

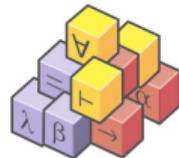
Getting Started with Isabelle/HOL

Simon Foster **Jim Woodcock**
University of York

16th August 2022

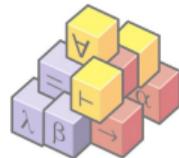
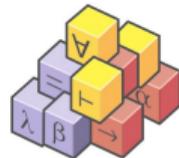
Overview

- 1 What is a Formal Proof?
- 2 Automated versus Interactive Theorem Provers
- 3 The Isabelle/HOL Proof Assistant
- 4 Theory Documents
- 5 Isabelle/jEdit and Proof IDE



Overview

- 1 What is a Formal Proof?
- 2 Automated versus Interactive Theorem Provers
- 3 The Isabelle/HOL Proof Assistant
- 4 Theory Documents
- 5 Isabelle/jEdit and Proof IDE



Formal Proof

- Proof: demonstrate truth of a statement. Argument and evidence.
- Formal proof: turning logical conjectures into theorems.
- Conjecture: statement that a formula is true, without yet having a proof.
- Assume $x > 0$ then $x + 2 > 0$.
- Assume $n > 2$. Then
- There are no three positive integers a, b, c with $a^2 + b^2 = c^2$.
- Proof shows how to formally derive conclusions from assumptions.
- By application of axioms, existing theorems and deduction rules.

Formal Proof

- **Proof:** demonstrate truth of a statement. Argument and evidence.
- Formal proof: turning logical conjectures into theorems.
- Conjecture: statement that a formula is true, without yet having a proof.
- Assume $x > 0$ then $x * 2 > 0$.
- Fermat's Last Theorem:
Assume $n > 2$. Then:
There are no three positive integers a, b, c with $a^n + b^n = c^n$.
- Proof shows how to formally derive conclusions from assumptions.
- By application of axioms, existing theorems and deduction rules.

Formal Proof

- **Proof**: demonstrate truth of a statement. Argument and evidence.
- **Formal proof**: turning logical **conjectures** into **theorems**.
- **Conjecture**: statement that a formula is true, without yet having a proof.
- Assume $x > 0$ then $x * 2 > 0$.
- Fermat's Last Theorem:
Assume $n > 2$. Then:
There are no three positive integers a, b, c with $a^n + b^n = c^n$.
- Proof shows how to formally derive **conclusions** from **assumptions**.
- By application of axioms, existing theorems and deduction rules.

Formal Proof

- **Proof**: demonstrate truth of a statement. Argument and evidence.
- **Formal proof**: turning logical **conjectures** into **theorems**.
- **Conjecture**: statement that a formula is true, without yet having a proof.
- Assume $x > 0$ then $x * 2 > 0$.
- Fermat's Last Theorem:
Assume $n > 2$. Then:
There are no three positive integers a, b, c with $a^n + b^n = c^n$.
- Proof shows how to formally derive **conclusions** from **assumptions**.
- By application of axioms, existing theorems and deduction rules.

Formal Proof

- **Proof**: demonstrate truth of a statement. Argument and evidence.
- **Formal proof**: turning logical **conjectures** into **theorems**.
- **Conjecture**: statement that a formula is true, without yet having a proof.
- Assume $x > 0$ then $x * 2 > 0$.

- Fermat's Last Theorem:

Assume $n > 2$. Then:

There are no three positive integers a, b, c with $a^n + b^n = c^n$.

- Proof shows how to formally derive **conclusions** from **assumptions**.
- By application of axioms, existing theorems and deduction rules.

Formal Proof

- **Proof**: demonstrate truth of a statement. Argument and evidence.
- **Formal proof**: turning logical **conjectures** into **theorems**.
- **Conjecture**: statement that a formula is true, without yet having a proof.
- Assume $x > 0$ then $x * 2 > 0$.
- **Fermat's Last Theorem**:

Assume $n > 2$. Then:

There are no three positive integers a, b, c with $a^n + b^n = c^n$.

- Proof shows how to formally derive **conclusions** from **assumptions**.
- By application of axioms, existing theorems and deduction rules.

Formal Proof

- **Proof**: demonstrate truth of a statement. Argument and evidence.
- **Formal proof**: turning logical **conjectures** into **theorems**.
- **Conjecture**: statement that a formula is true, without yet having a proof.
- Assume $x > 0$ then $x * 2 > 0$.
- **Fermat's Last Theorem**:

Assume $n > 2$. Then:

There are no three positive integers a, b, c with $a^n + b^n = c^n$.

- Proof shows how to formally derive **conclusions** from **assumptions**.
- By application of axioms, existing theorems and deduction rules.

Formal Proof

- **Proof**: demonstrate truth of a statement. Argument and evidence.
- **Formal proof**: turning logical **conjectures** into **theorems**.
- **Conjecture**: statement that a formula is true, without yet having a proof.
- Assume $x > 0$ then $x * 2 > 0$.
- **Fermat's Last Theorem**:

Assume $n > 2$. Then:

There are no three positive integers a, b, c with $a^n + b^n = c^n$.

- Proof shows how to formally derive **conclusions** from **assumptions**.
- By application of axioms, existing theorems and deduction rules.

Formal Proof

- **Proof**: demonstrate truth of a statement. Argument and evidence.
- **Formal proof**: turning logical **conjectures** into **theorems**.
- **Conjecture**: statement that a formula is true, without yet having a proof.
- Assume $x > 0$ then $x * 2 > 0$.
- **Fermat's Last Theorem**:
Assume $n > 2$. Then:
There are no three positive integers a, b, c with $a^n + b^n = c^n$.
- Proof shows how to formally derive **conclusions** from **assumptions**.
- By application of axioms, existing theorems and deduction rules.

Formal Proof

- **Proof**: demonstrate truth of a statement. Argument and evidence.
- **Formal proof**: turning logical **conjectures** into **theorems**.
- **Conjecture**: statement that a formula is true, without yet having a proof.
- Assume $x > 0$ then $x * 2 > 0$.
- **Fermat's Last Theorem**:

Assume $n > 2$. Then:

There are no three positive integers a, b, c with $a^n + b^n = c^n$.

- Proof shows how to formally derive **conclusions** from **assumptions**.
- By application of **axioms**, existing **theorems** and **deduction rules**.

Proof as Computation

Analogy between proof and a function mapping inputs to outputs.

Theorem provers and proof assistants help us in this process.

Two main classes: automated and interactive theorem provers.

Proof as Computation

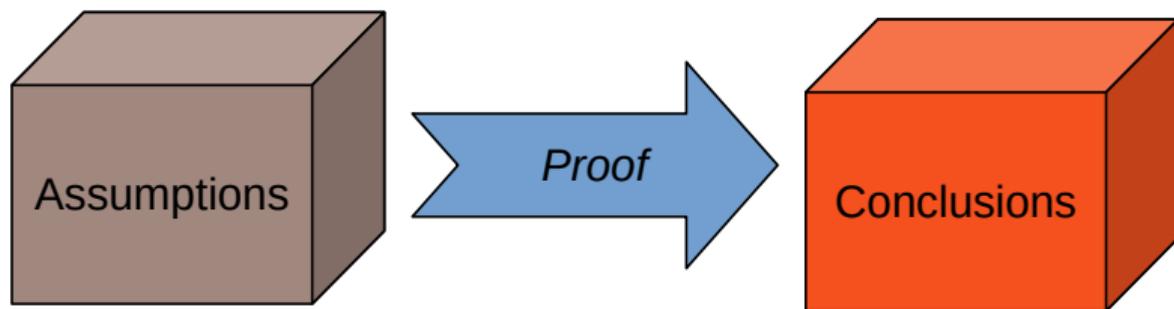
- Analogy between proof and a function mapping inputs to outputs:

• Theorem provers and proof assistants help us in this process.

• They may be seen as automated and interactive theorem provers.

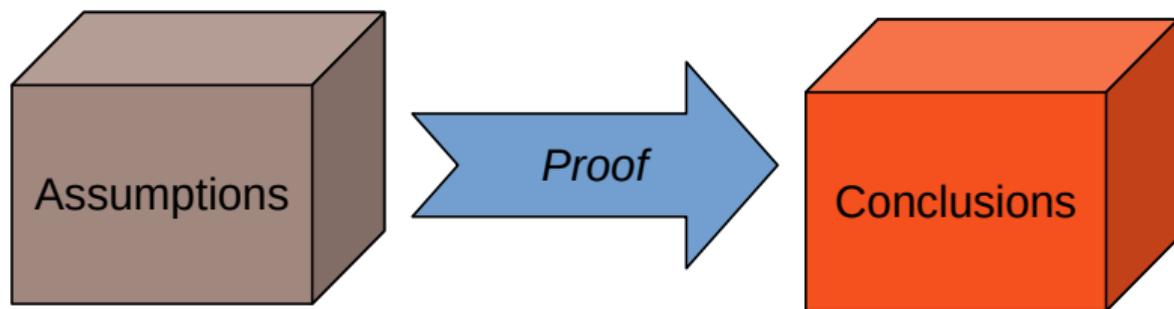
Proof as Computation

- Analogy between proof and a function mapping inputs to outputs:



Proof as Computation

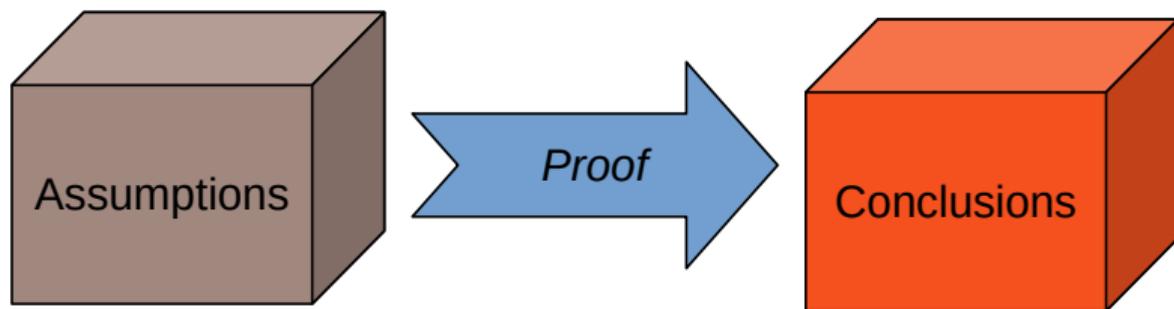
- Analogy between proof and a function mapping inputs to outputs:



- **Theorem provers** and **proof assistants** help us in this process.
- Two main classes: **automated** and **interactive** theorem provers.

Proof as Computation

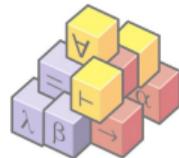
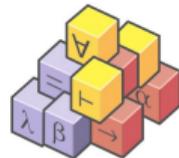
- Analogy between proof and a function mapping inputs to outputs:



- **Theorem provers** and **proof assistants** help us in this process.
- Two main classes: **automated** and **interactive** theorem provers.

Overview

- 1 What is a Formal Proof?
- 2 Automated versus Interactive Theorem Provers**
- 3 The Isabelle/HOL Proof Assistant
- 4 Theory Documents
- 5 Isabelle/jEdit and Proof IDE



Automated Theorem Provers

Automated theorem proving is an undecidable problem.

Are the push-button techniques like model checkers.

SMT solvers (like Microsoft's Z3) prove arithmetic theorems etc.

Usually limited to first-order logic.

variables range over individuals not over predicates.

In general, they do not handle induction, which requires higher order logic.

