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 $x + 2 > 0$.
- 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.

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:

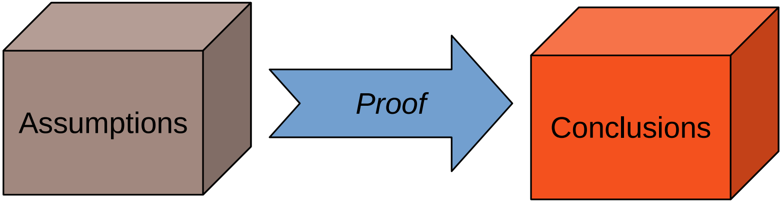
- Theorem provers and proof assistants help us in this process.
- Two main classes: automated and interactive theorem provers

Proof as Computation

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

Proof as Computation

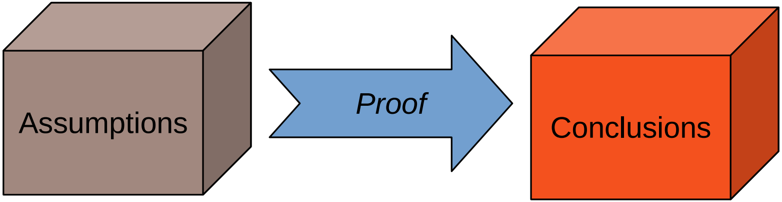
- Analogy between proof and a function mapping inputs to outputs:



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

Proof as Computation

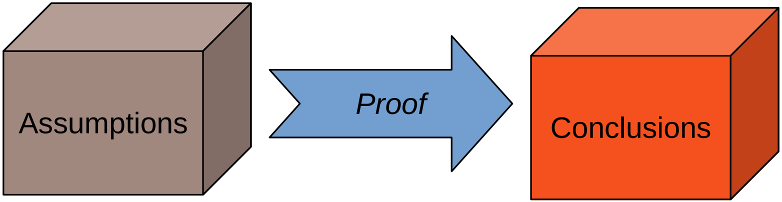
- Analogy between proof and a function mapping inputs to outputs:



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

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

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)

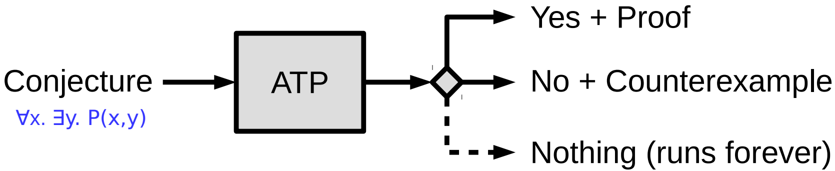
Automated Theorem Provers

- 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

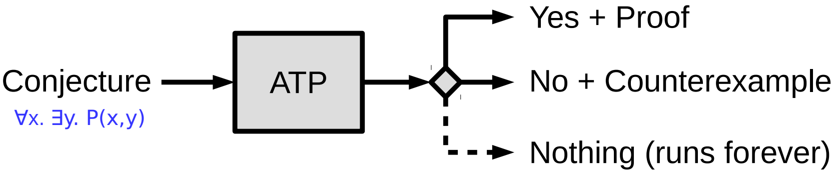
- Automated theorem proving is an undecidable problem.



- 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

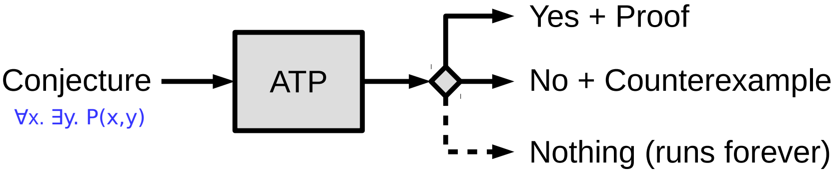
- Automated theorem proving is an undecidable problem.



- 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

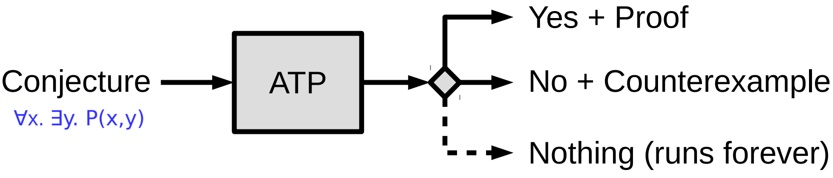
- Automated theorem proving is an undecidable problem.



