Fakultät IV Elektrotechnik und Informatik

Einführung in die Programmierung WS 2019/2020

Manfred Hauswirth / Damien Foucard / Christoph Seifert

Aufgabenblatt 8

Ausgabe: *10.01.2020*
Abgabe: *21.01.2020 09:59*

Thema: *Korrektheit*

Abgabemodalitäten

1. Alle abzugebenden Quelltexte müssen ohne Warnungen und Fehler auf den Rechnern des ZE-CM/eecsIT mittels `clang -std=c11 -Wall -g` kompilieren.
2. Die Abgabe erfolgt ausschließlich über unser SVN im Abgaben-Ordner. Nur wenn ein Test in Osiris angezeigt wird ist sichergestellt, dass die Abgabe erfolgt ist.
3. Du kannst bis zur Abgabefrist beliebig oft neue Versionen abgeben.
4. Die Abgabe erfolgt in folgendem Unterordner:
   `introprog-wise1920/Studierende/<L>/<TU-Login>/Abgaben/Blatt<XX>`
   wobei `<L>` durch den ersten Buchstabe des TU-Logins und `<XX>` durch die Nummer des Aufgabenblattes zu ersetzen sind. Die Ordner werden automatisch angelegt sobald die Abgabe freigeschaltet wird.
5. Du darfst den Abgabeordner für das Blatt nicht selbst erstellen. Die Ordner erstellen wir kurz nach der Ausgabe der Aufgabe.
6. Um die Änderungen (z. B. neue Abgabeordner) vom Server abzuholen, musst Du den Befehl `svn update` im Wurzelverzeichnis (also im Verzeichnis `introprog-wise1920`) des Repositories ausführen.
7. Im Abgabeordner gelten einige restriktive Regeln. Dort ist nur das Einchecken von Dateien mit den in der Aufgabe vorgegebenen Namen erlaubt, ausserdem werden die Abgabefristen vom Server überwacht. Beachte eventuelle Fehlermeldungen beim SVN-Commit. Lade nur Dateien hoch, die Du selbst bearbeiten sollst, insbesondere also keine Vorgaben.
8. Der Dateiname bei Abgaben mit C-Code soll dem der Vorgaben entsprechen. Entferne ausschließlich `_vorgabe` aus dem Name. Bei Textaufgaben sind, wenn nicht anders angegeben, `.txt` Dateien zugelassen, die als Plaintext-Datei zu speichern sind (keine Word-Dateien umbenennen).
9. Es gibt einen Ordner `Arbeitsverzeichnis`, in dem Du Dateien für Dich ablegen kannst.
10. Die Ergebnisse der automatischen Tests kannst Du auf OSIRIS einsehen:
    https://osiris.ods.tu-berlin.de/

Führen von Korrektheitsbeweisen (Aufbereitung von Teilen des Vorlesungsinhalts)

In diesem Abschnitt werden die in der Vorlesung vorgestellten Grundlagen von Korrektheitsbeweisen aufgegriffen und zum Teil stärker formalisiert. Das folgende Material soll insbesondere die Beweisstruktur für Schleifeninvarianten aufgreifen, welche in den darauf folgenden Übungsaufgaben benutzt werden.

Informale Einleitung

Bei Korrektheitsbeweisen von Algorithmen geht man im Allgemeinen so vor, dass man den Programmcode von oben nach unten durchgeht und den Zustand nach jedem ausgeführten Befehl mathematisch formal beschreibt. In diesem Abschnitt gehen wir kurz auf das Material aus der Vorlesung ein und verdeutlichen wie Korrektheitsbeweise geführt werden.

Das Beispiel aus Listing 1 soll uns kurz dazu dienen, eine sequentielle Ausführung zu beschreiben. Dem einfachen Algorithmus wird eine Zahl n übergeben, addiert auf diese 10 und gibt den entsprechenden Wert zurück.

Listing 1: Minimales sequentielles Programm

```
1 Add10(n)
2   X ← 10
3   Y ← n
4   Y ← X + Y
5   return Y
```

Die Korrektheit des Algorithmus Add10 kann wie folgt beschrieben werden: $\forall n \in \mathbb{R}.\text{Add10}(n) = n + 10$.

Der Korrektheitsbeweis dieses Algorithmus ist denkbar einfach: Sei $n \in \mathbb{R}$ eine beliebige Zahl. Nach der Ausführung der Zeile 2 gilt $X = 10$. Nach der Ausführung von Zeile 3 gilt $Y = n$. In Zeile 4 wird nun Y auf $X + Y$ gesetzt. Da wir wissen, dass $Y = n$ ist, erhalten wir somit $Y = n + 10$. Da Y zurückgegeben wird, ist $\text{Add10}(n) = n + 10$. Da n beliebig gewählt wurde, gilt die Aussage für alle Eingaben n und die Korrektheit ist somit bewiesen. Beachtet, dass dieser Beweis darauf beruht, den Zustand formal in jeder Zeile zu beschreiben. Der Zustand nach der Ausführung eines Befehls ergibt sich dabei einfach aus dem vorherigen Zustand und der Auswirkung des gerade ausgeführten Befehls.

Betrachten wir nun den Algorithmus der Max-Suche in Listing 2, welcher eine for-Schleife enthält. Während bei dem Algorithmus `Add10` nach jedem Befehl der Zustand der Variablen direkt angegeben werden konnte, ist dies für die Max-Suche (siehe Listing 2) nicht mehr so einfach, da hier eine for-Schleife verwendet wird. Bei jedem Durchlauf der Zeile 4 aus Listing 2 hat j einen anderen Wert und eine Beschreibung des Zustands muss daher auch die verschiedenen Werte von j reflektieren. Um den Zustand innerhalb (und nach) Schleifen zu beschreiben finden daher sogenannte Schleifeninvarianten Verwendung. Die Schleifeninvariante der for-Schleife in Zeilen 3-4 des Listings 2 lautet wie in der

Vorlesung besprochen $A[max]$ *ist ein größtes Element aus* $A[1, 2, \ldots, j-1]$*)* und beinhaltet somit die Laufvariable in der Quantifizierung der betrachteten Elemente: in jedem Schleifendurchlauf ist $A[max]$ eines der größten Elemente der ersten $j-1$ Elemente des Arrays.

