The Story Begins...

- **Gottlob Frege**: a German mathematician who started in geometry but became interested in logic and foundations of arithmetic.

- **1879**: Published “Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens” (Concept-Script: A Formal Language for Pure Thought Modeled on that of Arithmetic)
  - The birth of modern mathematical logic
  - First notation able to express arbitrarily complicated logical statements with quantifiers
    - Two-dimensional notation!
      - $\exists x : F(x)$
      - $\bigwedge x F(x)$
Formalization of Arithmetic

• 1884: *Die Grundlagen der Arithmetik* (The Foundations of Arithmetic)
• 1893: *Grundgesetze der Arithmetik* (Basic Laws of Arithmetic, Vol. 1)
• 1903: *Grundgesetze der Arithmetik* (Basic Laws of Arithmetic, Vol. 2)

• Frege’s goals:
  – isolate logical principles of inference
  – derive laws of arithmetic from first principles
  – set mathematics on a solid foundation of logic

The plot thickens...

Just as Volume 2 was going to print in 1903, Frege received a letter...
Bertrand Russell

• **Russell’s paradox:**

1. Set comprehension notation:
   \[ \{ z \mid P(z) \} \] “The set of z such that P(z)”

2. Let \( X \) be the set \( \{ Y \mid Y \not\in X \} \).

3. Ask the logical question:
   Does \( X \in X \) hold?

4. Paradox!
   If \( X \in X \) then \( X \not\in X \).
   If \( X \not\in X \) then \( X \in X \).

• Russell’s paradox destroyed Frege’s logical foundations...

Bertrand Russell
1872 - 1970
“Hardly anything more unfortunate can befall a scientific writer than to have one of the foundations of his edifice shaken after the work is finished. This was the position I was placed in by a letter of Mr. Bertrand Russell, just when the printing of this volume was nearing its completion.” – Frege, 1903
Aftermath of Frege and Russell

• Frege came up with a fix, but it made his logic trivial...

• 1908: Russell fixed the inconsistency of Frege’s logic by developing a *theory of types*.

• 1910, 1912, 1913, (revised 1927): *Principia Mathematica* (Whitehead & Russell)
  – Goal: axioms and rules from which *all* mathematical truths could be derived.
  – It was a bit unwieldy...

"From this proposition it will follow, when arithmetical addition has been defined, that 1+1=2."
—Volume I, 1st edition, *page 379*
Aftermath of Frege and Russell

For an entertaining (!) account mainly from Russell’s POV:
Logic in the 1930s and 1940s

- **1931:** Kurt Gödel’s first and second incompleteness theorems.
  - Demonstrated that any consistent formal theory capable of expressing arithmetic cannot be complete.

- **1936:** Church introduces the λ-calculus.

- **1936:** Turing introduces Turing machines.
  - Both independently use their technique to show that there is no algorithm that can determine whether a statement of first-order logic is provable.

- **1940:** Church introduces the *simple theory of types, aka higher-order logic*
Fast Forward...

- **1958** (Haskell Curry) and **1969** (William Howard) observe a remarkable correspondence:
  
<table>
<thead>
<tr>
<th>types</th>
<th>~</th>
<th>formulas</th>
</tr>
</thead>
<tbody>
<tr>
<td>programs</td>
<td>~</td>
<td>proofs</td>
</tr>
<tr>
<td>computation</td>
<td>~</td>
<td>simplification</td>
</tr>
</tbody>
</table>

- **1967 – 1980’s**: N.G. de Bruijn runs Automath project
  - uses the **Curry-Howard correspondence** for computer-verified mathematics

- **1971**: Jean-Yves Girard introduces **System F**
- **1972**: Girard introduces **F_\omega**
- **1972**: Per Martin-Löf introduces **intuitionistic type theory**
- **1974**: John Reynolds independently discovers **System F**

Higher order lambda calculi. Basis for modern type systems: OCaml, Haskell, Scala, Java, C#, ...
... to the Present

- **1984**: Coquand and Huet first begin implementing a new theorem prover “Coq”
- **1985**: Coquand introduces the calculus of constructions
  - combines features from intuitionistic type theory and F\(\omega\)
- **1989**: Coquand and Paulin extend CoC to the calculus of inductive constructions
  - adds “inductive types” as a primitive
- **1992**: Coq ported to Xavier Leroy’s Caml

- **1990’s**: up to Coq version 6.2
- **2000-2010**: Coq version 8.3
- **2012**: Coq version 8.4  ← We are here

Too many contributors to mention here...
So much for logical foundations... what about software?

