Script generated by TTT

Title: FDS (17.05.2019)

Date: Fri May 17 08:30:00 CEST 2019

Duration: 83:53 min

Pages: 103

6 Proof Automation

Automating Arithmetic

6 Proof Automation

Automating Arithmetic

110

Linear formulas

Only:

variables

numbers

number $*$ variable

$+, -$

Linear formulas

Only:

variables

numbers

number $*$ variable

$+, -$

$=, \leq, <$

$\neg, \wedge, \vee, \longrightarrow, \longleftrightarrow$

Examples

Linear: $3 * x + 5 * y \leq z \longrightarrow x < z$

Nonlinear: $x \leq x * x$

Extended linear formulas

Also allowed:

min, max

$even, odd$

$t\ div\ n,\ t\ mod\ n$ where n is a number

conversion functions

$nat, floor, ceiling, abs$

Automatic proof of arithmetic formulas

by $arith$

Proof method $arith$ tries to prove arithmetic formulas.

- Succeeds or fails
- Decision procedure for extended linear formulas

Automatic proof of arithmetic formulas

by $arith$

Proof method $arith$ tries to prove arithmetic formulas.

- Succeeds or fails
- Decision procedure for extended linear formulas
- Nonlinear subterms are viewed as (new) variables.
  Example: $x \leq x * x + f\ y$ is viewed as $x \leq u + v$

Automatic proof of arithmetic formulas

by (*simp add: algebra_simps*)

Automatic proof of arithmetic formulas

by (*simp add: algebra_simps*)

- The lemmas list *algebra_simps* helps to simplify arithmetic formulas

Automatic proof of arithmetic formulas

by (*simp add: algebra_simps*)

- The lemmas list *algebra_simps* helps to simplify arithmetic formulas
- It contains associativity, commutativity and distributivity of $+$ and $*$.
- This may prove the formula, may make it simpler, or may make it unreadable.

Automatic proof of arithmetic formulas

by (*simp add: field_simps*)

Automatic proof of arithmetic formulas

by (*simp add: field_simps*)

- The lemmas list *field_simps* extends *algebra_simps* by rules for /
- Can only cancel common terms in a quotient, e.g. $x * y \ / \ (x * z)$,

Automatic proof of arithmetic formulas

by (*simp add: field_simps*)

- The lemmas list *field_simps* extends *algebra_simps* by rules for /
- Can only cancel common terms in a quotient, e.g. $x * y \ / \ (x * z)$, if $x \neq 0$ can be proved.

Numerals

Numerals are syntactically different from Suc-terms.

Numerals

Numerals are syntactically different from Suc-terms.
Therefore numerals do not match Suc-patterns.

Example

Exponentiation $x \ \hat{} \ n$ is defined by Suc-recursion on n.

Numerals

Numerals are syntactically different from Suc-terms.
Therefore numerals do not match Suc-patterns.

Example

Exponentiation x ^ n is defined by Suc-recursion on n.
Therefore x ^ 2 is not simplified by *simp* and *auto*.

Numerals

Numerals are syntactically different from Suc-terms.
Therefore numerals do not match Suc-patterns.

Example

Exponentiation x ^ n is defined by Suc-recursion on n.
Therefore x ^ 2 is not simplified by *simp* and *auto*.

Numerals can be converted into Suc-terms with rule *numeral_eq_Suc*

Numerals

Numerals are syntactically different from Suc-terms.
Therefore numerals do not match Suc-patterns.

Example

Exponentiation x ^ n is defined by Suc-recursion on n.
Therefore x ^ 2 is not simplified by *simp* and *auto*.

Numerals can be converted into Suc-terms with rule *numeral_eq_Suc*

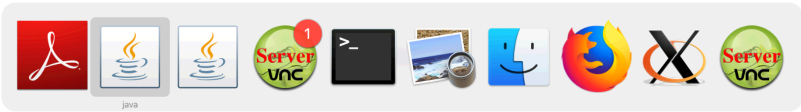
Example

simp add: numeral_eq_Suc rewrites x ^ 2 to $x * x$

`Auto_Proof_Demo.thy`

Arithmetic

Auto_Proof_Demo.thy



Step-by-step proofs can be necessary if automation fails and you have to explore where and why it failed by taking the goal apart.

