Chair of Embedded Systems
Department of Computer Science
University of Augsburg, Germany

Martin Frieb

Universität Augsburg University

Wintersemester 2020/2021

Informatik 1

Kapitel 12 – Korrektheit von Algorithmen

Inhaltsverzeichnis

12.1 Was ist Korrektheit? 2

12.2 Partielle Korrektheit 3
12.2.1 Wie kann man partielle Korrektheit beweisen? . . . 3
12.2.2 Zusicherungen in Algorithmusdarstellungen . . . 3
12.2.3 Exkurse in die Logik . . . 4
12.2.4 Hoare-Kalkül zur Korrektheit von Algorithmen . . . 7
12.2.5 Beispiel für den Nachweis der partiellen Korrektheit: Multiplikation . . . 12
12.2.6 Überblick: Wie weist man die partielle Korrektheit nach? . . . 16
12.2.7 Beispiel für den Nachweis der partiellen Korrektheit: Palindromtest . . . 17

12.3 Totale Korrektheit 20
12.3.1 Wie kann man die totale Korrektheit nachweisen? . . . 20
12.3.2 Wie entwirft man eine Terminierungsfunktion? . . . 21

Version 20210226-2126

12.1 Was ist Korrektheit?

Grundlegende Forderung an Algorithmen - Wiederholung
Eine grundlegende Forderung an einen Algorithmus ist, dass er eine vorgegebene Problemstellung löst.

Oft ist die Problemstellung aber zu vage, um zu überprüfen, ob die grundlegende Anforderung erfüllt ist. Deshalb präzisiert und formalisiert man die Problemstellung vor deren Bearbeitung. Eine solche Präzisierung nennt man **Problemspezifikation**, diese hilft dann oft auch bei der Lösung des Problems. Wir hatten sie in Kapitel 10.7 kennengelernt. Von dort kennen wir auch schon die beiden folgenden Definitionen.

Löst ein Algorithmus die durch eine Problemspezifikation gegebene Problemstellung, so nennt man ihn **korrekt**

> Definition: 10.37/12.1 Verifikationsverfahren
>
> Verifikationsverfahren dienen dem Nachweis der Korrektheit eines Algorithmus bzgl. einer Problemspezifikation.

> Definition: 10.38/12.2 Validierungsverfahren
>
> Validierungsverfahren dienen für den Nachweis der Korrektheit einer Problemspezifikation bzgl. eines realen Problems.

Was bedeutet Korrektheit formal?

> Definition: 12.3 Partielle und totale Korrektheit
>
> Ein Algorithmus `A` heißt
>
> - **partiell korrekt bzgl. einer Problemspezifikation** `S`, falls gilt:
>   Ist `e` eine gültige Eingabe (d.h ein Element der Menge der Eingabedaten aus `S`) und liefert `A` für `e` eine Ausgabe `a`, so ist `a` eine von `S` erlaubte Ausgabe für `e`
> - **total korrekt bzgl. einer Problemspezifikation** `S`, falls gilt:
>   `A` ist partiell korrekt und liefert für jede gültige Eingabe `e` eine Ausgabe `a`

Einfach ausgedrückt bedeutet das:

- partiell korrekt: „Es existiert für eine gültige Eingabe eine gültige Ausgabe bei der das Programm terminiert“. Die partielle Korrektheit fordert nicht, dass für alle gültige Eingaben eine Ausgabe geliefert wird, d.h. sie fordert nicht, dass der Algorithmus immer terminiert.
- total korrekt: „Für alle gültigen Eingaben wird eine gültige Ausgabe geliefert bei der das Programm terminiert“. Die totale Korrektheit fordert zusätzlich dazu, dass für alle gültige Eingaben eine Ausgabe geliefert wird. Das bedeutet, in jedem Fall einer gültigen Eingabe terminiert der Algorithmus.

12.2 Partielle Korrektheit

12.2.1 Wie kann man partielle Korrektheit beweisen?

Generelle Idee:
Um eine partielle Korrektheit zu beweisen, müssen für jede Anweisung im Algorithmus folgende drei Dinge formuliert werden:

- **Vorbedingung**: Formuliere eine Zusicherung, die vor der Anweisung erfüllt ist.
- **Nachbedingung**: Folgere aus der Art der Anweisung und der Vorbedingung eine Zusicherung, die nach der Anweisung erfüllt ist.
- Die Nachbedingung einer Anweisung dient dann als Vorbedingung für die nachfolgende Anweisung.
- **Ziel**: Die Nachbedingung der letzten Anweisung entspricht dem funktionalen Zusammenhang der Ausgabe von der Eingabe gemäß der vorgegebenen Problemspezifikation.

Was ist eine Zusicherung?
Eine **Zusicherung** ist eine Aussage (formuliert als prädikatenlogische Formel) über die Wertebelegungen der Programmvariablen.

12.2.2 Zusicherungen in Algorithmusdarstellungen

Bevor wir jedoch genaueres zur partiellen Korrektheit erklären oder zeigen können, muss zuvor noch deren Darstellungsform geklärt werden. Hierfür verwenden wir die folgende Notation:

- A: Anweisung
- p: Vorbedingung von A
- q: Nachbedingung von A

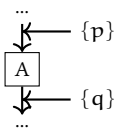
Im **Pseudocode** sähe die Zusicherung folgendermaßen aus:

```
1 ...
2 {p}
3 A;
4 {q}
5 ...
```

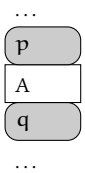
Beispiel:

```
1 ...
2 {p := i < 1}
3 A := i ← i + 1
4 {q := i < 2}
5 ...
```

Programmablaufplan:



Struktogramm:



12.2.3 Exkurse in die Logik

Exkurs: Freie und gebundene Variablen in prädikatenlogischen Formeln

Wir erinnern uns an freie und gebundene Variablen aus Kapitel 10.6. Kurz gefasst sind Variablen Platzhalter für beliebige Ausdrücke. Ein solcher Ausdruck kann eine logische Aussage sein, eine mathematische Formel, etc.. Diese Variablen können durch Quantoren (diese Zeichen: $\forall$, $\exists$) *gebunden* werden. Eine Variable, die nicht gebunden ist, nennt man *frei*.

Beispiel:

Seien die Variablen x, y gegeben:

- Bei Relationen, z.B. im Term $x < y$ sind x und y freie Variablen.
- Im Ausdruck $x > 5$ ist x eine freie Variable, da sie nicht an einen Quantor gebunden ist.
- Im Ausdruck $\exists x : x > 5$ ist x eine gebundene Variable, da x an den Existenzquantor gebunden ist.
- Im Ausdruck $\forall x : x + y = 10$ ist x eine gebundene Variable und y eine freie Variable.
- Im Ausdruck $(x = 1) \Rightarrow (\exists x : x + 1 = 9)$ ist das erste x eine freie Variable und das zweite x eine gebundene Variable. Auch gilt zu beachten, dass das erste x **nicht** das gleiche ist wie das zweite x.

- In der Menge $\{x \mid x + y = 10\}$ ist x eine gebunde Variable, y ist eine freie Variable.