We need Interactive Theorem Provers (ITPs)

Automated Theorem Provers

- Automated theorem proving is an **undecidable problem**.

• All the push-button techniques like model checkers

• SAT solvers like Minisat, CDCL, naive enumerative solvers, etc.

• Quantifier elimination

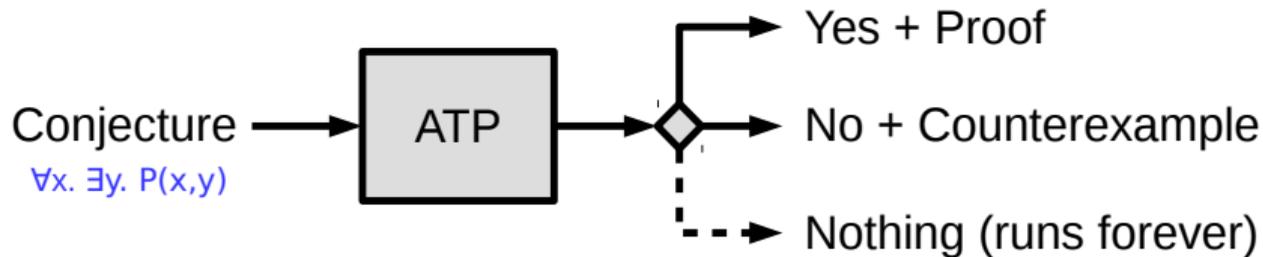
• Various things that are not covered

• In general, you have to induction, which requires higher order logic.

• Hybrid Interactive Theorem Provers (ITP)

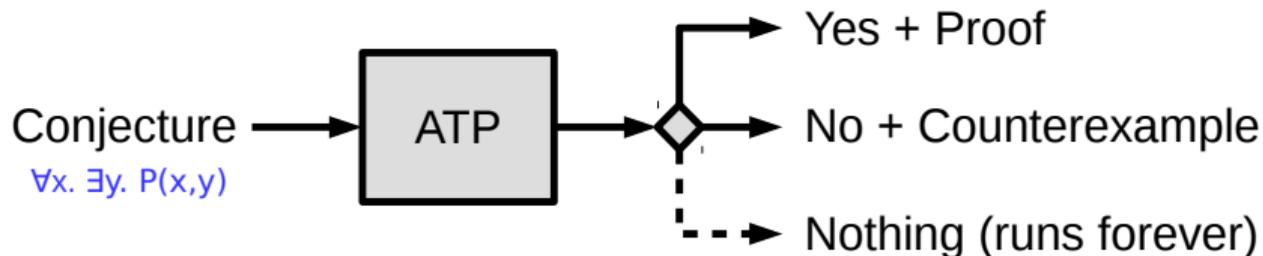
Automated Theorem Provers

- Automated theorem proving is an **undecidable problem**.



Automated Theorem Provers

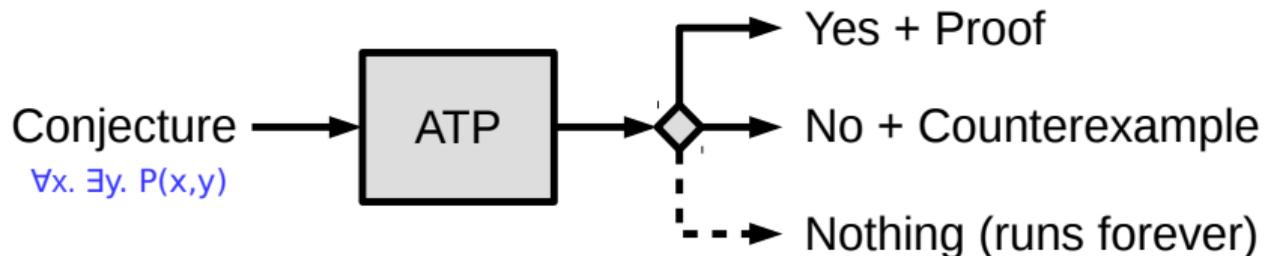
- Automated theorem proving is an **undecidable problem**.



- Attractive **push-button** technology. Like **model checkers**.
 - **SMT solvers** (like Microsoft's Z3) prove arithmetic theorems etc.
 - Usually limited to **first-order logic**.
 - Variables range over individuals not other predicates.
 - In general, cannot handle **induction**, which requires **higher order logic**.
 - We need **Interactive Theorem Provers (ITPs)**.

Automated Theorem Provers

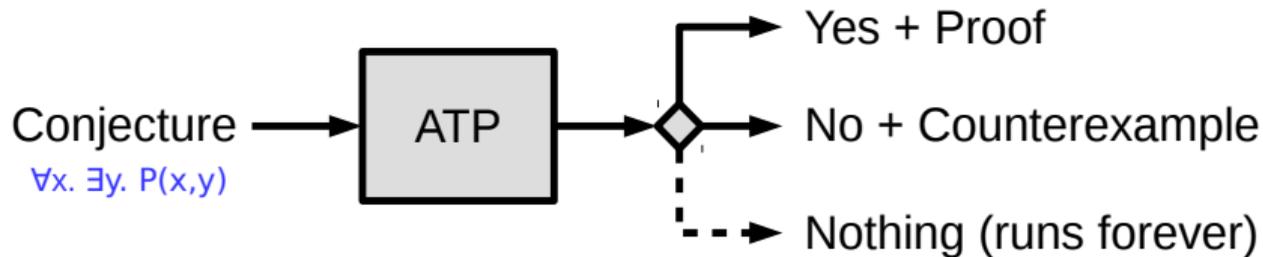
- Automated theorem proving is an **undecidable problem**.



- Attractive **push-button** technology. Like **model checkers**.
- **SMT solvers** (like Microsoft's Z3) prove arithmetic theorems etc.
 - Usually limited to **first-order logic**.
 - Variables range over individuals not other predicates.
 - In general, cannot handle **induction**, which requires **higher order logic**.
 - We need **Interactive Theorem Provers (ITPs)**.

Automated Theorem Provers

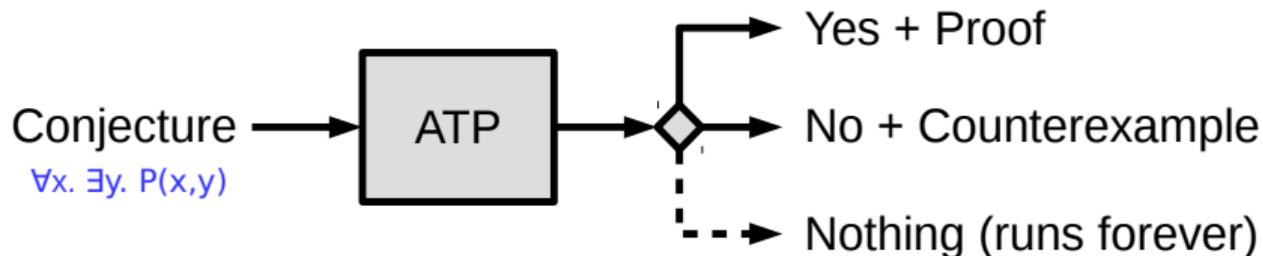
- Automated theorem proving is an **undecidable problem**.



- Attractive **push-button** technology. Like **model checkers**.
- **SMT solvers** (like Microsoft's Z3) prove arithmetic theorems etc.
- Usually limited to **first-order logic**.
- Variables range over individuals not other predicates.
- In general, cannot handle **induction**, which requires **higher order logic**.
- We need **Interactive Theorem Provers (ITPs)**.

Automated Theorem Provers

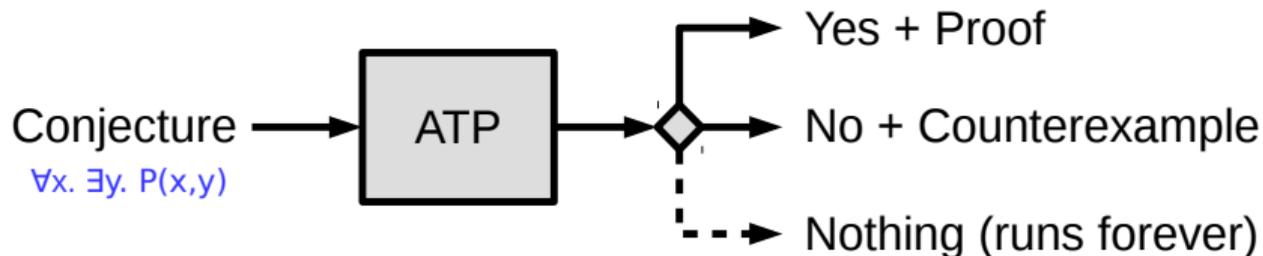
- Automated theorem proving is an **undecidable problem**.



- Attractive **push-button** technology. Like **model checkers**.
- **SMT solvers** (like Microsoft's Z3) prove arithmetic theorems etc.
- Usually limited to **first-order logic**.
- Variables range over individuals not other predicates.
- In general, cannot handle **induction**, which requires **higher order logic**.
- We need **Interactive Theorem Provers (ITPs)**.

Automated Theorem Provers

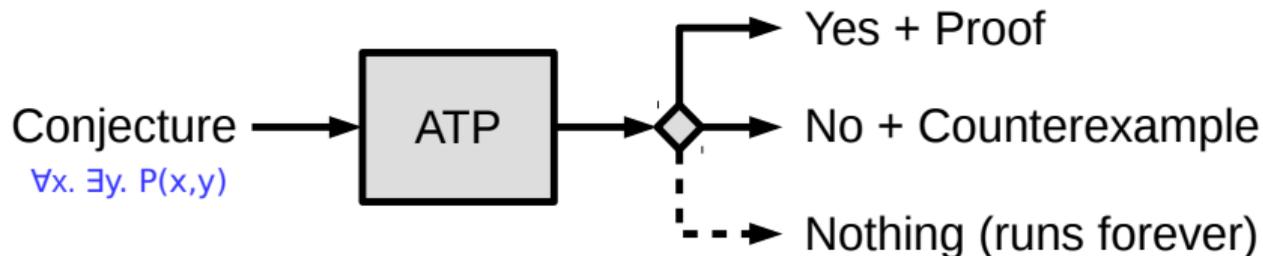
- Automated theorem proving is an **undecidable problem**.



- Attractive **push-button** technology. Like **model checkers**.
- **SMT solvers** (like Microsoft's Z3) prove arithmetic theorems etc.
- Usually limited to **first-order logic**.
- Variables range over individuals not other predicates.
- In general, cannot handle **induction**, which requires **higher order logic**.
- We need **Interactive Theorem Provers (ITPs)**.

Automated Theorem Provers

- Automated theorem proving is an **undecidable problem**.



- Attractive **push-button** technology. Like **model checkers**.
- SMT solvers** (like Microsoft's Z3) prove arithmetic theorems etc.
- Usually limited to **first-order logic**.
- Variables range over individuals not other predicates.
- In general, cannot handle **induction**, which requires **higher order logic**.
- We need **Interactive Theorem Provers** (ITPs).

Interactive Theorem Provers

- A proof is a **sequence of logical tactics** on a proof state
 - **Formal assumptions and outstanding conjectures** ("goals")
 - **Formal proof objects** (proof terms)
- **Proof tactics and decision rules** subdivide and eliminate proof goals
 - **divide and conquer** approach to proof
 - **Proof goals where winning condition is QED** (positive subgoals)

Interactive Theorem Provers

- A **proof** is a script or program that acts on a **proof state**.
- Proof state: assumptions and outstanding conjectures (“subgoals”).

