universität
innsbruck

Machine Learning for Theorem Proving

Lecture 1 (VU)

Cezary Kaliszyk

Administration (1/2)

VU3, Wednesdays 8:15am

- Attendance required
- Electronic devices useful

Grading

- Activities in the sessions
- Home assignments
- Presentations
- Final in-class assignment

Administration (2/2)

Course Prerequisites

- Logic basics
  - I assume the knowledge of propositional logic
  - When it comes to predicate logic, I will repeat some, in particular unification as it will come useful in the course
- Lambda-calculus
  - I assume the basic knowledge of untyped lambda calculus, I will repeat typing as it will be useful for theorem proving
- AI basics
  - Most basic AI algorithms will be re-introduced as we will need to adapt them to theorem proving problems

Covered Topics (1/3)

First Part

- kinds of theorem proving systems
- basic machine learning problems
- "traditional" methods to deal with them
  (that the whole domain is 20 years old so by traditional I mean 2000-2015)

Second Part

- new techniques and developments
- some very promising or controversial
- major results in the last five years

Covered Topics (2/3)

Part 1

Theorem proving systems Proof assistants, Automated theorem provers, ... and other systems where learning is of major use

Machine learning problems How do the theorem proving problems correspond to supervised learning, unsupervised, reinforcement learning and what are the specifics of the problems (little data, lots of features, complicated feature space, ...)

Lemma relevance First problem: How to select relevant facts in a large base of mathematical knowledge

Lemma mining Second problem: Given some mathematical knowledge can we use a computer to predict other likely statements? What about likely intermediate facts for a proof?

ATP guidance Can machine learning guide an existing prover with a fixed calculus and what kind of adaptations could be useful?

Strategy selection In many domains running multiple complementary strategies for short time is better then focusing on one strategy. Can we use learning to predict useful ones?

Covered Topics (3/3)

Part 2

Feature engineering Can we use the knowledge that we have about mathematics to characterize formulas / statements / proof state more meaningfully than by their syntax?

Deep learning Are there also features that can be learned

Statistical alignments Given different foundations, provers, and libraries can we find similar concepts to increase the amount of knowledge for learning?

Automating formalization Is it possible for machine learning to interpret human-written mathematical knowledge (for example in LaTeX) as opposed to defined logical syntax

Naming, metrics, refactoring Are learning methods for software engineering also applicable to proof repositories?

Unsupervised methods Can we automatically notice relations in the space of theorems

Kinds of theorem proving systems

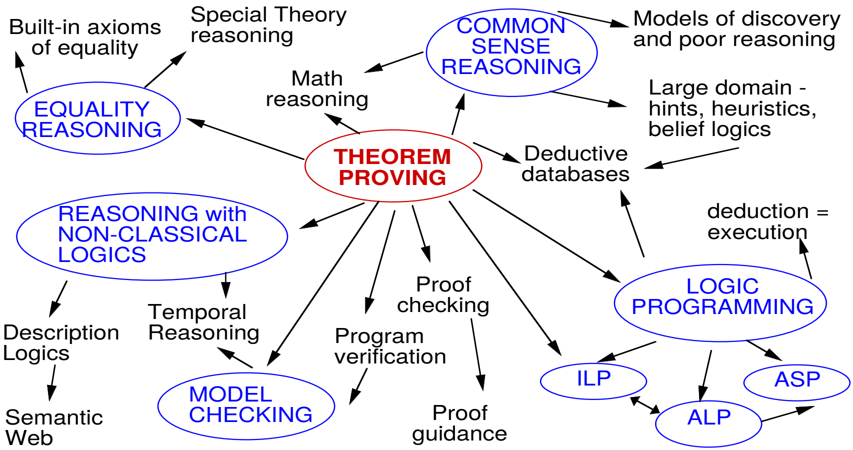
Research domain: Automated Reasoning

- Field of research since the fifties
- Understood as:

  Computer used to reason in a logic
- Traditionally part of artificial intelligence
  - But do not confuse it with machine learning: It focused on calculi, their properties and fixed human defined algorithms
- Many applications today

  program verification, mathematical deduction, ...
- Many different systems, different logics, different levels of precision, of automation. The systems themselves vary a lot.