Listing 2: Max-Search Algorithmus zum Finden des Indizes des größten Elements eines Arrays

```
1 Max-Search(Array A)
2   max ← 1
3   for j ← 2 to length(A) do
4     if A[j] > A[max] then max ← j
5   return max
```

Korrektheitsbeweis des Max-Suche Algorithmus / Schleifeninvarianten

Im Folgenden wird der in der Vorlesung vorgestellte Beweis des Max-Suche Algorithmus erneut behandelt. Dabei formalisieren wir den Beweis stärker als in der Vorlesung, damit die einzelnen Aussagen genauer nachvollzogen werden können. Weiterhin gehen wir insbesondere auf die Beweisstruktur der Schleifeninvarianten ein, welche bei dem Beweis benutzt wird. Gerade auf Grund etwaiger Fallstricke von Schleifeninvarianten – nämlich der Frage, wann eine solche genau zu gelten habe – ist das Studium des Materials besonders wichtig für das Verständnis von Korrektheitsbeweisen.

Wie in der Vorlesung beschrieben ist die folgende Aussage zu beweisen: $A[\text{Max-Search}(A)]$ ist ein größtes Element des Arrays A. Formal kann diese Aussage wie folgt ausgedrückt werden:

$$\forall k \in \{1, \ldots, \text{length}(A)\}.A[\text{Max-Search}(A)] \geq A[k] \wedge \text{Max-Search}(A) \in \{1, 2, \ldots, \text{length}(A)\}\ . \tag{1}$$

Somit ist $A[\text{Max-Search}(A)]$ größer als alle in A enthaltenen Elemente und gleichzeitig zeigt der Rückgabewert immer auf einen gültigen Index und somit auf ein Element im Array.

Die Idee des Beweises der Korrektheit besteht im Wesentlichen in der Erkenntnis, dass die Variable max in jedem Schleifendurchlauf auf das Maximum des Teilarrays $A[1, \ldots, j-1]$ zeigt. Nach der Analyse des Zustands nach der Ausführung von Zeile 2 wird diese Intuition mittels einer Schleifeninvariante formal beschrieben und die Gültigkeit stringent bewiesen.

Zunächst stellen wir bzgl. des Zustands nach der Ausführung der Zeile 2 fest, dass $max = 1$ gilt. Für die Zeilen 3-4 bedarf es wie oben beschrieben einer Schleifeninvarianten, um den Zustand in Abhängigkeit der Laufvariablen zu beschreiben. Die oben ausgedrückte Intuition, dass max auf das größte Element des Arrays $A[1, \ldots, j-1]$ zeigt, kann hierbei durch $\forall k \in \{1, \ldots, j-1\}.A[max] \geq A[k]$ formalisiert werden. Da weiterhin zu zeigen ist, dass max auch immer auf einen Index im Array A zeigt (siehe Formel (1)), benutzen wir des Weiteren die Aussage $max \in \{1, \ldots, j-1\}$. Wir erhalten somit als Schleifeninvariante (deren Gültigkeit noch zu zeigen ist):

$$\forall k \in \{1, \ldots, j-1\}.A[max] \geq A[k] \wedge max \in \{1, \ldots, j-1\} \tag{2}$$

Wir werden die Korrektheit dieser Schleifeninvariante (2) im Folgenden zeigen. Die Gültigkeit wird hierbei induktiv gezeigt, d.h., dass die Invariante bei der ersten Ausführung gegolten hat und nach jedem Schleifendurchlauf ihre Gültigkeit erhalten bleibt:

Initiale Gültigkeit der Invariante / Induktionsanfang
Es ist zu überprüfen, dass die Schleifeninvariante beim erstmaligen Betreten der Schleife gilt. Gemäß der Zuweisung aus Zeile 2 gilt $max = 1$. Die Schleifenvariable j wird mit 2 initialisiert. Die Schleifeninvariante lautet für den Wert $j = 2$:

$$\forall k \in \{1, \ldots, 1\}.A[max] \geq A[k] \wedge max \in \{1, \ldots, 1\}$$

Die Beschreibung der Menge $\{1, \ldots, 1\}$ vereinfacht sich zur Menge $\{1\}$ und somit erhalten wir:

$$\forall k \in \{1\}.A[max] \geq A[k] \wedge max \in \{1\}$$

Da $max = 1 \in \{1\}$ gilt und $A[1] \geq A[1]$ ist, gilt die Invariante initial.

Erhalt der Gültigkeit der Invariante / Induktionsschluss Wir nehmen nun an, dass die Invariante für einen beliebigen Wert $j = j_0 \in \{2, \ldots, \text{length}(A)\}$ gilt (Induktionsvoraussetzung) und zeigen, dass die Schleifeninvariante auch noch nach der Ausführung des Schleifenkörpers (Zeile 4) und nach der Inkrementierung der Variablen j – also für $j = j_0 + 1$ gilt.
Vor der Ausführung des Schleifenkörpers (Zeile 4) gilt gemäß unserer Annahme die Schleifeninvariante für ein $j_0 \in \{2, \ldots, \text{length}(A)\}$. Konkret gilt somit:

$$j = j_0 \wedge j_0 \in \{2, \ldots, \text{length}(A)\} \wedge \forall k \in \{1, \ldots, j_0 - 1\}.A[max] \geq A[k] \wedge max \in \{1, \ldots, j_0 - 1\}$$

Zeile 4 enthält eine if-Abfrage und wir untersuchen beide Fälle (Ausführung bzw. Nichtausführung der Zuweisung $max \leftarrow j$) separat:

Fall 1 $A[j] > A[max]$
In diesem Fall wird max auf j gesetzt. Es gilt nun folgendes:

- $\forall k \in \{1, \ldots, j_0\}.A[max] \geq A[k]$. Dies gilt, da (nach der Annahme zuvor) schon $\forall k \in \{1, \ldots, j_0 - 1\}.A[max] \geq A[k]$ gegolten hat und das aktuelle Element $A[j]$ gemäß der Fallunterscheidung größer als $A[max]$ war.
- $max \in \{1, \ldots, j_0\}$. Dies gilt, da $max = j = j_0$ gilt.
- Somit gilt die Schleifeninvariante auch für den Wert $j = j_0 + 1$ (d.h. nach Inkrementierung der Zählvariablen j).

Fall 2 $A[j] \leq A[max]$
In diesem Fall wird kein Befehl ausgeführt. Wir zeigen wiederum, dass auch in diesem Fall die Schleifeninvariante erhalten bleibt.

- Es gilt auch weiterhin $\forall k \in \{1, \ldots, j_0\}.A[max] \geq A[k]$. Dies gilt, da (nach der Annahme zuvor) schon $\forall k \in \{1, \ldots, j_0 - 1\}.A[max] \geq A[k]$ gegolten hat und das aktuelle Element $A[j]$ gemäß der Fallunterscheidung kleiner gleich $A[max]$ war.

