Ludwig-Maximilians-Universität München **SoSe 2015**
Institut für Informatik
Dr. M. Hölzl, C. Kroiß

Formale Techniken der Software-Entwicklung
Übungsblatt 7
Besprechung am 19.06.2015

Musterlösung

Aufgabe 1:

In der Vorlesung wurde eine einfache Theorie für den Snark-Theorembeweiser vorgestellt, mit der sich Listen ähnlich wie in z.B. ACL2 beschreiben lassen. Insbesondere findet man in der Datei `snark-prog.lisp` im Folien-Repository die folgenden Aussagen:

```
(assert '(car? (cons ?x ?xs) ?x))
(assert '(cdr? (cons ?x ?xs) ?xs))
(assert '(member? ?x (cons ?x ?xs)))
(assert '(implied-by (member? ?x (cons ?y ?xs))
   (member? ?x ?xs))))
```

Hierbei sind car, cdr und member jeweils 2-stellige Prädikate und cons ein Funktionssymbol. Diese Aussagen können in reiner Logik-Schreibweise als *Horn-Klauseln* aufgefasst werden, d.h. als Klauseln, die jeweils nur ein positives Literal enthalten. Das ergibt die Klauselmenge $\mathcal{K}$[1] :

$$\begin{aligned}\mathcal{K} = \{ & \mathsf{car}(\mathsf{cons}(x, xs), x),\\ & \mathsf{cdr}(\mathsf{cons}(x, xs), xs),\\ & \mathsf{member}(x, \mathsf{cons}(x, xs)),\\ & \mathsf{member}(x, xs) \implies \mathsf{member}(x, \mathsf{cons}(y, xs)) \}\end{aligned}$$

(a) Geben Sie ausgehend von den Klauseln in $\mathcal{K}$ zusätzliche Klauseln an, die die Semantik für folgenden Prädikate definieren:

$\mathsf{last}(x, l)$:
ist wahr, wenn x das letzte Element in der Liste l ist.

$\mathsf{butlast}(p, l)$:
ist wahr, wenn p das Präfix der Liste l ohne das letzte Element ist.

$\mathsf{palindrome}(l)$:
ist wahr, wenn die Liste l ein *Palindrom* darstellt, also z.B. $[A, B, C, B, A]$ oder $[A, B, B, A]$.

[1] Eigentlich is das vierte Element von $\mathcal{K}$ keine Klausel, weil eine Implikation enthalten ist. Die Darstellung hier ist jedoch etwas lesbarer als die tatsächliche Klauselform.

Lösung:

- $\mathsf{last}(x, \mathsf{cons}(x, nil))$
- $\mathsf{last}(x, ys) \implies \mathsf{last}(x, \mathsf{cons}(y, ys)) \equiv \{\neg\mathsf{last}(x, ys), \mathsf{last}(x, \mathsf{cons}(y, ys))\}$
- $\mathsf{butlast}(nil, \mathsf{cons}(x, nil))$
- $\mathsf{butlast}(bl, xs) \implies \mathsf{butlast}(\mathsf{cons}(x, bl), \mathsf{cons}(x, xs))$
  $\equiv \{\neg\mathsf{butlast}(bl, xs), \mathsf{butlast}(\mathsf{cons}(x, bl), \mathsf{cons}(x, xs))\}$
- $\mathsf{palindrome}(nil)$
- $\mathsf{palindrome}(\mathsf{cons}(x, nil))$
- $(\mathsf{last}(x, xs) \wedge \mathsf{butlast}(ys, xs) \wedge \mathsf{palindrome}(ys)) \implies \mathsf{palindrome}(\mathsf{cons}(x, xs))$
  $\equiv \{\neg\mathsf{last}(x, xs), \neg\mathsf{butlast}(ys, xs), \neg\mathsf{palindrome}(ys), \mathsf{palindrome}(\mathsf{cons}(x, xs))\}$

(b) Verwenden Sie den prädikatenlogischen Resolutionskalkül, um folgende Aussage zu beweisen:

$$\mathsf{butlast}(\mathsf{cons}(a, nil), \mathsf{cons}(a, \mathsf{cons}(b, nil)))$$

Lösung:

$$\begin{tikzcd}
\{\neg\mathsf{butlast}(\mathsf{cons}(a, nil), \mathsf{cons}(a, \mathsf{cons}(b, nil)))\} \arrow[d] & \{\neg\mathsf{butlast}(bl, xs), \mathsf{butlast}(\mathsf{cons}(x, bl), \mathsf{cons}(x, xs))\} \arrow[dl, "x \mapsto a, bl \mapsto nil, xs \mapsto \mathsf{cons}(b, nil)"'] \\
\{\neg\mathsf{butlast}(nil, cons(b, nil))\} \arrow[d] & \{\mathsf{butlast}(nil, \mathsf{cons}(y, nil))\} \arrow[dl, "y \mapsto b"'] \\
\{\} &
\end{tikzcd}$$

Aufgabe 2:

Modellieren Sie ein einfaches Telefonbuch-System in ACL2. Dabei sollen folgende Funktionen unterstützt werden:

`add-entry`:
: fügt einen Eintrag für einen angegebenen *Namen* mit einer angegebenen *Telefonnummer* zu einem Telefonbuch hinzu.

`entry-existsp`:
: überprüft, ob ein Telefonbuch einen Eintrag für einen bestimmten Namen enthält.

`get-number`:
: gibt die Telefonnummer für einen angegebenen Namen zurück.

(a) Spezifizieren Sie die oben angegebenen Features als ACL2-Funktionen.

Lösung:

```
(in-package "ACL2")

(defun add-entry (name number address-book)
  (cons (cons name number) address-book))

(defun entry-existsp (name address-book)
  (and (assoc name address-book) t))   ; Das "and" bewirkt hier,
                                       ; dass im positiven Fall
                                       ; "t" zurückgegeben wird.

(defun get-number (name address-book)
  (cdr (assoc name address-book)))
```

(b) Geben Sie ACL2-Theoreme an, um folgende Aussagen zu beweisen:

1. Nach dem Hinzufügen eines Eintrags für einen bestimmen Namen mit `add-entry` ergibt ein Test mit `entry-existsp`, dass tatsächlich ein Eintrag mit diesem Namen im Telefonbuch existiert.
2. Nach dem Hinzufügen eines Eintrags für einen Namen *name* mit der Nummer *num* ergibt die Suche mit `get-number` für den Namen *name* dieselbe Telefonnummer *num*.

Lösung:

```
(defthm entry-exists-after-adding-it
   (entry-existsp name
      (add-entry name number address-book))
   :rule-classes nil)

(defthm correct-number-after-adding-entry
   (equal (get-number name
      (add-entry name number addres-book))
     number)
   :rule-classes nil)
```