H0.
rewrite H1.
reflexivity.
Defined.
```

Лемма 5.10. IMP_EQV – "импликации эквивалентных формул – эквивалентны".

```
Lemma IMP_EQV : forall A0 B0 A1 B1 : Prop,
 (A0 <-> A1) -> (B0 <-> B1) -> ((A0 -> B0) <-> (A1 -> B1)).
Proof.
intros A0 B0 A1 B1 H0 H1.
rewrite H0.
rewrite H1.
reflexivity.
Defined.
```

Лемма 5.11. Если два свойства на некотором типе эквивалентны поточечно, то эквивалентны высказывания о том, что свойства верны на всех термах.

```
Lemma FORALL_EQV {X}: forall A0 A1 : X -> Prop,
 (forall m, A0 m <-> A1 m) -> ((forall m:X, A0 m) <-> (forall m:X, A1 m)).
Proof.
intros A0 A1 H0.
split.
+ intros.
  rewrite <- H0.
  exact (H m).
+ intros.
  rewrite -> H0.
  exact (H m).
Defined.
```

Лемма 5.12. Если два свойства на некотором типе эквивалентны поточечно, то эквивалентны высказывания о том, что существует терм данного типа.

```
Lemma EXISTS_EQV {X}: forall A0 A1 : X -> Prop,
 (forall m, A0 m <-> A1 m) -> ((exists m:X, A0 m) <-> (exists m:X, A1 m)).
Proof.
intros A0 A1 H0.
split.
+ intros.
  destruct H as [x Hx].
  exists x.
  rewrite <- H0.
  exact (Hx).
+ intros.
  destruct H as [x Hx].
  exists x.
  rewrite -> H0.
  exact (Hx).
Defined.
(* Lemmas END *)
```

5.6 Термы, подстановки и интерпретации

Определим термы и индукцию по ним. С этого параграфа и до конца данной главы, рассуждения проводятся внутри модуля, для которого заданы произвольные разрешимые типы индивидных переменных и алфавитов функциональных и предикатных символов.

```
(*** PREDICATE CALCULUS ***)
Module ALL_mod (SetVars FuncSymb PredSymb : UsualDecidableTypeFull).
Module Facts := BoolEqualityFacts SetVars.
```

Функциональный символ имеет валентность. Определим FSV – тип функциональных символов с отмеченной валентностью аналогично определению в [11].

Определение 5.4. Функциональный символ есть пара: буква в некотором алфавите и индекс – натуральное число, обозначающее валентность.

```
Record FSV := {
```

```
 fs : FuncSymb.t;
 fsv : nat;
}.
```

Далее определим, что есть терм.

Определение 5.5. Индуктивно зададим терм:

1. Всякая переменная – это терм.

2. Если $t_1, \ldots, t_n$ – термы, а f – функциональный символ валентности n, то $f(t_1, \ldots, t_n)$ – терм.

Определение 5.6. Назовём переменные – простыми термами, а все остальные термы – сложными термами.
Можно попытаться наивно задать терм как индуктивный тип определённый ниже.

```
Inductive Terms : Type :=
| FVC :> SetVars.t -> Terms
| FSC (f:FSV) : (Vector.t Terms (fsv f)) -> Terms.
```

Однако тогда Coq-ом генерируются неинформативные индуктивный и рекурсивный принципы.

```
Check Terms_ind.
(*
Terms_ind
     : forall P : Terms -> Prop,
       (forall t : SetVars.t, P t) ->
       (forall (f0 : FSV) (t : Vector.t Terms (fsv f0)),
        P (FSC f0 t)) -> forall t : Terms, P t
*)
Check Terms_rect.
(* Terms_rect
     : forall P : Terms -> Type,
       (forall t : SetVars.t, P t) ->
       (forall (f0 : FSV) (t : Vector.t Terms (fsv f0)),
        P (FSC f0 t)) -> forall t : Terms, P t *)
```

Получили, что система Coq выводит следующий индуктивный принцип:

Теорема 5.13. "Если для всех простых термов утверждение верно, и для всех сложных термов утверждение верно, то тогда утверждение верно для всех термом."

Этот принцип истинный, но слишком слабый, его не достаточно для доказательств индукцией по термам. Требуется следующий принцип:

Теорема 5.14. "Если для всех простых термов утверждение верно, и истинность утверждения для всякого сложного терма следует из истинности утверждения про все аргументы самого внешнего функционального символа, то тогда это утверждение верно про все термы."

Поэтому в данном случае необходимо переопределить схемы элиминации функциями других типов. Сначала отключим автоматическое определение схем элиминации по спецификации типаю

```
Unset Elimination Schemes.
Inductive Terms : Type :=
| FVC :> SetVars.t -> Terms
| FSC (f:FSV) : (Vector.t Terms (fsv f)) -> Terms.
Set Elimination Schemes.
```

После – задим их с помощью и с помощью паттерн-матчинга и неподвижных точек, базовых конструкций языка Gallina.

```
Definition Terms_ind (T : Terms -> Prop)
                     (H_FVC : forall sv, T (FVC sv))
                     (H_FSC : forall f v, (forall n, T (Vector.nth v n)) -> T (FSC f v)) :=
  fix loopt (t : Terms) : T t :=
```

```
    match t with
    | FVC sv => H_FVC sv
    | FSC f v =>
      let fix loopv s (v : Vector.t Terms s) : forall n, T (Vector.nth v n) :=
        match v with
        | @Vector.nil _ => Fin.case0 _
        | @Vector.cons _ t _ v => fun n => Fin.caseS' n (fun n => T (Vector.nth (Vector.cons _ t _ v) n))
                                                       (loopt t)
                                                       (loopv _ v)
        end in
      H_FSC f v (loopv _ v)
    end.
```

Следующее замечание не потребуется в дальнейшей работе. Для рекурсивного принципа Terms_rect для типов типа Type определение такое же, с точностью до замены вселенной Prop на вселенную Type. (Аналогично и для рекурсивного принципа Terms_rec для типов типа Set с заменой, соответственно, на вселенную Set.)

```
Definition Terms_rect (T : Terms -> Type)
                      (H_FVC : forall sv, T (FVC sv))
                      (H_FSC : forall f v, (forall n, T (Vector.nth v n)) -> T (FSC f v)) :=
  fix loopt (t : Terms) : T t :=
    match t with
    | FVC sv => H_FVC sv
    | FSC f v =>
      let fix loopv s (v : Vector.t Terms s) : forall n, T (Vector.nth v n) :=
        match v with
        | @Vector.nil _ => Fin.case0 _
        | @Vector.cons _ t _ v => fun n => Fin.caseS' n (fun n => T (Vector.nth (Vector.cons _ t _ v) n))
                                                       (loopt t)
                                                       (loopv _ v)
        end in
      H_FSC f v (loopv _ v)
    end.
```

Теперь можно определить некоторые стандартные математические понятия для заданного типа термов.

Определение 5.7. Обозначим как $u(t/\xi)$ – корректную подстановку терма t вместо переменной ξ в терм u.

1. Если $u = \kappa$ и $\kappa \in \mathrm{Var}$, то $\kappa(t/\xi) := \begin{cases} t, \text{если } \kappa = \xi \\ \kappa, \text{иначе} \end{cases}$,
2. $f(t_1, \ldots, t_k)(t/\xi) := f(t_1(t/\xi), \ldots, t_k(t/\xi))$.

Определение 5.8. Подстановка терма t вместо переменной ξ в терме u есть замена всех переменных ξ терме u на терм t.

```
Fixpoint substT (t:Terms) (xi: SetVars.t) (u:Terms): Terms
:=
   match u with
   | FVC s => if (SetVars.eqb s xi) then t else FVC s
   | FSC f t0 => FSC f (Vector.map (substT t xi) t0)
   end.
```

Определение 5.9. Переменная ξ – это параметр терма t, если $t = \xi$ или если t–сложный терм, а ξ – параметр одного из его аргументов.

```
Fixpoint isParamT (xi : SetVars.t) (t : Terms) {struct t} : bool :=
   match t with
   | FVC s => SetVars.eqb s xi
   | FSC f t0 => Vector.fold_left orb false (Vector.map (isParamT xi) t0)
   end.
```

Определение 5.10. Всякая переменная – терм. Для заданной оценки переменных val, интерпретация терма – это продолжение val, доопределённое на составных термах значением интерпретации функционального символа от на интепретации подтермов.

```
Section Interpretation1.
Context {X} {fsI:forall(q:FSV),(Vector.t X (fsv q))->X}.
Fixpoint teI
   (val:SetVars.t->X) (t:Terms): X :=
   match t with
   | FVC s => val s
   | FSC f t0 => fsI f (Vector.map (teI val) t0)
   end.
End Interpretation1.
```

5.7 Леммы про оценки термов

Пусть X – произвольный тип, а SetVars – разрешимый тип.

```
Section a.
Context {X:Type}.
```

Определение 5.11. cng – это оператор замены значения оценки(π) на некотором аргументе(ξ) из SetVars на некоторое значение(m).

$$(\text{cng } \pi\ \xi\ m) \stackrel{\text{def}}{:=} (\pi + (\xi \mapsto m))$$

```
Definition cng (val:SetVars.t -> X) (xi:SetVars.t) (m:X) :=
 (fun r:SetVars.t =>
 match SetVars.eqb r xi with
 | true => m
 | false => (val r)
 end).
```

Лемма 5.15. Утверждение dbl_cng: дважды сделать замену оценки в одной и той же точке – всё равно, что сделать только вторую замену.

```
Lemma dbl_cng (pi:SetVars.t->X) x m0 m1 :
 forall u, (cng (cng pi x m0) x m1) u = (cng pi x m1) u.
Proof.
intro u.
unfold cng.
destruct (SetVars.eqb u x).
reflexivity.
reflexivity.
Defined.

End a.
```

5.8 Формулы языка и их интерпретация.

Определение 5.12. Предикатный символ есть пара: буква в некотором алфавите и индекс – натуральное число, равное валентности.

```
Record PSV := MPSV{
 ps : PredSymb.t;
 psv : nat;
}.
```

Определение 5.13. Атомарная формула – это упорядоченная пара: предикатный символ некоторой валентности n и n-мерный вектор из термов.

Определение 5.14. Формула логики предикатов – это либо атомарная формула, либо набор из других формул, соединённых с помощью связок, либо набор: один из кванторов, переменная и формула.

```
Inductive Fo :=
 |Atom (p:PSV) : (Vector.t Terms (psv p)) -> Fo
 |Bot :Fo
 |Conj:Fo->Fo->Fo
 |Disj:Fo->Fo->Fo
 |Impl:Fo->Fo->Fo
 |Fora(x:SetVars.t)(f:Fo): Fo
 |Exis(x:SetVars.t)(f:Fo): Fo
.
```

Стандартным образом определим отрицание и истину через импликацию и константу “ложь”.

```
Definition Neg (A:Fo):Fo := Impl A Bot.
Definition Top:Fo := Neg Bot.
```

Введём нотацию формул для синтакса. Для этого просто припишем минус к стандартным обозначениям из Coq, тем, которые будут семантикой.

```
Module PredFormulasNotationsASCII.
 Notation " x --> y ":=(Impl x y) (at level 80, right associativity) : pretxtnot.
 Notation " x -/\ y ":=(Conj x y) (at level 80) : pretxtnot.
 Notation " x -\/ y ":=(Disj x y) (at level 80) : pretxtnot.
 Notation " -. x " :=(Neg x) (at level 80) : pretxtnot.
 Delimit Scope pretxtnot with etd.
End PredFormulasNotationsASCII.
```

Определение 5.15. Переменная ξ – параметр формулы f, если ξ встречается в f вне области действия кванторов по ξ.

```
Fixpoint isParamF (xi : SetVars.t) (f : Fo) {struct f} : bool :=
   match f with
   | Atom p t0 => Vector.fold_left orb false (Vector.map (isParamT xi) t0)
   | Bot => false
   | Conj f1 f2 | Disj f1 f2 | Impl f1 f2 => isParamF xi f1 || isParamF xi f2
   | Fora x f0 | Exis x f0 =>
       if SetVars.eqb x xi then false else isParamF xi f0
   end.
```

Определение 5.16. Корректная подстановка терма t вместо переменной ξ в формуле ϕ есть формула, в которой все свободные вхождения переменной ξ заменены на t, так что при этом никакие переменные из t не попали под действие одноимённых кванторов.

Определение 5.17. Другими словами, корректная подстановка терма t вместо переменной ξ в формуле – есть частично определённая функция замены некоторых ξ на t, определённая, если

1. заменяются только те вхождения ξ, которые не связаны квантором и
2. ни одно полученное подстановкой новое вхождение свободных переменной из t не связывается никаким квантором.

Функция substF возвращает результат t' корректной подстановки в виде терма Some t', а в случае некорректной подстановки принимает выделенное значение None.

