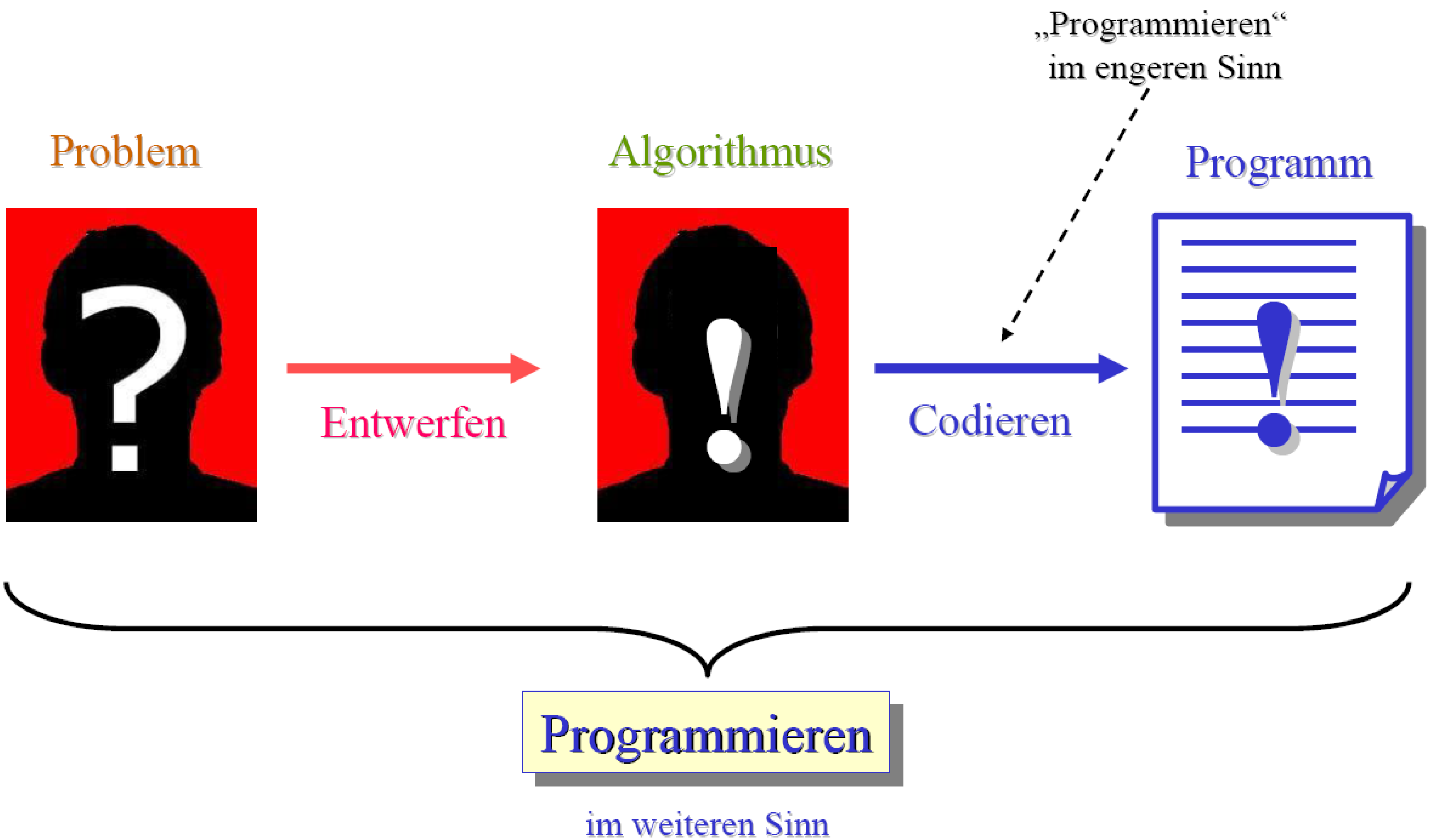
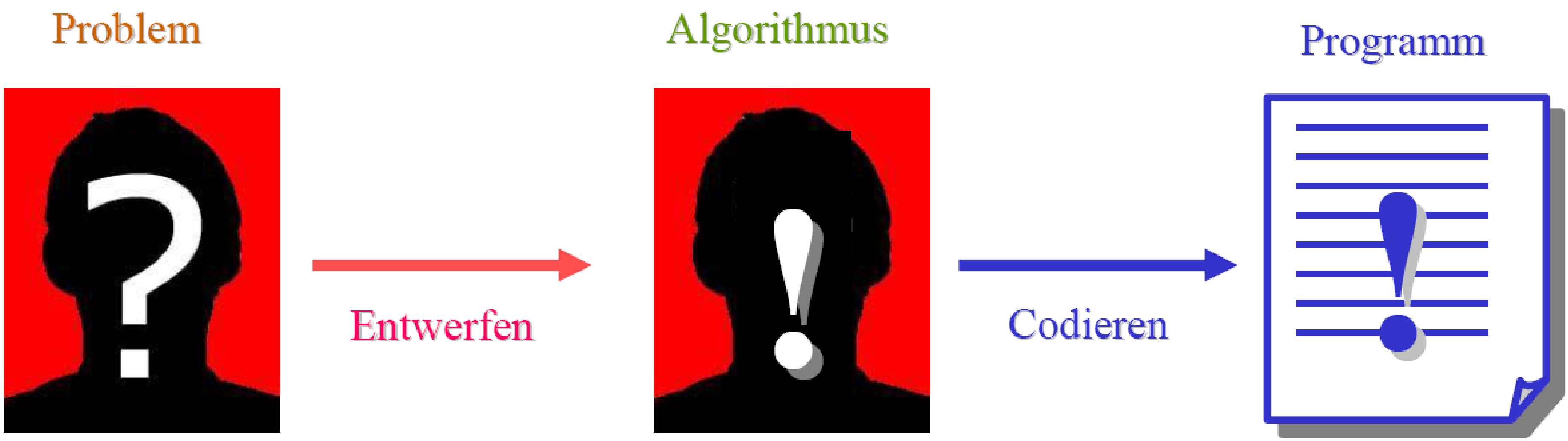
Algorithmen und algorithmische Sprachkonzepte

Programme und Algorithmen

Was ist Programmieren?



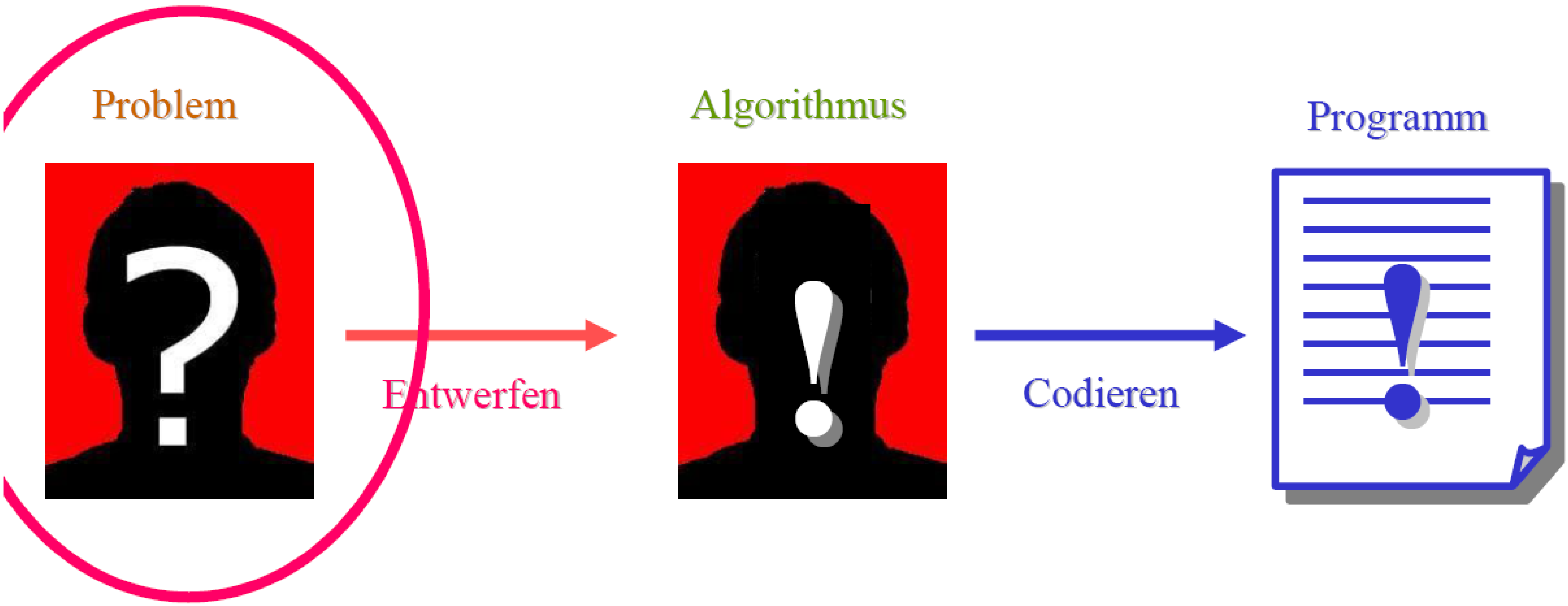
Vom Problem zum Programm



- kreativer Akt
- „Kunst“
- schwer zu vermitteln
- schwer zu lernen

- zum Teil eher „mechanisch“
- „Handwerk“
- leichter zu vermitteln
- leichter zu erlernen

