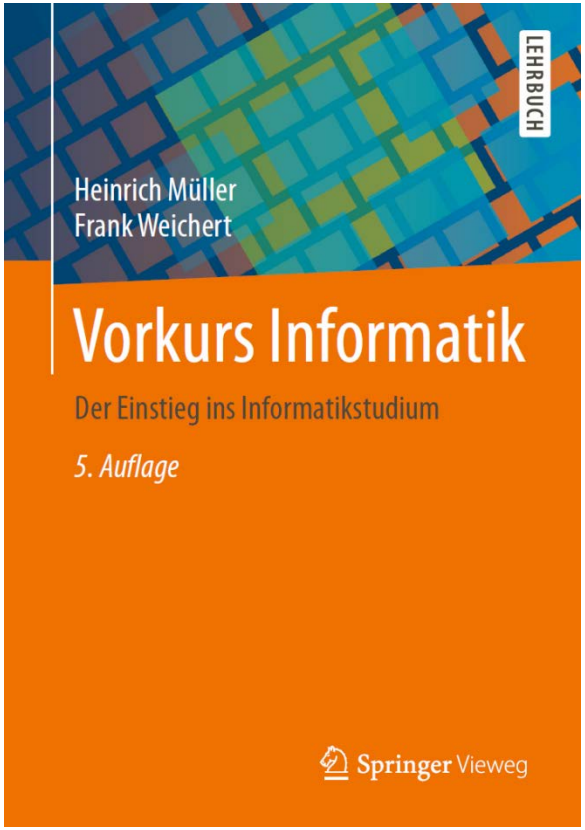
Informatik (I und II)

im Studiengang Game Design

Mike Scherfner

Literatur

Die Inhalte dieser Vorlesung basieren auf dem Buch von Müller und Weichert:



1. Informatik

Das Wort Informatik ist die Verschmelzung von Information und Mathematik. Geprägt wurde die Bezeichnung Informatik von Karl Steinbuch und kann auf seine erste Publikation

„Informatik: Automatische Informationsverarbeitung“

von 1957 zurückgeführt werden.

Was ist Informatik?

Typische Begriffe der Informatik:

Information, Informationssysteme, Computer, EDV, Rechner, Programmierung, Programmiersprachen, Software, Hardware, Internet, Textverarbeitung, Computerspiele

Definition 1:

Informatik ist die Wissenschaft von der systematischen Verarbeitung von Information, besonders der automatischen, mit Hilfe von Computern.

Definition 2:

Informatik ist die Wissenschaft von den Algorithmen und deren Verarbeitung durch Computer.

Was ist Informatik?

Information:
abstraktes Abbild (Modell) von Objekten der realen Welt

Informationserfassung:
Abstraktion (Modellbildung) - das Wesentliche vom Unwesentlichen trennen

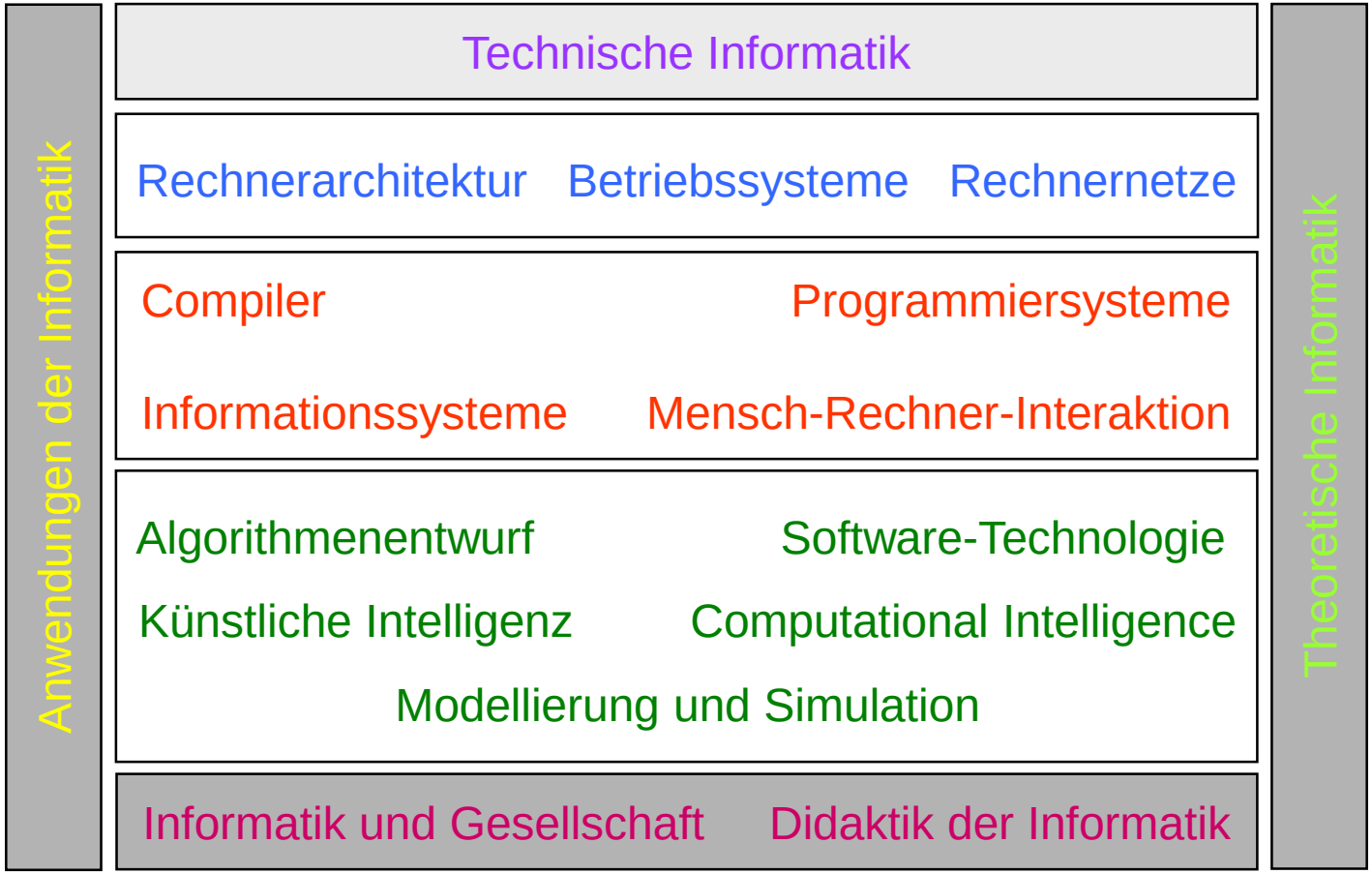
Informationsverarbeitung: Algorithmen und Systeme

Informatiker(innen) müssen

- von den Objekten und Vorgängen der realen Welt abstrahieren
- mit abstrakten Objekten umgehen.

Dazu gibt es Methoden - ganz praktische, die manchmal aber auch ziemlich „mathematisch“ aussehen können.

Teilgebiete der Informatik



Eine grobe Einteilung der Informatik

Technische Informatik:

Befasst sich mit der inneren Struktur und dem Bau von Computern und allen damit zusammenhängenden technischen Fragen

Praktische Informatik:

Umfasst die Prinzipien und Techniken der Programmierung

Angewandte Informatik:

Bildet die Brücke zwischen den Methoden der Informatik und Anwendungsproblemen

Theoretische Informatik:

Entwickelt mathematische Modelle von Computern und Hilfsmittel zu ihrer präzisen Beschreibung

2. Vom Problem über den Algorithmus zum Programm

Vom Problem über den Algorithmus zum Programm

Problem

Maschine

Vom Problem über den Algorithmus zum Programm

Reale Welt **Problem**

Rechner

Vom Problem über den Algorithmus zum Programm

Reale Welt **Problem**

Algorithmus

Programm

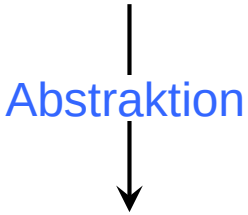
Rechner **Maschine**

Vom Problem über den Algorithmus zum Programm

Reale Welt	**Problem**
Abstrakte Objekte	**Algorithmus**
Informationsverarbeitendes System	**Programm**
Rechner	**Maschine**

