

Praktikum Spezifikation und Verifikation

1 Prädikatenlogik

Beweisen Sie die folgenden Formeln der Prädikatenlogik mit den Regeln des natürlichen Schließens wie auf Blatt 4.

lemma " $\exists x. (P\ x \longrightarrow (\forall x. P\ x))$ " *<proof>*

lemma " $(\exists x. \forall y. P\ x\ y) \longrightarrow (\forall y. \exists x. P\ x\ y)$ " *<proof>*

lemma " $(\forall x. P\ x \longrightarrow Q) = ((\exists x. P\ x) \longrightarrow Q)$ " *<proof>*

lemma " $\forall x. P\ x \longrightarrow Q\ (f\ x) \Longrightarrow \exists x. P\ x \Longrightarrow \exists x. Q\ (f\ x)$ " *<proof>*

Zusätzlich zu den Regeln der Aussagenlogik können Sie die folgenden Regeln verwenden:

exI: $P\ x \Longrightarrow \exists x. P\ x$,

exE: $[\exists x. P\ x; \bigwedge x. P\ x \Longrightarrow Q] \Longrightarrow Q$,

allI: $(\bigwedge x. P\ x) \Longrightarrow \forall x. P\ x$,

allE: $[\forall x. P\ x; P\ x \Longrightarrow R] \Longrightarrow R$.

▷ Abgabe 6. 6. 2001

2 Eine Implementierung von Mengen durch Intervall-Listen

Eine mögliche Implementierung von Mengen natürlicher Zahlen ist als Liste von Intervallen, ein Intervall kann dabei einfach als ein Paar natürlicher Zahlen aufgefaßt werden. Beispielsweise kann die Menge $\{\#2, \#3, \#5, \#7, \#8, \#9\}$ dargestellt werden durch die Liste $[(\#2, \#3), (\#5, \#5), (\#7, \#9)]$.

2.1 Definitionen

Wir führen also einen neuen Typ

```
types intervals = "(nat*nat) list"
```

ein. Diese Typ enthält alle Listen von Paaren natürlicher Zahlen, also auch solche, die nicht die Intervall-Listen-Darstellung einer Menge sind.

Führen Sie daher eine Invariante

```
consts inv :: "intervals => bool"
```

ein, also eine Bedingung, die genau diejenigen Intervall-Listen charakterisiert, die Mengen natürlicher Zahlen darstellen. Wegen der Eindeutigkeit sollen die Intervalle aufsteigend geordnet sein, die untere Grenze des Intervalls soll kleiner oder gleich der oberen Grenze sein und die Intervalle sollen so groß wie möglich sein (d.h. zwei direkt nebeneinanderliegende Intervalle sollen zu einem zusammengefasst werden). Zur Definition bietet es sich an, eine eingebettete Funktion

```
consts inv2 :: "nat => intervals => bool"
```

zu verwenden, die als zusätzlichen Parameter eine untere Grenze für die Intervall-Listen hat und die primitiv rekursiv definiert werden kann.

Definieren Sie außerdem eine Abstraktionsfunktion

```
consts set_of :: "intervals => nat set"
```

die zu einer Intervall-Liste die Menge der natürlichen Zahlen liefert, die durch die Liste repräsentiert wird. (Dabei kann man voraussetzen, dass die Liste die Invariante erfüllt.)

Definieren Sie primitiv rekursive Funktionen

```
consts add :: "(nat*nat) => intervals => intervals"
```

```
consts rem :: "(nat*nat) => intervals => intervals"
```

Zum Einfügen bzw. Löschen eines Intervalls in eine Intervall-Liste. Dabei sollen, soweit wie möglich, Intervalle zusammengefasst werden, so daß das Resultat wieder die Invariante erfüllt.

2.2 Beweis der Invarianten

```
declare Let_def [simp]
```

```
declare split_split [split]
```

Beweisen Sie zunächst die Monotonie von *inv2*.

lemma *inv2_monoton*: "*inv2 m ins* \implies *n* \leq *m* \implies *inv2 n ins*"
<proof>

Zeigen Sie, daß die Funktion *add* die Invariante erhält.

theorem *inv_add*: " $\llbracket i \leq j; \text{inv } \textit{ins} \rrbracket \implies \text{inv } (\textit{add } (i,j) \textit{ins})$ "
<proof>

Zeigen Sie, dass die Funktion *rem* die Invariante erhält.

theorem *inv_rem*: " $\llbracket i \leq j; \text{inv } \textit{ins} \rrbracket \implies \text{inv } (\textit{rem } (i,j) \textit{ins})$ "
<proof>

Hinweis: Beweisen Sie in beiden Fällen zuerst mit Induktion eine allgemeinere Aussage über *inv2* (siehe Abschnitt 8.3.1 im Tutorial).

2.3 Korrektheit der Implementierung

Zeigen Sie die Korrektheit der Funktion *add*.

theorem *set_of_add*:
" $\llbracket i \leq j; \text{inv } \textit{ins} \rrbracket \implies \text{set_of } (\textit{add } (i,j) \textit{ins}) = \text{set_of } [(i,j)] \cup \text{set_of } \textit{ins}$ "
<proof>

Zeigen Sie die Korrektheit der Funktion *rem*.

theorem *set_of_rem*:
" $\llbracket i \leq j; \text{inv } \textit{ins} \rrbracket \implies \text{set_of } (\textit{rem } (i,j) \textit{ins}) = \text{set_of } \textit{ins} - \text{set_of } [(i,j)]$ "
<proof>

Hinweis: Beweisen Sie zuerst mit Induktion eine allgemeinere Aussage. über *inv2*. Eventuell kann es hilfreich sein, auch noch einen Zusammenhang über *inv2* und *set_of* als Lemma zu beweisen.

2.4 Allgemeine Hinweise:

- Es stehen Ihnen die folgenden aus der Vorlesung bekannten Methoden zur Verfügung:
induct_tac as
simp
simp (no_asm_simp)
simp (no_asm)
simp add: r
simp only:

```

rule r
erule r
erule_tac x="t" in r
drule
drule_tac x="t" in r
arith
clarify
blast
auto
auto simp add: r
force
force simp add: r
force intro: r
subgoal_tac "g"

```

- Zum Beweis der Gleichheit zweier Mengen eignet sich die Anwendung der Regel `set_ext`: $(\bigwedge x. (x \in A) = (x \in B)) \implies A = B$.
- Nützliche Theoreme zum Vereinfachen von Gleichungen mit Mengen sind z. B.
`Un_Diff`: $A \cup B - C = A - C \cup (B - C)$
`Diff_triv`: $A \cap B = \{\} \implies A - B = A$.
- Freie Variablen in Theoremen können in der Reihenfolge ihres Auftretens in einem Theorem instantiiert werden mit dem Attribut `of`, zum Beispiel `r [of _ _ 0]` instantiiert die dritte in `r` auftretende Variable mit dem Wert `0`.
- Das Attribut `[rule_format]` bringt ein Theorem in das Standard Regel-Format. Das ist hilfreich, wenn man ein Theorem, das man mit Induktion beweisen will, in der Objekt-Logik formulieren will, aber im Regel-Format verwenden will.
- Das Attribut `[simplified]` vereinfacht ein Theorem mit dem Simplifier. Das ist insbesondere hilfreich in Kombination mit Variablen-Instantiierung.

- ▷ Abgabe Spezifikation und Invariante: 30. 5. 2001
- ▷ Abgabe Korrektheit von `add`: 6. 6. 2001
- ▷ Abgabe Korrektheit von `rem`: 13. 6. 2001