Sei $V(A)$ die Menge der in einem Term A auftretenden Variablen. Wir definieren induktiv folgende Mengen für Formeln p:

- $FV(p)$: Menge der **freien Variablen** von p
- $BV(p)$: Menge der **gebundenen Variablen** von p

Die freien Variablen in einem Term sind die freien Variablen, die in den Einzelbestandteilen des Terms vorkommen. Das heißt, wir zerlegen den Term in seine Teilterme, schauen, welche freien Variablen in diesen Teiltermen vorkommen und fassen die gefundenen freien Variablen in einer großen Menge zusammen. Analog verfahren wir mit den gebundenen Variablen.

Definition: 12.4 Freie und gebundene Variablen

- Für $p := (A_1, \ldots, A_n) \in R$ ist $FV(p) := V(A_1) \cup \ldots \cup V(A_n)$ und $BV(p) := \emptyset$
- Für $p := q \wedge r$ ist $FV(p) := FV(q) \cup FV(r)$ und $BV(p) := BV(q) \cup BV(r)$
- Für $p := q \vee r$ ist $FV(p) := FV(q) \cup FV(r)$ und $BV(p) := BV(q) \cup BV(r)$
- Für $p := \neg q$ ist $FV(p) := FV(q)$ und $BV(p) := BV(q)$
- Für $p := q \Rightarrow r$ ist $FV(p) := FV(q) \cup FV(r)$ und $BV(p) := BV(q) \cup BV(r)$
- Für $p := q \Leftrightarrow r$ ist $FV(p) := FV(q) \cup FV(r)$ und $BV(p) := BV(q) \cup BV(r)$
- Für $p := \forall x(q)$ ist $FV(p) := FV(q) \setminus \{x\}$ und $BV(p) := BV(q) \cup \{x\}$
- Für $p := \exists x(q)$ ist $FV(p) := FV(q) \setminus \{x\}$ und $BV(p) := BV(q) \cup \{x\}$

Exkurs: Substitution in prädikatenlogischen Formeln
Wir haben das Prinzip der Substitution schonmal bei der Variablen-Belegung β in Kapitel 10 verwendet, aber dort nicht so bezeichnet (siehe Kapitel 10.2, „Auswertung von Termen“).

Wir definieren induktiv eine Formel $p\{x \to A\}$, die durch **Substitution** einer Variable x in einer Formel p durch einen Term A entsteht.

Definition: 12.5 Substitution - Terme

- Es gilt: $x\{x \to A\} := A$
- Für eine Variable y mit $y \neq x$ gilt: $y\{x \to A\} := y$
- Für Konstante a gilt: $a\{x \to A\} := a$
- Für eine n-stellige Operation op und Terme $A_1, \ldots, A_n$ gilt: $op(A_1, \ldots, A_n)\{x \to A\} := op(A_1\{x \to A\}, \ldots, A_n\{x \to A\})$
- Für eine Funktion f und einen Term B gilt: $f(B)\{x \to A\} := f(B\{x \to A\})$

Informell bedeutet das:
Man „ersetzt“ eine Variable mit einem Term. Betrachtet man hierzu die Definition, bedeutet der erste Punkt $x\{x \to A\} := A$, dass jedes Vorkommen von x durch A ersetzt wird.

Beispiel:

Das Prinzip sollte schon aus der Mathematik bekannt sein. Auch in der Mathematik lässt sich die Substitution anwenden: $x = y + z$ Sei nun $y = a + 1$, sodass wir den rechten Teil der Formel substituieren können: $x = a + 1 + z$

In unserer Schreibweise sähe das folgendermaßen aus:

$$y + z\{y \to a + 1\} := a + 1 + z$$

Die Substitution in einem Term führt man auf den Einzelbestandteilen des Terms durch. Das heißt, wir zerlegen den Term in seine Teilterme und ersetzen in jedem dieser Teilterme die Vorkommen der entsprechenden Variable (sofern diese **nicht** gebunden ist) durch den Term, der eingesetzt werden soll.

Natürlich müssen auch bestimmte Rechenregeln beachtet werden:

Definition: 12.6 Substitution - Formeln

- Für $p := (A_1, \ldots, A_n) \in R$ gilt: $p\{x \to A\} := (A_1\{x \to A\}, \ldots, A_n\{x \to A\}) \in R$
- Für $p := q \wedge r$ gilt: $p\{x \to A\} := q\{x \to A\} \wedge r\{x \to A\}$
- Für $p := q \vee r$ gilt: $p\{x \to A\} := q\{x \to A\} \vee r\{x \to A\}$
- Für $p := \neg q$ ist $p\{x \to A\} := \neg q\{x \to A\}$
- Für $p := q \Rightarrow r$ gilt: $p\{x \to A\} := q\{x \to A\} \Rightarrow r\{x \to A\}$
- Für $p := q \Leftrightarrow r$ gilt: $p\{x \to A\} := q\{x \to A\} \Leftrightarrow r\{x \to A\}$
- Für $p := \forall x(q)$ und $r := \exists x(s)$ gilt: $p\{x \to A\} := \forall x(q)$ und $r\{x \to A\} := \exists x(s)$
- Für $p := \forall y(q)$ gilt:
  $p\{x \to A\} := \forall y(q)$, falls $x \notin FV(p)$
  $p\{x \to A\} := \forall y(q\{x \to A\})$, falls $x \in FV(p)$ und $y \notin V(A)$
  $p\{x \to A\} := \forall z(q\{y \to z\}\{x \to A\})$, falls $x \in FV(p)$ und $z \notin V(A) \cup FV(q)$ (z neue Variable)
- Für $p := \exists y(q)$ gelten analoge Definitionen wie für $p := \forall y(q)$.

Die oberen Regeln mögen eventuell kompliziert aussehen, sollten aber schon aus der Mathematik bekannt sein. Solange keine Quantoren vorkommen, führen wir die Substitution in allen Teiltermen eines Terms durch, da x in diesen Fällen nicht gebunden ist.

Im drittletzten Fall betrachten wir Formeln, in denen x durch einen Quantor bereits gebunden ist. x darf dann nicht ersetzt werden, da es sich um eine gebundene Variable handelt. Somit bleiben diese Formeln unverändert.

Im vorletzten Punkt betrachten wir eine Formel, bei der im Quantor nicht x, sondern etwas anderes (hier: y) steht. Dann muss man drei Fälle unterscheiden:

1. Wenn x keine freie Variable in der Formel ist, dann führen wir keine Ersetzung durch.
2. x ist eine freie Variable in der Formel **und** y **kommt nicht im Term** A vor, den wir statt x einsetzen wollen. Dann können wir die Ersetzung wie gewohnt durchführen.

3. x ist frei und wir dürften die Ersetzung somit durchführen, aber in dem Term, den wir statt x einsetzen, kommt y vor. Damit hätten wir das Problem, dass y im Bereich von A frei sein sollte, aber durch den Quantor gebunden wird. Daher müssen wir dafür sorgen, dass es innerhalb von A weiterhin frei ist. Das machen wir, indem wir y einfach durch eine neue Variable z ersetzen, die bisher nicht in A vorkommt. Zusätzlich darf z nicht in den freien Variablen von q vorkommen, da wir sonst neue Zusammenhänge herstellen würden, die die Formel aber nicht hergibt.

Eine Regel, die eventuell nicht sofort ersichtlich ist, wird im vorletzten Punkt beschrieben:
Es darf nicht jede Variable substituiert werden, sondern nur freie Variablen. Der Wert, mit dem man diese freie Variable ersetzen will, darf auch nicht eine gebundene Variable sein.