Vom Problem über den Algorithmus zum Programm

Reale Welt — **Problem**

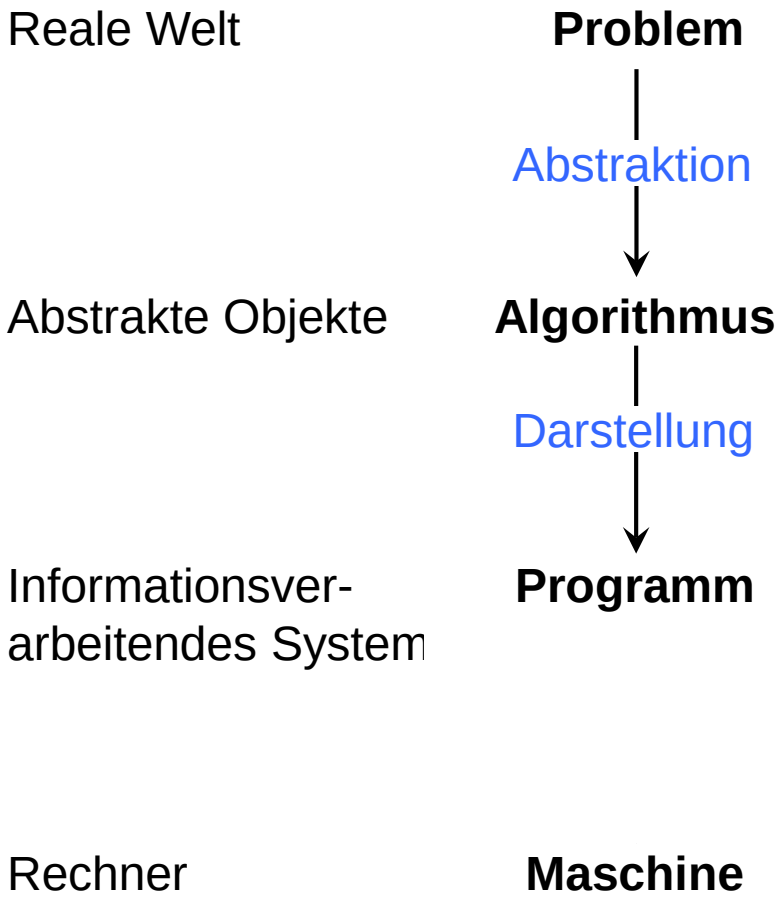


Abstrakte Objekte — **Algorithmus**

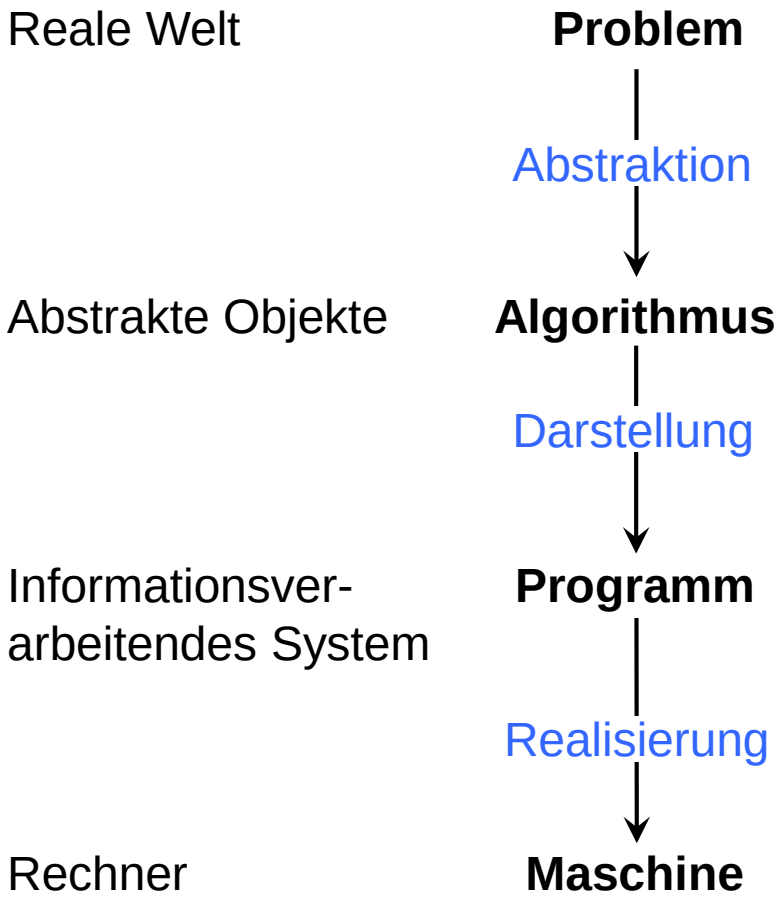
Informationsverarbeitendes System — **Programm**

Rechner — **Maschine**

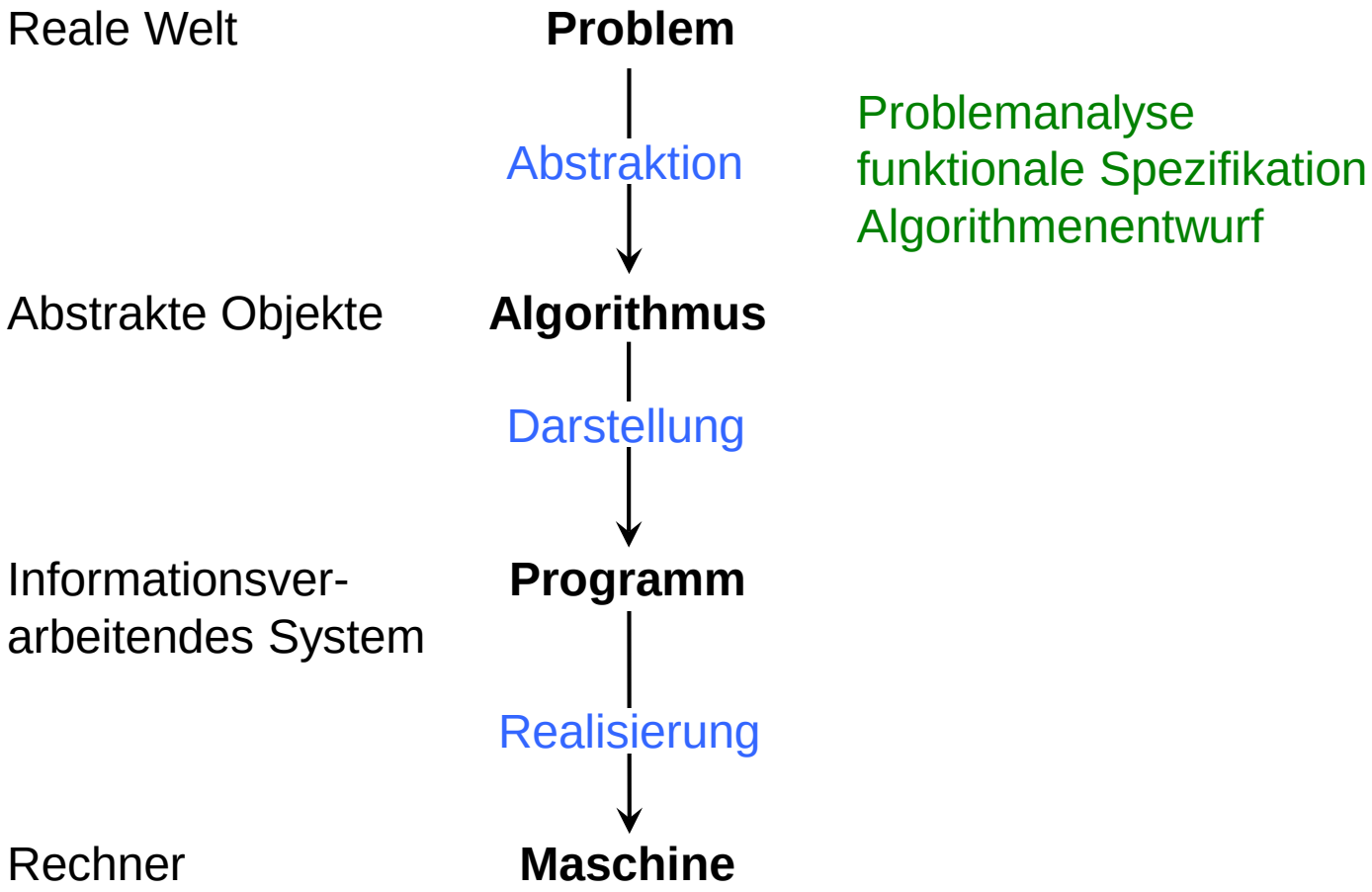
Vom Problem über den Algorithmus zum Programm



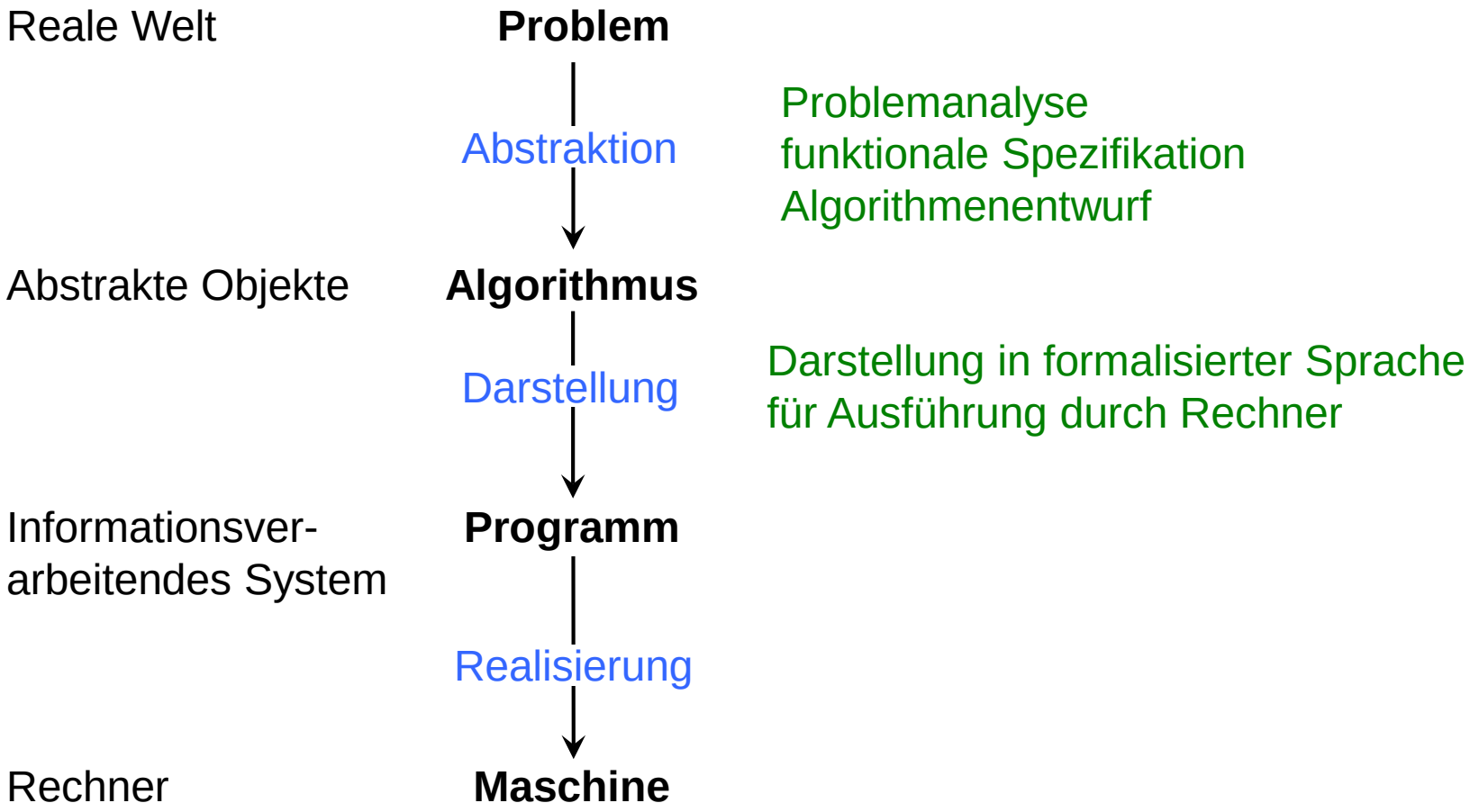
Vom Problem über den Algorithmus zum Programm



