COMP4161: Advanced Topics in Software Verification

lsar

DATA

Gerwin Klein, June Andronick, Ramana Kumar, Miki Tanaka S2/2017

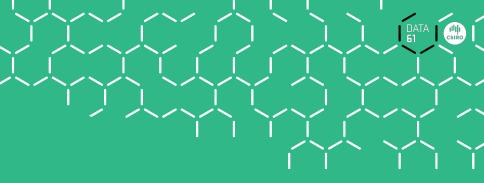


data61.csiro.au

Content

ment	DATA 61
→ Intro & motivation, getting started	[1]
 → Foundations & Principles Lambda Calculus, natural deduction 	[1 2]
 Higher Order Logic Term rewriting 	[1,2] [3 ^a] [4]
➔ Proof & Specification Techniques	
 Inductively defined sets, rule induction 	[5]
 Datatypes, recursion, induction 	[6, 7]
 Hoare logic, proofs about programs, C verification (mid-semester break) 	[8 ^b ,9]
Writing Automated Proof Methods	[10]
 Isar, codegen, typeclasses, locales 	[11 ^c ,12]

^aa1 due; ^ba2 due; ^ca3 due



Isar

A Language for Structured Proofs



Is this true: $(A \longrightarrow B) = (B \lor \neg A)$?

4 | COMP4161 | © Data61, CSIRO: provided under Creative Commons Attribution License



Is this true: $(A \longrightarrow B) = (B \lor \neg A)$?

YES!

apply (rule iffI) apply (cases A) apply (rule disjI1) apply (erule impE) apply assumption apply assumption apply (rule disjI2) apply (rule impI) apply (erule disjE) apply (erule notE) apply assumption done



Is this true: $(A \longrightarrow B) = (B \lor \neg A)$?

YES!

apply (rule iffI) apply (cases A) apply (rule disjI1) apply (erule impE) apply assumption apply assumption apply (rule disjI2) apply (rule impI) apply (erule disjE) apply (erule notE) apply assumption done

or by blast



```
Is this true: (A \longrightarrow B) = (B \lor \neg A)?
YES!
```

```
apply (rule iff1)
apply (cases A)
apply (rule disj11)
apply (erule impE)
apply assumption
apply assumption
apply (rule disj12)
apply assumption
apply (erule disj2)
apply assumption
apply (erule notE)
apply assumption
done
```

OK it's true. But WHY?

or by blast



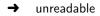
WHY is this true: $(A \longrightarrow B) = (B \lor \neg A)$?

Demo

Isar



apply scripts





apply scripts

- → unreadable
- → hard to maintain



apply scripts

- → unreadable
- → hard to maintain
- → do not scale



apply scripts

- → unreadable
- → hard to maintain
- → do not scale

No structure.



apply scripts

What about..

Elegance?

→

- → unreadable
- → hard to maintain
- \rightarrow do not scale

No structure.



apply scripts

- → unreadable
- → hard to maintain
- → do not scale

No structure.

What about..

- → Elegance?
- → Explaining deeper insights?



apply scripts

- → unreadable
- → hard to maintain
- → do not scale

No structure.

What about..

- → Elegance?
- → Explaining deeper insights?
 - → Large developments?



apply scripts

- → unreadable
- → hard to maintain
- → do not scale

What about..

- → Elegance?
- → Explaining deeper insights?
 - → Large developments?

No structure.

Isar!

A typical Isar proof



proof
 assume formula0
 have formula by simp
 ...
 have formula by blast
 show formulan+1 by ...
qed

A typical Isar proof



proof
 assume formula0
 have formula by simp
 :
 have formula by blast
 show formulan+1 by ...
qed

proves $formula_0 \implies formula_{n+1}$

A typical Isar proof



proof assume formula₀ have formula₁ by simp : have formula_n by blast show formula_{n+1} by ... qed

proves $formula_0 \implies formula_{n+1}$

(analogous to assumes/shows in lemma statements)



 $\begin{array}{l} \mathsf{proof} = \textbf{proof} \; [\mathsf{method}] \; \mathsf{statement}^* \; \textbf{qed} \\ | \; \; \textbf{by} \; \mathsf{method} \end{array}$