Beispiel:

Bei Relationen, z.B. im Term $x < y$ können bzw. sollen x oder y durch etwas Neues ersetzt werden.

Möchte man beispielsweise folgende Substitution durchführen, wäre das **nicht** erlaubt:

$$\forall x : x + y = 2\{x \to 5\}$$

Ein (komplexeres) Beispiel, das erlaubt ist, wäre folgendes:

$$(x = 1) \Rightarrow (\exists x : x + 1 = 9)\{x \to a\}$$

Wir erinnern uns: das erste x ist eine freie Variable und das zweite x ist eine gebundene Variable. Somit wird das erste x ersetzt, da es eine freie Variable ist und man bekommt folgende Aussage:

$$(a = 1) \Rightarrow (\exists x : x + 1 = 9)$$

12.2.4 Hoare-Kalkül zur Korrektheit von Algorithmen

Definition: 12.7 Hoare-Tripel

Ein **Hoare-Tripel** ist eine Aussage der Form

$$\{p\}S\{q\},$$

wobei gilt:

- S ist ein Ausschnitt eines Algorithmus.
- p ist eine prädikatenlogische Formel über die Variablen in S, genannt **Vorbedingung von** S.
- q ist eine prädikatenlogische Formel über die Variablen in S, genannt **Nachbedingung von** S.

$\{p\}S\{q\}$ ist **erfüllt**, wenn gilt: Gilt p vor Ausführung von S, so gilt q danach, falls S ohne Fehlerabbruch terminiert.

Wir definieren hier also die Begriffe, die wir auf Seite 3 kennengelernt haben, nochmal formal. Vor jeder Anweisung gilt eine Vorbedingung und nach deren Ausführung eine Nachbedingung.

Definition: 12.8 Zuweisungsregel

Für einen Term A und eine Formel p ist das Hoare-Tripel

$$\{p\{x \to A\}\}x \leftarrow A;\{p\}$$

erfüllt.

Die Zuweisungsregel kann man auch als einfache Textersetzung verstehen. In der Formel p wird nach den Vorkommen von Term A gesucht und anschließend durch x in der Formel p ersetzt.

Die Zuweisungsregel verwendet eine Substitution. Wir haben $x \leftarrow A;$ als Ausschnitt aus dem Algorithmus gegeben sowie die Vorbedingung $\{p\{x \to A\}\}$ und die Nachbedingung $\{p\}$.

Beispiel:

Ein Beispiel für das Hoare-Tripel und die Zuweisungsregel wäre folgende Aussage:

$$\{x = a \cdot (i + 1)\}i \leftarrow i + 1;\{x = a \cdot i\}$$

Der mittlere Teil $i \leftarrow i + 1;$ stellt den Ausschnitt aus einem Algorithmus dar.

Der vordere Teil $\{x = a \cdot (i+1)\}$ stellt die Vorbedingung dar. Diese muss gelten, bevor der Ausschnitt vom Algorithmus ausgeführt wird.

Der hintere Teil $\{x = a \cdot i\}$ gibt die Nachbedingung an. Dieser Teil muss gelten, nachdem der Ausschnitt ausgeführt wurde.

Dieses Beispiel ist erfüllt, da es auch die Zuweisungsregel erfüllt. Der Term A wäre in diesem Fall $i + 1$ und unsere Formel p ist $a \cdot (i + 1)$. Die Formel q ergibt sich aus der Vorbedingung und der Substitution und ist $a \cdot i$.

Definition: 12.10 Konsequenzregel

Für Formeln p, q, r, s mit $p \Rightarrow q$ und $r \Rightarrow s$ gilt:

$$\{q\}S\{r\} \Rightarrow \{p\}S\{s\}.$$

Bei der Konsequenzregel verwendet man Umformulierungen der Vor- und/oder Nachbedingung, damit anschließend die Zuweisungsregel angewandt werden kann. Alternativ kann man auch sagen: Die Konsequenzregel dient dem „Verstärken“ der Vorbedingung bzw. dem „Abschwächen“ der Nachbedingung. Das ist im Prinzip das, was wir intuitiv immer machen, wenn wir die Formeln etwas umformen, um besser mit ihnen arbeiten zu können (z.B. wenn wir aus einem $x >= 4$ irgendwann ein $x >= 0$ machen, da es besser für die Schleifeninvariante passt).

Beispiel:

Für ein besseres Verständnis könnte auch eine andere Sichtweise auf die Konsequenzenregel hilfreich sein.

Dabei kann man sich an Puzzlestücken orientieren: S ist ein Puzzlestück, bei dem man nur mit q vorne oder mit r hinten andocken kann. Zusätzlich gibt es noch zwei Puzzlestücke $p \to q$ und $r \to s$, die man dementsprechend anlegen kann. Das bedeutet, man kann das Puzzlestück p an das Puzzlestück q anlegen, bzw. r an s. Somit kann man das Puzzle weiter ausbauen – und die Aussagenkette von {q}S{r} zu {p}(q)S(r){s} verlängern.

Beispiel:

Sei folgendes Hoare-Tripel gegeben:

$$\{x \geqslant 0\}x \leftarrow -x;\{x \leqslant 0\}$$

Das heißt, dass das Vorzeichen von x umgekehrt wird. Sofern x vor der Anweisung nicht-negativ war, wird es danach nicht-positiv sein. Offensichtlich ist das erfüllt, jedoch wollen wir das durch Anwendung unserer Regeln nachweisen. Damit wir die Zusicherungsregel anwenden können, brauchen wir in der Vorbedingung aber den Ausdruck $-x$.

Wir wissen, dass $x \geqslant 0 \Rightarrow -x \leqslant 0$ gilt. Dies entspricht in der Konsequenzregel $p \Rightarrow q$. Die Nachbedingung wollen wir nicht verändern, d.h. wir brauchen kein s. Rein formal könnten wir aber s mit r gleichsetzen: $x \leqslant 0 \Rightarrow x \leqslant 0$, was offensichtlich gilt.

Aufgrund von $x \geqslant 0 \Rightarrow -x \leqslant 0$ können wir die Konsequenzregel anwenden und so $x \geqslant 0$ ersetzen durch $-x \leqslant 0$. Anschließend lässt sich die Zuweisungsregel anwenden, wodurch wir $-x$ durch x ersetzen. Somit haben wir formal zugesichert:

$$\{-x \leqslant 0\}x \leftarrow -x;\{x \leqslant 0\}$$

Beispiel:

Sei folgendes Hare-Tripel gegeben:

$\{?\}$
$x = 1;$
$\{x = 1\}$

Wir wollen hier nun zeigen, dass für ? die Formel true eingesetzt werden kann.
Mit der Zuweisungsregel kommen wir auf:

$\{1 = 1\}$

$x = 1;$
$\{x = 1\}$

Außerdem wissen wir, dass true $\Rightarrow 1 = 1$ gilt. Damit erhalten wir mit der Konsequenzregel:

$\{\text{true}\}$
$x = 1;$
$\{x = 1\}$

Im Sinne der Definition von Folie 13 sind hier also:

$p : \text{true}$
$q : 1 = 1$
$S : x = 5;$
$r : x = 5$
$s : x = 5$

Die folgenden Definitionen bzw. Regeln geben an, wie Hoare-Tripel zusammengefasst werden können:

Definition: 12.12 Regel für sequentielle Komposition

Für Formeln p, q, r gilt:

$$\{p\}S\{q\} \wedge \{q\}T\{r\} \Rightarrow \{p\}ST\{r\}.$$

Die **sequentielle Komposition** kann man als *Transitivität* für Formeln verstehen.

