Interactive Formal Verification

Review (1-7)

Tjark Weber
Computer Laboratory
University of Cambridge
Isabelle Theories

theory T imports Main A B
begin

end
Isabelle Theories

theory T imports Main A B
begin
end
theory T imports Main A B begin end
Isabelle Theories

theory T imports Main A B
begin

Main: contains all of Isabelle/HOL

Names of existing theories

end
Defining Types

• `typedef ('a,'b) t`
Defining Types

- `typedef ('a,'b) t` introduces an unspecified type
Defining Types

- `typedefcl ('a,'b) t`

  Introduces an unspecified type

  Optional: type arguments
Defining Types

- `typedef ('a,'b) t`
  - Introduces an unspecified type

- `type_synonym 'a multiset = ''a => nat`
  - Optional: type arguments
Defining Types

- **typedeflet** ('a,'b) t
  - Introduces an unspecified type
  - Optional: type arguments

- **type_synonym**
  - Introduces a new name for an existing type
  - 'a multiset = "'a => nat"
Defining Types

- `typedef ('a,'b) t`  
  Introduces an unspecified type

- `type_synonym 'a multiset = "'a => nat"`  
  Introduces a new name for an existing type

- `datatype 'a list = Nil | Cons 'a "'a list"`  
  Optional: type arguments
Defining Types

- \texttt{tiype}
  - \texttt{typedef ('a,'b) t}
  - \texttt{type	extunderscore synonym}
    - \texttt{'a multiset = '\textquoteleft a => nat''}
  - \texttt{datatype}
    - \texttt{'a list = Nil | Cons \ 'a ''\ 'a list''}

Introduces an unspecified type
Optional: type arguments
Introduces a new name for an existing type
Defines an inductive datatype
Defining Types

- **type_decls** ('a,'b) t
  - Introduces an unspecified type
  - Optional: type arguments

- **type_synonym**
  - 'a multiset = '"'a => nat"
    - Defines an inductive datatype

- **datatype** 'a list =
  - Nil | Cons '"'a '"'a list"
    - Constructor names and argument types
Defining Constants

• definition even :: "nat => bool"
  where "even n = (∃k. n = 2*k)"
Defining Constants

For non-recursive definitions

- **definition even :: "nat => bool"**
  
  where "even n = (∃k. n = 2*k)"
Defining Constants

- **definition** even :: "nat => bool"
  
  where "even n = (∃k. n = 2*k)"

For non-recursive definitions

Optional: the constant's type
Defining Constants

For non-recursive definitions

Optional: the constant's type

• definition even :: "nat => bool"
  where "even n = (∃k. n = 2*k)"

Provides a lemma: even_def
Defining Constants

For non-recursive definitions

- **definition** even :: "nat => bool"
  where "even n = (∃k. n = 2*k)"

Optional: the constant's type

- **fun** even' where
  "even' 0 = True"
  | "even' (Suc 0) = False"
  | "even' n = even' (n-2)"

Provides a lemma: even_def
Defining Constants

- **definition** even :: "nat => bool"
  where "even n = (∃k. n = 2*k)"

- **fun** even' where
  "even' 0 = True"
  | "even' (Suc 0) = False"
  | "even' n = even' (n-2)"

For non-recursive definitions
Optional: the constant's type
For recursive functions
Provides a lemma: even_def
Defining Constants

For non-recursive definitions

• definition even :: "nat => bool"
  where "even n = (∃k. n = 2*k)"

Optional: the constant's type

For recursive functions

• fun even' where

  "even' 0 = True"
  | "even' (Suc 0) = False"
  | "even' even' (n-2)"

Provides a lemma: even_def

Provides even'..simps and even'.induct
Defining Constants

• `inductive_set tcl` for R :: "('a*'a) set"
  where
  "(x,y):R ==> (x,y):tcl R"
  | "(x,y):tcl R ==> (y,z):tcl R
    ==> (x,z):tcl R"
Defining Constants

