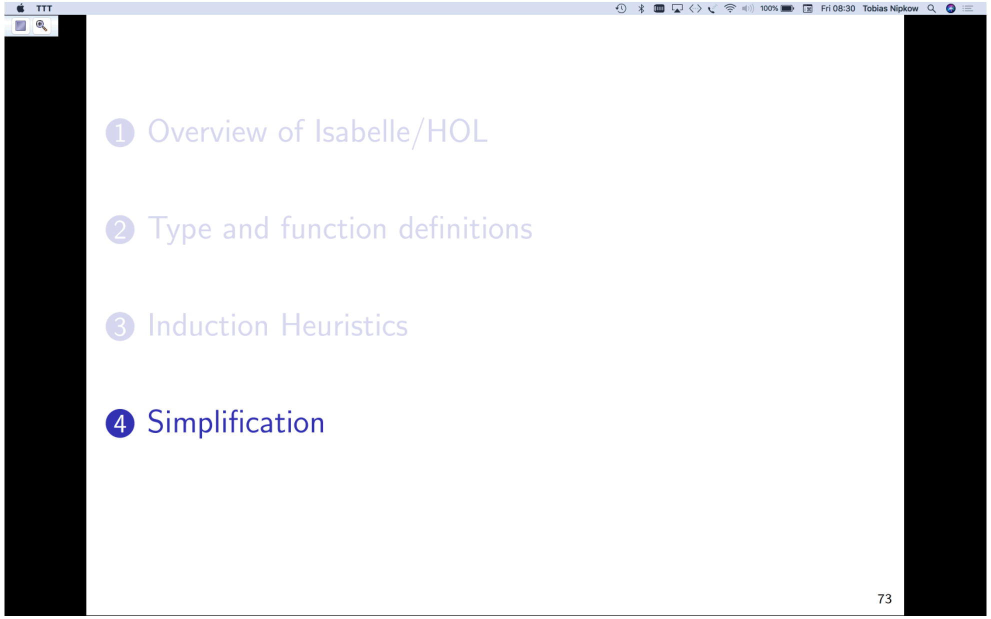
Script generated by TTT

Title: FDS (10.05.2019)

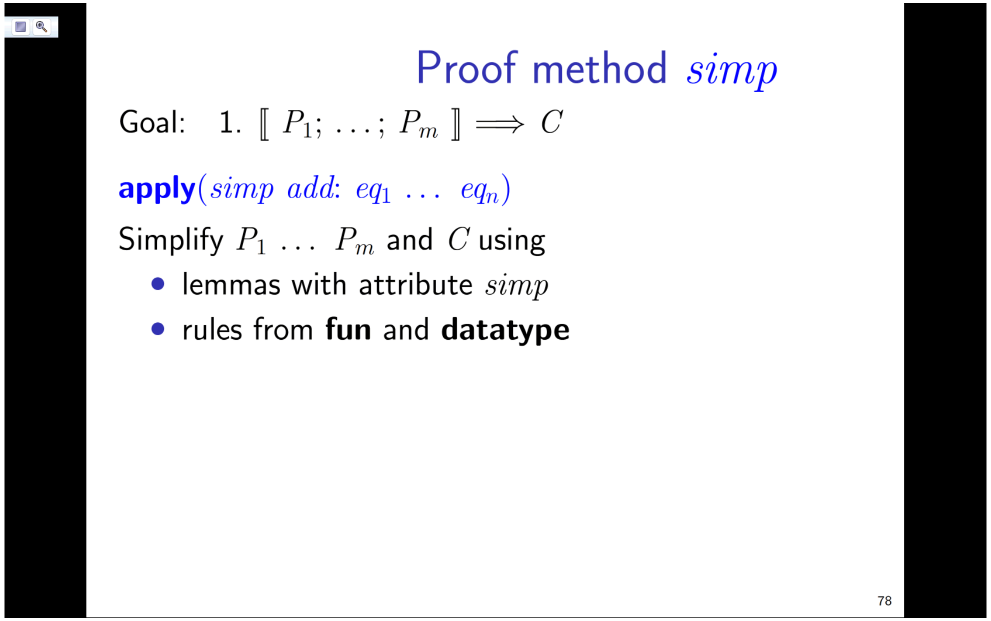
Date: Fri May 10 08:30:32 CEST 2019

Duration: 89:37 min

Pages: 76



1. Overview of Isabelle/HOL
2. Type and function definitions
3. Induction Heuristics
4. Simplification



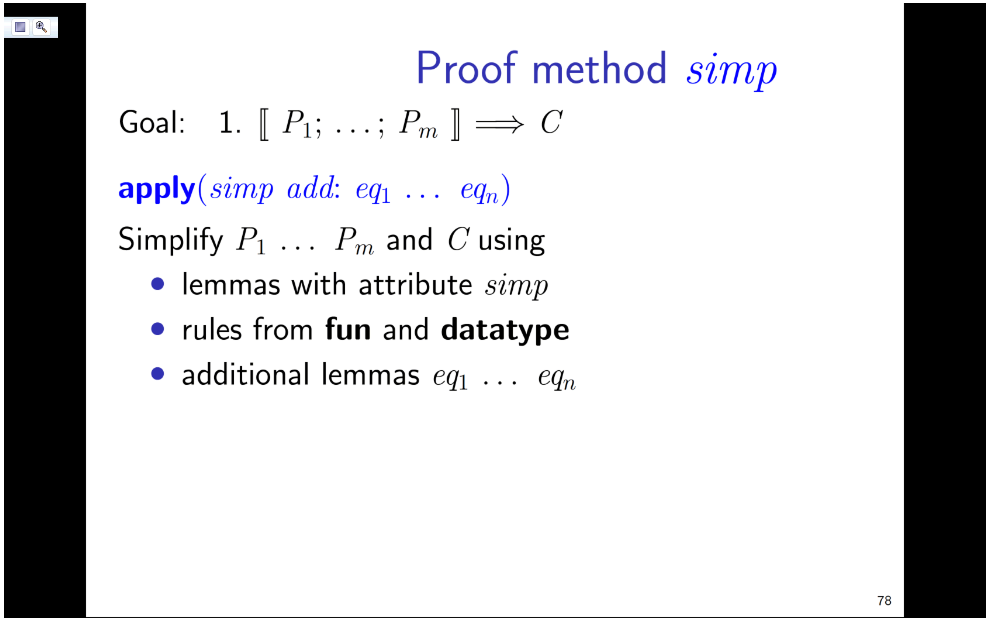
Proof method *simp*

Goal: 1. $[\![P_1; \ldots; P_m]\!] \Longrightarrow C$

apply$(simp\ add\!: eq_1\ \ldots\ eq_n)$

Simplify $P_1\ \ldots\ P_m$ and C using

- lemmas with attribute *simp*
- rules from **fun** and **datatype**



Proof method *simp*

Goal: 1. $[\![P_1; \ldots; P_m]\!] \Longrightarrow C$

apply$(simp\ add\!: eq_1\ \ldots\ eq_n)$

Simplify $P_1\ \ldots\ P_m$ and C using

- lemmas with attribute *simp*
- rules from **fun** and **datatype**
- additional lemmas $eq_1\ \ldots\ eq_n$

Proof method *simp*

Goal: 1. $\llbracket P_1; \ldots; P_m \rrbracket \Longrightarrow C$

apply$(simp\ add\colon\ eq_1\ \ldots\ eq_n)$

Simplify $P_1\ \ldots\ P_m$ and C using

- lemmas with attribute *simp*
- rules from **fun** and **datatype**
- additional lemmas $eq_1\ \ldots\ eq_n$
- assumptions $P_1\ \ldots\ P_m$

Proof method *simp*

Goal: 1. $\llbracket P_1; \ldots; P_m \rrbracket \Longrightarrow C$

apply$(simp\ add\colon\ eq_1\ \ldots\ eq_n)$

Simplify $P_1\ \ldots\ P_m$ and C using

- lemmas with attribute *simp*
- rules from **fun** and **datatype**
- additional lemmas $eq_1\ \ldots\ eq_n$
- assumptions $P_1\ \ldots\ P_m$