```
(* Substitution *)
Fixpoint substF (t : Terms) (xi : SetVars.t) (u : Fo)
 {struct u} : option Fo
 :=let f := (substF t xi) in
   match u with
   | Atom p t0 => Some (Atom p (map (substT t xi) t0))
   | Bot => Some Bot
   | Conj u1 u2 =>
     match (f u1),(f u2) with
     | Some f0,Some f1 => (Some (Conj f0 f1))
     | _,_ => None
     end
   | Disj u1 u2 =>
     match (f u1),(f u2) with
     | Some f0,Some f1 => (Some (Disj f0 f1))
     | _,_ => None
     end
   | Impl u1 u2 =>
     match (f u1),(f u2) with
     | Some f0,Some f1 => (Some (Impl f0 f1))
     | _,_ => None
     end
   | Fora x psi =>
       if isParamF xi (Fora x psi)
       then
        if isParamT x t
```

```
      then None
      else
       match f psi with
       | Some q => Some (Fora x q)
       | None => None
       end
     else Some (Fora x psi)
  | Exis x psi =>
     if isParamF xi (Exis x psi)
     then
      if isParamT x t
      then None
      else
       match f psi with
       | Some q => Some (Exis x q)
       | None => None
       end
     else Some (Exis x psi)
  end.
```

Снова рассмотрим некоторую интерпретацию функциональных и предикатных символов

```
Section Interpretation2.
Context {X} {fsI:forall(q:FSV),(Vector.t X (fsv q))->X}.
Context {prI:forall(q:PSV),(Vector.t X (psv q))->Omega}.
```

и зададим семантику языка первого порядка:

- атомарной формуле сопоставим значения интепретации предикатного символа на значениях интерпретаций приписанных к нему термов;
- импликации сопоставим тип функций из Prop в Prop;
- дизъюнкции сопоставим несвязную сумму двух типов из Prop;
- конъюнкции сопоставим бинарное произведение типов из Prop;
- квантору существования сопоставим зависимую сумму домена X и семейства типов из Prop;
- квантору всеобщности сопоставим зависимое произведение домена X и семейства типов из Prop.

Определение 5.18. (foI X fsI prI val f) – пропозиция, являющаяся значением формулы f при оценке val в текущей интерпретации домена – X, функциональных символов – fsI и предикатных символов – prI.

```
Fixpoint foI (val : SetVars.t -> X) (f : Fo) {struct f} : Omega :=
  match f with
  | Atom p t0 => prI p (map (@teI _ fsI val) t0)
  | Bot => OFalse
  | Conj f1 f2 => OAnd (foI val f1) (foI val f2)
  | Disj f1 f2 => OOr (foI val f1) (foI val f2)
  | Impl f1 f2 => OImp (foI val f1) (foI val f2)
  | Fora x f0 => forall m : X, foI (cng val x m) f0
  | Exis x f0 => exists m : X, foI (cng val x m) f0
  end.
End Interpretation2.
```

5.9 Предикат доказуемости.

В данном параграфе описывается предикат выводимости из контекста для интуиционисткого исчисления высказываний(ИИВ).

```
Import PredFormulasNotationsASCII.
Local Open Scope pretxtnot.
```

Выделение аксиом среди формул разделёно на две части – пропозициональную и специфическую для логики предикатов. Предикаты PROCA и PRECA выделяют среди формул те, которые являются примерами схем аксиом пропозициональной логики и логики предикатов, соответственно(PRECA содержит PROCA плюс аксиомы Бернайса).

```
(* intuitionistic "PROpositional" Calculus Axioms *)
Inductive PROCA : Fo -> Type :=
| Ha1 : forall A B, PROCA (A-->(B-->A))
| Ha2 : forall A B C, PROCA ((A-->(B-->C))-->((A-->B)-->(A-->C)))
| Ha3 : forall A B, PROCA (Conj A B --> A)
| Ha4 : forall A B, PROCA (Conj A B --> B)
| Ha5 : forall A B, PROCA (A --> (B --> Conj A B))
| Ha6 : forall A B, PROCA (A --> Disj A B)
| Ha7 : forall A B, PROCA (B --> Disj A B)
| Ha8 : forall A B C, PROCA ((A --> C) --> ((B --> C) --> (Disj A B --> C)))
| Ha9 : forall A B, PROCA (Neg A --> (A --> B))
| Ha10 : forall A B, PROCA ((A --> B) --> ((A --> Neg B) --> Neg A))
.
(* intuitionistic PREdicate Calculus Axioms *)
Inductive PRECA : Fo -> Type :=
| PRO :> forall A, (PROCA A) -> (PRECA A)
| Ha12 : forall (ph: Fo) (t:Terms) (xi:SetVars.t)
 (r:Fo) (s:(substF t xi ph)=Some r), PRECA ((Fora xi ph) --> r)
| Ha13 : forall (ph: Fo) (t:Terms) (xi:SetVars.t)
 (r:Fo) (s:(substF t xi ph)=Some r), PRECA (r --> (Exis xi ph) )
| Hb1 : forall (ps ph: Fo) (xi:SetVars.t) (H:isParamF xi ps = false),
PRECA (Impl (Fora xi (Impl ps ph)) (Impl ps (Fora xi ph)) )
| Hb2 : forall (ps ph: Fo) (xi:SetVars.t) (H:isParamF xi ps = false),
PRECA (Impl (Fora xi (Impl ph ps)) (Impl (Exis xi ph) ps ) )
.
```

Далее описан предикат выводимости из посылок PREPR. В данной реализации использована выводимость из списка посылок, что несколько ограничивает возможности доказательства только конечными контекстами(например, при пополнении контекстов до максимальных в лемме Линденбаума), однако в дальнейшем возможно обобщить последующие доказательства про выводимость и до произвольных контекстов также, как это было сделано в первой части работы для исчисления высказываний.

```
Definition NotParamC xi ctx :=
 forall F : Fo, InL F ctx -> isParamF xi F = false.

Section PREPR.
Context (ctx:list Fo).
Inductive PREPR : Fo -> Type :=
| hyp_E (A : Fo) : (InL A ctx) -> (PREPR A)
| Hax_E :> forall (A : Fo), (PRECA A) -> (PREPR A)
| MP_E (A B: Fo) : (PREPR A) -> (PREPR (Impl A B)) -> (PREPR B)
| GEN_E (A: Fo) (xi :SetVars.t) (nic:NotParamC xi ctx)
  : (PREPR A) -> (PREPR (Fora xi A))
.
End PREPR.
```

Далее приведём примеры тривиальных доказательств. Верны следующие утверждения про интуиционистскую логику предикатов:

Пример 5.16. Для любых формул A и B в ИИВ выводимо $(A \to (B \to A))$.

```
Definition a1 ctx A B : @PREPR ctx (Impl A (Impl B A)).
Proof. apply Hax_E, PRO, Ha1. Defined.
```

Пример 5.17. Для любых формул A, B и C в ИИВ выводимо $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$.

```
Definition a2 axi A B C : @PREPR axi ((A-->(B-->C))-->((A-->B)-->(A-->C))).
Proof. apply Hax_E, PRO, Ha2. Defined.
```

Пример 5.18. Для любых формул A и B в ИИВ выводимо $\forall x(A \to B) \to (A \to \forall xB)$ если $x \notin \mathrm{FV}(A)$.

```
Definition b1 axi (ps ph: Fo) (xi:SetVars.t) (H:isParamF xi ps = false):
@PREPR axi (Impl (Fora xi (Impl ps ph)) (Impl ps (Fora xi ph)) ).
Proof. apply Hax_E, Hb1, H. Defined.
```

Пример 5.19. Для любой формулы A в ИИВ выводимо $(A \to A)$.

```
Definition AtoA {ctx} (A:Fo) : PREPR ctx (A-->A).
Proof.
apply MP_E with (A:=(A-->(A-->A))).
apply Hax_E, PRO, Ha1.
apply MP_E with (A:= A-->((A-->A)-->A)).
apply Hax_E, PRO, Ha1.
apply Hax_E, PRO, Ha2.
Defined.

End Provability_mod.
```

5.10 Теорема о дедукции

Подгружаем нотацию и модуль BoolEqualityFact, содержащий теоремы про разрешимое равенство.

```
Notation SetVars := SetVars.t (only parsing).
Notation FuncSymb := FuncSymb.t (only parsing).
Notation PredSymb := PredSymb.t (only parsing).
Module Facts := BoolEqualityFacts SetVars.
```

Пример 5.20. Вывод первого правила Бернайса из первой аксиомы Бернайса.

```
(* Example: Bernays Rule is admissable *)
Definition B1 (ps ph:Fo) (xi:SetVars) ctx
(H:isParamF xi ps = false)
(T:NotParamC xi ctx) :
 PREPR ctx (ps ---> ph) -> PREPR ctx (ps ---> Fora xi ph).
Proof.
intro q.
apply MP_E with (A:=(Fora xi (ps ---> ph))).
+ apply (GEN_E).
  exact T.
  exact q.
+ apply (b1 _).
  exact H.
Defined.
```

Лемма 5.21. $\dfrac{\Gamma \vdash A}{\Gamma \vdash (B \to A)}$ – допустимое правило в интуиционистком исчсилении высказываний с контекстами.

```
Definition a1i (A B : Fo)(l : list Fo):(PREPR l B)->(PREPR l (Impl A B)).
Proof.
intros x.
apply MP_E with (A:= B).
exact x.
apply a1.
Defined.
```

Пример 5.22. Обратно, доказательство правила генерализации через правило Бернайса.

```
(* Example: Generalization from 1st Bernay's rule*)
Definition gen (A:Fo) (xi:SetVars) ctx
 (T:NotParamC xi ctx)
 : PREPR ctx (A) -> PREPR ctx (Fora xi A).
Proof.
intro q.
apply MP_E with (A:= Top).
unfold Top.
exact (@AtoA ctx Bot).
apply B1.
+ trivial.
+ exact T.
+ apply a1i.
  exact q.
Defined.
```

Теорема 5.23. (о дедукции) Если выводима секвенция $\Gamma, A \vdash B$, то выводима секвенция $\Gamma \vdash A \to B$.

```
(* Deduction theorem *)
Fixpoint Ded (A B:Fo)(il:list Fo)(m:(PREPR (cons A il) B))
{struct m}:(PREPR il (A-->B)).
Proof.
destruct m.
+ simpl in i.
```

```
  destruct i.
  − rewrite <− e.
    exact (AtoA A).
  − apply a1i.
    apply hyp_E with (ctx:=il) (1:=i).
+ apply a1i.
  apply Hax_E, p.
+ apply MP_E with (A:= (A−−>A0)).
  − simple refine (@Ded _ _ _ _).
    1 : exact m1.
  − apply MP_E with (A:= (A−−>(A0−−>B))).
    * simple refine (@Ded _ _ _ _).
      exact m2.
    * apply a2.
+ (*Last part about GEN*)
  apply MP_E with (A:= (Fora xi (A−−>A0))).
  − eapply GEN_E.
    { intros A1 M. apply nic. right. exact M. }
    simple refine (@Ded _ _ _ _ ).
    * exact m.
  − simpl.
    eapply Hax_E.
    eapply Hb1.
    apply nic. left. trivial.
Defined.
```

Теорема о дедукции позволяет упрощать доказательство теорем вроде следующей.

Следствие 5.23.1. При любом контексте Γ выводима данная секвенция:

$$\Gamma \vdash \big((A \to (B \to C)) \to (B \to (A \to C))\big).$$

```
Definition swap_args ctx A B C :
(PREPR ctx ((A −−> (B −−> C)) −−> (B −−> (A −−> C)) )).
Proof.
unshelve eapply Ded.
unshelve eapply Ded.
unshelve eapply Ded.
apply MP_E with (A:=B) .
+ apply hyp_E.
  simpl. firstorder.
+ apply MP_E with (A:=A) .
  apply hyp_E; firstorder.
  apply hyp_E; firstorder.
Defined.
```

5.11 Теорема о корректности

В заключительном разделе этой главы будет проведена формальная верификация доказательства теоремы о корректности для интуиционисткого исчисления высказываний с контекстами.

Теорема 5.24. О корректности Для любого домена и интепретации функциональных и предикатных символов на нём, всякая выводимая в логике предикатов из контекста l формула f истинна при некоторой оценке π тогда, когда при этой же оценке истинны все формулы из контекста l.

В секции cor зададим интерпретацию функциональных и предикатных символов и продолжим её на все формулы стандартным образом.

```
(** Soundness theorem section **)
Section cor.
Context (X:Type).
Context (fsI:forall(q:FSV),(Vector.t X (fsv q))−>X).
Context (prI:forall(q:PSV),(Vector.t X (psv q))−>Omega).
```

Лемма 5.25. Интерпретация результата действия подстановки на терм равна изменённой интерпретации исходного терма.