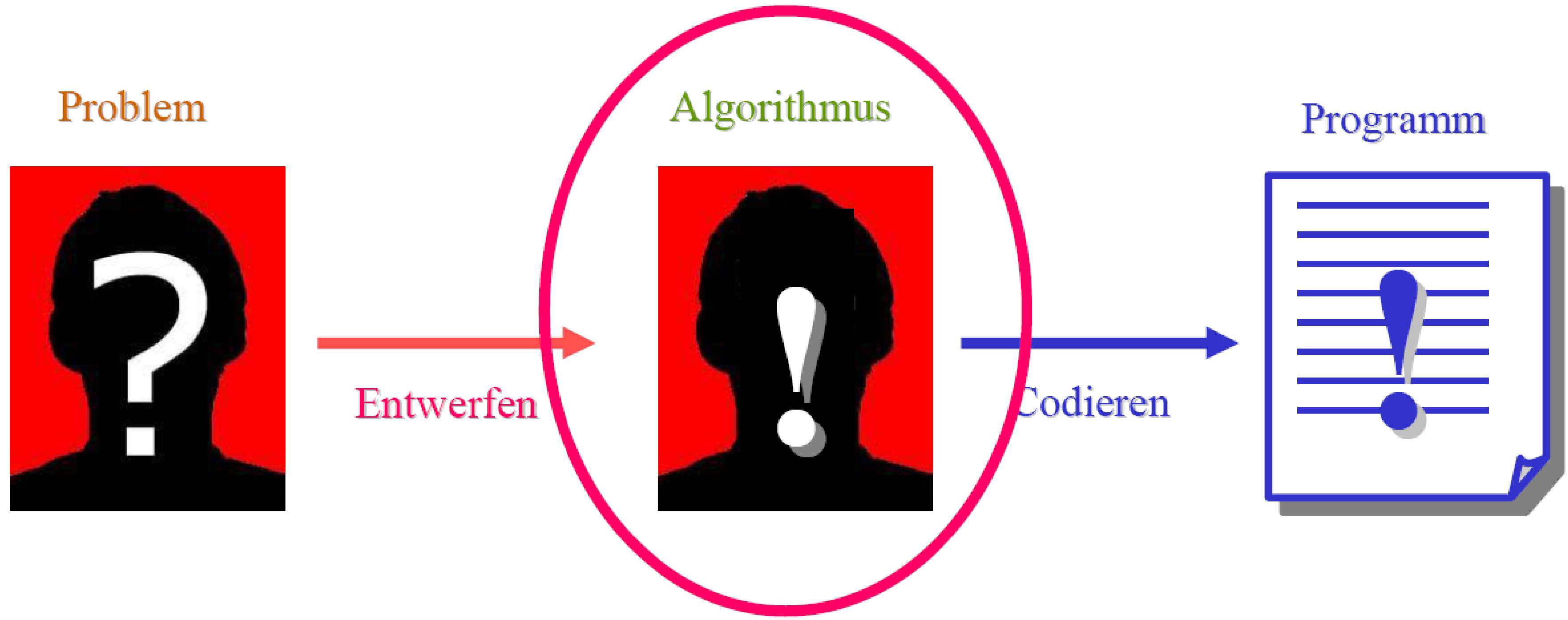
„Problem“



„Problem“:

- mit Rechnerhilfe zu lösende Aufgabenstellung
- abstrakt spezifiziertes Ziel, das erreicht werden soll
- Voraussetzungen (Kontext) sind ebenfalls spezifiziert

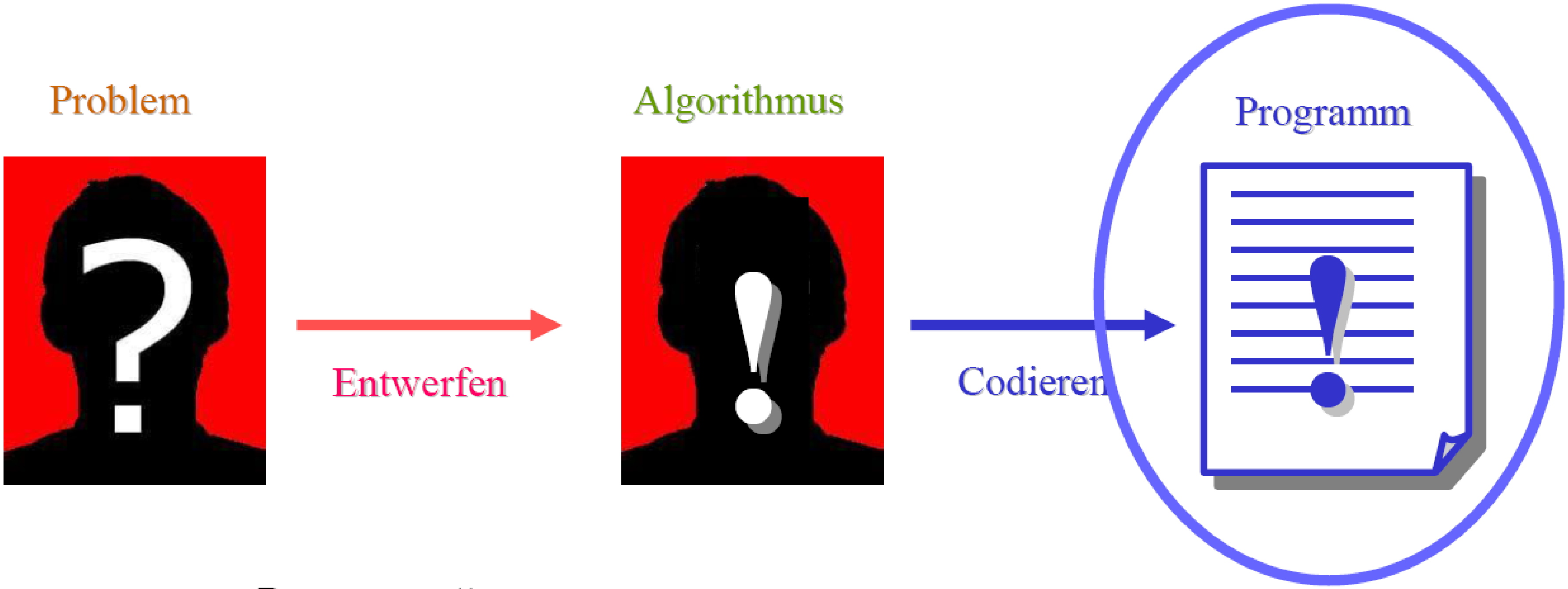
„Algorithmus“



„Algorithmus“:

- präzise beschriebener Ablaufplan eines Prozesses zur Lösung des Problems
- effektiv ausführbare Einzelschritte, möglichst effizient
- informell repräsentiert: Diagramm, natürliche Sprache, Idee