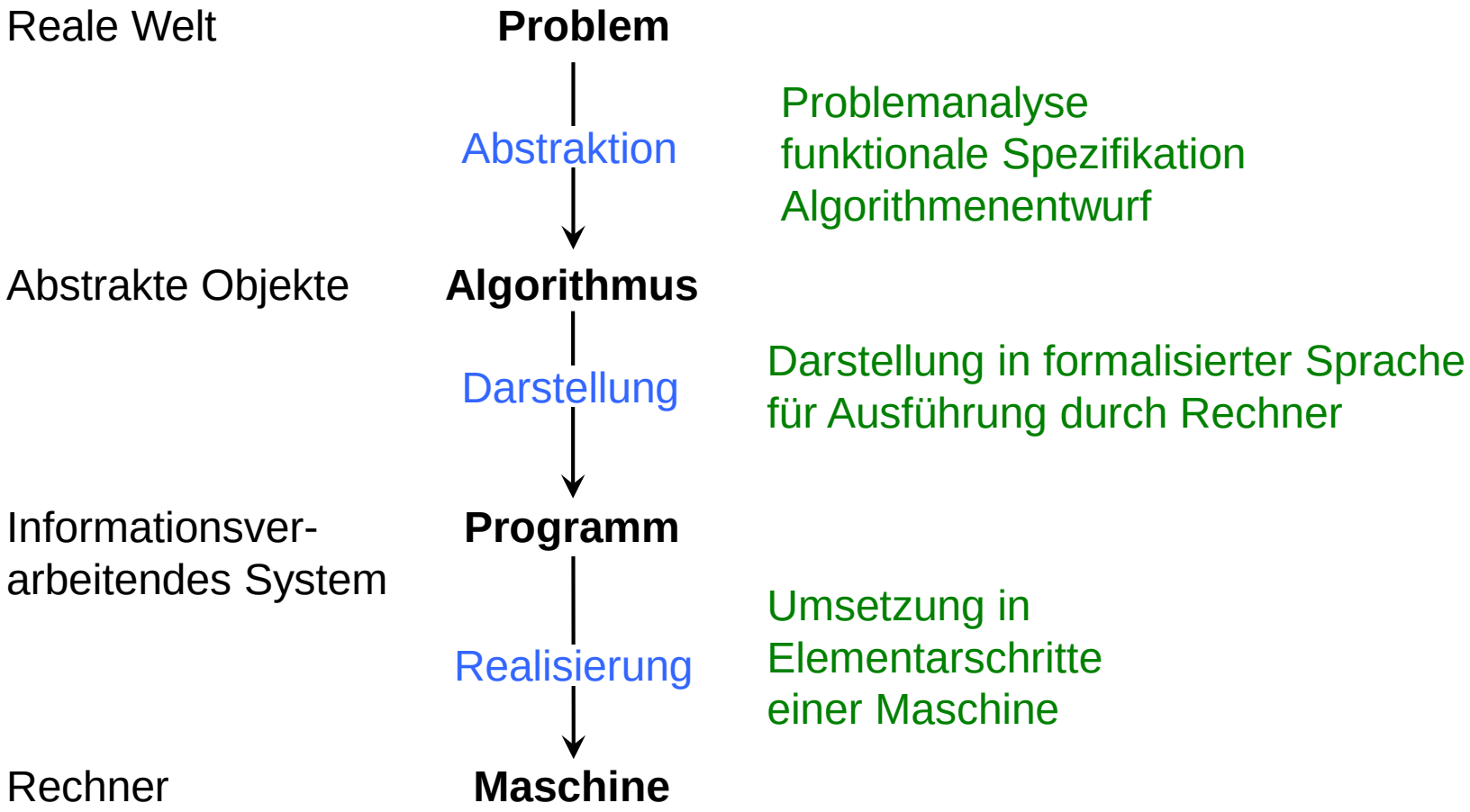
Vom Problem über den Algorithmus zum Programm



Vom Problem über den Algorithmus zum Programm



Vom Problem über den Algorithmus zum Programm



Algorithmus

Endliche Vorschrift zur eindeutigen Zuordnung von Ausgabegrößen zu Eingabegrößen in einer endlichen Zahl von Schritten.

Bsp.: Rezept, Montageanleitung, Gauß-Algorithmus

Bsp.: Backrezept „Krustelkuchen“

Zutaten: 1 kg Kartoffeln, 1 mittelgroße Zwiebel, 2 EL Mehl, 2 Eier, Salz, Pfeffer, Majoran, 60 g Butter,
2-3 EL Semmelbrösel, 125 ml saure Sahne

Zubereitung:

- Backofen auf 200 Grad vorheizen.
- Kartoffeln schälen, waschen, trockentupfen, fein reiben, in einem Passiertuch Flüssigkeit sorgfältig herausdrücken.
- Zwiebel schälen, fein würfeln.
- Zwiebel, Mehl und Eier zu den Kartoffeln geben, mit Salz, Pfeffer und Majoran würzen und aller gut mischen.
- Eine Kastenform (1 l Füllmenge) mit etwas flüssiger Butter fetten.
- Kartoffelmasse hineinschichten und Oberfläche mit der restlichen Butter begießen.
- 1 Stunde backen, nach 30 Min. Oberfläche mit Semmelbröseln bestreuen und saure Sahne daraufgießen.

1. Suche im Netz nach „Turing-Maschine“ und beschreibe diese.

2. Verfasse eine Kurzbiographie von Alan Turing und gib einen kurzen Überblick über die Geschichte der Informatik.

Vom Problem über den Algorithmus zum Programm

Algorithmus (*konkret)*

Eine Folge „einfacher“ Anweisungen, die folgende Eigenschaften hat:

- *Endlichkeit:* Die Beschreibung ist endlich lang.
- *Terminierung:* Nach Durchführung endlich vieler Operationen kommt das Verfahren zum Stillstand.
- *Eindeutige Reihenfolge:* Die Reihenfolge, in der die Operationen anzuwenden sind, ist festgelegt.
- *Eindeutige Wirkung:* Die Wirkung jeder Anweisung der Anweisungsfolge und damit der gesamten Folge ist eindeutig festgelegt.

Vorgehensweise bei der Lösung von Problemen

1. Problem formulieren
2. Problemanalyse, Problemabstraktion, Problemspezifikation
3. Algorithmenentwurf
4. Nachweis der Korrektheit
5. Aufwandsanalyse
6. Programmierung

Beispiel: Jüngster Studierender

1. Problem formulieren

Möglichkeiten:

1. Finde den jüngsten Studierenden.
2. Finde das Alter des jüngsten Studierenden.

Was ist gemeint? Eins von beiden? Beides?

Entscheidung: Möglichkeit 1 erscheint passend.

Beispiel: Jüngster Studierender

2. Problemanalyse, Problemabstraktion, Problemspezifikation

Problemanalyse:

Fragen:
Gibt es eine Lösung? Gibt es genau eine Lösung?

Antwort:
Für die Möglichkeit 2: jeweils „ja“

Für die Möglichkeit 1:
Es kann mehr als einen Studierenden geben, der die Anforderung erfüllt.

Beispiel: Jüngster Studierender

2. Problemanalyse, Problemabstraktion, Problemspezifikation

Problemabstraktion:

Was ist Alter?	Ganze positive Zahl
Was ist „jüngster“?	Definiert auf der Ordnungsrelation von Zahlen
Was heißt „finde“?	Die Altersangaben müssen verfügbar sein. (Bei der Möglichkeit 1 müssten Personenkennung und Alter verfügbar sein)

Entscheidungskorrektur: Möglichkeit 2, weil einfacher.

Beispiel: Jüngster Studierender

2. Problemanalyse, Problemabstraktion, Problemspezifikation

Problemspezifikation:

Problem: Minimum einer Menge von Zahlen

Gegeben: eine Folge $a_0, \ldots, a_{n-1}$ von positiven ganzen Zahlen, $n > 0$.

Gesucht: der kleinste Wert a der gegebenen Zahlen, d.h. $a = \min(a_0, \ldots, a_{n-1})$.

Beispiel: Jüngster Studierender

3. Algorithmenentwurf

wird später am verwandten Beispiel gezeigt.

Beispiel: Jüngster Studierender

4. Nachweis der Korrektheit

Fragen: Terminiert der Algorithmus?
Liefert er stets das richtige Ergebnis?

Beispiel: Jüngster Studierender

5. Aufwandsanalyse

wird hier nicht behandelt.

6. Programmierung

wird beim Übergang von „Informatik“ zu „Programmieren“ behandelt.

Algorithmenentwurf
Beispiel:
Minimum einer Menge von Zahlen

Minimum einer Menge von Zahlen

Gegeben: $a_0, a_1, \ldots, a_{n-1}$

Gesucht: Kleinster Wert der Eingabemenge

Algorithmus *Minimum*$(a_0, a_1, \ldots, a_{n-1})$

Durchlaufe die Elemente der Menge und merke den bisher kleinsten Wert.

Pseudocode:

Setze *merker* auf a_0;
Setze i auf 1;
Solange $i < n$ ist, fuehre aus:
    Wenn $a_i <$ *merker*, dann
        Setze *merker* auf a_i ;
    Erhoehe i um 1;
Gib *merker* zurueck;

Kurzschreibweise:

merker := a_0;
$i := 1$;
Solange $i < n$ ist, fuehre aus:
    {Wenn $a_i <$ *merker*, dann
        merker := a_i ;
    $i := i + 1$;}
Gib *merker* zurueck;

Minimum einer Menge von Zahlen

Folge als Detail-Beispiel:

11, 7, 8, 3, 15, 13, 9, 19, 18, 10, 4

$n = 11$

merker := a_0;
i := 1;
Solange $i < n$ ist, fuehre aus:
{Wenn a_i < *merker*, dann
merker := a_i ;
i := i + 1;}
Gib *merker* zurueck;

i	a_i	*merker*
		11
1	7	7
2	8	7
3	3	3
4	15	3
5	13	3
6	9	3
7	19	3
8	18	3
9	10	3
10	4	3
11		

Formuliere, analog zum letzten Beispiel, den Gauß-Algorithmus.