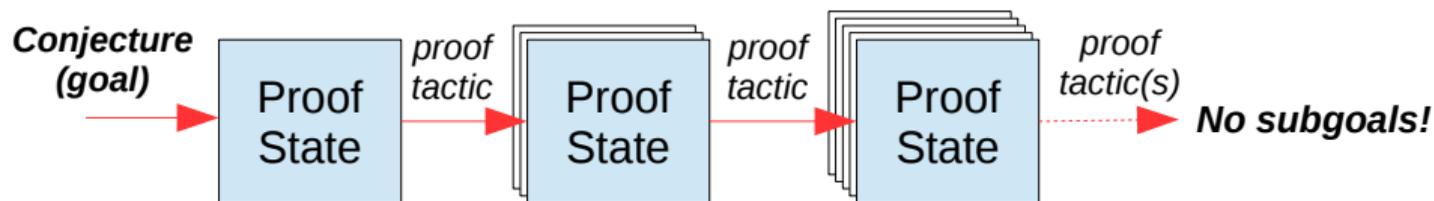
- Proof tactics and decision rules introduce and eliminate proof goals, lemmas and lemmas (referring to lemmas).
- Proof goals with wrong condition is QED (no more subgoals).

Interactive Theorem Provers

- A **proof** is a script or program that acts on a **proof state**.
- **Proof state**: assumptions and outstanding conjectures (“**subgoals**”).

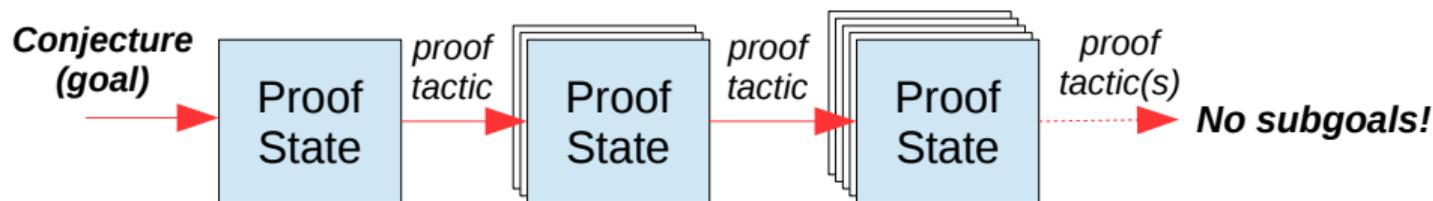
Interactive Theorem Provers

- A **proof** is a script or program that acts on a **proof state**.
- **Proof state**: assumptions and outstanding conjectures (“**subgoals**”).



Interactive Theorem Provers

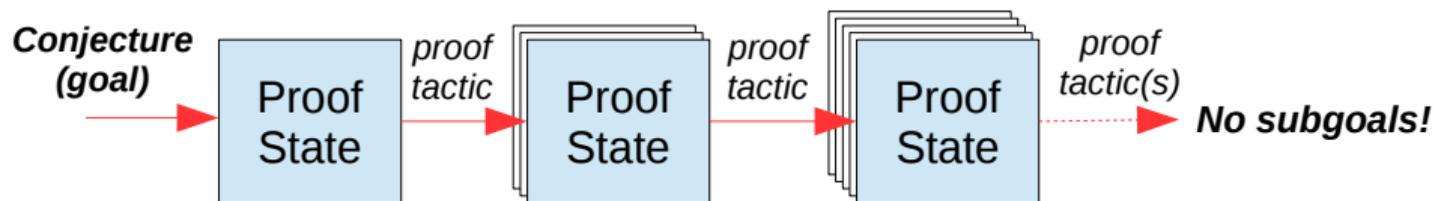
- A **proof** is a script or program that acts on a **proof state**.
- **Proof state**: assumptions and outstanding conjectures (“**subgoals**”).



- **Proof tactics** and deduction rules subdivide and eliminate **proof goals**.
- “Divide and conquer” approach to proof.
- Proof: game where winning condition is **QED** (no more subgoals).

Interactive Theorem Provers

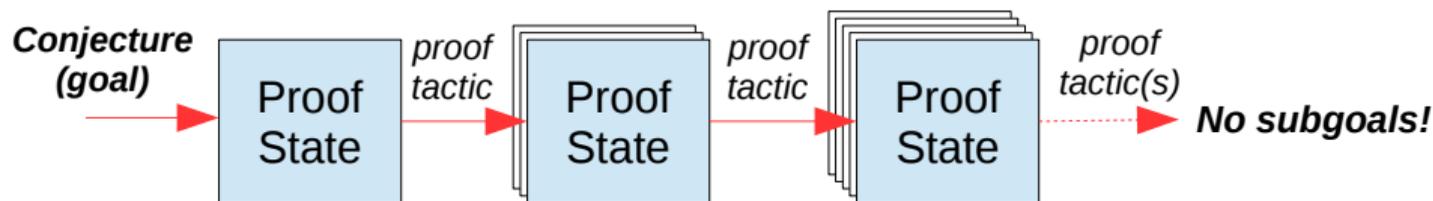
- A **proof** is a script or program that acts on a **proof state**.
- **Proof state**: assumptions and outstanding conjectures (“**subgoals**”).



- **Proof tactics** and deduction rules subdivide and eliminate **proof goals**.
- “Divide and conquer” approach to proof.
- Proof: game where winning condition is **QED** (no more subgoals).

Interactive Theorem Provers

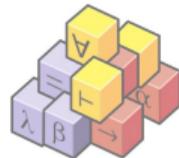
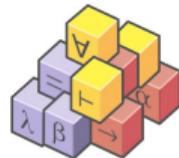
- A **proof** is a script or program that acts on a **proof state**.
- **Proof state**: assumptions and outstanding conjectures (“**subgoals**”).



- **Proof tactics** and deduction rules subdivide and eliminate **proof goals**.
- “Divide and conquer” approach to proof.
- **Proof**: game where winning condition is **QED** (no more subgoals).

Overview

- 1 What is a Formal Proof?
- 2 Automated versus Interactive Theorem Provers
- 3 The Isabelle/HOL Proof Assistant**
- 4 Theory Documents
- 5 Isabelle/jEdit and Proof IDE



Isabelle/HOL

- Interactive theorem prover for **Higher Order Logic** (started in 1986).

- `isabelle.in.tum.de`

- Developed by Larry Paulson, Tobias Nipkow, Norisanth Weirath

- HOL is a powerful functional specification language.

- Like ML, but data structures, recursive functions, type classes, etc.

- Readable proofs in Natural and LaTeX style (HOL4)

- Logic of the Theory of Functional Mathematics.

- <http://www.sri.com/~jha/HOL/>

- <http://www.in.tum.de/~hol/>

- <http://www.in.tum.de/~hol/doc/>

- <http://www.in.tum.de/~hol/doc/ML-Functional-Logic.pdf>

- <http://www.in.tum.de/~hol/doc/assured-software-development.pdf>

Isabelle/HOL

- Interactive theorem prover for **Higher Order Logic** (started in **1986**).
- `isabelle.in.tum.de`

Isabelle/HOL

- Interactive theorem prover for **Higher Order Logic** (started in 1986).
- isabelle.in.tum.de 

Isabelle/HOL

- Interactive theorem prover for **Higher Order Logic** (started in **1986**).
- isabelle.in.tum.de 
- Developed by Larry Paulson, Tobias Nipkow, Markarius Wenzel.
- HOL is a powerful functional specification language.
- Like Haskell: data structures, recursive functions, type classes, etc.
- **Readable proofs** in “natural deduction” style (Isar).
- Large online library of formalised mathematics.
- Archive of Formal Proofs www.isa-afp.org/.
- “Quantum and Classical Registers”, “Category Theory for ZFC”, ...
- Support for verified **code generation**.
- Implemented in the **ML** functional language and **Scala**.
- Program verification and **assured software development**.

Isabelle/HOL

- Interactive theorem prover for **Higher Order Logic** (started in **1986**).
- **isabelle.in.tum.de** 
- Developed by Larry Paulson, Tobias Nipkow, Markarius Wenzel.
- **HOL** is a powerful functional specification language.
- Like **Haskell**: data structures, recursive functions, type classes, etc.
- **Readable proofs** in “natural deduction” style (Isar).
- Large online library of formalised mathematics.
- Archive of Formal Proofs www.isa-afp.org/.
- “Quantum and Classical Registers”, “Category Theory for ZFC”, ...
- Support for verified **code generation**.
- Implemented in the **ML** functional language and **Scala**.
- Program verification and **assured software development**.

Isabelle/HOL

- Interactive theorem prover for **Higher Order Logic** (started in 1986).
- isabelle.in.tum.de 
- Developed by Larry Paulson, Tobias Nipkow, Markarius Wenzel.
- **HOL** is a powerful functional specification language.
- Like **Haskell**: data structures, recursive functions, type classes, etc.
- **Readable proofs** in “natural deduction” style (Isar).
- Large online library of formalised mathematics.
- Archive of Formal Proofs www.isa-afp.org/.
- “Quantum and Classical Registers”, “Category Theory for ZFC”, ...
- Support for verified **code generation**.
- Implemented in the **ML** functional language and **Scala**.
- Program verification and **assured software development**.

Isabelle/HOL

- Interactive theorem prover for **Higher Order Logic** (started in **1986**).
- isabelle.in.tum.de 
- Developed by Larry Paulson, Tobias Nipkow, Markarius Wenzel.
- **HOL** is a powerful functional specification language.
- Like **Haskell**: data structures, recursive functions, type classes, etc.
- **Readable proofs** in “natural deduction” style (**Isar**).
- Large online library of formalised mathematics.
- Archive of Formal Proofs www.isa-afp.org/.
- “Quantum and Classical Registers”, “Category Theory for ZFC”, ...
- Support for verified **code generation**.
- Implemented in the **ML** functional language and **Scala**.
- Program verification and **assured software development**.

Isabelle/HOL

- Interactive theorem prover for **Higher Order Logic** (started in **1986**).
- isabelle.in.tum.de 
- Developed by Larry Paulson, Tobias Nipkow, Markarius Wenzel.
- **HOL** is a powerful functional specification language.
- Like **Haskell**: data structures, recursive functions, type classes, etc.
- **Readable proofs** in “natural deduction” style (**Isar**).
- Large online library of **formalised mathematics**.
- Archive of Formal Proofs www.isa-afp.org/.
- “Quantum and Classical Registers”, “Category Theory for ZFC”, ...
- Support for verified **code generation**.
- Implemented in the **ML** functional language and **Scala**.
- Program verification and **assured software development**.

Isabelle/HOL

- Interactive theorem prover for **Higher Order Logic** (started in **1986**).
- isabelle.in.tum.de 
- Developed by Larry Paulson, Tobias Nipkow, Markarius Wenzel.
- **HOL** is a powerful functional specification language.
- Like **Haskell**: data structures, recursive functions, type classes, etc.
- **Readable proofs** in “natural deduction” style (**Isar**).
- Large online library of **formalised mathematics**.
- Archive of Formal Proofs www.isa-afp.org/.
- “Quantum and Classical Registers”, “Category Theory for ZFC”, ...
- Support for verified **code generation**.
- Implemented in the **ML** functional language and **Scala**.
- Program verification and **assured software development**.

Isabelle/HOL

- Interactive theorem prover for **Higher Order Logic** (started in **1986**).
- isabelle.in.tum.de 
- Developed by Larry Paulson, Tobias Nipkow, Markarius Wenzel.
- **HOL** is a powerful functional specification language.
- Like **Haskell**: data structures, recursive functions, type classes, etc.
- **Readable proofs** in “natural deduction” style (**Isar**).
- Large online library of **formalised mathematics**.
- Archive of Formal Proofs www.isa-afp.org/.
- “Quantum and Classical Registers”, “Category Theory for ZFC”, ...
- Support for verified **code generation**.
- Implemented in the **ML** functional language and **Scala**.
- Program verification and **assured software development**.