 $\begin{array}{l} \mathsf{proof} = \textbf{proof} \; [\mathsf{method}] \; \mathsf{statement}^* \; \textbf{qed} \\ | \; \; \textbf{by} \; \mathsf{method} \end{array}$

 $\mathsf{method} = (\mathsf{simp} \dots) \mid (\mathsf{blast} \dots) \mid (\mathsf{rule} \dots) \mid \dots$



 $\mathsf{method} = (\mathsf{simp} \dots) \mid (\mathsf{blast} \dots) \mid (\mathsf{rule} \dots) \mid \dots$



 $\mathsf{method} = (\mathsf{simp} \dots) \mid (\mathsf{blast} \dots) \mid (\mathsf{rule} \dots) \mid \dots$

proposition = [name:] formula



proof [method] statement* qed

lemma " $\llbracket A; B \rrbracket \implies A \land B$ "



proof [method] statement* qed

lemma " $\llbracket A; B \rrbracket \Longrightarrow A \land B$ " proof (rule conjl)



```
lemma "[\![A; B]\!] \Longrightarrow A \land B"
proof (rule conjl)
assume A: "A"
from A show "A" by assumption
```



```
lemma "\llbracket A; B \rrbracket \implies A \land B"

proof (rule conjl)

assume A: "A"

from A show "A" by assumption

next
```



```
lemma "[\![A; B]\!] \implies A \land B"

proof (rule conjl)

assume A: "A"

from A show "A" by assumption

next

assume B: "B"

from B show "B" by assumption
```



```
lemma "\llbracket A; B \rrbracket \implies A \land B"

proof (rule conjl)

assume A: "A"

from A show "A" by assumption

next

assume B: "B"

from B show "B" by assumption

ged
```



proof [method] statement* qed

```
lemma " [\![A; B]\!] \implies A \land B"

proof (rule conjl)

assume A: "A"

from A show "A" by assumption

next

assume B: "B"

from B show "B" by assumption

ged
```

→ proof (<method>) applies method to the stated goal



```
lemma " [\![A; B]\!] \implies A \land B"

proof (rule conjl)

assume A: " A"

from A show " A" by assumption

next

assume B: " B"

from B show " B" by assumption

ged
```

- → proof (<method>) applies method to the stated goal
- → proof applies a single rule that fits



proof [method] statement* qed

```
lemma "[\![A; B]\!] \implies A \land B"

proof (rule conjl)

assume A: "A"

from A show "A" by assumption

next

assume B: "B"

from B show "B" by assumption

ged
```

→ proof (<method>) applies method to the stated goal
 → proof - applies a single rule that fits
 → proof - does nothing to the goal



Look at the proof state!

lemma " $\llbracket A; B \rrbracket \Longrightarrow A \land B$ " proof (rule conjl)



Look at the proof state!

lemma " $\llbracket A; B \rrbracket \Longrightarrow A \land B$ " proof (rule conjl)

- → proof (rule conjl) changes proof state to
 - 1. $\llbracket A; B \rrbracket \Longrightarrow A$
 - 2. $\llbracket A; B \rrbracket \Longrightarrow B$



Look at the proof state!

lemma " $\llbracket A; B \rrbracket \Longrightarrow A \land B$ " proof (rule conjl)

→ proof (rule conjl) changes proof state to 1. $[A; B] \Longrightarrow A$ 2. $[A; B] \Longrightarrow B$

 \rightarrow so we need 2 shows: **show** "*A*" and **show** "*B*"



Look at the proof state!

lemma " $\llbracket A; B \rrbracket \Longrightarrow A \land B$ " proof (rule conjl)

- → proof (rule conjl) changes proof state to
 - 1. $\llbracket A; B \rrbracket \Longrightarrow A$
 - 2. $\llbracket A; B \rrbracket \Longrightarrow B$
- → so we need 2 shows: **show** "A" and **show** "B"
- → We are allowed to assume A, because A is in the assumptions of the proof state.



→ [prove]:

goal has been stated, proof needs to follow.



→ [prove]:

goal has been stated, proof needs to follow.

→ [state]:

proof block has opened or subgoal has been proved, new *from* statement, goal statement or assumptions can follow.



