定理証明支援系 Coq による ソフトウェア・数学の形式検証

奈良先端科学技術大学大学院 コロキウム講義

Reynald Affeldt (アフェルト レナルド)

産業技術総合研究所 (情報技術研究部門)・
奈良先端科学技術大学大学院 (セキュアソフトウェアシステム研究室)

2018 年 06 月 05 日

本発表

- 発表者: Reynald AFFELDT (アフェルト レナルド)
  - 所属:
    - 産業技術総合研究所 (つくば)
    - 奈良先端科学技術大学大学院 (セキュアソフトウェアシステム研究室: 大岩寛)
  - 専門: 形式論理に基づく検証
- 内容:
  - 定理証明支援系による形式検証の分野紹介
- 目的:
  - 皆がやってもらいたい!

Outline

定理証明支援系による形式検証: 動機

- 再確認
  - ソフトウェア安全性の保証, バグがないことの保証
    - 問題の例:
      OpenSSL (Debian の弱い鍵 (2006 年〜2008 年), Heartbleed (2014 年に発表))
    - (物理現象を除いた) ハードウェアへの応用
      (例: マイクロコード) (J. Harrison(当時:Intel 社) の研究に参考)
  - 数学の証明の正しさ
    - Kepler 予想の証明の査読 [Hal08]
    - “A technical argument by a trusted author, which is hard to check and looks similar to arguments known to be correct, is hardly ever checked in detail.” [Voe14]
- 安全な開発方法
  - 基盤ソフトウェア (例: CompCert コンパイラ [Ler09], seL4 マイクロカーネル [WKS+09])
  - 膨大な数学証明の時代:
    - Polymath プロジェクト
    - Kepler 予想の証明の形式化の国際協力 [Hal12]
    - “the future of both mathematics and programming lies in the fruitful combination of formal verification and the usual social processes that are already working in both scientific disciplines” [AGN09]

定理証明支援系とは?

- 定理証明支援系の役割:
  1. 証明の記述を支援（自動化, 記号, 抽象化）
  2. 証明の正しさを保証（型理論による）
- 定理証明支援系の強み:
  - 信頼性が高い:
    カーネル (中核部分) は “小さい” ため, 理論的な誤りは紙上で確認できる
  - 汎用性が高い:
    数学的帰納法, 整礎帰納法を利用できるので, 有限システムに制限されない (モデル検査と比べて)
- 定理証明支援系の例:
  - 型理論に基く: Coq, HOL Light, Isabelle/HOL, Agda 等
  - その他の理論に基く定理証明支援系: Mizar (1973 年から, Tarski-Grothendieck 集合論に基く, 計算力ない), ACL2, PVS 等
- 使い方: 対話的に証明を構成する

対話的な証明の流れ (Coq/SSReflect の場合)

ユーザ			定理証明支援系 Coq
言明	`Goal forall n : nat, n + n = 2 * n.`	⇝	型検査
		⇜	ゴール $?_1$
証明の記述 (1/3) (ゴール $?_1$ に対して)	`elim.`	⇝	証明項の構築 (開始)
		⇜	ゴール $?_2$, $?_3$

証明の記述 (2/3)
(ゴール $?_2$ に対して)
	`rewrite addn0.`
	`rewrite muln0.`
	`by [].`
		⇝	証明項の構築 (続き)

		⇜	ゴール $?_3$

証明の記述 (3/3)
(ゴール $?_3$ に対して)
	`move=> n IH.`
	`rewrite addnS.`
	`rewrite addSn.`
	`rewrite IH.`
	`rewrite mulnS.`
	`rewrite add2n.`
	`by [].`
		⇝	証明項の構築 (完了)

型理論に基づく定理証明支援系の歴史 I

- 19 世紀: 数学の基礎の研究の開始 (1879 年: G. Frege の Begriffsschrift; 1880 年代: G. Cantor による集合論)
- 1901 年: B. Russell が集合論の簡単な矛盾を発見 $(a = \{x \mid x \notin x\}, a \in a \leftrightarrow a \notin a) \Rightarrow$ “It is the distinction between logical types that is the key to the whole mystery.” (以上については [vH02] に参照)
- 1908 年: B. Russell の “vicious-circle principle”: “Whatever contains an apparent variable must not be a possible value of that variable”[Rus08] $\Rightarrow$ 型の hierarchy (individuals < first-order propositions < $\cdots$)
- 1910–1913 年: B. Russell と A. N. Whitehead の Principia Mathematica; 型を用いた集合論による数学の再構築; 批判があった (L. Wittgenstein 等); 数学世界に影響はなかった
- 1930 年代: H. B. Curry が命題論理とコンビネータの間の Curry 同型対応を発見
- 1940 年: A. Church の Simple Theory of Types[Chu40]; 型付 λ 計算を利用 (型 ι: “individuals”; 型 o: “propositions”); extensional; 定理証明支援系 HOL の基礎