Beispiel:

Seien zwei Hoare-Tripel gegeben:

$$\{a \geqslant 0\}\, a \leftarrow a + 1; \{a \geqslant 1\}$$
$$\{a \geqslant 1\}\, a \leftarrow 1; \{a \geqslant a\}$$

Dann können wir mit der Regel für sequentielle Komposition beide Tripel zusammenfassen:

$$\{a \geqslant 0\}\, a \leftarrow a + 1; a \leftarrow 1; \{a \geqslant a\}$$

Definition: 12.13 Regel für bedingte Anweisungen (Fallunterscheidung)

Für Formeln p, q, r gilt:

$$\{p \wedge q\}S\{r\} \wedge \{p \wedge \neg q\}T\{r\} \Rightarrow \{p\}$$

$$\text{Wenn } q \text{ dann } S \text{ sonst } T \{r\}$$

Dabei ist q die Bedingung der Fallunterscheidung.

Beispiel:

Seien zwei Hoare-Tripel gegeben:

$$\{a \geqslant 0 \wedge b \geqslant 0\}\, a \leftarrow a + b; \{a \geqslant 0 \wedge b \geqslant 0\}$$
$$\{a \geqslant 0 \wedge b < 0\}\, b \leftarrow -(b + 1); \{a \geqslant 0 \wedge b \geqslant 0\}$$

Dann können wir mit der Regel für bedingte Anweisungen beide Tripel zusammenfassen:

$$\text{Tripel1} \wedge \text{Tripel2} \Rightarrow \{a \geqslant 0\}$$

$$\begin{aligned}&\text{Wenn } b \geqslant 0 \text{ dann } a \leftarrow a + b;\\ &\text{sonst } b \leftarrow -(b + 1);\\ &\{a \geqslant 0 \wedge b \geqslant 0\}\end{aligned}$$

Definition: 12.14 Regel für wiederholte Anweisungen

Für Formeln p, q gilt:

$$\{p \wedge q\}S\{p\} \Rightarrow \{p\} \text{ Solange } q \text{ tue } S \{p \wedge \neg q\}.$$

p heißt **Schleifeninvariante**
q ist die Schleifenbedingung

Eine Schleifeninvariante ist eine Zusicherung, die nach jedem Schleifendurchlauf erfüllt ist und folgende Eigenschaften erfüllt:

- Die Invariante gilt vor und nach jedem Schleifendurchlauf.
- Nach dem Eintritt in die Schleife gilt zusätzlich die Schleifenbedingung.
- Nach dem Beenden der Schleife (dem Terminieren) gilt zusätzlich zur Schleifeninvariante die negierte Schleifenbedingung.

Beispiel:

$$\{a \geqslant 0 \land b \geqslant 0\}\, b \leftarrow b - a; \{a \geqslant 0\} \Rightarrow \{a \geqslant 0\}$$
$$\text{Solange } b \geqslant 0 \text{ tue } b \leftarrow b - a; \{a \geqslant 0 \land b < 0\}$$

Korrektheit der Regel für wiederholte Anweisungen:
Beweis mit vollständiger Induktion nach der Anzahl n der Schleifendurchläufe.

1. **Induktionsanfang ($n = 0$)**: Wird S keinmal durchlaufen, so ist $\neg q$ erfüllt. Da p per Voraussetzung vor Schleifenbeginn erfüllt ist, gilt insgesamt $p \land \neg q$
2. **Induktionsschritt ($n > 0$)**:
   Induktionsvoraussetzung: Die Aussage gilt, falls die Schleife nach n Durchläufen terminiert.
   Zu zeigen: Die Aussage gilt, falls die Schleife nach $n + 1$ Durchläufen terminiert.

   Die Schleife wird mindestens 1-mal durchlaufen. Deshalb ist vor dem ersten Schleifendurchlauf q erfüllt. Außerdem ist per Voraussetzung p vor dem ersten Schleifendurchlauf erfüllt, insgesamt gilt also $p \land q$ vor dem ersten Schleifendurchlauf. Per Voraussetzung gilt deshalb p nach dem ersten Schleifendurchlauf. Wegen der Induktionsvoraussetzung gilt dann $p \land \neg q$ nach den nächsten n Durchläufen.

12.2.5 Beispiel für den Nachweis der partiellen Korrektheit: Multiplikation

Wir weisen die partielle Korrektheit anhand eines kleinen Beispiels zur Multiplikation nach.

Eingabe: $a, n \in \mathbb{N}_0$	
$x \leftarrow 0$	
$i \leftarrow 0$	
solange $i < n$	
tue	$x \leftarrow x + a$
	$i \leftarrow i + 1$
Ausgabe: x	

Multiplikation durch Addition

- **Eingabe**: $a, n \in \mathbb{N}_0$
- **Ausgabe**: $x \in \mathbb{N}_0$
- **Funktionaler Zusammenhang**: $x = a \cdot n$

Wir bilden nun Schritt für Schritt Zusicherungen, so dass wir am Ende die partielle Korrektheit nachvollziehen können.

1. Was besagt die Problemspezifikation?
Der Problemspezifikation können wir entnehmen, dass die Eingaben $a, n \in \mathbb{N}_0$, die Ausgabe $x \in \mathbb{N}_0$ und am Ende soll $x = a \cdot n$ gelten. Somit können wir als Vorbedingungen des Algorithmus festhalten, dass $a \geqslant 0$ und $n \geqslant 0$. Nach der Ausführung des Algorithmus soll $x = a \cdot n$ und $x \geqslant 0$ gelten.

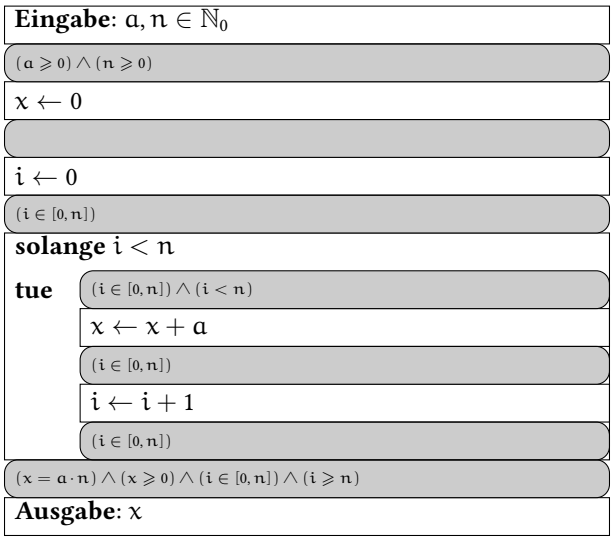
2. Schleifenbedingung
Um später Aussagen darüber treffen zu können, was in der Schleife passiert und was danach gilt, notieren wir, was wir aus der Schleifenbedingung ablesen können: Nach Eintritt in die Schleife muss $i < n$ gelten. Nach Terminierung der Schleife gilt die **negierte** Schleifenbedingung $i \geqslant n$.