For inductive sets

- **inductive_set** tcl
  
  for R :: "('a*'a) set"

  where

  "(x,y):R ==> (x,y):tcl R"
  
  | "(x,y):tcl R ==> (y,z):tcl R
  
  ==> (x,z):tcl R"
Defining Constants

For inductive sets

Optional: the constant's type

• `inductive_set tcl` for R :: "('a*'a) set"
where

  "(x,y):R ==> (x,y):tcl R"

  | "(x,y):tcl R ==> (y,z):tcl R
  ==> (x,z):tcl R"
Defining Constants

- `inductive_set tcl` for `R :: "('a*'a) set"` where
  
  
  - "(x,y):R ==> (x,y):tcl R"
  - "(x,y):tcl R ==> (y,z):tcl R ==> (x,z):tcl R"

Parameters (types are optional again)

Optional: the constant's type

For inductive sets
Defining Constants

• `inductive_set tcl` for R :: "('a*'a) set"
  where
  "(x,y):R ==> (x,y):tcl R"
  | "(x,y):tcl R ==> (y,z):tcl R ==> (x,z):tcl R"

For inductive sets
Optional: the constant's type
Parameters (types are optional again)

Provides `tcl.cases, tcl.induct, tcl.intros and tcl.simps`
Theorems and Proofs

- lemma add_com [simp]: "x+y = y+x"
- apply method
- done
- by method
- oops
- sorry
Theorems and Proofs

- lemma add_com [simp]: "x + y = y + x"
- apply method
- done
- by method
- oops
- sorry
Theorems and Proofs

- **lemma** add_com [simp]: "x + y = y + x"

- **apply** method

- **done**

- **by** method

- **oops**

- **sorry**
Theorems and Proofs

- **lemma** add_com [simp]: "x+y = y+x"
- **apply** method
- **done**
- **by** method
- **oops**
- **sorry**
Theorems and Proofs

- **lemma** `add_com [simp]: "x+y = y+x"
- **apply** method
- **done**
- **by** method
- **oops**
- **sorry**
Theorems and Proofs

- **lemma** `add_com [simp]`: "\(x+y = y+x\)"

- **apply** `method`

- **done**

- **by** `method`

- **oops**

- **sorry**
Theorems and Proofs

- **lemma** `add_com [simp]: "x+y = y+x"`
- **apply** `method`
- **done**
- **by** `method`
- **oops**
- **sorry**

Starts a proof

Optional: a name and attributes

Modifies some subgoal(s)

Finishes a proof

Finishes a proof in a single step

Aborts a proof attempt
Theorems and Proofs

- **lemma** `add_comm` [simp]: "x + y = y + x"
- **apply** method
- **done**
- **by** method
- **oops**
- **sorry**
Automated Proof Methods

- (induct x y arbitrary: z rule: r.induct)
- (simp add: l1 del: l2)
- (auto simp add: l1 intro: l2)
- (blast intro: l1 elim: l2)
- arith
- (metis l1 l2 l3)
- sledgehammer
Automated Proof Methods

- (induct x y arbitrary. z rule: r.induct)
- (simp add: l1 del: l2)
- (auto simp add: l1 intro: l2)
- (blast intro: l1 elim: l2)
- arith
- (metis l1 l2 l3)
- sledgehammer
Automated Proof Methods

- (induct x y arbitrary. z rule: r.induct)
- (simp add: l1 del: l2)
- (auto simp add: l1 intro: l2)
- (blast intro: l1 elim: l2)
- arith
- (metis l1 l2 l3)
- sledgehammer
Automated Proof Methods

- (induct x y arbitrary: z rule: r.induct)
- (simp add: l1 del: l2)
- (auto simp add: l1)
- (blast intro: l1 elim: l2)
- arith
- (metis l1 l2 l3)
- sledgehammer
Automated Proof Methods

- `(induct x y arbitrary: z rule: r.induct)`
- `(simp add: l1 del: l2)`
- `(auto simp add: l1)`
- `(blast intro: l1 elim: l2)`
- `arith`
- `(metis l1 l2 l3)`
- `sledgehammer`
Automated Proof Methods

- `(induct x y arbitrary. z rule: r.induct)`
- `(simp add: l1 del: l2)`
- `(auto simp add: l1)`
- `(blast intro: l1 elim: l2)`
- `arith`
- `(metis l1 l2 l3)`
- `sledgehammer`
Automated Proof Methods

- (induct x y arbitrary. z rule: r.induct)
- (simp add: l1 del: l2)
- (auto simp add: l1)
- (blast intro: l1 elim: l2)
- arith
- (metis l1 l2 l3)
- sledgehammer