SOFTWARE FOUNDATIONS
Building Reliable Software

• Suppose you work at (or run) a software company.

• Suppose, like Frege, you’ve sunk 30+ person-years into developing the “next big thing”:
  – Boeing Dreamliner2 flight controller
  – Autonomous vehicle control software for Nissan
  – Gene therapy DNA tailoring algorithms
  – Super-efficient green-energy power grid controller

• Suppose, like Frege, your company has invested a lot of material resources that are also at stake.

• How do you avoid getting a letter like the one from Russell?

Or, worse yet, *not* getting the letter to disastrous consequences?
Approaches to Reliability

• Social
  – Code reviews
  – Extreme/Pair programming

• Methodological
  – Design patterns
  – Test-driven development
  – Version control
  – Bug tracking

• Technological
  – Static analysis
    (“lint” tools, FindBugs, …)
  – Fuzzers

• Mathematical
  – Sound type systems
  – “Formal” verification

Less “formal”: Techniques may miss problems in programs

All of these methods should be used!

Even the most “formal” can still have holes:
  • did you prove the right thing?
  • do your assumptions match reality?

More “formal”: eliminate with certainty as many problems as possible.
This Course: Five Interwoven Threads

1. basic tools from logic for making and justifying precise claims about programs

2. the use of proof assistants to construct rigorous, machine checkable, logical arguments

3. the idea of functional programming, both as a method of programming and as a bridge between programming and logic

4. techniques for formal verification of properties of specific programs

5. the use of type systems for establishing guarantees for all programs in a given language (we might not make it this far)
Can it Scale?

- Use of theorem proving to verify “real” software is still considered to be the bleeding edge of PL research.

- **CompCert** – fully verified C compiler
  Leroy, INRIA

- **Ynot** – verified DBMS, web services
  Morrisett, Harvard

- **Verified Software Toolchain**
  Appel, Princeton

- **Bedrock**
  Chlipala, MIT

- **CertiKOS** – certified OS kernel
  Shao & Ford, Yale

- **Vellvm** – formalized LLVM IR
  Zdancewic, Penn
Does it work?

Finding and Understanding Bugs in C Compilers [Yang et al. PLDI 2011]

Random test-case generation

Source Programs

GCC

LLVM

{8 other C compilers}

64

Open

79 bugs: 25 critical

202 bugs

325 bugs in total

Verified Compiler: CompCert [Leroy et al.]

<10 bugs found in unverified front-end component
Yang et al. Conclude...

The striking thing about our CompCert results is that the middle-end bugs we found in all other compilers are absent. As of early 2011, the under-development version of CompCert is the only compiler we have tested for which Csmith cannot find wrong-code errors. This is not for lack of trying: we have devoted about six CPU-years to the task. The apparent unbreakability of CompCert supports a strong argument that developing compiler optimizations within a proof framework, where safety checks are explicit and machine-checked, has tangible benefits for compiler users.

(emphasis mine)
COQ
Coq in this Course

- We’ll use Coq version 8.4 (pl4 and pl3 are be fine)
  - Available in MEng/MPS lab on ground floor of Gates
  - Easy to install binaries on your own machine

- Two different user interfaces:
  - CoqIDE: a standalone GUI including (bare bones) editor
  - Proof General: an Emacs-based editing environment
Coq’s Full System
Subset Used in CSci 6907.85

To start.

By the end of the course
Getting acquainted with Coq.

BASICS.V