```
Section Lem1.
(* page 136 of the book *)
Definition lem1 (t : Terms) : forall (u :Terms)
(xi : SetVars.t) (pi : SetVars.t->X) ,
(@teI X fsI pi (substT t xi u))=(@teI X fsI (cng pi xi (@teI X fsI pi t)) u).
Proof.
fix lem1 1.
intros.
induction u as [s|f].
+ simpl.
  unfold cng.
  destruct (SetVars.eqb s xi) eqn:ek.
  * reflexivity.
  * simpl.
    reflexivity.
+ simpl.
  destruct f.
  simpl.
  apply ap.
  simpl in * |- *.
  apply (proj1 (
   eq_nth_iff X fsv0
   (Vector.map (teI pi) (Vector.map (substT t xi) v))
   (Vector.map (teI (cng pi xi (teI pi t))) v)
  )).
  intros.
  simpl in * |- *.
  rewrite -> (nth_map (teI pi) (Vector.map (substT t xi) v) p1 p2 H0).
  rewrite -> (nth_map (teI (cng pi xi (teI pi t))) v p2 p2 ).
  rewrite -> (nth_map (substT t xi) v p2 p2 eq_refl).
  exact (H p2).
  reflexivity.
Defined.
End Lem1.
```

Лемма 5.26. all_then_someP – если дизъюнкция значений некоторого предиката на элементах вектора равна лжи, то предикат равен лжи на каждом элементе вектора.

```
Lemma all_then_someP (A:Type)(n:nat)(p:Fin.t (n)) (v:Vector.t A (n)) (P:A->bool)
(H : Vector.fold_left orb false (Vector.map P v) = false)
 : (P (Vector.nth v p)) = false.
Proof.
rewrite <- (nth_map P v p p eq_refl).
apply all_then_someV. trivial.
Defined.
```

Лемма 5.27. NPthenNCAST – если терм не содержит некоторой переменной, то результат действия подстановки в эту переменную не изменяет терм.

```
(* Not a parameter then not changed afted substitution (for Terms) *)
Lemma NPthenNCAST (u:Terms)(xi:SetVars.t)(t:Terms) (H:(isParamT xi u=false))
: (substT t xi u) = u.
Proof. induction u.
+ simpl in * |- *.
  rewrite H. reflexivity.
+ simpl in * |- *.
  apply ap.
  apply eq_nth_iff.
  intros p1 p2 ppe.
  rewrite (nth_map _ _ _ p2 ppe).
  apply H0.
  simpl.
```

```
  apply all_then_someP.
  trivial.
Defined.
```

Лемма 5.28. Если ξ – не параметр атомарной формулы, то вектор её термов подстановка не меняет.

```
Lemma NPthenNCAST_vec:forall p xi t ts (H:(isParamF xi (Atom p ts)=false)),
  (Vector.map (substT t xi) ts) = ts.
Proof.
intros p xi t1 ts H.
apply eq_nth_iff.
intros p1 p2 H0.
rewrite -> (nth_map (substT t1 xi) ts p1 p2 H0).
apply NPthenNCAST.
apply all_then_someP.
simpl in H.
exact H.
Defined.
```

Лемма 5.29. Подстановка терма вместо переменной ξ не изменяет на формулы без параметра ξ.

```
(* Not a parameter then not changed afted substitution (for Formulas) *)
Fixpoint NPthenNCASF (mu:Fo) : forall (xi:SetVars.t)(t:Terms) (H:(isParamF xi mu=false))
   , substF t xi mu = Some mu .
Proof. (*induction mu eqn:u0.*)
destruct mu eqn:u0 ; simpl; intros xi t0 H.
* rewrite -> NPthenNCAST_vec; (trivial || assumption).
* trivial.
* simpl in H.
  apply orb_false_elim in H as [H0 H1].
  rewrite -> NPthenNCASF .
  rewrite -> NPthenNCASF .
  trivial.
  trivial.
  trivial.
* apply orb_false_elim in H as [H0 H1].
  rewrite -> NPthenNCASF.
  rewrite -> NPthenNCASF.
  trivial.
  trivial.
  trivial.
* apply orb_false_elim in H as [H0 H1].
  rewrite -> NPthenNCASF.
  rewrite -> NPthenNCASF.
  trivial.
  trivial.
  trivial.
* destruct (SetVars.eqb x xi) eqn:u2.
  trivial.
  destruct (isParamF xi f) eqn:u1.
  inversion H.
  trivial.
* destruct (SetVars.eqb x xi) eqn:u2.
  trivial.
  destruct (isParamF xi f) eqn:u1.
  inversion H.
  trivial.
Defined.
```

Лемма 5.30. weafunT – если оценки индивидных переменных совпадают поточечно, то интерпретации некоторого выбранного терма при этих оценках равны. ξ

```
(* p.137 *)
Section Lem2.
```

```
Lemma weafunT pi mu (q: forall z, pi z = mu z) t :
@teI X fsI pi t = @teI X fsI mu t.
Proof.
induction t.
+ simpl. exact (q sv).
+ simpl. apply ap.
  apply eq_nth_iff.
  intros p1 p2 HU.
  rewrite -> (nth_map (teI pi) v p1 p2 HU).
  rewrite -> (nth_map (teI mu) v p2 p2 eq_refl).
  apply H.
Defined.
```

Лемма 5.31. weafunF – если оценки индивидных переменных совпадают поточечно, то интерпретации некоторой выбранной формулы при этих оценках эквивалентны.

```
Lemma weafunF (pi mu:SetVars.t->X) (q: forall z, pi z = mu z) fi :
 @foI X fsI prI pi fi <-> @foI X fsI prI mu fi.
Proof.
revert pi mu q.
induction fi.
intros pi mu q.
+ simpl.
  apply EqualThenEquiv.
  apply ap.
  apply eq_nth_iff.
  intros p1 p2 HU.
  rewrite -> (nth_map (teI pi) t p1 p2 HU).
  rewrite -> (nth_map (teI mu) t p2 p2 eq_refl).
  apply weafunT.
  apply q.
+ simpl. reflexivity.
+ simpl. intros.
  rewrite -> (IHfi1 pi mu q).
  rewrite -> (IHfi2 pi mu q).
  reflexivity.
+ simpl. intros.
  rewrite -> (IHfi1 pi mu q).
  rewrite -> (IHfi2 pi mu q).
  reflexivity.
+ simpl.
  unfold OImp.
  split;
  intros;
  apply (IHfi2 pi mu q);
  apply H;
  apply (IHfi1 pi mu q);
  apply H0.
+ simpl.
  split.
  * intros.
    rewrite IHfi.
    apply H with (m:=m).
    intro z.
    unfold cng.
    destruct (SetVars.eqb z x).
    reflexivity.
    symmetry.
    apply q.
  * intros.
    rewrite IHfi.
    apply H.
    intro z.
    unfold cng.
    destruct (SetVars.eqb z x).
    reflexivity.
```

```
    apply q.
+ simpl.
  split.
  * intros.
    destruct H as [m H].
    exists m.
    rewrite IHfi.
    apply H.
    intro z.
    unfold cng.
    destruct (SetVars.eqb z x).
    reflexivity.
    symmetry.
    apply q.
  * intros.
    destruct H as [m H].
    exists m.
    rewrite IHfi.
    apply H.
    intro z.
    unfold cng.
    destruct (SetVars.eqb z x).
    reflexivity.
    apply q.
Defined.
```

Лемма 5.32. Если заменяются значения различных переменных у оценки, то оценка формулы, полученная в результате, не зависит от очерёдности замен.

```
Lemma cng_commF_EQV xe xi m0 m1 pi fi :
SetVars.eqb xe xi = false ->
(@foI X fsI prI (cng (cng pi xe m0) xi m1) fi <-> @foI X fsI prI (cng (cng pi xi m1) xe m0) fi).
Proof.
intros H.
apply weafunF.
intros z.
unfold cng.
destruct (SetVars.eqb z xi) eqn:e0, (SetVars.eqb z xe) eqn:e1.
pose (U0:= proj1 (SetVars.eqb_eq z xi) e0).
rewrite U0 in e1.
pose (U1:= proj1 (SetVars.eqb_eq xi xe) e1).
symmetry in U1.
pose (U2:= proj2 (SetVars.eqb_eq xe xi) U1).
rewrite U2 in H.
inversion H.
reflexivity. reflexivity. reflexivity.
Defined.
```

Лемма 5.33. Интерпретация результата действия корректной подстановки в атомарной формуле эквивалентна интерпретации этой формулы при изменённой оценке.

```
Lemma lem2caseAtom : forall (p : PSV) (t0 : Vector.t Terms (psv p))
(t : Terms) (xi : SetVars.t) (pi : SetVars.t->X)
(r:Fo) (H:(substF t xi (Atom p t0)) = Some r) ,
@foI X fsI prI pi r <-> @foI X fsI prI (cng pi xi (@teI X fsI pi t)) (Atom p t0).
Proof.
intros.
+ simpl in H.
  assert (Q:=SomeInj _ _ H).
  rewrite <- Q.
  simpl.
  apply EqualThenEquiv.
  apply ap.
  apply
    (proj1 (
```

```
      eq_nth_iff X (psv p)
      (Vector.map (teI pi) (Vector.map (substT t xi) t0))
      (Vector.map (teI (cng pi xi (teI pi t))) t0)
    )).
  rename t0 into v.
  intros p1 p2 H0.
  rewrite -> (nth_map (teI pi) (Vector.map (substT t xi) v) p1 p2 H0).
  rewrite -> (nth_map (teI (cng pi xi (teI pi t))) v p2 p2 ).
  rewrite -> (nth_map (substT t xi) v p2 p2 eq_refl).
  apply lem1. reflexivity.
Defined.
```

Лемма 5.34. На интерпретацию терма не влияет изменение оценки переменной, не входящей в терм.

```
Lemma NPthenNCACVT x t m pi :
 isParamT x t = false ->
 (@teI X fsI (cng pi x m) t) = (@teI X fsI pi t).
Proof.
intros H.
induction t.
unfold cng.
simpl in * |- *.
rewrite H.
reflexivity.
simpl in * |- *.
apply ap.
apply eq_nth_iff.
intros.
rewrite -> (nth_map (teI (cng pi x m)) v p1 p2 H1).
rewrite -> (nth_map (teI pi) v p2 p2 eq_refl).
apply H0.
apply (all_then_someP Terms (fsv f) p2 v (isParamT x) H).
Defined.
```

Лемма 5.35. Разрешимое равенство на множестве переменных – симметричное отношение.

```
Lemma eqb_comm x xi : SetVars.eqb xi x = SetVars.eqb x xi.
Proof.
destruct (SetVars.eqb xi x) eqn:e1.
symmetry.
assert (Y:= proj1 (SetVars.eqb_eq xi x) e1).
rewrite -> Y at 1.
rewrite <- Y at 1.
exact e1.
symmetry.
assert (n3:= proj2 (not_iff_compat (SetVars.eqb_eq x xi)) ).
apply not_true_iff_false.
apply n3.
intro q.
symmetry in q.
revert q.
fold (xi <> x).
assert (n5:= proj1 (not_iff_compat (SetVars.eqb_eq xi x)) ).
apply n5.
apply not_true_iff_false.
exact e1.
Defined.
```

Лемма 5.36. NPthenNCACVF – на интерпретацию формулы не влияет изменение оценки индивидной переменной, не входящей в формулу.

```
Lemma NPthenNCACVF xi fi m mu : isParamF xi fi = false ->
 @foI X fsI prI (cng mu xi m) fi <-> @foI X fsI prI mu fi.
Proof.
```

```
revert mu.
induction fi; intro mu;
intro H;
simpl in * |- *.
* apply EqualThenEquiv.
  apply ap.
  apply eq_nth_iff.
  intros p1 p2 H0.
  rewrite -> (nth_map (teI (cng mu xi m)) t p1 p2 H0).
  rewrite -> (nth_map (teI mu) t p2 p2 eq_refl).
  apply NPthenNCACVT.
  apply (all_then_someP Terms (psv p) p2 t (isParamT xi) H).
  (*1st done *)
* firstorder.
* apply AND_EQV.
  apply IHfi1. destruct (orb_false_elim _ _ H). apply H0.
  apply IHfi2. destruct (orb_false_elim _ _ H). apply H1.
* apply OR_EQV.
  apply IHfi1. destruct (orb_false_elim _ _ H). apply H0.
  apply IHfi2. destruct (orb_false_elim _ _ H). apply H1.
* apply IMP_EQV.
  apply IHfi1. destruct (orb_false_elim _ _ H). apply H0.
  apply IHfi2. destruct (orb_false_elim _ _ H). apply H1.
* apply FORALL_EQV. intro m0.
  destruct (SetVars.eqb x xi) eqn:e1.
  assert (C:=proj1 (SetVars.eqb_eq x xi) e1).
  rewrite <- C.
  pose (D:= dbl_cng mu x m m0).
  exact (weafunF _ _ D fi).
  rewrite cng_commF_EQV.
  (* IHfi is an inductive hypotheis *)
  apply IHfi.
  exact H.
  rewrite <-(eqb_comm xi x).
  exact e1.
* apply EXISTS_EQV. intro m0.
  fold (cng (cng mu xi m) x m0).
  fold (cng mu x m0).
  destruct (SetVars.eqb x xi) eqn:e1.
  assert (C:=proj1 (SetVars.eqb_eq x xi) e1).
  rewrite <- C.
  assert (D:= dbl_cng mu x m m0).
  exact (weafunF _ _ D fi).
  rewrite cng_commF_EQV.
  (* IHfi is an inductive hypothesis*)
  apply IHfi.
  exact H.
  rewrite <-(eqb_comm xi x).
  exact e1.
Defined.
```

Лемма 5.37. lem2 – интерпертация резульата действия корректной подстановки эквивалентна интерпретации исходной формулы при изменённой оценке.