Spread of theorem proving (1/2)



[diagram by K. Broda]

Spread of theorem proving (2/2)

Propositional logic

SAT

SMT

SUMO

First-order logic

Resolution

Orders

Paramodulation

Superposition

Non-classical

Description Logics

Modal Logics

ASP

Higher-order logics

Logical Frameworks

Type Theory

Set Theory

Prolog (Lambda)

ACL2

What is a Proof Assistant? (1/2)

A Proof Assistant is a

a computer program to assist a mathematician in the production of a proof that is mechanically checked

[defintion by H. Geuvers]

- By mathematician we also mean a computer scientist
- The assistance can be interpreted as just checking or some advice

What does a Proof Assistant do?

- Keep track of theories, definitions, assumptions
- Interaction - proof editing
- Proof checking
- Automation - proof search

What does it implement? (And how?)

- a formal logical system intended as foundation for mathematics
  - A language for stating properties and inference rules
- decision procedures

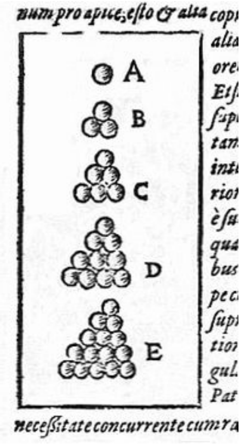
Naming convention

In the literature the names:

- Proof Assistant
- Interactive Theorem Prover

Are used as synonyms.

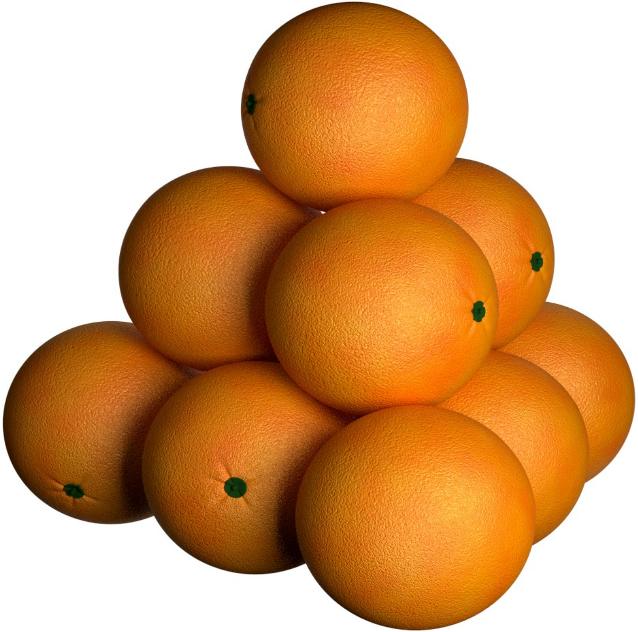
Cast study for proof assistants: The Kepler Conjecture



In year 1611 Kelper claimed that:

The most compact way of stacking balls of the same size in space is a pyramid.

$$V = \frac{\pi}{\sqrt{18}} \approx 74\%$$



The conjecture was open for almost 400 years!

Proved in 1998

- Tom Hales wrote a 300 page proof that additionally used computer programs
- Submitted to the Annals of Mathematics.
- The reviewers looked at it for 5 years and came back saying: We are 99% sure this is correct. We cannot be sure of the programs.

The conjecture was open for almost 400 years!

Proved in 1998

- Tom Hales wrote a 300 page proof that additionally used computer programs
- Submitted to the Annals of Mathematics.
- The reviewers looked at it for 5 years and came back saying: We are 99% sure this is correct. We cannot be sure of the programs.

Programs enumerated graphs with certain properties

Programs checked 1039 equalities and inequalities

For example computer algebra used to say that this is true:

$$\frac{\begin{array}{c}-x_1x_3-x_2x_4+x_1x_5+x_3x_6-x_5x_6+\\+x_2(-x_2+x_1+x_3-x_4+x_5+x_6)\end{array}}{\sqrt{4x_2\left(\begin{array}{c}x_2x_4(-x_2+x_1+x_3-x_4+x_5+x_6)+\\+x_1x_5(x_2-x_1+x_3+x_4-x_5+x_6)+\\+x_3x_6(x_2+x_1-x_3+x_4+x_5-x_6)-\\-x_1x_3x_4-x_2x_3x_5-x_2x_1x_6-x_4x_5x_6\end{array}\right)}} < \tan(\frac{\pi}{2}-0.74)$$

Solution? Formalize the proof!

