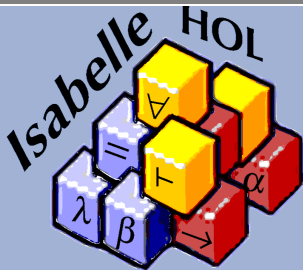


Theorembeweiserpraktikum

Anwendungen in der Sprachtechnologie

LEHRSTUHL PROGRAMMIERPARADIGMEN



Teil XVII

Verknüpfung von Beweismethoden

Hintereinanderausführung mit Backtracking

lemma " $\forall x. P x \implies \forall x. Q x \implies Q a$ "

apply (*erule allE*)

by *assumption* \longleftarrow geht schief!

stattdessen:

lemma " $\forall x. P x \implies \forall x. Q x \implies Q a$ "

by (*erule allE, assumption*) \longleftarrow geht gut!

, wendet Beweismethoden hintereinander an und verwendet dabei gemeinsamen Backtracking-Stack.

Structural Composition

apply ($m_1; m_2$) wendet m_2 auf alle Teilziele an, die durch die Anwendung von m_1 neu entstehen.

Alternative Choices

apply ($m_1 | m_2$) wendet m_1 an. (Nur) falls m_1 fehlschlägt wird m_2 angewendet.

Try

apply $m?$ wendet m an, bricht aber nicht ab falls Anwendung fehlschlägt.

Repeat

apply $m+$ wendet m mehrfach an (min. 1 mal), bis m fehlschlägt.

Restriction to subgoals

apply $m[n]$ wendet m an und schränkt den Beweiszustand dabei auf die ersten n Teilziele ein.

Teil XVIII

Automatische Beweissuche

Das Kommando **sledgehammer** führt eine automatische Beweissuche auf dem aktuellen Teilziel durch.

Optionen

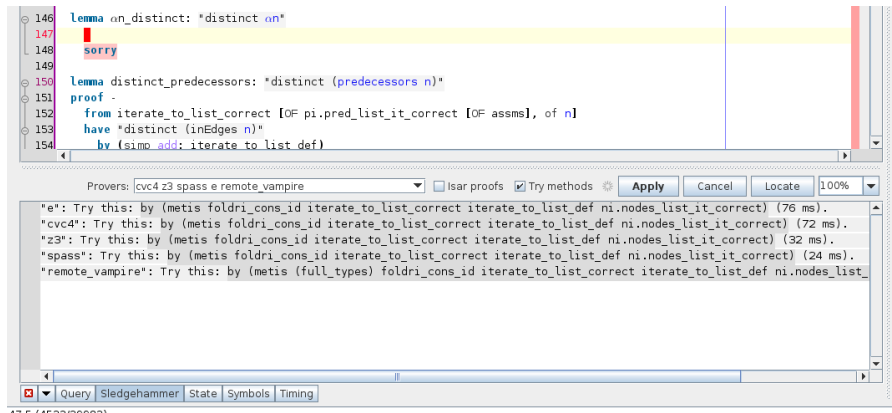
timeout Setzt die maximale Suchdauer in Sekunden fest.
Beispiel `[timeout = 360]`

isar_proofs Erzeuge Isar-Beweise anstelle von Ein-Zeilern (verwendet nur ATPs keine SMT-Solver).

provers Verwende nur die angegebenen Beweiser.
Beispiel `[provers = e spass remote_vampire]`

Siehe auch: `isabelle doc sledgehammer`

sledgehammer ist in Isabelle/jEdit auch als Panel verfügbar:



```
146 lemma on_distinct: "distinct on"
147
148 sorry
149
150 lemma distinct_predecessors: "distinct (predecessors n)"
151 proof -
152   from iterate_to_list_correct [OF pi.pred_list_it_correct [OF assms], of n]
153   have "distinct (inEdges n)"
154     by (simp add: iterate_to_list_def)
```