Isabelle/HOL

- Interactive theorem prover for **Higher Order Logic** (started in **1986**).
- isabelle.in.tum.de 
- Developed by Larry Paulson, Tobias Nipkow, Markarius Wenzel.
- **HOL** is a powerful functional specification language.
- Like **Haskell**: data structures, recursive functions, type classes, etc.
- **Readable proofs** in “natural deduction” style (**Isar**).
- Large online library of **formalised mathematics**.
- Archive of Formal Proofs www.isa-afp.org/.
- “Quantum and Classical Registers”, “Category Theory for ZFC”, ...
- Support for verified **code generation**.
- Implemented in the **ML** functional language and **Scala**.
- Program verification and **assured software development**.

Isabelle/HOL

- Interactive theorem prover for **Higher Order Logic** (started in **1986**).
- isabelle.in.tum.de 
- Developed by Larry Paulson, Tobias Nipkow, Markarius Wenzel.
- **HOL** is a powerful functional specification language.
- Like **Haskell**: data structures, recursive functions, type classes, etc.
- **Readable proofs** in “natural deduction” style (**Isar**).
- Large online library of **formalised mathematics**.
- Archive of Formal Proofs www.isa-afp.org/.
- “Quantum and Classical Registers”, “Category Theory for ZFC”, ...
- Support for verified **code generation**.
- Implemented in the **ML** functional language and **Scala**.
- Program verification and assured software development.

Isabelle/HOL

- Interactive theorem prover for **Higher Order Logic** (started in **1986**).
- isabelle.in.tum.de 
- Developed by Larry Paulson, Tobias Nipkow, Markarius Wenzel.
- **HOL** is a powerful functional specification language.
- Like **Haskell**: data structures, recursive functions, type classes, etc.
- **Readable proofs** in “natural deduction” style (**Isar**).
- Large online library of **formalised mathematics**.
- Archive of Formal Proofs www.isa-afp.org/.
- “Quantum and Classical Registers”, “Category Theory for ZFC”, ...
- Support for verified **code generation**.
- Implemented in the **ML** functional language and **Scala**.
- Program verification and **assured software development**.

Assured Development with Isabelle

- Isabelle is more than an interactive theorem prover.
- Powerful automated tools (e.g., ATPs) used in system development.
- Flexible Proof IDE (PIDE) for programming, proof, and tool integration.
- Machine-checked document model with an executable parser.
- Additional grammar categories support domain-specific languages.
- Unicode symbols for type setting mathematics.
- Support for mixing formal and informal content.
- Plugins to integrate with external tools (ATPs, SAT, Coq, etc.).
- An ideal platform for assured software development.

Assured Development with Isabelle

- Isabelle is more than an interactive theorem prover.
- Powerful automated tools (e.g., ATPs) used in system development.
- Flexible Proof IDE (PIDE) for programming, proof, and tool integration.
- Machine-checked document model with an extensible parser.
- Additional grammar categories support domain specific languages.
- Unicode symbols for type setting mathematics.
- Support for mixing formal and informal content.
- Plugins to interface with external tools (ATPs, SMT, CAS, etc.).
- An ideal platform for assured software development.

Assured Development with Isabelle

- Isabelle is more than an interactive theorem prover.
- Powerful automated tools (e.g., ATPs) used in system development.
- Flexible Proof IDE (PIDE) for programming, proof, and tool integration.
- Machine-checked document model with an extensible parser.
- Additional grammar categories support domain specific languages.
- Unicode symbols for type setting mathematics.
- Support for mixing formal and informal content.
- Plugins to interface with external tools (ATPs, SMT, CAS, etc.).
- An ideal platform for assured software development.

Assured Development with Isabelle

- Isabelle is more than an interactive theorem prover.
- Powerful automated tools (e.g., ATPs) used in system development.
- Flexible Proof IDE (**PIDE**) for programming, proof, and tool integration.
- Machine-checked **document model** with an extensible parser.
- Additional grammar categories support **domain specific languages**.
- **Unicode symbols** for type setting mathematics.
- Support for mixing **formal** and **informal** content.
- Plugins to interface with external tools (ATPs, SMT, CAS, etc.).
- An ideal platform for **assured software development**.

Assured Development with Isabelle

- Isabelle is more than an interactive theorem prover.
- Powerful automated tools (e.g., ATPs) used in system development.
- Flexible Proof IDE (**PIDE**) for programming, proof, and tool integration.
- Machine-checked **document model** with an extensible parser.
- Additional grammar categories support **domain specific languages**.
- **Unicode symbols** for type setting mathematics.
- Support for mixing **formal** and **informal** content.
- Plugins to interface with external tools (ATPs, SMT, CAS, etc.).
- An ideal platform for **assured software development**.

Assured Development with Isabelle

- Isabelle is more than an interactive theorem prover.
- Powerful automated tools (e.g., ATPs) used in system development.
- Flexible Proof IDE (**PIDE**) for programming, proof, and tool integration.
- Machine-checked **document model** with an extensible parser.
- Additional grammar categories support **domain specific languages**.
- Unicode symbols for type setting mathematics.
- Support for mixing formal and informal content.
- Plugins to interface with external tools (ATPs, SMT, CAS, etc.).
- An ideal platform for assured software development.

Assured Development with Isabelle

- Isabelle is more than an interactive theorem prover.
- Powerful automated tools (e.g., ATPs) used in system development.
- Flexible Proof IDE (**PIDE**) for programming, proof, and tool integration.
- Machine-checked **document model** with an extensible parser.
- Additional grammar categories support **domain specific languages**.
- **Unicode symbols** for type setting mathematics.
- Support for mixing **formal** and **informal** content.
- Plugins to interface with external tools (ATPs, SMT, CAS, etc.).
- An ideal platform for **assured software development**.

Assured Development with Isabelle

- Isabelle is more than an interactive theorem prover.
- Powerful automated tools (e.g., ATPs) used in system development.
- Flexible Proof IDE (**PIDE**) for programming, proof, and tool integration.
- Machine-checked **document model** with an extensible parser.
- Additional grammar categories support **domain specific languages**.
- **Unicode symbols** for type setting mathematics.
- Support for mixing **formal** and **informal** content.
- Plugins to interface with external tools (ATPs, SMT, CAS, etc.).
- An ideal platform for assured software development.

Assured Development with Isabelle

- Isabelle is more than an interactive theorem prover.
- Powerful automated tools (e.g., ATPs) used in system development.
- Flexible Proof IDE (**PIDE**) for programming, proof, and tool integration.
- Machine-checked **document model** with an extensible parser.
- Additional grammar categories support **domain specific languages**.
- **Unicode symbols** for type setting mathematics.
- Support for mixing **formal** and **informal** content.
- Plugins to interface with external tools (**ATPs**, **SMT**, **CAS**, etc.).
- An ideal platform for assured software development.

Assured Development with Isabelle

- Isabelle is more than an interactive theorem prover.
- Powerful automated tools (e.g., ATPs) used in system development.
- Flexible Proof IDE (**PIDE**) for programming, proof, and tool integration.
- Machine-checked **document model** with an extensible parser.
- Additional grammar categories support **domain specific languages**.
- **Unicode symbols** for type setting mathematics.
- Support for mixing **formal** and **informal** content.
- Plugins to interface with external tools (**ATPs**, **SMT**, **CAS**, etc.).
- An ideal platform for **assured software development**.

seL4.verified in Isabelle/HOL

- Formal verification of OS microkernel (self-extracting).
- seL4: small, secure microkernel with 1,700 lines of C.
- Different processes are isolated, strictly partitioned memory.
- Behaviour and implementation modelled in Isabelle/HOL (2009).

Isabelle/HOL proof gives a strong guarantee that Φ 's property holds

seL4.verified in Isabelle/HOL

- Formal verification of OS **microkernel** (sel4.systems/):

• sel4.systems/ (Security, Confidentiality, Availability, Integrity)

• sel4.systems/ (Formal verification of the microkernel)

• Behaviour and hardware abstractions models in Isabelle/HOL (2009)

• sel4.systems/ (Formal verification of the microkernel's property holds)

seL4.verified in Isabelle/HOL

- Formal verification of OS **microkernel** (sel4.systems/): **L4.verified**

SEL4: A Small End-to-End Verified Microkernel

Formal Verification of the seL4 Microkernel

Behavioral and System Model in Isabelle/HOL

seL4: A Small End-to-End Verified Microkernel

seL4.verified in Isabelle/HOL

- Formal verification of OS **microkernel** (sel4.systems/): **L4.verified**
- **seL4**: small secure microkernel with 8,700 lines of C.
- Different processes are **isolated**: strictly partitioned memory.
- Behaviour and implementation modelled in Isabelle/HOL (2009).

The binary code of the seL4 microkernel correctly implements the behaviour described in its abstract specification and nothing more.
Further, the Isabelle/HOL proof of seL4 formally verifies the security properties called integrity and confidentiality.

- Isabelle/HOL proof gives a strong guarantee that this property holds.

seL4.verified in Isabelle/HOL

- Formal verification of OS **microkernel** (sel4.systems/): **L4.verified**
- **seL4**: small secure microkernel with 8,700 lines of C.
- Different processes are **isolated**: strictly partitioned memory.
- Behaviour and implementation modelled in Isabelle/HOL (2009).

“The binary code of the seL4 microkernel correctly implements the behaviour described in its abstract specification and nothing more.”

“... safety properties of confidentiality and integrity.”

- Isabelle/HOL proof gives a strong guarantee that this property holds.

seL4.verified in Isabelle/HOL

- Formal verification of OS **microkernel** (sel4.systems/): **L4.verified**
- **seL4**: small secure microkernel with 8,700 lines of C.
- Different processes are **isolated**: strictly partitioned memory.
- Behaviour and implementation modelled in Isabelle/HOL (2009).

“The binary code of the seL4 microkernel correctly implements the behaviour described in its abstract specification and nothing more. Furthermore, the specification and the seL4 binary satisfy the classic security properties called integrity and confidentiality.”

- Isabelle/HOL proof gives a strong guarantee that this property holds.

seL4.verified in Isabelle/HOL

- Formal verification of OS **microkernel** (sel4.systems/): **L4.verified**
- **seL4**: small secure microkernel with 8,700 lines of C.
- Different processes are **isolated**: strictly partitioned memory.
- Behaviour and implementation modelled in Isabelle/HOL (2009).

*“The binary code of the seL4 microkernel **correctly implements** the behaviour described in its abstract specification and **nothing more**. Furthermore, the specification and the seL4 binary satisfy the classic **security properties** called integrity and confidentiality.”*

- Isabelle/HOL proof gives a strong guarantee that this property holds.

seL4.verified in Isabelle/HOL

- Formal verification of OS **microkernel** (sel4.systems/): **L4.verified**
- **seL4**: small secure microkernel with 8,700 lines of C.
- Different processes are **isolated**: strictly partitioned memory.
- Behaviour and implementation modelled in Isabelle/HOL (2009).

*“The binary code of the seL4 microkernel **correctly implements** the behaviour described in its abstract specification and **nothing more**. Furthermore, the specification and the seL4 binary satisfy the classic **security properties** called integrity and confidentiality.”*

- Isabelle/HOL proof gives a **strong guarantee** that this property holds.

