



Institut für Informatik  
Prof. Dr. Peter Thiemann  
Jochen Walter

Georges-Köhler-Allee 79  
D-79110 Freiburg i. Br.

Freiburg, den 10. November 2000

## Informatik 1, WiSe 2000/2001

### Übungsblatt 2

Die Aufgaben werden in den Übungs- und Programmiergruppen  
vom 9.11 bis zum 15.11. besprochen.

Die Lösungen müssen nicht abgegeben werden!

*Die Aufgaben auf diesem Blatt können in Teams bearbeitet werden. Bevor drscheme gestartet werden kann, muß setup lang eingegeben werden. Der Sprachumfang sollte auf „Advanced Student“ eingestellt werden. Die Aufgaben sind mit einem bis drei Sternen versehen, wobei die Aufgaben mit einem Stern am einfachsten, die mit dreien am schwierigsten sind.*

#### Aufgabe 1 (\*\*):

Bestimmen Sie die Wahrheitswerte folgender prädikatenlogischer Ausdrücke über der Grundmenge  $\mathbb{N}$  mit Hilfe der in der Vorlesung angegebenen Auswertungsregeln (also der Funktion  $J$ ):

- (a)  $(\exists m)(\forall n) \quad m > n$
- (b)  $(\forall m)(\exists n) \quad m < n$
- (c)  $(\exists m)m \geq 2 \wedge (\forall n)n \geq 2 \wedge m \neq n \wedge \neg(\exists q) \quad m = n \cdot q$

Erklären Sie in Worten die Bedeutung dieser Ausdrücke!

**Lösung zu Aufgabe 1:** In der Vorlesung wurde folgende Definition des Wahrheitswertes einer aussagenlogischen Aussage gegeben:

- Für primitive Aussagen  $a$  ist der Wahrheitswert durch eine Funktion  $J_0(a)$  von vornherein festgelegt.
- Falls  $A = a$ , so ist  $J(A) := J_0(a)$ .
- Falls  $A = 0$ , so ist  $J(A) := 0$ .
- Falls  $A = L$ , so ist  $J(A) := L$ .
- Falls  $A = (A' \wedge B')$ , so ist  $J(A) := \begin{cases} L & \text{falls } J(A') = J(B') = L \\ 0 & \text{sonst} \end{cases}$
- Falls  $A = (A' \vee B')$ , so ist  $J(A) := \begin{cases} 0 & \text{falls } J(A') = J(B') = 0 \\ L & \text{sonst} \end{cases}$
- Falls  $A = \neg A'$ , so ist  $J(A) := \begin{cases} 0 & \text{falls } J(A') = L \\ L & \text{falls } J(A') = 0 \end{cases}$
- Falls  $A = (A' \Rightarrow B')$ , so ist  $J(A) := \begin{cases} 0 & \text{falls } J(A') = L \text{ und } J(B') = 0 \\ L & \text{sonst} \end{cases}$

- Falls  $A = (A' \Leftrightarrow B')$ , so ist  $J(A) := \begin{cases} L & \text{falls } J(A') = J(B') \\ 0 & \text{sonst} \end{cases}$

Um prädikatenlogische Ausdrücke behandeln zu können, wird die Definition von  $J$  erweitert:

- $J(Q(x_1, \dots, x_n)) := \begin{cases} L & \text{falls } Q \text{ gilt für } x_1, \dots, x_n \in U \\ 0 & \text{sonst} \end{cases}$
- $J((\forall x)A) := \begin{cases} L & \text{falls } J(A) = L \text{ für jedes } x \in U \\ 0 & \text{sonst} \end{cases}$
- $J((\exists x)A) := \begin{cases} L & \text{falls } J(A) = L \text{ für mindestens ein } x \in U \\ 0 & \text{sonst} \end{cases}$

Mit diesen Definitionen sehen die Lösungen wie folgt aus:

$$(a) J((\exists m)(\forall n) \quad m > n) = \begin{cases} L & \text{falls } J((\forall n) \quad m > n) = L \text{ für mindestens ein } m \in \mathbb{N} \\ 0 & \text{sonst} \end{cases}$$

Gibt es ein  $m$  mit  $J((\forall n)m > n) = L$ ? Dazu müssen wir den Wahrheitswert von  $(\forall n)m > n$  berechnen:

$$J((\forall n)m > n) = \begin{cases} L & \text{falls } J(m > n) = L \text{ für jedes } n \in \mathbb{N} \\ 0 & \text{sonst} \end{cases}$$

Für  $n = m$  ist  $J(m > n) = 0$ . Also gilt  $J((\exists m)(\forall n) \quad m > n) = 0$ .