What are these *?-variables* ?

What are these *?-variables* ?

After you have finished a proof, Isabelle turns all free variables V in the theorem into $?V$.

What are these *?-variables* ?

After you have finished a proof, Isabelle turns all free variables V in the theorem into $?V$.

Example: theorem `conjI`: $[\![?P;\ ?Q]\!] \Longrightarrow ?P \wedge ?Q$

What are these *?-variables* ?

After you have finished a proof, Isabelle turns all free variables V in the theorem into $?V$.

Example: theorem `conjI`: $[\![?P;\ ?Q]\!] \Longrightarrow ?P \wedge ?Q$

These ?-variables can later be instantiated:

What are these *?-variables* ?

After you have finished a proof, Isabelle turns all free variables V in the theorem into $?V$.

Example: theorem `conjI`: $[\![?P;\ ?Q]\!] \Longrightarrow ?P \wedge ?Q$

These ?-variables can later be instantiated:

- By hand:
  `conjI[of "a=b" "False"]` $\leadsto$

What are these *?-variables* ?

After you have finished a proof, Isabelle turns all free variables V in the theorem into $?V$.

Example: theorem `conjI`: $[\![?P; ?Q]\!] \Longrightarrow ?P \wedge ?Q$

These ?-variables can later be instantiated:

- By hand:
  `conjI[of "a=b" "False"]` $\leadsto$
  $[\![a = b; False]\!] \Longrightarrow a = b \wedge False$

What are these *?-variables* ?

After you have finished a proof, Isabelle turns all free variables V in the theorem into $?V$.

Example: theorem `conjI`: $[\![?P; ?Q]\!] \Longrightarrow ?P \wedge ?Q$

These ?-variables can later be instantiated:

- By hand:
  `conjI[of "a=b" "False"]` $\leadsto$
  $[\![a = b; False]\!] \Longrightarrow a = b \wedge False$
- By unification:
  unifying $?P \wedge ?Q$ with $a{=}b \wedge False$

What are these *?-variables* ?

After you have finished a proof, Isabelle turns all free variables V in the theorem into $?V$.

Example: theorem `conjI`: $[\![?P; ?Q]\!] \Longrightarrow ?P \wedge ?Q$

These ?-variables can later be instantiated:

- By hand:
  `conjI[of "a=b" "False"]` $\leadsto$
  $[\![a = b; False]\!] \Longrightarrow a = b \wedge False$
- By unification:
  unifying $?P \wedge ?Q$ with $a{=}b \wedge False$
  sets $?P$ to $a{=}b$ and $?Q$ to $False$.

Rule application

Rule application

Example: rule: $[\![?P;\ ?Q]\!] \Longrightarrow ?P \wedge ?Q$

Rule application

Example: rule: $[\![?P;\ ?Q]\!] \Longrightarrow ?P \wedge ?Q$

subgoal: 1. $\ldots \Longrightarrow A \wedge B$

Rule application

Example: rule: $[\![?P;\ ?Q]\!] \Longrightarrow ?P \wedge ?Q$

subgoal: 1. $\ldots \Longrightarrow A \wedge B$

Result: 1. $\ldots \Longrightarrow A$

2. $\ldots \Longrightarrow B$

Rule application

Example: rule: $[\![?P;\ ?Q]\!] \Longrightarrow ?P \wedge ?Q$

subgoal: 1. $\ldots \Longrightarrow A \wedge B$

Result: 1. $\ldots \Longrightarrow A$

2. $\ldots \Longrightarrow B$

The general case: applying rule $[\![\ A_1;\ \ldots\ ;\ A_n\ ]\!] \Longrightarrow A$ to subgoal $\ldots \Longrightarrow C$:

Rule application

Example: rule: $[\![?P; ?Q]\!] \Longrightarrow ?P \wedge ?Q$

subgoal: $1.\ \ldots \Longrightarrow A \wedge B$

Result: $1.\ \ldots \Longrightarrow A$

$2.\ \ldots \Longrightarrow B$

The general case: applying rule $[\![\ A_1; \ldots ; A_n\ ]\!] \Longrightarrow A$ to subgoal $\ldots \Longrightarrow C$:

- Unify A and C
- Replace C with n new subgoals $A_1 \ldots A_n$

apply($rule\ xyz$)

Typical backwards rules

$$\frac{?P \quad ?Q}{?P \wedge ?Q}\ \texttt{conjI}$$

Typical backwards rules

$$\frac{?P \quad ?Q}{?P \wedge ?Q}\ \texttt{conjI}$$

$$\frac{?P \Longrightarrow ?Q}{?P \longrightarrow ?Q}\ \texttt{impI}$$

Typical backwards rules

$$\frac{?P \quad ?Q}{?P \wedge ?Q}\ \texttt{conjI}$$

$$\frac{?P \Longrightarrow ?Q}{?P \longrightarrow ?Q}\ \texttt{impI}$$ $$\frac{\bigwedge x.\ ?P\ x}{\forall x.\ ?P\ x}\ \texttt{allI}$$

Typical backwards rules

$$\frac{?P \quad ?Q}{?P \land ?Q}\ \texttt{conjI}$$

$$\frac{?P \Longrightarrow ?Q}{?P \longrightarrow ?Q}\ \texttt{impI} \qquad \frac{\bigwedge x.\ ?P\ x}{\forall\, x.\ ?P\ x}\ \texttt{allI}$$

$$\frac{?P \Longrightarrow ?Q \quad ?Q \Longrightarrow ?P}{?P = ?Q}\ \texttt{iffI}$$

Typical backwards rules

$$\frac{?P \quad ?Q}{?P \land ?Q}\ \texttt{conjI}$$

$$\frac{?P \Longrightarrow ?Q}{?P \longrightarrow ?Q}\ \texttt{impI} \qquad \frac{\bigwedge x.\ ?P\ x}{\forall\, x.\ ?P\ x}\ \texttt{allI}$$

$$\frac{?P \Longrightarrow ?Q \quad ?Q \Longrightarrow ?P}{?P = ?Q}\ \texttt{iffI}$$

They are known as introduction rules
because they *introduce* a particular connective.

Forward proof: OF

If r is a theorem $A \Longrightarrow B$

Forward proof: OF

If r is a theorem $A \Longrightarrow B$
and s is a theorem that unifies with A

Forward proof: OF

If r is a theorem $A \Longrightarrow B$
and s is a theorem that unifies with A then

$$r[OF\ s]$$

is the theorem obtained by proving A with s.

Forward proof: OF

If r is a theorem $A \Longrightarrow B$
and s is a theorem that unifies with A then

$$r[OF\ s]$$

is the theorem obtained by proving A with s.

Example: theorem `refl`: $?t = ?t$

```
conjI[OF refl[of "a"]]
```


Forward proof: OF

If r is a theorem $A \Longrightarrow B$
and s is a theorem that unifies with A then

$$r[OF\ s]$$

is the theorem obtained by proving A with s.

Example: theorem `refl`: $?t = ?t$

```
conjI[OF refl[of "a"]]
```


$$\rightsquigarrow$$

$$?Q \Longrightarrow a = a \wedge ?Q$$

Forward proof: OF

If r is a theorem $A \Longrightarrow B$
and s is a theorem that unifies with A then

$$r[OF\ s]$$

is the theorem obtained by proving A with s.

Example: theorem `refl`: $?t = ?t$

```
conjI[OF refl[of "a"]]
```

$$\rightsquigarrow$$

$$?Q \Longrightarrow a = a \wedge ?Q$$

Forward proof: OF

If r is a theorem $A \Longrightarrow B$
and s is a theorem that unifies with A then

$$r[OF\ s]$$

is the theorem obtained by proving A with s.

Example: theorem `refl`: $?t = ?t$

```
conjI[OF refl[of "a"]]
```

$$\rightsquigarrow$$

$$?Q \Longrightarrow a = a \wedge ?Q$$

The general case:

If r is a theorem $[\![A_1; \ldots; A_n]\!] \Longrightarrow A$
and $r_1, \ldots, r_m$ $(m {\leq} n)$ are theorems then

$$r[OF\ r_1\ \ldots\ r_m]$$

is the theorem obtained
by proving $A_1 \ldots A_m$ with $r_1 \ldots r_m$.