3. Grundkonzepte von Algorithmen

Kurzschreibweise:

merker := a_0; ← Wertzuweisungen an Variable

$i := 1;$

Solange $i < n$ ist, fuehre aus: ← Schleife

Block →

{

Wenn a_i < *merker*, dann ← Bedingte Anweisung

merker := a_i;

$i := i+1;$

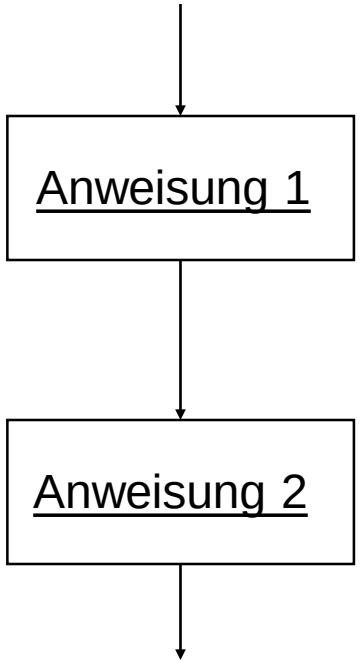
}

Gib *merker* zurueck; ← Rückgabeanweisung

Algorithmus:

Folge von Anweisungen: Anweisung 1; Anweisung 2; ...

Ablaufdiagramm:



Variable:

- hat einen Namen, über den sie angesprochen wird
- speichert Information

  Bsp: *merker*, a_0 , i, n

- Verwendung der gespeicherten Information durch Verwendung des Variablennamens: Anstelle des Variablennamens wird die gespeicherte Information eingesetzt

  Bsp: $i < n$

  $a_i < merker$

  $merker := a_i$;

  $i := 1$;

Wertzuweisung:

- Anweisung, mit der einer Variablen ein Wert gegeben wird
- Schreibweise: Variablenname := Ausdruck

Bsp.: $merker := a_0$;
$i := 1;$
$merker := a_i$;
$i := i + 1;$

- Ausführung einer Wertzuweisung:

Bsp.:			
	$i := 1;$	i	1
	$i := i + 1;$	i	2

Bedingte Anweisung:

Anweisung, mit der man Alternativen in einem Algorithmus formulieren kann

- Schreibweise: Wenn Bedingung, dann Anweisung 1;
  Sonst Anweisung 2;

Bsp.: Wenn $a_i < merker$, dann $merker := a_i$;

Ausführung einer bedingten Anweisung:

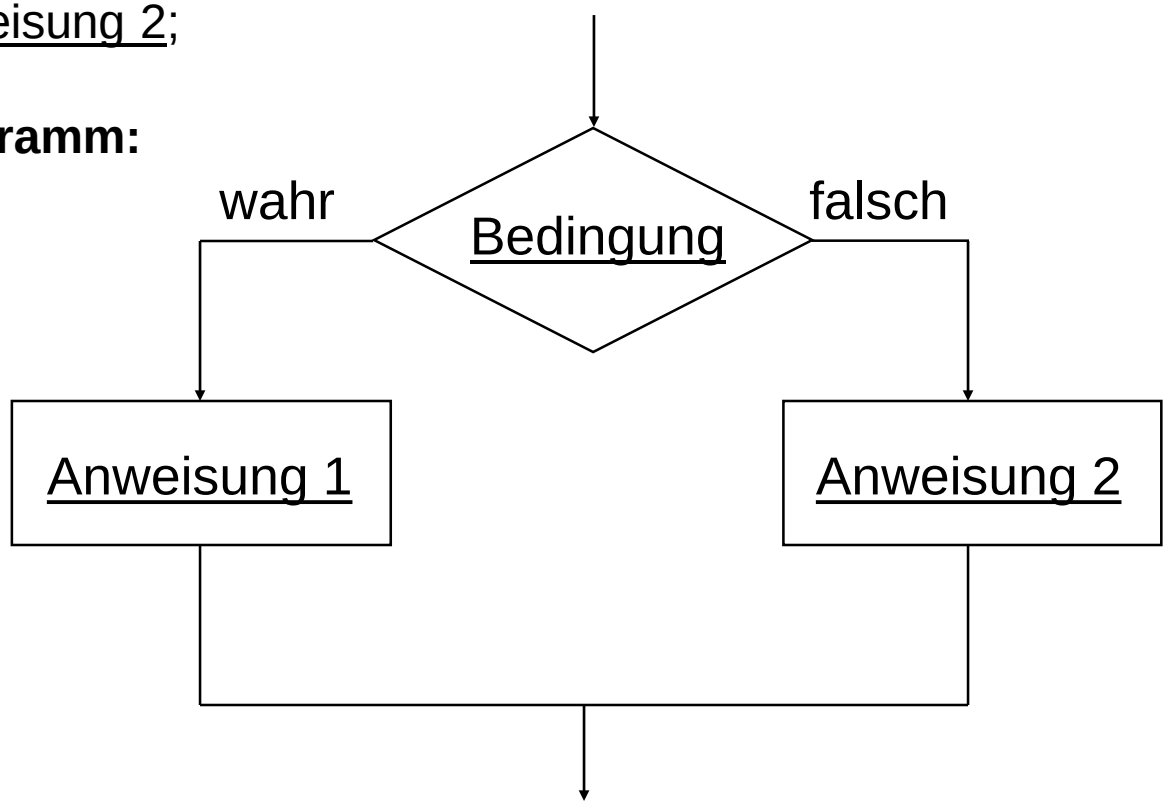
1. Werte die Bedingung aus.
2. Wenn die Bedingung erfüllt ist, dann führe Anweisung 1 aus.
3. Wenn die Bedingung nicht erfüllt ist, dann führe Anweisung 2 aus.

Wenn der Sonst-Teil fehlt, wird so verfahren, wie wenn er da wäre, aber Anweisung 2 nichts tut („leere Anweisung“).

Bedingte Anweisung:

Wenn Bedingung, dann Anweisung 1;
Sonst Anweisung 2;

Ablaufdiagramm:



Schleife:

- Anweisung, mit der man eine andere Anweisung mehrmals wiederholen kann.
- Schreibweise: Solange Bedingung, führe aus: Anweisung

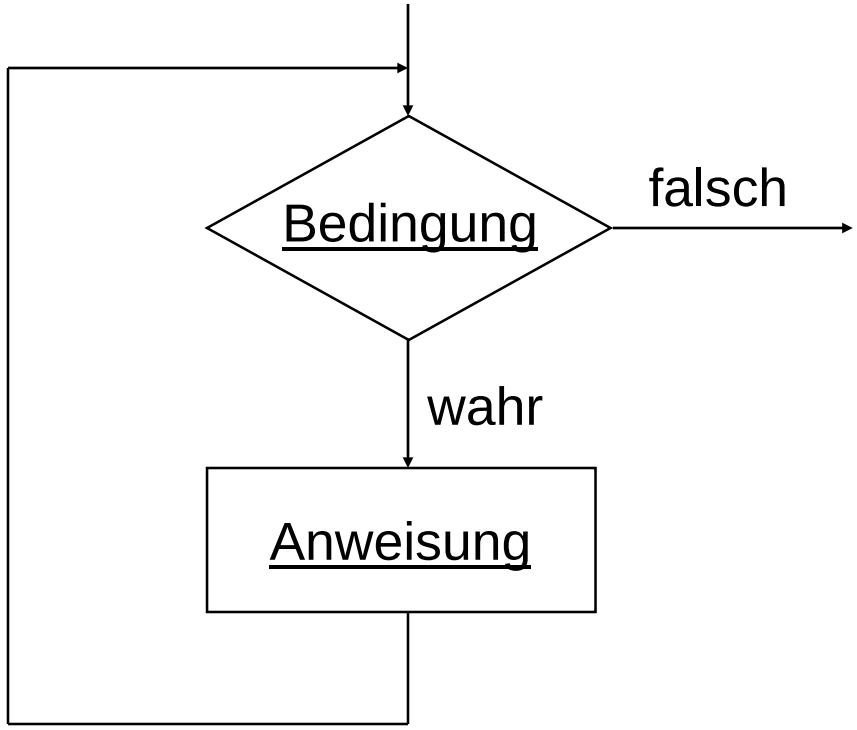
Bsp.: Solange $i < n$ ist, fuehre aus:
{Wenn $a_i < merker$, dann $merker := a_i$;
$i := i + 1$;}

Ausführung einer Schleife:

1. Werte die Bedingung aus.
2. Wenn die Bedingung erfüllt ist, dann führe Anweisung aus und fahre mit 1. fort.
3. Wenn die Bedingung nicht erfüllt ist, dann beende die Schleife.

Schleife:

Solange Bedingung, fuehre aus: Anweisung



Block:

- Zusammenfassung einer Folge von Anweisungen zu einer einzigen Anweisung
- Schreibweise: {Anweisung 1; .... Anweisung n;}

Bsp.: {Wenn $a_i < merker$, dann $merker := a_i$;
$i := i + 1;$}

Ausführung eines Blocks:

Führe die Anweisungen des Blocks nacheinander in der angegebenen Reihenfolge aus.

