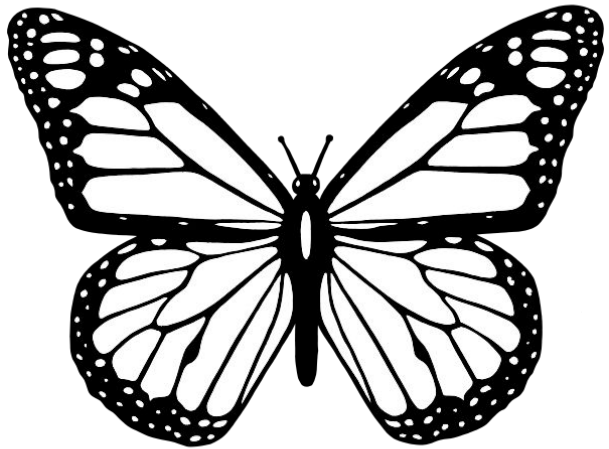
Formally Correct Deduction Methods for Computational Logic

PhD defense, Asta Halkjær From
DTU Compute, 2023-04-21

Formalities

- Study period — 2020-02-01 to 2023-01-31
- Supervisor — Jørgen Villadsen
- Co-supervisor — Nina Gierasimczuk
- Committee — Thomas Bolander
  Torben Braüner
  Lawrence C. Paulson
- Chairperson — Carsten Witt



Prelude

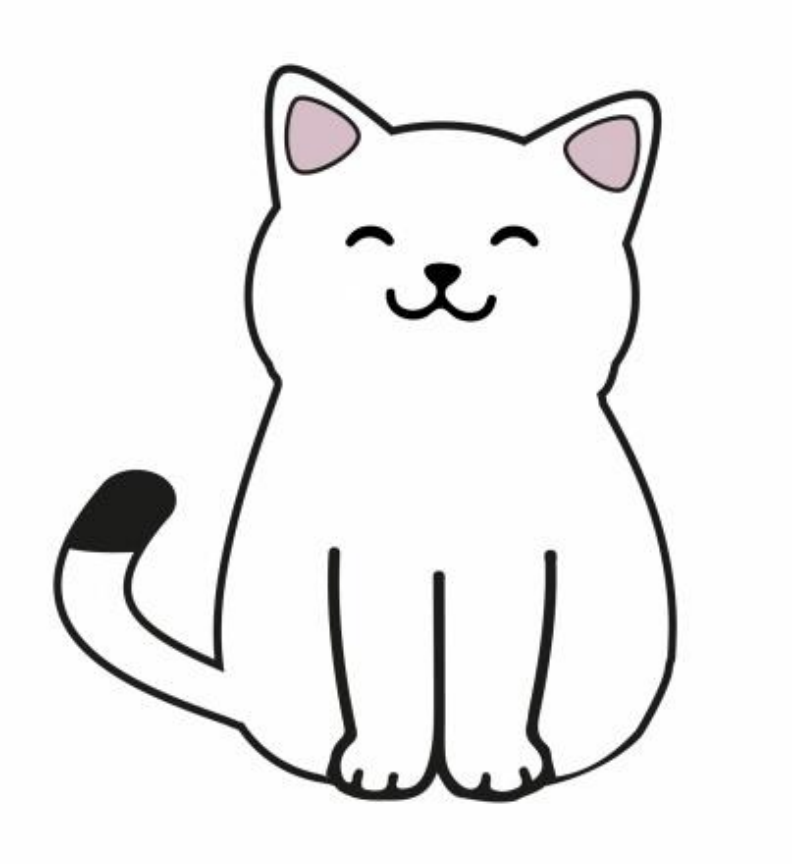
Deduction

Inferring new knowledge:

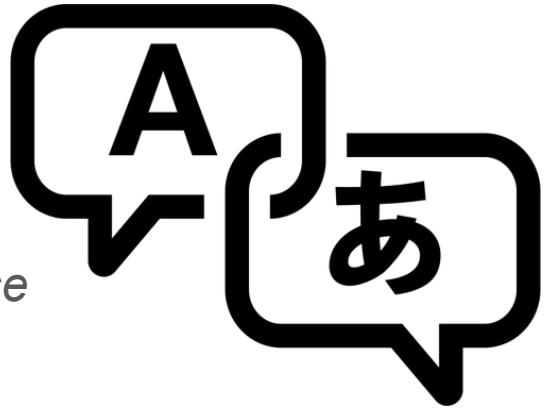
- **All** men **are** mortal
- Socrates **is a** man
- \+ Socrates **is** mortal

We study general principles:

- **All** cats **are** cute
- Betty **is a** cat
- \+ Betty **is** cute



Deduction *Methods* – Languages



We use languages with *placeholders*

if *water is wet* **then** *the moon is made of green cheese*

if *2 + 2 = 5* **then** *the sun shines at night*

if *x* **then** *y*

An *interpretation* assigns truth values to placeholders:

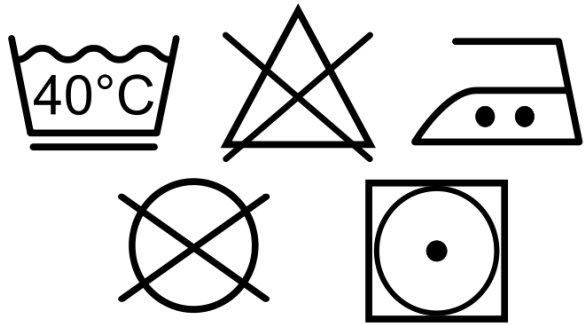
x := True, *y* := False or *x* := False, *y* := True or …

A *valid* formula is true under all interpretations

Valid: **if** *x* **and** *y* **then** *x*

Not valid: **if** *x* **then** *x* **and** *y*

Deduction *Methods* – Propositional Logic



We need a formal syntax

A *formula p* or *q* in *propositional logic* is either:

- a prop. symbol: *x, y* — a placeholder
- falsity: $\bot$ — “false”
- a conjunction: $p \wedge q$ — “*p* is true and *q* is true”
- an implication: $p \longrightarrow q$ — “if *p* is true then *q* is true”

Example: $p \longrightarrow q \longrightarrow p \wedge q$

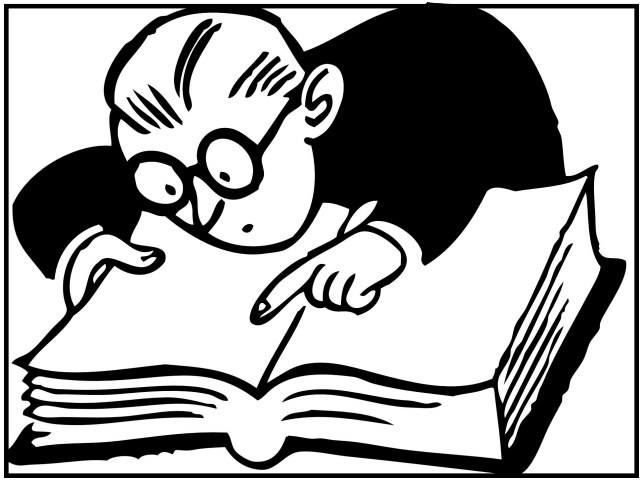
How do we *prove* that such a formula is valid?

Deduction *Methods* – Natural Deduction

$A \vdash p$ means “the assumptions A imply the conclusion p”

- \+ $p, A \vdash p$
- ● $A \vdash \bot$
- \+ $A \vdash p$
- ● $A \vdash p \land q$
- \+ $A \vdash p$
- ● $A \vdash p \land q$
- \+ $A \vdash q$
- ● $A \vdash p$
- ● $A \vdash q$
- \+ $A \vdash p \land q$
- ● $A \vdash p \longrightarrow q$
- ● $A \vdash p$
- \+ $A \vdash q$
- ● $p, A \vdash q$
- \+ $A \vdash p \longrightarrow q$

Natural Deduction *Example*



We can always infer assumptions:

+ $p, q \vdash p$
+ $p, q \vdash q$

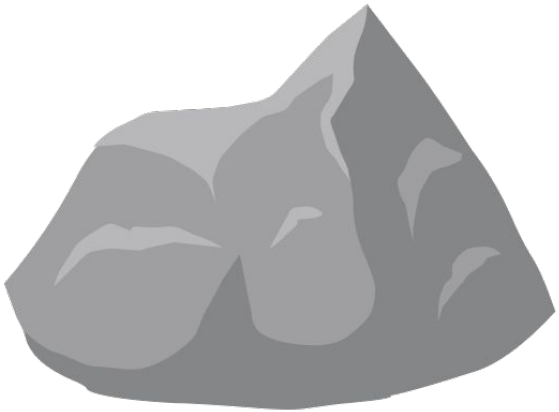
So we can infer the conjunction $p \wedge q$:

+ $p, q \vdash p \wedge q$

Then we can turn the assumptions into implications:

+ $p \vdash q \longrightarrow p \wedge q$
+ $\vdash p \longrightarrow q \longrightarrow p \wedge q$

Correct Deduction Methods



Logic famously turns mothers into rocks:

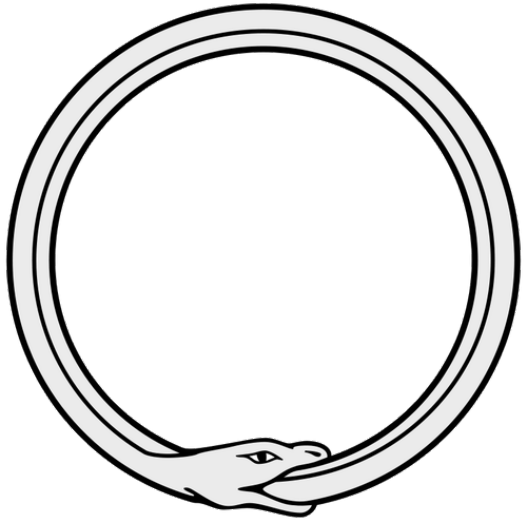
- Rocks cannot fly
- Mama cannot fly
+ Mama is a rock

No! The conclusion does not follow from the premises

Two important properties for deduction methods:

Soundness:	every provable formula is *valid*	(only good things)
Completeness:	every *valid* formula is provable	(every good thing)

Formally Correct Deduction Methods



How to prove e.g. soundness and completeness?

- By arguments in English/Danish/…?
- By arguments in a larger logic?

A formal logic gives a solid foundation

With a formal logic we can get computer assistance

Higher-Order Logic = Functional Programming + Logic

More of the same: syntax + natural deduction rules

Computational Logic

“Computational logic is the use of computers to establish facts in a logical formalism.”
– Larry Paulson

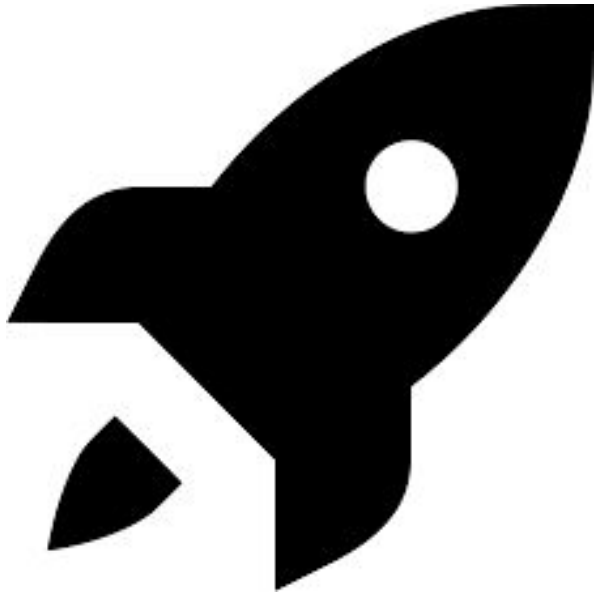
Isabelle/HOL is a *proof assistant* built on higher-order logic:

- The language of HOL
- Proof system for HOL
- *Automatic verification of our arguments*
- *Proof search*
- *Counterexample generation*
- *Code export*

Applications

Many things deemed worthy of formal verification:

- Encryption on the internet
- The coprocessor in your smartphone
- Amazon Web Services privacy
- Railway signalling
- Programming language compilers
- Mathematics



What is Isabelle/HOL like? – Syntax

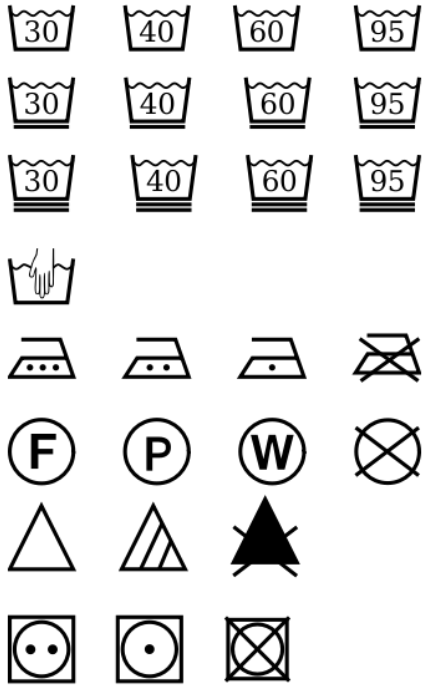
Explain the syntax to the computer:

```
datatype formula
  = Proposition string (‹·›)
  | Falsity (‹⊥›)
  | Conjunction formula formula (infixr ‹⋀› 65)
  | Implication formula formula (infixr ‹⟶› 55)
```

Now we can write our example formula:

```
term ‹p ⟶ q ⟶ p ⋀ q›
```

```
term ‹p ⟶ q ⟶ p ⋀ ›
```



What is Isabelle/HOL like? – Deduction

Explain the deduction rules to the computer:

```
inductive Deduction :: ‹formula set ⇒ formula ⇒ bool›
  (‹_ ⊢ _› [50, 50] 50) where
  Assm:  ‹p, A ⊢ p›
| FlsE:  ‹A ⊢ ⊥ ⟹ A ⊢ p›
| ConE1: ‹A ⊢ p ⋀ q ⟹ A ⊢ p›
| ConE2: ‹A ⊢ p ⋀ q ⟹ A ⊢ q›
| ConI:  ‹A ⊢ p ⟹ A ⊢ q ⟹ A ⊢ p ⋀ q›
| ImpE:  ‹A ⊢ p ⟶ q ⟹ A ⊢ p ⟹ A ⊢ q›
| ImpI:  ‹p, A ⊢ q ⟹ A ⊢ p ⟶ q›
```

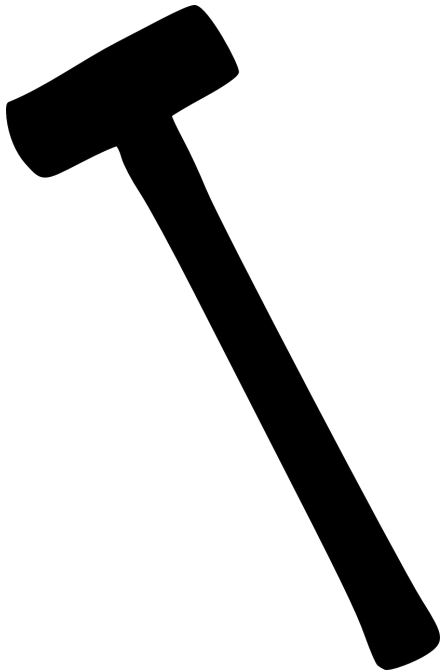
What is Isabelle/HOL like? – Deduction Example

We can type out our example:

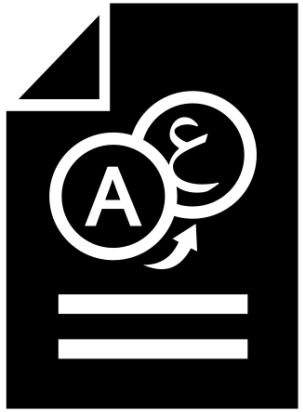
```
have ‹{p, q} ⊢ p› and ‹{p, q} ⊢ q›
  using Assm by (simp, metis insert_commute)+
then have ‹{p, q} ⊢ p ∧ q›
  using ConI by blast
then have ‹{p} ⊢ q ⟶ p ∧ q›
  using ImpI by (metis insert_commute)
then show ‹{} ⊢ p ⟶ q ⟶ p ∧ q›
  using ImpI by blast
```

Or prove it automatically:

```
lemma ‹{} ⊢ p ⟶ q ⟶ p ∧ q›
  sledgehammer (* by (metis Assm ConI ImpI insert_commute) *)
```



What is Isabelle/HOL like? – Semantics



Explain the meaning of formulas. `I` interprets the placeholders:

```
primrec semantics :: ‹(string ⇒ bool) ⇒ formula ⇒ bool›
  (‹_ ⊨ _› [50, 50] 50) where
  ‹I ⊨ ·x          ↔  I x›
| ‹I ⊨ ⊥           ↔  False›
| ‹I ⊨ p ⋀ q       ↔  I ⊨ p ⋀ I ⊨ q›
| ‹I ⊨ p ⟶ q       ↔  I ⊨ p ⟶ I ⊨ q›
```

x ⟶ *y*, for x := True, y := False

```
value ‹I(''x'' := True, ''y'' := False) ⊨ ·''x'' ⟶ ·''y''›
```