- Da die Variable max nicht angepasst wurde und zuvor schon $max \in \{1, \ldots, j_0 - 1\}$ gegolten hat, gilt nun auch sicherlich $max \in \{1, \ldots, j_0 - 1, j_0\}$.
- Somit bleibt auch in diesem Fall die Schleifeninvariante erhalten.

Da in beiden Fällen die Schleifeninvariante erhalten bleibt, das heißt auch noch für $j = j_0 + 1$ gilt, gilt die Schleifeninvariante vor und nach jedem Schleifendurchlauf.

Aus der Gültigkeit der Schleifeninvariante kann nun auf den Zustand nach der Ausführung der Schleife geschlossen werden, da die Schleifeninvariante vor jeder Ausführung gilt. Da bei for-Schleifen die Inkrementierung der Laufvariablen am Ende des Schleifenkörpers vorgenommen wird und erst dann verglichen wird, ob die Variable kleiner gleich dem Maximum ist, wird die Schleife bei dem Wert $j = \text{length}(A) + 1$ verlassen. Da die Schleifeninvariante immer vor der Ausführung des Schleifenkörpers gilt und für $j = \text{length}(A)+1$ die Schleife verlassen wird (die nächste auszuführende Zeile ist Zeile 5), erhalten wir durch Einsetzen von $j = \text{length}(A) + 1$ in die Schleifeninvariante die folgende Aussage vor der Ausführung von Zeile 5:

$$\forall k \in \{1, \ldots, \underbrace{\text{length}(A) + 1}_{j} - 1\}.A[max] \geq A[k] \wedge max \in \{1, \ldots, \underbrace{\text{length}(A) + 1}_{j} - 1\}$$

und somit

$$\forall k \in \{1, \ldots, \text{length}(A)\}.A[max] \geq A[k] \wedge max \in \{1, \ldots, \text{length}(A)\} .$$

Da in Zeile 5 der Wert von max zurückgegeben wird, erhalten wir die Korrektheit (vgl. Aussage 1 auf Seite 3) des Algorithmus:

$$\forall k \in \{1, \ldots, \text{length}(A)\}.A[\text{Max-Search}(A)] \geq A[k] \wedge \text{Max-Search}(A) \in \{1, \ldots, \text{length}(A)\} .$$

Bemerkungen zum obigen Korrektheitsbeweis

Gemäß der Devise aus der Vorlesung "Ein Korrektheitsbeweis vollzieht das Programm Schritt für Schritt nach." wurde im obigen Korrektheitsbeweis zunächst der Zustand nach der Zuweisung in Zeile 2 gezeigt, und dann der Zustand vor (Zeile 3) und nach jedem Eintritt in die Schleife (Zeile 4) mittels der Schleifeninvariante gezeigt. Somit wurde das Programm wirklich Schritt für Schritt – unter Verwendung der Schleifeninvariante – nachvollzogen ohne Annahmen bzgl. der Länge des Arrays A zu treffen.

Bei der Aufstellung und dem Beweis von Schleifeninvarianten sind folgende Dinge zu bemerken:

1. Eine Schleifeninvariante muss vor der ersten Ausführung der Schleife aber nach der Initialisierung der Laufvariablen gelten (vgl. Beweis der initialen Gültigkeit der Schleifeninvariante oben).
2. Der Erhalt der Schleifeninvariante wird gezeigt, indem man annimmt, dass die Invariante für einen beliebigen Wert der Laufvariablen (welcher auch wirklich angenommen werden kann) gilt und die Gültigkeit der Schleifeninvarianten auch immer noch nach der Ausführung des Schleifenkörpers und nach der Erhöhung der Laufvariablen gilt.
3. Aus der Gültigkeit der Schleifeninvariante bei Verlassen der Schleife (also für den ersten Wert der Laufvariablen, welcher nicht mehr im definierten Bereich liegt) muss sich ein sinnvoller Zustand folgern lassen, damit der darauffolgende Code auch bewiesen werden kann. Im Fall des Max-Suche Algorithmus war dieser "sinnvolle Zustand" de facto schon die Korrektheit des gesamten Algorithmus, da der Wert max direkt zurückgegeben wird.
4. In der Korrektheitsaussage $\forall k \in \{1, \ldots, \text{length}(A)\}.A[\text{Max-Search}(A)] \geq A[k] \wedge \text{Max-Search}(A) \in \{1, \ldots, \text{length}(A)\}$ wird das Array A nicht explizit beschrieben bzw. quantifiziert. Somit gilt die Korrektheitsaussage für alle, beliebigen (nicht-leeren) Arrays, da im Beweis keine Annahmen bzgl. des Arrays A getroffen werden mussten. Somit lässt sich sofort die Aussage

   $$\forall \text{Arrays } A, \text{length}(A) > 0.$$
   $$\Big(\big(\forall k \in \{1, \ldots, \text{length}(A)\}.A[\text{Max-Search}(A)] \geq A[k] \big) \wedge \text{Max-Search}(A) \in$$
   $$\{1, \ldots, \text{length}(A)\} \Big)$$

   schließen, was die formalste Art wäre die Korrektheit zu beschreiben.

Lösungsformat dieses Zettels

Ziel dieses Zettels ist das Erlernen des grundlegenden Rüstzeugs zum Führen von Korrektheitsbeweisen. Da formale Korrektheitsbeweise im Allgemeinen recht aufwändig sind, fordern wir auf diesem Zettel ausschließlich die Angabe der richtigen Korrektheitsaussagen sowie der Schleifeninvarianten, welche den Korrektheitsbeweis erlauben. Zu diesem Zweck geben wir für jede Aufgabe eine Menge von möglichen Quantifizierungen sowie möglicher Zustandsaussagen an, welche dann im Fragebogen benutzt werden können, um komplexere Zustandsbeschreibungen vorzunehmen. Abbildung 1 zeigt beispielhaft Quantifizierungen und Aussagen, welche für den Beweis der Korrektheit der Max-Suche benutzt werden könnten.

Das Abgabeformat ist dann wie folgt: Es muss immer eine Quantifizierung (als Zahl) mit einer entsprechenden Aussage zusammen angegeben werden. Mehrere solcher zusammengesetzter Ausdrücke können mittels des & Zeichens logisch verbunden werden. Das & Zeichen ist hierbei als logische Verundung (Konjunktion) zu verstehen.