The general case:

If r is a theorem $[\![A_1; \ldots; A_n]\!] \Longrightarrow A$
and $r_1, \ldots, r_m$ $(m {\leq} n)$ are theorems then

$$r[OF\ r_1\ \ldots\ r_m]$$

is the theorem obtained
by proving $A_1 \ldots A_m$ with $r_1 \ldots r_m$.

Example: theorem `refl`: $?t = ?t$

```
conjI[OF refl[of "a"] refl[of "b"]]
```


The general case:

If r is a theorem $[\![A_1; \ldots; A_n]\!] \Longrightarrow A$
and $r_1, \ldots, r_m$ $(m {\leq} n)$ are theorems then

$$r[OF\ r_1\ \ldots\ r_m]$$

is the theorem obtained
by proving $A_1 \ldots A_m$ with $r_1 \ldots r_m$.

Example: theorem `refl`: $?t = ?t$

```
conjI[OF refl[of "a"] refl[of "b"]]
```

$$\rightsquigarrow$$

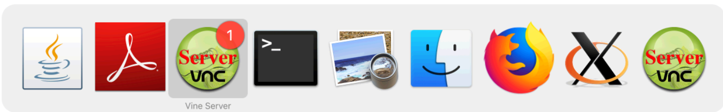
$$a = a \wedge b = b$$

From now on: ? mostly suppressed on slides

Single_Step_Demo.thy

TTT Fri 09:06 Tobias Nipkow

Single_Step_Demo.thy



Isabelle File Edit Search Markers Folding View Utilities Macros Plugins Help Fri 09:07 Tobias Nipkow

Isabelle2018/HOL - Single_Step_Demo.thy

$\Longrightarrow$ versus $\longrightarrow$

$\Longrightarrow$ is part of the Isabelle framework. It structures theorems and proof states: $[\![A_1; \ldots; A_n]\!] \Longrightarrow A$

$\longrightarrow$ is part of HOL and can occur inside the logical formulas A_i and A.

$\Longrightarrow$ versus $\longrightarrow$

$\Longrightarrow$ is part of the Isabelle framework. It structures theorems and proof states: $[\![A_1; \ldots; A_n]\!] \Longrightarrow A$

$\longrightarrow$ is part of HOL and can occur inside the logical formulas A_i and A.

Phrase theorems like this $[\![A_1; \ldots; A_n]\!] \Longrightarrow A$
not like this $A_1 \wedge \ldots \wedge A_n \longrightarrow A$

Chapter 5

Isar: A Language for Structured Proofs

Apply scripts

- unreadable

Apply scripts

- unreadable
- hard to maintain

Apply scripts versus Isar proofs

Apply script = assembly language program

Isar proof = structured program with assertions

Apply scripts versus Isar proofs

Apply script = assembly language program

Isar proof = structured program with assertions

But: **apply** still useful for proof exploration

A typical Isar proof

proof
  assume $formula_0$
  have $formula_1$ **by** $simp$
  $\vdots$
  have $formula_n$ **by** $blast$
  show $formula_{n+1}$ **by** $\ldots$
qed

A typical Isar proof

proof
  assume $formula_0$
  have $formula_1$ **by** $simp$
  ⋮
  have $formula_n$ **by** $blast$
  show $formula_{n+1}$ **by** $\ldots$
qed

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

Isar core syntax

proof = **proof** [method] step* **qed**
  | **by** method

Isar core syntax

proof = **proof** [method] step* **qed**
  | **by** method

method = $(simp \ldots) \mid (blast \ldots) \mid (induction \ldots) \mid \ldots$

Isar core syntax

proof = **proof** [method] step* **qed**
  | **by** method

method = $(simp \ldots) \mid (blast \ldots) \mid (induction \ldots) \mid \ldots$

step = **fix** variables ($\bigwedge$)
  | **assume** prop ($\Longrightarrow$)
  | [**from** fact$^+$] (**have** | **show**) prop proof

Isar core syntax

proof = **proof** [method] step* **qed**
| **by** method

method = $(\textit{simp} \ldots) \mid (\textit{blast} \ldots) \mid (\textit{induction} \ldots) \mid \ldots$

step = **fix** variables ($\bigwedge$)
| **assume** prop ($\Longrightarrow$)
| [**from** fact$^+$] (**have** | **show**) prop proof

prop = [name:] "formula"

Isar core syntax

proof = **proof** [method] step* **qed**
| **by** method

method = $(\textit{simp} \ldots) \mid (\textit{blast} \ldots) \mid (\textit{induction} \ldots) \mid \ldots$

step = **fix** variables ($\bigwedge$)
| **assume** prop ($\Longrightarrow$)
| [**from** fact$^+$] (**have** | **show**) prop proof

prop = [name:] "formula"

fact = name | ...