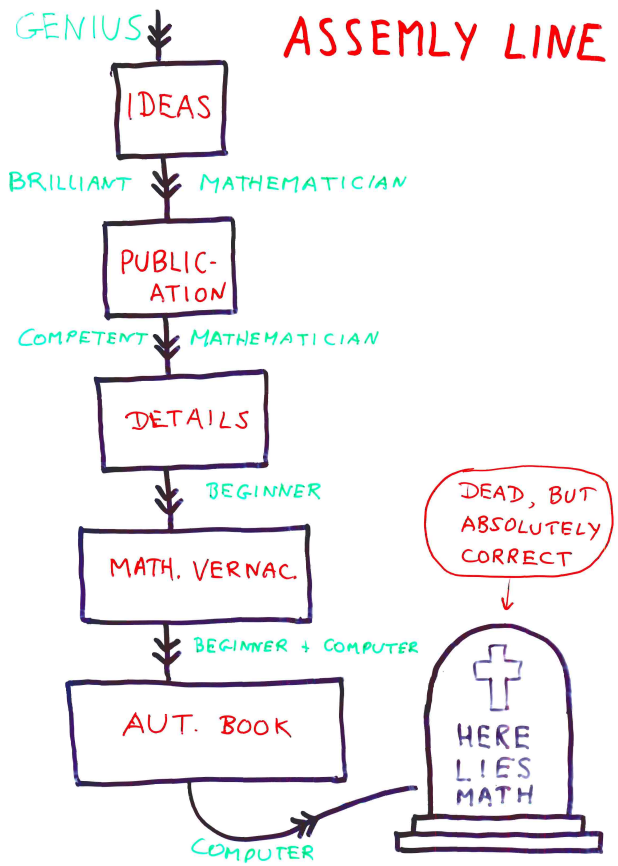
型理論に基づく定理証明支援系の歴史 II

- 1950 年代: カット除去と λ 計算の実行の間 (W. W. Tait)
- 1967–1968 年: N. G. de Bruijn, 定理証明支援系 AUTOMATH
- 1969 年: Curry-Howard 同型対応 [How80]: proof-checking = type-checking
- 1973 年: P. Martin-Löf の型理論; Leibniz equality を含む
- 1970 年代: R. Milner (1991 年のチューリング賞) の LCF (Logic for Computable Functions); 型理論の機械化 $\Rightarrow$ 型つきプログラミング言語 ML の発想
- 1984 年: 定理証明支援系 Coq の開発の開始 [CH84]
- 2005 年から: クリティカルな基盤ソフトウェアの検証 (CompCert, seL4), 膨大な数学の証明の形式化 (四色定理, Kepler 予想)

Outline

数学の形式化

[dB03] による:

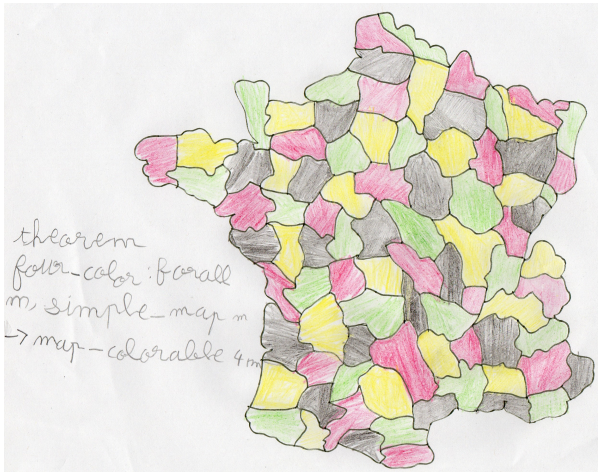


- 目的:
  - ミス, 穴のない証明の作成
  - 証明の向上 (最適化, 短さ)
  - 関連する定理の発見に繋る
- 難しさ:
  - 教科書の 1 頁: 約 1 週間かかる [Hal08, Wie14]
  - 200 頁の教科書: 約 5 人年かかる [Wie14]
- ソフトウェアとハードウェアの検証に繋る
  - 定理のライブラリ
  - 証明記述の技術

四色定理の形式化

歴史

「いかなる地図も隣接する領域が異なる色になるように塗るには 4 色あれば十分である」



- 1852 年: F. Guthrie (イギリス) による言明
- 試み: A. de Morgan, H. L. Lebesgue 等
- 誤った証明: A. Kempe, P. G. Tait
- 1976 年: K. Appel, W. Haken (イリノイ大学) による証明
  - 正しさは簡単に確認できないので, 一部の数学者から批判
    - 大量の場合分け (10,000 submaps), 一部の計算は IBM 370-168 のアセンブリプログラムに任されていた (1,000,000,000 colorings; 実行は 1200 時間 (約 2ヶ月間) かかった) [Hal13]
    - 実際に, 間もなく, 場合分けアセンブリに誤りが発見されたそう
- 1995 年: 証明の簡単化; コンピュータプログラムの改善 (C 言語, 当時の PC で約 3 時間)

四色定理の形式化

形式化

- 2000 年ごろ: G. Gonthier と B. Werner (INRIA, Microsoft Research) は Coq で形式化を開始
- 2004 年: 四色定理の形式化が完成 [Gon08]
  - 数時間で検証可能
  - 言明: 30 行以内のスクリプトで形式定義 (`fourcolor.v` に言明):

```
Theorem four_color (m : map R) : simple_map m -> map_colorable 4 m.
```

  - 証明: 約 60,000 行のスクリプト [Gon05]
- SSReflect(Coq の拡張) の開発のきっかけ; 現在, 数学の形式化以外でも広く利用

奇数位数定理

- 1911 年: W. Burnside による予想
- 1963 年: W. Feit と J. G. Thompson が証明
  - この時代の群論の結果として, 証明は長かった
  - 大学の群論や線形代数学等が必要, 大学院レベルの様々な理論も必要
- 1990 年代: 簡単化 ⇒ 証明: 255 ページ
- G. Gonthier らが Feit-Thompson 定理の形式化を取り組む
- (2011 年: G. Gonthier は EADS（現在: Airbus 社） Foundation 賞を受賞)
- 2012 年 9 月: 完成; `PFsection14.v` に言明:

```
Theorem Feit_Thompson (gT : finGroupType) (G : {group gT}) :
  odd #|G| -> solvable G.
```

- 7 年間の研究, 多くの協力者 (学会論文 [GAA+13] は 15 人)
- 約 164,000 行の Coq のスクリプト
  - 奇数位数定理の証明自体約 40,000 行; 紙上の証明に比べて 4.5 倍
  - その他: 再利用性の高い基礎ライブラリ

Kepler 予想の証明の形式化

歴史

「無限の空間において同半径の球を敷き詰めたとき, 最密な充填方法は面心立方格子である」

- 1611 年: J. Kepler が予想を発表
- Hilbert の第 18 問題
- 1998 年: T. C. Hales と S. P. Ferguson が証明を発表; Annals of Mathematics に投稿
- 証明: 300 ページ + 40,000 行のプログラム [Hal08]; 実行時間: 約 2,000 時間 (約 3ヶ月間)
  - 2012 年: プログラム ≤10,000 行; 実行時間: 約 20 時間 [Hal12, Hal13]
- 2005 年: 論文発表; しかし, 4 年間の査読を経ても証明の正しさを完全に保証出来ない [Hal08]

Kepler 予想の証明の形式化

形式化プロジェクトの概要

- 2003 年 (の数年後): Flyspeck (Formal Proof of Kepler Conjecture の略) プロジェクト開始
- 形式化に必要な時間の見積は難しい:
  - 2008 年: “Flyspeck may take as many as twenty work-years to complete.” [Hal08]
  - 2012 年: “The Flyspeck project is about 80% complete.” [Hal12]
  - 2014 年 8 月 10 日: 完成
    (`http://code.google.com/p/flyspeck/wiki/AnnouncingCompletion`)
- スクリプトのサイズ: 約 325,000 行と言われている
- 国際協力 (米国やベトナムやドイツ等からの開発者)
- その他の予想の証明 [Hal12]:
  - 1969 年の F. Tóth’s full contact 予想
  - 2000 年の K. Bezdek’s strong dodecahedral 予想

