Coq: построение и проверка математических доказательств на компьютере

Научно-исследовательский семинар по математической логике, 22 апреля 2020 г.

С. Л. Кузнецов

Математический институт им. В.А. Стеклова РАН, Москва
Московский государственный университет им. М.В. Ломоносова
Национальный исследовательский университет Высшая школа экономики, Москва

Disclaimer

Сегодняшний рассказ будет во многом описательным: практически невозможно за одну вводную лекцию изложить в деталях синтаксис нового и довольно странного языка программирования.

Терминология

- Система автоматического поиска вывода (**prover**): по формуле ψ проверяет, доказуема ли ψ в данной системе (выдаёт доказательство или ответ «недоказуемо»).

Терминология

- Система автоматического поиска вывода (**prover**): по формуле ψ проверяет, доказуема ли ψ в данной системе (выдаёт доказательство или ответ «недоказуемо»).
    - SAT-солверы: “выполнима ли данная булева формула?”

Терминология

- Система автоматического поиска вывода (**prover**): по формуле ψ проверяет, доказуема ли ψ в данной системе (выдаёт доказательство или ответ «недоказуемо»).
  - SAT-солверы: “выполнима ли данная булева формула?”

    (ψ выполнима $\iff$ $\neg\psi$ *недоказуема* в классической логике)

Терминология

- Система автоматического поиска вывода (**prover**): по формуле ψ проверяет, доказуема ли ψ в данной системе (выдаёт доказательство или ответ «недоказуемо»).
  - SAT-солверы: “выполнима ли данная булева формула?”

    (ψ выполнима $\iff$ $\neg\psi$ *недоказуема* в классической логике)
  - логики описаний (модальные логики для представления баз знаний): OWL, биомедицинская информатика, ...

Терминология

- Система автоматического поиска вывода (**prover**): по формуле ψ проверяет, доказуема ли ψ в данной системе (выдаёт доказательство или ответ «недоказуемо»).
  - SAT-солверы: “выполнима ли данная булева формула?”
    (ψ выполнима $\iff$ $\neg\psi$ *недоказуема* в классической логике)
  - логики описаний (модальные логики для представления баз знаний): OWL, биомедицинская информатика, ...
  - разрешимые теории 1-го порядка: теория действительных чисел, элементарная геометрия, ...

Терминология

- Система автоматического поиска вывода (**prover**): по формуле ψ проверяет, доказуема ли ψ в данной системе (выдаёт доказательство или ответ «недоказуемо»).
  - SAT-солверы: “выполнима ли данная булева формула?”
    (ψ выполнима $\iff$ $\neg\psi$ *недоказуема* в классической логике)
  - логики описаний (модальные логики для представления баз знаний): OWL, биомедицинская информатика, ...
  - разрешимые теории 1-го порядка: теория действительных чисел, элементарная геометрия, ...
  - субструктурные логики для математической лингвистики: Grail, CatLog, ...

Терминология

- Система автоматического поиска вывода (**prover**): по формуле ψ проверяет, доказуема ли ψ в данной системе (выдаёт доказательство или ответ «недоказуемо»).
  - SAT-солверы: “выполнима ли данная булева формула?”
    (ψ выполнима $\iff$ $\neg\psi$ *недоказуема* в классической логике)
  - логики описаний (модальные логики для представления баз знаний): OWL, биомедицинская информатика, ...
  - разрешимые теории 1-го порядка: теория действительных чисел, элементарная геометрия, ...
  - субструктурные логики для математической лингвистики: Grail, CatLog, ...
  - ...

Терминология

- Система автоматического поиска вывода (**prover**): по формуле ψ проверяет, доказуема ли ψ в данной системе (выдаёт доказательство или ответ «недоказуемо»).
  - SAT-солверы: “выполнима ли данная булева формула?”
    (ψ выполнима $\iff$ $\neg\psi$ *недоказуема* в классической логике)
  - логики описаний (модальные логики для представления баз знаний): OWL, биомедицинская информатика, ...
  - разрешимые теории 1-го порядка: теория действительных чисел, элементарная геометрия, ...
  - субструктурные логики для математической лингвистики: Grail, CatLog, ...
  - ...

Интересные математические теории — арифметика, теория множеств, ... — неразрешимы, что делает prover невозможным.

Терминология

- Программа проверки доказательств (**proof checker**): принимает в качестве входных данных формулу A **и** некоторый объект (текст) $\mathscr{P}$ и возвращает «да», если $\mathscr{P}$ является доказательством формулы A (в данной логической системе), и «нет» в противном случае.

Терминология

- Программа проверки доказательств (**proof checker**): принимает в качестве входных данных формулу A **и** некоторый объект (текст) $\mathscr{P}$ и возвращает «да», если $\mathscr{P}$ является доказательством формулы A (в данной логической системе), и «нет» в противном случае.
    - Возможны практически для любых логических систем (с конечными доказательствами), по определению понятия «доказательство».

Терминология

- Программа проверки доказательств (**proof checker**): принимает в качестве входных данных формулу A **и** некоторый объект (текст) $\mathscr{P}$ и возвращает «да», если $\mathscr{P}$ является доказательством формулы A (в данной логической системе), и «нет» в противном случае.
  - Возможны практически для любых логических систем (с конечными доказательствами), по определению понятия «доказательство».
  - Однако полезность таких систем ограничена, поскольку работа по построению доказательства $\mathscr{P}$ полностью возлагается на человека.

Терминология

- Золотая середина — программы интерактивного поиска доказательства (**proof assistant**): интерактивное программное обеспечения для разработки формального доказательства.

Терминология

- Золотая середина — программы интерактивного поиска доказательства (**proof assistant**): интерактивное программное обеспечения для разработки формального доказательства.
    - Agda, Coq, HOL/Isabelle, Mizar, ...

Терминология

- Золотая середина — программы интерактивного поиска доказательства (**proof assistant**): интерактивное программное обеспечения для разработки формального доказательства.
  - Agda, Coq, HOL/Isabelle, Mizar, ...
- Доказательство, полученное в ходе интерактивной работы, потом проверяется программой проверки доказательств.

Типичное формальное доказательство

Докажем, что φ влечёт φ.

Типичное формальное доказательство

Докажем, что φ влечёт φ.

(1)	$(\varphi \to ((\varphi \to \varphi) \to \varphi) \to ((\varphi \to (\varphi \to \varphi)) \to (\varphi \to \varphi))$	акс. 2 с $A = C = \varphi$, $B = (\varphi \to \varphi)$
(2)	$\varphi \to ((\varphi \to \varphi) \to \varphi)$	акс. 1 с $A = \varphi$, $B = (\varphi \to \varphi)$
(3)	$(\varphi \to (\varphi \to \varphi)) \to (\varphi \to \varphi)$	MP (2) (1)
(4)	$\varphi \to (\varphi \to \varphi)$	акс. 1 с $A = B = \varphi$
(5)	$\varphi \to \varphi$	MP (4) (3)

Типичное формальное доказательство

Докажем, что φ влечёт φ.

(1)	$(\varphi \to ((\varphi \to \varphi) \to \varphi) \to ((\varphi \to (\varphi \to \varphi)) \to (\varphi \to \varphi))$	акс. 2 с $A = C = \varphi$, $B = (\varphi \to \varphi)$
(2)	$\varphi \to ((\varphi \to \varphi) \to \varphi)$	акс. 1 с $A = \varphi$, $B = (\varphi \to \varphi)$
(3)	$(\varphi \to (\varphi \to \varphi)) \to (\varphi \to \varphi)$	MP (2) (1)
(4)	$\varphi \to (\varphi \to \varphi)$	акс. 1 с $A = B = \varphi$
(5)	$\varphi \to \varphi$	MP (4) (3)

Бррр...

Формальные доказательства

- **Pro:** проверяются формально и автоматически, что гарантирует отсутствие «человеческого фактора» (особенно для сложных рассуждений).

