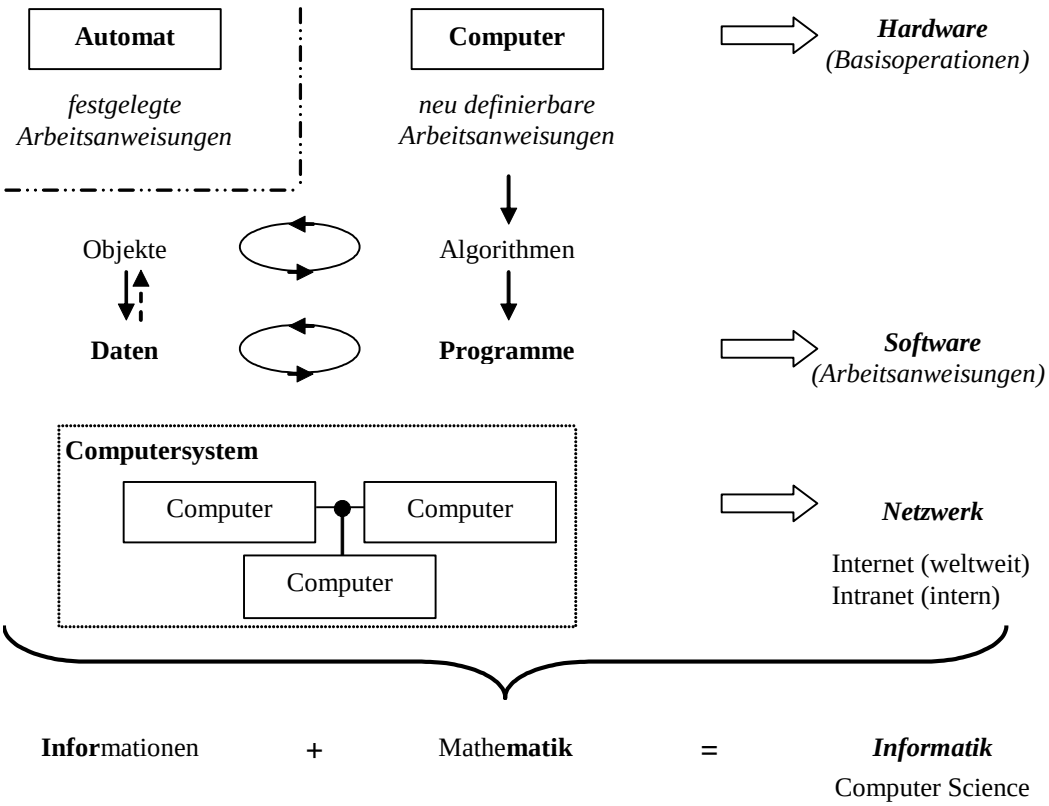
Inhalt

1 Algorithmen und Programme



Ein **Automat**, wie ein Getränkeautomat, kann nur *festgelegte Arbeitsanweisungen* ausführen. Im Unterschied dazu kann einem **Computer** die Vorschrift, nach der er arbeiten soll, jeweils *neu* vorgegeben werden.
So eine *Handlungsvorschrift* in Form eines **Algorithmus** *manipuliert* **Objekte** der realen Welt. Um dem Computer einen Algorithmus mitzuteilen, muss man diesen für ihn verständlich als **Programm** formulieren. Die zu manipulierenden Objekte müssen in ihm als **Daten** abgelegt werden.
Ein Computer setzt sich aus dem materiellen Teil, der **Hardware**, welche die *Basisoperationen* festlegt, und dem immateriellen Teil, der **Software**, welche es ermöglicht, *Arbeitsanweisungen* neu zu definieren, zusammen. Zur Software (*Systemsoftware*, *Anwendungssoftware*) gehören nicht nur die *Programme*, sondern auch die von diesen zu verarbeitenden *Daten*.
Computer können über Datenleitungen oder per Funk miteinander **vernetzt** sein, d.h. sie können untereinander Informationen austauschen. Man spricht dann von einem **Netzwerk**. **Intranet** ist die Vernetzung *innerhalb* einer Institution, **Internet** ist eine *weltweite* Vernetzung. Mehrere miteinander vernetzte Computer nennt man auch **Computersystem**.