- 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

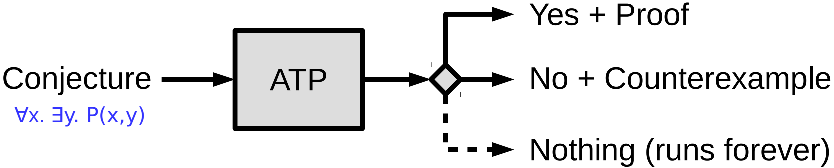
- Automated theorem proving is an undecidable problem.



- 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).

Automated Theorem Provers

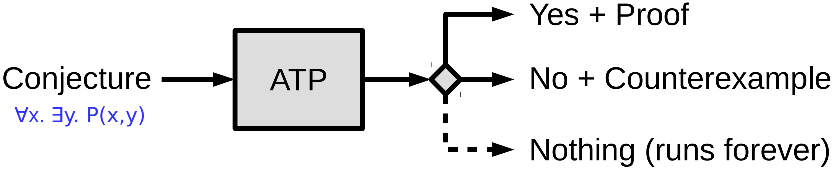
- Automated theorem proving is an undecidable problem.



- 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

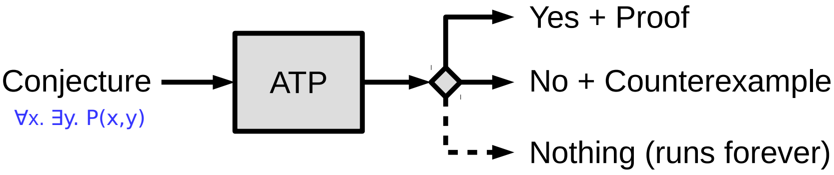
- Automated theorem proving is an undecidable problem.



- 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 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

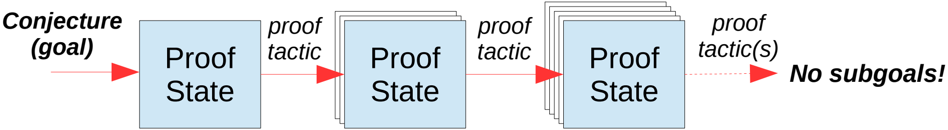
- A proof is a script or program that acts on a proof state.
- Proof state: assumptions and outstanding conjectures ("subgoals").
- Proof tactics and deduction rules subdivide and eliminate proof goals.
- "Divide and conquer" approach to proof.
- Proof: game where winning condition is QED (no more subgoals).

Interactive Theorem Provers

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

Interactive Theorem Provers

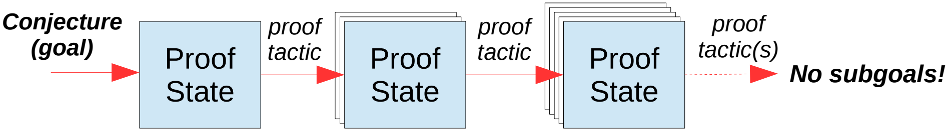
- A proof is a script or program that acts on a proof state.
- Proof state: assumptions and outstanding conjectures (“subgoals”).



- 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

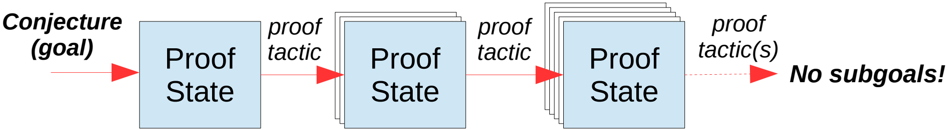
- A proof is a script or program that acts on a proof state.
- Proof state: assumptions and outstanding conjectures (“subgoals”).



- 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

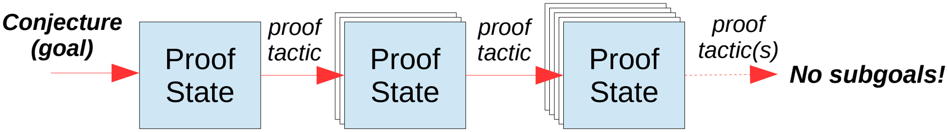
- A proof is a script or program that acts on a proof state.
- Proof state: assumptions and outstanding conjectures (“subgoals”).