Example: Cantor's theorem

lemma $\neg\ surj(f :: 'a \Rightarrow 'a\ set)$

Example: Cantor's theorem

lemma $\neg\ surj(f :: 'a \Rightarrow 'a\ set)$
proof

Example: Cantor's theorem

lemma $\neg\ surj(f :: 'a \Rightarrow 'a\ set)$
proof default proof: assume *surj*, show *False*

Example: Cantor's theorem

lemma $\neg\ surj(f :: 'a \Rightarrow 'a\ set)$
proof default proof: assume *surj*, show *False*
assume a: $surj\ f$

Example: Cantor's theorem

lemma $\neg\ surj(f :: 'a \Rightarrow 'a\ set)$
proof default proof: assume *surj*, show *False*
assume a: $surj\ f$
from a **have** b: $\forall\ A.\ \exists\ a.\ A = f\ a$

Example: Cantor's theorem

lemma $\neg$ *surj*$(f :: 'a \Rightarrow 'a\ set)$
proof default proof: assume *surj*, show *False*
assume a: *surj f*
from a **have** b: $\forall\ A.\ \exists\ a.\ A = f\ a$
by(*simp add*: *surj_def*)

Example: Cantor's theorem

lemma $\neg$ *surj*$(f :: 'a \Rightarrow 'a\ set)$
proof default proof: assume *surj*, show *False*
assume a: *surj f*
from a **have** b: $\forall\ A.\ \exists\ a.\ A = f\ a$
by(*simp add*: *surj_def*)

from b **have** c: $\exists\ a.\ \{x.\ x \notin f\ x\} = f\ a$

Example: Cantor's theorem

lemma $\neg$ *surj*$(f :: 'a \Rightarrow 'a\ set)$
proof default proof: assume *surj*, show *False*
assume a: *surj f*
from a **have** b: $\forall\ A.\ \exists\ a.\ A = f\ a$
by(*simp add*: *surj_def*)
from b **have** c: $\exists\ a.\ \{x.\ x \notin f\ x\} = f\ a$

by *blast*

Example: Cantor's theorem

lemma $\neg$ *surj*$(f :: 'a \Rightarrow 'a\ set)$
proof default proof: assume *surj*, show *False*
assume a: *surj f*
from a **have** b: $\forall\ A.\ \exists\ a.\ A = f\ a$
by(*simp add*: *surj_def*)
from b **have** c: $\exists\ a.\ \{x.\ x \notin f\ x\} = f\ a$
by *blast*

from c **show** *False*
by *blast*

Example: Cantor's theorem

lemma $\neg\ surj(f :: 'a \Rightarrow 'a\ set)$
proof default proof: assume *surj*, show *False*
  assume a: $surj\ f$
  from a **have** b: $\forall\ A.\ \exists\ a.\ A = f\ a$
    by$(simp\ add\colon surj\_def)$
  from b **have** c: $\exists\ a.\ \{x.\ x \notin f\ x\} = f\ a$
    by *blast*
  from c **show** *False*
    by *blast*
qed

Isar_Demo.thy

Cantor and abbreviations

Isar_Demo.thy

Abbreviations

this	=	the previous proposition proved or assumed
then	=	**from** *this*
thus	=	**then show**
hence	=	**then have**

Example: Cantor's theorem

lemma $\neg\ surj(f :: 'a \Rightarrow 'a\ set)$
proof default proof: assume *surj*, show *False*
  assume a: $surj\ f$
  from a **have** b: $\forall\ A.\ \exists\ a.\ A = f\ a$
    by($simp\ add$: $surj\_def$)
  from b **have** c: $\exists\ a.\ \{x.\ x \notin f\ x\} = f\ a$
    by $blast$
  from c **show** $False$
    by $blast$
qed

using and **with**

(**have**|**show**) prop **using** facts

Structured lemma statement

lemma
  fixes $f :: 'a \Rightarrow 'a\ set$
  assumes s: $surj\ f$
  shows $False$