Outline

C コンパイラ (CompCert) I

信頼性の高いコンパイラの構築

- コンパイルの前の C 言語のソースコードとコンパイルによって得たアセンブリは同じ動作をするかどうか; イメージ:

```
Definition compcert := fun (c : C_prg) => (t : ASM_prg).
Lemma correctness : forall c o, observe o c -> observe o (compcert c).
```

- CompCert は従来のコンパイラよりバグが少ないことが示された
  - テストによる比較実験 [YCER11]
  - 発見された CompCert のバグはまだ検証されていないところにあった
- 応用先: 組み込みシステム (最新の研究は Airbus 社等と共に)
- 2004 年から INRIA で X. Leroy らが形式検証を続けている [Ler09, BL09]
- 2013 年: X. Leroy は Microsoft Research Verified Software Milestone 賞を受賞
- 約 50,000 行のスクリプト, 4 人年だと言われている
- 定理証明支援系による検証に必要な時間を予測するのは一般的に難しい

C コンパイラ (CompCert) II

- 2013 年に受賞の際: X. Leroy:「2006 には (NB: 国際学会で CompCert の初発表), 検証の完成度は 80%ぐらいだと思っていた; 2013 年に振り返ってみると, 20%に過ぎなかったと認めなければならない」

- 最新の成果: Verasco's abstract interpreter (形式検証済みの自動解析); 受賞：Royal Society Milner Award 2016

コモンクライテリアによる認証取得 I

定理証明支援系の影響は IT 業界まで及ぶ

- IT 製品において, 安全性の根拠を示すことが求められている
- コンピュータセキュリティのための国際規格としてコモンクライテリアは有名
  - セキュリティを認証するための評価基準を定める
  - 1996 年から
  - ISO/IEC 15408
- 最も厳密な評価レベルは EAL7
  - その評価を取得するため, 定理証明支援系の利用は不可欠
- 欧州では 2000 年代からスマートカードの評価に定理証明支援系はしばしば使われている
  - 例: JavaCard (117,000 行の Coq スクリプト, 2003 年からの研究), JavaCard API の複数のバグの発見 [CN08]
- EAL7 の評価取得例 (フランスの Trusted Labs 社と共同):
  - JavaCard の実装: SIMEOS (2007 年) と m-NFC (2012 年) (Gemalto, フランス)
  - Multos (2013 年)
  - Samsung のマイコンの MMU (2013 年)

コモンクライテリアによる認証取得 II

- NB: 認証取得は, 競争的な強みとなるが, コストの負担は大きくなる
  - EAL7: 数千行のソースコードは数百万ドルかかると言われている [Bol10]
  - EAL6: 1 行, 1000 ドルとも言う [Kle10]

seL4 マイクロカーネル

- オーストラリアの NICTA(当時) 研究所
- 定理証明支援系: Isabelle/HOL
- seL4 のソースコード: C 言語; 約 8,700 行
- アプローチ:
  - ホーア論理
  - (段階的な) 詳細化 (refinement)
    - 形式仕様は抽象的なモデル, 中間モデルは Haskell, C 言語の実装まで
- 7,500 行のソースコードに対し約 200,000 行のスクリプト, 約 25 人年 [Kle10]
  - 全部を含む: C 言語の検証基盤, tools, 定理のライブラリ等
- 組み込み用のオペレーティングシステムとしてビジネスと繋がる
- 2014 年 7 月からオープンソース
- seL4 プロジェクトは計画的に運営されたので, 重要な情報を得た:
  - 1 行は 700 ドル (1,000 行, 70 万ドル)
  - カーネルだけだと, 約 12 人年/1 行は 350 ドル
  - 再現性: 予想ではこれから似たプロジェクトの際, 12 人年/1 行は 230 ドル

⇒ 信頼性の最も高いソフトウェアの開発方法として, 形式検証はより経済的な開発方法になる可能性がある

Outline

自然演繹による Hilbert の公理 S の証明

自然演繹の一部の規則

$$\frac{A \in \Gamma}{\Gamma \vdash A}\ \text{axiom} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ \to_i \qquad \frac{\Gamma \vdash A \to B \qquad \Gamma \vdash A}{\Gamma \vdash B}\ \to_e$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash A \to B \to C}\ \text{axiom} \quad \dfrac{}{\Gamma \vdash A}\ \text{axiom}}{\Gamma \vdash B \to C}\ \to_e \quad \dfrac{\dfrac{}{\Gamma \vdash A \to B}\ \text{axiom} \quad \dfrac{}{\Gamma \vdash A}\ \text{axiom}}{\Gamma \vdash B}\ \to_e}{\Gamma^1 \vdash C}\ \to_e}{A \to B \to C, A \to B \vdash A \to C}\ \to_i}{A \to B \to C \vdash (A \to B) \to A \to C}\ \to_i}{\vdash (A \to B \to C) \to (A \to B) \to A \to C}\ \to_i$$

[1] $\Gamma = A \to B \to C, A \to B, A$

証明の例 (1/7)

ゴールの記述

$$\vdash (A \to B \to C) \to (A \to B) \to A \to C$$

```
Lemma hilbertS (A B C : Prop) :
  (A -> B -> C) -> (A -> B) -> A -> C.
Proof.
```

証明の例 (2/7)

一番目の含意の導入

$$\frac{\overbrace{A \to B \to C}^{\texttt{H1}} \vdash (A \to B) \to A \to C}{\vdash (A \to B \to C) \to (A \to B) \to A \to C} \to_i$$

```
Lemma hilbertS (A B C : Prop) :
  (A -> B -> C) -> (A -> B) -> A -> C.
Proof.
refine (fun (H1 : A -> B -> C) => _).
```

証明の例 (3/7)

三つの含意の導入

$$\dfrac{\dfrac{\dfrac{\overbrace{A \to B \to C}^{\texttt{H1}}, \overbrace{A \to B}^{\texttt{H2}}, \overbrace{A}^{\texttt{H3}} \vdash C}{A \to B \to C, A \to B \vdash A \to C} \to_i}{A \to B \to C \vdash (A \to B) \to A \to C} \to_i}{\vdash (A \to B \to C) \to (A \to B) \to A \to C} \to_i$$

```
Lemma hilbertS (A B C : Prop) :
  (A -> B -> C) -> (A -> B) -> A -> C.
Proof.
refine (fun (H1 : A -> B -> C) => _).
refine (fun (H2 : A -> B) => _).
refine (fun (H3 : A) => _).
```