In Worten ausgedrückt bedeutet die Aussage „Es gibt eine natürliche Zahl, die größer als alle natürlichen Zahlen ist“.

$$(b) J((\forall m)(\exists n) \quad m < n) = \begin{cases} L & \text{falls } J((\exists n) \quad m < n) = L \text{ für jedes } m \in \mathbb{N} \\ 0 & \text{sonst} \end{cases}$$

Gilt für jedes  $m$   $J((\exists n) \quad m < n) = L$ ? Dazu müssen wir den Wahrheitswert von  $(\exists n) \quad m < n$  berechnen:

$$J((\exists n)m < n) = \begin{cases} L & \text{falls } J(m < n) = L \text{ für mindestens ein } n \in \mathbb{N} \\ 0 & \text{sonst} \end{cases}$$

Für  $n = m + 1$  gilt  $J(m < n) = L$ . Also gilt  $J((\forall m)(\exists n) \quad m < n) = L$ .

Diese Aussage bedeutet „Zu jeder natürlichen Zahl gibt es eine größere natürliche Zahl“, oder auch „ $\mathbb{N}$  ist unbeschränkt“.

$$(c) J((\exists m)m \geq 2 \wedge (\forall n)n \geq 2 \wedge m \neq n \wedge \neg(\exists q) \quad m = n \cdot q) = \begin{cases} L & \text{falls } J(m \geq 2 \wedge (\forall n)n \geq 2 \wedge m \neq n \wedge \neg(\exists q) \quad m = n \cdot q) = L \\ & \text{für mindestens ein } m \in \mathbb{N} \\ 0 & \text{sonst} \end{cases}$$

Gibt es ein  $m$  mit  $J(m \geq 2 \wedge (\forall n)n \geq 2 \wedge m \neq n \wedge \neg(\exists q) \quad m = n \cdot q) = L$ ?

Mit der Definition von  $J$  ergibt sich

$$J(\dots) = \begin{cases} L & \text{falls } J(m \geq 2) = J((\forall n)n \geq 2 \wedge m \neq n \wedge \neg(\exists q) \quad m = n \cdot q) = L \\ 0 & \text{sonst} \end{cases}$$

Für  $m > 2$  gilt offenkundigerweise  $J(m \geq 2) = L$ .

Um  $J((\forall n)n \geq 2 \wedge m \neq n \wedge \neg(\exists q) \quad m = n \cdot q)$  zu berechnen, verwenden wir wieder die Definition von  $J$ :

$$J(\dots) = \begin{cases} L & \text{falls } J(n \geq 2 \wedge m \neq n \wedge \neg(\exists q) \quad m = n \cdot q) = L \text{ f\u00fcr jedes } n \in \mathbb{N} \\ 0 & \text{sonst} \end{cases}$$

Hier m\u00fcbte man wieder die Definition von  $J$  einsetzen, doch sieht man auch so schon, da\u00df  $n \geq 2$  nicht f\u00fcr jedes  $n$  richtig ist. Der Wahrheitswert ist deshalb 0, und damit ist auch der Wahrheitswert der urspr\u00fcnglichen Aussage gleich 0.

