

Perlen der Informatik 2

7. Übung

1 Prädikatenlogik

Zusätzlich zu den Regeln der Aussagenlogik kommen jetzt die folgenden Regeln zum Einsatz:

$$\begin{aligned} \text{exI: } P\ x &\Longrightarrow \exists x. P\ x \\ \text{exE: } \exists x. P\ x &\Longrightarrow (\bigwedge x. P\ x \Longrightarrow Q) \Longrightarrow Q \\ \text{allI: } (\bigwedge x. P\ x) &\Longrightarrow \forall x. P\ x \\ \text{allE: } \forall x. P\ x &\Longrightarrow (P\ x \Longrightarrow R) \Longrightarrow R \end{aligned}$$

Zeigen Sie die folgenden prädikatenlogischen Aussagen:

$$\begin{aligned} \forall x. (P\ x \longrightarrow Q\ x) \wedge \neg Q\ y &\Longrightarrow \neg P\ y \\ (\exists x. P\ x) \wedge (\forall x. P\ x \longrightarrow Q\ x) &\Longrightarrow \exists x. Q\ x \\ \neg (\forall x. P\ x) &\Longrightarrow \exists x. \neg P\ x \end{aligned}$$

2 Knobelaufgabe: Rich Grandfather

Ab hier dürfen alle Lemmas verwendet werden; die Methoden bleiben beschränkt auf `rule`, `erule` und `assumption` sowie `drule` und `frule`; zusätzlich ist Fallunterscheidung auf booleschen Ausdrücken mit `cases` gestattet.

Beweisen Sie die folgende in klassischer Prädikatenlogik gültige Formel zunächst informell mit Papier und Bleistift. Verwenden Sie im Beweis Fallunterscheidung oder Widerspruch.

*Wenn jeder Arme einen reichen Vater hat,
dann gibt es einen Reichen mit einem reichen Großvater.*

$$(\forall x. \neg \text{rich } x \longrightarrow \text{rich } (\text{father } x)) \longrightarrow (\exists x. \text{rich } x \wedge \text{rich } (\text{father } (\text{father } x)))$$

3 Verfeinerung von Implementierungen: noch einmal Fibonacci

Ab hier und auch in Zukunft dürfen alle Methoden und Lemmas verwendet werden.

Gehen Sie nochmal zurück zur Definition der Fibonacci-Zahlen von Blatt 3: `fib :: nat => nat`.

Wir wollen nun Fibonacci-Zahlen mit SML ausrechnen; dazu verwenden wir den Code-Generator, um die 35. Fibonacci-Zahl auszurechnen, und stoppen die Zeit.

```
definition "example _ = fib 35"
```

```

export_code example in SML module_name Fibonacci
ML {* set Toplevel.timing *}
ML {* Fibonacci.example () *} — Die unit () stellt sicher, dass die eigentliche Auswertung erst
bei diesem Aufruf geschieht und nicht schon bei der Code-Generierung.
ML {* reset Toplevel.timing *}

```

Betrachten Sie den Code, der für *fib* generiert wird:

```

export_code fib in SML module_name Fibonacci file -

```

Machen Sie sich durch Inspektion des Codes klar, dass die Zahl der Aufrufe von *fib* exponentiell mit der Eingabezahl wächst.

Wir werden nun eine effizientere, äquivalente Implementierung entwickeln. Dazu definieren wir:

```

definition fib_step :: "nat ⇒ nat × nat" where
  fib_step_def: "fib_step n = (fib (Suc n), fib n)"

```

Zeigen Sie die folgenden Rekursions-Gleichungen für *fib_step*:

```

fib_step_Zero: fib_step 0 = (Suc 0, 0)
fib_step_Suc: fib_step (Suc n) = (let (n, m) = fib_step n in (n + m, n))

```

Wir geben diese Gleichungen an den Code-Generator:

```

declare fib_step_def [code del] fib_step_Zero [code] fib_step_Suc [code]

```

Die Implementierung von *fib* soll sich nun auf *fib_step* abstützen und insbesondere *keine* rekursiven Aufrufe für *fib* mehr enthalten. Schauen Sie sich die Funktionsgleichungen von *fib* an, überlegen Sie sich, welche Gleichung(en) ersetzt werden sollen; beweisen Sie für diese geeignete Gegenstücke mit gleicher linker Seite, aber anderer rechter Seite, und geben Sie diese an den Code-Generator.

Inspizieren Sie den generierten Code:

```

export_code fib in SML module_name Fibonacci file -

```

Überzeugen Sie sich von der kürzeren Laufzeit:

```

export_code example in SML module_name Fibonacci
ML {* set Toplevel.timing *}
ML {* Fibonacci.example () *}
ML {* reset Toplevel.timing *}

```

Noch einmal zu unserem Vorgehen: wir haben *fib* dadurch effizient(er) implementiert, indem wir eine (zunächst komplizierte) Variante *fib_step* in Abhängigkeit davon definiert haben, und darüber Gleichungen bewiesen haben, die sich als Code-Gleichungen eignen. Der Zusammenhang zwischen *fib* und *fib_step* ergibt sich bei diesem Ansatz aus der Konstruktion und ist trivial zu beweisen.