3. Schleifenzählvariable

Der funktionale Zusammenhang, der am Ende nach Ausführung des Algorithmus steht, muss sich aus den Schritten im Algorithmus ergeben. Der Algorithmus besteht in erster Linie aus einer Schleife, das Ergebnis wird dort nach und nach berechnet. Es ist also abhängig von den Iterationen der Schleife. Daher ist es zielführend, wenn wir eine Aussage über die Iterationen machen können. Sie hängen von der Schleifenzählvariable i ab.

Was können wir also über die Schleifenzählvariable sagen? Vor dem Schleifeneintritt bekommt i den Wert 0 zugewiesen, es gilt also $i = 0$. Zu Beginn der Schleife hat i stets einen Wert zwischen 0 und n (ausschließlich n), d.h. $i \in [0, n[$[1]. Am Ende der Schleife wurde i um 1 erhöht, somit gilt $i \in [1, n]$ (nun einschließlich n). Nach der Schleife hat i den Wert n. All diese Ergebnisse lassen sich vereinigen zu $0 \leqslant i \leqslant n$ oder auch $i \in [0, n]$.

Aus den bisherigen Erkenntnissen lassen sich bereits folgende Zusicherungen treffen:



4. Anwendung der Zuweisungsregel

Mit Hilfe der Zuweisungsregel können wir aus unseren bisherigen Zusicherungen noch einige weitere formulieren. Dazu müssen wir jedoch manchmal eine bestehende Zusicherung umformulieren, da die Zuweisungsregel wie eine Art Textersetzung funktioniert. Für den Inkrement von i brauchen wir eine Vorbedingung, die $i + 1$ enthält. Daher formen wir die Zusicherung $i < n$ um zu $i + 1 \leqslant n$. Durch die Zuweisung $i \leftarrow i + 1$ wird $i + 1$ aus der Vorbedingung ersetzt, so dass die Nachbedingung $i \leqslant n$ lautet.[2]

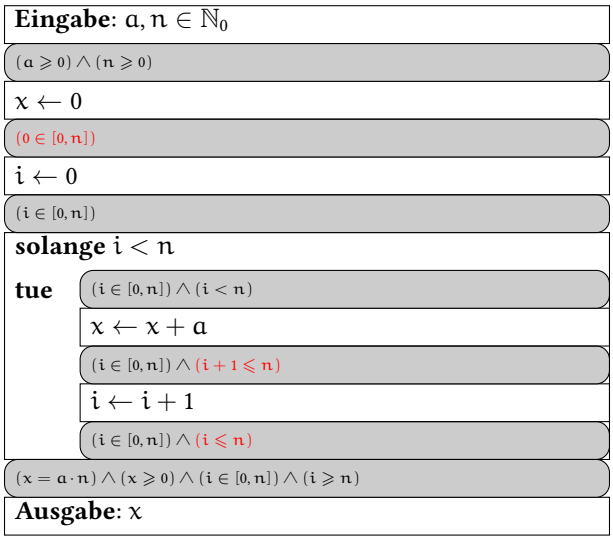
Wir können die Zuweisungsregel auch an einer anderen Stelle noch verwenden: Da $i \in [0, n]$ nach der Zuweisung $i \leftarrow 0$ gilt, muss in der zugehörigen Vorbedingung $0 \in [0, n]$ gelten. Hierzu müssen wir sicher sein, dass $n \geqslant 0$, da wir für $n < 0$ ein leeres Intervall haben, d.h. eine Menge ohne Zahlen (leere Menge). Nach den Vorbedingungen der Problemspezifikation gilt $n \geqslant 0$, somit

[1] In diesem Kapitel gehen wir davon aus, dass Intervalle aus ganzen Zahlen bestehen.

[2] Mit einem konkreten i und n wird auch klar, dass daraus nichts Falsches wird: Wenn $i = 3$ und $n = 4$, so lautet unsere ursprüngliche Zusicherung zum Schleifeneintritt $3 < 4$. Diese kann man auch anders schreiben als $3 + 1 \leqslant 4$ oder auch $4 \leqslant 4$.

muss das Intervall mindestens die Zahl 0 enthalten (im Fall $n = 0$ enthält es nur die 0). Damit ist auch sichergestellt, dass $0 \in [0, n]$ gilt.

Somit lassen sich nun folgende Zusicherungen treffen (Ergänzungen sind rot markiert):



Abgesehen davon, dass wir die Angaben aus der Problemspezifikation abgeschrieben haben, sind die Zusicherungen, die wir bislang notiert haben, sehr allgemein und somit für jede Schleife dieser Art verwendbar. Nun wird es Zeit, dass wir uns konkret mit der Lösung des Problems auseinandersetzen.

5. Schleifeninvariante

Um einen Zusammenhang zwischen der Schleife und der Ausgabe herzustellen, brauchen wir eine Schleifeninvariante. Damit wir eine Idee bekommen, was als Schleifeninvariante geeignet ist, schauen wir uns genau an, was in der Schleife passiert, z.B. für die Eingabe $a = 5, b = 3$ (Zahlen willkürlich so gewählt, dass ein paar Iterationen zustande kommen):

Schritt	1. Iteration	2. Iteration	3. Iteration	4. Iteration
$i < n$?	$0 < 3$	$1 < 3$	$2 < 3$	$3 < 3$
$x \leftarrow x + a$	$x \leftarrow 0 + 5$	$x \leftarrow 5 + 5$	$x \leftarrow 10 + 5$	
$i \leftarrow i + 1$	$i \leftarrow 0 + 1$	$i \leftarrow 1 + 1$	$i \leftarrow 2 + 1$	
Ausgabe				Ausgabe: 15

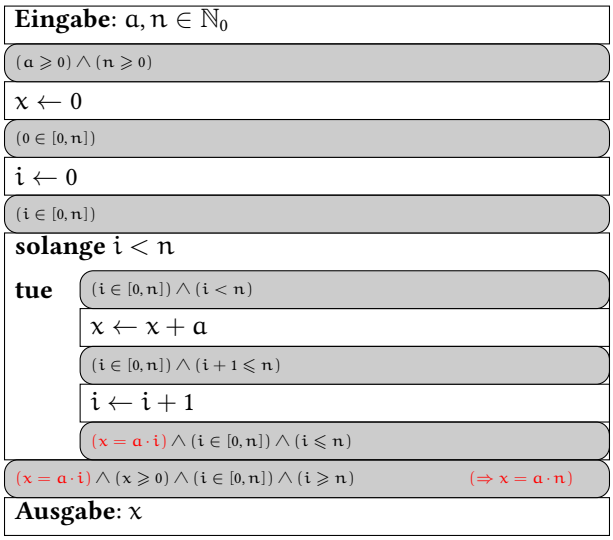
Wir sehen, dass sich in jedem Durchlauf i und x verändern. Konkret ist $x = a + a + ... + a$, wobei a i-mal aufsummiert wird. Um den funktionalen Zusammenhang aus der Schleife herzuleiten, würde es uns helfen, wenn wir am Ende des Algorithmus statt $x = a \cdot n$ schreiben könnten, dass $x = a \cdot i$.