Diese Aussage bedeutet: „Es gibt eine nat\u00fcrliche Zahl  $m$  f\u00fcr die gilt:  $m$  ist gr\u00f6\u00dfer als 2 und f\u00fcr alle nat\u00fcrlichen Zahlen  $n$  gilt:  $n$  ist gr\u00f6\u00dfer als 2 und  $m$  ist ungleich  $n$  und  $n$  ist kein Teiler von  $m$ .“

### Aufgabe 2 (\*\*):

Beweisen Sie mit Hilfe der in der Vorlesung angegebenen Auswertungsregeln f\u00fcr pr\u00e4dikatenlogische Ausdr\u00fccke: Ist  $Q$  ein einstelliges Pr\u00e4dikat, so gilt  $(\exists x)\neg Q(x) \Leftrightarrow \neg(\forall x)Q(x)$

Zeigen Sie dazu zun\u00e4chst, da\u00df man aus der linken Seite der \u00c4quivalenzaussage ihre rechte Seite ableiten kann. Beweisen Sie die andere Richtung in einem zweiten Schritt.

### L\u00f6sung zu Aufgabe 2:

$$\begin{aligned} J(\neg(\forall x)Q(x)) = L &\Rightarrow J((\forall x)Q(x)) = 0 \\ &\Rightarrow \text{nicht f\u00fcr jedes } x \in U \text{ gilt } J(Q(x)) = L \\ &\Rightarrow \text{es gibt ein } x \in U \text{ mit } J(Q(x)) = 0 \\ &\Rightarrow \text{es gibt ein } x \in U \text{ mit } J(\neg Q(x)) = L \\ &\Rightarrow J((\exists x)\neg Q(x)) = L \end{aligned}$$

$$\begin{aligned} J((\exists x)\neg Q(x)) = L &\Rightarrow \text{f\u00fcr mindestens ein } x \in U \text{ gilt } J(\neg Q(x)) = L \\ &\Rightarrow \text{f\u00fcr mindestens ein } x \in U \text{ gilt } J(Q(x)) = 0 \\ &\Rightarrow \text{nicht f\u00fcr jedes } x \in U \text{ gilt } J(Q(x)) = L \\ &\Rightarrow \text{es gilt nicht } J((\forall x)Q(x)) = L \\ &\Rightarrow J((\forall x)Q(x)) = 0 \\ &\Rightarrow J(\neg(\forall x)Q(x)) = 0 \end{aligned}$$

### Aufgabe 3 (\*\*):

Beweisen Sie: F\u00fcr zwei endliche Mengen  $A$  und  $B$  gilt  $|A \cup B| = |A| + |B| - |A \cap B|$ .

Beachten Sie dazu, da\u00df man eine Menge  $B$  mit  $|B| = n$  als  $B = \{b_1, b_2, \dots, b_n\} = \{b_1\} \cup \{b_2\} \cup \dots \cup \{b_n\}$  schreiben kann, wobei  $(\forall i \in \mathbb{N})(\forall j \in \mathbb{N}) \quad i \leq n \wedge j \leq n \wedge i \neq j \Rightarrow b_i \neq b_j$  gilt. Beweisen Sie die Aussage durch vollst\u00e4ndige Induktion \u00fcber  $n$ .

### Lösung zu Aufgabe 3:

Zunächst beweisen wir mit vollständiger Induktion, daß für  $A \cap B = \emptyset$  gilt  $|A \cup B| = |A| + |B|$ :

**Induktionsbasis:**  $B = \emptyset$ , also  $|B| = 0$ : Dann gilt  $A \cup B = A$  und damit  $|A \cup B| = |A| = |A| + |B|$ .

**Induktionsschritt:** Induktionsannahme ist, daß die Aussage  $|A \cup B| = |A| + |B|$  für alle Mengen  $B$  mit  $|B| = n$  gilt. Wir zeigen, daß die Aussage dann auch für alle Mengen  $B'$  mit  $|B'| = n + 1$  gilt.

Zunächst schreiben wir  $B'$  in der Form  $B' = B \cup \{b\}$  mit  $b \notin B$ . Dann gilt wegen  $A \cup (B \cup \{b\}) = (A \cup B) \cup \{b\}$  mit dreimaliger Anwendung der Induktionsannahme  $|A \cup (B \cup \{b\})| = |A \cup B| + |\{b\}| = |A| + |B| + |\{b\}| = |A| + |B \cup \{b\}|$ .