Proof method *simp*

Goal: 1. $\llbracket P_1; \ldots; P_m \rrbracket \Longrightarrow C$

apply$(simp\ add\colon\ eq_1\ \ldots\ eq_n)$

Simplify $P_1\ \ldots\ P_m$ and C using

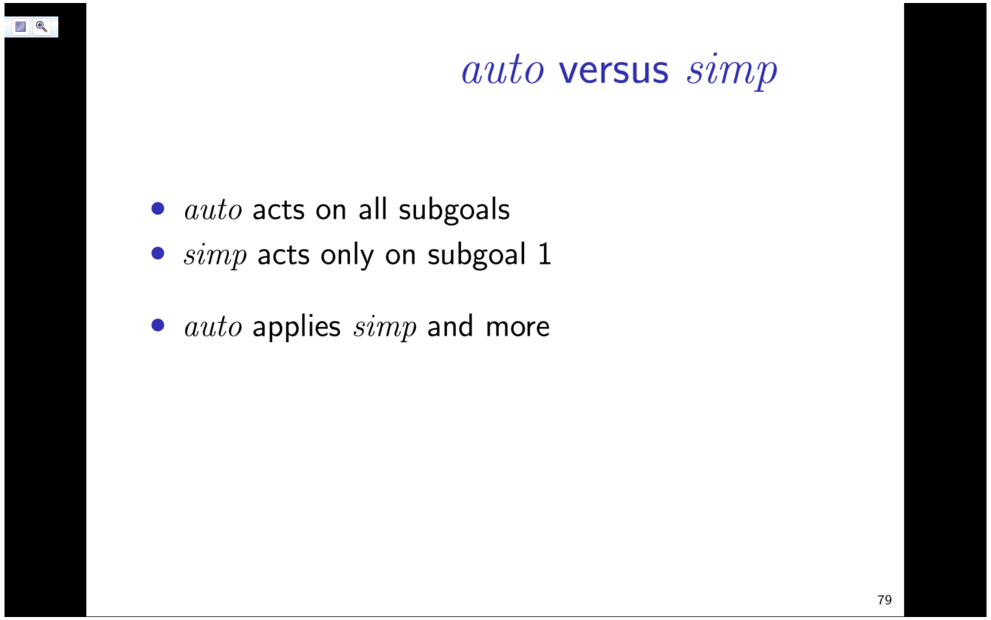
- lemmas with attribute *simp*
- rules from **fun** and **datatype**
- additional lemmas $eq_1\ \ldots\ eq_n$
- assumptions $P_1\ \ldots\ P_m$

Variations:

- $(simp\ \ldots\ del\colon\ \ldots)$ removes *simp*-lemmas
- *add* and *del* are optional

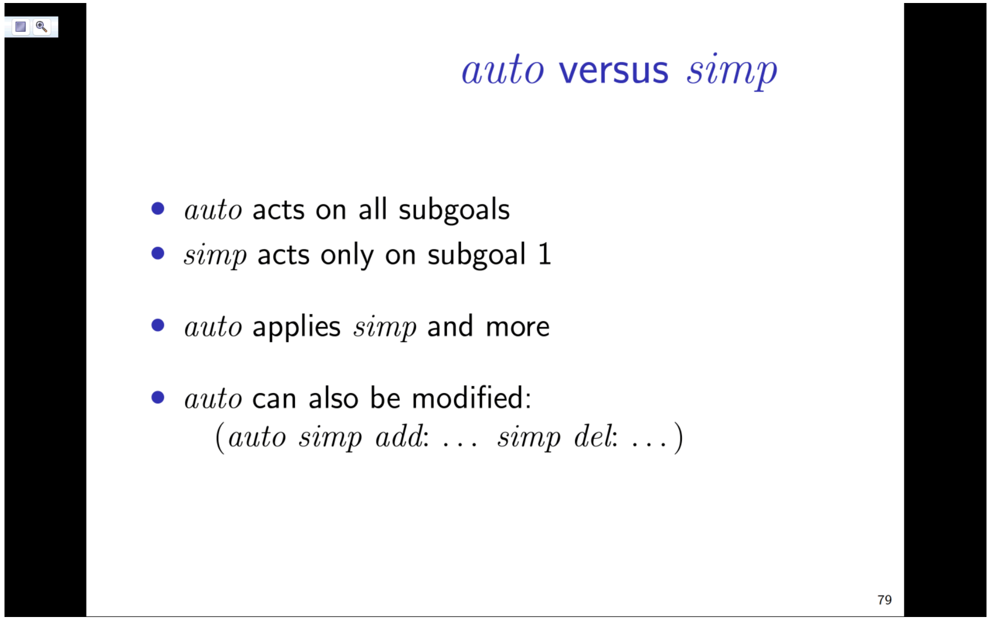
auto versus *simp*

- *auto* acts on all subgoals
- *simp* acts only on subgoal 1



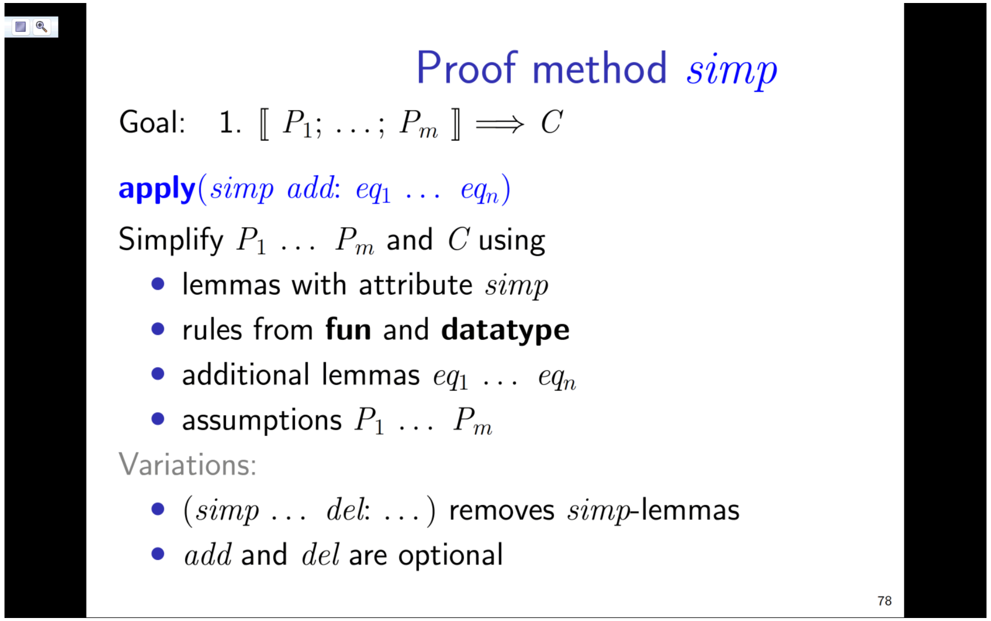
auto versus *simp*

- *auto* acts on all subgoals
- *simp* acts only on subgoal 1
- *auto* applies *simp* and more



auto versus *simp*

- *auto* acts on all subgoals
- *simp* acts only on subgoal 1
- *auto* applies *simp* and more
- *auto* can also be modified:
  $(auto\ simp\ add\colon \dots\ simp\ del\colon \dots)$



Proof method *simp*

Goal: 1. $[\![P_1; \dots; P_m]\!] \Longrightarrow C$

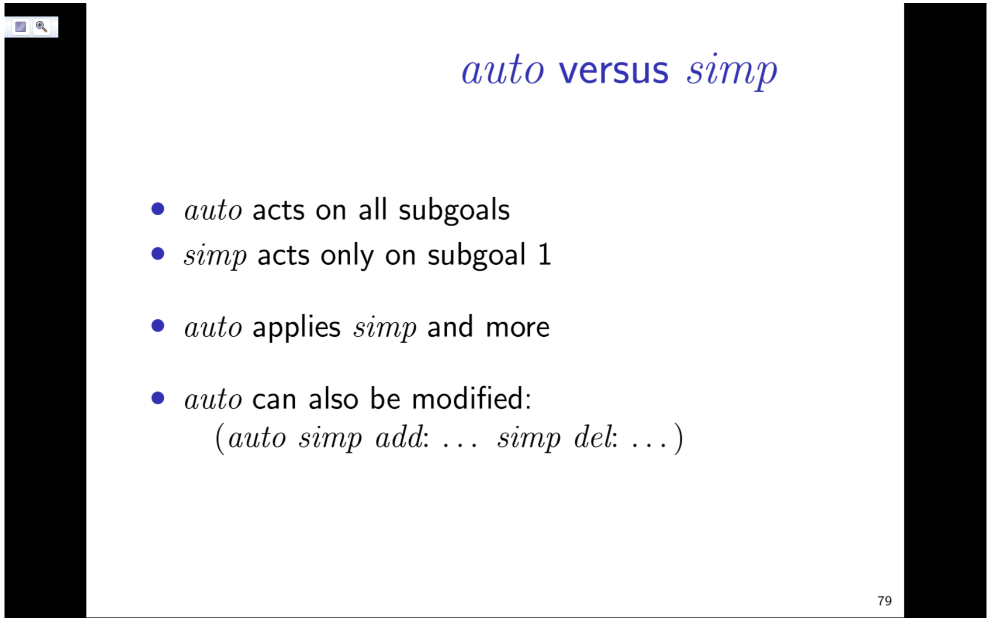
apply$(simp\ add\colon\ eq_1\ \dots\ eq_n)$