- 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).

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

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

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 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.

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 (sel4.systems/)
- 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)

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

seL4.verified in Isabelle/HOL

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

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)
- 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/):
- 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/):
- 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, 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).

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 $\mathbb{P}\,A$ 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/

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

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

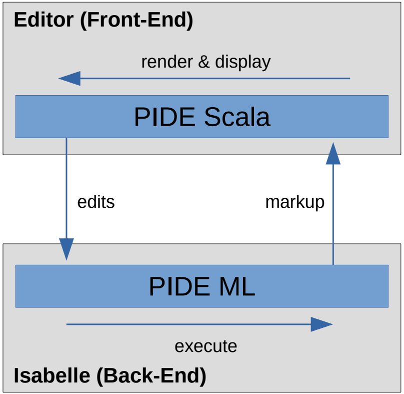
- Isabelle supports document-centric development approach.
- We interact with Isabelle editing theory documents in the frontend.
- Plugins contain ML code that execute updates in the backend.

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.

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 ⇒ nat" where
  "double x = x + x"

value "double 7" (* Returns 14 *)

end
```

A Simple Isabelle Theory

```
theory MyTheory
  imports Main
begin

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

value "double 7" (* Returns 14 *)

end
```

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.

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 ⇒ nat" where
"double x = x + x"

value "double 7" (* Returns 14 *)

end
```

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

A Simple Isabelle Theory

```
theory MyTheory
  imports Main
begin

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

value "double 7" (* Returns 14 *)

end
```

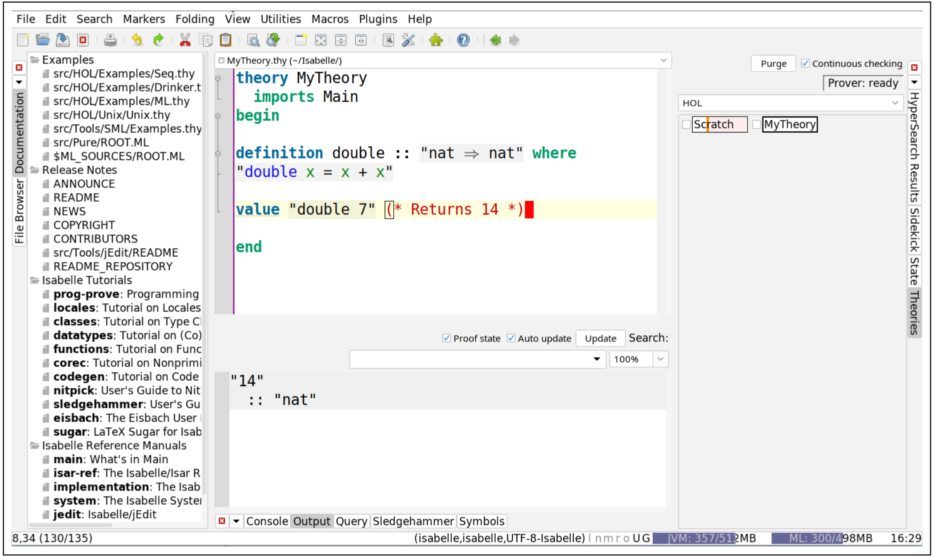
- **Outer syntax**: **theory**, **imports**, **begin**, **definition**, **value**, etc.
- **Inner syntax**: `"nat ⇒ nat"`, `"double x = x + x"`, `"double 7"`.