Формальные доказательства

- **Pro:** проверяются формально и автоматически, что гарантирует отсутствие «человеческого фактора» (особенно для сложных рассуждений).
- **Contra:** сложны в построении и нечитаемы человеком — в случае неудачно выбранной аксиоматической системы!

Формальные доказательства

- **Pro:** проверяются формально и автоматически, что гарантирует отсутствие «человеческого фактора» (особенно для сложных рассуждений).
- **Contra:** сложны в построении и нечитаемы человеком — в случае неудачно выбранной аксиоматической системы!
- Даже в случае *удачной* аксиоматики разработка («программирование») формального доказательства — это отдельная работа!

Формальные доказательства

- **Pro:** проверяются формально и автоматически, что гарантирует отсутствие «человеческого фактора» (особенно для сложных рассуждений).
- **Contra:** сложны в построении и нечитаемы человеком — в случае неудачно выбранной аксиоматической системы!
- Даже в случае *удачной* аксиоматики разработка («программирование») формального доказательства — это отдельная работа!
- «Pro» перевешивают «contra» в случаях:

Формальные доказательства

- **Pro:** проверяются формально и автоматически, что гарантирует отсутствие «человеческого фактора» (особенно для сложных рассуждений).
- **Contra:** сложны в построении и нечитаемы человеком — в случае неудачно выбранной аксиоматической системы!
- Даже в случае *удачной* аксиоматики разработка («программирование») формального доказательства — это отдельная работа!
- «Pro» перевешивают «contra» в случаях:
    - когда доказательства получены компьютерным перебором;

Формальные доказательства

- **Pro:** проверяются формально и автоматически, что гарантирует отсутствие «человеческого фактора» (особенно для сложных рассуждений).
- **Contra:** сложны в построении и нечитаемы человеком — в случае неудачно выбранной аксиоматической системы!
- Даже в случае *удачной* аксиоматики разработка («программирование») формального доказательства — это отдельная работа!
- «Pro» перевешивают «contra» в случаях:
  - когда доказательства получены компьютерным перебором;
  - когда доказательство написано человеком, но в силу объёма и специфики не может быть разобрано *типичным* математиком;

Формальные доказательства

- **Pro:** проверяются формально и автоматически, что гарантирует отсутствие «человеческого фактора» (особенно для сложных рассуждений).
- **Contra:** сложны в построении и нечитаемы человеком — в случае неудачно выбранной аксиоматической системы!
- Даже в случае *удачной* аксиоматики разработка («программирование») формального доказательства — это отдельная работа!
- «Pro» перевешивают «contra» в случаях:
  - когда доказательства получены компьютерным перебором;
  - когда доказательство написано человеком, но в силу объёма и специфики не может быть разобрано *типичным* математиком;
  - верификации программного обеспечения.

Система Coq

Основные факты:

Система Coq

Основные факты:

- Разрабатывается в INRIA, Франция, с 1989 г.

Система Coq

Основные факты:

- Разрабатывается в INRIA, Франция, с 1989 г.
- Свободное программное обеспечение (LGPL 2.1), кросс-платформенное, доступно на сайте `https://coq.inria.fr/`

Система Coq

Основные факты:

- Разрабатывается в INRIA, Франция, с 1989 г.
- Свободное программное обеспечение (LGPL 2.1), кросс-платформенное, доступно на сайте `https://coq.inria.fr/`
- 2013 ACM Software System Award (T. Coquand et al.)

Система Coq

Основные факты:

- Разрабатывается в INRIA, Франция, с 1989 г.
- Свободное программное обеспечение (LGPL 2.1), кросс-платформенное, доступно на сайте `https://coq.inria.fr/`
- 2013 ACM Software System Award (T. Coquand et al.)
- Система написана на языке OCaml.

Система Coq

Основные факты:

- Разрабатывается в INRIA, Франция, с 1989 г.
- Свободное программное обеспечение (LGPL 2.1), кросс-платформенное, доступно на сайте `https://coq.inria.fr/`
- 2013 ACM Software System Award (T. Coquand et al.)
- Система написана на языке OCaml.
- Coq реализует **систему типов высших порядков**, которую также можно рассматривать как **функциональный язык программирования с зависимыми типами**

Система Coq

Основные факты:

- Разрабатывается в INRIA, Франция, с 1989 г.
- Свободное программное обеспечение (LGPL 2.1), кросс-платформенное, доступно на сайте `https://coq.inria.fr/`
- 2013 ACM Software System Award (T. Coquand et al.)
- Система написана на языке OCaml.
- Coq реализует **систему типов высших порядков**, которую также можно рассматривать как **функциональный язык программирования с зависимыми типами** (соответствие Карри – Говарда).
- Доказательства в coq не классические, а **интуиционистские (конструктивные).**

Применения Coq

- **Математические:** формализация сложных доказательств математических теорем

Применения Coq

- **Математические:** формализация сложных доказательств математических теорем
  - Знаменитый пример: **теорема о четырёх красках,** *любой планарный граф можно раскрасить в 4 цвета так, что смежные вершины разноцветны.*

Применения Coq

- **Математические:** формализация сложных доказательств математических теорем
  - Знаменитый пример: **теорема о четырёх красках,** *любой планарный граф можно раскрасить в 4 цвета так, что смежные вершины разноцветны.*
    Доказательство формализовал в Coq Г. Гонтье (Microsoft Research) в 2005 г.

Применения Coq

- **Математические:** формализация сложных доказательств математических теорем
  - Знаменитый пример: **теорема о четырёх красках,** *любой планарный граф можно раскрасить в 4 цвета так, что смежные вершины разноцветны.*
    Доказательство формализовал в Coq Г. Гонтье (Microsoft Research) в 2005 г.
  - Другой пример: теорема Фейта–Томпсона о группах нечётного порядка.

Применения Coq

- **Математические:** формализация сложных доказательств математических теорем
  - Знаменитый пример: **теорема о четырёх красках,** *любой планарный граф можно раскрасить в 4 цвета так, что смежные вершины разноцветны.*
    Доказательство формализовал в Coq Г. Гонтье (Microsoft Research) в 2005 г.
  - Другой пример: теорема Фейта–Томпсона о группах нечётного порядка.
    Формализовал Г. Гонтье в 2012 г.

Применения Coq

- **Программирование:** *верификация* программного обеспечения, т.е. формальное доказательство его соответствия формально заданной спецификации

Применения Coq

- **Программирование:** *верификация* программного обеспечения, т.е. формальное доказательство его соответствия формально заданной спецификации
  ... и даже *извлечение* работающего и верифицированного кода из интуиционистских доказательств!

Применения Coq

- **Программирование:** *верификация* программного обеспечения, т.е. формальное доказательство его соответствия формально заданной спецификации
  ... и даже *извлечение* работающего и верифицированного кода из интуиционистских доказательств!
  - Пример: CompCert, сертифицированный компилятор C (К. Леруа, 2008).

Применения Coq

- **Программирование:** *верификация* программного обеспечения, т.е. формальное доказательство его соответствия формально заданной спецификации
  ... и даже *извлечение* работающего и верифицированного кода из интуиционистских доказательств!
  - Пример: CompCert, сертифицированный компилятор C (К. Леруа, 2008).
  - В целом, верификация гарантирует большую надёжность, чем тестирование (но всё равно не абсолютную).

Конструктивные и неконструктивные доказательства

Утверждение: существуют такие иррациональные числа α и β, что α^β рационально.

Конструктивные и неконструктивные доказательства

Утверждение: существуют такие иррациональные числа α и β, что α^β рационально.

- Конструктивное доказательство: $\alpha = \sqrt{2}$, $\beta = \log_{\sqrt{2}} 3$.

Конструктивные и неконструктивные доказательства

Утверждение: существуют такие иррациональные числа α и β, что α^β рационально.