Other Proof Tools

- There are many proof assistants besides Isabelle/HOL
 - Mathematically principled: Coq, Lean, Agda, Idris, Metamath
 - Highly automated: PVS, ACL2, Z/Everest
 - Usable and bespoke: KeY, KeYmaera X, Atelier B Prover
- Isabelle provides a reasonable trade-off between all three aspects
 - E.g. no dependent types but a high degree of automation
- Isabelle's industrial strength with a proven track record (as does Coq)
- Many exciting developments in other tools (e.g. Lean)

Other Proof Tools

- There are many proof assistants besides Isabelle/HOL.
- Mathematically principled: Coq, Lean, Agda, Idris, Mizar.
- Highly automated: PVS, ACL2, Z/Eves.
- Usable and bespoke: KeY, KeYmaera X, Atelier B Prover.
- Isabelle provides a reasonable trade-off between all three aspects.
- E.g., no dependent types but a high degree of automation.
- Isabelle: industrial strength with a proven track record (as does Coq).
- Worth watching developments in other tools (e.g., Lean).

Other Proof Tools

- There are many proof assistants besides Isabelle/HOL.
- **Mathematically principled**: Coq, Lean, Agda, Idris, Mizar.
- **Highly automated**: PVS, ACL2, Z/Eves.
- **Usable and bespoke**: KeY, KeYmaera X, Atelier B Prover.
- Isabelle provides a reasonable trade-off between all three aspects.
- E.g., no dependent types but a high degree of automation.
- Isabelle: **industrial strength** with a proven track record (as does Coq).
- Worth watching developments in other tools (e.g., **Lean**).

Other Proof Tools

- There are many proof assistants besides Isabelle/HOL.
- **Mathematically principled**: Coq, Lean, Agda, Idris, Mizar.
- **Highly automated**: PVS, ACL2, Z/Eves.
- **Usable and bespoke**: KeY, KeYmaera X, Atelier B Prover.
- Isabelle provides a reasonable trade-off between all three aspects.
- E.g., no dependent types but a high degree of automation.
- Isabelle: **industrial strength** with a proven track record (as does Coq).
- Worth watching developments in other tools (e.g., **Lean**).

Other Proof Tools

- There are many proof assistants besides Isabelle/HOL.
- **Mathematically principled**: Coq, Lean, Agda, Idris, Mizar.
- **Highly automated**: PVS, ACL2, Z/Eves.
- **Usable and bespoke**: KeY, KeYmaera X, Atelier B Prover.
- Isabelle provides a reasonable trade-off between all three aspects.
- E.g., no dependent types but a high degree of automation.
- Isabelle: **industrial strength** with a proven track record (as does Coq).
- Worth watching developments in other tools (e.g., **Lean**).

Other Proof Tools

- There are many proof assistants besides Isabelle/HOL.
- **Mathematically principled**: Coq, Lean, Agda, Idris, Mizar.
- **Highly automated**: PVS, ACL2, Z/Eves.
- **Usable and bespoke**: KeY, KeYmaera X, Atelier B Prover.
- Isabelle provides a reasonable trade-off between all three aspects.
 - E.g., no dependent types but a high degree of automation.
 - Isabelle: **industrial strength** with a proven track record (as does Coq).
 - Worth watching developments in other tools (e.g., **Lean**).

Other Proof Tools

- There are many proof assistants besides Isabelle/HOL.
- **Mathematically principled**: Coq, Lean, Agda, Idris, Mizar.
- **Highly automated**: PVS, ACL2, Z/Eves.
- **Usable and bespoke**: KeY, KeYmaera X, Atelier B Prover.
- Isabelle provides a reasonable trade-off between all three aspects.
- E.g., no **dependent types** but a high degree of automation.
- Isabelle: **industrial strength** with a proven track record (as does Coq).
- Worth watching developments in other tools (e.g., **Lean**).

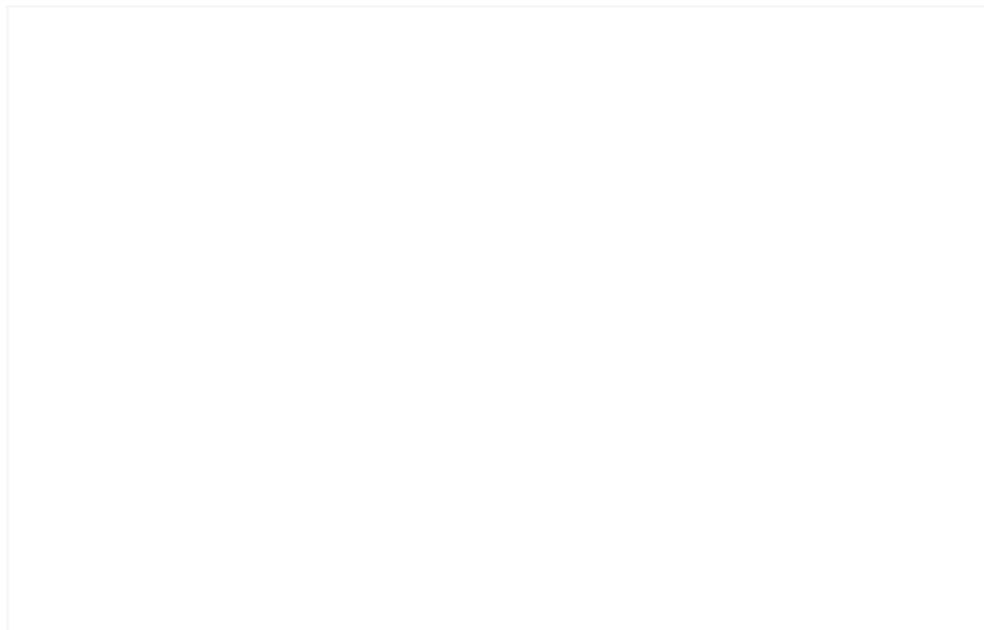
Other Proof Tools

- There are many proof assistants besides Isabelle/HOL.
- **Mathematically principled**: Coq, Lean, Agda, Idris, Mizar.
- **Highly automated**: PVS, ACL2, Z/Eves.
- **Usable and bespoke**: KeY, KeYmaera X, Atelier B Prover.
- Isabelle provides a reasonable trade-off between all three aspects.
- E.g., no **dependent types** but a high degree of automation.
- Isabelle: **industrial strength** with a proven track record (as does Coq).
- Worth watching developments in other tools (e.g., **Lean**).

Other Proof Tools

- There are many proof assistants besides Isabelle/HOL.
- **Mathematically principled**: Coq, Lean, Agda, Idris, Mizar.
- **Highly automated**: PVS, ACL2, Z/Eves.
- **Usable and bespoke**: KeY, KeYmaera X, Atelier B Prover.
- Isabelle provides a reasonable trade-off between all three aspects.
- E.g., no **dependent types** but a high degree of automation.
- Isabelle: **industrial strength** with a proven track record (as does Coq).
- Worth watching developments in other tools (e.g., **Lean**).

Mathematical Proofs in Isabelle/HOL



- Cantor's Theorem
- Cardinality of \mathbb{R} greater than cardinality of \mathbb{N}

• www.cse.unsw.edu.au/~dreyer/isabelle/
• www.haskell.org/haskellwiki/Haskell_in_HOL

Mathematical Proofs in Isabelle/HOL

```
theorem Cantor:
  fixes f :: "'a ⇒ 'a set"
  shows "∃ S :: 'a set. S ∉ range f"
proof
  let ?S = "{x. x ∉ f x}"
  show "?S ∉ range f"
  proof
    assume "?S ∈ range f"
    then obtain y where "?S = f y" ..
    thus False
    proof (rule equalityCE)
      assume "y ∈ f y" assume "y ∈ ?S" then have "y ∉ f y" ..
      with <y ∈ f y> show ?thesis by contradiction
    next
      assume "y ∉ ?S" assume "y ∉ f y" then have "y ∈ ?S" ..
      with <y ∉ ?S> show ?thesis by contradiction
    qed
  qed
qed
```

Mathematical Proofs in Isabelle/HOL

```
theorem Cantor:
  fixes f :: "'a ⇒ 'a set"
  shows "∃ S :: 'a set. S ∉ range f"
proof
  let ?S = "{x. x ∉ f x}"
  show "?S ∉ range f"
  proof
    assume "?S ∈ range f"
    then obtain y where "?S = f y" ..
    thus False
    proof (rule equalityCE)
      assume "y ∈ f y" assume "y ∈ ?S" then have "y ∉ f y" ..
      with <y ∈ f y> show ?thesis by contradiction
    next
      assume "y ∉ ?S" assume "y ∉ f y" then have "y ∈ ?S" ..
      with <y ∉ ?S> show ?thesis by contradiction
    qed
  qed
qed
```

- Cantor's Theorem.
- Cardinality $\mathbb{P} A$ greater than cardinality A .
- Top 100 theorems online:
- www.cse.unsw.edu.au/~kleing/top100/

Mathematical Proofs in Isabelle/HOL

```
theorem Cantor:
  fixes f :: "'a ⇒ 'a set"
  shows "∃ S :: 'a set. S ∉ range f"
proof
  let ?S = "{x. x ∉ f x}"
  show "?S ∉ range f"
  proof
    assume "?S ∈ range f"
    then obtain y where "?S = f y" ..
    thus False
    proof (rule equalityCE)
      assume "y ∈ f y" assume "y ∈ ?S" then have "y ∉ f y" ..
      with <y ∈ f y> show ?thesis by contradiction
    next
      assume "y ∉ ?S" assume "y ∉ f y" then have "y ∈ ?S" ..
      with <y ∉ ?S> show ?thesis by contradiction
    qed
  qed
qed
```

- Cantor's Theorem.
- Cardinality $\mathbb{P} A$ greater than cardinality A .

- Top 100 theorems online:
- www.cse.unsw.edu.au/~kleing/top100/

Mathematical Proofs in Isabelle/HOL

```
theorem Cantor:
  fixes f :: "'a ⇒ 'a set"
  shows "∃ S :: 'a set. S ∉ range f"
proof
  let ?S = "{x. x ∉ f x}"
  show "?S ∉ range f"
  proof
    assume "?S ∈ range f"
    then obtain y where "?S = f y" ..
    thus False
    proof (rule equalityCE)
      assume "y ∈ f y" assume "y ∈ ?S" then have "y ∉ f y" ..
      with <y ∈ f y> show ?thesis by contradiction
    next
      assume "y ∉ ?S" assume "y ∉ f y" then have "y ∈ ?S" ..
      with <y ∉ ?S> show ?thesis by contradiction
    qed
  qed
qed
```

- Cantor's Theorem.
- Cardinality $\mathbb{P} A$ greater than cardinality A .
- Top 100 theorems online:
 - www.cse.unsw.edu.au/~kleing/top100/

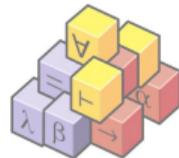
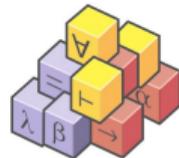
Mathematical Proofs in Isabelle/HOL

```
theorem Cantor:
  fixes f :: "'a ⇒ 'a set"
  shows "∃ S :: 'a set. S ∉ range f"
proof
  let ?S = "{x. x ∉ f x}"
  show "?S ∉ range f"
  proof
    assume "?S ∈ range f"
    then obtain y where "?S = f y" ..
    thus False
    proof (rule equalityCE)
      assume "y ∈ f y" assume "y ∈ ?S" then have "y ∉ f y" ..
      with <y ∈ f y> show ?thesis by contradiction
    next
      assume "y ∉ ?S" assume "y ∉ f y" then have "y ∈ ?S" ..
      with <y ∉ ?S> show ?thesis by contradiction
    qed
  qed
qed
```

- Cantor's Theorem.
- Cardinality $\mathbb{P} A$ greater than cardinality A .
- Top 100 theorems online:
- www.cse.unsw.edu.au/~kleing/top100/

Overview

- 1 What is a Formal Proof?
- 2 Automated versus Interactive Theorem Provers
- 3 The Isabelle/HOL Proof Assistant
- 4 Theory Documents**
- 5 Isabelle/jEdit and Proof IDE



Theory Documents

- Isabelle supports document-centric development approach.
- We interact with Isabelle editing theory documents in the frontend.
- Plugins contain ML code that executes updates in the backend.