Simplify $P_1 \dots P_m$ and C using

- lemmas with attribute *simp*
- rules from **fun** and **datatype**
- additional lemmas $eq_1 \dots eq_n$
- assumptions $P_1 \dots P_m$

Variations:

- $(simp \dots del\colon \dots)$ removes *simp*-lemmas
- *add* and *del* are optional



auto versus *simp*

- *auto* acts on all subgoals
- *simp* acts only on subgoal 1
- *auto* applies *simp* and more
- *auto* can also be modified:
  $(auto\ simp\ add\colon \dots\ simp\ del\colon \dots)$

Rewriting with definitions

Definitions (**definition**) must be used explicitly:

$$(\textit{simp add: f\_def} \dots)$$

Case splitting with *simp/auto*

Automatic:

$$P\ (\textsf{\textit{if}}\ A\ \textsf{\textit{then}}\ s\ \textsf{\textit{else}}\ t) \\ = \\ (A \longrightarrow P(s)) \wedge (\neg A \longrightarrow P(t))$$

Case splitting with *simp/auto*

Automatic:

$$P\ (\textsf{\textit{if}}\ A\ \textsf{\textit{then}}\ s\ \textsf{\textit{else}}\ t) \\ = \\ (A \longrightarrow P(s)) \wedge (\neg A \longrightarrow P(t))$$

By hand:

$$P\ (\textsf{\textit{case}}\ e\ \textsf{\textit{of}}\ 0 \Rightarrow a \mid \mathit{Suc}\ n \Rightarrow b) \\ = \\ (e = 0 \longrightarrow P(a)) \wedge (\forall\, n.\ e = \mathit{Suc}\ n \longrightarrow P(b))$$

Case splitting with *simp/auto*

Automatic:

$$P\ (\textsf{\textit{if}}\ A\ \textsf{\textit{then}}\ s\ \textsf{\textit{else}}\ t) \\ = \\ (A \longrightarrow P(s)) \wedge (\neg A \longrightarrow P(t))$$

By hand:

$$P\ (\textsf{\textit{case}}\ e\ \textsf{\textit{of}}\ 0 \Rightarrow a \mid \mathit{Suc}\ n \Rightarrow b) \\ = \\ (e = 0 \longrightarrow P(a)) \wedge (\forall\, n.\ e = \mathit{Suc}\ n \longrightarrow P(b))$$

Proof method: $(\textit{simp split: nat.split})$

Case splitting with *simp/auto*

Automatic:

$$P\ (\textit{if}\ A\ \textit{then}\ s\ \textit{else}\ t)$$
$$=$$
$$(A \longrightarrow P(s)) \wedge (\neg A \longrightarrow P(t))$$

By hand:

$$P\ (\textit{case}\ e\ \textit{of}\ 0 \Rightarrow a \mid Suc\ n \Rightarrow b)$$
$$=$$
$$(e = 0 \longrightarrow P(a)) \wedge (\forall\, n.\ e = Suc\ n \longrightarrow P(b))$$

Proof method: *(simp split: nat.split)*
Or *auto*. Similar for any datatype *t*: *t.split*

Splitting pairs with *simp/auto*

How to replace

$$P\ (\textit{let}\ (x,\ y) = t\ \textit{in}\ u\ x\ y)$$

Splitting pairs with *simp/auto*

How to replace

$$P\ (\textit{let}\ (x,\ y) = t\ \textit{in}\ u\ x\ y)$$
or
$$P\ (\textit{case}\ t\ \textit{of}\ (x,\ y) \Rightarrow u\ x\ y)$$

Splitting pairs with *simp/auto*

How to replace

$$P\ (\textit{let}\ (x,\ y) = t\ \textit{in}\ u\ x\ y)$$
or
$$P\ (\textit{case}\ t\ \textit{of}\ (x,\ y) \Rightarrow u\ x\ y)$$
by
$$\forall\, x\ y.\ t = (x,\ y) \longrightarrow P\ (u\ x\ y)$$

Splitting pairs with *simp/auto*

How to replace

$$P\ (\textit{let}\ (x,\ y) = t\ \textit{in}\ u\ x\ y)$$

or

$$P\ (\textit{case}\ t\ \textit{of}\ (x,\ y) \Rightarrow\ u\ x\ y)$$

by

$$\forall\, x\ y.\ t = (x,\ y) \longrightarrow P\ (u\ x\ y)$$

Proof method: $(\textit{simp split: prod.split})$

`Simp_Demo.thy`

Preview: sets

Type: $'a\ set$

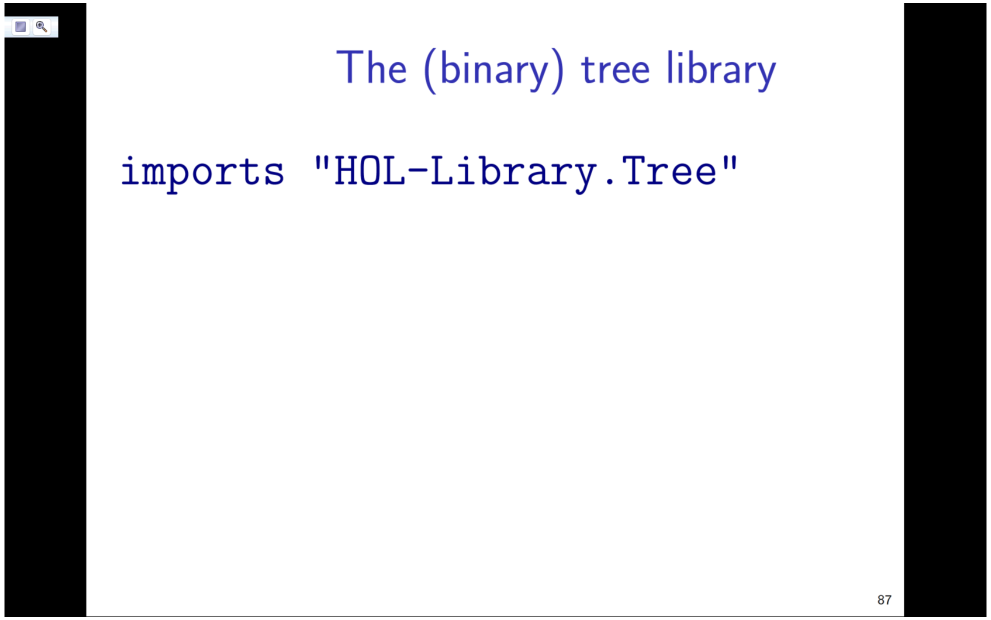
Preview: sets

Type: $'a\ set$

Operations: $a \in A$, $A \cup B$, . . .

Bounded quantification: $\forall\, a{\in}A.\ P$

Proof method $auto$ knows (a little) about sets.



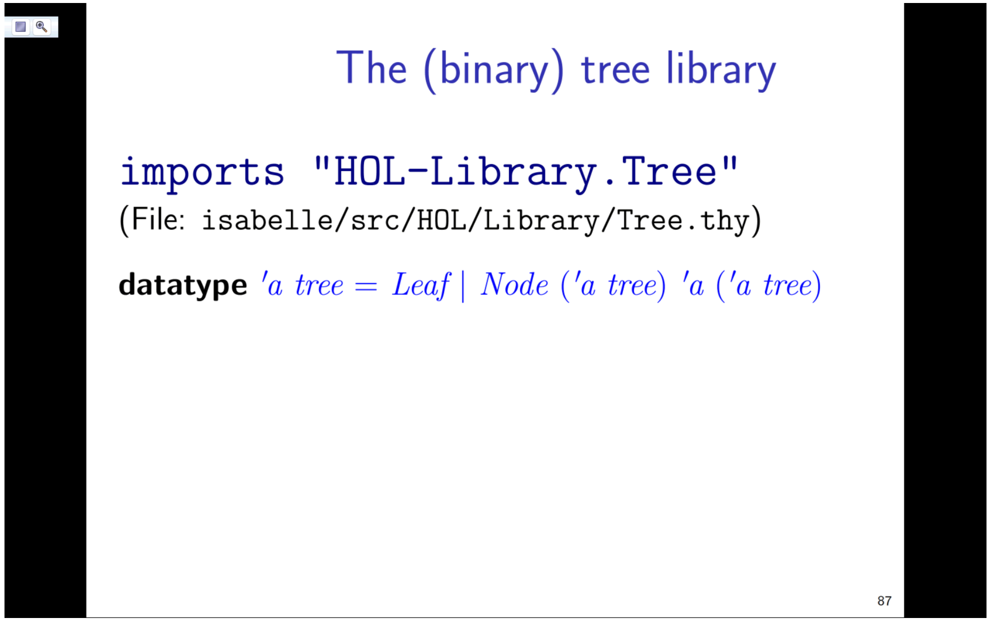
The (binary) tree library

```
imports "HOL-Library.Tree"
```