- Конструктивное доказательство: $\alpha = \sqrt{2}$, $\beta = \log_{\sqrt{2}} 3$.
- Неконструктивное доказательство: рассмотрим два случая:

  1. $\sqrt{2}^{\sqrt{2}}$ рационально. Тогда возьмём $\alpha = \beta = \sqrt{2}$.
  2. $\sqrt{2}^{\sqrt{2}}$ иррационально. Тогда возьмём $\alpha = \sqrt{2}^{\sqrt{2}}$ и $\beta = \sqrt{2}$ ($\alpha^\beta = 2$ рационально).

Интуиционистская логика

- Интуиционистская логика получается из классической исключением аксиомы $A \vee \neg A$ (закона исключённого третьего).

Интуиционистская логика

- Интуиционистская логика получается из классической исключением аксиомы $A \vee \neg A$ (закона исключённого третьего).
- В логике без этого закона неконструктивные доказательства невозможны.

Интуиционистская логика

- Интуиционистская логика получается из классической исключением аксиомы $A \vee \neg A$ (закона исключённого третьего).
- В логике без этого закона неконструктивные доказательства невозможны.
- Если в Coq нужно записать неклассическое доказательство, закон исключённого третьего добавляют как дополнительную аксиому (“`classic`”).

Конструктивные доказательства и λ-термы

- **Соответствие Карри – Говарда:**
  доказательства = объекты и программы;
  теоремы = типы данных.

Конструктивные доказательства и λ-термы

- **Соответствие Карри – Говарда:**
  доказательства = объекты и программы;
  теоремы = типы данных.
- Доказать $A \longrightarrow B$ значит построить функцию из A в B (из типа доказательств A в тип доказательств B).

Конструктивные доказательства и λ-термы

- **Соответствие Карри – Говарда:**
  доказательства = объекты и программы;
  теоремы = типы данных.
- Доказать $A \longrightarrow B$ значит построить функцию из A в B (из типа доказательств A в тип доказательств B).
- Доказать $(\forall x \in A)\, B(x)$ — построить функцию, по $a \in A$ возвращающую объект типа $B(a)$ *(зависимый тип)*.

Конструктивные доказательства и λ-термы

- **Соответствие Карри – Говарда:**
  доказательства = объекты и программы;
  теоремы = типы данных.
- Доказать $A \longrightarrow B$ значит построить функцию из A в B (из типа доказательств A в тип доказательств B).
- Доказать $(\forall x \in A)\, B(x)$ — построить функцию, по $a \in A$ возвращающую объект типа $B(a)$ *(зависимый тип).*
- Доказательство $(\exists x \in A)\, B(x)$ — это пара из объекта $a \in A$ и объекта типа $B(a)$ (доказательства $B(a)$).

Конструктивные доказательства и λ-термы

- **Соответствие Карри – Говарда:**
  доказательства = объекты и программы;
  теоремы = типы данных.
- Доказать $A \longrightarrow B$ значит построить функцию из A в B (из типа доказательств A в тип доказательств B).
- Доказать $(\forall x \in A)\, B(x)$ — построить функцию, по $a \in A$ возвращающую объект типа $B(a)$ *(зависимый тип).*
- Доказательство $(\exists x \in A)\, B(x)$ — это пара из объекта $a \in A$ и объекта типа $B(a)$ (доказательства $B(a)$).
- и т.д.

Конструктивные доказательства и λ-термы

- **Соответствие Карри – Говарда:**
  доказательства = объекты и программы;
  теоремы = типы данных.
- Доказать $A \longrightarrow B$ значит построить функцию из A в B (из типа доказательств A в тип доказательств B).
- Доказать $(\forall x \in A)\, B(x)$ — построить функцию, по $a \in A$ возвращающую объект типа $B(a)$ *(зависимый тип).*
- Доказательство $(\exists x \in A)\, B(x)$ — это пара из объекта $a \in A$ и объекта типа $B(a)$ (доказательства $B(a)$).
- и т.д.
- Пример: доказательство $(A \longrightarrow B) \longrightarrow ((B \longrightarrow C) \longrightarrow (A \longrightarrow C))$ записывается следующим *λ-термом:*

$$\lambda f.\lambda g.\lambda x.g(f(x))$$

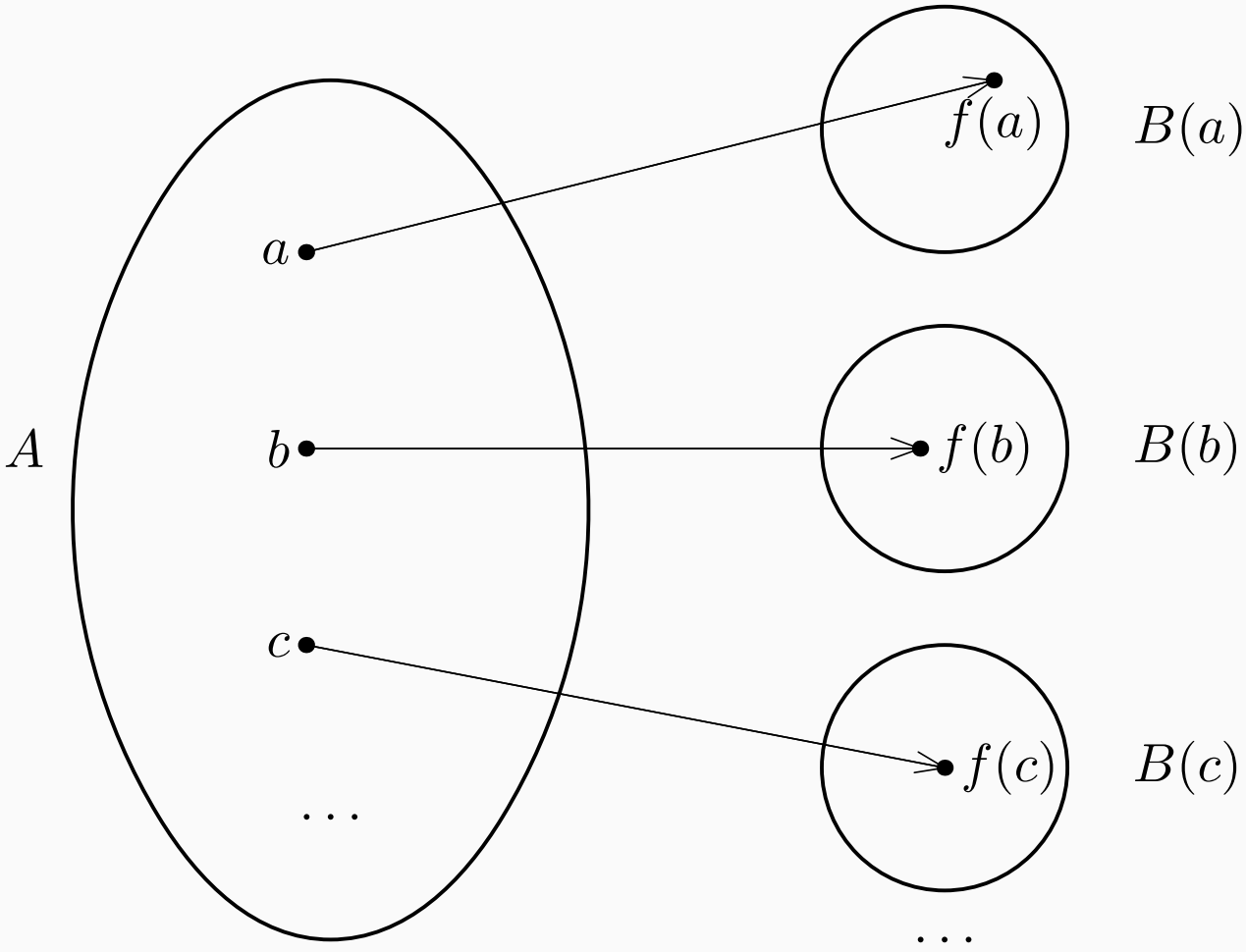
Конструктивные доказательства и λ-термы

- **Соответствие Карри – Говарда:**
  доказательства = объекты и программы;
  теоремы = типы данных.
- Доказать $A \longrightarrow B$ значит построить функцию из A в B (из типа доказательств A в тип доказательств B).
- Доказать $(\forall x \in A)\, B(x)$ — построить функцию, по $a \in A$ возвращающую объект типа $B(a)$ *(зависимый тип).*
- Доказательство $(\exists x \in A)\, B(x)$ — это пара из объекта $a \in A$ и объекта типа $B(a)$ (доказательства $B(a)$).
- и т.д.
- Пример: доказательство $(A \longrightarrow B) \longrightarrow ((B \longrightarrow C) \longrightarrow (A \longrightarrow C))$ записывается следующим *λ-термом:*

$$\lambda f.\lambda g.\lambda x.g(f(x))$$

  - Это — абстрактная программа для композиции функций.

