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

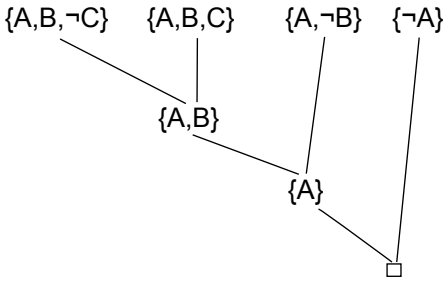
Formale Techniken der Software-Entwicklung
Übungsblatt 4
Besprechung am 22.05.2015

Musterlösung

Aufgabe 1:
Verwenden Sie den Resolutionskalkül, um zu zeigen, dass die folgenden Formeln nicht erfüllbar sind.

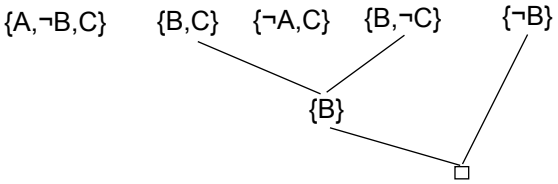
(a) $(A \vee B \vee \neg C) \wedge (A \vee B \vee C) \wedge (A \vee \neg B) \wedge \neg A$

Lösung:



(b) $(A \vee \neg B \vee C) \wedge (B \vee C) \wedge (\neg A \vee C) \wedge (B \vee \neg C) \wedge \neg B$

Lösung:



Aufgabe 2:
Beim Problem der Knotenfärbung eines Graphen wird jedem Knoten eines Graphen eine Farbe bzw. eine natürliche Zahl zugeordnet. Eine gültige Färbung ist dabei eine Zuordnung, in der für benachbarte Knoten nicht dieselbe Farbe gewählt wird (siehe z.B. `http://de.wikipedia.org/wiki/F%C3%A4rbung_(Graphentheorie)`).

(a) Geben Sie ein Schema an, mit dem sich das Problem der Knotenfärbung eines Graphen auf SAT übertragen lässt. Dieses Schema soll beschreiben, wie sich für einen beliebigen ungerichteten Graphen G und eine beliebige Farbanzahl k eine aussagenlogische Formel in konjunktiver Normalform angegeben lässt, deren Modelle als gültige Färbung von G mit k Farben interpretierbar sind.

Lösung:
Siehe Folien zur Übung am 22.5.2015.

(b) Wenden Sie Ihr Schema an, um eine aussagenlogische Formel anzugeben, die für den Graphen in Abbildung 1 eine gültige Färbung mit zwei Farben (z.B. Rot und Grün) modelliert.

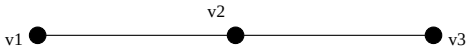


Abbildung 1: Graph für Aufgabe b

Lösung:
Siehe Folien zur Übung am 22.5.2015.

Aufgabe 3:
In dieser Aufgabe soll die ACL2-Funktion `sum-lists` betrachtet werden, die zwei Listen natürlicher Zahlen *elementweise* summiert, d.h. dass z.B. `(sum-lists '(1 2 3) '(1 2 3))` die Liste `'(2 4 6)` ergibt. Falls eine Liste länger ist als die andere, werden überschüssige Elemente der längeren Liste einfach an die Ergebnisliste angehängt, d.h. `(sum-lists '(1 2 3) '(1 2 3 4 5))` ergibt `'(2 4 6 4 5)`. `sum-lists` ist wie folgt definiert:

```
(in-package "ACL2")

(defun sum-lists (xs ys)
  (if (and (endp xs) (endp ys))
    '()
    (cons (+ (fix (first xs)) (fix (first ys)))
          (sum-lists (rest xs) (rest ys)))))
```

Hierbei wird die in ACL2 integriert Funktion `fix` verwendet, die für natürliche Zahlen die Zahl selbst und für alle anderen Argumente die Zahl 0 zurückgibt.

(a) Wenn die oben angegebene Funktionsdefinition in ACL2 submitted wird, dann führt das zu einem Fehler weil nicht automatisch ein passendes Maß gefunden wird. Ergänzen Sie daher die Funktionsdefinition um eine Maß-Angabe. Als Maß sollten Sie die Summe der beiden Eingabe-Listen verwenden. Für die Längenberechnung wiederum verwendet man die Funktion `len`.

Lösung:

```
(defun sum-lists (xs ys)
  (declare (xargs :measure (+ (len xs) (len ys)) ))
  (if (and (endp xs) (endp ys))
    '()
    (cons (+ (fix (first xs)) (fix (first ys)))
          (sum-lists (rest xs) (rest ys)))))
```

(b) Definieren Sie nun eine ACL2-Funktion `make-numlist`, die aus einer Liste mit beliebigen Elementen eine Liste erzeugt, die nur natürliche Zahlen enthält. Dazu sollen alle Elemente, die keine natürliche Zahlen sind, durch 0 ersetzt werden.

Lösung:

```
(defun make-numlist (l)
  (if (endp l)
    '()
    (cons (fix (first l)) (make-numlist (rest l)))))
```

(c) Definieren Sie zum Abschluss ein Theorem in ACL2, das folgende Behauptung beweist: „Die leere Liste ist ein Identitätselement für `sum-list`, d.h. `sum-list` von einer beliebigen Liste natürlicher Zahlen mit einer leeren Liste ergibt die unveränderte Eingabeliste."

Lösung:

```
(defthm nil-is-left-identity-for-sum-lists
  (and
   (equal (sum-lists (make-numlist l) '()) (make-numlist l))
   (equal (sum-lists '() (make-numlist l)) (make-numlist l))))
```