

Соq. Автоматизированное построение и проверка математических доказательств

Семинар международной лаборатории статистической и
вычислительной геномики

21 апреля 2020 г.

С. Л. Кузнецов

Математический институт им. В.А. Стеклова РАН, Москва

Национальный исследовательский университет Высшая школа экономики, Москва

Московский государственный университет им. М.В. Ломоносова

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

Что такое доказательство?

Что такое доказательство?

- **Неформальное** («для людей»): *доказательство — это рассуждение, которое убеждает типичного математика.*

Что такое доказательство?

- **Неформальное** («для людей»): доказательство — это рассуждение, которое убеждает типичного математика.
- **Формальное** («для роботов»): доказательство — это логически корректное рассуждение, устанавливающее истинность данного утверждения;

Что такое доказательство?

- **Неформальное** («для людей»): *доказательство — это рассуждение, которое убеждает типичного математика.*
- **Формальное** («для роботов»): *доказательство — это логически корректное рассуждение, устанавливающее истинность данного утверждения;*
т.е. конечный объект (последовательность формул, дерево, и т.п.), построенное по определённым **правилам**, корректность использования которых может быть **проверена** автоматически.

Что такое доказательство?

- **Неформальное** («для людей»): *доказательство — это рассуждение, которое убеждает типичного математика.*
- **Формальное** («для роботов»): *доказательство — это логически корректное рассуждение, устанавливающее истинность данного утверждения;*
т.е. конечный объект (последовательность формул, дерево, и т.п.), построенное по определённым **правилам**, корректность использования которых может быть **проверена** автоматически.

Devlin's Angle, by Keith Devlin, 2003:

https://www.maa.org/external_archive/devlin/devlin_06_03.html

Пример: логика предикатов, по Гильберту

Схемы аксиом (A, B, C — произвольные формулы):

$$A \rightarrow (B \rightarrow A)$$

$$(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$$

$$A \rightarrow (B \rightarrow (A \wedge B))$$

$$(A \wedge B) \rightarrow A \quad (A \wedge B) \rightarrow B$$

$$A \rightarrow (A \vee B) \quad B \rightarrow (A \vee B)$$

$$(A \rightarrow C) \rightarrow ((B \rightarrow C) \rightarrow ((A \vee B) \rightarrow C))$$

$$(A \rightarrow \neg B) \rightarrow ((A \rightarrow B) \rightarrow \neg A)$$

$$\neg\neg A \rightarrow A$$

$$\forall x A(x) \rightarrow A(t), \text{ если подстановка допустима}$$

$$A(t) \rightarrow \exists x A(x), \text{ если подстановка допустима}$$

$$\forall x (B \rightarrow A(x)) \rightarrow (B \rightarrow \forall x A(x)), \text{ если } x \text{ не свободна в } B$$

$$\forall x (A(x) \rightarrow B) \rightarrow ((\exists x A(x)) \rightarrow B), \text{ если } x \text{ не свободна в } B$$

Правила вывода:

$$\frac{A \quad A \rightarrow B}{B} \text{ modus ponens (MP)} \quad \frac{A(x)}{\forall x A(x)} \text{ Gen}$$

Пример: логика предикатов, по Гильберту

- **Доказательство (вывод)** — это последовательность (цепочка) формул, каждая из которых является либо аксиомой, либо получается из формул, стоящих ранее в цепочке, с помощью одного из правил вывода (например, если ранее встречались формулы φ и $\varphi \rightarrow \psi$, то по правилу *modus ponens* можно написать формулу ψ).

Пример: логика предикатов, по Гильберту

- **Доказательство (вывод)** — это последовательность (цепочка) формул, каждая из которых является либо аксиомой, либо получается из формул, стоящих ранее в цепочке, с помощью одного из правил вывода (например, если ранее встречались формулы φ и $\varphi \rightarrow \psi$, то по правилу *modus ponens* можно написать формулу ψ).
- **Результат (цель)** доказательства — последняя формула цепочки.

Пример: логика предикатов, по Гильберту

- **Доказательство (вывод)** — это последовательность (цепочка) формул, каждая из которых является либо аксиомой, либо получается из формул, стоящих ранее в цепочке, с помощью одного из правил вывода (например, если ранее встречались формулы φ и $\varphi \rightarrow \psi$, то по правилу *modus ponens* можно написать формулу ψ).
- **Результат (цель)** доказательства — последняя формула цепочки.
- Вопрос о том, является ли данная цепочка корректным доказательством, **алгоритмически разрешим**.