証明の例 (4/7)

含意の一番目の除去

$$\dfrac{\dfrac{\dfrac{\dfrac{A \to B \to C, A \to B, A \vdash A \qquad A \to B \to C, A \to B, A \vdash B}{\overbrace{A \to B \to C}^{\texttt{H1}}, A \to B, A \vdash C}\,\to_e}{A \to B \to C, A \to B \vdash A \to C}\,\to_i}{A \to B \to C \vdash (A \to B) \to A \to C}\,\to_i}{\vdash (A \to B \to C) \to (A \to B) \to A \to C}\,\to_i$$

($A \to B \to C$ の結論と C をマッチし, 二つのサブゴール A と B を生成する)

```
Lemma hilbertS (A B C : Prop) :
  (A -> B -> C) -> (A -> B) -> A -> C.
Proof.
...
refine (H1 _ _).
```

証明の例 (5/7)

「公理」ルールを適用

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \to B \to C, A \to B, \overbrace{A}^{\texttt{H3}} \vdash A}\,\text{ax} \qquad A \to B \to C, A \to B, A \vdash B}{A \to B \to C, A \to B, A \vdash C}\,\to_e}{A \to B \to C, A \to B \vdash A \to C}\,\to_i}{A \to B \to C \vdash (A \to B) \to A \to C}\,\to_i}{\vdash (A \to B \to C) \to (A \to B) \to A \to C}\,\to_i$$

```
Lemma hilbertS (A B C : Prop) :
  (A -> B -> C) -> (A -> B) -> A -> C.
Proof.
...
refine (H1 _ _).
exact H3.
```

証明の例 (6/7)

含意の二番目の除去

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \to B \to C, A \to B, A \vdash A}\,\mathrm{ax} \qquad \dfrac{A \to B \to C, A \to B, A \vdash A}{A \to B \to C, \overbrace{A \to B}^{\mathrm{ab}}, A \vdash B}\,\to_e}{A \to B \to C, A \to B, A \vdash C}\,\to_e}{A \to B \to C, A \to B \vdash A \to C}\,\to_i}{A \to B \to C \vdash (A \to B) \to A \to C}\,\to_i}{\vdash (A \to B \to C) \to (A \to B) \to A \to C}\,\to_i$$

($A \to B$ の結論と B をマッチし, サブゴール A を生成する)

```
Lemma hilbertS (A B C : Prop) :
  (A -> B -> C) -> (A -> B) -> A -> C.
Proof.
...
refine (H2 _).
```

証明の例 (7/7)

公理と Qed

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \to B \to C, A \to B, \overbrace{A}^{\text{a}} \vdash A}\,\text{ax}}{A \to B \to C, A \to B, A \vdash B}\,{\to_e}}{A \to B \to C, A \to B, A \vdash C}\,{\to_e}}{A \to B \to C, A \to B \vdash A \to C}\,{\to_i}}{A \to B \to C \vdash (A \to B) \to A \to C}\,{\to_i}}{\vdash (A \to B \to C) \to (A \to B) \to A \to C}\,{\to_i}$$

```
Lemma hilbertS (A B C : Prop) :
  (A -> B -> C) -> (A -> B) -> A -> C.
Proof.
refine (fun (H1 : A -> B -> C) => _).
refine (fun (H2 : A -> B) => _).
refine (fun (H3 : A) => _).
refine (H1 _ _).
  exact H3.
refine (H2 _).
exact H3.
Qed.
```

Outline

定理証明支援系 Coq

- 最も使われている定理証明支援系
  - 代表的な国際学会 ITP の論文の割合
  - プログラミング基礎の有名な国際学会 POPL(ACM) の論文の割合
    - 補佐ツールとして: 定理の正しさの確認, 検証フレームワークの開発基盤, プログラミング基礎の研究等
    - 2012 年と 2013 年に 20% 以上の論文は定理証明支援系を利用していた
- 受賞:
  - ACM SIGPLAN Programming Languages Software 2013 賞
  - ACM Software System 2013 賞
- 開発の開始: 1984 年 [CH84]
- 基礎: 型付きプログラミング言語
  - ≃ Calculus of Inductive Constructions [CP90, PM92]
    - Calculus of Constructions [CH84, CH85, CH86, CH88] の拡張
  - 具体的な実装は Gallina と呼ぶ

Gallina の中核部分の頁

t	:=	`Prop`	命題のソート
	|	x, A	変数
	|	$A \to B$	非依存型 product
	|	`fun` x => t	関数抽象
	|	$t_1\ t_2$	関数適用

Gallina の中核部分の型付け規則 (依存型なし)

- Gallina の型 judgment: $\Gamma \vdash t : A$
  - t = 証明, A = 型
  - Γ = ローカルコンテキスト
    $x_0 : A_0$ (仮定) または $x_1 : A_1 := t_1$(定義) を含む
- [CDT18, Sect. “Calculus of Inductive Constructions”, “Typing rules”] による:

$$\frac{x : A \in \Gamma \text{ or } \exists t. x : A := t \in \Gamma}{\Gamma \vdash x : A}\ \text{Var} \qquad \text{仮定の利用}$$

$$\frac{\Gamma \vdash A \to B : \texttt{Prop} \qquad \Gamma, x : A \vdash t : B}{\Gamma \vdash \texttt{fun}\, x \Rightarrow t : A \to B}\ \text{Lam} \qquad \text{補題の導入}$$

$$\frac{\Gamma \vdash t_1 : A \to B \qquad \Gamma \vdash t_2 : A}{\Gamma \vdash t_1\ t_2 : B}\ \text{App} \qquad \text{補題の利用}$$

- 型は命題として見られる! (Curry-Howard 同型対応)