Die korrekte Antwort auf die Frage nach der globalen Korrektheitsbedingung der Max-Suche (vgl. Aussage 1 auf Seite 3) wäre dementsprechend `C19 & A25` korrekt, d.h.:

$$\Big(\forall k \in \{1, \ldots, \text{length}(A)\}.A[\text{Max-Search}(A)] \geq A[k] \Big) \wedge \Big(\text{Max-Search}(A) \in \{1, \ldots, \text{length}(A)\} \Big)$$

Für die Frage nach der Schleifeninvariante der Max-Suche müsste als Antwort `F3 & A8` angegeben werden (vgl. Aussage 2 auf Seite 3):

$$\Big(\forall k \in \{1, \ldots, j - 1\}.A[max] \geq A[k] \Big) \wedge \Big(max \in \{1, \ldots, j - 1\} \Big)$$

Um Missverständnisse zu vermeiden gehen wir kurz noch etwas genauer auf die formale Bedeutung der Verbindung mehrerer Aussagen ein. Konkret beschreibt die Antwort $\alpha_1\beta_1$ & $\alpha_2\beta_2$ & $\ldots$ & $\alpha_n\beta_n$, wobei α_i ein Großbuchstabe und β_i eine der möglichen Zahlen sei, den logischen Ausdruck $(\alpha_1.\beta_1) \wedge (\alpha_2.\beta_2) \wedge \cdots \wedge (\alpha_n.\beta_n)$. Quantifizierungen erstrecken sich gemäß dieser Klammerung demnach nur auf den direkt folgenden Ausdruck und nicht auf später angegebene Ausdrücke.

(A)	(keine Quantifizierung)
(B)	$\forall k \in \{0, \ldots, \text{length}(A)\}$
(C)	$\forall k \in \{1, \ldots, \text{length}(A)\}$
(D)	$\forall k \in \{2, \ldots, \text{length}(A)\}$
(E)	$\forall k \in \{0, \ldots, j-1\}$
(F)	$\forall k \in \{1, \ldots, j-1\}$
(G)	$\forall k \in \{2, \ldots, j-1\}$
(H)	$\forall k \in \{0, \ldots, j\}$
(I)	$\forall k \in \{1, \ldots, j\}$
(J)	$\forall k \in \{2, \ldots, j\}$
(K)	$\forall k \in \{0, \ldots, j+1\}$
(L)	$\forall k \in \{1, \ldots, j+1\}$
(M)	$\forall k \in \{2, \ldots, j+1\}$
(N)	$\forall k \in \{j-1, \ldots, \text{length}(A)-1\}$
(O)	$\forall k \in \{j-1, \ldots, \text{length}(A)\}$
(P)	$\forall k \in \{j-1, \ldots, \text{length}(A)+1\}$
(Q)	$\forall k \in \{j, \ldots, \text{length}(A)-1\}$
(R)	$\forall k \in \{j, \ldots, \text{length}(A)\}$
(S)	$\forall k \in \{j, \ldots, \text{length}(A)+1\}$

(1)	$A[\text{max}] < A[k]$
(2)	$A[\text{max}] \leq A[k]$
(3)	$A[\text{max}] \geq A[k]$
(4)	$A[\text{max}] > A[k]$
(5)	$\text{max} \in \{1, \ldots, \text{length}(A)-1\}$
(6)	$\text{max} \in \{1, \ldots, \text{length}(A)\}$
(7)	$\text{max} \in \{1, \ldots, \text{length}(A)+1\}$
(8)	$\text{max} \in \{1, \ldots, j-1\}$
(9)	$\text{max} \in \{1, \ldots, j\}$
(10)	$\text{max} \in \{1, \ldots, j+1\}$
(11)	$\text{max} \in \{j-1, \ldots, \text{length}(A)-1\}$
(12)	$\text{max} \in \{j-1, \ldots, \text{length}(A)\}$
(13)	$\text{max} \in \{j-1, \ldots, \text{length}(A)+1\}$
(14)	$\text{max} \in \{j, j+\ldots, \text{length}(A)-1\}$
(15)	$\text{max} \in \{j, j+\ldots, \text{length}(A)\}$
(16)	$\text{max} \in \{j, j+\ldots, \text{length}(A)+1\}$
(17)	$A[\text{Max-Search(A)}] < A[k]$
(18)	$A[\text{Max-Search(A)}] \leq A[k]$
(19)	$A[\text{Max-Search(A)}] \geq A[k]$
(20)	$A[\text{Max-Search(A)}] > A[k]$
(21)	$\text{Max-Search(A)} \in \{0, \ldots, \text{length}(A)-1\}$
(22)	$\text{Max-Search(A)} \in \{0, \ldots, \text{length}(A)\}$
(23)	$\text{Max-Search(A)} \in \{0, \ldots, \text{length}(A)+1\}$
(24)	$\text{Max-Search(A)} \in \{1, \ldots, \text{length}(A)-1\}$
(25)	$\text{Max-Search(A)} \in \{1, \ldots, \text{length}(A)\}$
(26)	$\text{Max-Search(A)} \in \{1, \ldots, \text{length}(A)+1\}$

Abbildung 1: Beispielhafte Quantifizierungen (links) und Aussagen (rechts) für den Algorithmus Max-Search

Konventionen zum Formulieren von Korrektheitsaussagen

Wir benutzen auf diesem Zettel die folgenden Konventionen bzgl. der Beschreibung der Korrektheit von Algorithmen:

1. Wir trennen auf diesem Zettel Quantifizierungen im Allgemeinen von den Aussagen durch einen Punkt und schreiben somit z.B. $\forall x \in \mathbb{N}.x \geq x$. Wir benutzen den Punkt aus rein ästethischen Gründen und der teilweise einfacheren Lesbarkeit.
2. Korrektheitsaussagen für Algorithmen (speziell Funktionen) werden immer so formuliert, dass der Funktionsname in der Beschreibung benutzt wird. Die Korrektheit von MaxSearch wird somit mittels $\forall k \in \{1, \ldots, \text{length}(A)\}.A[\text{Max-Search}(A)] \geq A[k]$ spezifiziert. Korrektheitsangaben, welche sich auf die letzte Zeile eines Algorithmus beziehen, werden auf diesem Zettel nicht als korrekt betrachtet und geben keine Punkte. Die Angabe $\forall k \in \{1, \ldots, \text{length}(A)\}.A[max] \geq A[k]$ wäre somit nicht korrekt. Wir treffen diese Unterscheidung, um Missverständnisse zu vermeiden.
3. Korrektheitsaussagen für Algorithmen werden so getroffen, dass die Eingabeparameter nicht näher spezifiziert sind und somit beliebige Werte annehmen können. Die Korrektheit der Funktion Add10$(x)\{$return $x+10\}$ wird nach dieser Vereinbarung mit Add10$(x) = x+10$ und nicht mit $\forall x \in \mathbb{R}.\text{Add10}(x) = x + 10$ beschrieben.