Зависимый тип функций



Языки Coq, Calculus of Inductive Constructions

- Сам Coq написан на языке OCaml.

Языки Coq, Calculus of Inductive Constructions

- Сам Coq написан на языке OCaml.
- Внутренний язык Coq называется “Gallina” и реализует λ-исчисление поверх развитой системы типов (с зависимыми и индуктивными типами высших порядков).

Языки Coq, Calculus of Inductive Constructions

- Сам Coq написан на языке OCaml.
- Внутренний язык Coq называется “Gallina” и реализует λ-исчисление поверх развитой системы типов (с зависимыми и индуктивными типами высших порядков).
- Простейший пример индуктивного типа — тип натуральных чисел ($0 \in \mathbb{N}$; $n \in \mathbb{N} \Longrightarrow S(n) = n + 1 \in \mathbb{N}$). Более сложные — например, конечные деревья.

Языки Coq, Calculus of Inductive Constructions

- Сам Coq написан на языке OCaml.
- Внутренний язык Coq называется “Gallina” и реализует λ-исчисление поверх развитой системы типов (с зависимыми и индуктивными типами высших порядков).
- Простейший пример индуктивного типа — тип натуральных чисел ($0 \in \mathbb{N}$; $n \in \mathbb{N} \Longrightarrow S(n) = n + 1 \in \mathbb{N}$). Более сложные — например, конечные деревья.
- Теоретически, можно программировать доказательства непосредственно как λ-термы (команда `exact`).

Языки Coq, Calculus of Inductive Constructions

- Сам Coq написан на языке OCaml.
- Внутренний язык Coq называется “Gallina” и реализует λ-исчисление поверх развитой системы типов (с зависимыми и индуктивными типами высших порядков).
- Простейший пример индуктивного типа — тип натуральных чисел ($0 \in \mathbb{N}$; $n \in \mathbb{N} \Longrightarrow S(n) = n + 1 \in \mathbb{N}$). Более сложные — например, конечные деревья.
- Теоретически, можно программировать доказательства непосредственно как λ-термы (команда `exact`).
- В реальности используются *тактики*, реализующие поиск вывода «снизу вверх» (от цели к аксиомам).

Тактики и естественный вывод

- Работа с Coq'ом интерактивна.

Тактики и естественный вывод

- Работа с Coq'ом интерактивна.
- В каждый момент имеется несколько *задач на доказательство* (т.е. построение терма) вида $x_1 : A_1, \ldots, x_n : A_n \vdash \mathbf{(???)} : B$.

Тактики и естественный вывод

- Работа с Coq'ом интерактивна.
- В каждый момент имеется несколько *задач на доказательство* (т.е. построение терма) вида $x_1 : A_1, \ldots, x_n : A_n \vdash \mathbf{(???)} : B$.
- A_i — посылки, B — требуемое заключение.

Тактики и естественный вывод

- Работа с Coq'ом интерактивна.
- В каждый момент имеется несколько *задач на доказательство* (т.е. построение терма) вида $x_1 : A_1, \ldots, x_n : A_n \vdash \mathbf{(???)} : B$.
- A_i — посылки, B — требуемое заключение.
- Если $B = A_i$, доказывать ничего не надо.

Тактики и естественный вывод

- Работа с Coq'ом интерактивна.
- В каждый момент имеется несколько *задач на доказательство* (т.е. построение терма) вида $x_1 : A_1, \ldots, x_n : A_n \vdash \mathbf{(???)} : B$.
- A_i — посылки, B — требуемое заключение.
- Если $B = A_i$, доказывать ничего не надо.
- Применение тактики призвано *упростить* задачу поиска вывода.

Тактики и естественный вывод

- Работа с Coq'ом интерактивна.
- В каждый момент имеется несколько *задач на доказательство* (т.е. построение терма) вида $x_1 : A_1, \ldots, x_n : A_n \vdash \mathbf{(???)} : B$.
- A_i — посылки, B — требуемое заключение.
- Если $B = A_i$, доказывать ничего не надо.
- Применение тактики призвано *упростить* задачу поиска вывода.
- Например, от задачи $\Gamma \vdash (B \longrightarrow C)$ можно перейти к $\Gamma, B \vdash C$, а от задачи $\Gamma \vdash B_1 \vee B_2$ — к одной из задач $\Gamma \vdash B_i$ ($i = 1$ или 2).

Тактики и естественный вывод

- Работа с Coq'ом интерактивна.
- В каждый момент имеется несколько *задач на доказательство* (т.е. построение терма) вида $x_1 : A_1, \ldots, x_n : A_n \vdash \mathbf{(???)} : B$.
- A_i — посылки, B — требуемое заключение.
- Если $B = A_i$, доказывать ничего не надо.
- Применение тактики призвано *упростить* задачу поиска вывода.
- Например, от задачи $\Gamma \vdash (B \longrightarrow C)$ можно перейти к $\Gamma, B \vdash C$, а от задачи $\Gamma \vdash B_1 \vee B_2$ — к одной из задач $\Gamma \vdash B_i$ ($i = 1$ или 2).
- Простейшие тактики — это правила вывода CIC.

Calculus of Constructions

Секвенции CoC: $x_1 : A_1, \ldots, x_n : A_n \vdash u : B$;
при $j > i$ тип A_j может зависеть от x_i.

Универсумы (сорта): $\mathrm{Prop} \prec \mathrm{Set} \prec \mathrm{Type}_1 \prec \ldots \prec \mathrm{Type}_n \prec \ldots$

Правила вывода: [C. Paulin-Mohring. Introduction to the calculus of inductive constructions. In: All about Proofs, Proofs for All, College Publications, 2015.]

$$\frac{\Gamma \vdash}{\Gamma \vdash \mathrm{Prop} : \mathrm{Type}_1} \qquad \frac{\Gamma \vdash}{\Gamma \vdash \mathrm{Set} : \mathrm{Type}_1} \qquad \frac{\Gamma \vdash}{\Gamma \vdash \mathrm{Type}_i : \mathrm{Type}_{i+1}}$$

$$\frac{(x : A) \in \Gamma \quad \Gamma \vdash}{\Gamma \vdash x : A} \qquad \frac{\Gamma \vdash A : s \quad x \notin \Gamma \quad s \in \mathcal{S}}{\Gamma, x : A \vdash} \qquad \frac{\Gamma, x : A \vdash u : B}{\Gamma \vdash (\lambda x^A.u) : \prod_{x:A} B}$$

$$\frac{\Gamma \vdash f : \prod_{x:A} B \quad \Gamma \vdash a : A}{\Gamma \vdash (fa) : B[x := a]} \qquad \frac{\Gamma, x : A \vdash B : \mathrm{Prop}}{\Gamma \vdash (\prod_{x:A} B) : \mathrm{Prop}} \qquad \frac{\Gamma, x : A \vdash B : \mathrm{Set}}{\Gamma \vdash (\prod_{x:A} B) : \mathrm{Set}}$$

$$\frac{\Gamma, x : A \vdash B : \mathrm{Type}_i \quad \Gamma \vdash A : \mathrm{Type}_i}{\Gamma \vdash (\prod_{x:A} B) : \mathrm{Type}_i} \qquad \frac{A \preceq B \quad \Gamma \vdash u : A}{\Gamma \vdash u : B}$$

Индуктивные типы

- Вторая базовая конструкция типов в Coq — индуктивные определения типов.