Wenn wir die Zusicherungen ansehen, die wir am Ende des Algorithmus bereits notiert haben, sehen wir, dass wir mehrere Aussagen über i getroffen haben: $i \in [0, n]$ und $i \geqslant n$. i ist also eine Zahl zwischen 0 und n und gleichzeitig $\geqslant n$. Daraus ergibt sich, dass $i = n$. Somit können

wir auch den funktionalen Zusammenhang $x = a \cdot n$ am Ende des Algorithmus ersetzen durch $x = a \cdot i$.

Indem wir unsere beispielhafte Ausführung nochmal anschauen und rückwärts folgern, erkennen wir, dass die Nachbedingung $x = a \cdot i$ auch am Ende jedes Schleifendurchlaufs gilt.

Nun schauen unsere Zusicherungen so aus:

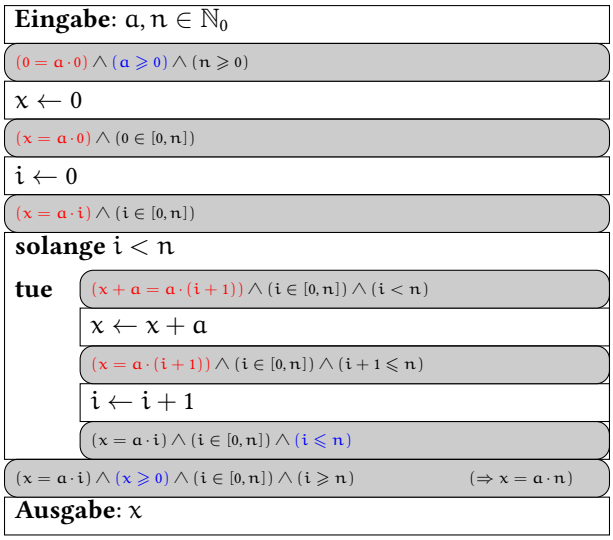


6. Ausbau der Schleifeninvariante mit Hilfe der Zuweisungsregel
In der Schleife können wir die Zuweisungsregel anwenden: Wir haben die Nachbedingung $x = a \cdot i$ vor der Zuweisung $i \leftarrow i + 1$ stehen. Daraus ergibt sich die Vorbedingung $x = a \cdot (i + 1)$. Wenn wir diese als Nachbedingung der Zuweisung $x \leftarrow x + a$ betrachten, erhalten wir unter weiterer Anwendung der Zuweisungsregel die zugehörige Vorbedingung $x + a = a \cdot (i + 1)$.

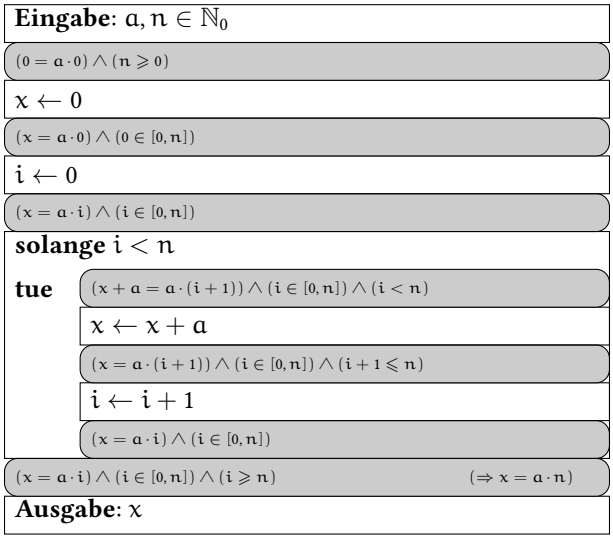
Die letzte Vorbedingung lässt sich mathematisch umformen zu $x = a \cdot i$, was auch als Vorbedingung für die Schleife verwendbar ist. Vor dem ersten Aufruf der Schleife ist $x = 0$ und $i = 0$, wodurch $x = a \cdot i$ der Aussage $0 = a \cdot 0$ entspricht, was für ein beliebiges a wahr ist. Somit haben wir sicher eine passende Schleifeninvariante gefunden und die entsprechenden Zusicherungen in unserer Schleife integriert.

Danach lässt sich die Zuweisungsregel noch weitere zwei Mal anwenden: Aus der Nachbedingung $x = a \cdot i$ nach der Anweisung $i \leftarrow 0$ wird die Vorbedingung $x = a \cdot 0$. Das ist wahr, da x an dieser Stelle den Wert 0 hat und $a \cdot 0$ sicher 0 ergibt. Unter nochmaliger Anwendung der Zuweisungsregel erhalten wir $0 = a \cdot 0$, was für ein beliebiges a wahr ist.

Abschließend erhalten wir:



Die blau eingefärbten Teile sind für den Nachweis der partiellen Korrektheit nicht nötig. Wenn wir sie weg lassen, erhalten wir folgende kompakte Version:



12.2.6 Überblick: Wie weist man die partielle Korrektheit nach?

Vorausetzungen für den Nachweis der partiellen Korrektheit
Um formale Zusicherungen formulieren zu können, muss der Algorithmus in einer **formalen programmiersprachenunabhängigen Darstellung** vorliegen, d.h. als Pseudocode, Struktogramm oder Programmablaufplan **ohne natürlichsprachliche Formulierungen**.

Formale Darstellung:
Erlaubt sind mathemetische Operationen und Funktionen:

- Wertzuweisung ($\leftarrow$)
- arithmetische Operationen ($+, -, /, *, \div, \text{mod}$)
- Mengenoperationen ($\cap, \cup, \setminus$)
- Vergleiche von mathematischen Objekten ($\leqslant, \geqslant, <, >, =, \neq$)
- Logische Operationen ($\neg, \wedge, \vee$)
- Mathematische Funktionen ($x^n, \log_b(x), b^x, \sin(x), \cos(x), \sqrt{x}, |\cdot|, \ldots$)
- ...

Allgemeines Vorgehen zum Nachweis der partiellen Korrektheit

1. Formuliere Vorbedingung des Algorithmus
   (= Spezifikation der Eingabedaten)
2. Formuliere Nachbedingung des Algorithmus
   (= Spezifikation des funktionalen Zusammenhangs)
3. Finde ausgehend von der Vorbedingung Anweisung für Anweisung passende Zwischenzusicherungen
4. Verliere dabei das Ziel (Nachbedingung des Algorithmus) nicht aus den Augen;
   schließe u.U auch rückwärts
5. Bei der Formulierung von Schleifeninvarianten stelle man sich die Frage: Wieso ergeben die Zwischenergebnisse nach jedem Schleifendurchlauf am Ende das gewünschte Endergebnis?

Die Vorbedingung lässt sich oft leichter formulieren, wenn man von der Nachbedingung ausgeht.

12.2.7 Beispiel für den Nachweis der partiellen Korrektheit: Palindromtest

Als weiteres Beispiel für den Nachweis der partiellen Korrektheit betrachten wir das Testen einer Zeichenkette, ob sie ein Palindrom ist. Als Palindrom werden hier Wörter, Wortreihen oder Sätze bezeichnet, die rückwärts gelesen genau denselben Text ergeben wie vorwärts.