„Programm“

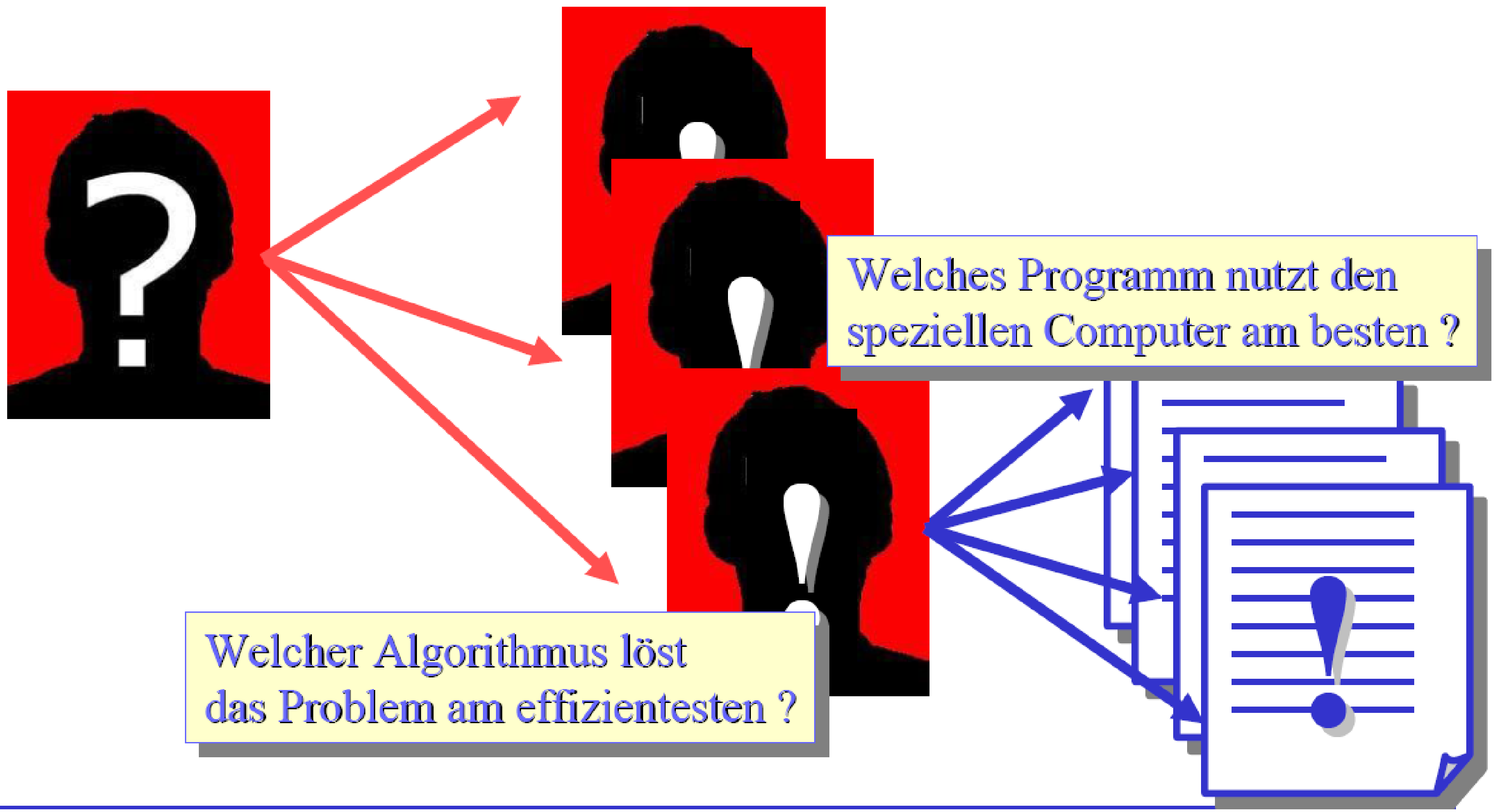


„Programm“:

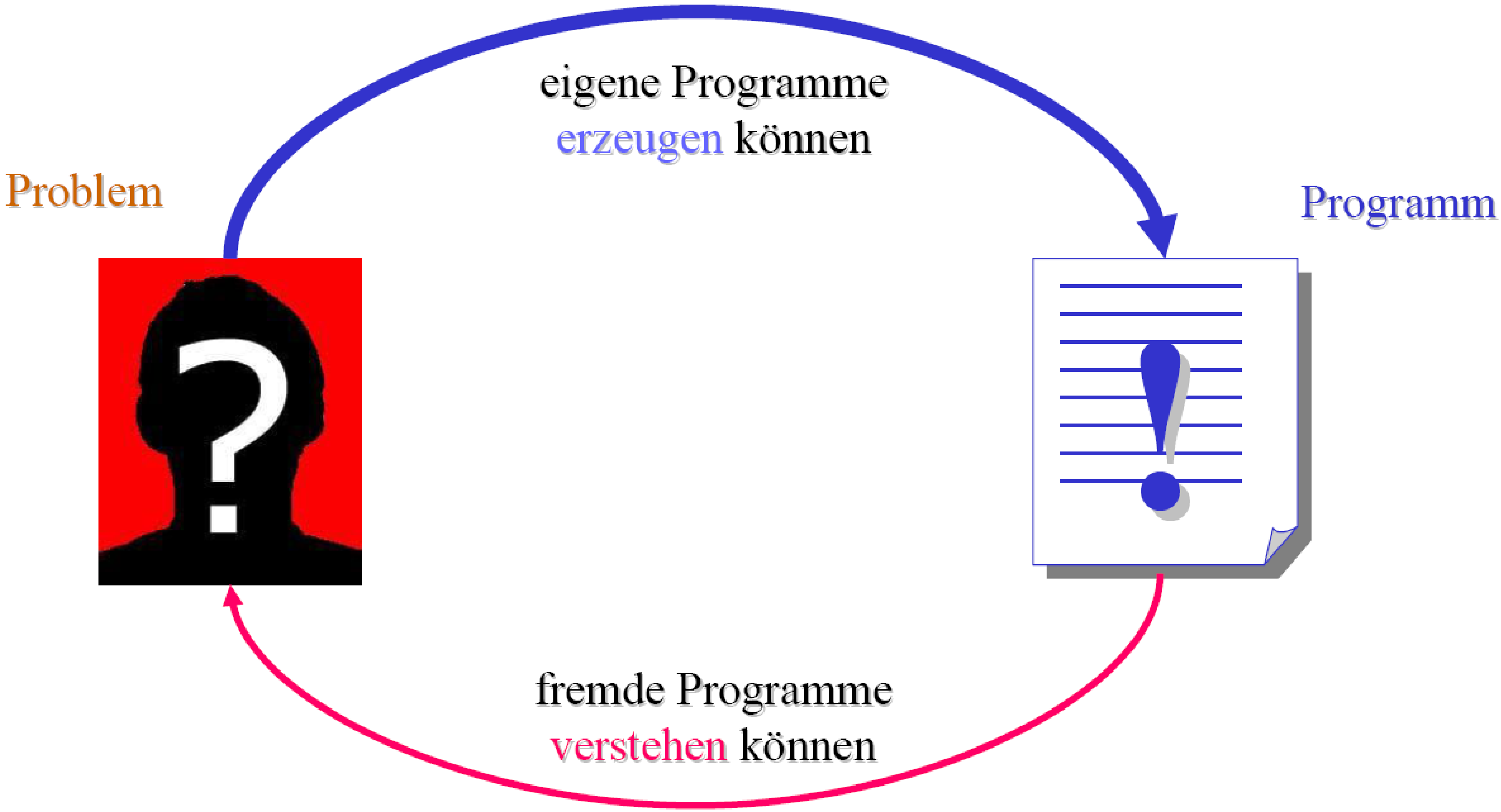
- textuelle Repräsentation eines Algorithmus
- präzisiert alle Details des Algorithmus
- in einer formalen Sprache abgefasst
- so detailliert, dass ein Computer den Algorithmus ausführen kann

Problem – Algorithmus – Programm: Lösungsalternativen

- Jedes Problem lässt sich durch verschiedene Algorithmen lösen.
- Jeder Algorithmus lässt sich durch verschiedene Programme darstellen.