→ [prove]:

goal has been stated, proof needs to follow.

→ [state]:

proof block has opened or subgoal has been proved, new *from* statement, goal statement or assumptions can follow.

→ [chain]:



→ [prove]:

goal has been stated, proof needs to follow.

→ [state]:

proof block has opened or subgoal has been proved, new *from* statement, goal statement or assumptions can follow.

→ [chain]:

from statement has been made, goal statement needs to follow.

lemma " $\llbracket A; B \rrbracket \Longrightarrow A \land B$ "



→ [prove]:

goal has been stated, proof needs to follow.

→ [state]:

proof block has opened or subgoal has been proved, new *from* statement, goal statement or assumptions can follow.

→ [chain]:

from statement has been made, goal statement needs to follow.

lemma " $\llbracket A; B \rrbracket \implies A \land B$ " [prove]



→ [prove]:

goal has been stated, proof needs to follow.

→ [state]:

proof block has opened or subgoal has been proved, new *from* statement, goal statement or assumptions can follow.

→ [chain]:

```
lemma "\llbracket A; B \rrbracket \implies A \land B" [prove]
proof (rule conjl) [state]
```



→ [prove]:

goal has been stated, proof needs to follow.

→ [state]:

proof block has opened or subgoal has been proved, new *from* statement, goal statement or assumptions can follow.

→ [chain]:

```
lemma "\llbracket A; B \rrbracket \implies A \land B" [prove]
proof (rule conjl) [state]
assume A: "A" [state]
```



→ [prove]:

goal has been stated, proof needs to follow.

→ [state]:

proof block has opened or subgoal has been proved, new *from* statement, goal statement or assumptions can follow.

→ [chain]:

```
\begin{array}{l} \textbf{lemma "} \llbracket A; B \rrbracket \Longrightarrow A \land B" \ \textbf{[prove]} \\ \textbf{proof (rule conjl) [state]} \\ \textbf{assume } A: "A" \ \textbf{[state]} \\ \textbf{from } A \ \textbf{[chain]} \end{array}
```



→ [prove]:

goal has been stated, proof needs to follow.

→ [state]:

proof block has opened or subgoal has been proved, new *from* statement, goal statement or assumptions can follow.

→ [chain]:

```
\begin{array}{l} \textbf{lemma "} \llbracket A; B \rrbracket \Longrightarrow A \land B" \ \textbf{[prove]} \\ \textbf{proof (rule conjl) [state]} \\ \textbf{assume } A: "A" \ \textbf{[state]} \\ \textbf{from } A \ \textbf{[chain] show "}A" \ \textbf{[prove] by assumption [state]} \\ \textbf{next [state] } \dots \end{array}
```

Have



Can be used to make intermediate steps.

Example:

Have



Can be used to make intermediate steps.

Example:

lemma "(x :: nat) + 1 = 1 + x"

Have



Can be used to make intermediate steps.

Example:

```
lemma "(x :: nat) + 1 = 1 + x"

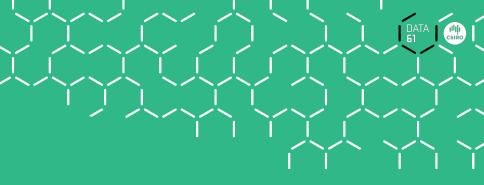
proof -

have A: "x + 1 = Suc x" by simp

have B: "1 + x = Suc x" by simp

show "x + 1 = 1 + x" by (simp only: A B)

ged
```





DATA 61

Backward reasoning: ... have " $A \land B$ " proof

DATA 61

Backward reasoning: ... have " $A \land B$ " proof

→ proof picks an intro rule automatically

Backward reasoning: ... have " $A \wedge B$ " proof

- → proof picks an intro rule automatically
- → conclusion of rule must unify with $A \land B$

Backward reasoning: ... have " $A \wedge B$ " proof

- → proof picks an intro rule automatically
- → conclusion of rule must unify with $A \land B$

Forward reasoning: ...

assume AB: " $A \land B$ " from AB have "..." proof



Backward reasoning: ... have " $A \wedge B$ " proof

- → proof picks an intro rule automatically
- → conclusion of rule must unify with $A \land B$