“False” :: “bool”

What is Isabelle/HOL like? – Soundness

“If we can derive p, then p is true for all interpretations I of the placeholders”:

```
corollary ‹{} ⊢ p ⟹ ∀I. I ⊨ p›
  using soundness by blast
```

“If we can prove p from A and everything in A is true, then p is true:

```
lemma soundness: ‹A ⊢ p ⟹ ∀q ∈ A. I ⊨ q ⟹ I ⊨ p›
  by (induct A p rule: Deduction.induct) simp_all
```

Isabelle proves it for us

Thesis Chapters

- Formalizing Henkin-Style Completeness of an Axiomatic System for Propositional Logic
- A Succinct Formalization of the Completeness of First-Order Logic
- Verifying a Sequent Calculus Prover for First-Order Logic with Functions in Isabelle/HOL
- Synthetic Completeness for a Terminating Seligman-Style Tableau System
- **Formalized Soundness and Completeness of Epistemic and Public Announcement Logic**
- **Aesop: White-Box Best-First Proof Search for Lean**
- **An Abstract Framework for Synthetic Completeness**

Epistemic and Public Announcement Logic

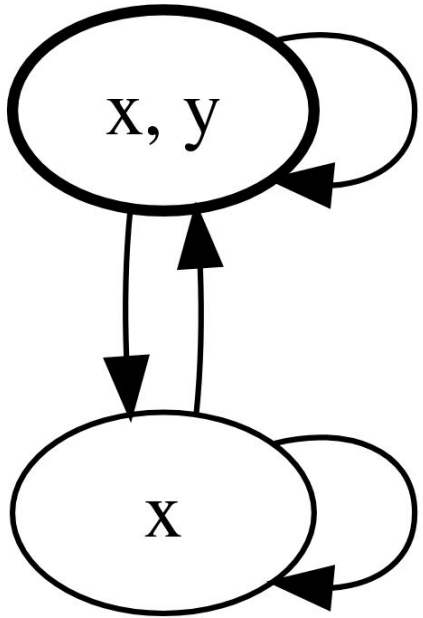
Epistemic Logic

Propositional logic: *one* interpretation of placeholders
Epistemic logic: a *graph* of interpretations:

- World for each interpretation
- Arrows between worlds
- An *actual* world, “our starting point”

Syntax is propositional logic +

- **K** *p* “we know *p*”
  “*p* is true everywhere an arrow points to”



Different Kinds of Knowledge

- Self-arrows: *true knowledge*

  If we know something, then it is true

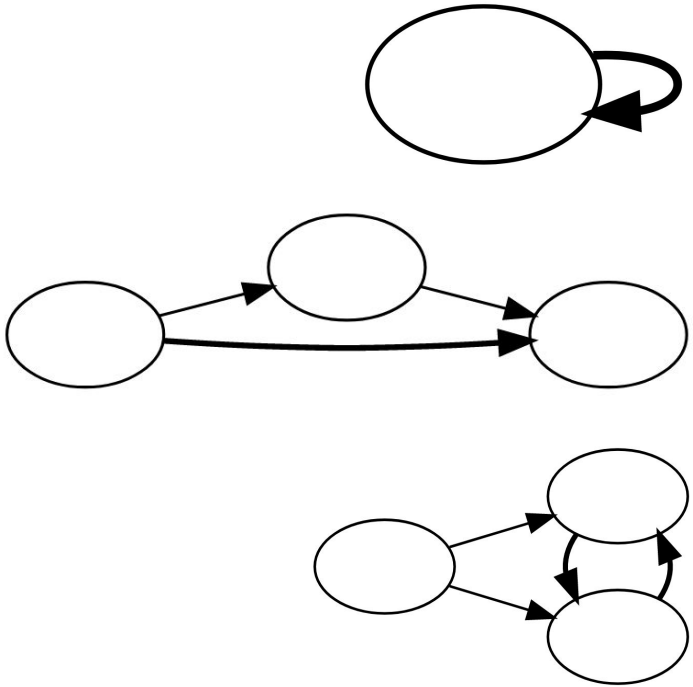
- Shortcut arrows: *positive introspection*

  We know what we know

- Euclidean arrows: *negative introspection*

  We know what we do not know

How to reason about all of them uniformly?



Different Kinds of Knowledge

- Self-arrows: *true knowledge*

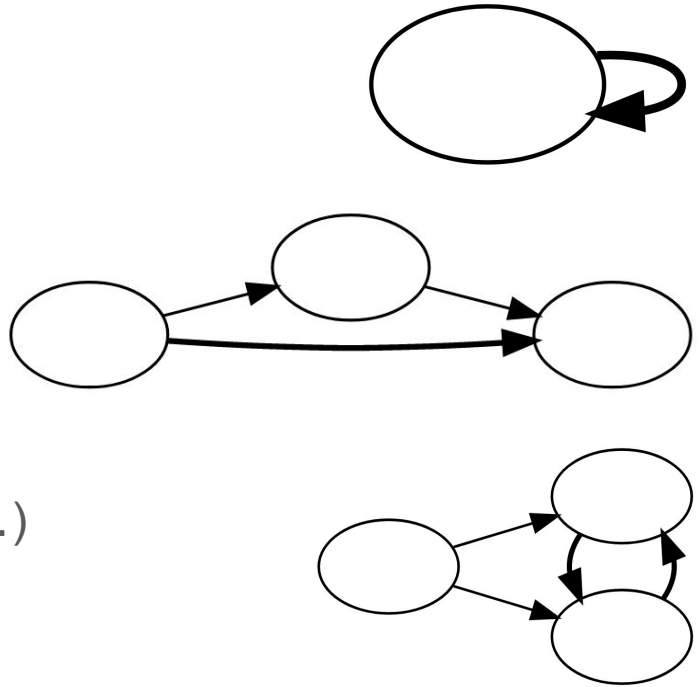
  $\mathbf{K}\, p \longrightarrow p$

- Shortcut arrows: *positive introspection*

  $\mathbf{K}\, p \longrightarrow \mathbf{K}\, (\mathbf{K}\, p)$

- Euclidean arrows: *negative introspection*

  $\neg\, \mathbf{K}\, p \longrightarrow \mathbf{K}\, (\neg\, \mathbf{K}\, i\, p)$ ($\neg\, p$ means $p \longrightarrow \bot$)



How to reason about all of them uniformly?