Mit diesem Zwischenergebnis kann man nun das eigentlich gesuchte Resultat (wiederum mit vollständiger Induktion) beweisen:

**Induktionsbasis:**  $B = \emptyset$ : Dann gilt wegen des obigen Zwischenergebnisses  $|A \cup B| = |A| + |B| + |A \cap B|$ .

**Induktionsschritt:** Induktionsannahme ist, daß die Aussage  $|A \cup B| = |A| + |B| + |A \cap B|$  für alle Mengen  $B$  mit  $|B| = n$  gilt. Wir zeigen, daß die Aussage dann auch für alle Mengen  $B'$  mit  $|B'| = n + 1$  gilt. Zunächst schreiben wir  $B'$  in der Form  $B' = B \cup \{b\}$  mit  $b \notin B$ .

Unter Verwendung der Induktionsannahme kann man  $|A \cup B'| = |(A \cup B) \cup \{b\}| = |(A \cup B)| + |\{b\}| - |(A \cup B) \cap \{b\}| = (1)$  ableiten.

Eine zweite Anwendung der Induktionsannahme und des Distributivgesetzes ergibt  $(1) = |A| + |B| - |A \cap B| + |\{b\}| - |A \cap \{b\} \cup \emptyset|$ , was sich zu  $|A| + |B| + |\{b\}| - |A \cap B| - |A \cap \{b\}| = (2)$  vereinfachen läßt. Zweimalige Anwendung des Zwischenergebnisses von oben schließlich ergibt  $(2) = |A| + |B'| - (|A \cap B| + |A \cap \{b\}|) = |A| + |B'| - ||A \cap B \cup A \cap \{b\}||$ , was sich dann mit Hilfe des Distributivgesetzes zu  $|A| + |B'| - |A \cap B'|$  vereinfachen läßt.

### Aufgabe 4 (\*\*\*):

- Beweisen Sie durch vollständige Induktion, daß Ihre rekursive Scheme-Funktion aus Aufgabe 5 korrekt ist.
- Beweisen Sie auch die Korrektheit der zweiten Funktion aus Aufgabe 5.

### Lösung zu Aufgabe 4:

- Zu zeigen ist, daß

$$(\text{multiply } m \ n) = m \cdot n \quad \forall (m, n) \in \mathbb{N}^2$$

**Induktionsanfang:** Für  $n = 0$  ist diese Aussage wahr. (Das folgt aus dem Wert des if-Ausdrucks im Substitutionsmodell von Scheme.)

**Induktionsschritt:** Die Induktionsannahme ist, daß die Aussage für  $n$  gilt. Wir beweisen, daß sie dann auch für  $n + 1$  gilt:  $(\text{multiply } m \ (+ \ n \ 1)) = (+ \ m \ (\text{multiply } m \ (- \ (+ \ n \ 1) \ 1))) = (+ \ m \ (\text{multiply } m \ n)) = (+ \ m \ (* \ m \ n)) = (* \ m \ (+ \ n \ 1))$ .

(Die erste Umformung geschieht wieder mit Hilfe des Substitutionsmodells, die dritte durch Anwendung der Induktionsannahme, die anderen sind eine einfache algebraische Umformungen.)

(b) Zu zeigen ist, daß

$$(\text{multiply-it-1 } m \ n \ a) = m \cdot n + a \quad \forall (m, n, a) \in \mathbb{N}^3$$

**Induktionsanfang:** Für  $n = 0$  ist diese Aussage wahr. (Das folgt aus dem Wert des `if`-Ausdrucks im Substitutionsmodell von `Scheme`.)

**Induktionsschritt:** Die Induktionsannahme ist, daß die Aussage für  $n$  gilt. Wir beweisen, daß sie dann auch für  $n+1$  gilt:  $(\text{multiply-it-1 } m \ (+ \ n \ 1) \ a) = (\text{multiply-it-1 } m \ (- \ (+ \ n \ 1) \ 1) \ (+ \ \text{acc } m)) = (\text{multiply-it-1 } m \ n \ (+ \ \text{acc } m)) = (+ \ (* \ m \ n) \ (+ \ \text{acc } m)) = (+ \ (* \ m \ (+ \ n \ 1)) \ \text{acc})$ .