Overview

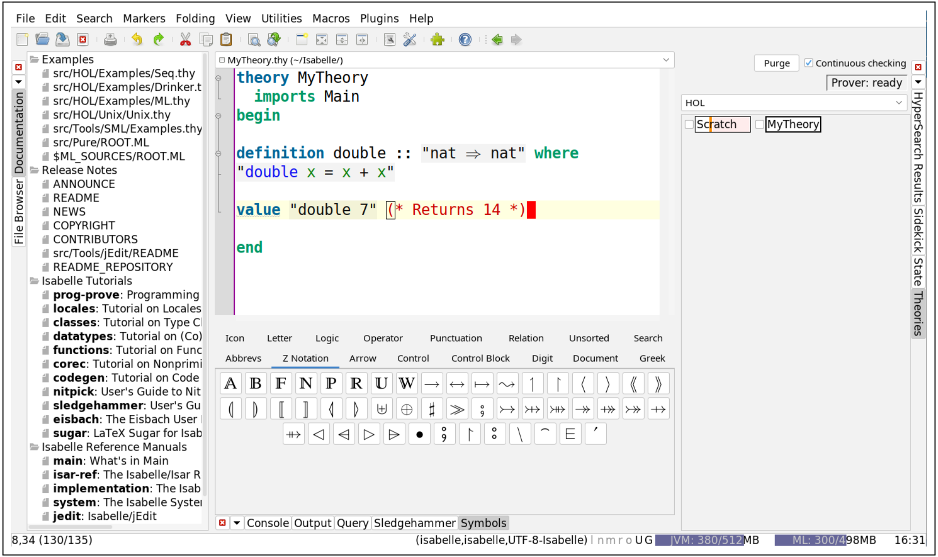
1. What is a Formal Proof?
2. Automated versus Interactive Theorem Provers
3. The Isabelle/HOL Proof Assistant
4. Theory Documents
5. Isabelle/jEdit and Proof IDE

Isabelle/jEdit Interface Overview

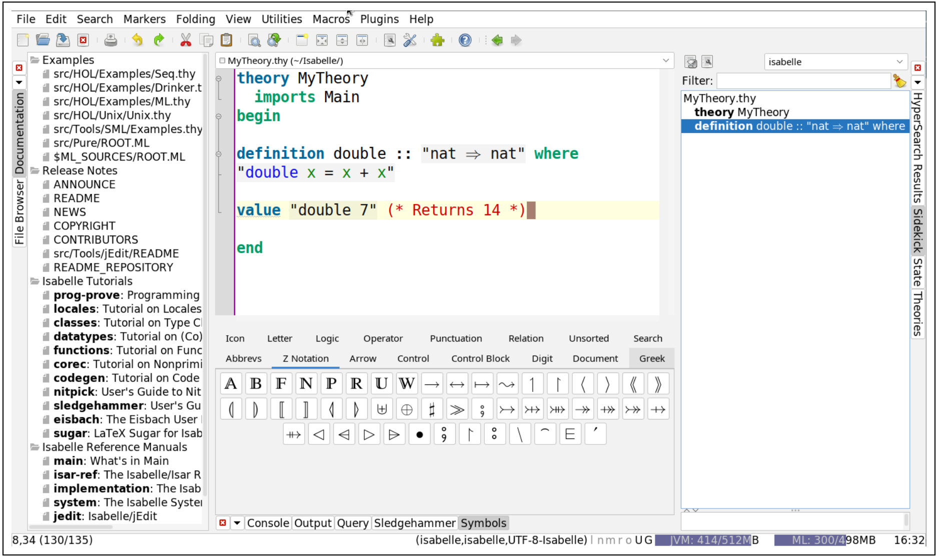
Isabelle/jEdit Interface Overview



Isabelle/jEdit Interface Overview



Isabelle/jEdit Interface Overview



Unicode Syntax

- Isabelle/jEdit includes Unicode font with mathematical symbols
- In theory files use symbol code (similar to LaTeX markup)
- `\<alpha>` $\rightsquigarrow \alpha$, `\<forall>` $\rightsquigarrow \forall$, `\<and>` $\rightsquigarrow \wedge$, etc.

- Autocompletion: partially typed symbol names appear in a list
- Select and press TAB key.
- ASCII: `-->` $\rightsquigarrow \longrightarrow$, `==>` $\rightsquigarrow \Longrightarrow$, `\/` $\rightsquigarrow \vee$, `[|` $\rightsquigarrow [\![$, 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 LaTeX markup).
- `\<alpha>` $\leadsto \alpha$, `\<forall>` $\leadsto \forall$, `\<and>` $\leadsto \wedge$, etc.