Hilfestellung beim Umgang mit logischen Aussagen

Die Einschränkung auf die angegebenen Quantifizierungen sowie Aussagen kann dazu führen, dass ihr die von euch gefundenen Korrektheitsaussagen / Schleifeninvarianten umformulieren müsst. Folgende Anmerkungen sollen euch als Hilfestellung dienen, eure Korrektheitsaussagen richtig zu formulieren und in unserem Format abzugeben.

- Jede Aussage $\forall k \in \{y, y+1, \ldots, z\}.P(k)$, wobei $P(k)$ eine logische Aussage ist, ist äquivalent zur Aussage $\forall k \in \{y+i, y+i+1, \ldots, z+i\}.P(k-i)$. So wäre z.B. die Aussage $\forall k \in \{3, \ldots, j+1\}.A[max] \geq A[k-2]$ logisch äquivalent zur Aussage $\forall k \in \{1, \ldots, j-1\}.A[max] \geq A[k]$. Dies gilt natürlich auch für den Existenzquantor: $\exists k \in \{y, y+1, \ldots, z\}.P(k)$ ist somit äquivalent zur Aussage $\exists k \in \{y+i, y+i+1, \ldots, z+i\}.P(k-i)$.
- In allen Aufgaben bieten wir Quantifizierungen mit dem Allquantor ($\forall$) bzw. dem Existenzquantor ($\exists$) an. Die eigentliche Bedeutung dieser Quantoren, nämlich, dass die folgende logische Aussage für alle oder mindestens ein Element gilt, ist leicht zu verstehen. Das Verhalten über der leeren Menge hingegen führt oft zu Problemen. Konkret:

  $\forall k \in \emptyset.P(k)$ Ist immer wahr, da die Menge keine Elemente enthält und die Aussage daher für kein einziges Element (also alle) gelten muss. Somit ist die Aussage $\forall k \in \emptyset.k \neq k$ wahr.

  $\exists k \in \emptyset.P(k)$ Ist immer falsch, da die Menge kein einziges Element enthält, für welches die Aussage $P(k)$ gelten könnte. Somit ist die Aussage $\exists k \in \emptyset.k = k$ falsch.
- Beachtet, dass die Mengenangabe $\{1, \ldots, i\}$ für alle $i \leq 0$ die leere Menge bezeichnet und für $i \geq 1$ nur natürliche Zahlen enthält. So ist $\{1, \ldots, 0\} = \emptyset$, $\{1, \ldots, 1\} = \{1\}$ und $\{1, \ldots, 4\} = \{1, 2, 3, 4\}$.
- Innerhalb eines atomaren logischen Ausdrucks wie z.B. $x < y$ können natürlich beliebige Umformungen durchgeführt werden: $x + 1 < y + 1$ bzw. $y > x$ sind äquivalent.
- Manche Algorithmen liefern nur einen "Booleschen Wert", d.h. "wahr" oder "falsch" zurück. Dies ist z.B. der Fall beim einfachen Primzahltest aus Listing 3. Konkret wird hier der Wert 1

("wahr") zurückgegben, falls es sich um eine Primzahl handelt, und 0 ("falsch"), falls es sich um keine Primzahl handelt[1]. Die Korrektheitsaussage, dass dieser Algorithmus für einen Parameter x den Wert 1 zurückgibt, falls es sich um eine Primzahl handelt, lässt sich wie folgt ausdrücken:

$$\big(\forall i \in \{2, \ldots, x-1\}.x \bmod i \neq 0\big) \Rightarrow \text{IstPrim}(x) = 1$$

Hierbei beschreibt die linke Seite mathematisch formal, dass es sich bei x um eine Primzahl handelt. Gemäß dieser Korrektheitsaussage könnte der Algorithmus auch bei Zahlen, welche nicht prim sind, eine 1 zurückgeben. Im Allgemeinen spezifiziert man daher genau-dann-wenn Formulierungen: Der Algorithmus soll eine 1 genau dann zurückliefern, wenn es sich um eine Primzahl handelt. Die Formulierung genau-dann-wenn ist immer als Implikation in beide Richtungen zu verstehen: Aus der Tatsache, dass es sich um eine Primzahl handelt, wird die Aussage, dass 1 zurückgeliefert wird gefolgert und aus der Aussage, dass der Algorithmus eine 1 zurückgibt, wird die Aussage, dass es sich um eine Primzahl handelt, gefordert. Formal lässt sich dies am einfachsten mit einer logischen Äquivalenz darstellen:

$$\big(\forall i \in \{2, \ldots, x-1\}.x \bmod i \neq 0\big) \Leftrightarrow \text{IstPrim}(x) = 1$$

Beachtet, dass bei dieser Korrektheitsaussage nur der Rückgabewert 1 von dem Algorithmus IstPrim von Bedeutung ist. Die Tatsache, dass der Algorithmus eine 0 zurückliefert, falls es sich um keine Primzahl handelt muss hier also nicht betrachtet werden.

Listing 3: Einfacher Primzahltest

```
1 IstPrim(Natuerliche Zahl x > 1)
2   resultat ← 1
3   for i ← 2 to x-1 do
4     if x mod i ← 0 then resultat ← 0
5   return resultat
```

Abgaben / Automatische Tests / Bewertung

Für jede Aufgabe dieses Zettels findet ihr im SVN die entsprechenden Fragebogen-Vorlagen. Konkret `introprog_korrektheit_mult_vorgabe.txt` für Aufgabe 1, `introprog_korrektheit_arraysuche_vorgabe.txt` für Aufgabe 2 und `introprog_korrektheit_bubblesort_vorgabe.txt` für Aufgabe 3.

Gebt eure Abgaben wiederum im SVN mit den Dateinamen `introprog_korrektheit_mult.txt`, `introprog_korrektheit_arraysuche.txt` und `introprog_korrektheit_bubblesort.txt` ab und vergesst nicht, eure Namen und Gruppen im Header der Fragebögen einzutragen.

[1] Der Algorithmus benutzt die Modulo-Funktion `mod`, welche den Rest bei ganzzahliger Division zurückliefert. Konkret ist z.B. $5 \bmod 2 = 1$, $13 \bmod 7 = 6$, etc. Eine Zahl x ist durch eine andere Zahl i genau dann teilbar, wenn $x \bmod i = 0$ gilt.

Es gibt in OSIRIS automatische Tests, welche eure Eingaben in den Fragebögen auf syntaktische Korrektheit testen. Es wird also überprüft, ob eure Antworten in dem richtigen Format vorliegen. Ihr bekommt jedoch kein Feedback, ob eure Antworten korrekt sind.