(Die erste Umformung geschieht wieder mit Hilfe des Substitutionsmodells, die dritte durch Anwendung der Induktionsannahme, die anderen sind eine einfache algebraische Umformungen.)

### Aufgabe 5 (Programmierung, \*\*):

Für die Multiplikation zweier natürlicher Zahlen gilt:

$$\begin{aligned} m \cdot 0 &:= 0 \\ m \cdot (n + 1) &:= m + m \cdot n \end{aligned}$$

- Geben Sie eine rekursive `Scheme`-Funktion (ähnlich der Funktion `factorial` aus der Vorlesung) an, die diese Definition verwendet, um zwei natürliche Zahlen zu multiplizieren, ohne auf die eingebaute Multiplikationsfunktion zurückzugreifen.
- Schreiben Sie nun eine iterative `Scheme`-Funktion (ähnlich der Funktion `it-factorial` aus der Vorlesung), die dasselbe bewerkstelligt.

### Lösung zu Aufgabe 5:

```
;;; SIGNATUR
;;; multiply: number number -> number
;;; ERKLÄRUNG
;;; (multiply m n) berechnet das Produkt der natürlichen Zahlen m und n
;;; BEISPIEL
;;; (multiply 2 3)
;;; => 6
;;; DEFINITION
(define multiply
  (lambda (m n)
    (if (= n 0)
        0
        (+ m (multiply m (- n 1))))))

;;; SIGNATUR
```

```

;;; multiply-it: number number -> number
;;; ERKLÄRUNG
;;; (multiply-it m n) berechnet das Produkt der natürlichen Zahlen m und n
;;; BEISPIEL
;;; (multiply-it 2 3)
;;; => 6
;;; DEFINITION
(define multiply-it
  (lambda (m n)
    (multiply-it-1 m n 0)))

;;; SIGNATUR
;;; multiply-it-1: number number number -> number
;;; ERKLÄRUNG
;;; (multiply-it m n acc) berechnet die Summe des Produktes der natürlichen
;;; Zahlen m und n sowie der Zahl acc.
;;; BEISPIEL
;;; (multiply-it-1 2 3 0)
;;; => 6
;;; DEFINITION
(define multiply-it-1
  (lambda (m n acc)
    (if (= n 0)
        acc
        (multiply-it-1 m (- n 1) (+ acc m)))))

```

### Aufgabe 6 (Programmierung, \*\*\*):

- Besorgen Sie sich den Scheme-Code aus der Vorlesung vom 31.10. (Teil K5) und erweitern Sie diese Datei um eine Datenstruktur für Dreiecke. Ein Dreieck soll durch seine Seitenlängen  $a, b$  und  $c$  und seine linke untere Ecke beschrieben werden. Um das Dreieck dadurch eindeutig zu charakterisieren, legen wir fest, daß Seite mit der Länge  $c$  waagrecht liegt und alle Punkte des Dreiecks nicht unterhalb dieser Seite liegen. Die linke untere Ecke des Dreiecks ist dann der am weitesten links liegende Punkt dieser Seite. Weiter soll die Benennung der Seiten mit  $a, b$  und  $c$  im Uhrzeigersinn erfolgen.
- Schreiben Sie eine Funktion `triangle-area` mit der Signatur `triangle → number`, die einem Dreieck seinen Flächeninhalt zuordnet.
- Schreiben Sie eine Funktion `triangle-move` mit der Signatur `triangle posn → triangle`, die eine verschobene Kopie des Dreiecks zurückliefert.