Informatik (***Infor***mation + Mathe***matik***, engl. **Computer Science**) ist seit 1960 eine eigenständige Wissenschaft. Sie beschäftigt sich mit der *theoretischen Analyse und Konzeption*, aber auch mit der *konkreten Realisierung von Computersystemen* in den Bereichen der *Hardware*, der *Software*, der *Organisationsstruktur* und der *Anwendung*.

Inhalt des Kurses ist die Programmierung solcher Computersysteme. Deshalb werden wir uns zunächst mit dem Algorithmenbegriff auseinandersetzen.

1.1 Algorithmus

Das Wort **Algorithmus** ist auf den persisch-arabischen Mathematiker und Astronom **MUHAMMAD IBN MUSA AL-HWÂRIZMÎ** (ca. 780 – 859, Bagdad) nach seinem Werk „*Kitab al-jabr w'al-muqabala*" um 825 zurückzuführen, „*al-jabr*" wurde später zu Algebra, **AL-HWÂRIZMÎ** zu Algorithmus.
Das Werk ist leider nicht mehr erhalten. Es behandelt algebraische Gleichungen, das indische Zahlensystem und das Rechnen in diesem. Eine lateinische Übersetzung aus dem 12. Jahrhundert „*Algorithmi de numero Inderum*" existiert noch.
Der Algorithmenbegriff wird seit der Jahrhundertwende 19./20. Jh. verstärkt diskutiert. Friedrich L. Bauer und Gerhard Goos[1] beschreiben diesen folgendermaßen:

Ein *Algorithmus* ist eine *präzise*, das heißt in einer festgelegten Sprache abgefasste, *endliche* Beschreibung eines *allgemeinen* Verfahrens unter Verwendung *ausführbarer* Verarbeitungsschritte zur Lösung einer Aufgabe.

Beispiel Problem $r = ggT(m, n)$

Bestimmen des größten gemeinsamen Teilers r zweier natürlicher Zahlen m und n.
Euklidischer Algorithmus (nach dem griechischen Mathematiker **EUKLEIDES von ALEXANDREIA**, auch **EUKLID**, ca. 365 - 300 v. u. Z.):

ggT(130, 55):	**130** / **55** =	2	Rest	20
	55 / 20 =	2	Rest	15
	20 / 15 =	1	Rest	5
	15 / **5** =	3	Rest	0

⇒ **ggT(130, 55) = 5**

(Mathematik Beweis in drei Schritten: Algorithmus terminiert, liefert gT, liefert ggT)

Ein Algorithmus ist eine *Handlungsvorschrift*, keine Problembeschreibung.

Ein Algorithmus heißt **abbrechend** (**terminierend**), wenn er die gesuchte Größe nach endlich vielen Schritten liefert.

[1] BAUER, F. L. und GOOS, G. (1971):
Informatik - Eine einführende Übersicht; Springer Verlag. 2 Bde., 1. Aufl. 1971, 4. Aufl. 1991/92;
russische Übersetzung, Mir, Moskau, 1976;
polnische Übersetzung, Wydawnictwa Naukowo - Techniczna, Warschau, 1977.

1.2 Programm

Wie setzt man in einem Computer einen Algorithmus um?

1.2.1 Hauptkomponenten des Aufbaus eines Rechners

John von Neumann – Architektur, 1947

Prozessor (CPU – central processing unit)

- ***Rechenwerk*** (ALU – arithmetical and logical unit):
  durch Schaltungen realisierte **Basisoperationen** des Rechners.
- ***Steuerwerk*** (CU – control unit)
  Koordinations-, Kontroll-, Organisationsaufgaben;
  versorgt das Rechenwerk mit **Arbeitsanweisungen**

Ein Computer ist mit Hilfe eines **Rechenwerks** in der Lage, *einfache Operationen* in hoher Geschwindigkeit auszuführen. Die *Reihenfolge* der auszuführenden Operationen und die notwendigen Operanden werden mittels eines **Steuerwerks** diesem zugeführt.