Provers: cvc4 z3 spass e remote_vampire Isar proofs Try methods 100%

"e": Try this: by (metis foldri_cons_id iterate_to_list_correct iterate_to_list_def ni.nodes_list_it_correct) (76 ms).
"cvc4": Try this: by (metis foldri_cons_id iterate_to_list_correct iterate_to_list_def ni.nodes_list_it_correct) (72 ms).
"z3": Try this: by (metis foldri_cons_id iterate_to_list_correct iterate_to_list_def ni.nodes_list_it_correct) (32 ms).
"spass": Try this: by (metis foldri_cons_id iterate_to_list_correct iterate_to_list_def ni.nodes_list_it_correct) (24 ms).
"remote_vampire": Try this: by (metis (full_types) foldri_cons_id iterate_to_list_correct iterate_to_list_def ni.nodes_list_

Query Sledgehammer State Symbols Timing

Teil XIX

Split-Regeln

Für jeden mit **datatype** erzeugten Typ T , wird auch eine Konstante $case_T$ erzeugt, welche für *Pattern Matching* verwendet wird.

Beispiel

```
datatype color = Red | Black
```

```
datatype 'a rbt = Empty | Node color "'a rbt" 'a "'a rbt"
```

liefert

```
case_color :: "'a ⇒ 'a ⇒ color ⇒ 'a"
```

```
case_rbt :: "'a ⇒ (color ⇒ 'b rbt ⇒ 'b ⇒ 'b rbt ⇒ 'a) ⇒ 'b rbt  
⇒ 'a"
```

mit Syntax

```
case_color f g c ≡ case c of Red ⇒ f | Black ⇒ g
```

```
case_rbt f g t ≡ case t of Empty ⇒ f | Node c l v r ⇒ g c l v r
```

Simplifikation von *case*-expressions

Wie kann der Simplifier mit *case*-expressions umgehen?

Wie kann der Simplifier mit *case*-expressions umgehen?

Lösung: Splitter

- Der *Splitter* ist Teil des Simplifiers.
- Wird aufgerufen, wenn Rewriting abgeschlossen.
- Wendet eine Split-Regel an (falls möglich).
- Startet Simplifikation auf dem/n neue/n Teilziel/en.

Split-Regel für *color*:

`color.split: P (case color of Red \Rightarrow f1 | Black \Rightarrow f2)`

`$\longleftrightarrow (color = Red \longrightarrow P f1) \wedge (color = Black \longrightarrow P f2)$`

Split-Regeln gibt es in zwei Varianten:

für Konklusion: $?P (f ?a ?b ?c) \longleftrightarrow ?Q ?a ?b ?c$

- Splitter unifiziert linke Seite mit Teiltermen der Konklusion aus dem aktuellen Ziel.
- Ersetze unifizierten Teilterm durch rechte Seite.

für Annahmen: $?P (f ?a ?b ?c) \longleftrightarrow \neg ?Q ?a ?b ?c$

- Splitter unifiziert linke Seite mit Teiltermen der Annahmen aus dem aktuellen Ziel.
- Ersetze unifizierten Teilterm durch rechte Seite
- **und** löse einige logische Verknüpfungen auf:
 - Top-Level Disjunktionen in $?Q$ werden zu Teilzielen
 - Konjunktionen in $?Q$ werden zu einzelnen Annahmen (des jeweiligen Teilziels)
 - Negationen von $?P$ in $?Q$ werden „richtig rumgedreht“.

Split in Konklusion:

lemma fixes $n :: nat$

shows "even (case c of Red $\Rightarrow 2*n$ | Black $\Rightarrow 4*n$)"

by (simp split: color.split)

$color.split: P$ (case color of Red $\Rightarrow f1$ | Black $\Rightarrow f2$)

$\longleftrightarrow (color = Red \longrightarrow P f1) \wedge (color = Black \longrightarrow P f2)$

Split in Konklusion:

```
lemma fixes n :: nat
  shows "even (case c of Red  $\Rightarrow$  2*n | Black  $\Rightarrow$  4*n)"
  by (simp split: color.split)
```