The (binary) tree library

```
imports "HOL-Library.Tree"
```

(File: `isabelle/src/HOL/Library/Tree.thy`)

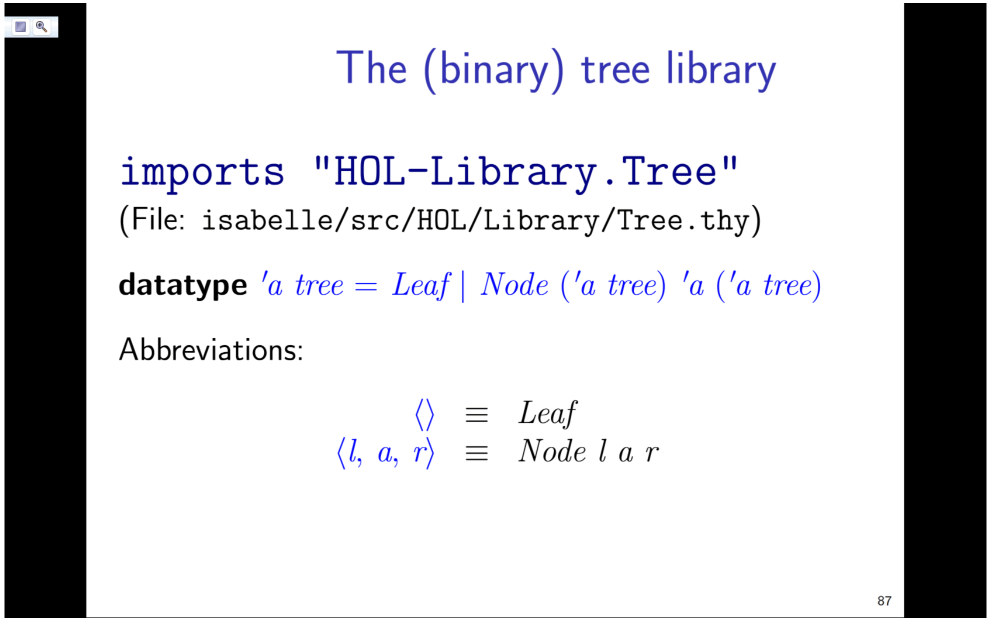


The (binary) tree library

```
imports "HOL-Library.Tree"
```

(File: `isabelle/src/HOL/Library/Tree.thy`)

datatype $'a\ tree = Leaf \mid Node\ ('a\ tree)\ 'a\ ('a\ tree)$



The (binary) tree library

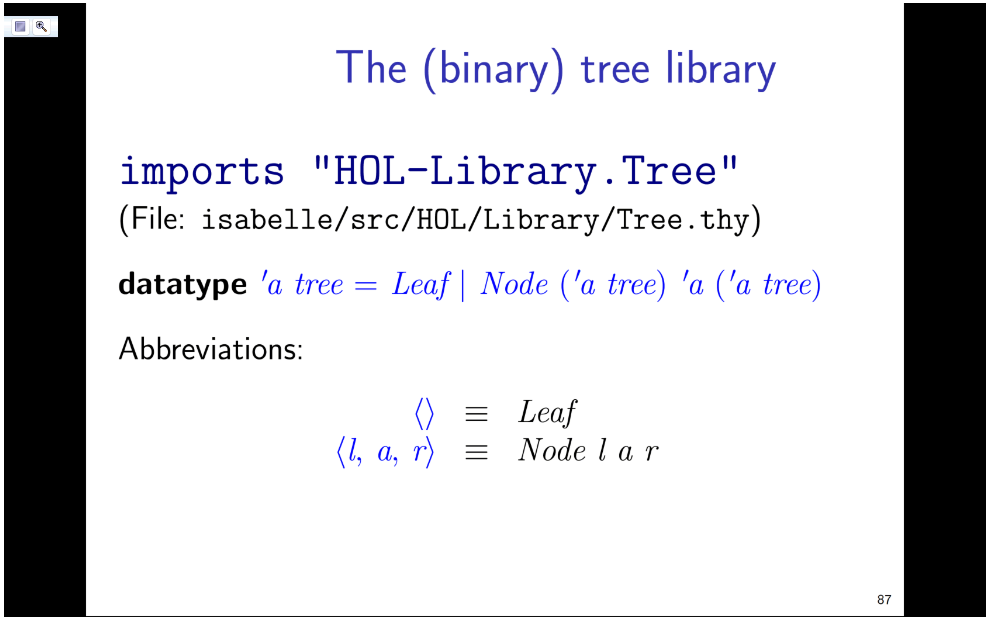
```
imports "HOL-Library.Tree"
```

(File: `isabelle/src/HOL/Library/Tree.thy`)

datatype $'a\ tree = Leaf \mid Node\ ('a\ tree)\ 'a\ ('a\ tree)$

Abbreviations:

$$\begin{array}{rcl} \langle\rangle & \equiv & Leaf \\ \langle l,\ a,\ r\rangle & \equiv & Node\ l\ a\ r \end{array}$$



The (binary) tree library

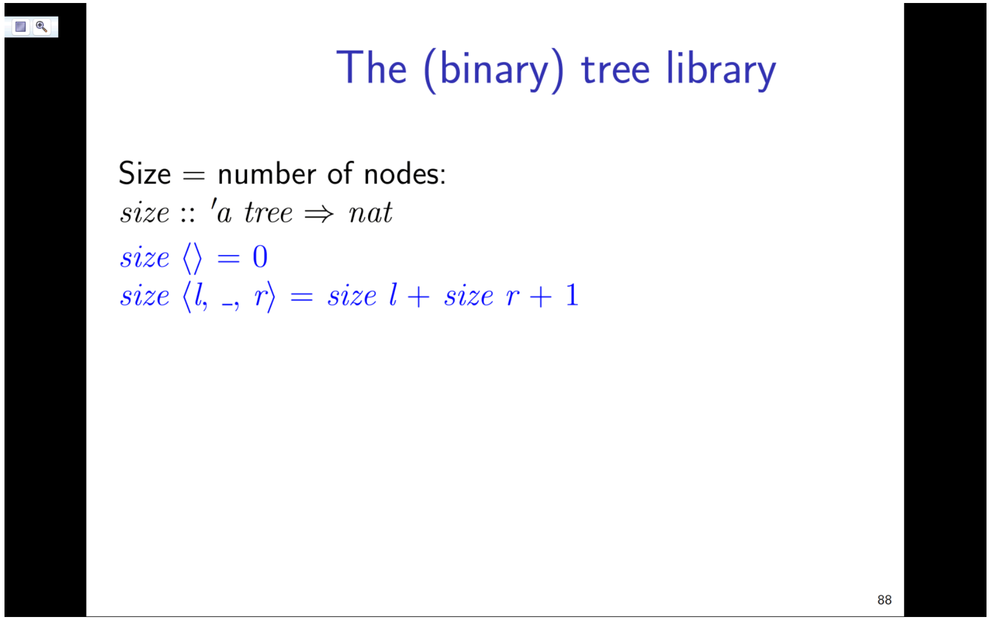
`imports "HOL-Library.Tree"`

(File: `isabelle/src/HOL/Library/Tree.thy`)

datatype $'a\ tree = Leaf \mid Node\ ('a\ tree)\ 'a\ ('a\ tree)$

Abbreviations:

$$\begin{array}{rcl} \langle\rangle & \equiv & Leaf \\ \langle l,\ a,\ r\rangle & \equiv & Node\ l\ a\ r \end{array}$$