証明項付きの Hilbert の公理 S の証明

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash H_1 : A \to B \to C}\,\mathrm{Var} \quad \dfrac{}{\Gamma \vdash H_3 : A}\,\mathrm{Var}}{\Gamma \vdash H_1\ H_3 : B \to C}\,\mathrm{App} \quad \dfrac{\dfrac{}{\Gamma \vdash H_2 : A \to B}\,\mathrm{Var} \quad \dfrac{}{\Gamma \vdash H_3 : A}\,\mathrm{Var}}{\Gamma \vdash H_2\ H_3 : B}\,\mathrm{App}}{\Gamma \vdash (H_1\ H_3)\,(H_2\ H_3) : C}\,\mathrm{App}}{H_1 : A \to B \to C, H_2 : A \to B \vdash \begin{array}{c}\lambda H_3 : A.\\ (H_1\ H_3)\,(H_2\ H_3)\end{array} : A \to C}\,\mathrm{Lam}}{H_1 : A \to B \to C \vdash \begin{array}{c}\lambda H_2 : A \to B . \lambda H_3 : A.\\ (H_1\ H_3)\,(H_2\ H_3)\end{array} : (A \to B) \to A \to C}\,\mathrm{Lam}}{\vdash \begin{array}{c}\lambda H_1 : A \to B \to C . \lambda H_2 : A \to B . \lambda H_3 : A.\\ (H_1\ H_3)\,(H_2\ H_3)\end{array} : (A \to B \to C) \to (A \to B) \to A \to C}\,\mathrm{Lam}$$

⇒ 証明のステップは型検査規則の適用として考えていい

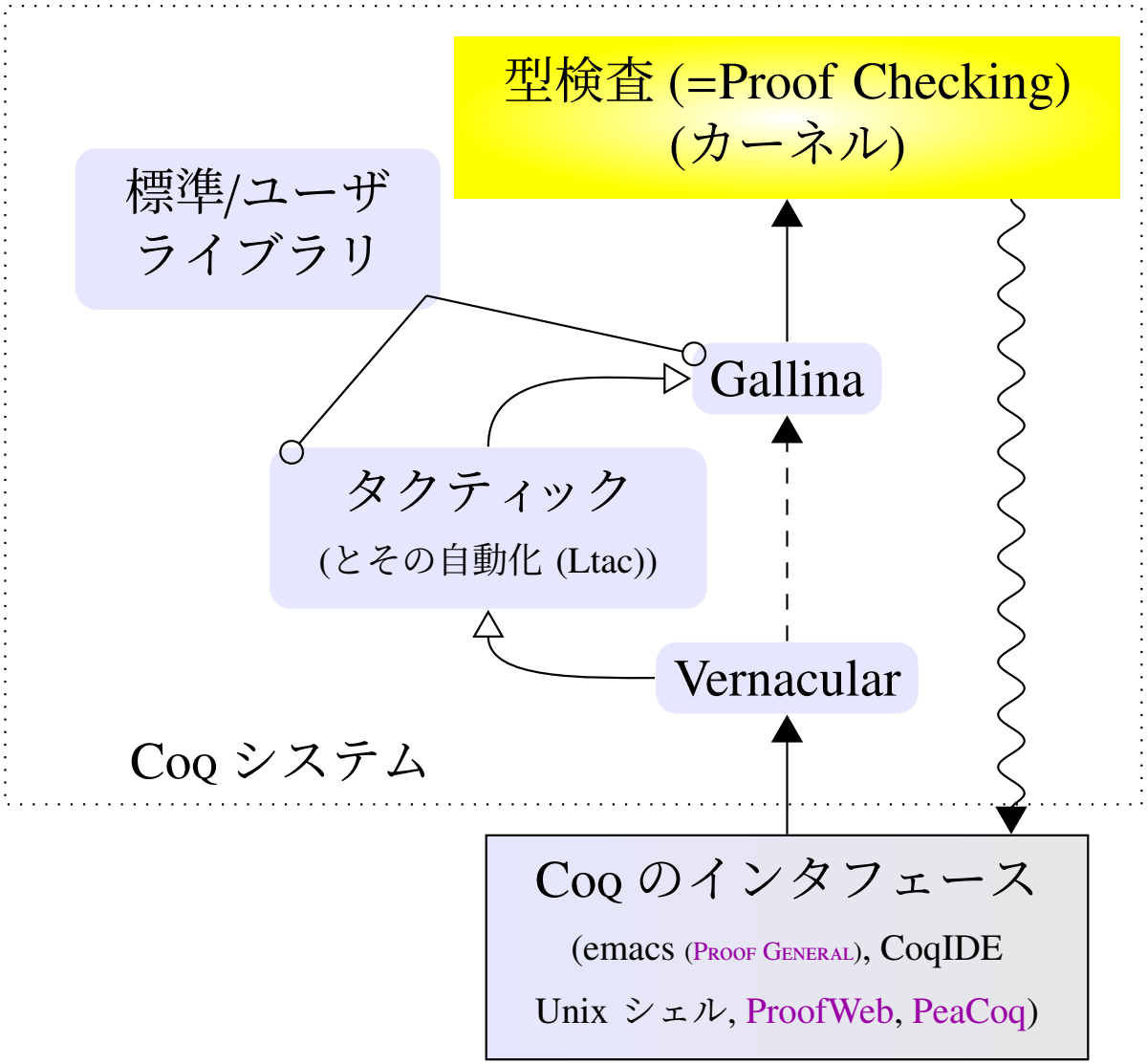
Coq による形式証明

- 証明をタクティックを用いて対話的に組み立てる
  - 直接に Gallina の項を書かない
- 紙上の証明 ≃ 連続したタクティック $\stackrel{def}{=}$ スクリプト

```
Lemma hilbertS (A B C : Prop) :
  (A -> B -> C) -> (A -> B) -> A -> C.
Proof.
move=> H1 H2 H3.
apply H1.
  exact H3.
apply H2.
exact H3.
Qed.
```

- 証明の管理は Vernacular 言語で行う
  - `Lemma`, `Goal`, `Qed` などは Gallina ではない
- Coq の表現力は命題論理に限らない!
  - 上記紹介した中核部分に依存型の拡張による (詳細を省く)

Coq システムのまとめ



—▶: Vernacular というコマンド言語で新しいデータ構造や証明を追加する

– –▶: 証明項は Gallina という関数型言語で直接記述できる

—▷: 実際は, 証明項をタクティックで間接に記述する

—○: Coq の標準ライブラリは検証済みの再利用可能なデータ構造や定理やタクティックを提供する

⇀▶: 記述した証明が型検査を通らない場合, ユーザまでフィードバック

型付きプログラミング言語 Gallina(依存型を含む)