```
Definition lem2 (t : Terms) : forall (fi : Fo) (xi : SetVars.t) (pi : SetVars.t->X)
 (r:Fo) (H:(substF t xi fi) = Some r),
 (@foI X fsI prI pi r)<->(@foI X fsI prI (cng pi xi (@teI X fsI pi t)) fi).
Proof.
fix lem2 1.
(*H depends on t xi fi r *)
intros fi xi pi r H.
revert pi r H.
induction fi;
intros pi r H.
+ apply lem2caseAtom.
  exact H.
```

```
+ inversion H. simpl. reflexivity.
+ simpl in *|-*.
  destruct (substF t xi fi1) as [f1|].
  destruct (substF t xi fi2) as [f2|].
  assert (Q:=SomeInj _ _ H).
  rewrite <- Q.
  simpl.
  unfold OAnd.
  apply AND_EQV.
  simpl in * |- *.
  * apply (IHfi1 pi f1 eq_refl).
  * apply (IHfi2 pi f2 eq_refl).
  * inversion H.
  * inversion H.
+ simpl in *|-*.
  destruct (substF t xi fi1) as [f1|].
  destruct (substF t xi fi2) as [f2|].
  assert (Q:=SomeInj _ _ H).
  rewrite <- Q.
  simpl in * |- *.
  apply OR_EQV.
  * apply (IHfi1 pi f1 eq_refl).
  * apply (IHfi2 pi f2 eq_refl).
  * inversion H.
  * inversion H.
+ simpl in *|-*.
  destruct (substF t xi fi1) as [f1|].
  destruct (substF t xi fi2) as [f2|].
  assert (Q:=SomeInj _ _ H).
  rewrite <- Q.
  simpl in * |- *.
  apply IMP_EQV.
  * apply (IHfi1 pi f1 eq_refl).
  * apply (IHfi2 pi f2 eq_refl).
  * inversion H.
  * inversion H.
+ simpl in * |- *.
  destruct (SetVars.eqb x xi) eqn:l2.
  assert (Q:=SomeInj _ _ H).
  rewrite <- Q.
  simpl.
  apply FORALL_EQV.
  intro m.
  assert (RA : x = xi).
  apply (SetVars.eqb_eq x xi ).
  exact l2.
  rewrite <- RA.
  rewrite -> (weafunF (cng (cng pi x (teI pi t)) x m) (cng pi x m)
   (dbl_cng pi x (teI pi t) m) fi).
  firstorder.
  destruct (isParamF xi fi) eqn:l1.
  destruct (isParamT x t) eqn:l3.
  inversion H.
  destruct (substF t xi fi) eqn:l4.
  assert (Q:=SomeInj _ _ H).
  rewrite <- Q.
  simpl.
  apply FORALL_EQV.
  intro m.
  rewrite cng_commF_EQV.
  2 : {
    rewrite -> eqb_comm .
    exact l2.
  }
  rewrite <- (NPthenNCACVT x t m pi l3).
  exact (IHfi (cng pi x m) f eq_refl).
```

```
  inversion H.
  pose (Q:=SomeInj _ _ H).
  rewrite <- Q.
  simpl.
  apply FORALL_EQV.
  intro m.
  rewrite cng_commF_EQV.
  symmetry.
  exact (NPthenNCACVF xi fi (teI pi t) (cng pi x m) l1).
  rewrite -> (eqb_comm x xi).
  exact l2. (* end of FORALL case*)
+ simpl in * |- *.
  destruct (SetVars.eqb x xi) eqn:l2.
  pose (Q:=SomeInj _ _ H).
  rewrite <- Q.
  simpl.
  apply EXISTS_EQV.
  intro m.
  assert (RA : x = xi).
  apply (SetVars.eqb_eq x xi ).
  exact l2.
  rewrite <- RA.
  rewrite -> (weafunF (cng (cng pi x (@teI X fsI pi t)) x m) (cng pi x m)
   (dbl_cng pi x (@teI X fsI pi t) m) fi).
  firstorder.
  destruct (isParamF xi fi) eqn:l1.
  pose(xint := (isParamT x t)).
  destruct (isParamT x t) eqn:l3.
  inversion H.
  destruct (substF t xi fi) eqn:l4.
  pose (Q:=SomeInj _ _ H).
  rewrite <- Q.
  simpl.
  apply EXISTS_EQV.
  intro m.
  fold (cng pi x m ).
  fold (cng (cng pi xi (@teI X fsI pi t)) x m ).
  rewrite cng_commF_EQV.
  2 : {
    rewrite -> eqb_comm .
    exact l2.
  }
  rewrite <- (NPthenNCACVT x t m pi l3).
  exact (IHfi (cng pi x m) f eq_refl).
  inversion H.
  pose (Q:=SomeInj _ _ H).
  rewrite <- Q.
  simpl.
  apply EXISTS_EQV.
  intro m.
  fold (cng pi x m ).
  fold (cng (cng pi xi (@teI X fsI pi t)) x m ).
  rewrite cng_commF_EQV.
  symmetry.
  exact (NPthenNCACVF xi fi (teI pi t) (cng pi x m) l1).
  rewrite -> (eqb_comm x xi).
  exact l2.
Defined. (* END OF LEM2 *)
End Lem2.
```

Лемма 5.38. Истинна любая интерпретация любой формулы вида $(\forall x.\phi) \to \phi(t/x)$, если входящая в неё подстановка - корректна.

```
Lemma UnivInst : forall (fi:Fo) (pi:SetVars.t->X) (x:SetVars.t) (t:Terms)
(r:Fo) (H:(substF t x fi)=Some r), @foI X fsI prI pi (Impl (Fora x fi) r).
Proof.
```

```
intros fi pi x t r H.
simpl.
intro H0.
apply (lem2 t fi x pi r H).
apply H0.
Defined.
```

Лемма 5.39. Истинна любая интерпретация любой формулы вида $\phi(t/x) \to (\exists x.\phi)$, если входящая в неё подстановка – корректна.

```
Lemma ExisGene : forall (fi:Fo) (pi:SetVars.t->X) (x:SetVars.t) (t:Terms)
(r:Fo) (H:(substF t x fi)=Some r), @foI X fsI prI pi (Impl r (Exis x fi)).
Proof.
intros fi pi x t r H.
simpl.
intro H0.
exists (@teI X fsI pi t).
fold (cng pi x (@teI X fsI pi t)).
apply (lem2 t fi x pi r H).
apply H0.
Defined.
```

Теорема 5.40. soundness – для любой интепретации всякая формула f выводимая в логике предикатов из контекста l истинна тогда, когда истинны все формулы из контекста l.

```
(* SOUNDNESS OF THE PREDICATE CALCULUS *)
Theorem soundness (f:Fo) (l:list Fo) (m : PREPR l f) :
 forall (val:SetVars.t->X),
 (forall h:Fo, (InL h l) -> (@foI X fsI prI val h)) ->
  @foI X fsI prI val f.
Proof.
induction m; intros val lfi.
+ exact (lfi A i).
+ destruct p eqn:k.
  ++ destruct p0.
     * simpl.
       intros a0 b.
       exact a0.
     * simpl.
       intros a0 b c.
       exact (a0 c (b c)).
     * simpl. intros [i0 i1]. assumption.
     * simpl. intros [i0 i1]. assumption.
     * simpl. intros m1 m2. split; assumption.
     * simpl. intros n. left. assumption.
     * simpl. intros n. right. assumption.
     * simpl. intros f1 f2 [h|h]. exact (f1 h). exact (f2 h).
     * simpl. intros i0 i1. destruct (i0 i1).
     * simpl. intros i0 i1 i2. apply (i1 i2). apply (i0 i2).
  ++ simpl in *|-*.
     apply (UnivInst ph val xi t r s).
  ++ simpl in *|-*.
     apply (ExisGene ph val xi t r s).
  ++ simpl in *|-*.
     unfold OImp.
     intros H0 H1 m.
     apply H0.
     rewrite -> (NPthenNCACVF xi ps0 m val H).
     exact H1.
  ++ simpl in *|-*.
     unfold OImp.
     intros H0 [m H1].
     rewrite <- (NPthenNCACVF xi ps0 m val H).
     eapply H0.
     exact H1.
```

```
+ simpl in * |- *.
  unfold OImp in IHm2.
  apply IHm2.
  apply lfi.
  apply IHm1.
  apply lfi.
+ simpl in * |- *.
  intro m0.
  eapply IHm. (* IHm is a (soundness A l) *)
  intros h J.
  apply <- NPthenNCACVF.
  2 : { apply nic. exact J. }
  apply lfi. exact J.
Defined.

End cor.
```

6 Модель теории множеств в Coq

Теория множеств – раздел математики, формализующий интуитивное понятие совокупности объектов и принадлежности объекта некоторой совокупности.

6.1 Теория множеств Цермело-Френкеля с аксиомой выбора.

Определение 6.1. Теория множеств цермело Френкеля(ZFC) есть первопорядковая теория со следующим наборов схем аксиом:

Таблица 6: Нелогические схемы аксиомы ZFC

название	утверждение	условие
Eq1($\in$)	$\forall x_1 \forall x_2 \forall y \big(x_1 = x_2 \to (x_1 \in y \to x_2 \in y)\big)$	-
Eq2($\in$)	$\forall x_1 \forall x_2 \forall y \big(x_1 = x_2 \to (y \in x_1 \to y \in x_2)\big)$	-
axExt	$\forall a_1 \forall a_2 \big(\forall b(b \in a_1 \leftrightarrow b \in a_2) \to a_1 = a_2\big)$	-
axPair	$\forall a_1 \forall a_2 \exists c \forall b \big(b \in c \leftrightarrow (b = a_1 \vee b = a_2)\big)$	-
axUnion	$\forall a \exists d \forall b \big(b \in d \leftrightarrow \exists c(b \in c \wedge c \in a)\big)$	-
axPow	$\forall a \exists d \forall b \big(b \in d \leftrightarrow \forall c(c \in b \to c \in a)\big)$	-
axInf	$\exists a \big(\varnothing \in a \wedge \forall b(b \in a \to b \cup \{b\} \in a)\big)$	-
axSep	$\forall w_1, \ldots, w_n \forall A \exists B \forall x (x \in B \Leftrightarrow [x \in A \wedge \varphi(x, w_1, \ldots, w_n, A)])$	$B \notin \mathrm{FV}(\varphi)$
axRepl	$\forall x \exists! y \varphi \to \forall a \exists d \forall c \big(c \in d \leftrightarrow \exists b(b \in a \wedge \varphi(b/x)(c/y))\big)$	$\mathrm{FV}(\varphi) \subseteq \{x, y\}$ $\mathrm{FFI}(b, x, \varphi)$ $\mathrm{FFI}(c, y, \varphi)$
axChoice	$\forall a \Big(a \neq \varnothing \wedge \forall b(b \in a \to b \neq \varnothing) \wedge$ $\wedge \forall b_1 \forall b_2 (b_1 \neq b_2 \wedge \{b_1, b_2\} \subseteq a \to b_1 \cap b_2 = \varnothing) \to$ $\to \exists d \forall b \big(b \in a \to \exists c(b \cap d = \{c\})\big)\Big)$	-
axRegul	$\forall a \Big(a \neq \varnothing \to \exists b \big(b \in a \wedge \forall c(c \in b \to c \notin a)\big)\Big)$	-

Здесь $FV(\psi)$ обозначает множество свободных переменных формулы ψ, а $\mathrm{FFI}(b, x, \varphi)$ – высказывание "терм b свободен для подстановке вместо переменной x в формулу φ".

6.2 Подходы к вложению теорий множеств в Coq.

Существуют следующие варианты вложения множеств в CIC:

1. Библиотека Ensembles, как часть стандартной библиотеки Coq. В ней под "множествами" понимается предикаты на элементах некоторого типа. Однако тип может не быть множеством, например тип Ens из пункта 3.

2. Задавать множества аксиомами, как было сделано в проекте по верификации теории категорий в Coq `https://github.com/coq-contribs/cats-in-zfc/`.