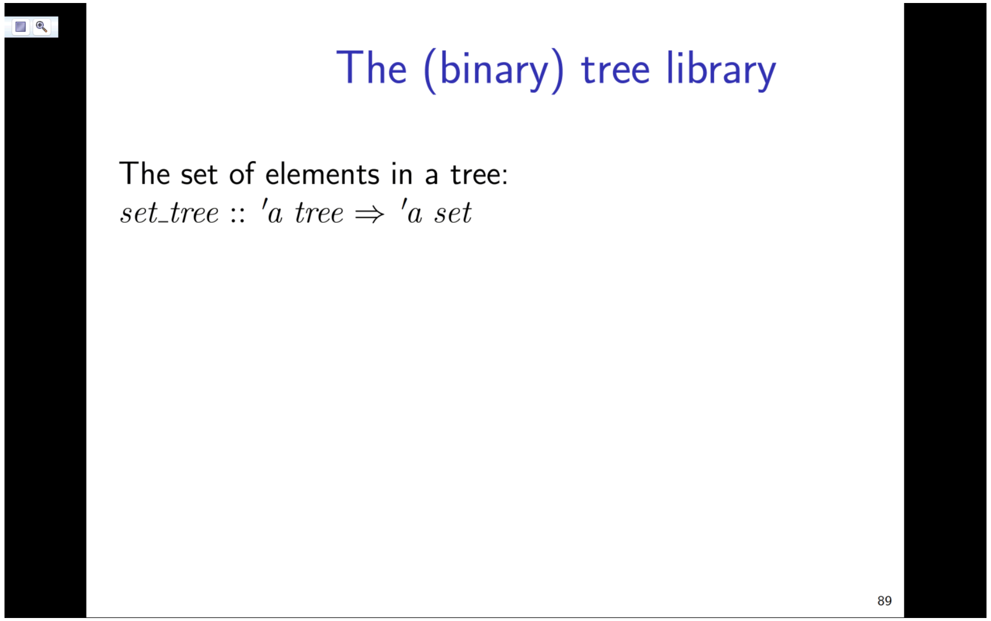
The (binary) tree library

Size = number of nodes:

$size :: 'a\ tree \Rightarrow nat$

$size\ \langle\rangle = 0$

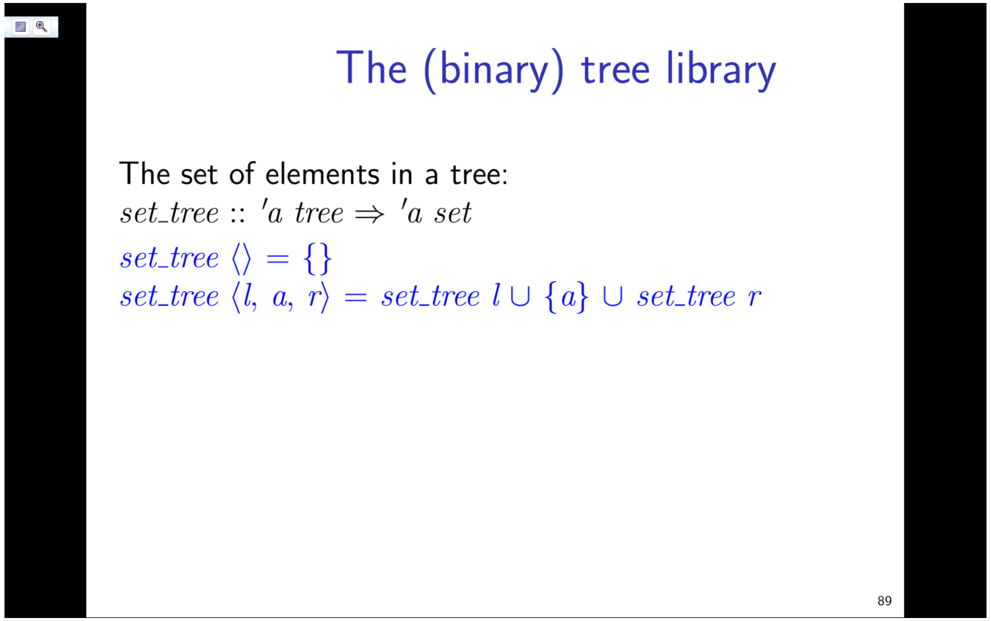
$size\ \langle l,\ \_,\ r\rangle = size\ l + size\ r + 1$



The (binary) tree library

The set of elements in a tree:

$set\_tree :: 'a\ tree \Rightarrow 'a\ set$



The (binary) tree library

The set of elements in a tree:

$set\_tree :: 'a\ tree \Rightarrow 'a\ set$

$set\_tree\ \langle\rangle = \{\}$

$set\_tree\ \langle l,\ a,\ r\rangle = set\_tree\ l \cup \{a\} \cup set\_tree\ r$

The (binary) tree library

The set of elements in a tree:
$set\_tree :: 'a\ tree \Rightarrow 'a\ set$

$set\_tree\ \langle\rangle = \{\}$
$set\_tree\ \langle l,\ a,\ r\rangle = set\_tree\ l \cup \{a\} \cup set\_tree\ r$

Inorder listing:
$inorder :: 'a\ tree \Rightarrow 'a\ list$

$inorder\ \langle\rangle = []$
$inorder\ \langle l,\ x,\ r\rangle = inorder\ l\ @\ [x]\ @\ inorder\ r$

The (binary) tree library

Binary search tree invariant:
$bst :: 'a\ tree \Rightarrow bool$

The (binary) tree library

Binary search tree invariant:
$bst :: 'a\ tree \Rightarrow bool$

$bst\ \langle\rangle = True$
$bst\ \langle l,\ a,\ r\rangle =$
$(bst\ l \wedge$
$bst\ r \wedge$
$(\forall x{\in}set\_tree\ l.\ x < a) \wedge (\forall x{\in}set\_tree\ r.\ a < x))$

The (binary) tree library

Binary search tree invariant:
$bst :: 'a\ tree \Rightarrow bool$

$bst\ \langle\rangle = True$
$bst\ \langle l,\ a,\ r\rangle =$
$(bst\ l \wedge$
$bst\ r \wedge$
$(\forall x{\in}set\_tree\ l.\ x < a) \wedge (\forall x{\in}set\_tree\ r.\ a < x))$

For any type $'a$?

Isabelle's type classes

Isabelle's type classes

A *type class* is defined by

- a set of required functions (the interface)

Isabelle's type classes

A *type class* is defined by

- a set of required functions (the interface)
- and a set of axioms about those functions

Isabelle's type classes

A *type class* is defined by

- a set of required functions (the interface)
- and a set of axioms about those functions

Example: class $\mathit{linorder}$: linear orders with $\leq$, $<$

Isabelle's type classes

A *type class* is defined by

- a set of required functions (the interface)
- and a set of axioms about those functions

Example: class $linorder$: linear orders with $\leq$, $<$

A type belongs to some class if

- the interface functions are defined on that type

Isabelle's type classes

A *type class* is defined by

- a set of required functions (the interface)
- and a set of axioms about those functions

Example: class $linorder$: linear orders with $\leq$, $<$

A type belongs to some class if

- the interface functions are defined on that type
- and satisfy the axioms of the class

Isabelle's type classes

A *type class* is defined by

- a set of required functions (the interface)
- and a set of axioms about those functions

Example: class $linorder$: linear orders with $\leq$, $<$

A type belongs to some class if

- the interface functions are defined on that type
- and satisfy the axioms of the class (proof needed!)

Isabelle's type classes

A *type class* is defined by

- a set of required functions (the interface)
- and a set of axioms about those functions

Example: class $linorder$: linear orders with $\leq$, $<$

A type belongs to some class if

- the interface functions are defined on that type
- and satisfy the axioms of the class (proof needed!)

Notation: $\tau :: C$ means type τ belongs to class C

Isabelle's type classes

A *type class* is defined by

- a set of required functions (the interface)
- and a set of axioms about those functions

Example: class $linorder$: linear orders with $\leq$, $<$

A type belongs to some class if

- the interface functions are defined on that type
- and satisfy the axioms of the class (proof needed!)