Lernziele dieser Vorlesung: Imperatives Programmieren

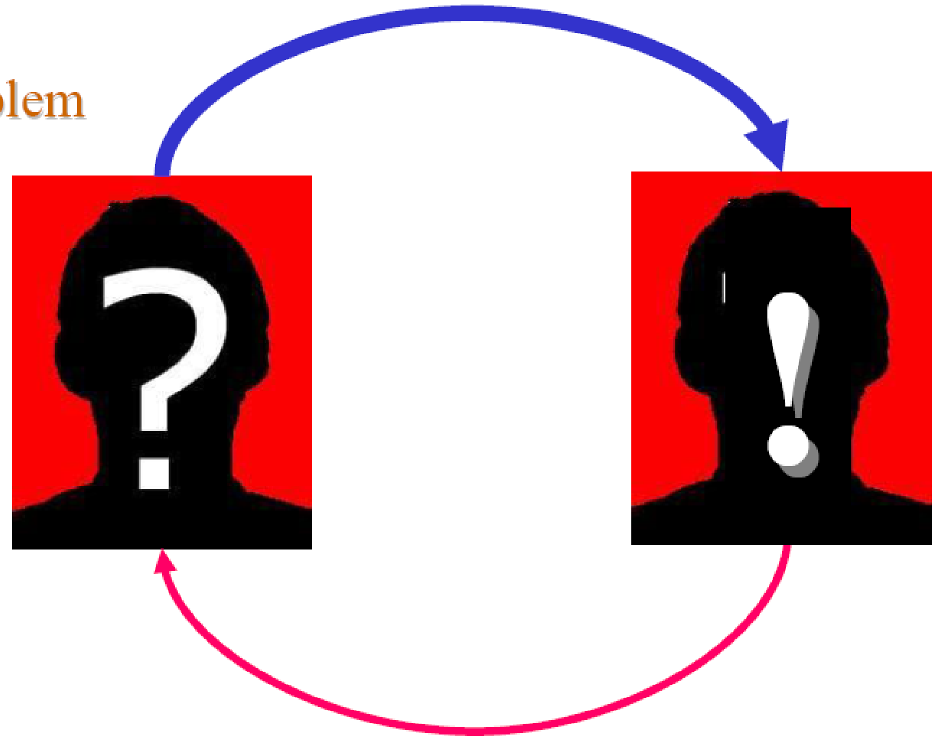


Lernziele dieser Vorlesung: Algorithmisches Denken

eigene Algorithmen
entwickeln können

Problem

Algorithmus



fremde Algorithmen
verstehen und analysieren können

Herkunft des Wortes „Algorithmus“

- Der Begriff „Algorithmus“ leitet sich aus dem (verballhornten und latinisierten) Eigennamen des Autors eines der ersten „algorithmischen Bücher“ ab

  - *Über das Rechnen mit indischen Ziffern*
    - ⇨ *Al-Kitāb al-Dschamʿ wa-l-tafrīq bi-ḥisāb al-Hind*
    - ⇨ *Verfasst um 825*
    - ⇨ *Von* **Muhammad ibn Musa, Abu Dscha'far al-Chwarizmi**
      - – محمد بن موسى ابو جعفر الخوارزمي
      - – Geboren um 780 in Choresmien
        - (heute Xiva in Usbekistan)
      - – Gestorben um 835 (oder 850)
      - – Mathematiker, Astronom und Geograph, der den größten Teil seines Lebens in Baghdad verbrachte und dort im „Haus der Weisheit“ tätig war
  - Buch erschien vierhundert Jahre später ins Lateinische übersetzt unter dem Titel
    - ⇨ *„Liber Algorithmi de numero Indorum“*

Herkunft des Wortes „Algorithmus“

Bemerkung: Ein anderes Buch von Muhammad ibn Musa Abu Dscha'far al-Chwarizmi hat auch einen weiteren grundlegenden Begriff der Mathematik geliefert:

Algebra

Al-Kitāb al-muḫtaṣar fī hīsāb al-ğabr wa’l-muqābala (arabisch الكتاب المختصر في حساب الجبر و المقابلة „Ein kurzgefasstes Buch über die Rechenverfahren durch Ergänzen und Ausgleichen)

- Beinhaltet Verfahren zum Lösen quadratischer Gleichungen
  - *al-ğabr* („Vervollständigen“, „Wiederherstellen“, „Ganzmachen“) – Beseitigung der negativen Ausdrücke
  - *al-muqabalah* („Ausgleich“) – Zusammenfassung der Ausdrücke gleicher Potenz je Seite.

Begriff des Algorithmus

- Genauere Begriffserläuterung
  - Ein **Algorithmus** (algorithm) ist die Beschreibung eines Verfahrens, um aus gewissen Eingabegrößen bestimmte Ausgabegrößen zu berechnen. Dabei muss folgendes gegeben sein:
    - ⇨ **Spezifikation von Ein- und Ausgabegrößen**
    - ⇨ **Durchführbarkeit**
    - ⇨ **Korrektheit**

Spezifikation

- **Eingabespezifikation**: Es muss genau spezifiziert sein, welche Eingabegrößen erforderlich sind und welchen Anforderungen diese Größen genügen müssen, damit das Verfahren funktioniert
- **Ausgabespezifikation**: Es muss genau spezifiziert sein, welche Ausgabegrößen (Resultate) mit welchen Eigenschaften berechnet werden

Durchführbarkeit

- **Endliche Beschreibung**: das Verfahren muss in einem endlichen Text vollständig beschrieben sein
- **Effektivität**: Jeder Schritt des Verfahrens muss effektiv (d.h. tatsächlich) „mechanisch“ ausführbar sein
  - ⇨ **Bem.: „**Effektivität“ ist nicht zu verwechseln mit „Effizienz“ („Wirtschaftlichkeit“)
- **Determiniertheit**: Der Verfahrensablauf ist zu jedem Zeitpunkt fest vorgeschrieben

- **Korrektheit**
  - **partielle Korrektheit**: Jedes berechnete Ergebnis genügt der Ausgabespezifikation, sofern die Eingaben der Eingabespezifikation genügt haben
  - **Terminierung**: Der Algorithmus hält nach endlich vielen Schritten mit einem Ergebnis an, sofern die Eingaben der Eingabespezifikation genügt haben

Begriff des Algorithmus

- **Bemerkung:** Nach unserer Begriffsbestimmung gäbe es also keine
  - nicht-deterministische,
  - nicht-terminierende
  - ...

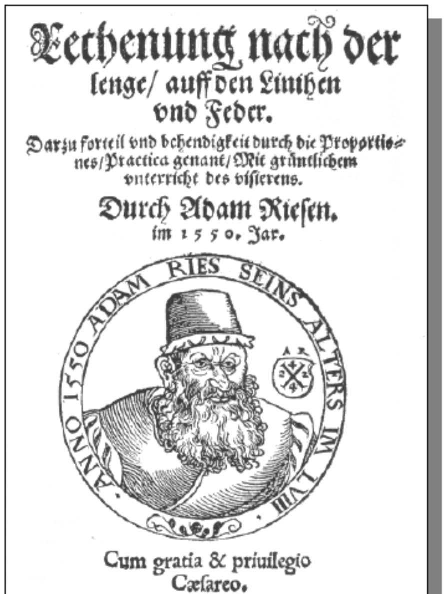
    ⇨ Algorithmen
- Diese Begriffe werden aber durchaus verwendet!
  - Methode erfüllt alle Anforderungen an einen Algorithmus, bis auf die mit „nicht“ gekennzeichneten
- Ebenso wird oftmals von „Algorithmen“ gesprochen, auch wenn die Ein- und Ausgabespezifikationen fehlen

Beschreibung von Algorithmen

- Für die Beschreibung von Algorithmen gibt es viele Möglichkeiten
  - Alltagssprache
  - Konkrete Programmiersprache
  - Dazwischen gibt es eine Vielzahl von Notationen, die den Übergang zwischen Problembeschreibung und Programm erleichtern sollen
    - Flussdiagramme
    - Pseudocode

Beschreibung von Algorithmen

- Beispiel von Algorithmenbeschreibung in Alltagssprache
  - Adams Riese Algorithmus zum duplieren einer Zahl



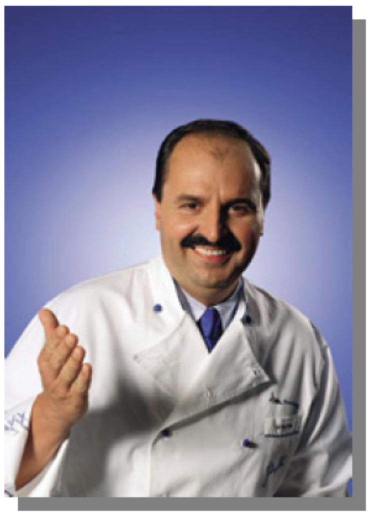
Dupliren

Lehret wie du ein zahl zweyfaltigen solt.
Thu ihm also: Schreib die zahl vor dich,
mach ein Linien darunter,
heb an zu forderst,
duplir die erste Figur.
Kompt ein zahl die du mit einer Figur schreiben magst,
so setz die unden.
Wo mit zweyen, schreib die erste, die ander behalt im sinn.
Darnach duplir die ander
und gib darzu, das du behalten hast,
und schreib abermals die erste Figur, wo zwo vorhanden,
und duplir fort bis zur letzten,
die schreibe ganz aus, als folgende Exempel ausweisen.