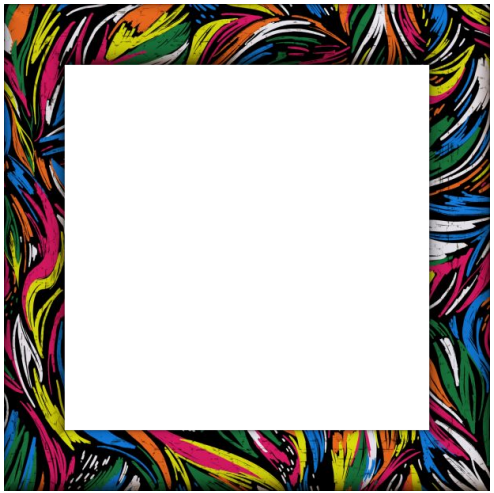
A Parameterized Proof System

A ⊢ p means “we can prove p from axioms A”

```
inductive AK :: ‹('i fm ⇒ bool) ⇒ 'i fm ⇒ bool› (‹_ ⊢ _› [50, 50] 50)
  for A :: ‹'i fm ⇒ bool› where
  A1: ‹tautology p ⟹ A ⊢ p›
| A2: ‹A ⊢ K i p ⋀ K i (p ⟶ q) ⟶ K i q›
| Ax: ‹A p ⟹ A ⊢ p›
| R1: ‹A ⊢ p ⟹ A ⊢ p ⟶ q ⟹ A ⊢ q›
| R2: ‹A ⊢ p ⟹ A ⊢ K i p›
```

With A we can include different kinds of knowledge

We can work *abstractly* for as long as possible



Soundness

Some axioms are only valid on some types of graphs:

- **K** $p \longrightarrow p$ requires self-arrows, etc.

When is our deduction method sound for graphs of type `T`?

When the axioms are all valid on graphs of type `T`:

```
theorem soundness:
  assumes ‹∧M w q. A q ⟹ T M ⟹ w ∈ 𝒲 M ⟹ M, w ⊨ q›
  shows ‹A ⊢ p ⟹ T M ⟹ w ∈ 𝒲 M ⟹ M, w ⊨ p›
```

Instead of one theorem per axiom, we have a family of theorems

Completeness-via-Canonicity

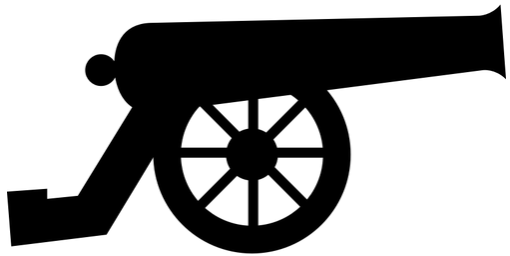
Given a set of axioms, there is a *canonical* way to build a graph

- Axiom **K** *p* ⟶ **K** (**K** *p*) gives the canonical graph *shortcut arrows*

Completeness-via-canonicity arguments are well known:

- Prove that your axioms give the canonical graph property *T*
- \+ Get completeness for your logic on graphs of type *T*

```
corollary completeness:
  assumes ‹T; {} ⊩ p› and ‹T (canonical A)›
  shows ‹A ⊢ p›
```



Public Announcement Logic

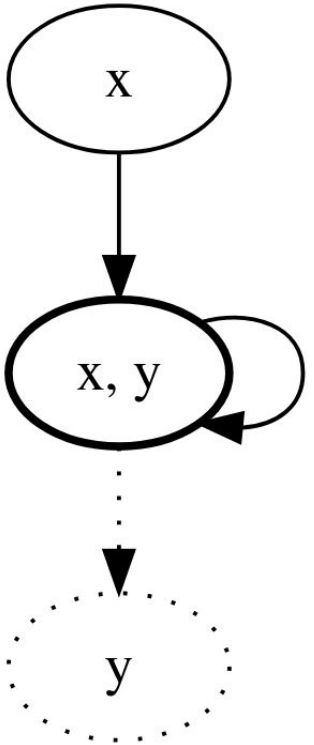
Someone says *r* and we believe them
This *changes* the graph of our uncertainty

Syntax is epistemic logic +

- **[*r*!] *p*** “after the announcement of *r*, *p* is true”
  “considering only worlds satisfying *r*, *p* is true”

Examples:

- **[*x*!] K** *x* “we are told the sky is blue, so now we know”
- **[⊥!] K** *p* “we are told a lie, so now we know everything”



announcing *x*

Soundness and Completeness

Soundness requires well behaved announcements:

- Axioms *A* are valid on graphs of type *T*
- Announcements preserve property *T*

\+ Soundness of PAL+*A* on graphs of type *T*

Public announcements are shorthand for pure epistemic logic:

- EL completeness for axioms *A* on graphs of type *T*

\+ PAL completeness for axioms *A* on graphs of type *T*

Aesop: Proof Search for Lean 4

Collaboration with Jannis Limperg

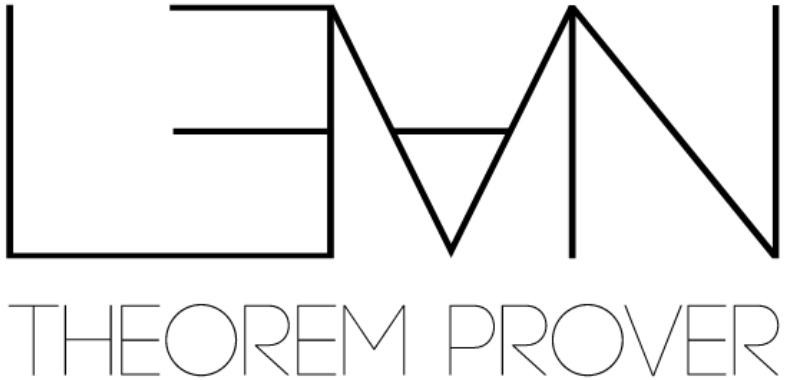
Lean 4

A proof assistant based on dependent type theory

Younger than Isabelle/HOL

How do we get the same automation?