Both the informal text and the programs

- Formalize the 300 pages of the proof using Proof Assistants
- Implement the code for checking the properties in the proof assistants
- Prove the code correct
- Run the programs inside the Proof Assistant

Flyspeck Project took 20 years

- Project results published 2017
- Many Proof Assistants and contributors, major success

Case Study for Proof assistants

In 1994 Intel released the new processor Pentium P5

Floating Point Unit has an efficient machine instruction for div

- It performs a lookup in a division lookup table and computes some offset to get results of the division according to the IEEE specification
- However there was a bug: For certain inputs division result were off

Case Study for Proof assistants

In 1994 Intel released the new processor Pentium P5

Floating Point Unit has an efficient machine instruction for div

- It performs a lookup in a division lookup table and computes some offset to get results of the division according to the IEEE specification
- However there was a bug: For certain inputs division result were off

Replacement

- Few customers cared, still 450M$
- This was the birth of one of the proof assistants that we will see in the course (HOL Light)
- Intel and AMD processors are formally verified since then

What does a formal proof look like

```
theorem sqrt2_not_rational:
  "sqrt (real 2) ∉ ℚ"
proof





















qed
```

What does a formal proof look like

```
theorem sqrt2_not_rational:
  "sqrt (real 2) ∉ ℚ"
proof
  assume "sqrt (real 2) ∈ ℚ"








  thus False
qed
```

What does a formal proof look like

```
theorem sqrt2_not_rational:
  "sqrt (real 2) ∉ ℚ"
proof
  assume "sqrt (real 2) ∈ ℚ"
  then obtain m n :: nat where
    n_nonzero: "n ≠ 0" and sqrt_rat: "¦sqrt (real 2)¦ = real m / real n"
    and lowest_terms: "gcd m n = 1" ..
```

```
  thus False
qed
```

What does a formal proof look like

```
theorem sqrt2_not_rational:
  "sqrt (real 2) ∉ ℚ"
proof
  assume "sqrt (real 2) ∈ ℚ"
  then obtain m n :: nat where
    n_nonzero: "n ≠ 0" and sqrt_rat: "¦sqrt (real 2)¦ = real m / real n"
    and lowest terms: "gcd m n = 1" ..






          have eq: "m² = 2 * n²"
  hence "2 dvd m²" ..
                    have dvd m: "2 dvd m"




  hence "2 dvd n²" ..
                    have "2 dvd n"
             have "2 dvd gcd m n"

  thus False
qed
```

What does a formal proof look like

```
theorem sqrt2_not_rational:
  "sqrt (real 2) ∉ ℚ"
proof
  assume "sqrt (real 2) ∈ ℚ"
  then obtain m n :: nat where
    n_nonzero: "n ≠ 0" and sqrt_rat: "¦sqrt (real 2)¦ = real m / real n"
    and lowest_terms: "gcd m n = 1" ..
  from n_nonzero and sqrt_rat have "real m = ¦sqrt (real 2)¦ * real n" by simp
  then have "real (m²) = (sqrt (real 2))² * real (n²)"
    by (auto simp add: power2_eq_square)
  also have "(sqrt (real 2))² = real 2" by simp
  also have "... * real (m²) = real (2 * n²)" by simp
  finally have eq: "m² = 2 * n²" ..
  hence "2 dvd m²" ..
  with two_is_prime have dvd_m: "2 dvd m" by (rule prime_dvd_power_two)
  then obtain k where "m = 2 * k" ..
  with eq have "2 * n² = 2² * k²" by (auto simp add: power2_eq_square mult_ac)
  hence "n² = 2 * k²" by simp
  hence "2 dvd n²" ..
  with two_is_prime have "2 dvd n" by (rule prime_dvd_power_two)
  with dvd_m have "2 dvd gcd m n" by (rule gcd_greatest)
  with lowest_terms have "2 dvd 1" by simp
  thus False by arith
qed
```

What does a formal proof look like

```
theorem sqrt2_not_rational:
  "sqrt (real 2) ∉ ℚ"
proof
  assume "sqrt (real 2) ∈ ℚ"
  then obtain m n :: nat where
    n_nonzero: "n ≠ 0" and sqrt_rat: "¦sqrt (real 2)¦ = real m / real n"
    and lowest_terms: "gcd m n = 1" ..
  from n_nonzero and sqrt_rat have "real m = ¦sqrt (real 2)¦ * real n" by simp
  then have "real (m²) = (sqrt (real 2))² * real (n²)"
    by (auto simp add: power2_eq_square)
  also have "(sqrt (real 2))² = real 2" by simp
  also have "... * real (m²) = real (2 * n²)" by simp
  finally have eq: "m² = 2 * n²" ..
  hence "2 dvd m²" ..
  with two_is_prime have dvd_m: "2 dvd m" by (rule prime_dvd_power_two)
  then obtain k where "m = 2 * k" ..
  with eq have "2 * n² = 2² * k²" by (auto simp add: power2_eq_square mult_ac)
  hence "n² = 2 * k²" by simp
  hence "2 dvd n²" ..
  with two_is_prime have "2 dvd n" by (rule prime_dvd_power_two)
  with dvd_m have "2 dvd gcd m n" by (rule gcd_greatest)
  with lowest_terms have "2 dvd 1" by simp
  thus False by arith
qed
```