41232	98765	68704
82464	197530	137408

Beschreibung von Algorithmen

- Weiteres Beispiel einer Algorithmenbeschreibung in Alltagssprache
  - Johann Lafers „Algorithmus“ zur Zubereitung eines Wiener Schnitzels



Die Kalbsschnitzel zwischen zwei Klarsichtfolien,
welche vorher leicht mit etwas Öl eingerieben wurden,
dünn ausklopfen. Mit Salz und Pfeffer würzen.

Die Eier und die geschlagene Sahne mit Hilfe einer Gabel
in einer Schüssel verquirlen. Salzen und pfeffern.

Die Schnitzel in Mehl wenden,
überschüssiges Mehl abklopfen, dann durch das Ei ziehen
und anschließend in den Semmelbröseln panieren.

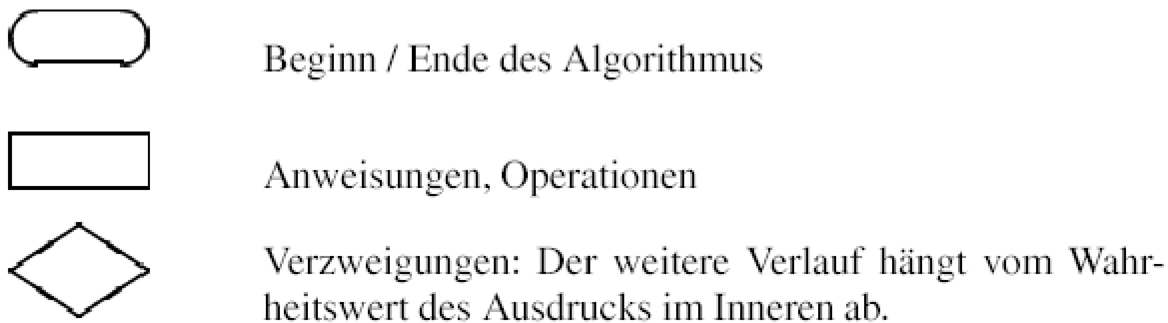
Butterschmalz in einer Pfanne erhitzen und
die Schnitzel von beiden Seiten goldgelb ausbacken.
Auf Küchenpapier abtropfen lassen und auf den Tellern mit
je einer Zitronenspalte anrichten.

Beschreibung von Algorithmen: Steuerungsverlauf

- Die Anordnung der Anweisungen eines Algorithmus, die bestimmt, in welcher Reihenfolge Dinge geschehen, heißt
  - **Steuerungsverlauf** (control flow) des Algorithmus
    - ⇨ Wird auch **Kontrollfluss** (flow of control) genannt
  - Manchmal wird auch der Programmablauf oder **Kontrollfaden** (thread of control), also die tatsächlich abgespulten Schritte und Anweisungen so bezeichnet
- Der Steuerungsverlauf lässt sich durch Flussdiagramme und Pseudocode genauer fassen als mit Alltagssprache

Beschreibung von Algorithmen: Flussdiagramme

- Der Steuerungsverlauf kann mit der Notation der **Flussdiagramme** (flow chart) graphisch dargestellt werden
  - Die Sprache der Flussdiagramme benutzt folgende Symbole

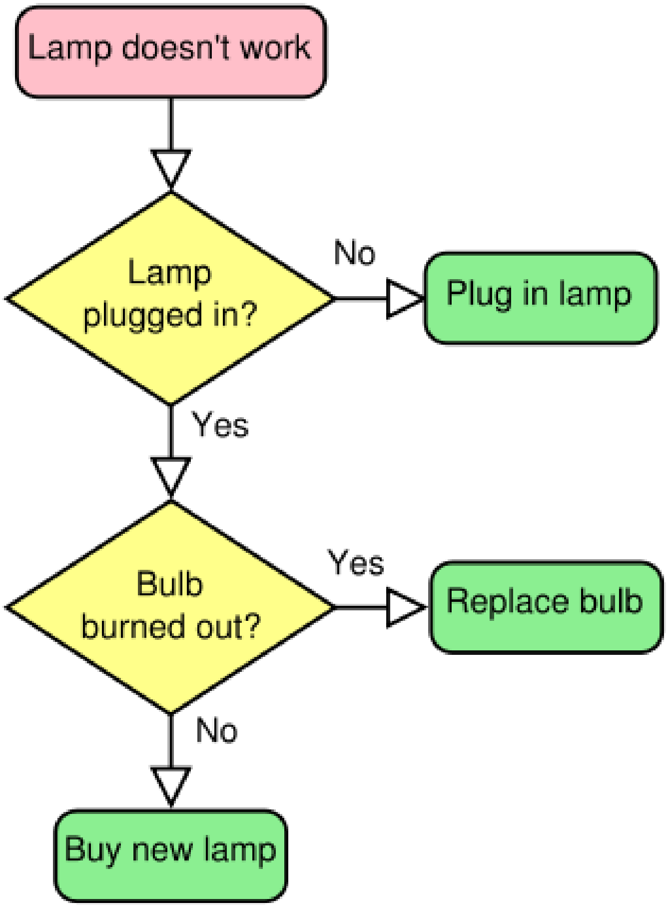


⇨ Werden mit Pfeilen verbunden

⇨ Die Ausführung solcher Ablaufpläne folgt den Pfeilen zwischen den Kästchen

Beschreibung von Algorithmen

- Beispiel: Was ist zu tun, wenn eine Lampe nicht brennt



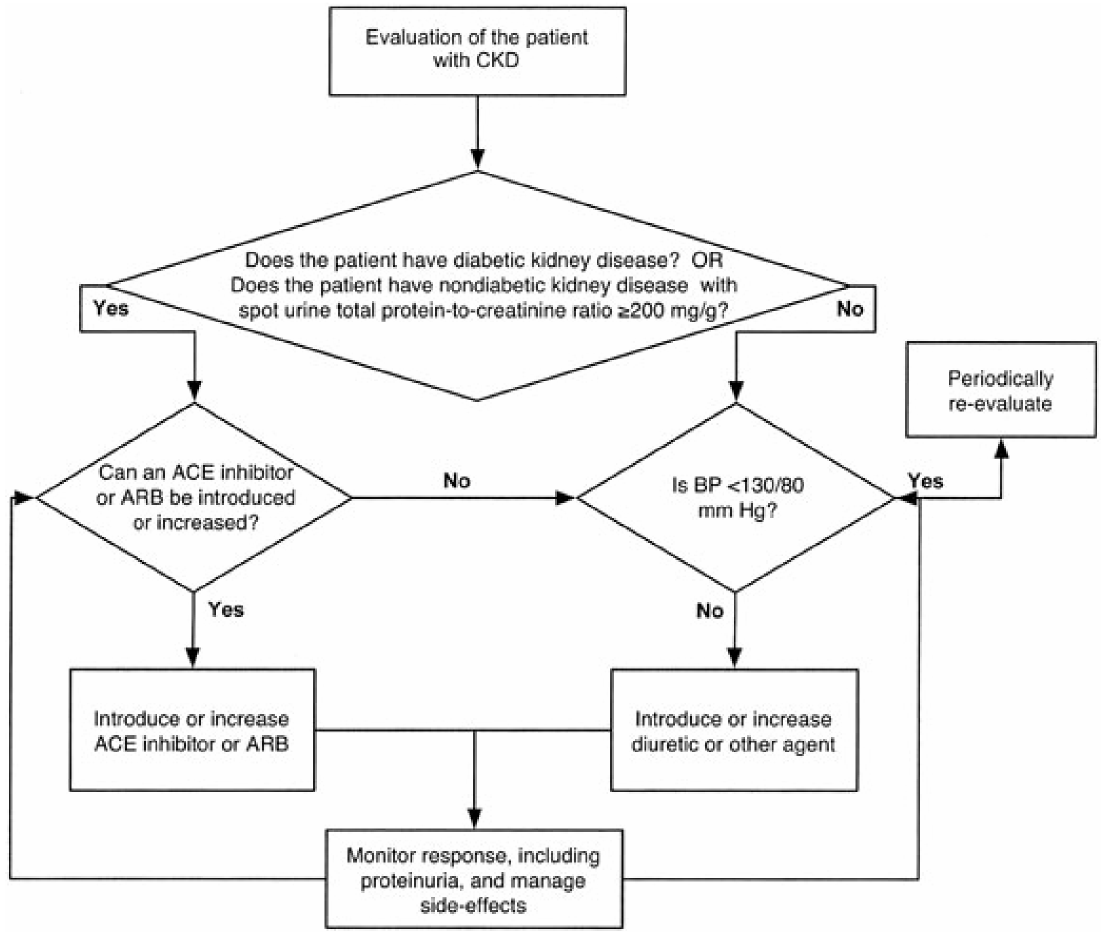

```
LampDoesNotWork()
if Lamp_is_plugged_in = false then
        goto Plug_in_lamp
if Bulb_is_burned_out = true then
        goto Replace_bulb
print "Buy new lamp"
return

Replace_bulb:
        print "Replace bulb"
        return
Plug_in_lamp:
        print "Buy new lamp"
        return
```