Aufgrund der engen Verzahnung der Korrektheitsaussagen sowie der Schleifeninvarianten, werden die möglichen Punkte im Allgemeinen nur vergeben, wenn die Aussagen schlüssig sind. Aus der Schleifeninvariante $\exists k \in \{1, \ldots, j-1\}.A[max] \geq A[k]$ kann z.B. nicht die Korrektheit des Max-Suche Algorithmus gefolgert werden. Selbst, wenn die Korrektheitsaussage korrekt angegeben worden wäre, würde es somit keinen Punkt in der Bewertung geben. Vor diesem Hintergrund seid ihr gut damit beraten, die Beweise wirklich zu führen und immer zu überprüfen, ob sich die entsprechenden Aussagen auseinander ableiten lassen.

Aufgabe 1 Korrektheit der einfachen Multiplikation (3 Punkte)

Die Funktion Mult (siehe Listing 4) berechnet das Produkt zweier natürlicher Zahlen (größer Null).

Listing 4: Algorithmus zur Multiplikation zweier Zahlen

```
1 Mult(Natuerliche Zahlen x,y > 0)
2   resultat ← 0
3   for i ← 1 to y do
4     resultat ← resultat + x
5   return resultat
```

Beantwortet die Fragen der Datei `introprog_korrektheit_mult_vorgabe.txt` unter Verwendung der in Abbildung 2 angegebenen Antwortmöglichkeiten. Ladet eure Antwort als Datei `introprog_korrektheit_mult.txt` ins SVN.

(A) (keine Quantifizierung)
(B) $\forall k \in \{0, \ldots, i\}$
(C) $\forall k \in \{1, \ldots, i\}$
(D) $\forall k \in \{2, \ldots, i\}$
(E) $\forall k \in \{0, \ldots, i-1\}$
(F) $\forall k \in \{1, \ldots, i-1\}$
(G) $\forall k \in \{2, \ldots, i-1\}$
(H) $\forall k \in \{0, \ldots, i+1\}$
(I) $\forall k \in \{1, \ldots, i+1\}$
(J) $\forall k \in \{2, \ldots, i+1\}$
(K) $\exists k \in \{0, \ldots, i\}$
(L) $\exists k \in \{1, \ldots, i\}$
(M) $\exists k \in \{2, \ldots, i\}$
(N) $\exists k \in \{0, \ldots, i-1\}$
(O) $\exists k \in \{1, \ldots, i-1\}$
(P) $\exists k \in \{2, \ldots, i-1\}$
(Q) $\exists k \in \{0, \ldots, i+1\}$
(R) $\exists k \in \{1, \ldots, i+1\}$
(S) $\exists k \in \{2, \ldots, i+1\}$

(1) $\text{Mult}(x, y) = (x-1) \cdot (y-1)$
(2) $\text{Mult}(x, y) = (x-1) \cdot y$
(3) $\text{Mult}(x, y) = (x-1) \cdot (y+1)$
(4) $\text{Mult}(x, y) = x \cdot (y-1)$
(5) $\text{Mult}(x, y) = x \cdot y$
(6) $\text{Mult}(x, y) = x \cdot (y+1)$
(7) $\text{Mult}(x, y) = (x+1) \cdot (y-1)$
(8) $\text{Mult}(x, y) = (x+1) \cdot y$
(9) $\text{Mult}(x, y) = (x+1) \cdot (y+1)$
(10) $\text{resultat} = (x-1) \cdot (y-1)$
(11) $\text{resultat} = (x-1) \cdot y$
(12) $\text{resultat} = (x-1) \cdot (y+1)$
(13) $\text{resultat} = x \cdot (y-1)$
(14) $\text{resultat} = x \cdot y$
(15) $\text{resultat} = x \cdot (y+1)$
(16) $\text{resultat} = (x+1) \cdot (y-1)$
(17) $\text{resultat} = (x+1) \cdot y$
(18) $\text{resultat} = (x+1) \cdot (y+1)$
(19) $\text{resultat} = (x-i) \cdot (y-1)$
(20) $\text{resultat} = (x-i) \cdot y$
(21) $\text{resultat} = (x-i) \cdot (y+1)$
(22) $\text{resultat} = (x-i+1) \cdot (y-1)$
(23) $\text{resultat} = (x-i+1) \cdot y$
(24) $\text{resultat} = (x-i+1) \cdot (y+1)$
(25) $\text{resultat} = (x-1) \cdot (y-i-1)$
(26) $\text{resultat} = (x-1) \cdot (y-i)$
(27) $\text{resultat} = (x-1) \cdot (y-i+1)$
(28) $\text{resultat} = x \cdot (y-i-1)$
(29) $\text{resultat} = x \cdot (y-i)$
(30) $\text{resultat} = x \cdot (y-i+1)$
(31) $\text{resultat} = (i-1) \cdot (y-1)$
(32) $\text{resultat} = (i-1) \cdot y$
(33) $\text{resultat} = (i-1) \cdot (y+1)$
(34) $\text{resultat} = x \cdot (i-1)$
(35) $\text{resultat} = x \cdot i$
(36) $\text{resultat} = x \cdot (i+1)$

Abbildung 2: Mögliche Quantifizierungen (links) und Aussagen (rechts) für den Algorithmus Mult

Aufgabe 2 Korrektheit der Array-Suche (3 Punkte)

In Listing 5 ist der Pseudocode der Funktion ArraySuche gegeben, der ein (nicht-leeres) Array und ein Wert x übergeben wird. Die Funktion liefert genau dann den Wert 1 zurück, wenn der Wert x (mindestens einmal) im Array A enthalten ist.

Listing 5: Algorithmus zur Suche eines Werts im Array A

```
1 ArraySuche(Array A, Wert x)
2   resultat ← 0
3   for j ← 1 to length(A) do
4     if A[j] = x then resultat ← 1
5   return resultat
```

Beantwortet die Fragen der Datei `introprog_korrektheit_arraysuche_vorgabe.txt` unter Verwendung der in Abbildung 3 angegebenen Antwortmöglichkeiten. Ladet eure Antwort als Datei `introprog_korrektheit_arraysuche.txt` ins SVN.