Forward reasoning: ... assume AB: " $A \land B$ " from AB have "..." proof

→ now **proof** picks an **elim** rule automatically



Backward reasoning: ... have " $A \wedge B$ " proof

- → proof picks an intro rule automatically
- → conclusion of rule must unify with $A \land B$

Forward reasoning: ...

assume AB: " $A \land B$ " from AB have "..." proof

- → now **proof** picks an **elim** rule automatically
- → triggered by **from**



Backward reasoning: ... have " $A \wedge B$ " proof

- → proof picks an intro rule automatically
- → conclusion of rule must unify with $A \land B$

Forward reasoning: ...

assume AB: " $A \land B$ " from AB have "..." proof

- → now **proof** picks an **elim** rule automatically
- → triggered by **from**
- ➔ first assumption of rule must unify with AB



Backward reasoning: ... have " $A \wedge B$ " proof

- → proof picks an intro rule automatically
- → conclusion of rule must unify with $A \land B$

Forward reasoning: ...

assume AB: " $A \land B$ " from AB have "..." proof

- → now proof picks an elim rule automatically
- → triggered by **from**
- \rightarrow first assumption of rule must unify with AB

General case: from $A_1 \ldots A_n$ have R proof

- → first *n* assumptions of rule must unify with $A_1 \ldots A_n$
- → conclusion of rule must unify with R





fix $v_1 \ldots v_n$



fix $v_1 \ldots v_n$

Introduces new arbitrary but fixed variables $(\sim \text{ parameters}, \wedge)$



fix $v_1 \ldots v_n$

Introduces new arbitrary but fixed variables $(\sim \text{ parameters}, \Lambda)$

obtain $v_1 \dots v_n$ where < prop > < proof >

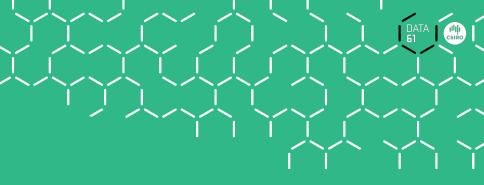


fix $v_1 \ldots v_n$

Introduces new arbitrary but fixed variables $(\sim \text{ parameters}, \Lambda)$

obtain $v_1 \dots v_n$ where < prop > < proof >

Introduces new variables together with property







this = the previous fact proved or assumed



- this = the previous fact proved or assumed
- **then** = **from** this



- this $\ = \$ the previous fact proved or assumed
- then = from this
- thus = then show



- this = the previous fact proved or assumed
- then = from this
- thus = then show
- hence = then have



this = the previous fact proved or assumed

then	=	from this
thus	=	then show
hence	=	then have
with $A_1 \ldots A_n$	=	from $A_1 \ldots A_n$ this



this $=$	the previous	fact proved	or assumed
----------	--------------	-------------	------------

then	=	from this
thus	=	then show
hence	=	then have
with $A_1 \ldots A_n$	=	from $A_1 \ldots A_n$ this

?thesis = the last enclosing goal statement

Moreover and Ultimately



have $X_1: P_1 \dots$ have $X_2: P_2 \dots$: have $X_n: P_n \dots$ from $X_1 \dots X_n$ show \dots

Moreover and Ultimately



have $X_1: P_1 \dots$ have $X_2: P_2 \dots$: have $X_n: P_n \dots$ from $X_1 \dots X_n$ show \dots

wastes lots of brain power on names $X_1 \dots X_n$

Moreover and Ultimately



have $X_1: P_1 ...$ have $X_2: P_2 ...$

have X_n : $P_n \ldots$ from $X_1 \ldots X_n$ show \ldots have $P_1 \dots$ moreover have $P_2 \dots$

moreover have $P_n \dots$ ultimately show ...