Пример: логика предикатов, по Гильберту

- **Доказательство (вывод)** — это последовательность (цепочка) формул, каждая из которых является либо аксиомой, либо получается из формул, стоящих ранее в цепочке, с помощью одного из правил вывода (например, если ранее встречались формулы φ и $\varphi \rightarrow \psi$, то по правилу *modus ponens* можно написать формулу ψ).
- **Результат (цель)** доказательства — последняя формула цепочки.
- Вопрос о том, является ли данная цепочка корректным доказательством, **алгоритмически разрешим**.
- Более сложная задача *построения* цепочки-вывода для данной цели разрешима далеко не всегда.

Докажем, что φ влечёт φ .

Типичное формальное доказательство

Докажем, что φ влечёт φ .

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

Типичное формальное доказательство

Докажем, что φ влечёт φ .

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

Бррр...

Формальные доказательства

- **Pro:** проверяются формально и автоматически, что гарантирует отсутствие «человеческого фактора»

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...

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

- 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?”

- 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, ...

- 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, ...

- 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, ...

- 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, ...
 - ...

- 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.

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

- Программа проверки доказательств (**proof checker**): принимает в качестве входных данных формулу A и некоторый объект (текст) \mathcal{P} и возвращает «да», если \mathcal{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».

- Программа проверки доказательств (**proof checker**): принимает в качестве входных данных формулу A и некоторый объект (текст) \mathcal{P} и возвращает «да», если \mathcal{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 \mathcal{P} полностью возлагается на человека.

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

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

- 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.

Основные факты:

Основные факты:

- Разрабатывается в INRIA, Франция, с 1989 г.

Основные факты:

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

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.)

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.

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**

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)**.

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

- **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.*

- **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 Соq Г. Гонтье (Microsoft Research) в 2005 г.

- **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 Соq Г. Гонтье (Microsoft Research) в 2005 г.
 - Другой пример: теорема Фейта–Томпсона о группах нечётного порядка.

- **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 Соq Г. Гонтье (Microsoft Research) в 2005 г.
 - Другой пример: теорема Фейта–Томпсона о группах нечётного порядка.
Формализовал Г. Гонтье в 2012 г.

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

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

- **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).

- **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).

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

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

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

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

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

- 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”).

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

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

- **Соответствие Карри – Говарда:**
доказательства = объекты и программы;
теоремы = типы данных.
- Доказать $A \rightarrow B$ 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*).

- **Соответствие Карри – Говарда:**
доказательства = объекты и программы;
теоремы = типы данных.
- Доказать $A \rightarrow B$ 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 $P(a)$).

- **Соответствие Карри – Говарда:**
доказательства = объекты и программы;
теоремы = типы данных.
- Доказать $A \rightarrow B$ 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 $P(a)$).
- и т.д.

- **Соответствие Карри – Говарда:**
доказательства = объекты и программы;
теоремы = типы данных.
- Доказать $A \rightarrow B$ 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 $P(a)$).
- и т.д.
- Пример: доказательство $(A \rightarrow B) \rightarrow ((B \rightarrow C) \rightarrow (A \rightarrow C))$ записывается следующим λ -термом:

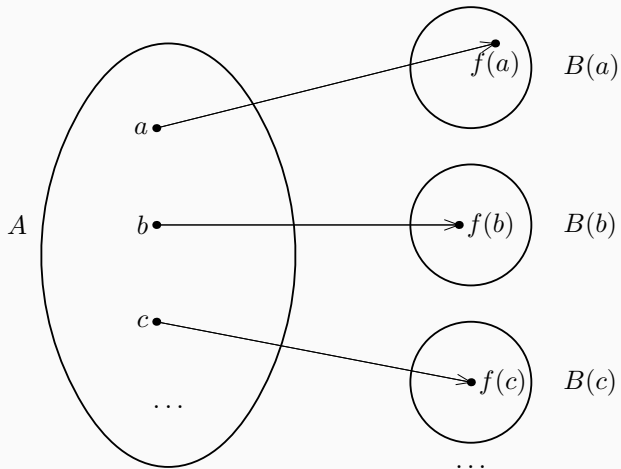
$$\lambda f.\lambda g.\lambda x.g(f(x))$$

- **Соответствие Карри – Говарда:**
доказательства = объекты и программы;
теоремы = типы данных.
- Доказать $A \rightarrow B$ 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 $P(a)$).
- и т.д.
- Пример: доказательство $(A \rightarrow B) \rightarrow ((B \rightarrow C) \rightarrow (A \rightarrow C))$ записывается следующим λ -термом:

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

- Это – абстрактная программа для композиции функций.

Зависимый тип функций



- Сам Coq написан на языке OCaml.

- 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).