- Autocompletion: partially typed symbol names appear in a list
- Select and press `TAB` key.
- ASCII: `-->` $\leadsto \longrightarrow$, `==>` $\leadsto \Longrightarrow$, `\/` $\leadsto \vee$, `(|` $\leadsto (\!|$, 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 LaTeX 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: `-->` $\rightsquigarrow$ $\longrightarrow$, `==>` $\rightsquigarrow$ $\Longrightarrow$, `\/` $\rightsquigarrow$ $\vee$, `(|` $\rightsquigarrow$ ⦇, 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 LaTeX markup).
- \<alpha> $\rightsquigarrow \alpha$, \<forall> $\rightsquigarrow \forall$, \<and> $\rightsquigarrow \wedge$, etc.

- Autocompletion: partially typed symbol names appear in a list
- Select and press TAB key.
- ASCII: --> $\rightsquigarrow \longrightarrow$, ==> $\rightsquigarrow \Longrightarrow$, \/ $\rightsquigarrow \vee$, (| $\rightsquigarrow$ ⦇, 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 LaTeX markup).
- \<alpha> $\leadsto$ α, \<forall> $\leadsto$ $\forall$, \<and> $\leadsto$ $\wedge$, etc.

- Autocompletion: partially typed symbol names appear in a list
- Select and press TAB key.
- ASCII: --> $\leadsto$ $\longrightarrow$, ==> $\leadsto$ $\Longrightarrow$, \/ $\leadsto$ $\vee$, (| $\leadsto$ ⦇, 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 LaTeX markup).
- `\<alpha>` $\leadsto$ α, `\<forall>` $\leadsto$ $\forall$, `\<and>` $\leadsto$ $\wedge$, etc.

- Autocompletion: partially typed symbol names appear in a list
- Select and press `TAB` key.
- ASCII: `-->` $\leadsto$ $\longrightarrow$, `==>` $\leadsto$ $\Longrightarrow$, `\/` $\leadsto$ $\vee$, `(|` $\leadsto$ $\llparenthesis$, 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 $\LaTeX$ markup).
- `\<alpha>` $\leadsto$ α, `\<forall>` $\leadsto$ $\forall$, `\<and>` $\leadsto$ $\wedge$, etc.

- Autocompletion: partially typed symbol names appear in a list
- Select and press `TAB` key.
- ASCII: `-->` $\leadsto$ $\longrightarrow$, `==>` $\leadsto$ $\Longrightarrow$, `\/` $\leadsto$ $\vee$, `(|` $\leadsto$ $\llparenthesis$, 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 LaTeX markup).
- \<alpha> $\leadsto$ α, \<forall> $\leadsto$ $\forall$, \<and> $\leadsto$ $\wedge$, etc.

- Autocompletion: partially typed symbol names appear in a list
- Select and press TAB key.
- ASCII: --> $\leadsto$ $\longrightarrow$, ==> $\leadsto$ $\Longrightarrow$, \/ $\leadsto$ $\vee$, (| $\leadsto$ ⦇, 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 LaTeX markup).
- \<alpha> $\leadsto \alpha$, \<forall> $\leadsto \forall$, \<and> $\leadsto \wedge$, etc.

- Autocompletion: partially typed symbol names appear in a list
- Select and press TAB key.
- ASCII: --> $\leadsto \longrightarrow$, ==> $\leadsto \Longrightarrow$, \/ $\leadsto \vee$, (| $\leadsto$ ⦇, 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 LaTeX markup).
- \<alpha> $\leadsto$ α, \<forall> $\leadsto$ $\forall$, \<and> $\leadsto$ $\wedge$, etc.

- Autocompletion: partially typed symbol names appear in a list
- Select and press TAB key.
- ASCII: --> $\leadsto$ $\longrightarrow$, ==> $\leadsto$ $\Longrightarrow$, \/ $\leadsto$ $\vee$, (| $\leadsto$ $\llparenthesis$, 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 LaTeX markup).
- \<alpha> $\leadsto \alpha$, \<forall> $\leadsto \forall$, \<and> $\leadsto \wedge$, etc.

- Autocompletion: partially typed symbol names appear in a list
- Select and press TAB key.
- ASCII: --> $\leadsto \longrightarrow$, ==> $\leadsto \Longrightarrow$, \/ $\leadsto \vee$, (| $\leadsto$ ⦇, 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/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.

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

- 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

- 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.