```
Definition E := Type.
(* elements of a set are themselves sets *)
(***) Parameter R : forall x : E, x -> E.
(***) Axiom R_inj : forall (x : E) (a b : x), R a = R b -> a = b.
Definition inc (x y : E) := exists a : y, R a = x.
Definition sub (a b : E) := forall x : E, inc x a -> inc x b.
(* a set is determined by its elements *)
(***) Axiom extensionality : forall a b : E, sub a b -> sub b a -> a = b.
(* we also need extensionality for general product types *)
(***) Axiom
       prod_extensionality :
         forall (x : Type) (y : x -> Type) (u v : forall a : x, y a),
(forall a : x, u a = v a) -> u = v.
```

   При таком подходе необходимо доказать, что добавленные аксиомы не позволяют вывести противоречие. Опять же, тип E нельзя считать множествами, так как тип всех множеств Ens из пункта 3 – это терм типа E.

3. Индуктивный тип Ens, о котором пойдём речь в данной главе.

6.3 Основные теоретико-множественные определения и свойства.

Зададим Ens – тип, все термы которого соответствуют некоторым множества. "Ensemble" – "множество" по-французски. Данное соответствие – не инъекция, так как одно и тоже множество может быть задано разными термами. Также ясно, что данное соответствие – не сюръекция, так как термов – счётное число, а совокупность всех множеств – собственный класс. Тем не менее в дальнейшем термы типа Ens также будем называть множествами, при этом смысл будет однозначно восстанавливается из контекста.

Определение 6.2. Всякий терм типа Ens задаётся некоторым типом A и функцией типа A->Ens. Соответсенно назовём тип "базисным", а функцию – "опорной".

```
Inductive Ens : Type :=
    sup : forall A : Type, (A -> Ens) -> Ens.
```

Зависимая функция sup – это конструктор множеств, которая по заданному типу и функции из этого типа в тип множеств задаёт множество как образ этой функции. Индуктивное определение строит универсум множеств Ens по частям. Если какая-то часть уже построена и есть отображение f, которое вкладывает некоторый тип A в эту часть, то терм (sup A f) также объявляется множеством и добавляется в Ens. Теперь приведём пример.

Определение 6.3. Vide – терм типа Ens, соответствующий пустому множеству.

```
Definition Vide : Ens :=
  sup False (fun x : False => match x return Ens with
                              end).
```

Действительно, после определения принадлежности, можно доказать, что ни одно ни odby nthv nbgf Ens не принадлежит Vide.

Всякий тип в CIC – тоже в свою очередь имеет некоторый тип. Тип всех типов Type имеет некоторый тип. Если считать, что Type:Type, то выводится нетривальный парадокс Жирара[9]. Ключевое слово Type коде на языке Галлина – есть не более, чем нотация. Внутренний, скрытый от пользователя, механизм присваивает каждой вселенной типов натуральный индекс так, что $\text{Type}_i : \text{Type}_{i+1}$. Это позволяет не допустить парадокс парадокса Рассела в данной библиотеке. В Coq встроена проверка того, что во время аппликации терм, использующийся в качестве аргумента для функции из вселенной Type_i, содержится в меньшей вселенной типа Type_j, где $j < i$.

```
Definition A:=Type.
Check A.
Fail Check A:A.
(* The term "A" has type "Type@{Top.1+1}" while it is expected to have type "A"
    (universe inconsistency: Cannot enforce Top.1 < Top.1 because Top.1 = Top.1). *)
```

Именно поэтому “множество всех множеств”, т.е. терм “sup Ens (id Ens))”, невозможно построить:

```
Inductive Ens : Type :=
    sup : forall A : Type, (A -> Ens) -> Ens.

Fail Check sup Ens.
```

Данный код выдаёт ошибку «The term “Ens” has type “Type@Top.2+1” while it is expected to have type “Type@Top.2” (universe inconsistency)»

Более точно, элементы типа Ens – различные способы задания множеств. Этот тип – сетоид, т.е. тип с определённым на нём отношением эквивалентности. Отношением эквивалентности при этом вляется экстенсиональное равенство. Все определения в разрабатываемой библиотеке должны быть корректны, т.е. функции, получив экстенсионально равные аргументы, должны возвращать экстенсионально равные термы; то же касается и отношений, чья область значений - это тип Prop, в данном случае значения должны быть эквивалентны как пропозиции.

Определение 6.4. Экстенсиональное равенство на множествах рекурсивно определяется следующим образом. Множество E1 равно множеству E2, если для каждого элмента множества E1 найдётся равный ему элемент множества E2 и, наоборот, для каждого элемента множества E2 найдётся равный ему элемент множества E1.

```
(* Extensional equality of sets *)
Fixpoint EQ (E1 E2: Ens) {struct E2}: Prop :=
  match E1, E2 with
  | sup A f, sup B g =>
          (forall x : A, exists y : B, EQ (f x) (g y)) /\
          (forall y : B, exists x : A, EQ (f x) (g y))
  end.
```

Причина введения нового равенства следующая: стандартный тип равенства Лейбница (которое далее будем называть “графическим” или “по определению”) из библиотеки Coq не подходит для описания равенства множеств. Пример экстенсионально равных, но не равных по определению множеств.

Множество $\{\emptyset\}$, может быть задано следующими способами: как SV1:=sup unit (fun _ => Vide) и как SV2:=sup bool (fun _ => Vide). При этом SV1=SV2 и SV1:≠SV2. Верифицируем это формально, потребуется следующее утверждение.

Лемма 6.1. Если два типа равны, то между ними существует биекция, в часности можно утверждать, что существуют функции f и g такие, что $\forall(x : A)\forall(y : B).f(x) = y \to g(y) = x$.

```
Lemma exists_bijectionT (A B: Type):
A = B -> exists f, exists g, forall (x:A) (y:B),
 f x = y -> g y = x.
Proof.
 intro H; subst.
 exists id.
 exists id.
 intros. subst.
 reflexivity.
Qed.
```

Лемма 6.2. Тип bool не равен типу unit.

```
Lemma unit_neq_boolT: @eq Type bool unit -> False.
Proof.
intro Heq.
Check exists_bijectionT bool unit Heq.
destruct (exists_bijectionT _ _ Heq) as [f [g Hfg]].
```

```
destruct (f true) eqn: Hft.
destruct (f false) eqn: Hff.
pose proof (Hfg _ _ Hft) as Hgtt.
pose proof ( Hfg _ _ Hff) as Hgtf.
rewrite Hgtf in Hgtt.
inversion Hgtt.
Qed.
```

Определение в Coq множеств SV1 и SV2.

```
Definition SV1 : Ens := sup unit (fun _ => Vide).
Definition SV2 : Ens := sup bool (fun _ => Vide).
```

Теорема 6.3. Термы, задающие множества SV1 и SV2 не равны.

```
Theorem graph_not_ext : SV2 <> SV1.
Proof.
intro H.
inversion H.
destruct (unit_neq_boolT H1).
Defined.
```

Теорема 6.4. Множества SV1 и SV2 экстенсионально равны.

```
Theorem ex_EQ_ext : EQ SV2 SV1.
Proof.
simpl.
split.
+ intros.
  exists tt.
  split; intros [].
+ intros [].
  exists true.
  split; intros [].
Defined.
```

Определим теперь предикат принадлежности.

Определение 6.5. Множество E1 принадлежит множеству E2, если E1 экстенсионально равно образу некоторого терма из базового типа множества E2.

```
(* Membership on sets *)
Definition IN (E1 E2 : Ens) : Prop :=
  match E2 with
  | sup A f => exists y : A, EQ E1 (f y)
  end.
```

Определение 6.6. Множество E1 принадлежит множеству E2, если всякое множество принадлежащее E1 принадлежит и E2.

```
(* INCLUSION *)
Definition INC : Ens -> Ens -> Prop
 := (fun E1 E2 : Ens =>
      forall E : Ens, IN E E1 -> IN E E2
    ).
```

Докажем теперь, что эктенсиональное равенство – отношение эквивалентности.

Лемма 6.5. Отношение экстенсионального равенства рефлексивно.

```
(* EQ is an equivalence relation *)
Fixpoint EQ_refl (E : Ens) : EQ E E.
Proof.
destruct E as [A f].
```

```
split; intros z; exists z; exact (EQ_refl (f z)).
Defined.
```

Лемма 6.6. Отношение экстенсионального равенства транзитивно.

```
Fixpoint EQ_tran (E1 E2 E3 : Ens) {struct E2}:
 EQ E1 E2 -> EQ E2 E3 -> EQ E1 E3.
Proof.
destruct E1 as [A1 f1], E2 as [A2 f2], E3 as [A3 f3].
intros E1eqE2 E2eqE3.
destruct E1eqE2 as [E12 E21].
destruct E2eqE3 as [E23 E32].
simpl in |- *.
split.
+ intro x1.
  destruct (E12 x1) as [x2 P12].
  destruct (E23 x2) as [x3 P23].
  exists x3. apply (EQ_tran (f1 x1) (f2 x2) (f3 x3) P12 P23).
+ intro x3.
  destruct (E32 x3) as [x2 P32].
  destruct (E21 x2) as [x1 P21].
  exists x1. apply (EQ_tran (f1 x1) (f2 x2) (f3 x3) P21 P32).
Defined.
```

Лемма 6.7. Отношение экстенсионального равенства симметрично.

```
Fixpoint EQ_sym (E1 E2 : Ens) {struct E2}: EQ E1 E2 -> EQ E2 E1.
Proof.
intro H.
destruct E1 as [A f], E2 as [B g].
simpl in * |- *.
destruct H as [A2B B2A]; split.
+ intro b. destruct (B2A b) as [a J]. exists a. apply EQ_sym with (1:=J).
+ intro a. destruct (A2B a) as [b J]. exists b. apply EQ_sym with (1:=J).
Defined.
```

Добавим симметричность и рефлексивность в список теорем, используемой тактикой auto with.

```
Hint Resolve EQ_sym EQ_refl : zfc.
```

Следствие 6.7.1. Равенство корректно относительно замены левого аргумента на экстенсионально равный.

```
Theorem EQ_sound_left (c a b : Ens) (aeqb : EQ a b)
 (H : EQ a c) : EQ b c.
Proof.
apply EQ_sym in aeqb.
eapply EQ_tran.
exact aeqb.
exact H.
Defined.
```

Следствие 6.7.2. Равенство корректно относительно замены правого аргумента на экстенсионально равный.

```
Definition EQ_sound_right (a b c : Ens) (aeqb : EQ b c)
 (H : EQ a b) : EQ a c
:= EQ_tran _ _ _ H aeqb.
```

Определение 6.7. Пусть P – некоторый одноместный предикат на термах типа Ens. Если его истинностное значение не меняется при замене аргумента на экстенсионально равный, то называем его корректно определённым предикатом на множествах и обозначаем этот факт как (SoundPred P).

```
Definition SoundPred
:= (fun (P:Ens->Prop)=>(forall w1 w2 : Ens, EQ w1 w2 -> P w1 -> P w2)).
```

Теорема 6.8. Отношение $x \in y$ корректно определено относительно замены x на экстенсионально равный ему терм типа Ens.

```
(* Membership is extentional (i.e. is stable w.r.t. EQ) *)
Theorem IN_sound_left :
 forall E : Ens, SoundPred (fun y=>IN y E).
Proof.
intros C A B AeqB AinC.
destruct C as [T F].
simpl in * |- *.
destruct AinC as [Y AeqFY].
exists Y.
apply EQ_tran with A.
+ apply EQ_sym. exact AeqB.
+ apply AeqFY.
Defined.
```

Теорема 6.9. Отношение $x \in y$ корректно определено относительно замены y на экстенсионально равный ему терм типа Ens.

```
Theorem IN_sound_right :
 forall E : Ens, SoundPred (IN E).
(* forall E E' E'' : Ens, EQ E' E'' -> IN E E' -> IN E E''. *)
Proof.
intros A B C BeqC AinB.
destruct B as [Y G].
destruct C as [Z H].
simpl in *|-*.
destruct BeqC as [Y2Z Z2Y].
destruct AinB as [y AeqGy].
destruct (Y2Z y) as [z GyeqHz].
exists z.
eapply EQ_tran with (1:=AeqGy) (2:=GyeqHz).
Defined.
```

6.4 Выполнимость аксиом в модели

6.4.1 Доказательство аксиомы экстенсиональности в модели

Докажем, что в моделе выполняется аксиома

$$\text{axExt} : \forall a_1 \forall a_2 \big(\forall b(b \in a_1 \leftrightarrow b \in a_2) \leftrightarrow a_1 = a_2\big)$$

Сначала справа-налево, тривиальным образом, в силу корректности отношения принадлежности.