+ $\vdash p \longrightarrow q \longrightarrow p \wedge q$
+ $p \vdash q \longrightarrow p \wedge q$
+ $p, q \vdash p \wedge q$
+ $p, q \vdash q$
+ $p, q \vdash p$



Search Tree

We have a goal we want to prove

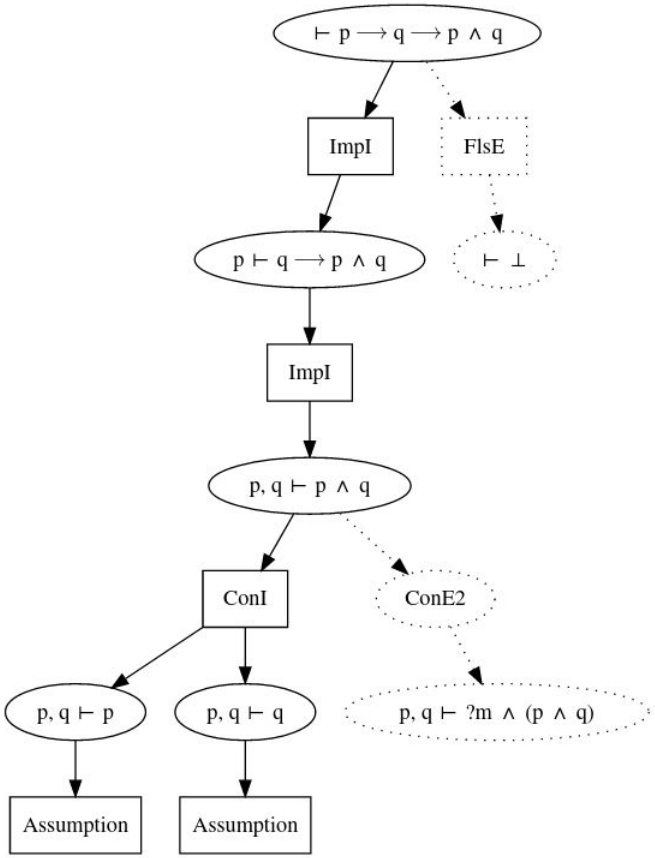
+ $\vdash p \longrightarrow q \longrightarrow p \wedge q$

We have rules that transform goals into subgoals

- $A \vdash p$
- $A \vdash q$
+ $A \vdash p \wedge q$

Some rules have no subgoals

Build a tree!



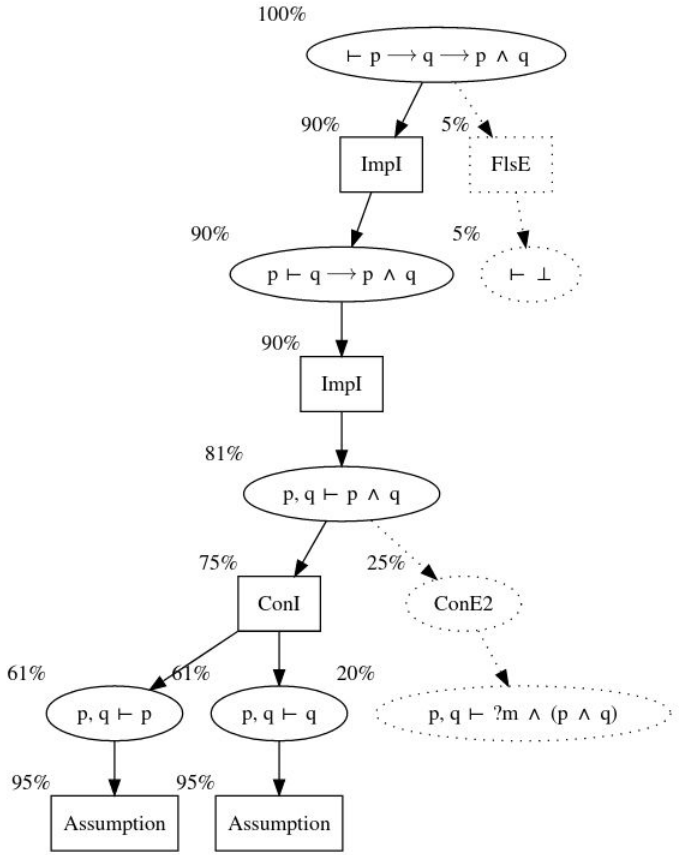
How to Expand the Tree?