### Lösung zu Aufgabe 6:

```
(load "constr-03.ss")
```

```

;;; Datentyp "triangle"
;;; Ein "polygon" ist eine Datenstruktur (make-triangle origin a b c), wobei
;;; origin den Ort der linken unteren Ecke beschreibt und a, b und c die
;;; Seitenlängen sind.
(define-struct triangle (origin a b c))

;;; SIGNATUR
;;; triangle-area: triangle -> number
;;; ERKLÄRUNG
;;; (triangle-area t) berechnet den Flächeninhalt des Dreiecks t.
;;; Dazu wird die Heronsche Flächenformel verwandt.
;;; BEISPIEL
;;; (triangle-area (make-triangle (make-posn 0 0) 3 4 5))
;;; => 6
;;; DEFINITION
(define triangle-area
  (lambda (t)
    (let ((a (triangle-a t)) (b (triangle-b t)) (c (triangle-c t)))
      (let ((s (/ (+ a b c) 2)))
        (sqrt (* s (- s a) (- s b) (- s c)))))))

;;; SIGNATUR
;;; triangle-move: triangle posn -> triangle
;;; ERKLÄRUNG
;;; (triangle-move t p) gibt eine um p verschobene Kopie von t zurück.
;;; BEISPIEL
;;; (triangle-move (make-triangle (make-posn 0 0) 3 4 5) (make-posn 1 2))
;;; => (make-triangle (make-posn 1 2) 3 4 5)
;;; DEFINITION
(define triangle-move
  (lambda (t p)
    (make-triangle (posn-move (triangle-origin t) p)
                   (triangle-a t)
                   (triangle-b t)
                   (triangle-c t))))

```

### Aufgabe 7 (Programmierung, \*\*):

Unter einer Bounding-Box eines geometrischen Objekts versteht man das kleinste Rechteck, das alle Punkte des Objekts enthält.

Erweitern Sie den Scheme-Code aus der vorigen Aufgabe durch Scheme-Funktionen `circle-bounding-box`, `rectangle-bounding-box` und `triangle-bounding-box` mit den Signaturen `circle`  $\rightarrow$  `rectangle`, `rectangle`  $\rightarrow$  `rectangle` und `triangle`  $\rightarrow$  `rectangle`, die die Bounding-Box ihres Arguments zurückliefern. Die Bounding-Box soll dabei in jedem Fall eine neu angelegte Datenstruktur sein.

## Lösung zu Aufgabe 7:

```
(load "ub2-6.scm")

;;; SIGNATUR
;;; circle-bounding-box: circle -> rectangle
;;; ERKLÄRUNG
;;; (circle-bounding-box c) liefert die Bounding-Box des Kreises c zurück.
;;; BEISPIEL
;;; (circle-bounding-box (make-circle (make-posn 0 0) 1))
;;; => (make-rectangle (make-posn -1 -1) (make-posn 2 2))
;;; DEFINITION
(define circle-bounding-box
  (lambda (c)
    (let ((x (posn-x (circle-origin c)))
          (y (posn-y (circle-origin c)))
          (r (circle-radius c)))
      (make-rectangle (make-posn (- x r) (- y r))
                     (make-posn (* 2 r) (* 2 r))))))

;;; SIGNATUR
;;; rectangle-bounding-box: rectangle -> rectangle
;;; ERKLÄRUNG
;;; (rectangle-bounding-box r) liefert die Bounding-Box von r zurück.
;;; BEISPIEL
;;; (rectangle-bounding-box (make-rectangle (make-posn 0 0) (make-posn 2 3)))
;;; => (make-rectangle (make-posn 0 0) (make-posn 2 3))
;;; DEFINITION
(define rectangle-bounding-box
  (lambda (r)
    (rectangle-move r (make-posn 0 0))))

;;; SIGNATUR
;;; triangle-bounding-box: triangle -> rectangle
;;; ERKLÄRUNG
;;; (triangle-bounding-box t) liefert die Bounding-Box von Dreieck t zurück.
;;; BEISPIEL
;;; Linksüberhängend:
;;; (triangle-bounding-box (make-triangle (make-posn 0 0) 2 3 2))
;;; (make-rectangle (make-posn -0.25 0) (make-posn 2.25 #i1.984313483298443))
;;; Rechtsüberhängend:
;;; (triangle-bounding-box (make-triangle (make-posn 0 0) 3 2 2))
;;; => (make-rectangle (make-posn 0 0) (make-posn 2.25 #i1.984313483298443))
;;; Nicht überhängend:
;;; (triangle-bounding-box (make-triangle (make-posn 0 0) 2 2 2))
```

```

;;; => (make-rectangle (make-posn 0 0) (make-posn 2 #i1.7320508075688772))
;;; DEFINITION
(define triangle-bounding-box
  (lambda (t)
    (let ((x (posn-x (triangle-origin t))) (y (posn-y (triangle-origin t)))
          (a (triangle-a t)) (b (triangle-b t)) (c (triangle-c t)))
      (let ((cos-alpha (/ (- (+ (* b b) (* c c)) (* a a)) (* 2 b c)))
            (cos-beta (/ (- (+ (* a a) (* c c)) (* b b)) (* 2 a c))))
        (let ((hc (* a (sqrt (- 1 (* cos-beta cos-beta)))))
              (if (< cos-beta 0)
                  (let ((cprime (* a (- cos-beta))))
                    (make-rectangle (make-posn (- x cprime) y)
                                     (make-posn (+ cprime c) hc)))
                  (if (< cos-alpha 0)
                      (let ((cprime (* b (- cos-alpha))))
                        (make-rectangle (make-posn x y)
                                         (make-posn (+ cprime c) hc)))
                      (make-rectangle (make-posn x y)
                                       (make-posn c hc))))))))))