wastes lots of brain power on names $X_1 \dots X_n$

General Case Distinctions



show formula proof -



show formula proof - have $P_1 \lor P_2 \lor P_3$ <proof>





show formula proof have $P_1 \lor P_2 \lor P_3$ <proof> moreover { assume P_1 ... have ?thesis <proof> } moreover { assume P_2 ... have ?thesis <proof> }



show formula proof have $P_1 \lor P_2 \lor P_3$ <proof> moreover { assume P_1 ... have ?thesis <proof> } moreover { assume P_2 ... have ?thesis <proof> } moreover { assume P_3 ... have ?thesis <proof> }





```
show formula

proof -

have P_1 \lor P_2 \lor P_3 <proof>

moreover { assume P_1 ... have ?thesis <proof> }

moreover { assume P_2 ... have ?thesis <proof> }

moreover { assume P_3 ... have ?thesis <proof> }

ultimately show ?thesis by blast

qed

{ ... } is a proof block similar to proof ... qed
```

```
| COMP4161 | © Data61, CSIRO: provided under Creative Commons Attribution License
```



```
show formula

proof -

have P_1 \lor P_2 \lor P_3 <proof>

moreover { assume P_1 ... have ?thesis <proof> }

moreover { assume P_2 ... have ?thesis <proof> }

moreover { assume P_3 ... have ?thesis <proof> }

ultimately show ?thesis by blast

qed

{ ... } is a proof block similar to proof ... qed
```

```
\{ \text{ assume } P_1 \dots \text{ have } P < proof > \} \\ \text{ stands for } P_1 \Longrightarrow P
```

Mixing proof styles



```
from ...

have ...

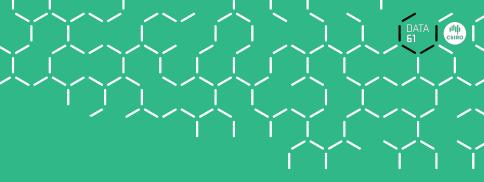
apply - make incoming facts assumptions

apply (...)

:

apply (...)

done
```



Datatypes in Isar

Datatype case distinction



```
proof (cases term)

case Constructor<sub>1</sub>

:

next

:

next

case (Constructor<sub>k</sub> \vec{x})

...\vec{x} ...

qed
```

Datatype case distinction



```
proof (cases term)

case Constructor<sub>1</sub>

:

next

:

next

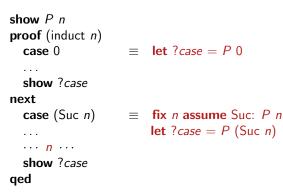
case (Constructor<sub>k</sub> \vec{x})

...\vec{x} ...

qed
```

case (Constructor_{*i*} \vec{x}) \equiv **fix** \vec{x} **assume** Constructor_{*i*} : "*term* = Constructor_{*i*} \vec{x} "

Structural induction for nat





Structural induction: \implies and \bigwedge



```
show "\bigwedge x. A n \Longrightarrow P n"

proof (induct n)

case 0

...

show ?case

next

case (Suc n)

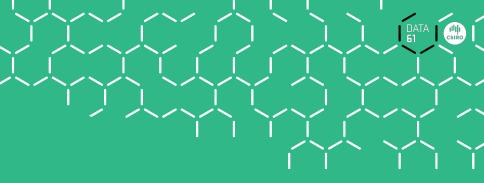
...

show ?case

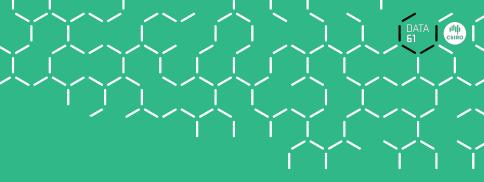
qed
```

= fix x assume 0: "A 0"let ?case = "P 0"

```
 = fix n and x 
assume Suc: " <math>\land x. A n \Longrightarrow P n"
" A (Suc n)"
let ?case = " P (Suc n)"
```



Demo: Datatypes in Isar



Calculational Reasoning

Prove:
$$x \cdot x^{-1} = 1$$



using: assoc: $(x \cdot y) \cdot z = x \cdot (y \cdot z)$ left_inv: $x^{-1} \cdot x = 1$ left_one: $1 \cdot x = x$