Notation: $\tau :: C$ means type τ belongs to class C

Example: $bst :: ('a :: linorder)\ tree \Rightarrow bool$

Case study

BST_Demo.thy

Case study

BST_Demo.thy

Chapter 4

Logic and Proof Beyond Equality

5 Logical Formulas

6 Proof Automation

7 Single Step Proofs

Syntax (in decreasing precedence):

$$\begin{array}{lllll} \mathit{form} & ::= & (\mathit{form}) & \mid\ \mathit{term} = \mathit{term} & \mid\ \neg \mathit{form} \\ & \mid & \mathit{form} \wedge \mathit{form} & \mid\ \mathit{form} \vee \mathit{form} & \mid\ \mathit{form} \longrightarrow \mathit{form} \\ & \mid & \forall x.\ \mathit{form} & \mid\ \exists x.\ \mathit{form} & \end{array}$$

Syntax (in decreasing precedence):

$$\begin{array}{lllll} \mathit{form} & ::= & (\mathit{form}) & \mid\ \mathit{term} = \mathit{term} & \mid\ \neg \mathit{form} \\ & \mid & \mathit{form} \wedge \mathit{form} & \mid\ \mathit{form} \vee \mathit{form} & \mid\ \mathit{form} \longrightarrow \mathit{form} \\ & \mid & \forall x.\ \mathit{form} & \mid\ \exists x.\ \mathit{form} & \end{array}$$

Examples:

$$\neg\ A \wedge B \vee C \quad \equiv \quad ((\neg\ A) \wedge B) \vee C$$

Syntax (in decreasing precedence):

$$\begin{array}{lllll} \mathit{form} & ::= & (\mathit{form}) & \mid\ \mathit{term} = \mathit{term} & \mid\ \neg \mathit{form} \\ & \mid & \mathit{form} \wedge \mathit{form} & \mid\ \mathit{form} \vee \mathit{form} & \mid\ \mathit{form} \longrightarrow \mathit{form} \\ & \mid & \forall x.\ \mathit{form} & \mid\ \exists x.\ \mathit{form} & \end{array}$$

Examples:

$$\neg\ A \wedge B \vee C \quad \equiv \quad ((\neg\ A) \wedge B) \vee C$$

$$s = t \wedge C \quad \equiv \quad (s = t) \wedge C$$

Syntax (in decreasing precedence):

$$\begin{array}{lllllll} form & ::= & (form) & \mid & term = term & \mid & \neg form \\ & \mid & form \wedge form & \mid & form \vee form & \mid & form \longrightarrow form \\ & \mid & \forall x.\ form & \mid & \exists x.\ form & & \end{array}$$

Examples:

$$\begin{array}{rcl} \neg\ A \wedge B \vee C & \equiv & ((\neg\ A) \wedge B) \vee C \\ s = t \wedge C & \equiv & (s = t) \wedge C \\ A \wedge B = B \wedge A & \equiv & A \wedge (B = B) \wedge A \end{array}$$

Syntax (in decreasing precedence):

$$\begin{array}{lllllll} form & ::= & (form) & \mid & term = term & \mid & \neg form \\ & \mid & form \wedge form & \mid & form \vee form & \mid & form \longrightarrow form \\ & \mid & \forall x.\ form & \mid & \exists x.\ form & & \end{array}$$

Examples:

$$\begin{array}{rcl} \neg\ A \wedge B \vee C & \equiv & ((\neg\ A) \wedge B) \vee C \\ s = t \wedge C & \equiv & (s = t) \wedge C \\ A \wedge B = B \wedge A & \equiv & A \wedge (B = B) \wedge A \end{array}$$

Syntax (in decreasing precedence):

$$\begin{array}{lllllll} form & ::= & (form) & \mid & term = term & \mid & \neg form \\ & \mid & form \wedge form & \mid & form \vee form & \mid & form \longrightarrow form \\ & \mid & \forall x.\ form & \mid & \exists x.\ form & & \end{array}$$

Examples:

$$\begin{array}{rcl} \neg\ A \wedge B \vee C & \equiv & ((\neg\ A) \wedge B) \vee C \\ s = t \wedge C & \equiv & (s = t) \wedge C \\ A \wedge B = B \wedge A & \equiv & A \wedge (B = B) \wedge A \\ \forall\, x.\ P\ x \wedge\ Q\ x & \equiv & \forall\, x.\ (P\ x \wedge\ Q\ x) \end{array}$$

Syntax (in decreasing precedence):

$$\begin{array}{lllllll} form & ::= & (form) & \mid & term = term & \mid & \neg form \\ & \mid & form \wedge form & \mid & form \vee form & \mid & form \longrightarrow form \\ & \mid & \forall x.\ form & \mid & \exists x.\ form & & \end{array}$$

Examples:

$$\begin{array}{rcl} \neg\ A \wedge B \vee C & \equiv & ((\neg\ A) \wedge B) \vee C \\ s = t \wedge C & \equiv & (s = t) \wedge C \\ A \wedge B = B \wedge A & \equiv & A \wedge (B = B) \wedge A \\ \forall\, x.\ P\ x \wedge\ Q\ x & \equiv & \forall\, x.\ (P\ x \wedge\ Q\ x) \end{array}$$

Input syntax: $\longleftrightarrow$ (same precedence as $\longrightarrow$)

Variable binding convention:

$$\forall\, x\ y.\ P\ x\ y\ \equiv\ \forall\, x.\ \forall\, y.\ P\ x\ y$$

Warning

Quantifiers have low precedence
and need to be parenthesized (if in some context)

$$\textbf{!}\quad P \wedge \forall\, x.\ Q\ x\ \rightsquigarrow\ P \wedge (\forall\, x.\ Q\ x)\quad \textbf{!}$$

Mathematical symbols

. . . and their ascii representations:

$\forall$	`\<forall>`	`ALL`
$\exists$	`\<exists>`	`EX`
λ	`\<lambda>`	`%`
$\longrightarrow$	`-->`	
$\longleftrightarrow$	`<->`	
$\wedge$	`/\`	`&`
$\vee$	`\/`	`|`
$\neg$	`\<not>`	`~`
$\neq$	`\<noteq>`	`~=`

Sets over type $'a$

$'a\ set$

- $\{\},\quad \{e_1,\ldots,e_n\}$

Sets over type $'a$

$'a$ set

- $\{\}, \quad \{e_1, \ldots, e_n\}$
- $e \in A, \quad A \subseteq B$
- $A \cup B, \quad A \cap B, \quad A - B, \quad - A$

Sets over type $'a$

$'a$ set

- $\{\}, \quad \{e_1, \ldots, e_n\}$
- $e \in A, \quad A \subseteq B$
- $A \cup B, \quad A \cap B, \quad A - B, \quad - A$
- $\{x.\ P\}$ where x is a variable

Sets over type $'a$

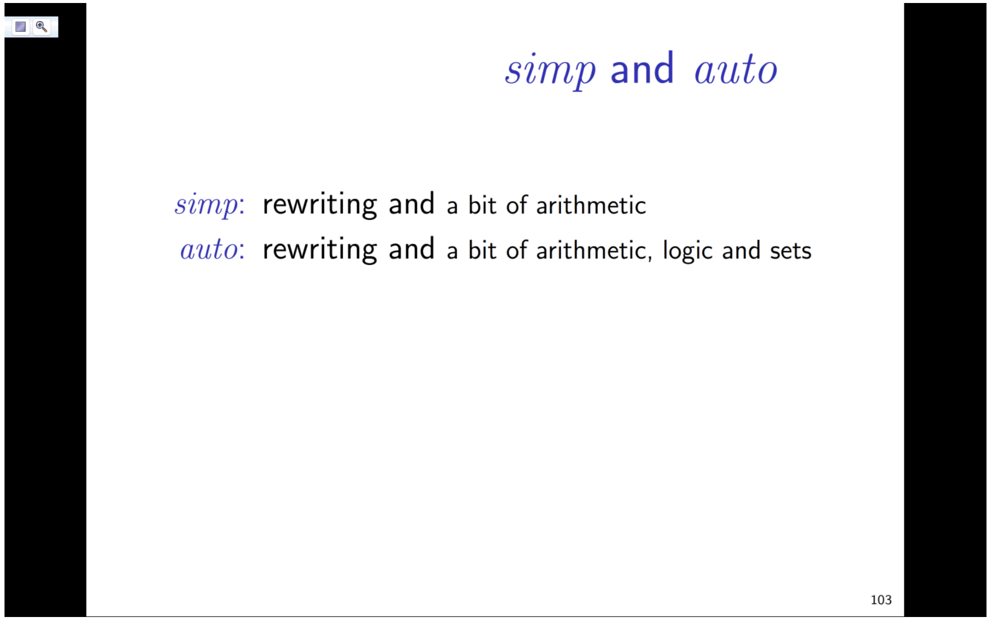
$'a$ set

- $\{\}, \quad \{e_1, \ldots, e_n\}$
- $e \in A, \quad A \subseteq B$
- $A \cup B, \quad A \cap B, \quad A - B, \quad - A$
- $\{x.\ P\}$ where x is a variable
- $\ldots$