Rückgabeanweisung:

- gibt einen Wert an die aufrufende Instanz zurück
  (mehr dazu später)

- Schreibweise: Gib Ausdruck zurück;

Bsp.: Gib *merker* zurueck;

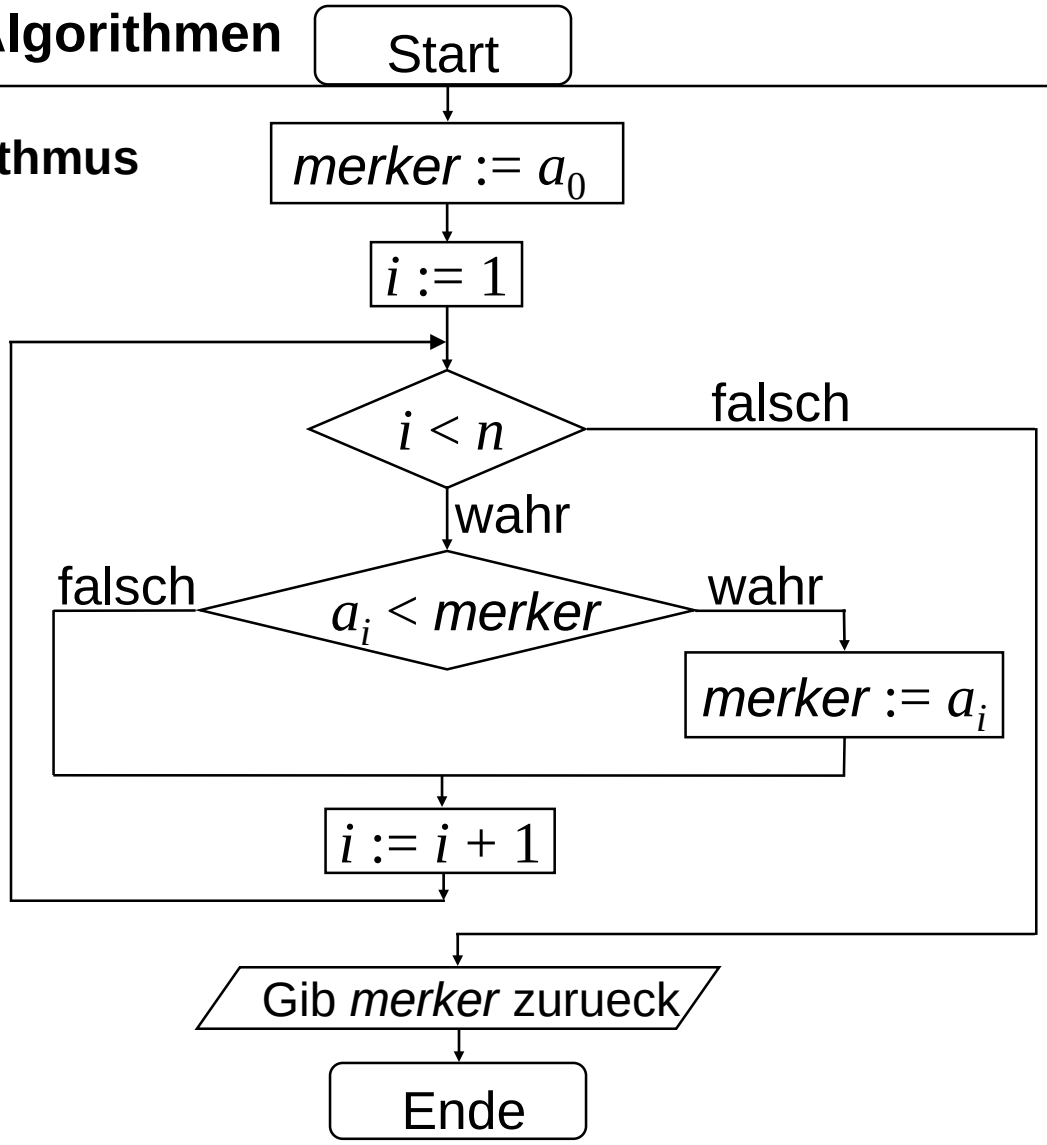
Druckanweisung:

Analog zur Rückgabeanweisung, nur dass das Ausgabeziel ein Ausgabegerät wie Bildschirm oder Drucker ist.

Formuliere den behandelten Algorithmus als Ablaufdiagramm.

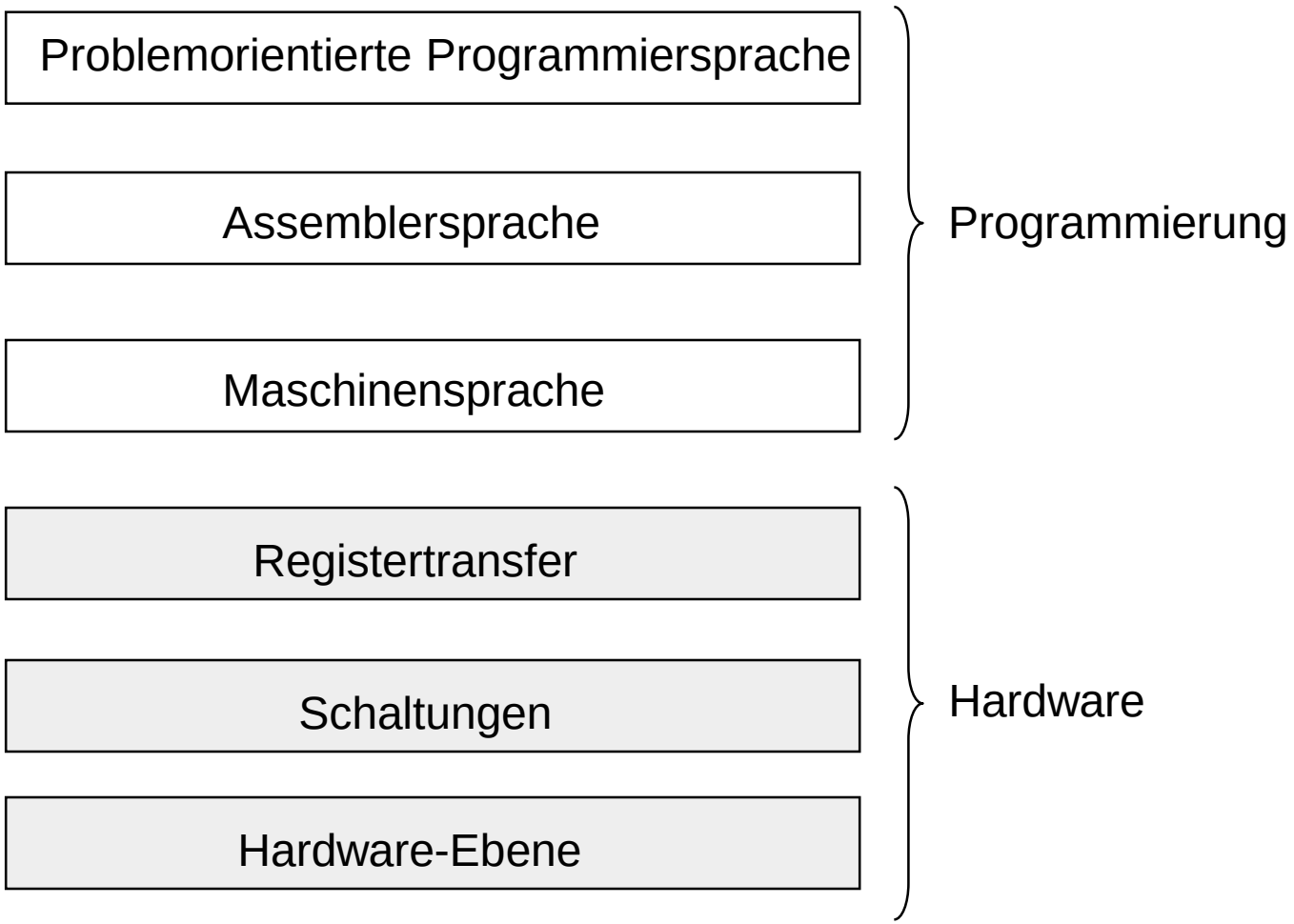
Darstellung des Algorithmus als Ablaufdiagramm:

merker := a_0;
i := 1;
Solange $i < n$ ist, fuehre aus:
{Wenn a_i < *merker*, dann
merker := a_i ;
i := i+1;}
Gib *merker* zurueck;



4. Rechnerarchitektur und Maschinensprache

Ebenen heutiger Rechner



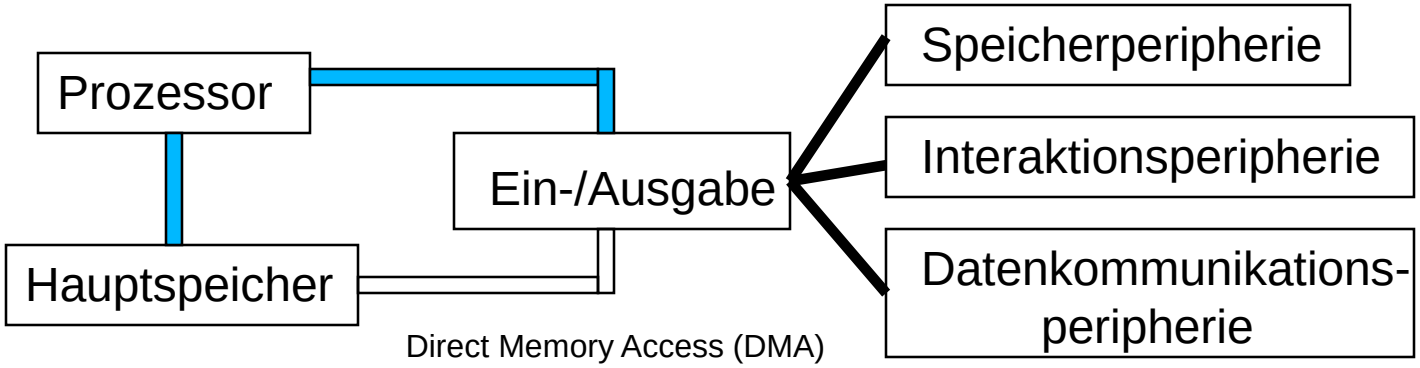
Grundstruktur heutiger Rechner

- **von-Neumann-Rechnerarchitektur**
  - Rechner = Prozessor + Speicher + Ein-/Ausgabe
  - Programm und Daten im Speicher
  - Abarbeiten des Programms durch den Prozessor