Hinweis: Bei der Kombination der Quantifizierungen (B) bis (W) mit den Aussagen (1) bis (30) bezieht sich die Quantifizierung nur auf die linke Seite der Aussage. `B1` (siehe Abb. 3) entspricht somit:

$$\Big(\forall k \in \{1, \ldots, \text{length}(A)\}.A[k] < x\Big) \Leftarrow \text{ArraySuche}(A, x) = 1$$

(A) (keine Quantifizierung)
(B) $\forall k \in \{1, \ldots, \text{length}(A)\}$
(C) $\forall k \in \{2, \ldots, \text{length}(A)\}$
(D) $\forall k \in \{1, \ldots, j-1\}$
(E) $\forall k \in \{2, \ldots, j-1\}$
(F) $\forall k \in \{1, \ldots, j\}$
(G) $\forall k \in \{2, \ldots, j\}$
(H) $\forall k \in \{1, \ldots, j+1\}$
(I) $\forall k \in \{2, \ldots, j+1\}$
(J) $\forall k \in \{j-1, \ldots, \text{length}(A)-1\}$
(K) $\forall k \in \{j-1, \ldots, \text{length}(A)\}$
(L) $\forall k \in \{j-1, \ldots, \text{length}(A)+1\}$
(M) $\exists k \in \{1, \ldots, \text{length}(A)\}$
(N) $\exists k \in \{2, \ldots, \text{length}(A)\}$
(O) $\exists k \in \{1, \ldots, j-1\}$
(P) $\exists k \in \{2, \ldots, j-1\}$
(Q) $\exists k \in \{1, \ldots, j\}$
(R) $\exists k \in \{2, \ldots, j\}$
(S) $\exists k \in \{1, \ldots, j+1\}$
(T) $\exists k \in \{2, \ldots, j+1\}$
(U) $\exists k \in \{j-1, \ldots, \text{length}(A)-1\}$
(V) $\exists k \in \{j-1, \ldots, \text{length}(A)\}$
(W) $\exists k \in \{j-1, \ldots, \text{length}(A)+1\}$

(1) $A[k] < x \Leftarrow \text{ArraySuche}(A, x) = 1$
(2) $A[k] \leq x \Leftarrow \text{ArraySuche}(A, x) = 1$
(3) $A[k] = x \Leftarrow \text{ArraySuche}(A, x) = 1$
(4) $A[k] \geq x \Leftarrow \text{ArraySuche}(A, x) = 1$
(5) $A[k] > x \Leftarrow \text{ArraySuche}(A, x) = 1$
(6) $A[k] < x \Leftrightarrow \text{ArraySuche}(A, x) = 1$
(7) $A[k] \leq x \Leftrightarrow \text{ArraySuche}(A, x) = 1$
(8) $A[k] = x \Leftrightarrow \text{ArraySuche}(A, x) = 1$
(9) $A[k] \geq x \Leftrightarrow \text{ArraySuche}(A, x) = 1$
(10) $A[k] > x \Leftrightarrow \text{ArraySuche}(A, x) = 1$
(11) $A[k] < x \Rightarrow \text{ArraySuche}(A, x) = 1$
(12) $A[k] \leq x \Rightarrow \text{ArraySuche}(A, x) = 1$
(13) $A[k] = x \Rightarrow \text{ArraySuche}(A, x) = 1$
(14) $A[k] \geq x \Rightarrow \text{ArraySuche}(A, x) = 1$
(15) $A[k] > x \Rightarrow \text{ArraySuche}(A, x) = 1$
(16) $A[k] < x \Leftarrow \text{resultat} = 1$
(17) $A[k] \leq x \Leftarrow \text{resultat} = 1$
(18) $A[k] = x \Leftarrow \text{resultat} = 1$
(19) $A[k] \geq x \Leftarrow \text{resultat} = 1$
(20) $A[k] > x \Leftarrow \text{resultat} = 1$
(21) $A[k] < x \Leftrightarrow \text{resultat} = 1$
(22) $A[k] \leq x \Leftrightarrow \text{resultat} = 1$
(23) $A[k] = x \Leftrightarrow \text{resultat} = 1$
(24) $A[k] \geq x \Leftrightarrow \text{resultat} = 1$
(25) $A[k] > x \Leftrightarrow \text{resultat} = 1$
(26) $A[k] < x \Rightarrow \text{resultat} = 1$
(27) $A[k] \leq x \Rightarrow \text{resultat} = 1$
(28) $A[k] = x \Rightarrow \text{resultat} = 1$
(29) $A[k] \geq x \Rightarrow \text{resultat} = 1$
(30) $A[k] > x \Rightarrow \text{resultat} = 1$

Abbildung 3: Mögliche Quantifizierungen (links) und Aussagen (rechts) für den Algorithmus Array-Suche

Aufgabe 3 Korrektheit Bubblesort (4 Punkte)

In Listing 6 findet ihr den Pseudocode von dem Algorithmus Bubblesort, welcher ein übergebenes Array aufsteigend sortiert und dieses wieder zurückgibt. Die Funktion `swap(A,i,j)` vertauscht hierbei die Werte in A, welche an der Stelle i und j stehen.

Listing 6: Bubblesort

```
1 BubbleSort(Array A)
2   for j ← length(A) downto 2 do
3       for i ←  1 to j-1 do
4           if A[i] > A[i+1] then swap(A,i,i+1)
5   return A
```

Beantwortet die Fragen der Datei `introprog_korrektheit_bubblesort_vorgabe.txt` unter Verwendung der in Abbildung 4 angegebenen Antwortmöglichkeiten. Ladet eure Antwort als Datei `introprog_korrektheit_bubblesort.txt` ins SVN.

Hinweis: Die Korrektheitsbedingung, dass alle Elemente des Arrays aufsteigend sortiert sind, reicht bei dieser Aufgabe aus. Es muss nicht gezeigt werden, dass auch nach der Sortierung die gleichen Elemente enthalten sind.

Hinweis: Die Kombination der Quantifizierungen (T) bis (Y) mit den Aussagen (1) bis (50) aus Abbildung 4 ist so zu lesen, dass die Quantifizierung stärker bindet als die Implikation. `V1` (siehe Abbildung 4) entspricht somit:

$$j < length(A) \Rightarrow \Big(\forall k \in \{1, \ldots, j-1\}.(A' = \text{BubbleSort}(A) : A'[k] < A'[k+1])\Big)$$

Hinweis: Das erstmalige Führen eines Korrektheitsbeweises mit verschachtelten Schleifen ist recht komplex. In der Vorlesung wurde der Korrektheitsbeweis von InsertionSort vorgestellt. In ISIS findet ihr weiterhin den formalen Beweis der Korrektheit von SelectionSort als Anschauungsmaterial, welcher auch im Tutorium besprochen wird. Die folgende Beweisstruktur, welche auch bei dem Beweis von SelectionSort benutzt wird, kann euch eventuell helfen, euren Beweis zu ordnen. Beachtet hierbei, dass die folgende Struktur nur eine Hilfestellung sein soll, um euren eigenen Beweis zu führen. Die eigentliche bewertete Aufgabe findet ihr in der Datei `introprog_korrektheit_bubblesort.txt` im SVN. Um die darin gestellten Fragen korrekt zu beantworten, solltet ihr den Beweis jedoch selbstständig durchführen.