Structured lemma statement

lemma
  fixes $f :: 'a \Rightarrow 'a\ set$
  assumes s: $surj\ f$
  shows $False$
proof $-$

Structured lemma statement

lemma
fixes $f :: 'a \Rightarrow 'a\ set$
assumes s: $surj\ f$
shows $False$
proof – no automatic proof step
have $\exists\ a.\ \{x.\ x \notin f\ x\} = f\ a$ **using** s
by$(auto\ simp$: $surj\_def)$

Structured lemma statement

lemma
fixes $f :: 'a \Rightarrow 'a\ set$
assumes s: $surj\ f$
shows $False$
proof – no automatic proof step
have $\exists\ a.\ \{x.\ x \notin f\ x\} = f\ a$ **using** s
by$(auto\ simp$: $surj\_def)$
thus $False$ **by** $blast$
qed

Structured lemma statement

lemma
fixes $f :: 'a \Rightarrow 'a\ set$
assumes s: $surj\ f$
shows $False$
proof – no automatic proof step
have $\exists\ a.\ \{x.\ x \notin f\ x\} = f\ a$ **using** s
by$(auto\ simp$: $surj\_def)$
thus $False$ **by** $blast$
qed

Proves $surj\ f \Longrightarrow False$

Structured lemma statement

lemma
fixes $f :: 'a \Rightarrow 'a\ set$
assumes s: $surj\ f$
shows $False$
proof – no automatic proof step
have $\exists\ a.\ \{x.\ x \notin f\ x\} = f\ a$ **using** s
by$(auto\ simp$: $surj\_def)$
thus $False$ **by** $blast$
qed

Proves $surj\ f \Longrightarrow False$
but $surj\ f$ *becomes local fact* s *in proof.*

The essence of structured proofs

Assumptions and intermediate facts
can be named and referred to explicitly and selectively

Structured lemma statements

fixes $x :: \tau_1$ **and** $y :: \tau_2$ ...
assumes a: P **and** b: Q ...
shows R

Structured lemma statements

fixes $x :: \tau_1$ **and** $y :: \tau_2$ ...
assumes a: P **and** b: Q ...
shows R

- **fixes** and **assumes** sections optional

Case distinction

show R
proof $cases$
  assume P
  ⋮
  show R $\langle proof \rangle$
next
  assume $\neg P$
  ⋮
  show R $\langle proof \rangle$
qed

Case distinction

show R
proof $cases$
  assume P
  ⋮
  show R $\langle proof \rangle$
next
  assume $\neg P$
  ⋮
  show R $\langle proof \rangle$
qed

have $P \vee Q$ $\langle proof \rangle$
then show R
proof
  assume P
  ⋮
  show R $\langle proof \rangle$
next
  assume Q
  ⋮
  show R $\langle proof \rangle$
qed

Contradiction

show $\neg P$
proof
  assume P
  ⋮
  show $False$ $\langle proof \rangle$
qed

Contradiction

show $\neg P$
proof
  assume P
  ⋮
  show $False$ $\langle proof \rangle$
qed

Contradiction

show $\neg P$
proof
  assume P
  $\vdots$
  show $False$ $\langle proof \rangle$
qed

show P
proof $(rule\ ccontr)$
  assume $\neg P$
  $\vdots$
  show $False$ $\langle proof \rangle$
qed

$\longleftrightarrow$

show $P \longleftrightarrow Q$
proof
  assume P
  $\vdots$
  show Q $\langle proof \rangle$
next
  assume Q
  $\vdots$
  show P $\langle proof \rangle$
qed

$\forall$ and $\exists$ introduction

show $\forall x.\ P(x)$
proof
  fix x local fixed variable
  show $P(x)$ $\langle proof \rangle$
qed

$\forall$ and $\exists$ introduction

show $\forall x.\ P(x)$
proof
  fix x local fixed variable
  show $P(x)$ $\langle proof \rangle$