For more explanation see Wiedijk's “17 Provers of the World”
https://www.springer.com/gp/book/9783540307044

Proof Assistant: Longer definition

- Keep track of theories, definitions, assumptions
  - set up a theory that describes mathematical concepts (or models a computer system)
  - express logical properties of the objects
- Interaction - proof editing
  - typically interactive
  - specified theory and proofs can be edited
  - provides information about required proof obligations
  - allows further refinement of the proof
  - often manually providing a direction in which to proceed.
- Automation - proof search
  - various strategies
  - decision procedures
- Proof checking
  - checking of complete proofs
  - sometimes providing certificates of correctness
- Why should we trust it?
  - small core

Can a Proof Assistant do all proofs?

Can a Proof Assistant do all proofs?

No: Think of decidability

- Validity of formulas is undecidable
- (for non-trivial logical systems, already semi-decidable in first-order logic, undecidable in more complex systems)

Automated Theorem Provers can do all proofs?

Can a Proof Assistant do all proofs?

No: Think of decidability

- Validity of formulas is undecidable
- (for non-trivial logical systems, already semi-decidable in first-order logic, undecidable in more complex systems)

Automated Theorem Provers can do all proofs?

- Work in specific domains
- Require and adjustment of your problem
- Answers: Valid (Theorem with proof)
- Or: Countersatisfiable (Possibly with counter-model)
- But often will diverge...

Proof Assistants

- Are generally applicable
- Direct modelling of problems
- But need to be interactive

What are the other classes of tools?

ATPs

- Built in automation (model elimination, resolution)
- Most known tools: Vampire, Eprover, SPASS, ...
- Applications: Robbin's conjecture, Program verification, and specific domains of mathematics

Model Checkers

- Space state abstraction
- Spin, Uppaal, ...

Computer Algebra

- Solving equations, simplifications, numerical approximations
- Maple, Mathematica, ...

Additional Literature (not required)

These are much more detailed than needed for this course.
May be useful for your final topic.

- Andrea Asperti, Herman Geuvers, and Raja Natarajan.
  Social processes, program verification and all that.
  Mathematical Structures in Computer Science, 19(5):877–896, 2009.

- John Harrison, Josef Urban, and Freek Wiedijk.
  History of interactive theorem proving.
  In Jörg H. Siekmann, editor, *Computational Logic*, volume 9 of *Handbook of the History of Logic*, pages 135–214. Elsevier, 2014.

- Freek Wiedijk, editor.
  The Seventeen Provers of the World, Foreword by Dana S. Scott, volume 3600 of *Lecture Notes in Computer Science*.
  Springer, 2006.

Summary

This Lecture

- Theorem Proving Overview
- Proof Assistants
- Comparison with other tools

Next

- Machine Learning problems
- Details on the problems
- Lemma selection
- Statistical methods
- k-nearest neighbours and naive Bayes classifiers

Work Here / Homework

Find Common Formats for Theorem Proving Problems

- What are their specifics?

Work Here / Homework

Find Common Formats for Theorem Proving Problems

- What are their specifics?

Specify one LICS exercise in TPTP

- Can an automated prover solve it?
- Compare the number of useful/unused inferences/proof size

Work Here / Homework

Find Common Formats for Theorem Proving Problems

- What are their specifics?

Specify one LICS exercise in TPTP

- Can an automated prover solve it?
- Compare the number of useful/unused inferences/proof size

Specify it in an interactive theorem prover

- Can you perform any natural-deduction like inference step?

Work Here / Homework

Find Common Formats for Theorem Proving Problems

- What are their specifics?

Specify one LICS exercise in TPTP

- Can an automated prover solve it?
- Compare the number of useful/unused inferences/proof size

Specify it in an interactive theorem prover

- Can you perform any natural-deduction like inference step?