Rechnerübung zu Theorembeweiser und ihre Anwendungen

Prof. Dr.-Ing. Gregor Snelting
Dipl.-Inf. Univ. Daniel Wasserrab

Lehrstuhl Programmierparadigmen
IPD Snelting
Universität Karlsruhe (TH)

Teil III

Fallunterscheidung und Simplifikation

Fallunterscheidung

In (klassischen) Beweisen Fallunterscheidung wichtiges Hilfsmittel

In Isabelle: einfach durch `apply` der Methode *cases P*
(*P* beliebiges boolesches Prädikat)
teilt aktuelles Subgoal in 2 neue auf:

- erstes mit *P* zusätzlich in den Annahmen
- zweites mit ¬ *P* in den Annahmen

Fallunterscheidung

In (klassischen) Beweisen Fallunterscheidung wichtiges Hilfsmittel

In Isabelle: einfach durch `apply` der Methode *`cases P`*
(*P* beliebiges boolesches Prädikat)
teilt aktuelles Subgoal in 2 neue auf:

- erstes mit *P* zusätzlich in den Annahmen
- zweites mit ¬ *P* in den Annahmen

Fallunterscheidung: Beispiel

aktuelles Subgoal: 1. $[\![B;\ C]\!] \Longrightarrow A \wedge B \vee \neg A \wedge C$
so nicht lösbar!

jedoch nach `apply (cases A)` neue Subgoals:

1. $[\![B;\ C;\ A]\!] \Longrightarrow A \wedge B \vee \neg A \wedge C$
2. $[\![B;\ C;\ \neg A]\!] \Longrightarrow A \wedge B \vee \neg A \wedge C$

jetzt einfach mit Introduktionsregeln für $\wedge$ und $\vee$ zu lösen

Fallunterscheidung: Beispiel

aktuelles Subgoal: 1. $[\![B;\ C]\!] \Longrightarrow A \wedge B \vee \neg A \wedge C$
so nicht lösbar!

jedoch nach `apply (cases A)` neue Subgoals:

1. $[\![B;\ C;\ A]\!] \Longrightarrow A \wedge B \vee \neg A \wedge C$
2. $[\![B;\ C;\ \neg A]\!] \Longrightarrow A \wedge B \vee \neg A \wedge C$

jetzt einfach mit Introduktionsregeln für $\wedge$ und $\vee$ zu lösen

Fallunterscheidung: Beispiel

aktuelles Subgoal: 1. $[\![B;\ C]\!] \Longrightarrow A \wedge B \vee \neg A \wedge C$
so nicht lösbar!

jedoch nach `apply (cases A)` neue Subgoals:

1. $[\![B;\ C;\ A]\!] \Longrightarrow A \wedge B \vee \neg A \wedge C$
2. $[\![B;\ C;\ \neg A]\!] \Longrightarrow A \wedge B \vee \neg A \wedge C$

jetzt einfach mit Introduktionsregeln für $\wedge$ und $\vee$ zu lösen

Fallunterscheidung: Beispiel

aktuelles Subgoal: 1. $[\![B;\ C]\!] \Longrightarrow A \wedge B \vee \neg A \wedge C$
so nicht lösbar!

jedoch nach `apply (cases A)` neue Subgoals:

1. $[\![B;\ C;\ A]\!] \Longrightarrow A \wedge B \vee \neg A \wedge C$
2. $[\![B;\ C;\ \neg A]\!] \Longrightarrow A \wedge B \vee \neg A \wedge C$

jetzt einfach mit Introduktionsregeln für $\wedge$ und $\vee$ zu lösen

Fallunterscheidung: Beispiel

aktuelles Subgoal: 1. $[\![B;\ C]\!] \Longrightarrow A \wedge B \vee \neg A \wedge C$
so nicht lösbar!

jedoch nach `apply (cases A)` neue Subgoals:

1. $[\![B;\ C;\ A]\!] \Longrightarrow A \wedge B \vee \neg A \wedge C$
2. $[\![B;\ C;\ \neg A]\!] \Longrightarrow A \wedge B \vee \neg A \wedge C$

jetzt einfach mit Introduktionsregeln für $\wedge$ und $\vee$ zu lösen

Einschub: `definition`

Ermöglicht direkte Definition von nichtrekursiven Funktionen
verbindet Deklaration mit Definition der Funktion
Beispiel: Funktion `nand` (= "not and")

```
definition nand :: bool ⇒ bool ⇒ bool
 where "nand A B ≡ ¬ (A ∧ B)"
```

automatisch generierte Simplifikationsregel: *nand_def*
(allgemein *Funktionsname_def*)
kann dem Simplifier übergeben werden

Einschub: `definition`

Ermöglicht direkte Definition von nichtrekursiven Funktionen
verbindet Deklaration mit Definition der Funktion
Beispiel: Funktion `nand` (= "not and")

```
definition nand :: bool ⇒ bool ⇒ bool
 where "nand A B ≡ ¬ (A ∧ B)"
```

automatisch generierte Simplifikationsregel: *nand_def*
(allgemein *Funktionsname_def*)
kann dem Simplifier übergeben werden

Einschub: `definition`

Ermöglicht direkte Definition von nichtrekursiven Funktionen
verbindet Deklaration mit Definition der Funktion
Beispiel: Funktion `nand` (= "not and")

```
definition nand :: bool ⇒ bool ⇒ bool
 where "nand A B ≡ ¬ (A ∧ B)"
```

automatisch generierte Simplifikationsregel: *nand_def*
(allgemein *Funktionsname_def*)
kann dem Simplifier übergeben werden

Einschub: `definition`

`nand` binärer Operator $\Longrightarrow$ Infixoperator bietet sich an

Syntaxdefinition in Isabelle durch Anhängen des Ausdrucks *(infixl "⋈" n)* an die Deklarationszeile, wobei

- `infixl` für linksgebundenen Infixoperator steht, analog `infixr` für rechtsgebundene
- $\bowtie$ ein beliebig wählbares Symbol für den Operator ist
- *n* Zahl, welche die Präzedenz dieses Operators angibt

Kann jetzt *A ⋈ B* statt *nand A B* schreiben

Simplifikation

genauer: Termersetzung, weil Ausdruck nicht notwendigerweise einfacher
Simplifikationsregeln: Gleichung
entsprechende Taktik: `simp`

- besitzt Pool an Termersetzungsregeln
- prüft für jede solche Regel, ob Term mit linker Seite einer Gleichung unifizierbar
- falls ja, ersetzen mit entsprechend unifizierter rechten Seite

Beispiel: aktuelles Subgoal: $C \Longrightarrow$ *if b then s1 else s2*
`simp` wendet folgende Termersetzungsregel an:
if ?x then ?m else ?n = ((?x $\longrightarrow$ *?m)* $\wedge$ *(*$\neg$ *?x* $\longrightarrow$ *?n))*
Resultat: $C \Longrightarrow$ *(b* $\longrightarrow$ *s1)* $\wedge$ *(*$\neg$ *b* $\longrightarrow$ *s2)*

Simplifikation

genauer: Termersetzung, weil Ausdruck nicht notwendigerweise einfacher
Simplifikationsregeln: Gleichung
entsprechende Taktik: `simp`

- besitzt Pool an Termersetzungsregeln
- prüft für jede solche Regel, ob Term mit linker Seite einer Gleichung unifizierbar
- falls ja, ersetzen mit entsprechend unifizierter rechten Seite

Beispiel: aktuelles Subgoal: $C \Longrightarrow$ *if b then s1 else s2*
`simp` wendet folgende Termersetzungsregel an:
if ?x then ?m else ?n = ((?x $\longrightarrow$ *?m)* $\wedge$ *(*$\neg$ *?x* $\longrightarrow$ *?n))*
Resultat: $C \Longrightarrow$ *(b* $\longrightarrow$ *s1)* $\wedge$ *(*$\neg$ *b* $\longrightarrow$ *s2)*