- 項 (NB: 型を含む):

t	:=	`Prop` | `Set` | `Type`	ソート
	|	x, A	変数
	|	`forall` $x : A, B$ | $A \to B$	product
	|	`fun` x => t	関数抽象
	|	`let` x:=t_1 `in` t_2	ローカル定義
	|	$t_1\ t_2$	関数適用
	|	c	constant
	|	`match` t `with` | $pattern$ => t `end`	
	|	`fix` $f\ x : A := t$	無名の不動点

- 帰納的に定義された型の値は構成子 (つまり, constant) となり, `match` で消費する (再帰的で帰納的に定義された型なら, `fix` と組合せて使える)

型付きプログラミング言語 Gallina の片付け規則

型付規則の説明を省略 ([CDT18] に参考); 2 つの指摘だけ：

- Conversion ルール:
  - Coq が H. Poincaré 原理 (1902 年) を実現している:
    - 「2 + 2 = 4 の証明は計算の問題である」
  - Gallina 項は “definitional equality” を満たすと同じものとみなす:

$$\frac{\Gamma \vdash t : A \qquad A =_{\beta\delta\zeta\iota} B}{\Gamma \vdash t : B}\ \mathrm{Conv}$$

- Gallina の関数の停止は不可欠
  - 型検査決定可能にするため, 停止性の保証が要ることが分る
  - 停止しない関数の返り値の型は何でも相応しいから

Outline

ホーア論理の規則

- 最小の規則の集合:

$$\frac{\{P\}c\{Q\} \quad \{Q\}d\{R\}}{\{P\}c; d\{R\}}\ \text{seq} \qquad \frac{}{\{Q\{e/v\}\}v \leftarrow e\{Q\}}\ \text{assign}$$

$$\frac{\{P \wedge t = \mathit{true}\}c\{P\}}{\{P\}\texttt{while}(t)\{c\}\{P \wedge t = \mathit{false}\}}\ \text{while} \qquad \frac{P \to P' \quad \{P'\}c\{Q'\} \quad Q' \to Q}{\{P\}c\{Q\}}\ \text{conseq}$$

while 規則の条件は**不変式**と呼ぶ.

assign 規則の例:

$$\{\ \underbrace{x = 1}_{Q\{(x+1)/x\}}\ \}x \leftarrow x + 1\{\underbrace{x = 2}_{Q}\}$$

- 事前/事後条件と不変式から, 検証を自動化できるはず
- 実際に, 対話的な証明によく頼る

ホーア論理による証明の例

- プログラム $while(x \neq 0)\{ret = ret * x; x = x - 1\}$ が $x!$ を計算することを次のように示す
- 最初に目的を書く:

$$\frac{?}{\{x = X \wedge ret = 1\}while(x \neq 0)\{ret = ret * x; x = x - 1\}\{ret = X!\}}$$

ホーア論理による証明の例

$$\dfrac{\dfrac{?}{\{ret * x! = X!\}while(x \neq 0)\{ret = ret * x; x = x - 1\}\{ret * x! = X! \wedge x = 0\}}}{\{x = X \wedge ret = 1\}while(x \neq 0)\{ret = ret * x; x = x - 1\}\{ret = X!\}}\ \text{conseq}$$

ホーア論理による証明の例

$$\dfrac{\dfrac{\dfrac{?}{\{ret * x! = X! \wedge x \neq 0\}ret = ret * x; x = x - 1\{ret * x! = X!\}}}{\{ret * x! = X!\}while(x \neq 0)\{ret = ret * x; x = x - 1\}\{ret * x! = X! \wedge x = 0\}}\ \text{while}}{\{x = X \wedge ret = 1\}while(x \neq 0)\{ret = ret * x; x = x - 1\}\{ret = X!\}}\ \text{conseq}$$

ホーア論理による証明の例

$$\dfrac{\dfrac{\dfrac{}{\{ret * x! = X! \wedge x \neq 0\}\ ret = ret * x\ \{?\}} \qquad \dfrac{?}{\{?\}\ x = x - 1\ \{ret * x! = X!\}}}{\{ret * x! = X! \wedge x \neq 0\}ret = ret * x; x = x - 1\{ret * x! = X!\}}\ \text{seq}}{\{ret * x! = X!\}while(x \neq 0)\{ret = ret * x; x = x - 1\}\{ret * x! = X! \wedge x = 0\}}\ \text{while}$$

$$\dfrac{\{ret * x! = X!\}while(x \neq 0)\{ret = ret * x; x = x - 1\}\{ret * x! = X! \wedge x = 0\}}{\{x = X \wedge ret = 1\}while(x \neq 0)\{ret = ret * x; x = x - 1\}\{ret = X!\}}\ \text{conseq}$$

ホーア論理による証明の例

$$\dfrac{\dfrac{\dfrac{?}{\{ret * x! = X! \wedge x \neq 0\}\ ret = ret * x\ \{?\}} \qquad \dfrac{}{\{Q\{x-1/x\}\}\ x = x - 1\ \{\underbrace{ret * x! = X!}_{Q}\}}\ \text{assign}}{\{ret * x! = X! \wedge x \neq 0\}ret = ret * x; x = x - 1\{ret * x! = X!\}}\ \text{seq}}{\{ret * x! = X!\}while(x \neq 0)\{ret = ret * x; x = x - 1\}\{ret * x! = X! \wedge x = 0\}}\ \text{while}$$

$$\dfrac{\{ret * x! = X!\}while(x \neq 0)\{ret = ret * x; x = x - 1\}\{ret * x! = X! \wedge x = 0\}}{\{x = X \wedge ret = 1\}while(x \neq 0)\{ret = ret * x; x = x - 1\}\{ret = X!\}}\ \text{conseq}$$

ホーア論理による証明の例

$$\dfrac{\dfrac{\dfrac{?}{\{ret * x! = X! \wedge x \neq 0\}\ ret = ret * x\ \{ret * (x-1)! = X!\}} \qquad \dfrac{}{\{Q\{x-1/x\}\}\ x = x - 1\ \{\underbrace{ret * x! = X!}_{Q}\}}\ \text{assign}}{\{ret * x! = X! \wedge x \neq 0\}ret = ret * x; x = x - 1\{ret * x! = X!\}}\ \text{seq}}{\{ret * x! = X!\}while(x \neq 0)\{ret = ret * x; x = x - 1\}\{ret * x! = X! \wedge x = 0\}}\ \text{while}$$

$$\dfrac{\{ret * x! = X!\}while(x \neq 0)\{ret = ret * x; x = x - 1\}\{ret * x! = X! \wedge x = 0\}}{\{x = X \wedge ret = 1\}while(x \neq 0)\{ret = ret * x; x = x - 1\}\{ret = X!\}}\ \text{conseq}$$