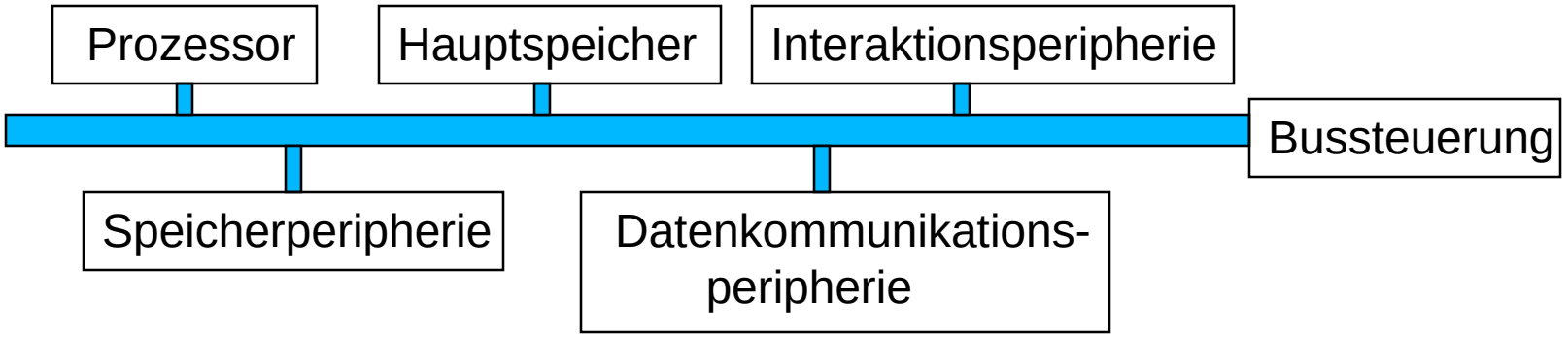
von-Neumann-Architektur

Möglichkeiten der Integration der Funktionseinheiten:

- *Einzelverbindungswege*



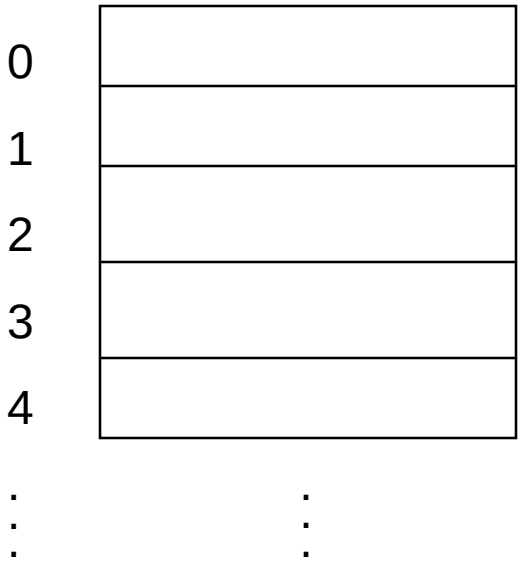
- *Bus* (*Sammelschiene*)



von-Neumann-Architektur

(Haupt-)Speicher:

- setzt sich aus Speicherzellen zusammen, die unter Adressen ansprechbar sind:

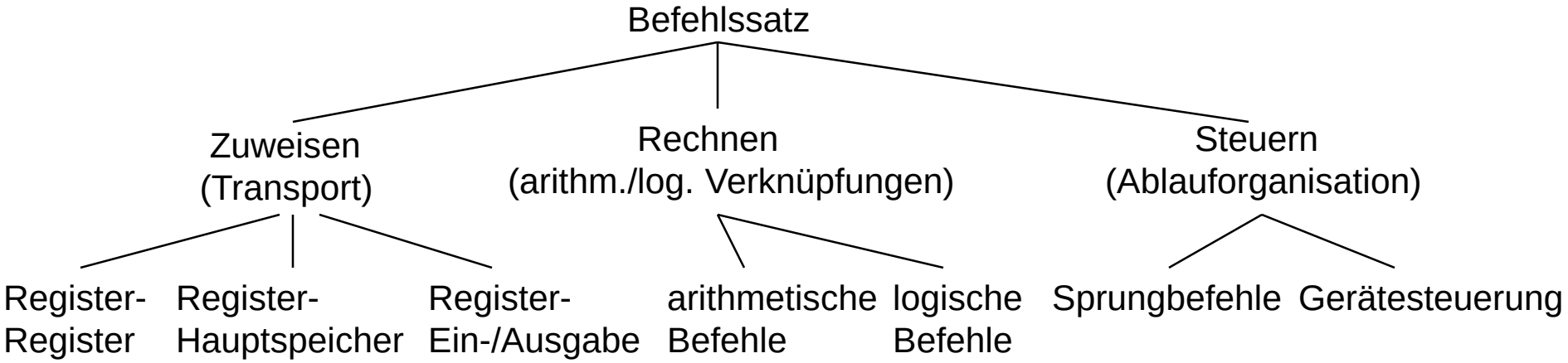


von-Neumann-Architektur

Prozessor:

- führt Maschinenbefehle aus. Die Befehle sind im *Befehlssatz* festgelegt.

Einteilung eines typischen Befehlssatzes:



Aufbau eines typischen Maschinenbefehls: Operationscode Operanden

Bsp.: addiere r_4, r_3, r_2; hole r_5, a_1; multipliziere r_6, r_5, r_4;

von-Neumann-Architektur

Prozessor:

- Funktionsgruppen eines Prozessors:

Leitwerk: steuert die Abarbeitung des Programms

Rechenwerk: führt die arithmetischen und logischen Befehle sowie eventuelle Dienstleistungen (z.B. Adressrechnungen) für das Leitwerk aus

Register: Speicherzellen im Prozessor für Operanden und Zwischenergebnisse - sind schneller als der Hauptspeicher zugreifbar

Bitte die Tafel beachten.

von-Neumann-Architektur

Prozessor:

- *Befehlsausführungszyklus:*

Solange kein Halte-Befehl aufgetreten ist, führe aus:

Hole den nächsten Befehl vom Hauptspeicher in das Befehlsregister;

erhöhe den Befehlszeiger;

führe den Befehl aus, der im Befehlsregister steht;

Weiter auf der Tafel mit einem Beispiel.

5. Schaltungen

Zweiwertige Informationsdarstellung: Bits, Dualsystem und mehr

Zweiwertige Informationsdarstellung

Bit:

- Kleinste Informationseinheit
- Zwei mögliche Werte: 0 oder 1
- Realisierung: Transistor: auf/zu

Byte:

- Informationseinheit aus 8 Bits:

  Bsp: 00100001

  Anwendung: z. B. Codierung von Zeichen (Textzeichen und Steuerzeichen)

Verschiedenes über Zahlensysteme und ihre Relevanz für den Computer (an der Tafel).

Wie lassen sich negative Zahlen bei binärer Darstellung realisieren? Welche Probleme gibt es bei der Verwendung eines Vorzeichenbits? Was sind und was sollen Einer- und Zweierkomplement?

Bitte bearbeitet die Fragen in (wenigstens zwei) Gruppen und stellt Eure Ergebnisse den anderen Kursteilnehmern vor.

Zweiwertige Informationsdarstellung

Positive ganze Zahlen (Überblick/Zusammenfassung):

- Darstellung als *Dualzahl (bzw. Binärzahl)*:

$$z = z_n\, 2^n + z_{n-1}\, 2^{n-1} + z_{n-2}\, 2^{n-2} + \ldots z_0 2^0$$

Bsp.: dezimal 7 ist dual 111 $\quad 1\,2^2 + 1\,2^1 + 1\,2^0$

dezimal 14 ist dual 1110 $\quad 1\,2^3 + 1\,2^2 + 1\,2^1 + 0\,2^0$

- Dualzahlen sind ein Spezialfall der *p-adischen Darstellung* ($p = 2$):

$$z = z_n\, p^n + z_{n-1}\, p^{n-1} + z_{n-2}\, p^{n-2} + \ldots z_0\, p^0$$

Bsp.: Hexadezimalsystem: $p = 16$, „Ziffern:“ 0-9, A-F

dezimal 15 ist hexadezimal F
dezimal 16 ist hexadezimal 10
dezimal 27 ist hexadezimal $1B$

Positive ganze Zahlen (Überblick/Zusammenfassung):

Berechnung der p-adischen Darstellung einer Dezimalzahl d:

$$z_i = (d \; div \; p^i) \; mod \; p.$$

div: ganzzahlige Division ohne Rest.
mod (modulo): Berechnung des Restes.

Bsp.:
Dezimal 14 in Dualdarstellung:

$z_0 = (14 \text{ div } 1) \quad \text{mod } 2 = 14 \quad \text{mod } 2 = 0$
$z_1 = (14 \text{ div } 2) \quad \text{mod } 2 = 7 \quad \text{mod } 2 = 1$
$z_2 = (14 \text{ div } 4) \quad \text{mod } 2 = 3 \quad \text{mod } 2 = 1$
$z_3 = (14 \text{ div } 8) \quad \text{mod } 2 = 1 \quad \text{mod } 2 = 1$

$14 = 1\,2^3 + 1\,2^2 + 1\,2^1 + 0\,2^0$

Zweiwertige Informationsverarbeitung: Boolesche Funktionen

Berechnung Boolescher Funktionen:

Wir betrachten sog. n-stellige Boolesche Funktionen:

Definitionsbereich: Menge der $0/1$-Folgen der Länge n

Wertebereich: $\{0,1\}$

Definition Boolescher Funktionen durch Wertetabellen:

Bsp.:

a	b	c	$f(a, b, c)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

f ist genau dann gleich 1, wenn die Mehrzahl der Parameterwerte gleich 1 ist.

Berechnung Boolescher Funktionen:

Boolesche Formeln:

Darstellung einer Booleschen Funktion durch Verknüpfung von elementaren Booleschen Funktionen.

Beispiele für elementare Boolesche Funktionen: *and, or, nand, nor, not*

a	*b*	*and*
0	0	0
0	1	0
1	0	0
1	1	1

a	*b*	*or*
0	0	0
0	1	1
1	0	1
1	1	1

a	*b*	*nand*
0	0	1
0	1	1
1	0	1
1	1	0

a	*b*	*nor*
0	0	1
0	1	0
1	0	0
1	1	0

a	*not*
0	1
1	0

nand = nicht *and*, *nor* = nicht *or*, 1 = „*true*“, 0 = „*false*“

Berechnung Boolescher Funktionen:

Hier ein Verfahren zur Herleitung einer Booleschen Formel, der sog. **disjunktiven Normalform**, aus *and-, or-* und *not*-Verknüpfungen:

Bsp.:

a	b	c	$f(a,b,c)$	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\overline{a} * b * c$
1	0	0	0	
1	0	1	1	$a * \overline{b} * c$
1	1	0	1	$a * b * \overline{c}$
1	1	1	1	$a * b * c$

$*$ entspricht *and* (math. $\wedge$)

$+$ entspricht *or* (math. $\vee$)

$\overline{}$ entspricht *not* (math. $\neg$)

Ergebnis: $f = \overline{a} * b * c + a * \overline{b} * c + a * b * \overline{c} + a * b * c,$

bzw. $(\neg\, a \wedge b \wedge c) \vee (a \wedge \neg\, b \wedge c) \vee (a \wedge b \wedge \neg\, c) \vee (a \wedge b \wedge c).$

Zweiwertige Informationsverarbeitung

Vorgehensweise zur Berechnung Boolescher Funktionen:

Verfahren zur Herleitung einer Booleschen Formel, der sog. **disjunktiven Normalform**, aus *and*-, *or*- und *not*-Verknüpfungen:

1. a) Für alle Zeilen der Tabelle, die den Wert 1 liefern, forme eine Boolesche Formel, die alle Eingabeparameter direkt oder negiert enthält,
   b) ein Parameter wird genau dann negiert, wenn sein Wert in der Zeile gleich 0 ist,
   c) die Parameter werden mit *and* verknüpft.