5 Logical Formulas

6 Proof Automation

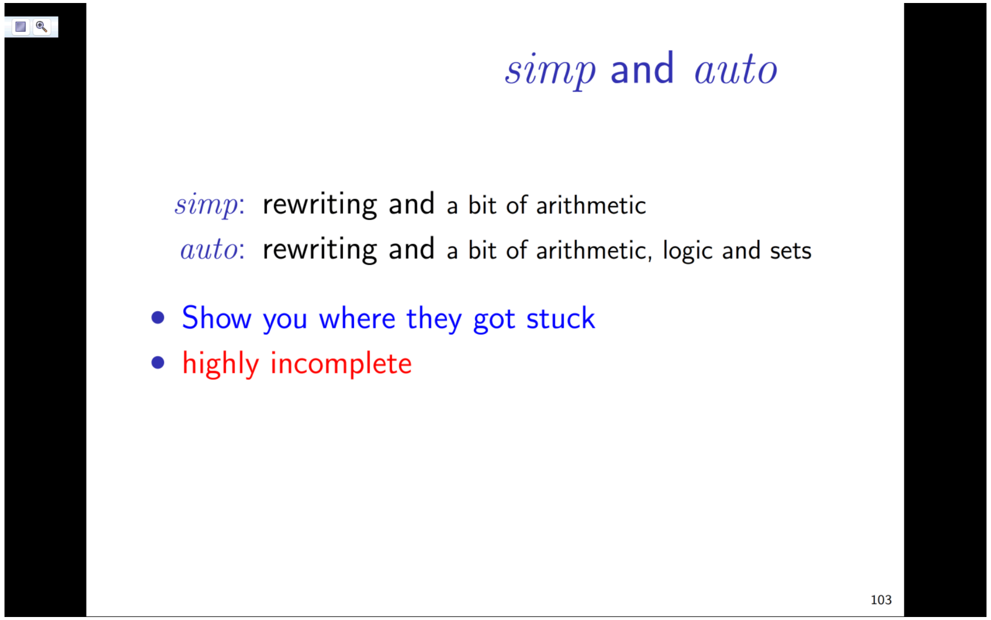
7 Single Step Proofs



simp and *auto*

simp: rewriting and a bit of arithmetic

auto: rewriting and a bit of arithmetic, logic and sets

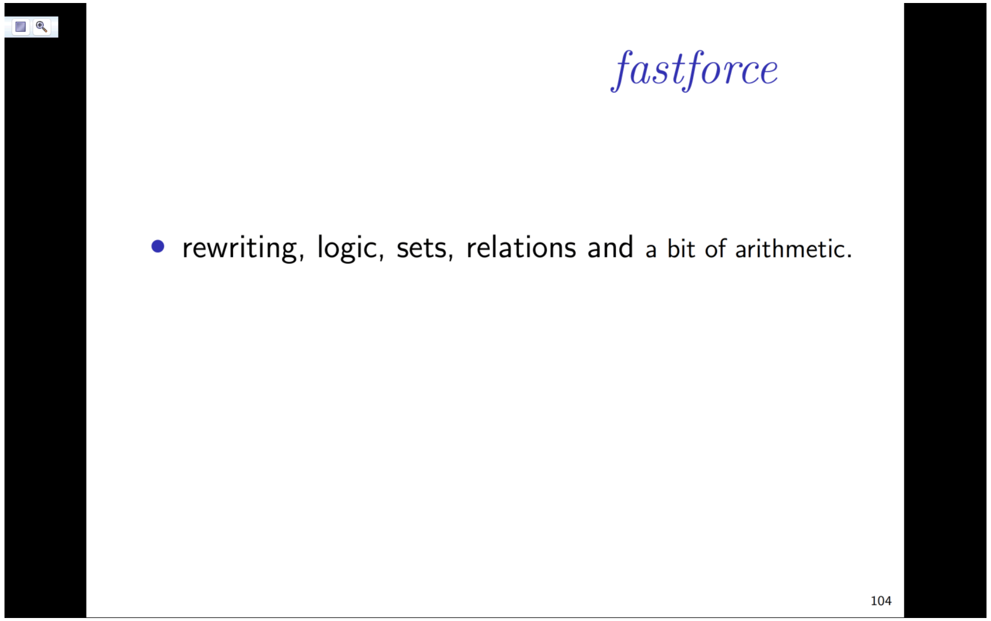


simp and *auto*

simp: rewriting and a bit of arithmetic

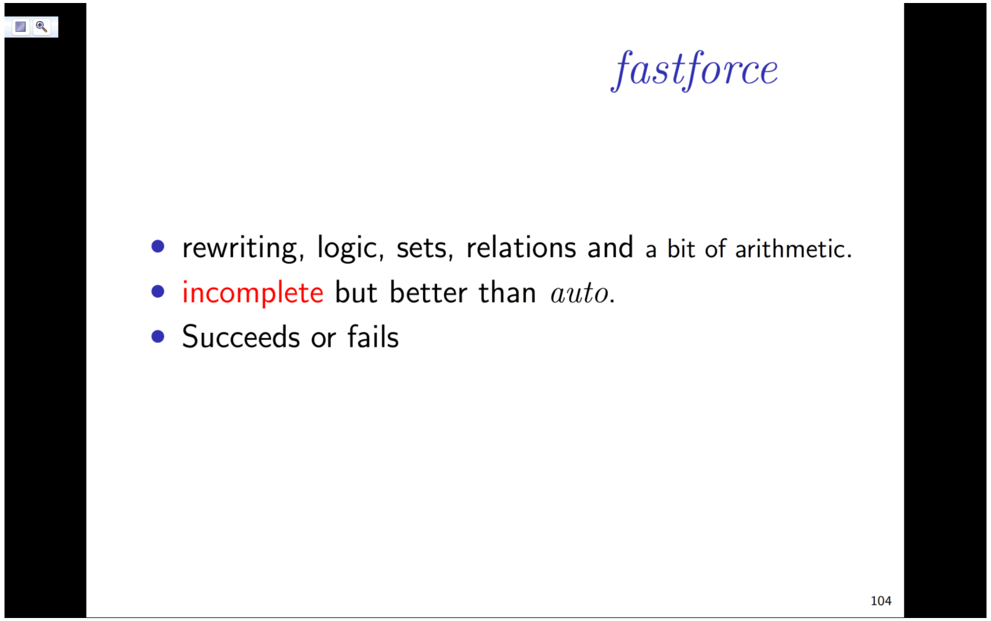
auto: rewriting and a bit of arithmetic, logic and sets

- Show you where they got stuck
- highly incomplete



fastforce

- rewriting, logic, sets, relations and a bit of arithmetic.



fastforce

- rewriting, logic, sets, relations and a bit of arithmetic.
- incomplete but better than *auto*.
- Succeeds or fails

fastforce

- rewriting, logic, sets, relations and a bit of arithmetic.
- incomplete but better than *auto*.
- Succeeds or fails
- Extensible with new *simp*-rules

blast

- A complete proof search procedure for FOL . . .
- . . . but (almost) without "="

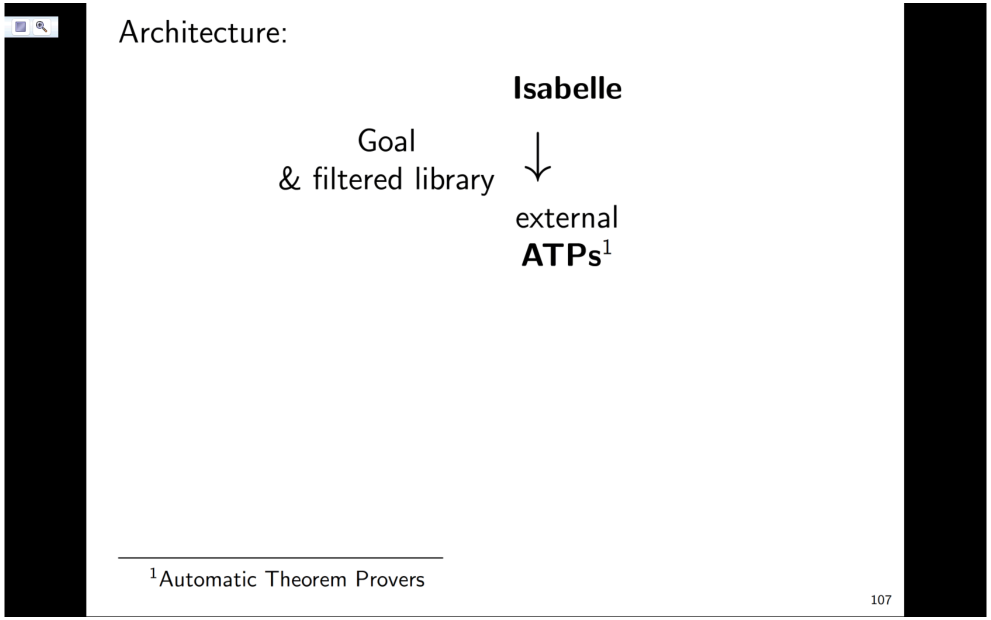
Sledgehammer

Architecture:

Isabelle

external
ATPs[1]