- Induction
- Simplification
- Simplification and some logic
- Good for sets and quantifiers
- Good for arithmetic goals
- Powerful first-order prover
Automated Proof Methods

- \[(\text{induct } x \ y \ \text{arbitrary. } z \ \text{rule: } r.\text{induct})\]
- \[(\text{simp add: } l1 \ \text{del: } l2)\]
- \[(\text{auto simp add: } l1)\]
- \[(\text{blast intro: } l1 \ \text{elim: } l2)\]
- \[\text{arith}\]
- \[(\text{metis } l1 \ l2 \ l3)\]
- \[\text{sledgehammer}\]

**Induction**
- Powerful first-order prover
- Finds lemmas for metis

**Simplification**
- Simplification and some logic
- Good for sets and quantifiers
- Good for arithmetic goals

**Simplification and some logic**
- Good for sets and quantifiers
- Good for arithmetic goals
Basic Methods for Rules

\[ \text{thm: } \boxed{[| P_1; \ldots; P_n |]} \implies Q \]

- (rule thm)
- (erule thm)
- (drule thm)
- (frule thm)
- (rule_tac x="..." and y="..." in thm)
Basic Methods for Rules

\[ \text{thm: } "[| P_1; \ldots; P_n |] \implies Q" \]

- (rule thm)
- (erule thm)
- (drule thm)
- (frule thm)
- (rule_tac x="\ldots" and y="\ldots" in thm)
Basic Methods for Rules

thm: "[| P1; ...; Pn |] ==> Q"

- (rule thm)
- (erule thm)
- (drule thm)
- (frule thm)
- (rule_tac x="..." and y="..." in thm)
Basic Methods for Rules

thm: "[| P1; ...; Pn |] ==> Q"

- (rule thm)
- (erule thm)
- (drule thm)
- (frule thm)
- (rule_tac x="..." and y="..." in thm)
Basic Methods for Rules

\[ \text{thm: } "[| P_1; \ldots; P_n |] \implies Q" \]

- \text{(rule thm)}
  - Unifies Q with the conclusion

- \text{(erule thm)}
  - Unifies Q; unifies P_1 with some assumption

- \text{(drule thm)}
  - Unifies P_1 with some assumption

- \text{(frule thm)}
  - Like \text{drule}, but does not delete the assumption

- \text{(rule_tac x="\ldots" and y="\ldots" in thm)}
Basic Methods for Rules

thm: "[| P1; ...; Pn |] ==> Q"

- (rule thm)
  - Unifies Q with the conclusion

- (erule thm)
  - Unifies Q; unifies P1 with some assumption

- (drule thm)
  - Unifies P1 with some assumption

- (frule thm)
  - Like drule, but does not delete the assumption

- (rule_tac x="..." and y="..." in thm)
  - Manual instantiation of variables
Insiders' Tips

- `term "..."`
- `thm name`
- Find theorems
- Isabelle > Settings > Display ...
- Isabelle > Show me ...
Insiders' Tips

- `term "..."`
- `thm name`
- Find theorems
- Isabelle > Settings > Display ...
- Isabelle > Show me ...
Insiders' Tips

- **term "..."**
  - Prints a term (with its type)

- **thm name**
  - Prints a specific theorem

- Find theorems
- Isabelle > Settings > Display ...
- Isabelle > Show me ...
Insiders' Tips

- **term "..."** - Prints a term (with its type)
- **thm name** - Prints a specific theorem
- Find theorems - Search for theorems by pattern
- Isabelle > Settings > Display ...
- Isabelle > Show me ...
Insiders' Tips

- **term "..."**
  - Prints a term (with its type)
- **thm name**
  - Prints a specific theorem
- **Find theorems**
  - Search for theorems by pattern
- **Isabelle > Settings > Display ...**
  - Show types, sorts etc.
- **Isabelle > Show me ...**
Insiders' Tips

- **term "..."**
  - Prints a term (with its type)

- **thm name**
  - Prints a specific theorem

- **Find theorems**
  - Search for theorems by pattern

- **Isabelle > Settings > Display ...**
  - Show types, sorts etc.

- **Isabelle > Show me ...**
  - Show all commands, all methods etc.