Индуктивные типы

- Вторая базовая конструкция типов в Coq — индуктивные определения типов.
- Стандартный пример — натуральные числа.

Индуктивные типы

- Вторая базовая конструкция типов в Coq — индуктивные определения типов.
- Стандартный пример — натуральные числа.
- Более интересный пример — конечные списки.

```
Inductive list (A : Type) : Type :=
nil : list A
| cons : A -> list A -> list A
```

Индуктивные типы

- Вторая базовая конструкция типов в Coq — индуктивные определения типов.
- Стандартный пример — натуральные числа.
- Более интересный пример — конечные списки.

```
Inductive list (A : Type) : Type :=
nil : list A
| cons : A -> list A -> list A
```

- На индуктивных типах можно определять функции конструкцией `fix` (неподвижная точка, рекурсор).

Индуктивные типы

- Вторая базовая конструкция типов в Coq — индуктивные определения типов.
- Стандартный пример — натуральные числа.
- Более интересный пример — конечные списки.

```
Inductive list (A : Type) : Type :=
nil : list A
| cons : A -> list A -> list A
```

- На индуктивных типах можно определять функции конструкцией `fix` (неподвижная точка, рекурсор).
- В расширении Coq есть также *коиндуктивные* типы.

Calculus of Inductive Constructions

- CIC — базовое исчисление Coq.

Calculus of Inductive Constructions

- CIC — базовое исчисление Coq.
- Как версия λ-исчисления, CIC обладает свойством слабой нормализуемости (WN).

Calculus of Inductive Constructions

- CIC — базовое исчисление Coq.
- Как версия λ-исчисления, CIC обладает свойством слабой нормализуемости (WN).
- Сильной нормализуемости (SN) по техническим причинам нет.

Calculus of Inductive Constructions

- CIC — базовое исчисление Coq.
- Как версия λ-исчисления, CIC обладает свойством слабой нормализуемости (WN).
- Сильной нормализуемости (SN) по техническим причинам нет.
- CIC взаимно интерпретируемо с теорией множеств ZFC + существование бесконечного числа недостижимых кардиналов.

  B. Werner. Sets in types, types in sets (TACS 1997)

Calculus of Inductive Constructions

- В CIC всего два конструктора типов: зависимое произведение и индуктивные определения.

Calculus of Inductive Constructions

- В CIC всего два конструктора типов: зависимое произведение и индуктивные определения.
- Остальные логические операции, кроме $\forall$ и $\longrightarrow$, определяются «индуктивно».

Calculus of Inductive Constructions

- В CIC всего два конструктора типов: зависимое произведение и индуктивные определения.
- Остальные логические операции, кроме $\forall$ и $\longrightarrow$, определяются «индуктивно».
- `Inductive or (A B : Prop) : Prop :=`
  `or_introl : A -> A \/ B | or_intror : B -> A \/ B`

Calculus of Inductive Constructions

- В CIC всего два конструктора типов: зависимое произведение и индуктивные определения.
- Остальные логические операции, кроме $\forall$ и $\longrightarrow$, определяются «индуктивно».
- `Inductive or (A B : Prop) : Prop :=`
  `or_introl : A -> A \/ B | or_intror : B -> A \/ B`
- `Inductive ex (A : Type) (P : A -> Prop) : Prop :=`
  `ex_intro : forall x : A, P x -> exists y, P y`

Тактики

- Есть и более сложные тактики.

Тактики

- Есть и более сложные тактики.
- Например, $x + y = 5, x + 2y = 7 \vdash y = 2$ доказывается (полностью) применением тактики omega.

Тактики

- Есть и более сложные тактики.
- Например, $x + y = 5, x + 2y = 7 \vdash y = 2$ доказывается (полностью) применением тактики omega.
- Тактика omega реализует поиск вывод в бесквантворной арифметике Пресбургера.

Тактики

- Есть и более сложные тактики.
- Например, $x + y = 5, x + 2y = 7 \vdash y = 2$ доказывается (полностью) применением тактики omega.
- Тактика omega реализует поиск вывод в бесквантворной арифметике Пресбургера.
- Язык тактик развивается, каждый может создать свои тактики.

Тактики

- Есть и более сложные тактики.
- Например, $x + y = 5, x + 2y = 7 \vdash y = 2$ доказывается (полностью) применением тактики omega.
- Тактика omega реализует поиск вывод в бесквантворной арифметике Пресбургера.
- Язык тактик развивается, каждый может создать свои тактики.
- Не ставит ли это под угрозу безошибочность доказательств?

Тактики

- Есть и более сложные тактики.
- Например, $x + y = 5, x + 2y = 7 \vdash y = 2$ доказывается (полностью) применением тактики omega.
- Тактика omega реализует поиск вывод в бесквантворной арифметике Пресбургера.
- Язык тактик развивается, каждый может создать свои тактики.
- Не ставит ли это под угрозу безошибочность доказательств?
- Тактика работает в два прохода:

Тактики

- Есть и более сложные тактики.
- Например, $x + y = 5, x + 2y = 7 \vdash y = 2$ доказывается (полностью) применением тактики omega.
- Тактика omega реализует поиск вывод в бесквантворной арифметике Пресбургера.
- Язык тактик развивается, каждый может создать свои тактики.
- Не ставит ли это под угрозу безошибочность доказательств?
- Тактика работает в два прохода:
  1. По исходной цели выдаёт новые подцели.

Тактики

- Есть и более сложные тактики.
- Например, $x + y = 5, x + 2y = 7 \vdash y = 2$ доказывается (полностью) применением тактики omega.
- Тактика omega реализует поиск вывод в бесквантворной арифметике Пресбургера.
- Язык тактик развивается, каждый может создать свои тактики.
- Не ставит ли это под угрозу безошибочность доказательств?
- Тактика работает в два прохода:
  1. По исходной цели выдаёт новые подцели.
  2. После завершения доказательства подцелей собирает из полученных λ-термов λ-терм для исходной цели.

Тактики

- Есть и более сложные тактики.
- Например, $x + y = 5, x + 2y = 7 \vdash y = 2$ доказывается (полностью) применением тактики omega.
- Тактика omega реализует поиск вывод в бесквантворной арифметике Пресбургера.
- Язык тактик развивается, каждый может создать свои тактики.
- Не ставит ли это под угрозу безошибочность доказательств?
- Тактика работает в два прохода:
  1. По исходной цели выдаёт новые подцели.
  2. После завершения доказательства подцелей собирает из полученных λ-термов λ-терм для исходной цели.
- При этом финальный λ-терм проверяется на корректность ядром Coq (которое небольшое и не меняется).

Двойственность и извлечение кода

- Квантор существования $(\exists x \in A)\, B(x)$ двойствен теоретико-множественной свёртке $\{x \in A \mid B(x)\}$.

Двойственность и извлечение кода

- Квантор существования $(\exists x \in A)\, B(x)$ двойствен теоретико-множественной свёртке $\{x \in A \mid B(x)\}$.
- Квантор всеобщности $(\forall x \in A)\, B(x)$ — уточнённый тип функций $f : A \longrightarrow \bigcup B(x)$, с условием *(спецификацией)* $f(a) \in B(a)$ для каждого $a \in A$.