Many paths lead nowhere

- Give each rule a *success* probability
- Start at goal priority 100%
- Multiply to get subgoal priority

Always work on the goal with *highest priority*

(values for illustration only)



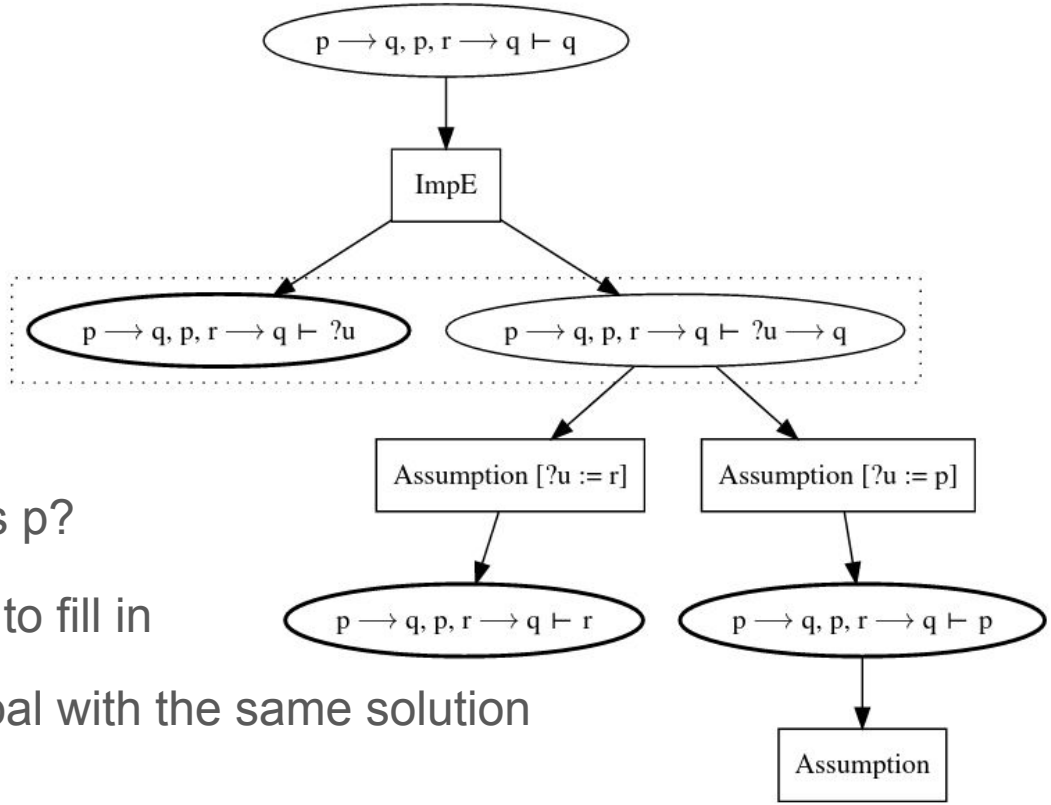
Metavariables

Recall implication elimination:

- $A \vdash p \longrightarrow q$
- $A \vdash p$
- \+ $A \vdash q$

When we use it, how do we guess p?

- Leave a ?hole for a later rule to fill in
- Fill a **copy** of the other subgoal with the same solution



Applications

Used by the *mathlib* project in porting from Lean 3 to Lean 4.

They have extended it to special situations:

- `aesop` *trivial* proofs
- `aesop_cat` category theory
- `aesop_graph` graph theory
- `Sym2` algebra
- `continuity` topology

Abstract Synthetic Completeness Framework

Synthetic Completeness?

Proving soundness is usually easy: (only good things)

- Each axiom and rule makes *sense*

Proving completeness is trickier: (every good thing)

- We have *enough* rules and axioms

Two main approaches:

- Analytic: build *countermodels* from failing proof attempts
- Synthetic: build models for *consistent* sets

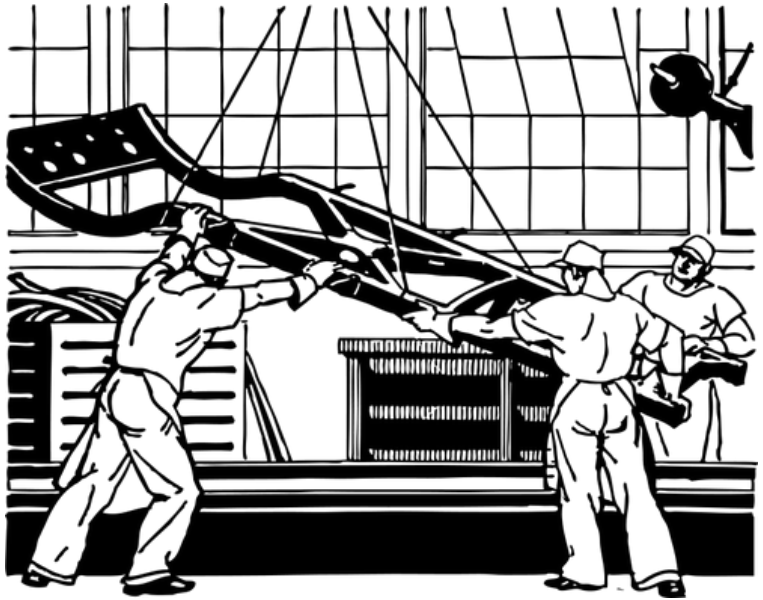
Formalized Synthetic Completeness

The synthetic method works for many logics:

- modal and hybrid logic
- first-order logic
- second-order logic

By formalizing it we get:

- Computer assistance in applying it
- Interactive documentation
- Assurance of its correctness



Maximal Consistent Sets

Assumptions that prove falsity, $A \vdash \bot$, are *contradictory, "in conflict"*

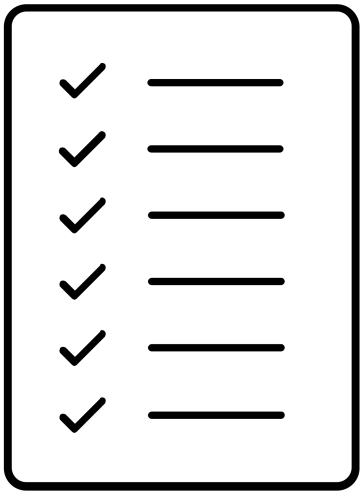
Say we have a *consistent, "harmonious"* set of formulas S_n

(*Sketch*) We can go through every possible formula:

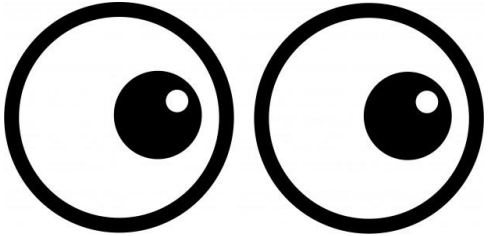
- If adding it preserves consistency, add it
- If adding it creates a contradiction, leave it out

In the end every formula or its negation is in there

Isabelle/HOL only superficially different from set theory



Witnesses



We want a few more formulas, though:

- For “there exists someone mortal”, we want, e.g., “Socrates is mortal”

Using “Socrates” again and again could lead to a contradiction

- In set theory, they add extra words for “Aristotle”, “Plato”, etc.
- In Isabelle/HOL, this changes the *type* of the language

  I assume enough witnesses from the start

Slightly different ceremony but much the same

Requirements

All of this can be done *abstractly*:

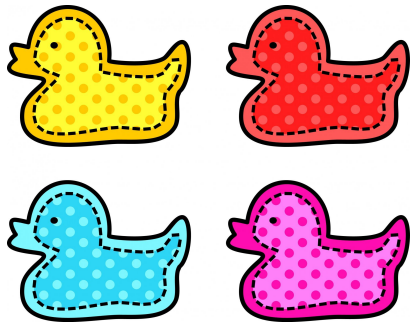
- Subsets of consistent sets are consistent
- Inconsistent sets have finite inconsistencies
- There are infinitely many formulas

\+ Maximal consistent sets

- Formulas and their witnesses are finite
- Fresh witnesses preserve consistency
- We have enough witnesses

\+ *Witnessed* maximal consistent sets

Where is my Completeness?



The semantics has a certain *shape*:

- The *truth* of a formula depends on the *truth* of its parts **in a certain way**

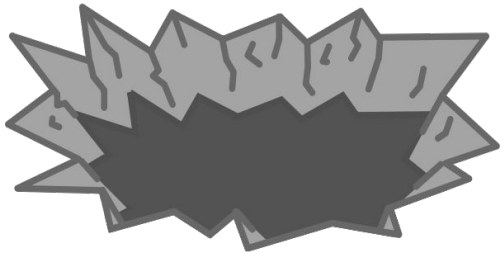
If we can prove that MCSs have the same *shape*:

- The *membership* of a formula depends on the *membership* of its parts **in the same way**

Then we can relate *truth* and *membership*

What is the *shape* we are looking for?

From Semantics to *Semics*

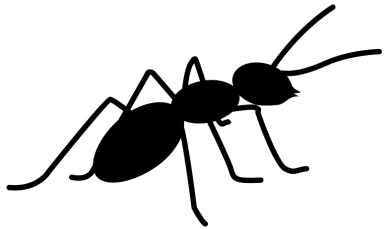


Recall when a propositional formula is true:

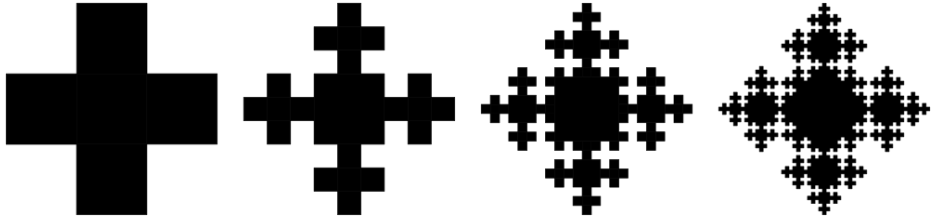
- a prop. symbol: *x, y* — when the interpretation says so
- falsity: ⊥ — never
- a conjunction: $p \wedge q$ — *p* is true and *q* is true

This is *self-referential*. We can leave a hole [ ] instead:

- a prop. symbol: *x, y* — when the interpretation says so
- falsity: ⊥ — never
- a conjunction: $p \wedge q$ — *p* [ ] and *q* [ ]



Filling Holes



Now we can use this *shape*:

p ∧ *q* **is in MCS** if and only if *p* **is in MCS** and *q* **is in MCS**

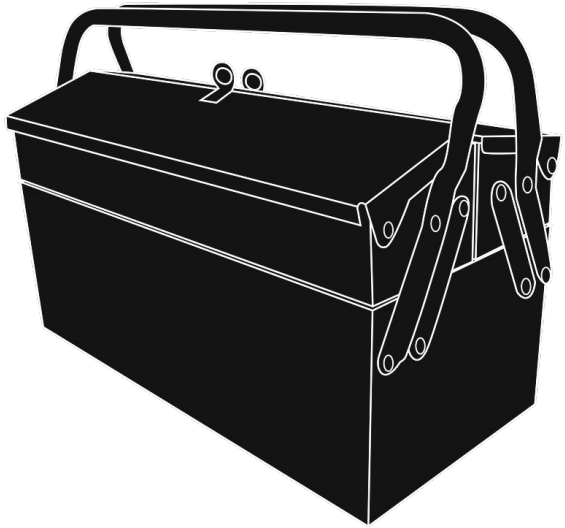
Old functional programming trick:

- Take a recursive definition
- Abstract away the recursive occurrences

❏ Filling in the [original semantics] gives the original semantics

❏ The fixpoint [semics] is the original semantics

Resulting Framework



We can build MCSs for at least:

- propositional tableau and sequent calculus
- first-order and hybrid logic natural deduction
- modal logic system K

The *semics* trick is useful when the logic has a *Cut* rule

Future work:

- A *semics* equation for *downwards* saturated sets (if ~~and only if~~)
- Formalize compactness theorem for uncountable languages?

Conclusion

Conclusion

Different logics can express different things

Proof assistants can help us with our reasoning

- Encourage *proof engineering*
- Automate *trivial* proofs
- Express general *frameworks*

There are many cool logics out there

Go formalize their completeness