2. Verknüpfe die aus 1. entstehenden Formeln mit *or*.

Optimierungsmöglichkeit:

Resultierende Formeln können häufig verkürzt werden (weniger Operationen).

Berechne die disjunktive Normalform für:

- *or*
- *and*
- *nor*

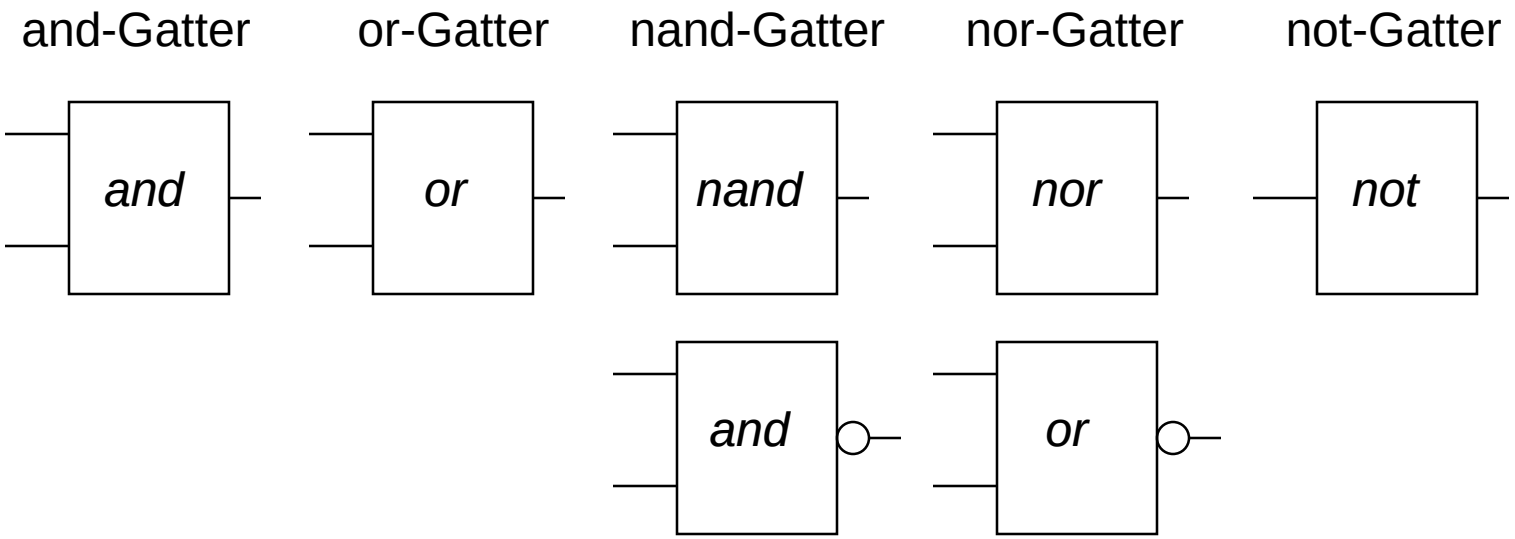
Vergleiche die Ergebnisse mit den Wertetabellen für die ursprünglichen obigen Verknüpfungen.

Zweiwertige Informationsverarbeitung: Schaltungen

Boolescher Schaltkreis:

Darstellung einer Booleschen Funktion durch Verknüpfung von Gattern.

Gatter entsprechen den elementaren Booleschen Funktionen.
Beispiele für elementare Boolesche Funktionen: *and, or, nand, nor, not*



Wir betrachten „reale“ Schaltungen (Schaltkreise) an der Tafel.

Finde solche Schaltungen, die jeweils die Verknüpfungen
and, or und *not* repräsentieren.

Zweiwertige Informationsverarbeitung

Zusammenfassung am Beispiel:

1. Zweiwertige Informationsdarstellung: Bit, Byte, positive ganze Zahl

2. Zweiwertige Informationsverarbeitung: Boolesche Funktionen

a	b	S
0	0	0
0	1	1
1	0	1
1	1	0

$\overline{a} * b$

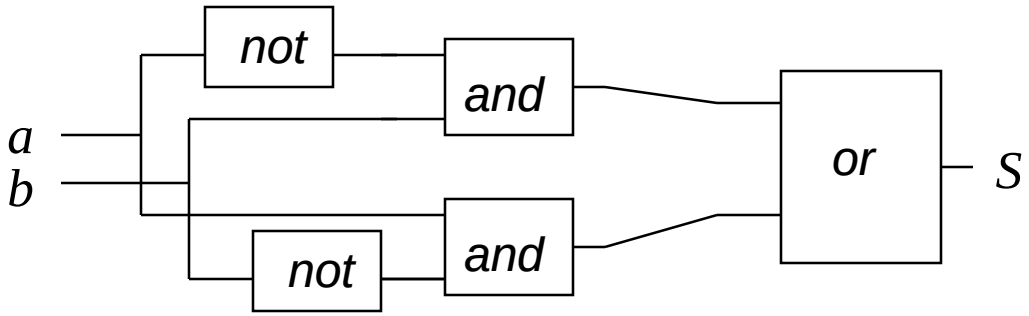
$a * \overline{b}$

$$S = \overline{a} * b + a * \overline{b}$$

Finde eine Schaltung mit den zuvor angegebenen Gattern, welche die gefundene Boolsche Funktion der letzten Seite darstellt.

3. Zweiwertige Informationsverarbeitung: Schaltungen

$S = \overline{a} * b + a * \overline{b}$, also:

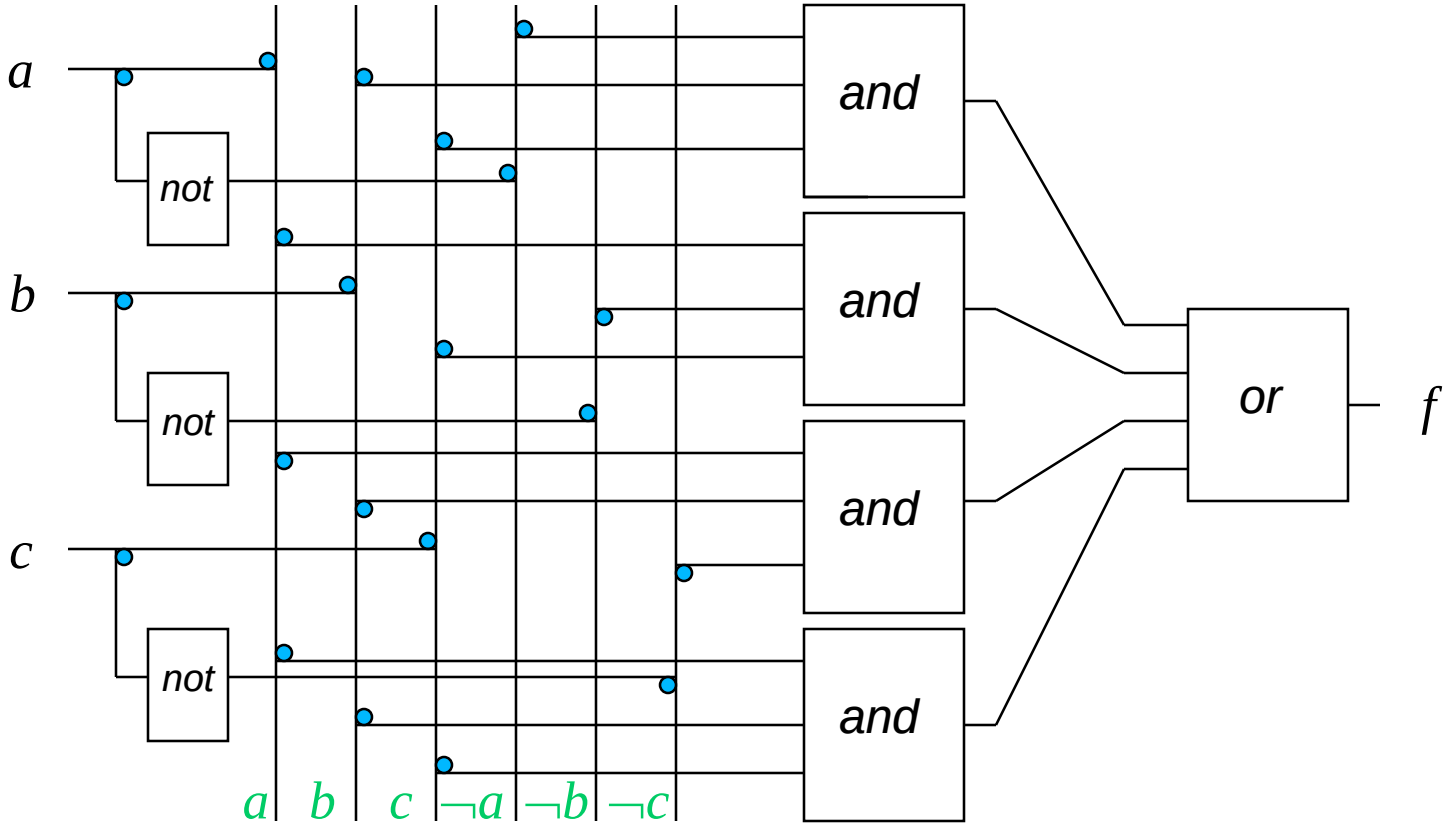


Einschub (zur Entspannung):

Rechnerintern ist es sinnvoll, dass Dinge sortiert werden (können). Aus diesem Grund sehen wir uns die Verfahren „Selection Sort“, „Insertion Sort“ und „Quick Sort“ an.