Двойственность и извлечение кода

- Квантор существования $(\exists x \in A)\, B(x)$ двойствен теоретико-множественной свёртке $\{x \in A \mid B(x)\}$.
- Квантор всеобщности $(\forall x \in A)\, B(x)$ — уточнённый тип функций $f : A \longrightarrow \bigcup B(x)$, с условием *(спецификацией)* $f(a) \in B(a)$ для каждого $a \in A$.
- $\forall\exists$-формула специфицирует функцию: $(\forall x \in A)\, (\exists y \in B)\, S(x, y)$.

Двойственность и извлечение кода

- Квантор существования $(\exists x \in A)\, B(x)$ двойствен теоретико-множественной свёртке $\{x \in A \mid B(x)\}$.
- Квантор всеобщности $(\forall x \in A)\, B(x)$ — уточнённый тип функций $f : A \longrightarrow \bigcup B(x)$, с условием *(спецификацией)* $f(a) \in B(a)$ для каждого $a \in A$.
- $\forall\exists$-формула специфицирует функцию: $(\forall x \in A)\,(\exists y \in B)\, S(x, y)$.
- Каждое доказательство содержит вычислимую функцию $f : A \longmapsto B$, причём гарантировано выполнение спецификации $S(a, f(a))$ для каждого $a \in A$.

Двойственность и извлечение кода

- Пример: наибольший общий делитель

$$\mathrm{GCD} : (\forall a, b \in \mathbb{Z})\{d \in \mathbb{Z} \mid (a \vdots d) \wedge (b \vdots d) \wedge$$
$$\wedge (\forall d' \in \mathbb{Z})((a \vdots d') \wedge (b \vdots d') \longrightarrow (d \vdots d'))\}.$$

Двойственность и извлечение кода

- Пример: наибольший общий делитель

$$\mathrm{GCD} \;:\; (\forall a, b \in \mathbb{Z})\,\{d \in \mathbb{Z} \mid (a \vdots d) \wedge (b \vdots d) \wedge \\ \wedge\, (\forall d' \in \mathbb{Z})\,((a \vdots d') \wedge (b \vdots d') \longrightarrow (d \vdots d'))\}.$$

- Спецификация не задаёт результат однозначно (в данном случае годятся d и $-d$).

Двойственность и извлечение кода

- Пример: наибольший общий делитель

$$\mathrm{GCD} : (\forall a, b \in \mathbb{Z})\,\{d \in \mathbb{Z} \mid (a \vdots d) \wedge (b \vdots d) \wedge \\ \wedge\, (\forall d' \in \mathbb{Z})\,((a \vdots d') \wedge (b \vdots d') \longrightarrow (d \vdots d'))\}.$$

- Спецификация не задаёт результат однозначно (в данном случае годятся d и $-d$).
- Coq искусственно разделяет *построения* (универсум «Set») и *рассуждения* (универсум «Prop»).

Двойственность и извлечение кода

- Пример: наибольший общий делитель

$$\mathrm{GCD} : (\forall a, b \in \mathbb{Z})\,\{d \in \mathbb{Z} \mid (a \vdots d) \wedge (b \vdots d) \wedge \\ \wedge\, (\forall d' \in \mathbb{Z})\,((a \vdots d') \wedge (b \vdots d') \longrightarrow (d \vdots d'))\}.$$

- Спецификация не задаёт результат однозначно (в данном случае годятся d и $-d$).
- Coq искусственно разделяет *построения* (универсум «Set») и *рассуждения* (универсум «Prop»).
- При извлечении программного кода Prop-часть уничтожается: остаётся только функция $a, b \longmapsto d$, исчезает доказательство спецификации.

Построения и рассуждения

- Set-часть должна быть чисто интуиционистской.

Построения и рассуждения

- Set-часть должна быть чисто интуиционистской.
- Если мы допустим аксиому $A \vee \neg A$ (здесь $\vee : \mathrm{Prop} \times \mathrm{Prop} \longrightarrow \mathrm{Set}$), то нам придётся её *реализовать:* написать функцию разбора случаев.

Построения и рассуждения

- Set-часть должна быть чисто интуиционистской.
- Если мы допустим аксиому $A \vee \neg A$ (здесь $\vee : \mathrm{Prop} \times \mathrm{Prop} \longrightarrow \mathrm{Set}$), то нам придётся её *реализовать:* написать функцию разбора случаев.
- Например: $(\forall x, y \in \mathbb{Z})\,((x < y) \vee (y \leq x))$.

Построения и рассуждения

- Set-часть должна быть чисто интуиционистской.
- Если мы допустим аксиому $A \vee \neg A$ (здесь $\vee : \text{Prop} \times \text{Prop} \longrightarrow \text{Set}$), то нам придётся её *реализовать:* написать функцию разбора случаев.
- Например: $(\forall x, y \in \mathbb{Z})\,((x < y) \vee (y \leq x))$.
- В Prop-части (рассуждениях) можно использовать `classic`.

Построения и рассуждения

- Set-часть должна быть чисто интуиционистской.
- Если мы допустим аксиому $A \vee \neg A$ (здесь $\vee : \text{Prop} \times \text{Prop} \longrightarrow \text{Set}$), то нам придётся её *реализовать:* написать функцию разбора случаев.
- Например: $(\forall x, y \in \mathbb{Z})\,((x < y) \vee (y \leq x))$.
- В Prop-части (рассуждениях) можно использовать `classic`.
- Prop-рассуждения *непрозрачны:* их детали нельзя использовать в Set-части.

Построения и рассуждения

- Set-часть должна быть чисто интуиционистской.
- Если мы допустим аксиому $A \vee \neg A$ (здесь $\vee : \text{Prop} \times \text{Prop} \longrightarrow \text{Set}$), то нам придётся её *реализовать:* написать функцию разбора случаев.
- Например: $(\forall x, y \in \mathbb{Z})\,((x < y) \vee (y \leq x))$.
- В Prop-части (рассуждениях) можно использовать `classic`.
- Prop-рассуждения *непрозрачны:* их детали нельзя использовать в Set-части.
- Таким образом достигается принцип “proof irrelevance”: доказательства можно заменять, не меняя код.

Prop vs. Set

- Иногда разделение Prop и Set неудобно.

Prop vs. Set

- Иногда разделение Prop и Set неудобно.
- Пример из современного решения задачи 4 красок (https://github.com/math-comp/fourcolor)

```
Theorem four_color_hypermap : forall G, planar_bridgeless G ->
four_colorable G.
```

```
Definition four_colorable := exists k, coloring k.
```

Prop vs. Set

- Иногда разделение Prop и Set неудобно.
- Пример из современного решения задачи 4 красок (https://github.com/math-comp/fourcolor)

  ```
  Theorem four_color_hypermap : forall G, planar_bridgeless G ->
  four_colorable G.
  ```

  ```
  Definition four_colorable := exists k, coloring k.
  ```

- Чтобы можно было извлекать раскраску `k`, нужно переформулировать в Set:
  `{ k | coloring k }` — и сделать это по всему доказательству.

Prop vs. Set

- Иногда разделение Prop и Set неудобно.
- Пример из современного решения задачи 4 красок (https://github.com/math-comp/fourcolor)

  ```
  Theorem four_color_hypermap : forall G, planar_bridgeless G ->
  four_colorable G.
  ```

  ```
  Definition four_colorable := exists k, coloring k.
  ```

- Чтобы можно было извлекать раскраску `k`, нужно переформулировать в Set:
  `{ k | coloring k }` — и сделать это по всему доказательству.
- **Задача (для курсовой?):** переформулировать «комбинаторную» версию теоремы о 4 красках в универсуме Set и извлечь код-раскрашивалку.

Верификация внешнего кода

- Пусть есть уже готовая функция $f : A \longrightarrow B$, написанная на другом языке программирования.