[1] Automatic Theorem Provers



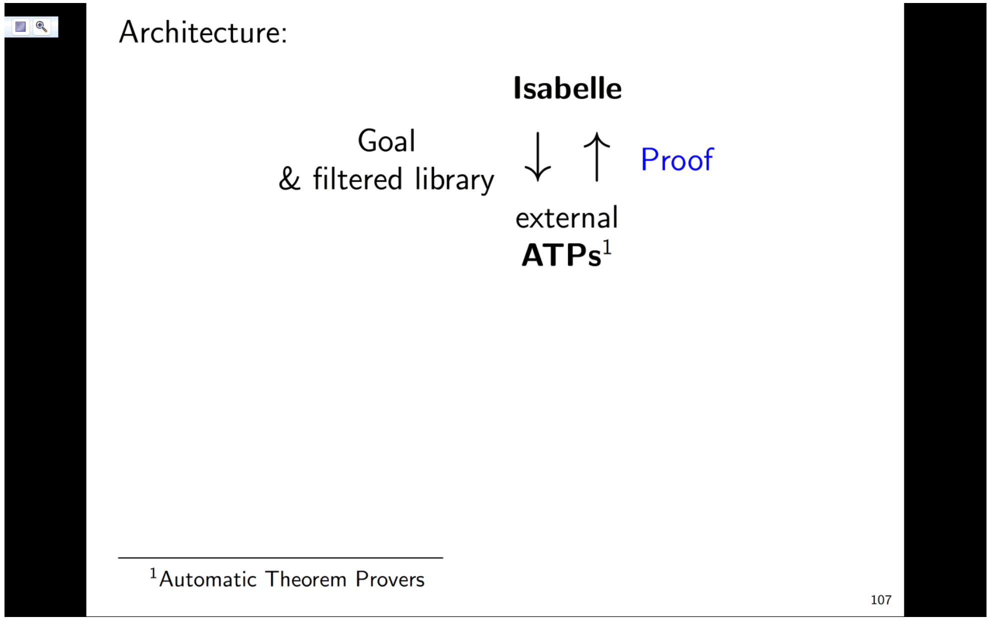
Architecture:

Isabelle

Goal
& filtered library ↓

external
ATPs[1]

[1] Automatic Theorem Provers



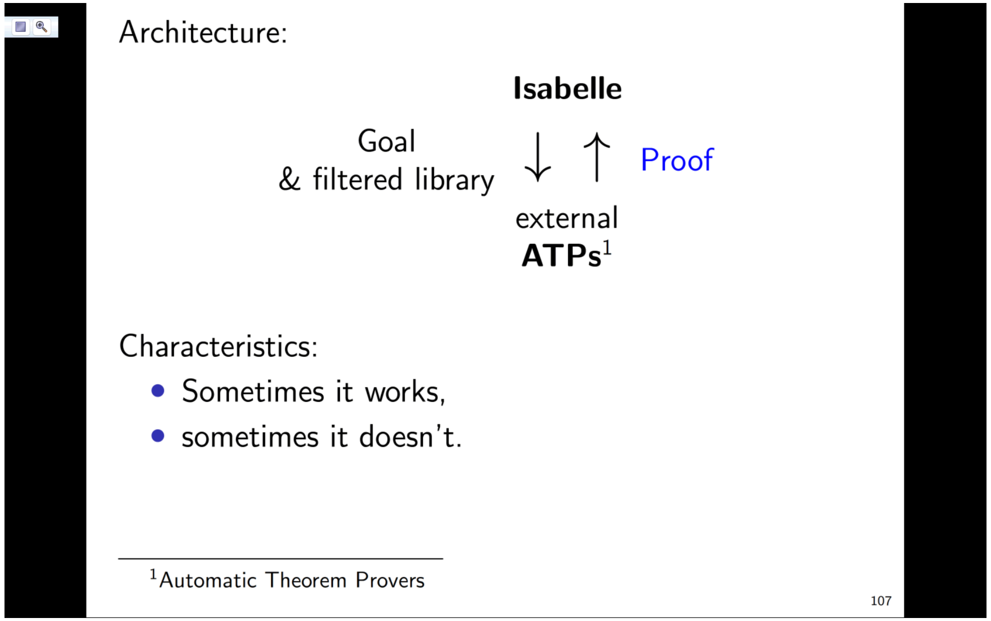
Architecture:

Isabelle

Goal
& filtered library ↓ ↑ Proof

external
ATPs[1]

[1] Automatic Theorem Provers



Architecture:

Isabelle

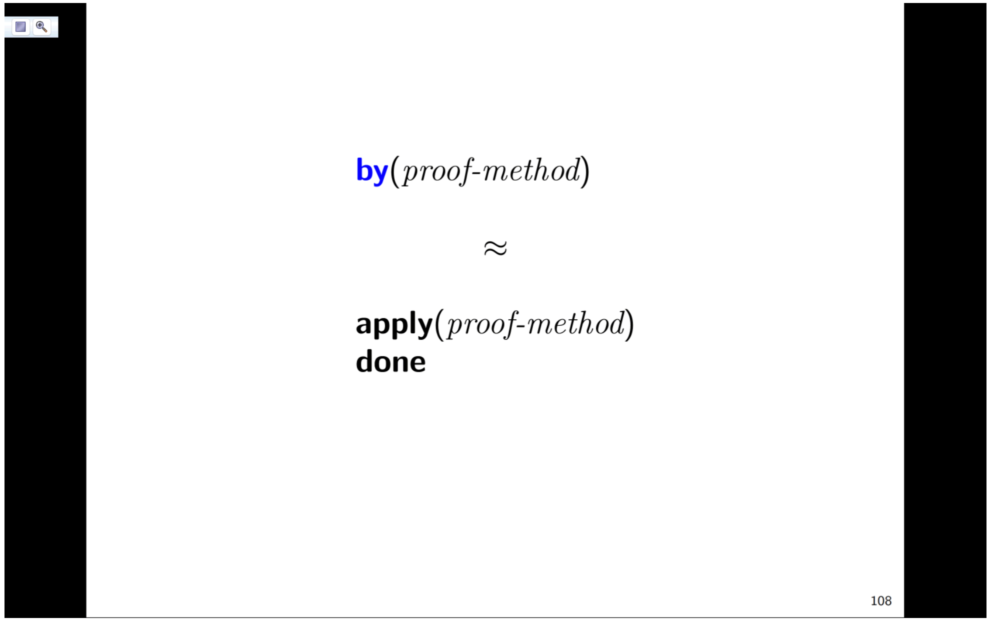
Goal
& filtered library ↓ ↑ Proof

external
ATPs[1]

Characteristics:

- Sometimes it works,
- sometimes it doesn't.

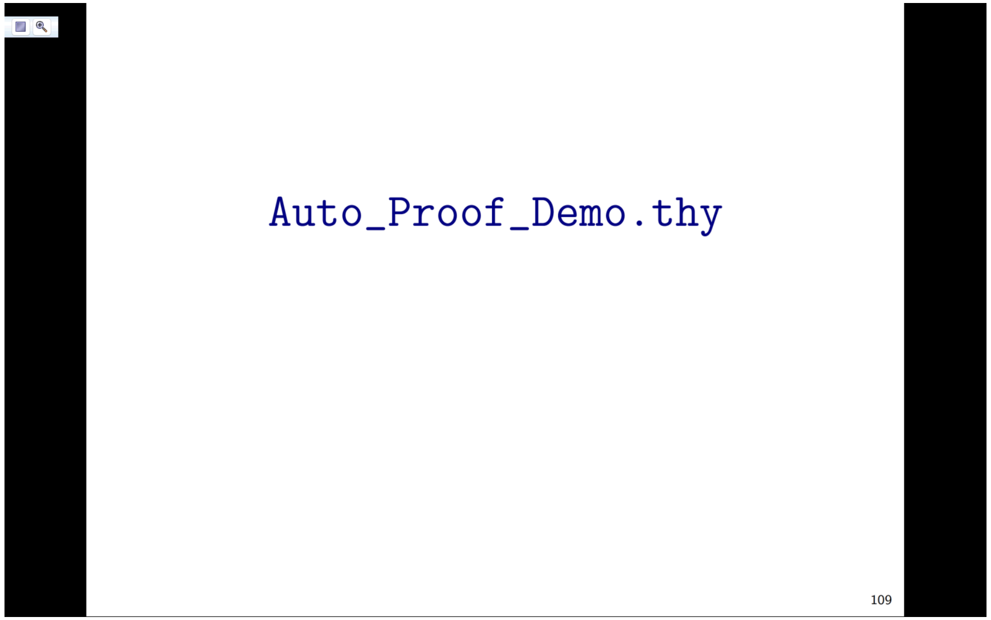
[1] Automatic Theorem Provers



by(*proof-method*)

≈

apply(*proof-method*)
done



Auto_Proof_Demo.thy