- **Eingabe**: $a_1 \ldots a_n \in A^+$
- **Ausgabe**: $k \in \{0, 1\}$
- **Funktionaler Zusammenhang**:
  $((k = 1) \wedge (\forall i \in \{1, 2, \ldots, n \div 2\}(a_i = a_{n+1-i})))$
  $\vee$
  $((k = 0) \wedge (\exists i \in \{1, 2, \ldots, n \div 2\}(a_i \neq a_{n+1-i})))$

In Worten erklärt:

- Fall 1: Die Ausgabe des Algorithmus ist $k = 1$ und es liegt ein Palindrom vor, d.h. das erste und das letzte Zeichen stimmen überein, das zweite und das vorletzte usw.
- Fall 2: Die Ausgabe ist $k = 0$ und es liegt kein Palindrom vor, d.h. an mindestens einer Stelle stimmt das i-te Zeichen **nicht** mit dem i-ten Zeichen von hinten überein.

```
1 Algorithmus: Palindromtest
  Eingabe: a_1 ... a_n ∈ A^+
2 m ← 1;
3 solange (m ⩽ n ÷ 2) ∧ (a_m = a_{n+1−m}) tue
4 |  m ← m + 1;
5 wenn m > n ÷ 2 dann
6 |  k ← 1;
7 sonst
8 |  k ← 0;
  Ausgabe: k
```

Der Algorithmus gliedert sich in zwei Teile: In der *solange*-Schleife wird die Palindrom-Eigenschaft überprüft bis die Mitte erreicht ist oder die Eigenschaft verletzt ist. Die *wenn-dann*-Fallunterscheidung wird erreicht, sobald eine dieser beiden Anforderungen nicht mehr erfüllt ist. Hier wird überprüft, welche verletzt ist, um dann die Rückgabe passend zu setzen.

Wir bilden nun wieder Schritt für Schritt Zusicherungen, um die partielle Korrektheit nachzuweisen.

1. Was sagt die Problemspezifikation?
Wir wissen, dass die Eingabe $a_1 \dots a_n \in A^+$ und der funktionale Zusammenhang $((k = 1) \wedge (\forall i \in \{1, 2, \dots, n \div 2\}(a_i = a_{n+1-i}))) \vee ((k = 0) \wedge (\exists i \in \{1, 2, \dots, n \div 2\}(a_i \neq a_{n+1-i})))$.

2. Schleifenbedingung und Fallunterscheidung
Nach Eintritt der Schleife muss die Schleifenbedingung $(m \leqslant n \div 2) \wedge (a_m = a_{n+1-m})$ erfüllt sein, nach Verlassen der Schleife muss die negierte Schleifenbedingung gelten. Da die Schleifenbedingung aus zwei Teilen besteht, die mit einem logischen Und verbunden sind, ist die negierte Schleifenbedingung die Negation der beiden Teile, die durch ein logisches Oder verbunden sind: $(m > n \div 2) \vee (a_m \neq a_{n+1-m})$.

Bei der Fallunterscheidung muss im Wenn-Fall die Fallbedingung $m > n \div 2$ gelten, wohingegen sie im Sonst-Fall nicht gelten darf, d.h. $m \leqslant n \div 2$ gelten muss. Diese beiden Zusicherungen gelten auch nach den Zuweisungen auf k innerhalb der Fallunterscheidung, da dabei keine Variable aus diesen Zusicherungen verändert wird.

3. Verbindung von Funktionalem Zusammenhang mit Fallunterscheidung
Der funktionale Zusammenhang besteht aus zwei Teilen, die jeweils auf einen Teil der Fallunterscheidung zurückgehen. Im zweiten Teil wird verlangt, dass $m \leqslant n \div 2$ und zugleich $k = 0$ gelten muss. Dies geht einher mit der Zuweisung $k \leftarrow 0$ im Sonst-Fall der Fallunterscheidung. Der erste Teil hingegen fordert, dass $k = 1$ gelten muss, was zur Zuweisung $k \leftarrow 1$ im Wenn-Fall passt. Entsprechend muss der erste Teil des funktionalen Zusammenhangs am Ende des Wenn-Falls zugesichert werden und der zweite Teil am Ende des Sonst-Falls.

Anhand der bisherigen Erkenntnisse können wir folgende Zusicherungen treffen:

Eingabe: $a_1 \dots a_n \in A^+$

1 $a_1 \dots a_n \in A^+$
2 $m \leftarrow 1$;
3 **solange** $(m \leqslant n \div 2) \wedge (a_m = a_{n+1-m})$ **tue**
4 $\quad (m \leqslant n \div 2) \wedge (a_m = a_{n+1-m})$
5 $\quad m \leftarrow m + 1$;
6 $(m > n \div 2) \vee (a_m \neq a_{n+1-m})$
7 **wenn** $m > n \div 2$ **dann**
8 $\quad m > n \div 2$
9 $\quad k \leftarrow 1$;
10 $\quad (m > n \div 2) \wedge (\forall i \in \{1, 2, \dots, n \div 2\}(a_i = a_{n+1-i})) \wedge (k = 1)$
11 **sonst**
12 $\quad m \leqslant n \div 2$
13 $\quad k \leftarrow 0$;
14 $\quad (m \leqslant n \div 2) \wedge (\exists i \in \{1, 2, \dots, n \div 2\}(a_i \neq a_{n+1-i})) \wedge (k = 0)$
15 $((k = 1) \wedge (\forall i \in \{1, 2, \dots, n \div 2\}(a_i = a_{n+1-i}))) \vee ((k = 0) \wedge (\exists i \in \{1, 2, \dots, n \div 2\}(a_i \neq a_{n+1-i})))$

Ausgabe: k

4. Schleifeninvariante

Wir benötigen noch eine Schleifeninvariante, um nachzuweisen, wie der funktionale Zusammenhang erreicht wird. Sie sollte die Frage beantworten *Wieso ergeben die Zwischenergebnisse nach jedem Schleifendurchlauf am Ende das gewünschte Ergebnis?* Werfen wir hierzu einen beispielhaften Blick auf ein paar Iterationen:
Wählen wir als Eingabe z.B. $abacba$, so ist $n = 6$ und $n \div 2 = 3$

Schritt	1. Iteration	2. Iteration	3. Iteration
$m \leqslant n \div 2$?	$1 \leqslant 3$	$2 \leqslant 3$	$3 \leqslant 3$
$a_m = a_{n+1-m}$?	$a = a$	$b = b$	$a \neq c$
$m \leftarrow m + 1$	$m \leftarrow 1 + 1$	$m \leftarrow 2 + 1$	

Das einzige, was sich in der Schleife verändert, ist m, d.h. wir brauchen m für unsere Schleifeninvariante. Gleichzeitig gibt m an, wie viel vom eingegebenen Wort übereinstimmt bzw. an welcher Stelle es nicht mehr übereinstimmt. Schauen wir uns nochmal den funktionalen Zusammenhang an:

$$((k = 1) \wedge (\forall i \in \{1, 2, \dots, n \div 2\}(a_i = a_{n+1-i}))) \vee ((k = 0) \wedge (\exists i \in \{1, 2, \dots, n \div 2\}(a_i \neq a_{n+1-i})))$$

Im Fall $k = 1$ wollen wir für alle Werte von 1 bis $n \div 2$ aussagen können, dass das i-te Zeichen von vorne mit dem i-ten Zeichen von hinten übereinstimmt. In unserer Schleife können wir das immer für die Werte von 1 bis $m - 1$ aussagen, an der Position m steht evtl. das Zeichen, bei dem die Aussage nicht mehr zutrifft. Damit können wir aber auch sagen, welchen Wert i im Fall $k = 0$ annimmt, nämlich den Wert m. Somit verschwindet der Existenz-Quantor überall in unseren Zusicherungen, da wir einen konkreten Wert angeben können – wir ersetzen das durch den Existenz-Quantor gebundene i durch m.