Верификация внешнего кода

- Пусть есть уже готовая функция $f : A \longrightarrow B$, написанная на другом языке программирования.
- Доказываем $(\forall x \in A)\, S(x, \tilde{f}(x))$, где $\tilde{f}$ — перевод f на язык Gallina.

Верификация внешнего кода

- Пусть есть уже готовая функция $f : A \longrightarrow B$, написанная на другом языке программирования.
- Доказываем $(\forall x \in A)\, S(x, \tilde{f}(x))$, где $\tilde{f}$ — перевод f на язык Gallina.
- Пример — hs-to-coq: Haskell $\longrightarrow$ Gallina.

Верификация внешнего кода

- ```
  addone [] = []
  addone (x:xs) = ((x+1):xs)
  ```
  — транслируется без проблем.
  ```

Верификация внешнего кода

- ```
  addone [] = []
  addone (x:xs) = ((x+1):xs)
  ```
  — транслируется без проблем.
- ```
  fib 0 = 0
  fib 1 = 1
  fib x = (x-1) + (x-2)
  ```
  — при трансляции нужно использовать не обычный рекурсор, а wf-рекурсор.

Верификация внешнего кода

- `addone [] = []`

  `addone (x:xs) = ((x+1):xs)`

  — транслируется без проблем.
- `fib 0 = 0`

  `fib 1 = 1`

  `fib x = (x-1) + (x-2)`

  — при трансляции нужно использовать не обычный рекурсор, а wf-рекурсор.
- $f(2n) = 2f(n)$, $f(2n + 1) = f(2n + 100)$ — здесь нужно строить новый рекурсор.

Верификация внешнего кода

- ```
  addone [] = []
  addone (x:xs) = ((x+1):xs)
  ```
  — транслируется без проблем.
- ```
  fib 0 = 0
  fib 1 = 1
  fib x = (x-1) + (x-2)
  ```
  — при трансляции нужно использовать не обычный рекурсор, а wf-рекурсор.
- $f(2n) = 2f(n)$, $f(2n + 1) = f(2n + 100)$ — здесь нужно строить новый рекурсор.
- За счёт свойства WN все функции, представимые в Coq, всюду определены, и это доказуемо в CIC.

Верификация внешнего кода