```

Von diesen Funktionen bedarf nur `triangle-bounding-box` eine nähere Erklärung. Zunächst vereinbaren wir, daß mit  $\alpha$  ( $\beta$ ,  $\gamma$ ) der der Seite  $a$  ( $b$ ,  $c$ ) gegenüberliegende Winkel bezeichnet wird. Wir bezeichnen mit  $(x, y)$  die Koordinaten des am weitesten links liegenden Punktes der Dreieckseite  $c$ .

Um die Aufgabe lösen zu können, unterscheidet man zwischen linksüberhängenden, rechtsüberhängenden und nichtüberhängenden Dreiecken.

**Linksüberhängende Dreiecke:** Wir verwenden die trigonometrische Gleichung

$$\cos \beta = \frac{a^2 + c^2 - b^2}{2ac},$$

um festzustellen, ob das Dreieck linksüberhängend ist. Das ist dann der Fall, wenn  $\beta > 90^\circ$  oder  $\cos \beta < 0$  ist.

Dann ist die Breite der Bounding-Box zusammen aus  $c$  und dem Überhang  $c' = a \cos(180^\circ - \beta) = -a \cos \beta$ .

Die Koordinaten der linken unteren Ecke der Bounding-Box sind  $(x - c', y)$ .

**Rechtsüberhängende Dreiecke:** Wir verwenden die trigonometrische Gleichung

$$\cos \alpha = \frac{b^2 + c^2 - a^2}{2bc},$$

um festzustellen, ob das Dreieck rechtsüberhängend ist. Das ist dann der Fall, wenn  $\alpha > 90^\circ$  oder  $\cos \alpha < 0$  ist.

Dann ist die Breite der Bounding-Box zusammen aus  $c$  und dem Überhang  $c' = b \cos(180^\circ - \alpha) = -b \cos \beta$ .

Die Koordinaten der linken unteren Ecke der Bounding-Box sind  $(x, y)$ .

**Nichtüberhängende Dreiecke:** Wenn das Dreieck weder links- noch rechtsüberhängend ist, ist es nichtüberhängend.

In diesem Fall ist die Breite der Bounding-Box gleich der Grundseite  $c$  des Dreiecks.

Die Koordinaten der linken unteren Ecke der Bounding-Box sind wieder  $(x, y)$ .

Die Höhe der Bounding-box ist in allen drei Fällen gleich der Höhe  $h_c$  des Dreiecks auf  $c$ :

$$h_c = a \sin \beta = a \sqrt{1 - \cos^2 \beta},$$

wobei wir die bekannte Formel  $\sin^2 \beta + \cos^2 \beta = 1$  verwendet haben.