A-1 Gebt zunächst die Schleifeninvariante für die äußere Schleife an. Aus dieser muss sich die Korrektheitsaussage des Algorithmus nach der Durchführung der äußeren Schleife ableiten lassen (siehe C-1).

A-2 Zeigt die initiale Gültigkeit der äußeren Schleifeninvariante, d.h. dass die Aussage nach der Initialisierung von j = length(A) in Zeile 2 gilt.

B-1 Überlegt euch nun eine Schleifeninvariante für die innere Schleife, aus welcher sich die Gültigkeit der äußeren Schleifeninvariante ableiten lässt.

B-2 Zeigt die initiale Gültigkeit der inneren Schleifenvariante, d.h., dass die Invariante für jedes i = 1 gilt.

B-3 Zeigt, dass die Ausführung des inneren Schleifenkörpers (Zeile 4) die Gültigkeit der inneren Schleifeninvariante erhält.

A-3 Zeigt, dass die Gültigkeit der äußeren Schleifeninvariante durch die Ausführung der inneren Schleife (Zeile 3-4) erhalten bleibt. Benutzt dabei, dass die innere Schleifeninvariante beim Austritt aus der inneren for-Schleife gegolten hat.

C-1 Zeigt unter Verwendung der äußeren Schleifeninvariante die Korrektheit des Algorithmus, also dass das zurückgegebene Array aufsteigend sortiert ist.

(A) (keine Quantifizierung)	(1) $(A' = \text{BubbleSort}(A) : A'[k] < A'[k+1])$
(B) $\forall k \in \{0, \ldots, \text{length}(A) - 1\}$	(2) $(A' = \text{BubbleSort}(A) : A'[k] \leq A'[k+1])$
(C) $\forall k \in \{1, \ldots, \text{length}(A) - 1\}$	(3) $(A' = \text{BubbleSort}(A) : A'[k] = A'[k+1])$
(D) $\forall k \in \{2, \ldots, \text{length}(A) - 1\}$	(4) $(A' = \text{BubbleSort}(A) : A'[k] > A'[k+1])$
(E) $\forall k \in \{0, \ldots, \text{length}(A)\}$	(5) $(A' = \text{BubbleSort}(A) : A'[k] \geq A'[k+1])$
(F) $\forall k \in \{1, \ldots, \text{length}(A)\}$	(6) $A[k] < A[k+1]$
(G) $\forall k \in \{2, \ldots, \text{length}(A)\}$	(7) $A[k] \leq A[k+1]$
(H) $\forall k \in \{1, \ldots, i-1\}$	(8) $A[k] = A[k+1]$
(I) $\forall k \in \{1, \ldots, i\}$	(9) $A[k] \geq A[k+1]$
(J) $\forall k \in \{1, \ldots, j-1\}$	(10) $A[k] > A[k+1]$
(K) $\forall k \in \{1, \ldots, j\}$	(11) $A[i] < A[k]$
(L) $\forall k \in \{j, \ldots, \text{length}(A) - 1\}$	(12) $A[i] \leq A[k]$
(M) $\forall k \in \{j, \ldots, \text{length}(A)\}$	(13) $A[i] = A[k]$
(N) $\forall k \in \{j, \ldots, \text{length}(A) + 1\}$	(14) $A[i] \geq A[k]$
(O) $\forall k \in \{i, \ldots, \text{length}(A) - 1\}$	(15) $A[i] > A[k]$
(P) $\forall k \in \{i, \ldots, \text{length}(A)\}$	(16) $A[i] < A[k+1]$
(Q) $\forall k \in \{i, \ldots, \text{length}(A) + 1\}$	(17) $A[i] \leq A[k+1]$
(R) $i \in \{1, \ldots, \text{length}(A) - 1\} \Rightarrow \forall k \in \{1, \ldots, j-1\}$	(18) $A[i] = A[k+1]$
(S) $i \in \{1, \ldots, \text{length}(A) - 1\} \Rightarrow \forall k \in \{1, \ldots, j\}$	(19) $A[i] \geq A[k+1]$
(T) $i \in \{1, \ldots, \text{length}(A)\} \Rightarrow \forall k \in \{1, \ldots, j-1\}$	(20) $A[i] > A[k+1]$
(U) $i \in \{1, \ldots, \text{length}(A)\} \Rightarrow \forall k \in \{1, \ldots, j\}$	(21) $A[i+1] < A[k]$
(V) $j \in \{1, \ldots, \text{length}(A) - 1\} \Rightarrow \forall k \in \{1, \ldots, j-1\}$	(22) $A[i+1] \leq A[k]$
(W) $j \in \{1, \ldots, \text{length}(A) - 1\} \Rightarrow \forall k \in \{1, \ldots, j\}$	(23) $A[i+1] = A[k]$
(X) $j \in \{1, \ldots, \text{length}(A)\} \Rightarrow \forall k \in \{1, \ldots, j-1\}$	(24) $A[i+1] \geq A[k]$
(Y) $j \in \{1, \ldots, \text{length}(A)\} \Rightarrow \forall k \in \{1, \ldots, j\}$	(25) $A[i+1] > A[k]$

Abbildung 4: Mögliche Quantifizierungen (links) und Aussagen (rechts) für den BubbleSort Algorithmus

(26) $A[i+1] < A[k+1]$
(27) $A[i+1] \leq A[k+1]$
(28) $A[i+1] = A[k+1]$
(29) $A[i+1] \geq A[k+1]$
(30) $A[i+1] > A[k+1]$
(31) $A[j] < A[k]$
(32) $A[j] \leq A[k]$
(33) $A[j] = A[k]$
(34) $A[j] \geq A[k]$
(35) $A[j] > A[k]$
(36) $A[j] < A[k+1]$
(37) $A[j] \leq A[k+1]$
(38) $A[j] = A[k+1]$
(39) $A[j] \geq A[k+1]$
(40) $A[j] > A[k+1]$
(41) $A[j+1] < A[k]$
(42) $A[j+1] \leq A[k]$
(43) $A[j+1] = A[k]$
(44) $A[j+1] \geq A[k]$
(45) $A[j+1] > A[k]$
(46) $A[j+1] < A[k+1]$
(47) $A[j+1] \leq A[k+1]$
(48) $A[j+1] = A[k+1]$
(49) $A[j+1] \geq A[k+1]$
(50) $A[j+1] > A[k+1]$

Abbildung 5: Mögliche Aussagen (rechts) für den BubbleSort Algorithmus (cont.)