- ```
  addone [] = []
  addone (x:xs) = ((x+1):xs)
  ```
  — транслируется без проблем.
- ```
  fib 0 = 0
  fib 1 = 1
  fib x = (x-1) + (x-2)
  ```
  — при трансляции нужно использовать не обычный рекурсор, а wf-рекурсор.
- $f(2n) = 2f(n)$, $f(2n + 1) = f(2n + 100)$ — здесь нужно строить новый рекурсор.
- За счёт свойства WN все функции, представимые в Coq, всюду определены, и это доказуемо в CIC.
- Следовательно, язык Gallina *не полон по Тьюрингу*, даже для класса всюду определённых вычислимых функций.

Надёжность верификации

- Формальные доказательства математических теорем можно считать **намного** более надёжными, чем проверяемые людьми традиционные доказательства (тексты статей).

Надёжность верификации

- Формальные доказательства математических теорем можно считать **намного** более надёжными, чем проверяемые людьми традиционные доказательства (тексты статей).
    - Формулировки (включая необходимые определения) достаточно компактны, чтобы непосредственно проверить, что мы доказываем то, что хотели.

Надёжность верификации

- Формальные доказательства математических теорем можно считать **намного** более надёжными, чем проверяемые людьми традиционные доказательства (тексты статей).
  - Формулировки (включая необходимые определения) достаточно компактны, чтобы непосредственно проверить, что мы доказываем то, что хотели.
  - Доказательства сложны и используют развитую библиотеку тактик, но компилируются в окончательные термы в «исходном» языке, проверяемом ядром Coq.

Надёжность верификации

- Формальные доказательства математических теорем можно считать **намного** более надёжными, чем проверяемые людьми традиционные доказательства (тексты статей).
  - Формулировки (включая необходимые определения) достаточно компактны, чтобы непосредственно проверить, что мы доказываем то, что хотели.
  - Доказательства сложны и используют развитую библиотеку тактик, но компилируются в окончательные термы в «исходном» языке, проверяемом ядром Coq.
  - Вероятность «взаимной компенсации» ошибки внутри Coq и ошибки при формализации конкретного доказательства крайне мала.

Надёжность верификации

- Формальные доказательства математических теорем можно считать **намного** более надёжными, чем проверяемые людьми традиционные доказательства (тексты статей).
  - Формулировки (включая необходимые определения) достаточно компактны, чтобы непосредственно проверить, что мы доказываем то, что хотели.
  - Доказательства сложны и используют развитую библиотеку тактик, но компилируются в окончательные термы в «исходном» языке, проверяемом ядром Coq.
  - Вероятность «взаимной компенсации» ошибки внутри Coq и ошибки при формализации конкретного доказательства крайне мала.
  - Корректность ядра Coq проверена в самом Coq:
    M. Sozeau et al. Coq Coq correct! Verification of type checking and erasure for Coq, in Coq (ACM POPL 2020).

Надёжность верификации

- Надёжность верификации программ — это более тонкий вопрос.

Надёжность верификации

- Надёжность верификации программ — это более тонкий вопрос.
  - Что мы верифицируем?

Надёжность верификации

- Надёжность верификации программ — это более тонкий вопрос.
  - Что мы верифицируем?
  - Например, в случае CompCert гарантируется корректность работы (операционная эквивалентность исходного и откомпилированного кода) только в случае, если компилятор не выдал ошибку.

Надёжность верификации

- Надёжность верификации программ — это более тонкий вопрос.
  - Что мы верифицируем?
  - Например, в случае CompCert гарантируется корректность работы (операционная эквивалентность исходного и откомпилированного кода) только в случае, если компилятор не выдал ошибку.
  - При этом подчёркивается, что компилятор может выдать ошибку и на корректном коде (например, ошибки переполнения при компиляции).

Надёжность верификации

- Надёжность верификации программ — это более тонкий вопрос.
  - Что мы верифицируем?
  - Например, в случае CompCert гарантируется корректность работы (операционная эквивалентность исходного и откомпилированного кода) только в случае, если компилятор не выдал ошибку.
  - При этом подчёркивается, что компилятор может выдать ошибку и на корректном коде (например, ошибки переполнения при компиляции).
  - В CompCert найдены ошибки, связанные с некорректными предположениями («аксиомами»).

Надёжность верификации

- Надёжность верификации программ — это более тонкий вопрос.
  - Что мы верифицируем?
  - Например, в случае CompCert гарантируется корректность работы (операционная эквивалентность исходного и откомпилированного кода) только в случае, если компилятор не выдал ошибку.
  - При этом подчёркивается, что компилятор может выдать ошибку и на корректном коде (например, ошибки переполнения при компиляции).
  - В CompCert найдены ошибки, связанные с некорректными предположениями («аксиомами»).
  - Модель типов данных (особенно — чисел с плавающей запятой) может отличаться от реализации.

Надёжность верификации

- Надёжность верификации программ — это более тонкий вопрос.
  - Что мы верифицируем?
  - Например, в случае CompCert гарантируется корректность работы (операционная эквивалентность исходного и откомпилированного кода) только в случае, если компилятор не выдал ошибку.
  - При этом подчёркивается, что компилятор может выдать ошибку и на корректном коде (например, ошибки переполнения при компиляции).
  - В CompCert найдены ошибки, связанные с некорректными предположениями («аксиомами»).
  - Модель типов данных (особенно — чисел с плавающей запятой) может отличаться от реализации.
  - ...

Надёжность верификации

- Надёжность верификации программ — это более тонкий вопрос.
  - Что мы верифицируем?
  - Например, в случае CompCert гарантируется корректность работы (операционная эквивалентность исходного и откомпилированного кода) только в случае, если компилятор не выдал ошибку.
  - При этом подчёркивается, что компилятор может выдать ошибку и на корректном коде (например, ошибки переполнения при компиляции).
  - В CompCert найдены ошибки, связанные с некорректными предположениями («аксиомами»).
  - Модель типов данных (особенно — чисел с плавающей запятой) может отличаться от реализации.
  - ...
- Тем не менее, верификация зачастую лучше тестирования.

Заключение

- Формальная верификация позволяет избежать человеческих ошибок.

Заключение

- Формальная верификация позволяет избежать человеческих ошибок.
- К сожалению, сам процесс формализации — это отдельная программистская работа.

Заключение

- Формальная верификация позволяет избежать человеческих ошибок.
- К сожалению, сам процесс формализации — это отдельная программистская работа.
    - Бюрократизм и дотошность компьютерной системы (нельзя сказать «аналогично», «ясно, что» и т.п.).

Заключение

- Формальная верификация позволяет избежать человеческих ошибок.
- К сожалению, сам процесс формализации — это отдельная программистская работа.
  - Бюрократизм и дотошность компьютерной системы (нельзя сказать «аналогично», «ясно, что» и т.п.).
  - Недостаточно развитые библиотеки теорем и автоматических тактик.

Заключение

- Формальная верификация позволяет избежать человеческих ошибок.
- К сожалению, сам процесс формализации — это отдельная программистская работа.
  - Бюрократизм и дотошность компьютерной системы (нельзя сказать «аналогично», «ясно, что» и т.п.).
  - Недостаточно развитые библиотеки теорем и автоматических тактик.
  - Необходимость выбрать правильную систему формальных определений (иначе закопаемся).

Заключение

- Формальная верификация позволяет избежать человеческих ошибок.
- К сожалению, сам процесс формализации — это отдельная программистская работа.
  - Бюрократизм и дотошность компьютерной системы (нельзя сказать «аналогично», «ясно, что» и т.п.).
  - Недостаточно развитые библиотеки теорем и автоматических тактик.
  - Необходимость выбрать правильную систему формальных определений (иначе закопаемся).
  - Проблемы совместимости версий и прочие технические трудности.

Заключение

- Формальная верификация позволяет избежать человеческих ошибок.
- К сожалению, сам процесс формализации — это отдельная программистская работа.
  - Бюрократизм и дотошность компьютерной системы (нельзя сказать «аналогично», «ясно, что» и т.п.).
  - Недостаточно развитые библиотеки теорем и автоматических тактик.
  - Необходимость выбрать правильную систему формальных определений (иначе закопаемся).
  - Проблемы совместимости версий и прочие технические трудности.
  - Порог вхождения: необходимость изучить нетривиальный язык и парадигму программирования.

Заключение

- Социальный фактор: получение новых математических результатов интересно и почётно; формализация старых — работа не особо творческая и не приносящая славы.

Заключение

- Социальный фактор: получение новых математических результатов интересно и почётно; формализация старых — работа не особо творческая и не приносящая славы.
    - Иногда, как в случае с проблемой 4 красок, эту работу оплачивают как программистскую.

Заключение

- Социальный фактор: получение новых математических результатов интересно и почётно; формализация старых — работа не особо творческая и не приносящая славы.
    - Иногда, как в случае с проблемой 4 красок, эту работу оплачивают как программистскую.
- Работа по формализации довольно механическая, но при этом её нельзя полностью переложить на компьютер.

Заключение

- Социальный фактор: получение новых математических результатов интересно и почётно; формализация старых — работа не особо творческая и не приносящая славы.
    - Иногда, как в случае с проблемой 4 красок, эту работу оплачивают как программистскую.
- Работа по формализации довольно механическая, но при этом её нельзя полностью переложить на компьютер.
- Задача поиска доказательства алгоритмически неразрешима; человек при этом, даже если он сам не сильный математик, может пользоваться опубликованным неформальным доказательством как путеводной нитью.

Искусственный интеллект?

- Возможно ли обучить систему искусственного интеллекта, чтобы она хотя бы иногда умела правильно применять тактики и «выигрывать в игре против верификатора»?

Искусственный интеллект?

- Возможно ли обучить систему искусственного интеллекта, чтобы она хотя бы иногда умела правильно применять тактики и «выигрывать в игре против верификатора»?
    - ML4PG [Heras, Komendantskaya et al. 2012-...], U. of Dundee;

Искусственный интеллект?

- Возможно ли обучить систему искусственного интеллекта, чтобы она хотя бы иногда умела правильно применять тактики и «выигрывать в игре против верификатора»?
  - ML4PG [Heras, Komendantskaya et al. 2012-...], U. of Dundee;
  - HOList [Bansal et al. 2019], Google Research.

Искусственный интеллект?

- Возможно ли обучить систему искусственного интеллекта, чтобы она хотя бы иногда умела правильно применять тактики и «выигрывать в игре против верификатора»?
  - ML4PG [Heras, Komendantskaya et al. 2012-...], U. of Dundee;
  - HOList [Bansal et al. 2019], Google Research.
- Проблемы «неточности» нет: всё статистическое остаётся «на черновике», чистовой выход (если получится) — строгое доказательство, принятое верификатором.

Искусственный интеллект?

- Возможно ли обучить систему искусственного интеллекта, чтобы она хотя бы иногда умела правильно применять тактики и «выигрывать в игре против верификатора»?
  - ML4PG [Heras, Komendantskaya et al. 2012-...], U. of Dundee;
  - HOList [Bansal et al. 2019], Google Research.
- Проблемы «неточности» нет: всё статистическое остаётся «на черновике», чистовой выход (если получится) — строгое доказательство, принятое верификатором.
- Обучение производится на уже известных формальных доказательствах.

Искусственный интеллект?

- Возможно ли обучить систему искусственного интеллекта, чтобы она хотя бы иногда умела правильно применять тактики и «выигрывать в игре против верификатора»?
  - ML4PG [Heras, Komendantskaya et al. 2012-...], U. of Dundee;
  - HOList [Bansal et al. 2019], Google Research.
- Проблемы «неточности» нет: всё статистическое остаётся «на черновике», чистовой выход (если получится) — строгое доказательство, принятое верификатором.
- Обучение производится на уже известных формальных доказательствах.
  - Это недостаток: формализованных доказательств мало, они неоднородны и отличаются от новой задачи.

Искусственный интеллект?

- Возможно ли обучить систему искусственного интеллекта, чтобы она хотя бы иногда умела правильно применять тактики и «выигрывать в игре против верификатора»?
  - ML4PG [Heras, Komendantskaya et al. 2012-...], U. of Dundee;
  - HOList [Bansal et al. 2019], Google Research.
- Проблемы «неточности» нет: всё статистическое остаётся «на черновике», чистовой выход (если получится) — строгое доказательство, принятое верификатором.
- Обучение производится на уже известных формальных доказательствах.
  - Это недостаток: формализованных доказательств мало, они неоднородны и отличаются от новой задачи.
- Можем ли мы использовать как «подсказки» неформальные тексты?

Искусственный интеллект?

- Возможно ли обучить систему искусственного интеллекта, чтобы она хотя бы иногда умела правильно применять тактики и «выигрывать в игре против верификатора»?
  - ML4PG [Heras, Komendantskaya et al. 2012-...], U. of Dundee;
  - HOList [Bansal et al. 2019], Google Research.
- Проблемы «неточности» нет: всё статистическое остаётся «на черновике», чистовой выход (если получится) — строгое доказательство, принятое верификатором.
- Обучение производится на уже известных формальных доказательствах.
  - Это недостаток: формализованных доказательств мало, они неоднородны и отличаются от новой задачи.
- Можем ли мы использовать как «подсказки» неформальные тексты?
  - Здесь возникают лингвистические вопросы...