Theory Documents

- Isabelle supports **document-centric** development approach.
- We interact with Isabelle editing theory documents in the **frontend**.
- **Plugins** contain ML code that execute updates in the **backend**.

Theory Documents

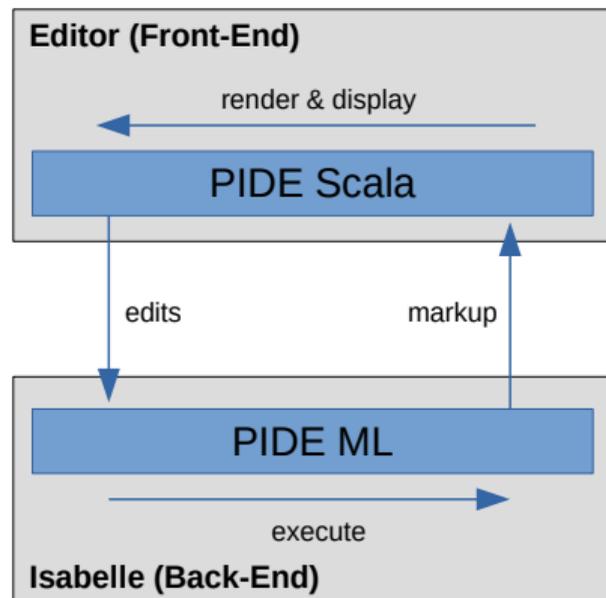
- Isabelle supports **document-centric** development approach.
- We interact with Isabelle editing theory documents in the **frontend**.
- **Plugins** contain ML code that execute updates in the **backend**.

Theory Documents

- Isabelle supports **document-centric** development approach.
- We interact with Isabelle editing theory documents in the **frontend**.
- **Plugins** contain ML code that execute updates in the **backend**.

Theory Documents

- Isabelle supports **document-centric** development approach.
- We interact with Isabelle editing theory documents in the **frontend**.
- **Plugins** contain ML code that execute updates in the **backend**.



A Simple Isabelle Theory

```
theory MyTheory
  imports Main
begin

definition double :: "nat  $\Rightarrow$  nat" where
  "double x = x + x"

value "double 7" (* Returns 14 *)

end
```

A Simple Isabelle Theory

```
theory MyTheory
  imports Main
begin

definition double :: "nat  $\Rightarrow$  nat" where
  "double x = x + x"

value "double 7" (* Returns 14 *)

end
```

Theory Document Structure

- **Definition**
 - Definitional layer with simple combinatory parser.
 - Sequence or hierarchy of commands.
 - Commands have a major keyword and several minor keywords.
 - Can also refer to terms (inner syntax).
- **Implementation**
 - "Meta" (or derived by "speech marks") or contextual glyphs used.
 - More sophisticated multi-stage parser supporting meta.
 - Binary, ternary, quaternary (etc.) operators with various associativities.
 - Define program's logic terms, which are checked and certified.

Theory Document Structure

- **Outer Syntax.**

- Definitional layer with simple combinatory parser.
- Sequence or hierarchy of **commands**.
- Commands have a **major** keyword and several **minor** keywords.
- Can also refer to terms (inner syntax).

- **Inner Syntax.**

- Transition denoted by speech marks "... " or cartouche glyphs <...>.
- More sophisticated multi-stage parser supporting mixfix.
- Binary, ternary, quaternary (etc.) operators with various associations.
- Isabelle produces logic terms, which are **checked** and **certified**.

Theory Document Structure

- **Outer Syntax.**
- Definitional layer with simple combinatory parser.
 - Sequence or hierarchy of **commands**.
 - Commands have a **major** keyword and several **minor** keywords.
 - Can also refer to terms (inner syntax).
- **Inner Syntax.**
 - Transition denoted by speech marks "... " or cartouche glyphs <...>.
 - More sophisticated multi-stage parser supporting mixfix.
 - Binary, ternary, quaternary (etc.) operators with various associations.
 - Isabelle produces logic terms, which are **checked** and **certified**.

Theory Document Structure

- **Outer Syntax.**
- Definitional layer with simple combinatory parser.
- Sequence or hierarchy of **commands**.
- Commands have a **major** keyword and several **minor** keywords.
- Can also refer to terms (inner syntax).

- **Inner Syntax.**
- Transition denoted by speech marks "... " or cartouche glyphs <...>.
- More sophisticated multi-stage parser supporting mixfix.
- Binary, ternary, quaternary (etc.) operators with various associations.
- Isabelle produces logic terms, which are **checked** and **certified**.

Theory Document Structure

- **Outer Syntax.**
- Definitional layer with simple combinatory parser.
- Sequence or hierarchy of **commands**.
- Commands have a **major** keyword and several **minor** keywords.
- Can also refer to terms (inner syntax).

- **Inner Syntax.**
- Transition denoted by speech marks "... " or cartouche glyphs <...>.
- More sophisticated multi-stage parser supporting mixfix.
- Binary, ternary, quaternary (etc.) operators with various associations.
- Isabelle produces logic terms, which are **checked** and **certified**.

Theory Document Structure

- **Outer Syntax.**
- Definitional layer with simple combinatory parser.
- Sequence or hierarchy of **commands**.
- Commands have a **major** keyword and several **minor** keywords.
- Can also refer to **terms** (inner syntax).

- **Inner Syntax.**
- Transition denoted by speech marks "... " or cartouche glyphs <...>.
- More sophisticated multi-stage parser supporting mixfix.
- Binary, ternary, quaternary (etc.) operators with various associations.
- Isabelle produces logic terms, which are **checked** and **certified**.

Theory Document Structure

- **Outer Syntax.**
 - Definitional layer with simple combinatory parser.
 - Sequence or hierarchy of **commands**.
 - Commands have a **major** keyword and several **minor** keywords.
 - Can also refer to **terms** (inner syntax).
- **Inner Syntax.**
 - Transition denoted by speech marks "... " or cartouche glyphs <...>.
 - More sophisticated multi-stage parser supporting mixfix.
 - Binary, ternary, quaternary (etc.) operators with various associations.
 - Isabelle produces logic terms, which are **checked** and **certified**.

Theory Document Structure

- **Outer Syntax.**
 - Definitional layer with simple combinatory parser.
 - Sequence or hierarchy of **commands**.
 - Commands have a **major** keyword and several **minor** keywords.
 - Can also refer to **terms** (inner syntax).
- **Inner Syntax.**
 - Transition denoted by speech marks "... " or cartouche glyphs <...>.
 - More sophisticated multi-stage parser supporting **mixfix**.
 - Binary, ternary, quaternary (etc.) operators with various associations.
 - Isabelle produces logic terms, which are **checked** and **certified**.

Theory Document Structure

- **Outer Syntax.**
 - Definitional layer with simple combinatory parser.
 - Sequence or hierarchy of **commands**.
 - Commands have a **major** keyword and several **minor** keywords.
 - Can also refer to **terms** (inner syntax).
- **Inner Syntax.**
 - Transition denoted by speech marks "... " or cartouche glyphs <...>.
 - More sophisticated multi-stage parser supporting **mixfix**.
 - Binary, ternary, quaternary (etc.) operators with various associations.
 - Isabelle produces logic terms, which are **checked** and **certified**.

Theory Document Structure

- **Outer Syntax.**
 - Definitional layer with simple combinatory parser.
 - Sequence or hierarchy of **commands**.
 - Commands have a **major** keyword and several **minor** keywords.
 - Can also refer to **terms** (inner syntax).
- **Inner Syntax.**
 - Transition denoted by speech marks "... " or cartouche glyphs <...>.
 - More sophisticated multi-stage parser supporting **mixfix**.
 - Binary, ternary, quaternary (etc.) operators with various associations.
 - Isabelle produces logic terms, which are **checked and certified**.

Theory Document Structure

- **Outer Syntax.**
 - Definitional layer with simple combinatory parser.
 - Sequence or hierarchy of **commands**.
 - Commands have a **major** keyword and several **minor** keywords.
 - Can also refer to **terms** (inner syntax).
- **Inner Syntax.**
 - Transition denoted by speech marks "... " or cartouche glyphs <...>.
 - More sophisticated multi-stage parser supporting **mixfix**.
 - Binary, ternary, quaternary (etc.) operators with various associations.
 - Isabelle produces logic terms, which are **checked** and **certified**.

A Simple Isabelle Theory

```
theory MyTheory
  imports Main
begin

definition double :: "nat  $\Rightarrow$  nat" where
"double x = x + x"

value "double 7" (* Returns 14 *)

end
```

- **Outer syntax:** `theory`, `imports`, `begin`, `definition`, `value`, etc.
- **Inner syntax:** `"nat \Rightarrow nat"`, `"double x = x + x"`, `"double 7"`.

A Simple Isabelle Theory

```
theory MyTheory
  imports Main
begin

definition double :: "nat  $\Rightarrow$  nat" where
"double x = x + x"

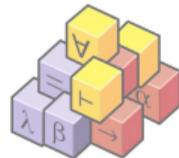
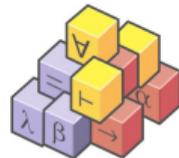
value "double 7" (* Returns 14 *)

end
```

- **Outer syntax:** `theory`, `imports`, `begin`, `definition`, `value`, etc.
- **Inner syntax:** `"nat \Rightarrow nat"`, `"double x = x + x"`, `"double 7"`.

Overview

- 1 What is a Formal Proof?
- 2 Automated versus Interactive Theorem Provers
- 3 The Isabelle/HOL Proof Assistant
- 4 Theory Documents
- 5 Isabelle/jEdit and Proof IDE



Isabelle/jEdit Interface Overview

Isabelle/jEdit Interface Overview

The screenshot displays the Isabelle/jEdit interface. The main window shows the source code for a theory named `MyTheory`. The code includes an import of `Main`, a `begin` block, a `definition` for `double` that maps `nat` to `nat` with the rule `double x = x + x`, and a `value` statement for `"double 7"` with a comment `(* Returns 14 *)`. The `end` block is also present.

On the left, the File Browser shows a tree structure of files, including `Examples`, `Release Notes`, and `Isabelle Tutorials`. The `Isabelle Tutorials` folder is expanded, showing various guides like `prog-prove`, `locales`, `classes`, `datatypes`, `functions`, `corec`, `codegen`, `nitpick`, `sledgehammer`, `eisbach`, `sugar`, `main`, `isar-ref`, `implementation`, and `system`.

On the right, the HyperSearch Results Sidebar is visible, showing a search for `HOL` with results for `Scratch` and `MyTheory`. The `MyTheory` result is selected.

At the bottom, the Console Output shows the result of the `value` statement: `"14"` and `:: "nat"`. The status bar at the very bottom indicates the current file is `(isabelle,isabelle,UTF-8-Isabelle)` and shows system information like `VM: 357/512MB` and `ML: 300/498MB`.