From wikipedia.org

Beschreibung von Algorithmen

- Flussdiagramme finden auch zunehmend außerhalb der Informatik Verwendung

Beispiel aus einer „Clinical Guideline“



Prinzipien des Algorithmenentwurfs

- Neben den Bedingungen, die schon in die Begriffsdefinition eingegangen sind, gibt es weitere wichtige Prinzipien, die beim Entwurf zu beachten sind
  - Effizienz
    - ⇨ Der Algorithmus soll möglichst wenig **Aufwand** verursachen
      - – Das Ergebnis mit möglichst wenig Rechenschritten (oder mit möglichst wenig Speicherbedarf) erzielen
    - ⇨ Frage der Komplexität von Algorithmen wichtiges Thema in der „Algorithmen und Berechnungskomplexität“
  - Korrektheit beweisbar?
    - ⇨ Ein nicht-korrekter Algorithmus ist nach unserer Definition kein Algorithmus!
    - ⇨ Trotzdem sind nicht-korrekte Verfahren eher die Regel als die Ausnahme ☹

Grundschema des Algorithmenaufbaus

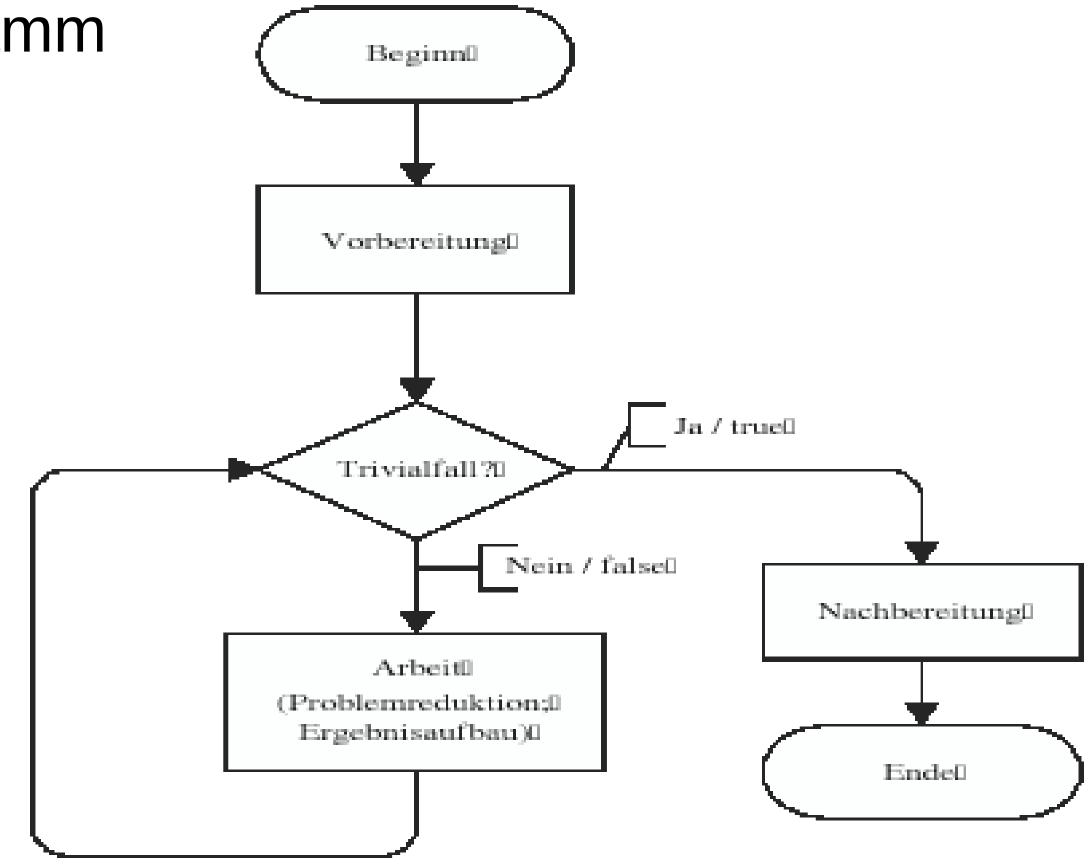
- Folgendes Grundschema wird uns bei vielen Algorithmen begegnen

Grundschema des Algorithmenaufbaus

	Name des Algorithmus und Liste der Parameter
	Spezifikation des Ein-/Ausgabeverhaltens
Schritt 1	**Vorbereitung:** Einführung von Hilfsgrößen etc.
Schritt 2	**Trivialfall?** Prüfe, ob ein einfacher Fall vorliegt. Falls ja, Beendigung mit Ergebnis.
Schritt 3	**Arbeit (Problemreduktion, Ergebnisaufbau):** Reduziere die Problemstellung X auf eine einfachere Form X', mit $X > X'$ bezüglich einer wohlfundierten Ordnung $>$. Baue entsprechend der Reduktion einen Teil des Ergebnisses auf.
Schritt 4	**Rekursion** bzw. **Iteration:** Rufe zur Weiterverarbeitung den Algorithmus mit dem reduzierten X' erneut auf (Rekursion), bzw. fahre mit X' anstelle X bei Schritt 2 fort (Iteration). $\square$

Flussdiagramme

- Grundschema des Algorithmenaufbaus als Flussdiagramm



Grundschema des Algorithmenaufbaus: Beispiel

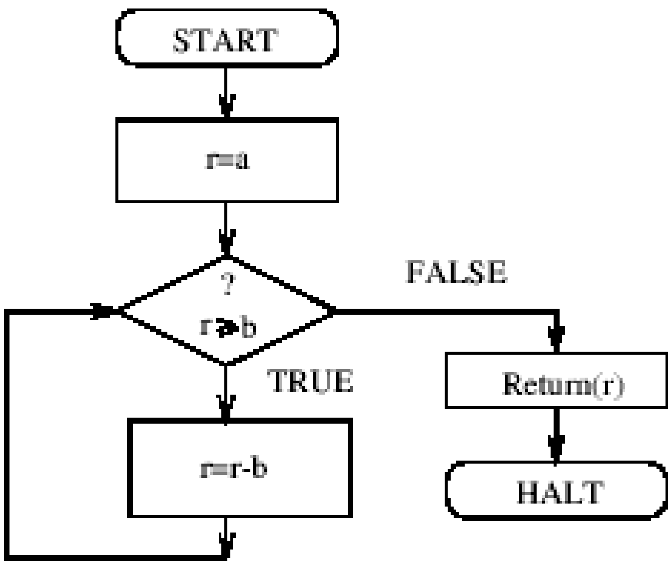
- **Beispiel:** Man finde ein Verfahren zur Berechnung des Rests *r* der Ganzzahldivision *a/b*, also für *cb+r=a* und *r<b*, wobei 0≤*a* und 0≤*b*
  - Diese Funktion wird meist „Modulus-Funktion“ genannt, da $r \equiv a \bmod b$

mod(a,b)
// Anforderungen:
// $a, b \in \mathbb{Z}, \ a \geq 0, b > 0.$
// Zusicherung:
// Das Resultat ist der Rest der Division a/b.