ホーア論理による証明の例

$$\dfrac{\dfrac{\dfrac{}{\{P\{ret * x/ret\}\}ret = ret * x\{P\}}\ \text{assign}}{\{ret * x! = X! \wedge x \neq 0\}\ ret = ret * x\ \{\underbrace{ret * (x-1)! = X!}_{P}\}}\ \text{conseq} \qquad \dfrac{\dfrac{}{\{Q\{x-1/x\}\}x = x - 1\{Q\}}\ \text{assign}}{\{ret * (x-1)! = X! \wedge 0 \leq x - 1\}\ x = x - 1\ \{Q\}}\ \text{stren}}{\{ret * x! = X! \wedge x \neq 0\}ret = ret * x; x = x - 1\{ret * x! = X!\}}\ \text{seq}$$

$$\dfrac{\{ret * x! = X!\}while(x \neq 0)\{ret = ret * x; x = x - 1\}\{ret * x! = X! \wedge x = 0\}}{\{x = X \wedge ret = 1\}while(x \neq 0)\{ret = ret * x; x = x - 1\}\{ret = X!\}}\ \text{while}\ \text{conseq}$$

上記のリーゾニングを Coq で表現できる. ただし, インフラが要る

Outline

まとめ

- 定理証明支援系による形式検証の意義
  - 動機・歴史
- 分野紹介として, 具体的な応用例をサーベイした
  - 四色定理, Kepler 予想, C コンパイラなど
- 形式論理・定理証明支援系 Coq の基本
  - プログラム = 証明! (Curry-Howard 同型対応)
- 現実的なプログラムの形式検証の基本
  - プログラミング言語の研究の応用お一例

Coq 関する参考文献

- 解説記事
  - 紹介 [Aff14a]
  - 符号理論への応用 [Aff16]
  - 標準的な代数のライブラリの紹介 [Aff17b]
- チュートリアル, 集中講義 [Aff14c, Aff14b, Aff15, Aff17a]
  - Coq のタクティックの詳細な説明, 練習問題
- 教科書
  - 英語: [BC04, Chl13]
  - 日本語: [HA18] (2018 年 4 月発売)
- Reference manual [CDT18]

研究紹介

（興味があれば、声かけてください!）

- プログラムの形式検証
  - 多倍長整数関数・アセンブリ言語 [ANY12, Aff13]
    - 狙い: 数理計算, 暗号スキームの実装
  - C 言語のプログラム [AS14]
    - 狙い: セキュリティプロトコル, IoT（フランスとの共同研究）
- 数学の形式検証
  - 情報理論 [AHS14], 符号理論 [AG15]
    - 狙い: 確率論, 圧縮されたデータによるアルゴリズムの検証 (AI, ビッグデータに応用)
  - 幾何学 [AC17], 実数による解析
    - 狙い: ロボティックスのデザインの検証

[AC17] Reynald Affeldt and Cyril Cohen, *Formal foundations of 3D geometry to model robot manipulators*, 6th ACM SIGPLAN Conference on Certified Programs and Proofs (CPP 2017), Paris, France, January 16–17, 2017, ACM Press, Jan 2017, pp. 30–42.

[Aff13] Reynald Affeldt, *On construction of a library of formally verified low-level arithmetic functions*, Innovations in Systems and Software Engineering **9** (2013), no. 2, 59–77, NASA/Springer.

[Aff14a] ———, *Formal verification using proof-assistants: Recent results and introduction to Coq*, Information Processing **55** (2014), 482–491, (in Japanese) Information Processing Society of Japan.

[Aff14b] ———, *Formal verification using the Coq proof-assistant*, Nagoya University Graduate School of Mathematics, intensive course, Dec 2014, (in Japanese), 2014/12/15–19, `https://staff.aist.go.jp/reynald.affeldt/ssrcoq`.

[Aff14c] ———, *Introduction to the Coq proof-assistant*, 31st Meeting of the Japan Society for Software Science and Technology, Tutorial, Nagoya University, 2014/09/07, (in Japanese), Sep 2014.

[Aff15] ———, *Introduction to the Coq/SSReflect proof-assistant*, Kyoto University Department of Mathematics/Research Institute for Mathematical Sciences (RIMS), intensive course, Jul 2015, (in Japanese), 2015/07/21–24, `https://staff.aist.go.jp/reynald.affeldt/ssrcoq`.

[Aff16] ———, *Towards formal coding theory*, Chapter 15 of Evolving Theory of Error-correcting Codes, Manabu Hagiwara (editor) (2016), 171–189, (in Japanese) Nippon Hyoronsha.

[Aff17a] ———, *Introduction to Coq and Mathematical Components*, Chiba University intensive course, Jan 2017, (in Japanese), 2017/01/25–26, `https://staff.aist.go.jp/reynald.affeldt/coq`.

[Aff17b] ———, *Introduction to Mathematical Components*, Computer Software **34** (2017), 64–74, (in Japanese) Japan Society for Software Science and Technology.

[AG15] Reynald Affeldt and Jacques Garrigue, *Formalization of error-correcting codes: from Hamming to modern coding theory*, 6th Conference on Interactive Theorem Proving (ITP 2015), Nanjing, China, August 24–27, 2015, Lecture Notes in Computer Science, vol. 9236, Springer, Aug 2015, pp. 17–33.

[AGN09] Andrea Asperti, Herman Geuvers, and Raja Natarajan, *Social processes, program verification and all that*, Mathematical Structures in Computer Science **19** (2009), no. 5, 877–896.

[AHS14] Reynald Affeldt, Manabu Hagiwara, and Jonas Sénizergues, *Formalization of Shannon's theorems*, Journal of Automated Reasoning **53** (2014), no. 1, 63–103.

[ANY12] Reynald Affeldt, David Nowak, and Kiyoshi Yamada, *Certifying assembly with formal security proofs: the case of BBS*, Science of Computer Programming **77** (2012), no. 10–11, 1058–1074.

[AS14] Reynald Affeldt and Kazuhiko Sakaguchi, *An intrinsic encoding of a subset of C and its application to TLS network packet processing*, Journal of Formalized Reasoning **7** (2014), no. 1, 63–104.