Simplifikation

Auch bedingte Ersetzungsregeln sind möglich, also in der Form

$[\![\ldots]\!] \Longrightarrow \ldots = \ldots$

dazu: Prämissen der Regel mit aktuellen Annahmen unifizierbar

Simplifier modifizieren:

- selbstgeschriebene Simplifikationslemmas zu Taktik hinzufügen:
  `apply(simp add:`$Regel_1, Regel_2, \ldots$`)`
- nur bestimmte Ersetzungsregeln verwenden:
  `apply(simp only:`$Regel_1, Regel_2, \ldots$`)`
- Ersetzungsregeln aus dem Standardpool von `simp` entfernen:
  `apply(simp del:`$Regel_1, Regel_2, \ldots$`)`

Simplifikation

Auch bedingte Ersetzungsregeln sind möglich, also in der Form

$[\![\ldots]\!] \Longrightarrow \ldots = \ldots$

dazu: Prämissen der Regel mit aktuellen Annahmen unifizierbar

Simplifier modifizieren:

- selbstgeschriebene Simplifikationslemmas zu Taktik hinzufügen:
  `apply(simp add:`$Regel_1, Regel_2, \ldots$`)`
- nur bestimmte Ersetzungsregeln verwenden:
  `apply(simp only:`$Regel_1, Regel_2, \ldots$`)`
- Ersetzungsregeln aus dem Standardpool von `simp` entfernen:
  `apply(simp del:`$Regel_1, Regel_2, \ldots$`)`

Simplifikation

Auch möglich: Ersetzungsregeln in den Standardpool von `simp` einfügen
Zwei Varianten:

- Zusatz `[simp]` hinter Lemmanamen
  Beispiel: *lemma bla [simp]: "A = True* $\Longrightarrow$ *A* $\wedge$ *B = B"*
- mittels `declare [simp]`
  Beispiel: *declare [simp]: foo bar*
  Analog: mittels `declare [simp del]` Ersetzungsregeln
  aus Standardpool entfernen

Vorsicht!

Nur Regeln zu Standardpool hinzufügen, dessen rechte Seite einfacher als linke Seite!

Sicherstellen, dass `simp` durch neue Regeln nicht in Endlosschleifen hängenbleibt!

Simplifikation

Auch möglich: Ersetzungsregeln in den Standardpool von `simp` einfügen
Zwei Varianten:

- Zusatz `[simp]` hinter Lemmanamen
  Beispiel: *lemma bla [simp]: "A = True* $\Longrightarrow$ *A* $\wedge$ *B = B"*
- mittels `declare [simp]`
  Beispiel: *declare [simp]: foo bar*
  Analog: mittels `declare [simp del]` Ersetzungsregeln
  aus Standardpool entfernen

Vorsicht!

Nur Regeln zu Standardpool hinzufügen, dessen rechte Seite einfacher als linke Seite!

Sicherstellen, dass `simp` durch neue Regeln nicht in Endlosschleifen hängenbleibt!

Arbeiten mit *apply*-Skripten

apply-Skripten verkürzen:

- nach *apply* mehrere Regelanwendungen bzw. Taktiken hintereinander
  Bsp: *apply(simp,rule foo,auto)*
- reguläre Ausdrücke:
  - *+* hinter Regel bzw. Taktik wendet diese ein- oder mehrfach an
    Bsp: *apply(assumption)+*
  - *?* analog für null- oder einfache Anwendung
    Bsp: *apply(assumption)?*
  - */* zwischen zwei regeln bzw. Taktiken wendet die erste an,
    bei Nichtgelingen zweite; Bsp: *apply(assumption|arith)*
- *by* ersetzt letzte *apply*-Regelanwendung bzw. Taktik und **done**
  alle *assumption* Anwendungen nach *apply* automatisch angewandt
  Bsp: statt *apply(rule foo,assumption,assumption)* **done**
  nur *by(rule foo)*