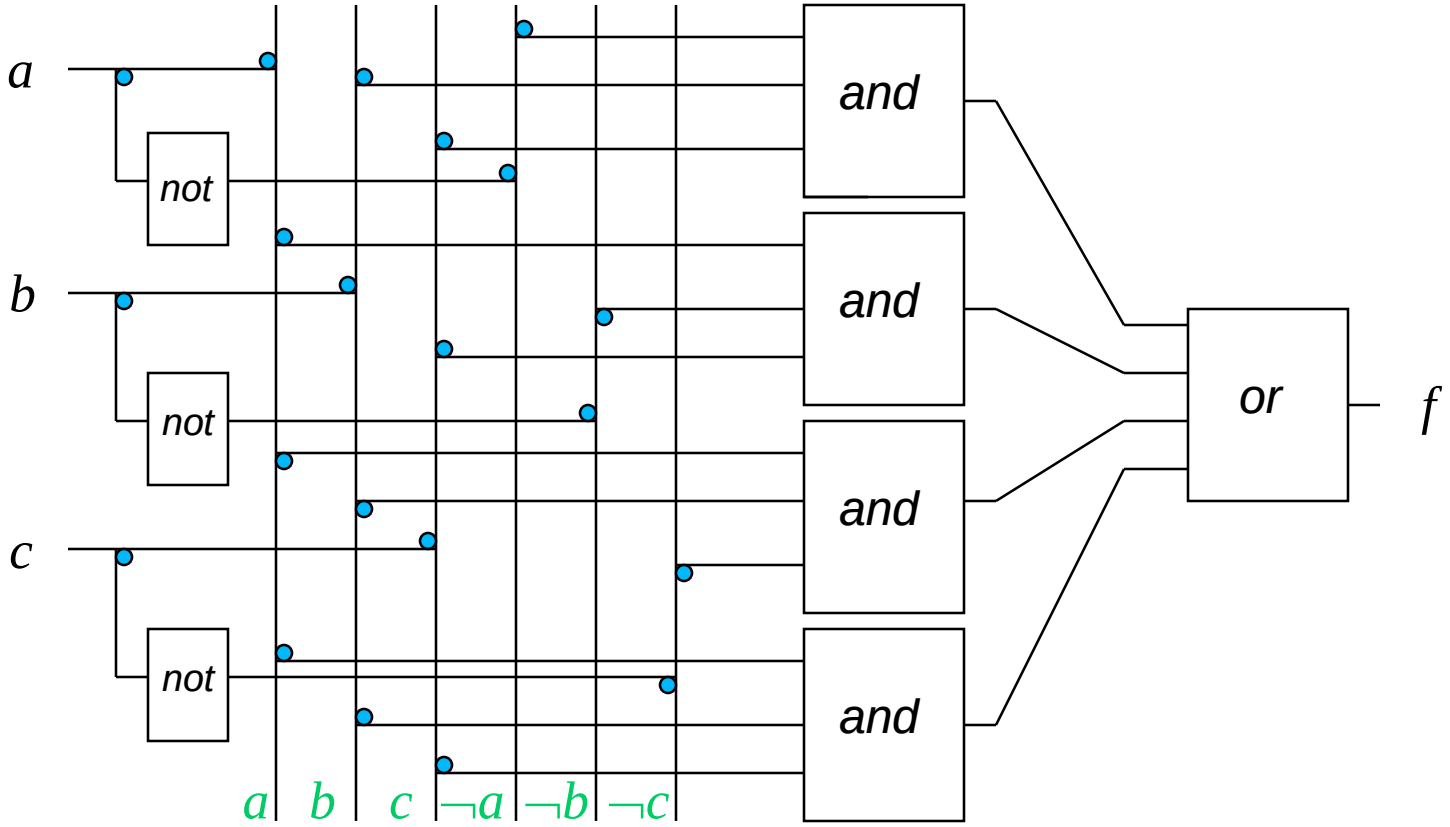
Zweiwertige Informationsverarbeitung

Frage: Was passiert in der Schaltung unten?
Finde die entsprechende Boolsche Funktion. (An den blauen Punkten sind die Leitungen verbunden.)



Zweiwertige Informationsverarbeitung

Antwort: $f = \bar{a} * b * c + a * \bar{b} * c + a * b * \bar{c} + a * b * c$



Beispiel: 1-Bit-Addierer

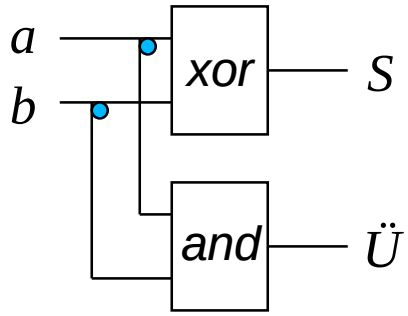
Halbaddierer:

a	b	S	$\ddot{U}$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Beispiel: 1-Bit-Addierer

Halbaddierer:

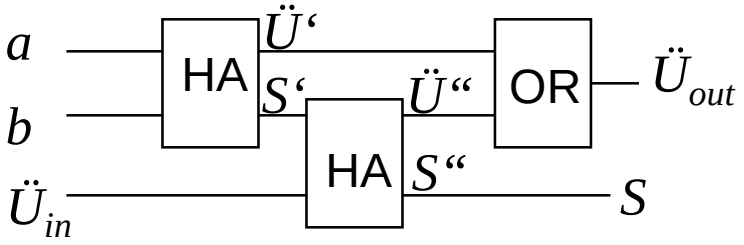
a	b	S	$\ddot{U}$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Beispiel: 1-Bit-Addierer

Addierer:

a	b	$\ddot{U}_{in}$	S	$\ddot{U}_{out}$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

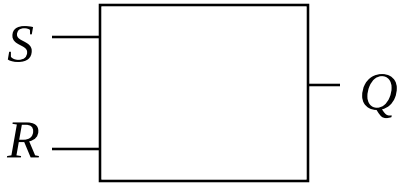


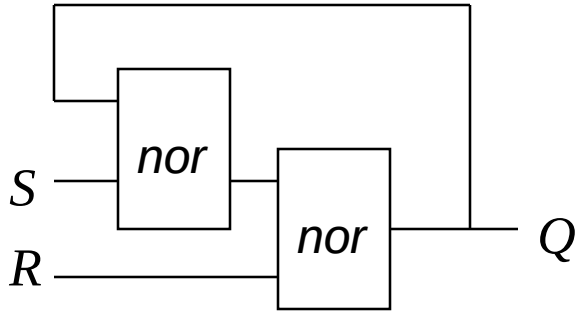
HA: Halbaddierer

Speicher:

Bsp.: RS-Flipflop: R = „reset“, S = „set“, speichert 1-Bit-Information

Lösung: Realisierung durch rückgekoppelte Schaltung:

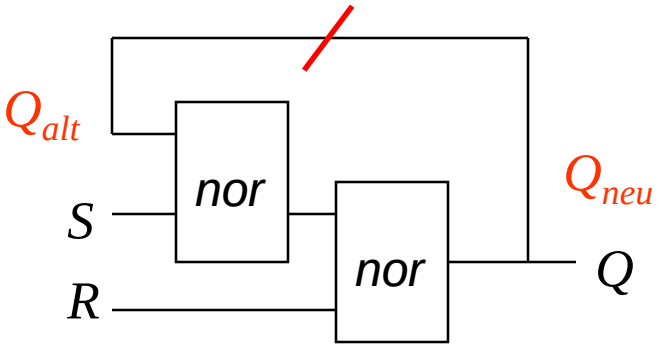




Gespeicherter Wert: Q
Setzen des Speichers auf 1: $S = 1, R = 0$
Setzen des Speichers auf 0: $S = 0, R = 1$
Ausgabe des gespeicherten Werts: $S = 0, R = 0$, d.h. $Q = 0$ oder $Q = 1$
nicht verwendet: $S = 1, R = 1$

Problem: Zwei Werte ($Q = 0/1$) für die gleiche Eingabe ($S = 0, R = 0$)

Speicher:

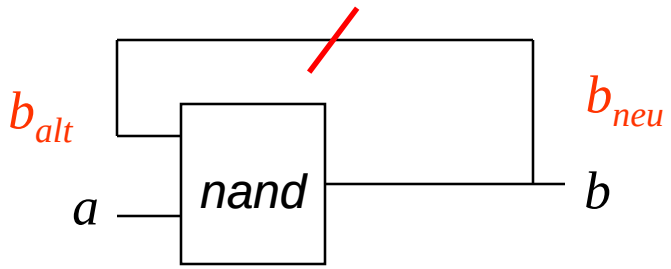


RS-Flipflop: Analyse des Verhaltens

Zustand	R	S	Q_{alt}	Q_{neu}	
0	0	0	0	0	stabil, gibt den gespeicherten Wert aus
1	0	0	1	1	stabil, gibt den gespeicherten Wert aus
2	0	1	0	1	instabil, speichert den Wert 1 („set“), $\rightarrow 3$
3	0	1	1	1	stabil, wird auf 1 gesetzt („set“)
4	1	0	0	0	stabil, wird auf 0 gesetzt („reset“)
5	1	0	1	0	instabil, wird auf 0 gesetzt („reset“), $\rightarrow 4$
6	1	1	0	0	stabil, für Speicherfunktion nicht verwendet
7	1	1	1	0	instabil, $\rightarrow 6$

Stabiler Zustand: $Q_{alt} = Q_{neu}$

Beispiel für einen schwingenden Schaltkreis:



Zustand	a	b_{alt}	b_{neu}	
0	0	0	1	instabil, $\rightarrow 1$
1	0	1	1	stabil, $\rightarrow 1$, $b = 1$
2	1	0	1	instabil, $\rightarrow 3$
3	1	1	0	instabil, $\rightarrow 2$

Ergebnis:

Eingabe $a = 0$:
Ausgabe $b = 1$

Eingabe $a = 1$:
Ausgabe b schwingt zwischen 0 und 1

Es folgt eine Einführung in die Grundlagen neuronaler Netze an der Tafel.