1. Kopiere a nach r.
2. Prüfe, ob r größer oder gleich b ist.
3. Falls nein, gib das Resultat r aus.
4. Falls ja, ziehe von r den Wert b ab (und speichere das Resultat in r).
5. Mache weiter mit Schritt 2.

Modulus-Funktion als Flussdiagramm

- **Beispiel:** Flussdiagramm für iterative Beschreibung der Modulus-Funktion



Wir verfolgen den Ablauf des Kontrollflusses für die Eingabe `mod(7, 3)`.

START

`a == 7, b == 3`
`r = 7`
`7` $\geq$ `3 ?` TRUE
`r = 7-3`
`4` $\geq$ `3 ?` TRUE
`r = 4-3`
`1` $\geq$ `3 ?` FALSE
Return(1)

HALT

Steuerungsverlauf

- Die Konstruktion „fahre fort mit Schritt 2“ stellt einen Sprung (jump) im Steuerungsverlauf dar
  - Dies ist die elementarste Form, eine Wiederholung oder sonstige Verzweigung im Ablauf auszudrücken
  - Dadurch erhalten wir die elementar-iterative Beschreibungsform von Algorithmen

Elementar-iterative Beschreibungsform

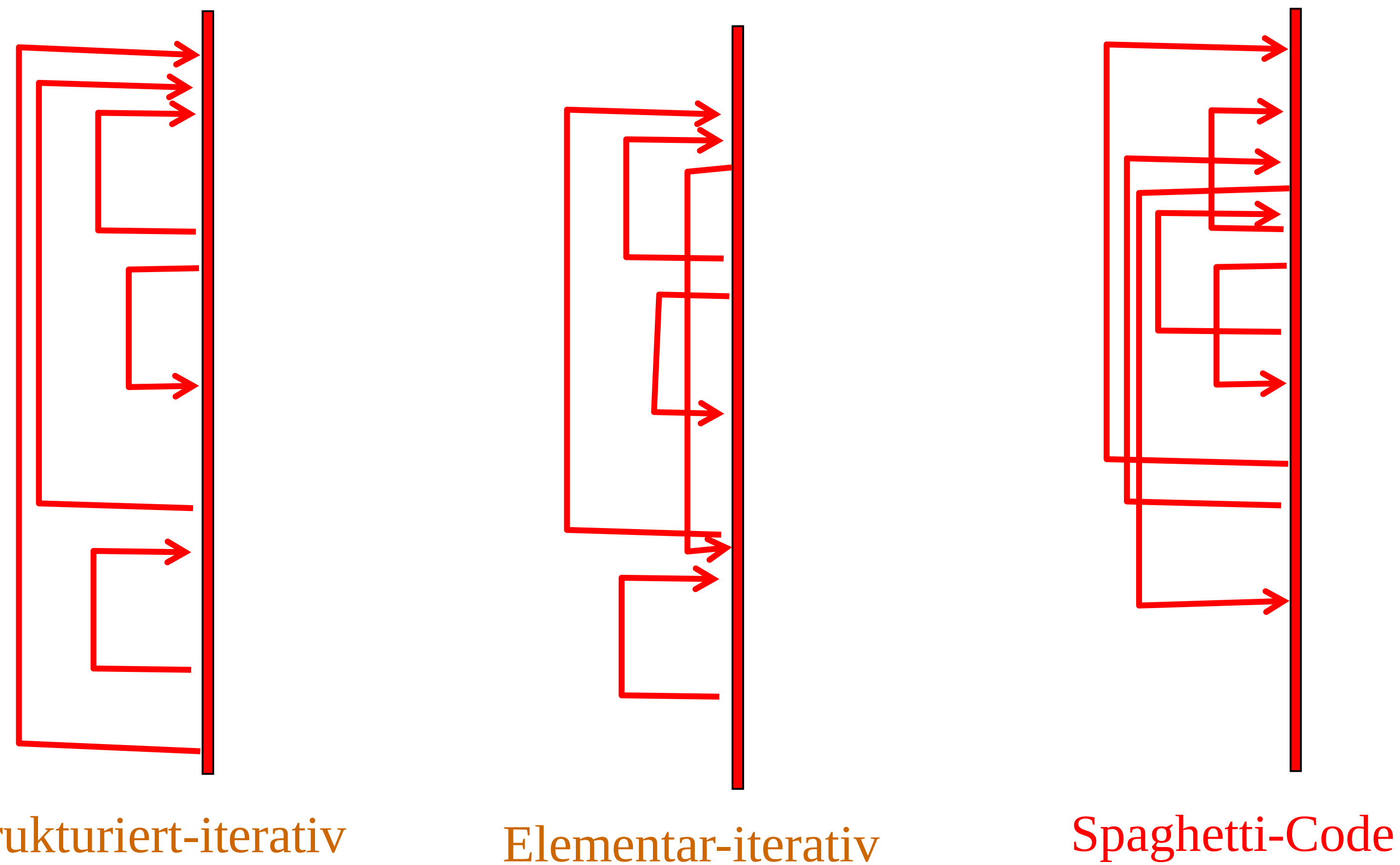
- Die elementar-iterative Beschreibungsform hat die nützliche und angenehme Eigenschaft:
  - Wir können über einzelne Schritte des Verfahrens sprechen
- Die „fahre fort"-Konstruktion entspricht unmittelbar der goto-Anweisung im Programmieren
  - Zur Anwendung von goto werden Schritte mit einer Marke (Label) versehen, um das Ziel des Sprunges zu kennzeichnen
- Anwendung von goto ist aber sehr gefährlich!
  - Strukturiert komplexe Programm nicht ausreichend
    - ⇨ Steuerungsverlauf kann verworren und unübersichtlich sein

Strukturiert-iterative Beschreibungsform

- Um den Steuerungsverlauf auch bei komplexen Algorithmen übersichtlich zu halten, schränkt man die Sprünge ein:
  - Schleifen der Flussdiagramme sind höchstens ineinander geschachtelt
  - Schleifen überkreuzen sich nicht!
- Im Arbeitsschritt des Grundschemas würde man z. B. nur wieder eine geschlossene Schleife oder einen (vorzeitigen) Sprung zurück zum Test des Trivialfalls erlauben
- Wir sprechen in diesem Fall von **strukturierten Sprüngen** im Gegensatz zu **freien Sprüngen**, die prinzipiell beliebige Ziele haben können

Strukturiert-iterative versus elemantar-iterative Beschreibungsform

- **Beispiel:** Schemata einiger Kontrollflüsse



Strukturiert-iterative Beschreibungsform

- Sprünge kommen zunächst nur noch *implizit* bei der Ausführung höherer Iterationsstrukturen vor
  - Dieses sind Fallunterscheidungen wie if-then-else
  - Oder insbesondere bei Schleifenkonstrukten (loop), wie etwa
    - ⇨ while
  - Diese bewirken, dass der Programmfluss in einer Schleife von einem Test zu einem Bearbeitungsschritt und wieder zurück zum Test geht

Strukturiert-iterative Beschreibungsform: while

- Die while-Schleife entspricht also der Konstruktion

```
M:    if (Bedingung)
        {Anweisungssequenz;
         goto M;
        } fi
```

Rekursive Beschreibungsform

- Im rekursiven Ansatz versucht man, ein vorgelegtes Problem $P(X)$ nach folgendem Schema in zwei Teilen zu lösen:

  1. [Basis] Gib eine direkte Lösung für den Fall an, daß die Problemstellung (Eingabe) X einfacher Natur ist.
  2. [Schritt] Führe eine Lösung für das Problem $P(X)$ für komplexe Problemstellungen X durch einen Schritt der Problemreduktion auf die Lösung des gleichen Problems für eine einfachere Problemstellung $P(X')$ zurück. Dabei muß $X > X'$ gelten für eine geeignete wohlfundierte Ordnungsrelation „>“.