- 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 ($0 \in \mathbb{N}$ ¹; $n \in \mathbb{N} \Rightarrow S(n) = n + 1 \in \mathbb{N}$).
Более сложные — например, конечные деревья.

¹Vive la France!

- 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 ($0 \in \mathbb{N}$ ¹; $n \in \mathbb{N} \Rightarrow S(n) = n + 1 \in \mathbb{N}$). Более сложные — например, конечные деревья.
- Теоретически, можно программировать доказательства непосредственно как λ -термы (команда exact).

¹Vive la France!

- 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 ($0 \in \mathbb{N}$ ¹; $n \in \mathbb{N} \Rightarrow S(n) = n + 1 \in \mathbb{N}$). 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).

¹Vive la France!

- Работа с Соq'ом интерактивна.

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

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

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

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

- Работа с Соq'ом интерактивна.
- В каждый момент имеется несколько *задач на доказательство* (т.е. построение терма) вида $x_1 : A_1, \dots, x_n : A_n \vdash (???) : B$.
- A_i — посылки, B — требуемое заключение.
- Если $B = A_i$, доказывать ничего не надо.
- Применение тактики призвано *упростить* задачу поиска вывода.
- Например, от задачи $\Gamma \vdash (B \rightarrow C)$ 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 Соq'ом интерактивна.
- В каждый момент имеется несколько *задач на доказательство* (т.е. построение терма) вида $x_1 : A_1, \dots, x_n : A_n \vdash (???) : B$.
- A_i — посылки, B — требуемое заключение.
- Если $B = A_i$, доказывать ничего не надо.
- Применение тактики призвано *упростить* задачу поиска вывода.
- Например, от задачи $\Gamma \vdash (B \rightarrow C)$ 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).
- Этот процесс называется *естественным выводом*.

- Есть и более сложные тактики.

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

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

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

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

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

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

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

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

- 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 $f : A \rightarrow \bigcup B(x)$, с условием (*спецификацией*) $f(a) \in B(a)$ для каждого $a \in A$.

- 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 $f : A \rightarrow \bigcup B(x)$, с условием (*спецификацией*) $f(a) \in B(a)$ для каждого $a \in A$.
- $\forall\exists$ -формула специфицирует функцию:
 $(\forall x \in A) (\exists y \in B) S(x, y)$.

- 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 $f : A \rightarrow \bigcup B(x)$, 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 $f : A \mapsto B$, причём гарантировано выполнение спецификации $S(a, f(a))$ для каждого $a \in A$.

- Пример: наибольший общий делитель

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

- Пример: наибольший общий делитель

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

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

- Пример: наибольший общий делитель

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

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

- Пример: наибольший общий делитель

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

- 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 $a, b \mapsto d$, исчезает доказательство спецификации.

- Set-часть должна быть чисто интуиционистской.

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

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

- Set-часть должна быть чисто интуиционистской.
- Если мы допустим аксиому $A \vee \neg A$ (здесь $\vee : \text{Prop} \times \text{Prop} \rightarrow \text{Set}$), 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`.

- Set-часть должна быть чисто интуиционистской.
- Если мы допустим аксиому $A \vee \neg A$ (здесь $\vee : \text{Prop} \times \text{Prop} \rightarrow \text{Set}$), 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.

- Set-часть должна быть чисто интуиционистской.
- Если мы допустим аксиому $A \vee \neg A$ (здесь $\vee : \text{Prop} \times \text{Prop} \rightarrow \text{Set}$), 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.

- Иногда разделение Prop и Set неудобно.

- 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.
```

- 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:
 $\{ k \mid \text{coloring } k \}$ — и сделать это по всему доказательству.

- 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:
 $\{ k \mid \text{coloring } k \}$ — и сделать это по всему доказательству.
- Это должно быть выполнимым, но довольно техническим упражнением.

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

- 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.

- 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 компактным ядром Coq.

- 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 компактным ядром Coq.
 - Вероятность «взаимной компенсации» ошибки внутри Coq и ошибки при формализации конкретного доказательства крайне мала.

Надёжность верификации

- Надёжность верификации программ — это более тонкий вопрос.

Надёжность верификации

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

- 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 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).
 - 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).
 - 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).
 - Модель типов данных (особенно — чисел с плавающей запятой) может отличаться от реализации.
 - ...
- Тем не менее, верификация во многом лучше тестирования.

- Формальная верификация позволяет избежать человеческих ошибок.

Заключение

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

- 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.).

- 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.

- 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.

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

- 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.

- 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.

- 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», чистой 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», чистой 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», чистой 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», чистой 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», чистой 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...