```
Theorem axExt_right : forall x y : Ens,
   EQ x y -> forall z, (IN z x <-> IN z y).
Proof.
intros; split.
+ apply IN_sound_right. exact H.
+ apply IN_sound_right. apply EQ_sym, H.
Defined.
```

Далее доказательство слева-направо:

```
Theorem axExt_left : forall (x y : Ens),
  (forall z, IN z x <-> IN z y) -> EQ x y.
Proof.
intros x y K.
destruct x as [A f], y as [B g].
```

```
simpl in * |- *.
split.
- intro x.
  apply K.
  exists x.
  apply EQ_refl.
- intro y.
  assert (Q: exists b : B, EQ (g y) (g b)).
  { exists y. apply EQ_refl. }
  destruct (proj2 (K (g y)) Q) as [x H].
  exists x.
  apply EQ_sym.
  exact H.
Defined.
```

Наконец, объединим обе стрелки в эквивалентность

```
Theorem axExt : forall x y : Ens,
   EQ x y <-> forall z, (IN z x <-> IN z y).
Proof.
intros; split.
+ apply axExt_right.
+ apply axExt_left.
Defined.
```

6.4.2 Свойства отношения включения

Теорема 6.10. Если $a = b$, то $a \subseteq b$.

```
Theorem EQ_INC : forall a b : Ens, EQ a b -> INC a b.
Proof.
intros a b H z.
eapply axExt_right in H.
destruct H as [H1 H2].
exact H1.
Defined.

Hint Resolve EQ_sym EQ_refl EQ_INC: zfc.
```

Теорема 6.11. Отношение $\subseteq$ – рефлексивное, то есть для любого a, $a \subseteq a$.

```
(* Inclusion is a sound, reflexive, antisymmetric and
   transitive relation. *)
Definition INC_refl : forall E : Ens, INC E E
 := fun (E a : Ens) (H : IN a E) => H.
```

Теорема 6.12. Отношение $\subseteq$ – антисимметричное, то есть для любых a и b, из $a \subseteq b$ и $b \subseteq a$ следует $a = b$.

```
Theorem INC_antisym : forall a b : Ens,
 INC a b -> INC b a -> EQ a b.
Proof.
intros A B H1 H2.
apply axExt_left.
intro z. split.
+ apply H1.
+ apply H2.
Defined.
```

Теорема 6.13. Отношение $\subseteq$ – транзитивное, то есть для любых a,b,c, из $a \subseteq b$ и $b \subseteq c$ следует $a \subseteq c$.

```
Theorem INC_tran : forall a b c : Ens,
 INC a b -> INC b c -> INC a c.
```

```
Proof.
unfold INC in |- *; auto with zfc.
Defined.
Hint Resolve INC_antisym: zfc.
```

Теорема 6.14. Отношение $x \subseteq y$ – корректно определено относительно замены аргумента x на экстенсионально равный.

```
Theorem INC_sound_left :
 forall A B C : Ens, EQ A B -> INC A C -> INC B C.
Proof.
intros A B C AeqB AincC Z ZinB.
apply AincC.
eapply IN_sound_right.
+ apply EQ_sym. exact AeqB.
+ exact ZinB.
Defined.
```

Теорема 6.15. Отношение $x \subseteq y$ – корректно определено относительно замены аргумента y на экстенсионально равный.

```
Theorem INC_sound_right :
 forall A B C : Ens, EQ B C -> INC A B -> INC A C.
Proof.
intros A B C BeqC AincB Z ZinA.
eapply IN_sound_right.
+ exact BeqC.
+ apply AincB.
  exact ZinA.
Defined.
```

6.4.3 Аксиома пары

В данном разделе докажем, что для любых двух множеств существует множество, содержащее только их.

Определение 6.8. Пара – это терм типа Ens с базисным типом bool.

```
Definition Pair (A B:Ens) : Ens
 := sup bool (fun b : bool => if b then A else B).
```

Лемма 6.16. Терм $\{a, b\}$ корректно определён относительно замены a на экстенсионально равный терм.

```
(* The pair construction is extensional *)
Theorem Pair_sound_left :
 forall A A' B : Ens, EQ A A' -> EQ (Pair A B) (Pair A' B).
Proof.
unfold Pair in |- *.
simpl in |- *.
intros A A' B AeqA';
split; (intros [|];
 [exists true; exact AeqA' | exists false; exact (EQ_refl B)]
).
Defined.
```

Лемма 6.17. Терм $\{a, b\}$ корректно определён относительно замены b на экстенсионально равный терм.

```
Theorem Pair_sound_right :
 forall A B B' : Ens, EQ B B' -> EQ (Pair A B) (Pair A B').
Proof.
unfold Pair in |- *; simpl in |- *; intros; split.
+ simple induction x.
  exists true; auto with zfc.
```

```
  exists false; auto with zfc.
+ simple induction y.
  exists true; auto with zfc.
  exists false; auto with zfc.
Defined.
Hint Resolve Pair_sound_right Pair_sound_left: zfc.
```

Лемма 6.18. $\forall a \forall b (a \in \{a, b\})$.

```
(* The axioms of the pair *)
Theorem IN_Pair_left : forall E E' : Ens, IN E (Pair E E').
Proof.
unfold Pair in |- *. simpl in |- *. exists true. simpl in |- *.
auto with zfc.
Defined.
```

Лемма 6.19. $\forall a \forall b (b \in \{a, b\})$.

```
Theorem IN_Pair_right : forall E E' : Ens, IN E' (Pair E E').
Proof.
unfold Pair in |- *. simpl in |- *. exists false. simpl in |- *.
exact (EQ_refl E').
Defined.
```

Лемма 6.20. $\forall a \forall b (c \in \{a, b\} \to (c = a \vee c = b))$.

```
Theorem Pair_IN :
 forall E E' A : Ens, IN A (Pair E E') -> EQ A E \/ EQ A E'.
Proof.
unfold Pair in |- *; simpl in |- *.
intros E E' A [b P].
destruct b; auto with zfc.
Defined.
Hint Resolve IN_Pair_left IN_Pair_right nothing_IN_Vide: zfc.
```

Теорема 6.21. Аксиома пары верна для типа множеств Ens.

$$\text{axPair} : \forall a_1 \forall a_2 \exists c \forall b \big(b \in c \leftrightarrow (b = a_1 \vee b = a_2)\big)$$

```
(* axExt see above *)
Theorem axPair : forall a b : Ens, exists w:Ens,
   forall z, (IN z w <-> EQ z a \/ EQ z b).
Proof.
intros a b.
exists (Pair a b).
intro z.
split.
+ apply Pair_IN.
+ intros [H|H].
  - eapply IN_sound_left.
    apply EQ_sym; exact H.
    apply IN_Pair_left.
  - eapply IN_sound_left.
    apply EQ_sym in H; exact H.
    apply IN_Pair_right.
Defined.
```

6.4.4 Аксиома объединения

Определение 6.9. π_1 – одноместная функция, возвращающая базовый тип произвольного множества.

```
(* Projections of a set: *)
(* 1: its base type *)
Definition pi1 (X:Ens):Type
 := match X with
    | sup A _ => A
    end.
```

Определение 6.10. $\pi_2(X)$ – есть опорная функция множества X из его базового типа $\pi_1(X)$ в тип множеств Ens.

```
(* 2: the function *)
Definition pi2 (X:Ens) (m:pi1 X):Ens
:= match X as E return (pi1 E -> Ens) with
   | sup A f => fun k : pi1 (sup A f) => f k
   end m.
```

Если E = sup A F , то у Union(E) опорный тип {x:A & π1 (f x)}, который представляет всевозможные пары (x,p), где x:A, а p принадлежит опорному типу множества (f x). Опорная функция по такой паре восстанавливает опорную функцию множества (f x) и применяет ее к p. Другими словами, чтобы определить объединение элементов множества A, используется сигма-тип, дизъюнктивно объединяющий базовые типы множеств из $\mathrm{image}_{\pi_2}(\pi_1(A))$.

Определение 6.11. Union(X) – это образ дизъюнктивной суммы опорных функций элементов X.

```
(* The Union set *)
Definition Union : forall E : Ens, Ens.
Proof.
intros [A f].
apply (sup { x : A & pi1 (f x)} ).
intros [a b].
exact (pi2 (f a) b).
Defined.
```

Теорема 6.22. (транспортная) Для экстенсионально равных множеств существует функция между их базовыми типами, такая что всякий образ её аргумента экстенсионально равен образу её значения.

```
Theorem EQ_EXType :
 forall E E' : Ens,
 EQ E E' ->
 forall a : pi1 E,
 exists b : pi1 E', EQ (pi2 E a) (pi2 E' b).
Proof.
intros [A f] [A' f'] [e1 e2].
simpl in |- *.
apply e1.
Defined.
```

Теорема 6.23.

```
Theorem IN_EXType :
 forall E E' : Ens,
 IN E' E -> exists a : pi1 E, EQ E' (pi2 E a).
Proof.
intros [A f]. simpl.
intros [A' f']. trivial.
Defined.
```

Теорема 6.24. Теоретико-множественный факт

$$\forall A \forall B \big(B \in \bigcup A \to \exists C (C \in E \wedge B \in C)\big).$$

```
Theorem Union_IN :
```

```
 forall E E' : Ens,
 IN E' (Union E) -> exists E1 : Ens, IN E1 E /\ IN E' E1.
Proof.
intros [A f].
simpl in |- *.
simple induction 1.
intros [a b].
intros.
exists (f a).
split.
+ exists a; auto with zfc.
+ apply IN_sound_left with (pi2 (f a) b).
  1 : auto with zfc.
  simpl in |- *.
  destruct (f a). simpl.
  exists b. auto with zfc.
Defined.
```

Теорема 6.25. Теоретико-множественный факт

$$\forall A \forall B \forall C \big(B \in A \wedge C \in B \to C \in \bigcup A\big).$$

```
Theorem IN_Union :
 forall E E' E'' : Ens, IN E' E -> IN E'' E' -> IN E'' (Union E).
Proof.
intros E E' E'' H H0.
destruct (IN_EXType E E' H) as [x e].
destruct E as [A f].
assert (e1 : EQ (pi2 (sup A f) x) E').
{ apply EQ_sym; exact e. }
assert (i1:IN E'' (pi2 (sup A f) x)).
{ apply IN_sound_right with E'; auto with zfc. }
apply IN_EXType in i1 as [x0 e2].
simpl in x0.
exists (existT (fun x : A => pi1 (f x)) x x0).
exact e2.
Defined.
```

Теорема 6.26. Аксиома объединения верна для типа множеств Ens.

$$\text{axUnion} : \forall a \exists d \forall b \big(b \in d \leftrightarrow \exists c (b \in c \wedge c \in a)\big)$$

```
Theorem axUnion : forall X : Ens, exists Y:Ens,
   forall z, (IN z Y <-> exists m:Ens, IN m X /\ IN z m).
Proof.
intros X.
exists (Union X).
intro z; split; intro H.
+ apply Union_IN.
  assumption.
+ destruct H as [m [minX zinm]].
  eapply IN_Union.
  exact minX.
  exact zinm.
Defined.
```

6.4.5 Аксиома степени

Цель данного раздела – определить терм множества всех подмножеств и доказать, что аксиома степени верна для типа множеств Ens.

$$\text{axPow} : \forall a \exists d \forall b \big(b \in d \leftrightarrow \forall c (c \in b \to c \in a)\big)$$

Определение 6.12. Зададим множество подмножеств множества E как терм типа Ens, следующим способом:

1. Его базовый тип – это тип предикатов на базовом типе множества E.

2. Его опорная функция – это сопоставляет предикату P из базового типа E множество

   (a) с базовым типом $\Sigma_{(a:A)}P(a)$ и

   (b) опорной функцией $\lambda c.f(\pi_1(c))$, где f – опорная функция множества E.