Prove:

$$x \cdot x^{-1} = 1 \cdot (x \cdot x^{-1})$$

 $\dots = 1 \cdot x \cdot x^{-1}$
 $\dots = (x^{-1})^{-1} \cdot x^{-1} \cdot x \cdot x^{-1}$
 $\dots = (x^{-1})^{-1} \cdot (x^{-1} \cdot x) \cdot x^{-1}$
 $\dots = (x^{-1})^{-1} \cdot 1 \cdot x^{-1}$
 $\dots = (x^{-1})^{-1} \cdot (1 \cdot x^{-1})$
 $\dots = (x^{-1})^{-1} \cdot x^{-1}$



$$\begin{array}{ll} \text{assoc:} & (x \cdot y) \cdot z = x \cdot (y \cdot z) \\ \text{left_inv:} & x^{-1} \cdot x = 1 \\ \text{left_one:} & 1 \cdot x = x \end{array}$$

Prove:

$$x \cdot x^{-1} = 1 \cdot (x \cdot x^{-1})$$

 $\dots = 1 \cdot x \cdot x^{-1}$
 $\dots = (x^{-1})^{-1} \cdot x^{-1} \cdot x \cdot x^{-1}$
 $\dots = (x^{-1})^{-1} \cdot (x^{-1} \cdot x) \cdot x^{-1}$
 $\dots = (x^{-1})^{-1} \cdot 1 \cdot x^{-1}$
 $\dots = (x^{-1})^{-1} \cdot (1 \cdot x^{-1})$
 $\dots = 1$



$$\begin{array}{ll} \text{assoc:} & (x \cdot y) \cdot z = x \cdot (y \cdot z) \\ \text{left_inv:} & x^{-1} \cdot x = 1 \\ \text{left_one:} & 1 \cdot x = x \end{array}$$

Can we do this in Isabelle?

Prove:

$$x \cdot x^{-1} = 1 \cdot (x \cdot x^{-1})$$

 $\dots = 1 \cdot x \cdot x^{-1}$
 $\dots = (x^{-1})^{-1} \cdot x^{-1} \cdot x \cdot x^{-1}$
 $\dots = (x^{-1})^{-1} \cdot (x^{-1} \cdot x) \cdot x^{-1}$
 $\dots = (x^{-1})^{-1} \cdot 1 \cdot x^{-1}$
 $\dots = (x^{-1})^{-1} \cdot (1 \cdot x^{-1})$
 $\dots = 1$



assoc:
$$(x \cdot y) \cdot z = x \cdot (y \cdot z)$$

left_inv: $x^{-1} \cdot x = 1$
left_one: $1 \cdot x = x$

Can we do this in Isabelle?

→ Simplifier: too eager

Prove:

$$x \cdot x^{-1} = 1 \cdot (x \cdot x^{-1})$$

 $\dots = 1 \cdot x \cdot x^{-1}$
 $\dots = (x^{-1})^{-1} \cdot x^{-1} \cdot x \cdot x^{-1}$
 $\dots = (x^{-1})^{-1} \cdot (x^{-1} \cdot x) \cdot x^{-1}$
 $\dots = (x^{-1})^{-1} \cdot 1 \cdot x^{-1}$
 $\dots = (x^{-1})^{-1} \cdot (1 \cdot x^{-1})$
 $\dots = 1$

$$\begin{array}{ll} \text{assoc:} & (x \cdot y) \cdot z = x \cdot (y \cdot z) \\ \text{left_inv:} & x^{-1} \cdot x = 1 \\ \text{left_one:} & 1 \cdot x = x \end{array}$$

Can we do this in Isabelle?

- → Simplifier: too eager
- ➔ Manual: difficult in apply style

Prove:

$$x \cdot x^{-1} = 1 \cdot (x \cdot x^{-1})$$

 $\dots = 1 \cdot x \cdot x^{-1}$
 $\dots = (x^{-1})^{-1} \cdot x^{-1} \cdot x \cdot x^{-1}$
 $\dots = (x^{-1})^{-1} \cdot (x^{-1} \cdot x) \cdot x^{-1}$
 $\dots = (x^{-1})^{-1} \cdot 1 \cdot x^{-1}$
 $\dots = (x^{-1})^{-1} \cdot (1 \cdot x^{-1})$
 $\dots = 1$

assoc:
$$(x \cdot y) \cdot z = x \cdot (y \cdot z)$$

left_inv: $x^{-1} \cdot x = 1$
left_one: $1 \cdot x = x$

Can we do this in Isabelle?