Gleichzeitig haben wir eine Abschwächung des funktionalen Zusammenhangs gefunden, die als Schleifeninvariante verwendbar ist: Der vordere Teil des funktionalen Zusammenhangs gilt bis $m-1$, also können wir schreiben $\forall i \in \{1,2,\ldots,m-1\}(a_i = a_{n+1-i})$.

Die restlichen Schritte sollten schon aus Kapitel 12.2.5 klar sein. Damit erhalten wir schließlich:

Eingabe: $a_1 \ldots a_n \in A^+$
1 $\{\forall i \in \emptyset(a_i = a_{n+1-i})\}$
2 $m \leftarrow 1;$
3 $\{\forall i \in \{1,\ldots,m-1\}(a_i = a_{n+1-i})\}$
4 **solange** $(m \leqslant n \div 2) \wedge (a_m = a_{n+1-m})$ **tue**
5 $\{\forall i \in \{1,\ldots,(m+1)-1\}(a_i = a_{n+1-i})$
6 $m \leftarrow m+1;$
7 $\{\forall i \in \{1,\ldots,m-1\}(a_i = a_{n+1-i})\}$
8 $\{\forall i \in \{1,\ldots,m-1\}(a_i = a_{n+1-i}) \wedge ((m > n \div 2) \vee (a_m \neq a_{n+1-m}))\}$
9 **wenn** $m > n \div 2$ **dann**
10 $\{\forall i \in \{1,\ldots,m-1\}(a_i = a_{n+1-i}) \wedge (m > n \div 2)\}$
11 $k \leftarrow 1;$
12 $\{\forall i \in \{1,2,\ldots,n \div 2\}(a_i = a_{n+1-i}) \wedge (k = 1)\}$
13 **sonst**
14 $\{(m \leqslant n \div 2) \wedge (a_m \neq a_{n+1-m})\}$
15 $k \leftarrow 0;$
16 $\{(m \leqslant n \div 2) \wedge (a_m \neq a_{n+1-m}) \wedge (k = 0)\}$
17 $\{(\forall i \in \{1,2,\ldots,n \div 2\}(a_i = a_{n+1-i}) \wedge (k = 1))$
18 $\vee((a_m \neq a_{n+1-m}) \wedge (m \leqslant n \div 2) \wedge (k = 0))\}$
Ausgabe: k

12.3 Totale Korrektheit

12.3.1 Wie kann man die totale Korrektheit nachweisen?

Für den Nachweis der totalen Korrektheit muss man zusätzlich zur partiellen Korrektheit zeigen:
Der Algorithmus terminiert für alle (gültigen) Eingaben

Die generelle Idee ist also:
Zeige für jede Schleife mit Hilfe einer **Terminierungsfunktion**, dass diese (für jede Eingabe) nach endlich vielen Durchläufen abbricht.

Was ist eine Terminierungsfunktion?

Definition: 12.16 Terminierungsfunktion

Sei $\mathtt{Alg}$ ein Algorithmus und $B(x_1, \ldots, x_n)$ eine Schleifenbedingung von $\mathtt{Alg}$, die von den Variablen $x_1, \ldots, x_n$ abhängt.

Für eine Belegung der Variablen $x_1, \ldots, x_n$ mit Werten $a_1, \ldots, a_n$ **vor** einem Schleifendurchlauf sei $a'_1, \ldots, a'_n$ deren Belegung **nach** dem Schleifendurchlauf.

Eine **Terminierungsfunktion** (zu B) ist eine Funktion

$$T(x_1, \ldots, x_n) =: y \in \mathbb{Z}$$

mit folgender Eigenschaft: Für jede Belegung $a_1, \ldots, a_n$ von $x_1, \ldots, x_n$, die $B(a_1, \ldots, a_n)$ erfüllt, gilt

- $T(a_1, \ldots, a_n) > 0$
- $T(a_1, \ldots, a_n) > T(a'_1, \ldots, a'_n)$

Was bedeutet eine Terminierungsfunktion?
Existiert zu einer Schleife eine Terminierungsfunktion T, so terminiert die Schleife nach endlich vielen Durchläufen, denn:

- Der Wert von T ist positiv, solange die Schleifenbedingung erfüllt ist.
- Da der Wert von T ganzzahlig ist und mit jedem Schleifendurchlauf abnimmt, muss er nach endlich vielen Schleifendurchläufen 0 oder negativ werden
- Dann ist die Schleifenbedingung nicht mehr erfüllt, d.h. die Schleife terminiert.

12.3.2 Wie entwirft man eine Terminierungsfunktion?

In der Regel werden Terminierungsfunktionen zu Algorithmen aufgestellt, in denen Schleifen vorkommen und somit die Terminierung dieser Algorithmen abhängig vom Zustand der Schleifenvariable ist. Basierend darauf verfolgt man eine Reihe an Schritten, um eine passende Terminierungsfunktion zu finden:

- Analyse der Schleife innerhalb des Algorithmus.
- Die Schleifenbedingung gibt einen Hinweis auf den Endzustand der Schleifenvariable.
- Innerhalb der Schleife sollte eine oder mehrere Anweisungen dazu führen, die Schleifenvariable zu ändern.
- Nun sollten Endzustand und (ggf. rückwärts) schrittweises Nachvollziehen der Anweisungen zum Ändern der Schleifenvariable Ideen zum Entwurf der Terminierungsfunktion geben.

Allgemein erhält man meist sehr einfach eine Terminierungsfunktion, indem man die Schleifenbedingung so umformt, dass ihr Vergleich auf > 0 lautet.

Beispiel für eine Terminierungsfunktion
Schleife:

```
1 i ← 1;
2 solange i ⩽ 10 tue
3 |  i ← i + 1;
```

Entwurf der Terminierungsfunktion
Die Schleifenbedingung lautet $i \leqslant 10$. Wir formen diese um:

$$i \leqslant 10 \Leftrightarrow 10 - i \geqslant 0 \Leftrightarrow 11 - i > 0$$

Damit erhalten wir $T(i) := 11 - i$

Nachweis:
Wir weisen nun nach, dass unsere gesuchte Terminierungsfunktion die geforderten Eigenschaften erfüllt. Für $(i \leqslant 10)$ gilt:

1. Für jede gültige Eingabe muss die Schleifenbedingung positiv sein. Der höchte Wert, den i annehmen kann, ist 10.
   $T(i) = 11 - i \geqslant 11 - 10 = 1 > 0$.
   Damit ist die Rückgabe der Terminierungsfunktion auch beim größten i positiv.
2. Der Wert von i muss in jeder Schleifeniteration abnehmen:
   $T(i) = 11 - i$
   $T(i') = 11 - i' = 11 - (i + 1) = 10 - i$
   $11 - i > 10 - i \Leftrightarrow 1 > 0$.
   Damit ist der Wert der Terminierungsfunktion in der nächsten Iteration kleiner als in der aktuellen.