```
(* The powerset and its axioms *)
Definition Power (E : Ens) : Ens :=
  match E with
  | sup A f =>
      sup _
        (fun P : A -> Prop =>
         sup _
           (fun c : sigT (fun a : A => P a) =>
            match c with
            | @existT _ _ a p => f a
            end))
  end.
```

Теорема 6.27. Всякий элемент E' множества всех подмножеств множества E – подмножество E.

$$\forall E \forall E' (E' \in E \to E' \subseteq E)$$

```
Theorem IN_Power_INC : forall E E' : Ens, IN E' (Power E) -> INC E' E.
Proof.
intros [A f].
intros E'.
intros [P H]. revert H.
destruct E' as [A' f'].
intros [HA HB].
intros E'' [a' e].
destruct (HA a') as [[a p] H].
intros; exists a.
apply EQ_tran with (f' a'); auto with zfc.
Defined.
```

Теорема 6.28. Всякое подмножество E – элемент множества всех подмножеств множества E.

$$\forall E \forall E' (E' \subseteq E \to E' \in E)$$

```
Theorem INC_IN_Power : forall E E' : Ens, INC E' E -> IN E' (Power E).
Proof.
intros [A f].
intros [A' f'] i.
exists (fun a : A => IN (f a) (sup A' f')).
simpl in |- *.
split.
+ intros.
  elim (i (f' x)).
  - intros a e.
    cut (EQ (f a) (f' x)); auto with zfc.
    intros e1.
    exists
     (existT (fun a : A => exists y : A', EQ (f a) (f' y)) a
```

```
      (ex_intro (fun y : A' => EQ (f a) (f' y)) x e1)).
    simpl in |- *.
    auto with zfc.
  - auto with zfc.
    simpl in |- *.
    exists x; auto with zfc.
+ simple induction y; simpl in |- *.
  simple induction 1; intros.
  exists x0; auto with zfc.
Defined.
```

Теорема 6.29. Аксиома степени верна для типа множеств Ens.

$$\text{axPow} : \forall a \exists d \forall b \big(b \in d \leftrightarrow \forall c (c \in b \to c \in a)\big)$$

```
Theorem axPower : forall X : Ens, exists Y:Ens,
   forall z, (IN z Y <-> INC z X).
Proof.
intros.
exists (Power X).
intro z; split; intro H.
+ apply IN_Power_INC.
  exact H.
+ apply INC_IN_Power.
  exact H.
Defined.
```

6.4.6 Аксиома регулярности

Теорема 6.30. Аксиома регулярности верна для типа множеств Ens.

$$\text{axRegul} : \forall a \Big(a \neq \varnothing \to \exists b \big(b \in a \wedge \forall c (c \in b \to c \notin a)\big)\Big)$$

Для доказательства этого факта, сначала докажем, что для термов типа Ens выполняется $\in$-индукция.

Теорема 6.31. Если R – корректно определённый предикат на множествах, то

$$\big(\forall x (\forall y . y \in x \to R(y)) \to R(x)\big) \to \forall z . R(z).$$

```
(* Epsilon induction. *)
Theorem eps_ind (R:Ens->Prop) (Sou_R:forall a b, EQ a b -> (R a -> R b))
: (forall x:Ens, (forall y, IN y x -> R y) -> R x) -> forall z, R z.
Proof.
intros.
induction z.
apply H.
simpl.
intros y q.
destruct q as [a G].
apply (Sou_R (e a) y).
apply EQ_sym, G.
apply H0.
Defined.
```

Определение 6.13. Множество a – $\in$-минимальный элемемент в b, если $((a \in b) \wedge (\forall c . c \in b \to c \notin a))$.

```
Definition epsmin a b := IN a b /\ forall c, IN c b -> ~IN c a.
```

Определение 6.14. Множество x – подрегулярное, если всякое содержащее его множество содержит ϵ-минимальный элемент.

```
Definition regular_over x := forall u : Ens, (IN x u -> exists y, epsmin y u).
```

Теорема 6.32. Одноместный предикат regular_over корректно определён.

```
Theorem regular_over_sound : forall a b : Ens,
 EQ a b -> regular_over a <-> regular_over b.
Proof.
intros.
unfold regular_over.
split; intros.
+ eapply IN_sound_left in H1.
  apply H0. apply H1.
  apply EQ_sym. exact H.
+ eapply IN_sound_left in H1.
  apply H0. apply H1.
  exact H.
Defined.
```

Теорема 6.33. Если каждый элемент множества вполне упорядочен отношением принадлежности, то и само множество тоже вполне упорядочено отношением принадлежности. Здесь я следую доказательству из [5].

Определение 6.15. c — бесконечная цепь, если

```
Definition UC c := forall m, IN m c -> exists n, IN n m /\ IN n c.
```

Определение 6.16. Множество x фундировано, если не существует бесконечной цепи c такой, что $x \in c$.

```
Definition WF x := ~(exists c, UC c /\ IN x c).
```

Докажем теперь индуктивной шаг для эпсилон-индукции.

```
Theorem Zuhair_1 (a:Ens): (forall x, IN x a -> WF x) -> WF a.
Proof.
unfold WF.
intros H K0.
pose (K:=K0).
destruct K as [c [M1 M2]].
unfold UC in M1.
pose (B:=M1 a M2).
destruct B as [n [nina ninc]].
apply (H n nina).
exists c.
split.
exact M1.
exact ninc.
Defined.
```

Теорема 6.34. Предикат WF корректно определён.

```
Theorem WF_sound : SoundPred WF.
(*forall a b : Ens, EQ a b -> WF a -> WF b.*)
Proof.
unfold WF, SoundPred in *|-*.
intros.
intro B.
apply H0.
+ (*intros A B.*)
  (*apply A.*)
  destruct B as [c [a1 a2]].
  exists c.
  split. exact a1.
  eapply IN_sound_left. (*with (E:=b).*)
  apply EQ_sym. exact H.
  exact a2.
(*+ intros A B.
```

```
  apply A.
  destruct B as [c [a1 a2]].
  exists c.
  split. exact a1.
  eapply IN_sound_left. (* with (E:=a).*)
  exact H.
  exact a2.*)
Defined.
```

Теорема 6.35. Всякое множество – фундировано.

```
(* Induction. "Every set is well-founded." *)
Theorem Zuhair_2 (y:Ens): WF y.
Proof.
apply eps_ind.
- exact WF_sound.
- intros a. exact (Zuhair_1 a).
Defined.
```

Теорема 6.36. Всякое множество надрегулярно. Текст формальный доказательства из [6].

```
Theorem Blass x : regular_over x.
Proof.
unfold regular_over.
pose (A:=Zuhair_2 x); unfold WF in A.
intros u xinu.
(* Series of the equivalent tranformations.*)
apply not_ex_all_not with (n:=u) in A.
apply not_and_or in A.
 destruct A as [H1|H2].
 2 : destruct (H2 xinu).
 unfold UC in H1.
apply not_all_ex_not in H1.
 destruct H1 as [yy yH].
 exists yy.
apply imply_to_and in yH.
 destruct yH as [Ha Hb].
 split. exact Ha.
 intro z.
apply not_ex_all_not with (n:=z) in Hb.
apply not_and_or in Hb.
 intro v.
 destruct Hb as [L0|L1]. exact L0.
 destruct (L1 v).
Defined.
```

Теорема 6.37. Для термов типа Ens выполняется аксиома регулярности.

$$\text{axRegul} : \forall a\Big(a \neq \varnothing \to \exists b\big(b \in a \wedge \forall c(c \in b \to c \notin a)\big)\Big)$$

```
Theorem axReg (x:Ens) : (exists a, IN a x)->(exists y, IN y x /\ ~exists z, IN z y /\ IN z x).
Proof.
pose (Q:= Blass).
unfold regular_over in Q.
intro e.
destruct e as [z zinx].
pose (f:= Q z x zinx).
destruct f as [g G].
exists g.
destruct G as [G1 G2].
split.
+ exact G1.
+ intro s.
```

```
  destruct s as [w [W1 W2]].
  exact (G2 w W2 W1).
Defined.
```

6.4.7 Аксиома бесконечности

Определим унарную операцию следования на множествах:

```
Definition succ (E : Ens) := Union (Pair E (Sing E)).
```

Рекурсией по номеру n определим функцию, сопоставляющую натуральному числу n множество, получившееся при n-кратном применении операции следования.

```
Definition Nat : nat -> Ens.
Proof.
simple induction 1; intros.
exact Vide.
exact (succ X).
Defined.
```

Это позволяет определить множество натуральных чисел.

Определение 6.17. Omega – множество с базовым типом nat и опорной функцией Nat.

```
Definition Omega : Ens := sup nat Nat.
```

Теорема 6.38. Аксиома бесконечности верна для типа множеств Ens.

$$\text{axInf:}\exists a\big(\varnothing \in a \wedge \forall b(b \in a \to b \cup \{b\} \in a)\big)$$

Как и ранее Vide обозначает пустое множество, а succ(Y)=$Y \cup \{Y\}$.

```
Theorem axInf : exists X, (IN Vide X /\ forall Y, (IN Y X -> IN (succ Y) X)).
Proof.
exists Omega.
split.
+ unfold Omega.
  unfold IN.
  exists 0.
  apply EQ_refl.
+ intros Y YinOm.
  apply IN_Omega_EXType in YinOm.
  destruct YinOm as [x H].
  assert (as1: EQ (succ (Nat x)) (succ Y)).
  apply succ_sound. exact H.
  apply (IN_sound_left _ _ _ as1).
  apply (Nat_IN_Omega (S x)).
Defined.
```

6.4.8 Аксиома выделения

Теорема 6.39. Аксиома выделения верна для типа множеств Ens.

$$\text{axSep:}\forall w_1, \ldots, w_n \forall A \exists B \forall x(x \in B \Leftrightarrow [x \in A \wedge \varphi(x, w_1, \ldots, w_n, A)]), \text{ где } B \notin \mathrm{FV}(\varphi)$$

Определение 6.18. Пусть M=sup A f – произвольный терм типа Ens. Тогда (Comp M P) – множество, базовый тип которого $\Sigma_{x:A}P(fx)$, а опорная функция – $\lambda x.f(\pi_1(x))$.

```
Definition Comp : Ens -> (Ens -> Prop) -> Ens.
Proof.
intros [A f] P.
apply (sup (sig (fun x:A => P (f x)))).
intros [x _].
exact (f x).
Defined.
```

Теорема 6.40. Множество, полученное сепарацией – подмножество исходного множества.

```
(* The comprehension/separation axioms *)
Theorem Comp_INC : forall (E : Ens) (P : Ens -> Prop), INC (Comp E P) E.
Proof.
intros E P z zinCompEP.
destruct E as [A f].
simpl in *|-*.
destruct zinCompEP as [w R].
destruct w as [a Pfa].
exists a. exact R.
Defined.
```

Теорема 6.41. Всякий элемент множества, полученного сепарацией множества M корректным свойством P, обладает свойством P.

```
Theorem IN_Comp_P : forall (E A : Ens) (P : Ens -> Prop), (SoundPred P) -> IN A (Comp E P) -> P A.
Proof.
intros E A P H H0.
destruct E,H0,x as [a p].
apply H with (2:=p).
apply EQ_sym. assumption.
Defined.
```

Теорема 6.42. Обратно, всякий элемент множества M обладающий свойством P, является элементом сепарации M свойством P.

```
(* I2AST p.13, thm 4.12, (<-) *)
Theorem IN_P_Comp : forall (E A : Ens) (P : Ens -> Prop), (SoundPred P) ->
  IN A E -> P A -> IN A (Comp E P).
Proof.
intros.
destruct E.
simpl in *|-*.
destruct H0 as [a p].
unshelve eapply ex_intro.
exists a.
apply H with (2:=H1).
exact p.
simpl.
exact p.
Defined.
```

Определение 6.19. Назовём классами все эквивалентные на экстенсионально равных аргументах предикаты на Ens.

Определение 6.20. class – тип классов.
Произвольная совокупность классов – это, вообще говоря, конгломерат. Тип class можно интерпретировать как конгломерат всех классов, а его термы – как некоторые классы.

```
Record class := Build_class {
 prty :> Ens->Prop;
 sound : SoundPred prty;
}.
```

Теорема 6.43. Для любого класса и множества X существует множество Y являющееся пересечением C и X.