Isabelle/jEdit Interface Overview

The screenshot displays the Isabelle/jEdit interface. The main window shows a code editor with the following content:

```
theory MyTheory
  imports Main
begin

definition double :: "nat ⇒ nat" where
  "double x = x + x"

value "double 7" (* Returns 14 *)

end
```

The left sidebar contains a file browser with the following structure:

- Examples
 - src/HOL/Examples/Seq.thy
 - src/HOL/Examples/Drinker.t
 - src/HOL/Examples/ML.thy
 - src/HOL/Unix/Unix.thy
 - src/Tools/SML/Examples.thy
 - src/Pure/ROOT.ML
 - \$ML_SOURCES/ROOT.ML
- Release Notes
 - ANNOUNCE
 - README
 - NEWS
 - COPYRIGHT
 - CONTRIBUTORS
 - src/Tools/jEdit/README
 - README_REPOSITORY
- Isabelle Tutorials
 - prog-prove: Programming
 - locales: Tutorial on Locales
 - classes: Tutorial on Type C
 - datatypes: Tutorial on (Co)
 - functions: Tutorial on Func
 - corec: Tutorial on Nonprimi
 - codegen: Tutorial on Code
 - nitpick: User's Guide to Nit
 - sledgehammer: User's Gu
 - eisbach: The Eisbach User
 - sugar: LaTeX Sugar for Isab
- Isabelle Reference Manuals
 - main: What's in Main
 - isar-ref: The Isabelle/Isar R
 - implementation: The Isab
 - system: The Isabelle System
 - jedit: Isabelle/jEdit

The right sidebar shows a search panel with the following content:

- Purge Continuous checking
- Prover: ready
- HOL
 - Satch
 - MyTheory
- HyperSearch Results
- Sidekick
- State
- Theories

The bottom status bar shows the following information:

8,34 (130/135) (isabelle,isabelle,UTF-8-Isabelle) | n m r o U G | VM: 380/512 MB | ML: 300/498 MB | 16:31

Isabelle/jEdit Interface Overview

The screenshot displays the Isabelle/jEdit interface. The main window shows a code editor for `MyTheory.thy` with the following content:

```
theory MyTheory
  imports Main
begin

definition double :: "nat ⇒ nat" where
  "double x = x + x"

value "double 7" (* Returns 14 *)

end
```

The left sidebar contains a file browser with a tree view of the project structure, including folders like `Examples`, `Release Notes`, and `Isabelle Tutorials`. The right sidebar shows a search results pane with a filter and a list of search results, including `MyTheory.thy` and `theory MyTheory`.

At the bottom, there is a toolbar with various symbols and a status bar showing the current file encoding and system information.

Unicode Syntax

• LaTeX 2_ε includes commands for math symbols.

• In TeXy files use symbol code (similar to \<math> markup).

\<alpha> ~ α , \<forall> ~ \forall , \<dashv> ~ \dashv , \<sigma> ~ σ

• Autocompletion: partially typed symbol names appear in a list.

• Select and press TAB key.

\<ASCII> ~ \<math> , \<math> ~ \<math> , \<math> ~ \<math> , \<math> ~ \<math>

• Text modifiers: \<bold> , \<slit> , \<sup> , \<sub> , \

• In TeXy, use the symbol table at the bottom of the TeXy window.

• Hovering the mouse over a symbol gives its name and abbreviation.

• See also §2.7 of the TeXbook for reference manual on more shortcuts.

Unicode Syntax

- Isabelle/jEdit includes **Unicode font** with mathematical symbols.
- In theory files use **symbol code** (similar to \LaTeX markup).
- $\backslash\langle\text{alpha}\rangle \rightsquigarrow \alpha$, $\backslash\langle\text{forall}\rangle \rightsquigarrow \forall$, $\backslash\langle\text{and}\rangle \rightsquigarrow \wedge$, etc.
- **Autocompletion**: partially typed symbol names appear in a list
- Select and press TAB key.
- **ASCII**: $--> \rightsquigarrow \longrightarrow$, $==> \rightsquigarrow \implies$, $\backslash/ \rightsquigarrow \vee$, $(| \rightsquigarrow (|$, etc.
- **Text modifiers**: $\backslash\langle^{\text{bold}}\rangle$, $\backslash\langle^{\text{sub}}\rangle$, $\backslash\langle^{\text{sup}}\rangle$, etc.
- If in doubt, use the **symbol table** at the bottom of the jEdit window.
- Hovering the mouse over a symbol gives its name and abbreviations.
- See also §2.2 of the Isabelle/jEdit reference manual on more shortcuts.

Unicode Syntax

- Isabelle/jEdit includes **Unicode font** with mathematical symbols.
- In theory files use **symbol code** (similar to **L^AT_EX** markup).
- `\<alpha>` \rightsquigarrow α , `\<forall>` \rightsquigarrow \forall , `\<and>` \rightsquigarrow \wedge , etc.
- **Autocompletion**: partially typed symbol names appear in a list
- Select and press **TAB** key.
- **ASCII**: `-->` \rightsquigarrow \longrightarrow , `==>` \rightsquigarrow \implies , `\|` \rightsquigarrow \forall , `(|` \rightsquigarrow $(|$, etc.
- **Text modifiers**: `\<^bold>`, `\<^sub>`, `\<^sup>`, etc.
- If in doubt, use the **symbol table** at the bottom of the jEdit window.
- Hovering the mouse over a symbol gives its name and abbreviations.
- See also §2.2 of the Isabelle/jEdit reference manual on more shortcuts.

Unicode Syntax

- Isabelle/jEdit includes **Unicode font** with mathematical symbols.
- In theory files use **symbol code** (similar to **L^AT_EX** markup).
- `\<alpha>` \rightsquigarrow α , `\<forall>` \rightsquigarrow \forall , `\<and>` \rightsquigarrow \wedge , etc.
- **Autocompletion**: partially typed symbol names appear in a list
- Select and press **TAB** key.
- **ASCII**: `-->` \rightsquigarrow \longrightarrow , `==>` \rightsquigarrow \implies , `\|` \rightsquigarrow \forall , `(|` \rightsquigarrow $(|$, etc.
- **Text modifiers**: `\<^bold>`, `\<^sub>`, `\<^sup>`, etc.
- If in doubt, use the **symbol table** at the bottom of the jEdit window.
- Hovering the mouse over a symbol gives its name and abbreviations.
- See also §2.2 of the Isabelle/jEdit reference manual on more shortcuts.

Unicode Syntax

- Isabelle/jEdit includes **Unicode font** with mathematical symbols.
- In theory files use **symbol code** (similar to **L^AT_EX** markup).
- `\<alpha>` \rightsquigarrow α , `\<forall>` \rightsquigarrow \forall , `\<and>` \rightsquigarrow \wedge , etc.
- **Autocompletion**: partially typed symbol names appear in a list
 - Select and press **TAB** key.
 - ASCII: `-->` \rightsquigarrow \longrightarrow , `==>` \rightsquigarrow \implies , `\` \rightsquigarrow \vee , `(|` \rightsquigarrow $(|$, etc.
 - Text modifiers: `\<^bold>`, `\<^sub>`, `\<^sup>`, etc.
- If in doubt, use the **symbol table** at the bottom of the jEdit window.
- Hovering the mouse over a symbol gives its name and abbreviations.
- See also §2.2 of the Isabelle/jEdit reference manual on more shortcuts.

Unicode Syntax

- Isabelle/jEdit includes **Unicode font** with mathematical symbols.
- In theory files use **symbol code** (similar to **L^AT_EX** markup).
- `\<alpha>` \rightsquigarrow α , `\<forall>` \rightsquigarrow \forall , `\<and>` \rightsquigarrow \wedge , etc.
- **Autocompletion**: partially typed symbol names appear in a list
- Select and press **TAB key**.
- ASCII: `-->` \rightsquigarrow \longrightarrow , `==>` \rightsquigarrow \implies , `\` \rightsquigarrow \vee , `(|` \rightsquigarrow $(|$, etc.
- Text modifiers: `\<^bold>`, `\<^sub>`, `\<^sup>`, etc.
- If in doubt, use the **symbol table** at the bottom of the jEdit window.
- Hovering the mouse over a symbol gives its name and abbreviations.
- See also §2.2 of the Isabelle/jEdit reference manual on more shortcuts.

Unicode Syntax

- Isabelle/jEdit includes **Unicode font** with mathematical symbols.
- In theory files use **symbol code** (similar to **L^AT_EX** markup).
- `\<alpha>` \rightsquigarrow α , `\<forall>` \rightsquigarrow \forall , `\<and>` \rightsquigarrow \wedge , etc.
- **Autocompletion**: partially typed symbol names appear in a list
- Select and press **TAB key**.
- **ASCII**: `-->` \rightsquigarrow \longrightarrow , `==>` \rightsquigarrow \implies , `\|` \rightsquigarrow \Vdash , `(|` \rightsquigarrow \lvert , etc.
- **Text modifiers**: `\<^bold>`, `\<^sub>`, `\<^sup>`, etc.
- If in doubt, use the **symbol table** at the bottom of the jEdit window.
- Hovering the mouse over a symbol gives its name and abbreviations.
- See also §2.2 of the Isabelle/jEdit reference manual on more shortcuts.

Unicode Syntax

- Isabelle/jEdit includes **Unicode font** with mathematical symbols.
- In theory files use **symbol code** (similar to **L^AT_EX** markup).
- `\<alpha>` \rightsquigarrow α , `\<forall>` \rightsquigarrow \forall , `\<and>` \rightsquigarrow \wedge , etc.
- **Autocompletion**: partially typed symbol names appear in a list
- Select and press **TAB key**.
- **ASCII**: `-->` \rightsquigarrow \longrightarrow , `==>` \rightsquigarrow \implies , `\|` \rightsquigarrow \Vdash , `(|` \rightsquigarrow \lvert , etc.
- **Text modifiers**: `\<^bold>`, `\<^sub>`, `\<^sup>`, etc.
- If in doubt, use the **symbol table** at the bottom of the jEdit window.
- Hovering the mouse over a symbol gives its name and abbreviations.
- See also §2.2 of the Isabelle/jEdit reference manual on more shortcuts.

Unicode Syntax

- Isabelle/jEdit includes **Unicode font** with mathematical symbols.
- In theory files use **symbol code** (similar to $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ markup).
- $\backslash\langle\text{alpha}\rangle \rightsquigarrow \alpha$, $\backslash\langle\text{forall}\rangle \rightsquigarrow \forall$, $\backslash\langle\text{and}\rangle \rightsquigarrow \wedge$, etc.
- **Autocompletion**: partially typed symbol names appear in a list
- Select and press **TAB key**.
- **ASCII**: $--> \rightsquigarrow \longrightarrow$, $==> \rightsquigarrow \implies$, $\backslash/ \rightsquigarrow \vee$, $(| \rightsquigarrow \lvert$, etc.
- **Text modifiers**: $\backslash\langle^{\text{bold}}\rangle$, $\backslash\langle^{\text{sub}}\rangle$, $\backslash\langle^{\text{sup}}\rangle$, etc.
- If in doubt, use the **symbol table** at the bottom of the jEdit window.
 - Hovering the mouse over a symbol gives its name and abbreviations.
 - See also §2.2 of the Isabelle/jEdit reference manual on more shortcuts.

Unicode Syntax

- Isabelle/jEdit includes **Unicode font** with mathematical symbols.
- In theory files use **symbol code** (similar to **L^AT_EX** markup).
- `\<alpha>` \rightsquigarrow α , `\<forall>` \rightsquigarrow \forall , `\<and>` \rightsquigarrow \wedge , etc.
- **Autocompletion**: partially typed symbol names appear in a list
- Select and press **TAB key**.
- **ASCII**: `-->` \rightsquigarrow \longrightarrow , `==>` \rightsquigarrow \implies , `\|` \rightsquigarrow \Vdash , `(|` \rightsquigarrow \lvert , etc.
- **Text modifiers**: `\<^bold>`, `\<^sub>`, `\<^sup>`, etc.
- If in doubt, use the **symbol table** at the bottom of the jEdit window.
- Hovering the mouse over a symbol gives its name and abbreviations.
- See also §2.2 of the Isabelle/jEdit reference manual on more shortcuts.