[BC04] Yves Bertot and Pierre Castéran, *Interactive theorem proving and program development—Coq'Art: The calculus of inductive constructions*, Springer, 2004.

[BL09] Sandrine Blazy and Xavier Leroy, *Mechanized semantics for the Clight subset of the C language*, J. Autom. Reasoning **43** (2009), no. 3, 263–288.

[Bol10] Dominique Bolignano, *Applying formal methods in the large*, 4th International Conference on Interactive Theorem Proving (ITP 2013), Rennes, France, July 22–26, 2013, Lecture Notes in Computer Science, vol. 7998, Springer, 2010, p. 1.

[CDT18] The Coq Development Team, *The Coq proof assistant reference manual*, INRIA, 2018, Version 8.8.0. Available at: `https://coq.inria.fr/refman/`.

[CH84] Thierry Coquand and Gérard Huet, *A theory of constructions (preliminary version)*, International Symposium on Semantics of Data Types, Sophia-Antipolis, 1984, Jun. 1984.

[CH85] ———, *Constructions: A higher order proof system for mechanizing mathematics*, European Conference on Computer Algebra, Linz, Austria EUROCAL 85, April 1–3, 1985, Linz, Austria, vol. 1 (Invited Lectures), Apr. 1985, pp. 151–184.

[CH86] ———, *Concepts mathématiques et informatiques formalisés dans le calcul des constructions*, Tech. report, INRIA Rocquencourt, Apr. 1986.

[CH88] ———, *The calculus of constructions*, Information and Computation **76** (1988), 95–120.

[Chl13] Adam Chlipala, *Certified programming with dependent types—A pragmatic introduction to the coq proof assistant*, MIT Press, 2013.

[Chu40] Alonzo Church, *A formulation of the simple theory of types*, The Journal of Symbolic Logic **5** (1940), no. 2, 56–68.

[CN08] Boutheina Chetali and Quang-Huy Nguyen, *Industrial use of formal methods for a high-level security evaluation*, 15th International Symposium on Formal Methods, FM 2008, Turku, Finland, May 26–30, 2008, Lecture Notes in Computer Science, vol. 5014, Springer, 2008, pp. 198–213.

[CP90] Thierry Coquand and Christine Paulin, *Inductively defined types*, International Conference on Computer Logic (COLOG-88), Tallinn, USSR, December 1988, Lecture Notes in Computer Science, vol. 417, Springer, 1990, pp. 50–66.

[dB03] N. G. de Bruijn, *Memories of the AUTOMATH project*, Invited lecture at the Mathematics Knowledge Management Symposium 25–29 November 2003 Heriot-Watt University, Edinburgh, Scotland, Nov. 2003, Available at: `http://www.win.tue.nl/automath/aboutautomath.htm`. Last access: 2018/06/04.

[GAA+13] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi, and Laurent Théry, *A machine-checked proof of the odd order theorem*, 4th International Conference on Interactive Theorem Proving (ITP 2013), Rennes, France, July 22–26, 2013, 2013, pp. 163–179.

[Gon05] Georges Gonthier, *A computer-checked proof of the four colour theorem*, Tech. report, Microsoft Research, Cambridge, 2005, Available at: `http://www2.tcs.ifi.lmu.de/~abel/lehre/WS07-08/CAFR/4colproof.pdf`. Last access: 2018/06/04.

[Gon08] ———, *Formal proof—the four-color theorem*, Notices of the American Mathematical Society **55** (2008), no. 11, 1382–1393.

[HA18] Manabu Hagiwara and Reynald Affeldt, *Formal proof using Coq/SSReflect/MathComp: Start formalization of mathematics with free software*, Morikita Publishing, 2018, (in Japanese).

[Hal08] Thomas C. Hales, *Formal proof*, Notices of the American Mathematical Society **55** (2008), no. 11, 1370–1380.

[Hal12] ———, *Lessons learned from the flyspeck project*, International Spring School on Formalization of Mathematics (MAP 2012), March 12–16, 2012, Sophia Antipolis, France, Mar. 2012, Available at: `http://www-sop.inria.fr/manifestations/MapSpringSchool/contents/ThomasHales.pdf`. Last access: 2018/06/04.

[Hal13] ———, *Mathematics in the age of the Turing machine*, arXiv:1302.2898v1 [math.HO], Feb. 2013.

[How80] William A. Howard, *To H. B. Curry: Essays on combinatory logic, lambda calculus and formalism*, ch. The formulae-as-types notion of construction, pp. 479–490, Academic Press Inc., Sep. 1980, Original paper manuscript from 1969.

[Kle10] Gerwin Klein, *From a verified kernel towards verified systems*, 8th Asian Symposium on Programming Languages and Systems (APLAS 2010), Shanghai, China, November 28–December 1, 2010, Lecture Notes in Computer Science, vol. 6461, Springer, 2010, pp. 21–33.

[Ler09] Xavier Leroy, *A formally verified compiler back-end*, J. Autom. Reasoning **43** (2009), no. 4, 363–446.

[PM92] Christine Paulin-Mohring, *Inductive definitions in the sytem Coq rules and properties*, Tech. report, LIP, École Normales Supérieure de Lyon, Dec. 1992.

[Rus08] Bertrand Russell, *Mathematical logic as based on the theory of types*, American Journal of Mathematics **30** (1908), no. 3, 222–262.

[vH02] Jean van Heijenoort, *From Frege to Gödel—a source book in mathematical logic, 1879-1931*, Harvard University Press, 2002.

[Voe14] Vladimir Voevodsky, *Univalent foundations*, March 2014, Lecture at IAS. Available at: `http://www.math.ias.edu/~vladimir/Site3/Univalent_Foundations_files/2014_IAS.pdf`. Last access: 2018/06/04.

[Wie14] Freek Wiedijk, *Formal proof done right*, Workshop on Formalization of Mathematics in Proof Assistants, Institut Henri Poincaré, May 2014, Oral presentation.

[WKS+09] Simon Winwood, Gerwin Klein, Thomas Sewell, June Andronick, David Cock, and Michael Norrish, *Mind the gap*, 22nd International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2009), Munich, Germany, August 17–20, 2009, Lecture Notes in Computer Science, vol. 5674, Springer, 2009, pp. 500–515.

[YCER11] Xuejun Yang, Yang Chen, Eric Eide, and John Regehr, *Finding and understanding bugs in C compilers*, 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4–8, 2011, ACM, 2011, pp. 283–294.