- → Simplifier: too eager
- ➔ Manual: difficult in apply style
- ➔ Isar: with the methods we know, too verbose



The Problem

a = b $\dots = c$ $\dots = d$ shows a = d by transitivity of =



The Problem

a = b $\dots = c$ $\dots = d$ shows a = d by transitivity of =

Each step usually nontrivial (requires own subproof)



The Problem

$$a = b$$

... = c
... = d
shows $a = d$ by transitivity of =

Each step usually nontrivial (requires own subproof) Solution in Isar:

→ Keywords also and finally to delimit steps



The Problem

$$a = b$$

... = c
... = d
shows $a = d$ by transitivity of =

Each step usually nontrivial (requires own subproof) Solution in Isar:

- → Keywords also and finally to delimit steps
- → ...: predefined schematic term variable, refers to right hand side of last expression



The Problem

$$a = b$$

... = c
... = d
shows $a = d$ by transitivity of =

Each step usually nontrivial (requires own subproof) Solution in Isar:

- → Keywords also and finally to delimit steps
- → ...: predefined schematic term variable, refers to right hand side of last expression
- ➔ Automatic use of transitivity rules to connect steps

have " $t_0 = t_1$ " [proof] also



have " $t_0 = t_1$ " [proof] also



have " $t_0 = t_1$ " [proof] also have " $\ldots = t_2$ " [proof]



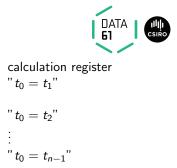
have " $t_0 = t_1$ " [proof] also have " $\ldots = t_2$ " [proof] also



$$"t_0 = t_2"$$

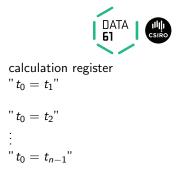
have "
$$t_0 = t_1$$
" [proof]
also
have " $\ldots = t_2$ " [proof]
also

also



have "
$$t_0 = t_1$$
" [proof]
also
have " $\ldots = t_2$ " [proof]
also

also have " $\cdots = t_n$ " [proof]



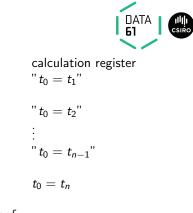
have "
$$t_0 = t_1$$
" [proof]
also
have " $\ldots = t_2$ " [proof]
also

also

have " $\cdots = t_n$ " [proof] finally

calculation register
"
$$t_0 = t_1$$
"
" $t_0 = t_2$ "
:
" $t_0 = t_{n-1}$ "
 $t_0 = t_n$

have " $t_0 = t_1$ " [proof]calculaalso" $t_0 =$ have " $\ldots = t_2$ " [proof]" $t_0 =$ also" $t_0 =$ \vdots \vdots also" $t_0 =$ have " $\cdots = t_n$ " [proof] $t_0 = t_0$ finally $t_0 = t_0$ show P- 'finally' pipes fact " $t_0 = t_n$ " into the proof



More about also



→ Works for all combinations of =, \leq and <.

More about also



- → Works for all combinations of =, \leq and <.
- → Uses all rules declared as [trans].

More about also



- → Works for all combinations of =, \leq and <.
- → Uses all rules declared as [trans].
- ➔ To view all combinations: print_trans_rules



have = " $l_1 \odot r_1$ " [proof] also have "... $\odot r_2$ " [proof] also



have = " $l_1 \odot r_1$ " [proof] also have "... $\odot r_2$ " [proof] also

Anatomy of a [trans] rule:

→ Usual form: plain transitivity $\llbracket l_1 \odot r_1; r_1 \odot r_2 \rrbracket \Longrightarrow l_1 \odot r_2$



have = " $l_1 \odot r_1$ " [proof] also have "... $\odot r_2$ " [proof] also

Anatomy of a [trans] rule:

- → Usual form: plain transitivity $\llbracket l_1 \odot r_1; r_1 \odot r_2 \rrbracket \Longrightarrow l_1 \odot r_2$
- → More general form: $\llbracket P \ l_1 \ r_1; Q \ r_1 \ r_2; A \rrbracket \Longrightarrow C \ l_1 \ r_2$



have = " $l_1 \odot r_1$ " [proof] also have "... $\odot r_2$ " [proof] also

Anatomy of a [trans] rule:

- → Usual form: plain transitivity $\llbracket l_1 \odot r_1; r_1 \odot r_2 \rrbracket \Longrightarrow l_1 \odot r_2$
- → More general form: $\llbracket P \ l_1 \ r_1; Q \ r_1 \ r_2; A \rrbracket \Longrightarrow C \ l_1 \ r_2$

Examples:

→ pure transitivity: $\llbracket a = b; b = c \rrbracket \implies a = c$



have = " $l_1 \odot r_1$ " [proof] also have "... $\odot r_2$ " [proof] also

Anatomy of a [trans] rule:

- → Usual form: plain transitivity $\llbracket l_1 \odot r_1; r_1 \odot r_2 \rrbracket \Longrightarrow l_1 \odot r_2$
- → More general form: $\llbracket P \ l_1 \ r_1; Q \ r_1 \ r_2; A \rrbracket \Longrightarrow C \ l_1 \ r_2$

- → pure transitivity: $\llbracket a = b; b = c \rrbracket \implies a = c$
- → mixed: $\llbracket a \le b; b < c \rrbracket \implies a < c$



have = " $l_1 \odot r_1$ " [proof] also have "... $\odot r_2$ " [proof] also

Anatomy of a [trans] rule:

- → Usual form: plain transitivity $\llbracket l_1 \odot r_1; r_1 \odot r_2 \rrbracket \Longrightarrow l_1 \odot r_2$
- → More general form: $\llbracket P \ l_1 \ r_1; Q \ r_1 \ r_2; A \rrbracket \Longrightarrow C \ l_1 \ r_2$

- → pure transitivity: $\llbracket a = b; b = c \rrbracket \implies a = c$
- → mixed: $\llbracket a \le b; b < c \rrbracket \implies a < c$
- → substitution: $\llbracket P \ a; a = b \rrbracket \implies P \ b$



have = " $l_1 \odot r_1$ " [proof] also have "... $\odot r_2$ " [proof] also

Anatomy of a [trans] rule:

- → Usual form: plain transitivity $\llbracket l_1 \odot r_1; r_1 \odot r_2 \rrbracket \Longrightarrow l_1 \odot r_2$
- → More general form: $\llbracket P \ l_1 \ r_1; Q \ r_1 \ r_2; A \rrbracket \Longrightarrow C \ l_1 \ r_2$

- → pure transitivity: $\llbracket a = b; b = c \rrbracket \implies a = c$
- → mixed: $\llbracket a \le b; b < c \rrbracket \implies a < c$
- → substitution: $\llbracket P \ a; a = b \rrbracket \implies P \ b$
- → antisymmetry: $\llbracket a < b; b < a \rrbracket \implies False$

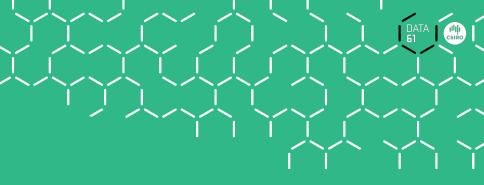


have = " $l_1 \odot r_1$ " [proof] also have "... $\odot r_2$ " [proof] also

Anatomy of a [trans] rule:

- → Usual form: plain transitivity $\llbracket l_1 \odot r_1; r_1 \odot r_2 \rrbracket \Longrightarrow l_1 \odot r_2$
- → More general form: $\llbracket P \ l_1 \ r_1; Q \ r_1 \ r_2; A \rrbracket \Longrightarrow C \ l_1 \ r_2$

- → pure transitivity: $\llbracket a = b; b = c \rrbracket \implies a = c$
- → mixed: $\llbracket a \le b; b < c \rrbracket \implies a < c$
- → substitution: $\llbracket P \ a; a = b \rrbracket \implies P \ b$
- → antisymmetry: $\llbracket a < b; b < a \rrbracket \implies False$
- → monotonicity: $\llbracket a = f \ b; b < c; \land x \ y. \ x < y \implies f \ x < f \ y \rrbracket \implies a < f \ c$



Demo