Unicode Syntax

- Isabelle/jEdit includes **Unicode font** with mathematical symbols.
- In theory files use **symbol code** (similar to **L^AT_EX** markup).
- `\<alpha>` \rightsquigarrow α , `\<forall>` \rightsquigarrow \forall , `\<and>` \rightsquigarrow \wedge , etc.
- **Autocompletion**: partially typed symbol names appear in a list
- Select and press **TAB key**.
- **ASCII**: `-->` \rightsquigarrow \longrightarrow , `==>` \rightsquigarrow \implies , `\|` \rightsquigarrow \Vdash , `(|` \rightsquigarrow \lvert , etc.
- **Text modifiers**: `\<^bold>`, `\<^sub>`, `\<^sup>`, etc.
- If in doubt, use the **symbol table** at the bottom of the jEdit window.
- Hovering the mouse over a symbol gives its name and abbreviations.
- See also §2.2 of the **Isabelle/jEdit reference manual** on more shortcuts.

Overview of Isabelle Documentation

- For more, see isabelle.in.tum.de/documentation.html.
- Available within Isabelle/Edt in the left-hand Documentation tab.
- [Tutorial: Programming and Proving in Isabelle/HOL](#) (2005)
- Introduction to functional programming and how proofs
- Part of the Concrete Semantics book by Nipkow and Klein
- [Isabelle: The Isabelle/Isar Reference Manual](#) (2002)
- Comprehensive overview of commands, syntax, and tactics
- [Isabelle/Edt](#)
- Documentation of the `Edt` interface, including the `Edt` window
- [Isabelle: The Isabelle System Manual](#) (2002)
- Documentation of the Isabelle tool architecture.

Overview of Isabelle Documentation

- For more, see isabelle.in.tum.de/documentation.html.
- Available within Isabelle/jEdit in the left-hand Documentation tab.
- `prog-prove`: *Programming and Proving in Isabelle/HOL*.
- Introduction to functional programming and Isar proofs.
- Part of the *Concrete Semantics* book by Nipkow and Klein.
- `isar-ref`: *The Isabelle/Isar Reference Manual*.
- Comprehensive overview of commands, syntax, and tactics.
- `jedit`: *Isabelle/jEdit*.
- Documentation on the jEdit interface, including shortcuts + tools.
- `system`: *The Isabelle System Manual*.
- Documentation on the Isabelle tool architecture.

Overview of Isabelle Documentation

- For more, see isabelle.in.tum.de/documentation.html.
- Available within Isabelle/jEdit in the left-hand **Documentation** tab.
- `prog-prove`: *Programming and Proving in Isabelle/HOL*.
- Introduction to functional programming and Isar proofs.
- Part of the *Concrete Semantics* book by Nipkow and Klein.
- `isar-ref`: *The Isabelle/Isar Reference Manual*.
- Comprehensive overview of commands, syntax, and tactics.
- `jedit`: *Isabelle/jEdit*.
- Documentation on the jEdit interface, including shortcuts + tools.
- `system`: *The Isabelle System Manual*.
- Documentation on the Isabelle tool architecture.

Overview of Isabelle Documentation

- For more, see isabelle.in.tum.de/documentation.html.
- Available within Isabelle/jEdit in the left-hand **Documentation** tab.
- **prog-prove**: *Programming and Proving in Isabelle/HOL*.
 - Introduction to functional programming and Isar proofs.
 - Part of the *Concrete Semantics* book by Nipkow and Klein.
- **isar-ref**: *The Isabelle/Isar Reference Manual*.
 - Comprehensive overview of commands, syntax, and tactics.
- **jedit**: *Isabelle/jEdit*.
 - Documentation on the jEdit interface, including shortcuts + tools.
- **system**: *The Isabelle System Manual*.
 - Documentation on the Isabelle tool architecture.

Overview of Isabelle Documentation

- For more, see isabelle.in.tum.de/documentation.html.
- Available within Isabelle/jEdit in the left-hand **Documentation** tab.
- **prog-prove**: *Programming and Proving in Isabelle/HOL*.
- Introduction to functional programming and Isar proofs.
 - Part of the *Concrete Semantics* book by Nipkow and Klein.
 - **isar-ref**: *The Isabelle/Isar Reference Manual*.
 - Comprehensive overview of commands, syntax, and tactics.
- **jedit**: *Isabelle/jEdit*.
 - Documentation on the jEdit interface, including shortcuts + tools.
- **system**: *The Isabelle System Manual*.
 - Documentation on the Isabelle tool architecture.

Overview of Isabelle Documentation

- For more, see isabelle.in.tum.de/documentation.html.
- Available within Isabelle/jEdit in the left-hand **Documentation** tab.
- **prog-prove**: *Programming and Proving in Isabelle/HOL*.
- Introduction to functional programming and Isar proofs.
- Part of the *Concrete Semantics* book by Nipkow and Klein.
- **isar-ref**: *The Isabelle/Isar Reference Manual*.
- Comprehensive overview of commands, syntax, and tactics.
- **jedit**: *Isabelle/jEdit*.
- Documentation on the jEdit interface, including shortcuts + tools.
- **system**: *The Isabelle System Manual*.
- Documentation on the Isabelle tool architecture.

Overview of Isabelle Documentation

- For more, see isabelle.in.tum.de/documentation.html.
- Available within Isabelle/jEdit in the left-hand **Documentation** tab.
- **prog-prove**: *Programming and Proving in Isabelle/HOL*.
- Introduction to functional programming and Isar proofs.
- Part of the *Concrete Semantics* book by Nipkow and Klein.
- **isar-ref**: *The Isabelle/Isar Reference Manual*.
 - Comprehensive overview of commands, syntax, and tactics.
- **jedit**: *Isabelle/jEdit*.
 - Documentation on the jEdit interface, including shortcuts + tools.
- **system**: *The Isabelle System Manual*.
 - Documentation on the Isabelle tool architecture.

Overview of Isabelle Documentation

- For more, see isabelle.in.tum.de/documentation.html.
- Available within Isabelle/jEdit in the left-hand **Documentation** tab.
- **prog-prove**: *Programming and Proving in Isabelle/HOL*.
- Introduction to functional programming and Isar proofs.
- Part of the *Concrete Semantics* book by Nipkow and Klein.
- **isar-ref**: *The Isabelle/Isar Reference Manual*.
- Comprehensive overview of commands, syntax, and tactics.
- **jedit**: *Isabelle/jEdit*.
- Documentation on the jEdit interface, including shortcuts + tools.
- **system**: *The Isabelle System Manual*.
- Documentation on the Isabelle tool architecture.

Overview of Isabelle Documentation

- For more, see isabelle.in.tum.de/documentation.html.
- Available within Isabelle/jEdit in the left-hand **Documentation** tab.
- **prog-prove**: *Programming and Proving in Isabelle/HOL*.
- Introduction to functional programming and Isar proofs.
- Part of the *Concrete Semantics* book by Nipkow and Klein.
- **isar-ref**: *The Isabelle/Isar Reference Manual*.
- Comprehensive overview of commands, syntax, and tactics.
- **jedit**: *Isabelle/jEdit*.
- Documentation on the jEdit interface, including shortcuts + tools.
- **system**: *The Isabelle System Manual*.
- Documentation on the Isabelle tool architecture.

Overview of Isabelle Documentation

- For more, see isabelle.in.tum.de/documentation.html.
- Available within Isabelle/jEdit in the left-hand **Documentation** tab.
- **prog-prove**: *Programming and Proving in Isabelle/HOL*.
- Introduction to functional programming and Isar proofs.
- Part of the *Concrete Semantics* book by Nipkow and Klein.
- **isar-ref**: *The Isabelle/Isar Reference Manual*.
- Comprehensive overview of commands, syntax, and tactics.
- **jedit**: *Isabelle/jEdit*.
- Documentation on the jEdit interface, including shortcuts + tools.
- **system**: *The Isabelle System Manual*.
- Documentation on the Isabelle tool architecture.

Overview of Isabelle Documentation

- For more, see isabelle.in.tum.de/documentation.html.
- Available within Isabelle/jEdit in the left-hand **Documentation** tab.
- **prog-prove**: *Programming and Proving in Isabelle/HOL*.
- Introduction to functional programming and Isar proofs.
- Part of the *Concrete Semantics* book by Nipkow and Klein.
- **isar-ref**: *The Isabelle/Isar Reference Manual*.
- Comprehensive overview of commands, syntax, and tactics.
- **jedit**: *Isabelle/jEdit*.
- Documentation on the jEdit interface, including shortcuts + tools.
- **system**: *The Isabelle System Manual*.
- Documentation on the Isabelle tool architecture.

Overview of Isabelle Documentation

- For more, see isabelle.in.tum.de/documentation.html.
- Available within Isabelle/jEdit in the left-hand **Documentation** tab.
- **prog-prove**: *Programming and Proving in Isabelle/HOL*.
- Introduction to functional programming and Isar proofs.
- Part of the *Concrete Semantics* book by Nipkow and Klein.
- **isar-ref**: *The Isabelle/Isar Reference Manual*.
- Comprehensive overview of commands, syntax, and tactics.
- **jedit**: *Isabelle/jEdit*.
- Documentation on the jEdit interface, including shortcuts + tools.
- **system**: *The Isabelle System Manual*.
- Documentation on the Isabelle tool architecture.

Conclusion

This Lecture

- Automated and interactive theorem provers.
- Introduction to Isabelle/HOL.
- Isabelle as a platform for assured software development.
- Theory documents.

Next Lecture

- Functional programming in Isabelle/HOL.

Conclusion

This Lecture

Next Lecture

- Functional programming in Isabelle/HOL.

Conclusion

This Lecture

- Automated and interactive theorem provers.
- Introduction to Isabelle/HOL.
- Isabelle as a platform for assured software development.
- Theory documents.

Next Lecture

- Functional programming in Isabelle/HOL.

Conclusion

This Lecture

- Automated and interactive theorem provers.
- Introduction to Isabelle/HOL.
- Isabelle as a platform for assured software development.
- Theory documents.

Next Lecture

- Functional programming in Isabelle/HOL.

Conclusion

This Lecture

- Automated and interactive theorem provers.
- Introduction to Isabelle/HOL.
- Isabelle as a platform for assured software development.
- Theory documents.

Next Lecture

- Functional programming in Isabelle/HOL.

Conclusion

This Lecture

- Automated and interactive theorem provers.
- Introduction to Isabelle/HOL.
- Isabelle as a platform for assured software development.
- Theory documents.

Next Lecture

- Functional programming in Isabelle/HOL.

Conclusion

This Lecture

- Automated and interactive theorem provers.
- Introduction to Isabelle/HOL.
- Isabelle as a platform for assured software development.
- Theory documents.

Next Lecture

- Functional programming in Isabelle/HOL.

Conclusion

This Lecture

- Automated and interactive theorem provers.
- Introduction to Isabelle/HOL.
- Isabelle as a platform for assured software development.
- Theory documents.

Next Lecture

Conclusion

This Lecture

- Automated and interactive theorem provers.
- Introduction to Isabelle/HOL.
- Isabelle as a platform for assured software development.
- Theory documents.

Next Lecture

- Functional programming in Isabelle/HOL.