```
Theorem schSepar (c:class) :
forall X:Ens, exists Y:Ens, forall z, (IN z Y <-> IN z X /\ (prty c z)).
Proof.
intros X.
exists (Comp X c).
```

```
intro z; split; intro H.
+ pose (H':=H).
  apply IN_Comp_P in H'.
  - eapply Comp_INC in H.
    firstorder.
  - unfold SoundPred. exact (sound c). (*unfold SoundPred. apply schSepar_lem.*)
+ apply IN_P_Comp.
  - unfold SoundPred. apply (sound c). (*apply schSepar_lem*)
  - firstorder.
  - firstorder.
Defined.
```

6.4.9 Аксиома замены

В данном разделе, который посвящён аксиоме замены, оставим утверждения без доказательств, поскольку доказательства есть исходном коде [7] для статьи [8]. Они не были модифицированы или передоказаны в рамках данной работы. Приведём лишь определения и результаты.

$$\text{axRepl:}\forall x \exists! y \varphi \to \forall a \exists d \forall c \big(c \in d \leftrightarrow \exists b (b \in a \wedge \varphi(b/x)(c/y))\big),$$

$$\text{если } \begin{cases} \mathrm{FV}(\varphi) \subseteq \{x, y\} \\ \mathrm{FFI}(b, x, \varphi) \\ \mathrm{FFI}(c, y, \varphi) \end{cases}.$$

Определение 6.21. Для типа Ens верна аксиома замены, если следующий тип населён:

```
Definition replacement :=
  forall P : Ens -> Ens -> Prop,
  functional P ->
  (forall x y y' : Ens, EQ y y' -> P x y -> P x y') ->
  (forall x x' y : Ens, EQ x x' -> P x y -> P x' y) ->
  forall X : Ens,
  exists Y : Ens,
     forall y : Ens,
     (IN y Y -> exists x : Ens, IN x X /\ P x y) /\
((exists x : Ens, IN x X /\ P x y) -> IN y Y).
```

Определение 6.22. Для типа Ens верна аксиома коллекции, если следующий тип населён:

```
Definition collection :=
  forall P : Ens -> Ens -> Prop,
  (forall x x' y : Ens, EQ x x' -> P x y -> P x' y) ->
  (forall E : Ens, exists m, (P E m)) ->
  forall E : Ens,
  exists A : Ens,
     forall x : Ens, IN x E -> exists y : Ens, IN y A /\ P x y).
```

Определение 6.23. Для типа Ens верна аксиома выбора, если следующий тип населён:

```
Definition choice :=
  forall (A B : Type) (P : A -> B -> Prop),
  (forall a : A, exists b : B, P a b) ->
exists f : A -> B, forall a : A, P a (f a).
```

Далее доказана лемма, что принятие аксиомы выбора влечёт истинность аксиомы коллекции.

```
Lemma Choice_Collection : choice -> collection.
```

После представлен результат, в следующей формулировке:

Теорема 6.44. Из закона исключённого третьего для высказываний и аксиомы коллекции следует аксиома замены.

```
Theorem classical_Collection_Replacement :
(forall S : Prop, S \/ (S -> False)) -> collection -> replacement.
```

6.4.10 Аксиома выбора

В исходной статье Вернера указан этот факт, однако не было приведено доказательства.

Теорема 6.45. Аксиома выбора верна для типа множеств Ens, если принять теоретико-типовую аксиому выбора.

$$\text{axChoice}:\forall a\Big(a \neq \varnothing \wedge \forall b(b \in a \to b \neq \varnothing) \wedge \forall b_1 \forall b_2 (b_1 \neq b_2 \wedge \{b_1, b_2\} \subseteq a \to b_1 \cap b_2 = \varnothing) \to \exists d \forall b\big(b \in a \to \exists c(b \cap d = \{c\})\big)\Big)$$

Докажем следующие леммы, которые потребуются в дальнейшем:

Лемма 6.46. Всякому множеству $x \in s$ соответствует терм из базисного типа множества s. Определим таким образом функцию in2term.

```
Lemma in2term (s x:Ens) : IN x s -> (pi1 s).
Proof. intro xins. destruct s. simpl in xins.
apply ex2sig in xins.
destruct xins as [y ep].
exact y.
Defined.
```

Лемма 6.47. Функция in2term – секция функции π_2.

```
Lemma goodlem (b US:Ens) (binUS:IN b US) : EQ b (pi2 US (in2term US b binUS)).
Proof.
unfold in2term.
destruct US.
simpl.
simpl in binUS.
destruct (ex2sig binUS). (* A (fun y : A => EQ b (e y)) *)
assumption.
Defined.
```

В данной секции рассмотрим любое множество S, которое не содержит пустого множества .

```
Section AC_sec.
Context (S:Ens).
Context (H:~IN Vide S).
```

Определение 6.24. Пусть AC_A – базовый тип множества S.

Определение 6.25. Пусть AC_B – базовый тип множества $\bigcup S$.

Определение 6.26. Тогда (AC_T a b) – поднятое на термы базовых типов отношение принадлежности между образами термов (a:AC_A) и (b:AC_B).

```
Definition AC_A:=pi1 S.
Definition AC_B:=pi1 (Union S).
Definition AC_T:AC_A->AC_B->Prop
 := fun a b => IN (pi2 (Union S) b) (pi2 S a).
Theorem AC_hyp : forall a : AC_A, (exists b : AC_B, AC_T a b).
Proof.
intro a.
unfold AC_A, AC_B, AC_T in *|-*.
pose (XinS := lem4 S a).
pose (X:=pi2 S a). (* Множество'X' соответствуеттерму'a'.*)
(*'X' is nonempty *)
(* so there exists q, 'IN q X' *)
pose (J:=lem3 S H X XinS).
destruct J as [b binX].
```

```
pose (binUS := IN_Union S X b XinS binX).
exists (in2term _ _ binUS).
fold X in XinS |- *.
simpl.
eapply IN_sound_left.
apply goodlem.
assumption.
Defined.
```

Определение 6.27. (AC_R a1 a2) – поднятое на термы базового типа отношение равенства между образами термов (a1:AC_A) и (a2:AC_A).

```
Definition AC_R : AC_A->AC_A->Prop := fun a1 a2 => EQ (pi2 S a1) (pi2 S a2).
```

Лемма 6.48. AC_R – отношение эквивалентности.

```
Theorem AC_eqvR : RelationClasses.Equivalence AC_R.
Proof.
constructor.
+ intro x. apply EQ_refl.
+ intros x1 x2 K. apply EQ_sym. exact K.
+ intros x1 x2 x3 K1 K2. eapply EQ_tran. exact K1. exact K2.
Defined.
```

Лемма 6.49. AC_T – корректное слева бинарное отношение.

```
Theorem T_sound : (forall (x x' : AC_A) (y : AC_B),
 AC_R x x' -> AC_T x y -> AC_T x' y).
Proof.
intros x1 x2 y Rx1x2 Txy.
eapply IN_sound_right.
exact Rx1x2.
exact Txy.
Defined.
```

Воспользуемся теоретико типовой аксиомой выбора, расширяющей доказуемость в системе Coq.

$$\begin{aligned}\text{SetoidFunctionalChoice}:\forall A\forall B\forall(R:A\to A\to\text{Prop}),\forall(T:A\to B\to\text{Prop}),\text{Equivalence(R)}\to\\ \to(\forall x\forall x'\forall y,R(x,x')\to T(x,y)\to T(x',y))\to\\ \to(\forall x,\exists y,T(x,y))\to\\ \to\exists(f:A\to B),\forall x:A,T(x,f(x))\wedge(\forall x':A,R(x,x')\to f(x)=f(x')).\end{aligned}$$

Определение 6.28. SFC – частный случай SetoidFunctionalChoice, где $A := \pi_1(S)$ и $B := \pi_1(\bigcup S)$.

```
Definition SFC:= axSFC AC_A AC_B.
```

Определение 6.29. Afp – частный случай SFC, где R:=AC_R, T:=AC_T.

```
Definition Afp := ex2sig (SFC AC_R AC_T AC_eqvR T_sound AC_hyp).
Definition Afu := fun v : pi1 S =>
 OrdPair (pi2 S v) (pi2 (Union S) ((proj1_sig Afp) v)).
Definition Achfu : Ens := (sup (pi1 S) Afu).
```

Теорема 6.50. Теоретико-множественное отношение выбора Achfu – тотальное.

```
Theorem Achfu_total (X:Ens) (G:IN X S): exists Q, IN (OrdPair X Q) Achfu /\ IN Q X.
Proof.
pose (t:=in2term S X G).
pose (p:=Afu t).
unfold Afu in p.
exists (pi2 (Union S) (proj1_sig Afp t)).
```

```
split.
{ unfold Achfu.
  unfold IN.
exists t.
unfold t.
apply OrdPair_sound_both.
apply goodlem.
apply EQ_refl. }
{
pose (Y:= proj2_sig Afp t).
unfold AC_T in Y.
destruct Y as [Y1 Y2].
eapply IN_sound_right.
2 : exact Y1.
unfold t.
apply EQ_sym.
apply goodlem. }
Defined.
```

Лемма 6.51. Графически равные множества – экстенсионально равны.

```
Lemma eq2EQ (E1 E2:Ens) (K:E1=E2): EQ E1 E2.
Proof. destruct K. apply EQ_refl. Defined.
```

Теорема 6.52. Теоретико-множественное отношение выбора Achfu – функциональное.

```
Theorem Achfu_func : forall X:Ens, (IN X S) ->
(forall Q1 Q2, IN (OrdPair X Q1) Achfu /\ IN (OrdPair X Q2) Achfu -> EQ Q1 Q2).
Proof.
intros X W Q1 Q2 [K1 K2].
unfold Achfu, IN in K1,K2.
unfold Afu in K1,K2.
destruct K1 as [y1 K1], K2 as [y2 K2].

apply OrdPair_inj in K1 as [L1 R1].
apply OrdPair_inj in K2 as [L2 R2].
eapply EQ_tran. exact R1.
eapply EQ_tran. 2 : { apply EQ_sym. exact R2. }
pose (J:=(proj2_sig Afp)).

apply eq2EQ.
apply f_equal.

simpl in J. unfold AC_T in J.
destruct (J y1) as [_ QR].
apply (QR y2).
unfold AC_R.
eapply EQ_tran. apply EQ_sym. exact L1. exact L2.
Defined.
```

Теорема 6.53. Для любого множества S, не содержащего пустого множества, существует функция $f : S \to \bigcup S$, такая что $\forall a \in S (f(a) \in a)$.

```
Theorem axChoice : exists f:Ens,
forall X, IN X S -> (* '<->' for the restriction of f on S *)
((exists Q, IN (OrdPair X Q) f /\ IN Q X) /\
 (forall Q1 Q2, IN (OrdPair X Q1) f /\ IN (OrdPair X Q2) f -> EQ Q1 Q2)).
Proof.
exists Achfu.
intros X.
 intro G.
+ split.
  - (* totality of the relation: existence of the ordered pair *)
    apply Achfu_total with (1:=G).
```

```
  – (* functionality of the relation *)
    apply Achfu_func with (1:=G).
Defined.

End AC_sec.
```

Таким образом, при принятых допущениях, завершено обоснование того, что для термов типа Ens выполняются аксиомы теории множеств Цермело-Френкеля с аксиомой выбора.

7 Результаты

Следующие цели были достигнуты:

* Часть I

1. Определены четыре семантики логики высказываний в Coq, семантика двойного отрицания, булева семантика, классическая семантика и семантика Крипке.
2. Верифицированы результаты о семантике Крипке(принцип сохранения истинности, теорема о дедукции и корректность семантик).
3. Верифицированы теорема о корректности для семантике двойного отрицания для классического исчисления высказываний.
4. Верифицированы классические результаты о булевой семантике исчисления высказываний (теорема о корректности, теорема о дедукции, теорема о полноте).

* Часть II

5. В Coq определена выводимость в интуиционистком исчислении предикатов с контекстами.
6. Верифицирована теорема о дедукции интуиционисткого исчисления предикатов.
7. Верифицирована теорема о корректности интуиционисткого исчисления предикатов.

* Часть III

8. Обновлено вложение теории множеств ZF в систему Coq. Каждая теорема была доказана заново, доказательства упрощены. Избавление от использования лишних неподвижных точек сократило доказательства и сделало их более понятными и упростило.
9. Доказана теоретико-множественная аксиома выбора, с использованием неконструктивной теоретико-типовой аксиомы выбора, то есть получена модель ZFC.
10. Таким образом, создан теоретико-множественный инструмент для доказательства классических теорем внутри Coq.

8 Список литературы

* [1] Верещагин Н. К., Шень А. “Лекции по математической логике и теории алгоритмов.” М: МЦНМО, 2002.
* [2] В.Н. Крупский, С.Л. Кузнецов, “Практикум по математической логике. COQ.”
* [3] Т. Йех “Введение в теорию множеств”
* [4] Т. Йех “Теория множеств”
* [5] Zuhair, `https://math.stackexchange.com/a/2659806/251394`
* [6] A. Blass, `https://math.stackexchange.com/a/2973879/251394`
* [7] P. Aczel, B. Werner `https://github.com/coq-contribs/zfc/blob/master/Replacement.v`
* [8] B. Werner “The encoding of Zermelo-Fraenkel Set Theory in Coq”, in the Proceedings of TACS’97.
* [9] Jean-Yves Girard. Interpretation fonctionelle et elimination des coupures de l’arithm etique d’ordre sup erior. PhD thesis, Universit e Paris VII, 1972)
* [10] `https://coq.inria.fr/refman/language/module-system.html#typing-modules`
* [11] Э.Мендельсон, “Введение в логику”.
* [12] В.Н. Крупский, В.Е. Плиско, “Математическая логика и теория алгоритмов”
* [13] FourColorRepo `https://github.com/tangentforks/FourColorTheorem`
* [14] Ejgallego, `https://stackoverflow.com/a/55665533/4944714`