Split in Annahme:

```
lemma fixes n :: nat
  assumes "even (case c of Red  $\Rightarrow$  n | Black  $\Rightarrow$  m)"
  shows "even (n*m)"
  by (simp split: color.split_asm)
```

```
color.split: P (case color of Red  $\Rightarrow$  f1 | Black  $\Rightarrow$  f2)
 $\longleftrightarrow$  (color = Red  $\longrightarrow$  P f1)  $\wedge$  (color = Black  $\longrightarrow$  P f2)
```

```
color.split_asm: P (case color of Red  $\Rightarrow$  f1 | Black  $\Rightarrow$  f2)
 $\longleftrightarrow$   $\neg$  (color = Red  $\wedge$   $\neg$  P f1  $\vee$  color = Black  $\wedge$   $\neg$  P f2)
```

Split in Konklusion:

```
lemma fixes n :: nat
  shows "even (case c of Red  $\Rightarrow$  2*n | Black  $\Rightarrow$  4*n)"
  by (simp split: color.split)
```

Split in Annahme:

```
lemma fixes n :: nat
  assumes "even (case c of Red  $\Rightarrow$  n | Black  $\Rightarrow$  m)"
  shows "even (n*m)"
  by (simp split: color.split_asm)
```

```
color.split: P (case color of Red  $\Rightarrow$  f1 | Black  $\Rightarrow$  f2)
 $\longleftrightarrow$  (color = Red  $\longrightarrow$  P f1)  $\wedge$  (color = Black  $\longrightarrow$  P f2)
```

```
color.split_asm: P (case color of Red  $\Rightarrow$  f1 | Black  $\Rightarrow$  f2)
 $\longleftrightarrow$   $\neg$  (color = Red  $\wedge$   $\neg$  P f1  $\vee$  color = Black  $\wedge$   $\neg$  P f2)
```

```
color.splits = color.split color.split_asm
```

Split-Regeln gibt es nicht nur für *case*-expressions eines Datentyps, sondern für beliebige Funktionssymbole.

Beispiel: *if*

split_if:

$$P (\text{if } Q \text{ then } x \text{ else } y) \longleftrightarrow (Q \longrightarrow P x) \wedge (\neg Q \longrightarrow P y)$$

split_if_asm:

$$P (\text{if } Q \text{ then } x \text{ else } y) \longleftrightarrow \neg (Q \wedge \neg P x \vee \neg Q \wedge \neg P y)$$

Teil XX

Maps

Map: partielle Abbildung, also rechte Seite undefiniert für manche linke Seite

Typen inklusive Undefiniertheit in Isabelle: *'a option*

datatype *'a option = None | Some 'a*

- enthält alle Werte in *'a* mit *Some* vornangesetzt
- spezieller Wert *None* für Undefiniertheit

Beispiel: *bool option* besitzt die Elemente *None*, *Some True* und *Some False*

also Maps in Isabelle von Typ $'a$ nach $'b$ haben Typ $'a \Rightarrow 'b$ *option*
oder kurz $'a \rightarrow 'b$ (\rightarrow ist $\langle \text{rightarrow} \rangle$, Kürzel $\cdot \rightarrow$)

- leere Map (also überall undefiniert): *empty*
- Update der Map M , so dass x auf y zeigt: $M(x \mapsto y)$ (\mapsto : $| - \rangle$)
- Wert von x in Map M auf undefiniert setzen: $M(x := None)$
(Hinweis: $M(x \mapsto y)$ entspricht $M(x := Some\ y)$)
- x hat in Map M Wert y , wenn gilt: $M\ x = Some\ y$
- x ist in Map M undefiniert, wenn gilt: $M\ x = None$
- um Map eigenen Typnamen zu geben: **type_synonym**
Beispiel: **type_synonym** *nenv* = *nat* \rightarrow *bool*

Falls mehr Infos zu Maps nötig: *Isabelle-Verzeichnis/src/HOL/Map.thy*