- **Bem.:** Rekursive und iterative Beschreibungsformen sind gleich mächtig
  - Nach einer Formalisierung des Algorithmenbegriffs kann dies auch bewiesen werden!
    - ⇨ Etwa in der Vorlesung Informatik IV

Rekursive Beschreibungsform: Beispiel

- **Beispiel:** In gängiger mathematischer Notation könnte ein Verfahren zur Berechnung der Modulus-Funktion *a* mod *b* wie folgt aussehen:

$$\mathrm{mod}(a,b) = \begin{cases} a & \text{falls } a < b \\ \mathrm{mod}(a-b,\, b) & \text{falls } a \geq b \end{cases}$$

Rekursive Beschreibungsform: Beispiel

- **Beispiel (Forts.):** Um festzustellen, ob diese Berechnungsvorschrift einen Algorithmus darstellt, müssen wir folgende Fragen beantworten:
  - **Spezifikation**
    - Eingabe
    - Ausgabe
  - **Durchführbarkeit**
    - Endliche Beschreibung
    - Effektivität
    - Determiniertheit
  - **Korrektheit**
    - Partielle Korrektheit
    - Terminierung

Rekursive Beschreibungsform: Korrektheitsbeweis des Beispiels

- **Beispiel (Forts.):**
  - Spezifikation

a) **Eingabe:** Für welche Art von Zahlen wurde das Problem gestellt bzw. gilt unsere Rechenvorschrift? Antwort: Offensichtlich gilt $a, b \in \mathbb{Z}$, da sonst kein „Rest der Ganzzahldivision" definiert ist. Aus dem gleichen Grund müssen wir $b \neq 0$ fordern. Es gibt aber durchaus unterschiedliche Ansichten darüber, wie $a \bmod b$ zu definieren ist, falls $ab < 0$. Wir schließen diesen Fall der Einfachheit halber aus und fordern $a \geq 0, b > 0$.

b) **Ausgabe:** Was (genau) wird berechnet, bzw. wie ist $(a \bmod b)$ genau mathematisch definiert? Antwort: $(a \bmod b) := a - (a/b) \cdot b$. Hierbei ist a/b die Ganzzahldivision. Demnach fordern wir für das Resultat r der Berechnung $r = \mathrm{mod}(a, b)$ nach dem angegebenen Verfahren, daß $r = a - (a/b) \cdot b$ für alle $a, b \in \mathbb{Z}$, $a \geq 0$, $b > 0$.

Rekursive Beschreibungsform: Korrektheitsbeweis des Beispiels

- **Beispiel (Forts.):**

2. **Durchführbarkeit**
   a) **Endliche Beschreibung:** Dies ist offensichtlich gegeben.
   b) **Effektivität:** Fallunterscheidung und Subtraktion sowie erneuter (rekursiver) Eintritt in das Verfahren sind mechanisch ausführbar.
   c) **Determiniertheit:** Diese ist gegeben, da sich die Fälle $a < b$ und $a \geq b$ wechselseitig ausschließen. Mit den Bedingungen $a \leq b$ und $a \geq b$ wäre die Determiniertheit z. B. verletzt.

Rekursive Beschreibungsform: Korrektheitsbeweis des Beispiels

- **Beispiel (Forts.):**
  - **Partielle Korrektheit**
    - ⇨ Wir beweisen per Induktion über die Anzahl der Aufrufe von mod, dass
    - ⇨ mod(a, b) = (a mod b)
      - – d.h. das Ergebnis des Verfahrens stimmt mit der mathematischen Definition überein
    - ⇨ **[Basis]** Falls a<b, so ist

      (a mod b) = a - a/b • b = a - 0 • b = a = mod(a, b)

Notation für die mathematische Funktion „Modulus“

Notation für die durch unseren rekursiven Algorithmus gegebene Funktion „Modulus“

Notation für die Ganzzahldivision zweier ganzer Zahlen

Rekursive Beschreibungsform: Korrektheitsbeweis des Beispiels

- **Beispiel (Forts.) partielle Korrektheit von mod**
  - **[Induktionsschritt]** Wir nehmen als Induktionshypothese an, dass für a-b ≥ 0 und b>0 das folgende gilt:
    - ⇨ mod(a-b,b) = ((a-b) mod b)
  - Zunächst ist
    - ⇨ (a mod b) = a - (a/b) • b =

      a - b - ((a/b) • b - b) = (a-b) - ((a/b) -1) • b
  - Da a ≥ b ist weiter (a-b)/b = a/b - 1
  - Eingesetzt erhalten wir

    (a-b) - ((a-b)/b • b =

    ((a-b) mod b) =

    mod(a-b, b) =

    mod(a, b)
  - Die letzte Gleichheit gilt auf Grund der Konstruktion des Verfahrens
  - Die Induktionshypothese durften wir anwenden,
    - ⇨ da a-b ≥ 0 und b>0 falls b>0 und a ≥ b.

Rekursiver Algrithmus führt diese Reduktion in einem Schritt aus

mod(a, b) = mod(a-b, b)

Dies erfordert einen eigenen kleinen Beweis, der die Definition der Ganzzahldivision / verwendet:

a/b ist größtes c so dass c • b ≤ a

Definition von mod

Induktionshypothese

Rekursive Beschreibungsform: Korrektheitsbeweis des Beispiels

- **Beispiel (Forts.):**
  - Korrektheit

    b) **Terminierung:** Die Rekursion hält für $a \geq 0$, $b > 0$ immer an. Sei $(a_1, b_1), (a_2, b_2), \ldots, (a_i, b_i), (a_{i+1}, b_{i+1}), \ldots$ die Folge der Eingabetupel zu einer Aufrufsequenz von mod(a, b). Falls die Folge unendlich ist, so existiert eine unendliche Folge $a_1, a_2, \ldots, a_i, a_{i+1}, \ldots$ Es ist aber $a_i > a_{i+1}$, da $a_{i+1} = a_i - b$ mit $b > 0$, und gleichzeitig ist $a_i \geq b > 0$ nach Konstruktion des Verfahrens. Dies ist ein Widerspruch, da ausgehend von einem endlichen Wert keine unendlich absteigende Folge positiver natürlicher Zahlen existiert.

Bemerkungen: Diese Überlegungen stellen einen **Korrektheitsbeweis** dar

Die Termination konnte bei diesem Algorithmus also bewiesen werden; es gibt aber kein mechanisches Verfahren das bei einem beliebigen Algorithmus entscheiden kann, ob dieser terminiert oder nicht („Halte-Problem“)

Rekursive Beschreibungsform: Beispiel in PASCAL und C/Java

- Das rekursive Verfahren in mathematischer Notation können wir mit minimalen Änderungen in Programmiersprachen umsetzen, z.B.
  - PASCAL

```
FUNCTION modulus(a, b: integer): integer;
BEGIN
    IF  a < b THEN modulus := a
    ELSE  modulus :=  modulus(a-b, b)
END ;
```

  - oder in C (C++, JAVA)

```
int mod(int a, int b) {
  if(a<b) {return(a);}
  else {return(mod(a-b,b));}
}
```

Verifikation iterativer Algorithmen

- Die Verifikation iterativer Algorithmen wollen wir am Ende des Semesters betrachten
  - Wenn Ihnen Werkzeuge der mathematischen Logik schon etwas geläufiger sind
    - ⇨ Vgl. Vorlesung „Logik und diskrete Strukturen“
  - Und Sie etwas mehr „Programmiererfahrung“ gewonnen haben
    - ⇨ Und die Schwierigkeiten, „korrekte Programme“ zu entwickeln, etwas näher kennen gelernt haben

Algorithmen, Programmiersprachen, Maschinenmodelle

- Algorithmenbegriff beinhaltet ein „effektiv“ („mechanisch“) durchführbar
- Entwicklung von Algorithmen kann weitgehend ohne ein konkretes Maschinenmodell erfolgen
- Auch (moderne, höhere) Programmiersprachen sind so entworfen, dass Programme auf verschiedenen Computern ablaufen können
  - Es wird von speziellen Eigenschaften i.A. abstrahiert
- Trotzdem fließen Eigenschaften heutiger Computer auch in das Design von Programmiersprachen mit ein
  - Programme sollen nicht nur *effektiv* durchführbar sein, sondern auch *effizient*!

- Jetzt ist es an der Zeit, eine Programmiersprache näher kennen zu lernen!