⇒ **Basisoperationen**: Das zu lösende Problem muss durch einfache Operationen beschreibbar sein.

⇒ **Arbeitsanweisungen**: Die Reihenfolge der auszuführenden Operationen muss bekannt sein, notwendige Daten müssen bereitstehen.

Ein *Programm* legt fest, welche *Arbeitsanweisungen* in welcher Reihenfolge zur Lösung des Problems mit Hilfe der vorhandenen *Basisoperationen* notwendig sind.

Modellrechner

Wir gehen davon aus, dass unser *Modellrechner* folgendermaßen aufgebaut ist[2]:

Speicher

Als *Arbeitsspeicher* steht eine endliche Anzahl von Zellen zur Verfügung.
Diese können genau eine Zahl einer nach oben beschränkten und damit endlichen Teilmenge der Menge der natürlichen Zahlen aufnehmen, $M \subset N$.

constant `0, 123`

Die einzelnen Zellen werden symbolisch mit einzelnen Kleinbuchstaben adressiert:

identifier `a, b, z`

Basisoperationen

Das *Rechenwerk* führt verschiedene Operationen aus. Operanden sind natürliche Zahlen oder die Inhalte von Speicherplätzen.
Rechenoperationen:
Addition `+`, Subtraktion `-`, Multiplikation `*`, ganzzahlige Division `/`, Rest `%`

expression `a % b`

[2] Die vollständige Definition der Syntax befindet sich im Anhang.

Vergleichsoperationen: größer >, gleich == und ungleich !=

condition `a != 0`

Arbeitsanweisungen

1. ***Wertzuweisungen*** als ***einfachste Arbeitsanweisungen***:
   Es wird einem Speicherplatz ein Wert zugewiesen. Links steht der Name des Speicherplatzes, dann folgt ein ‚=' und schließlich ein Wert als zuzuweisender Speicherinhalt. Dieser wird direkt als natürliche Zahl angegeben, aus einem anderen Speicherplatz übernommen oder vom Rechenwerk berechnet. Abgeschlossen wird eine Wertzuweisung mit einem Semikolon:

   assignment
   ```
   a = 10;
   b = a;
   c = a * b;
   ```

2. ***Zusammengesetzte Arbeitsanweisungen***:
   a. Mehrere Arbeitsanweisungen können *hintereinander* ausgeführt werden.

      sequence `a = m; b = n; c = a % b;`

   b. Arbeitsanweisungen können *wiederholt* ausgeführt werden.

      iteration `while( n != 0) { s = s + n; n = n - 1;}`

   c. Eine *Auswahl* von Arbeitsanweisungen soll ausgeführt werden.

      selection
      ```
      if( n > m) { c = m; m = n; n = c;}
      if( m > n) { x = m;} else { x = n;}
      ```

3. Abschluss: Nichts sonst ist erlaubt.

Beispiel Programm *r* **=** ***ggT(*** *m, n****)***

```
a = m;
b = n;
c = a % b;
while( c != 0)
{
  a = b;
  b = c;
  c = a % b;
}
r = b;
```

Protokoll (Speicherbelegungsänderung)

	m	*n*	*r*	*a*	*b*	*c*
Eingabe	**130**	**55**				
Verarbeitung				130	55	20
				55	20	15
				20	15	5
				15	5	0
Ausgabe			**5**			

Weitere Beispiele

Programm *f* **=** ***fak(*** *n****)***

```
f = 1;
a = n;
while( a != 1) { f = f * a; a = a - 1;}
```

Protokoll (Speicherbelegungsänderung)

	n	*f*	*a*
Eingabe	**3**		
Verarbeitung		1	3
		3	2
		6	1
Ausgabe		**6**	

Programm *s* **=** ***summeGauss(*** *n****)***

```
r = n % 2;
if( r == 0) { g = n; u = n + 1;} else { g = n + 1; u = n;}
s = g / 2; s = s * u;
```

1.2.2 Ein Programm für einen Algorithmus

Ein **Programm** für einen Algorithmus ist eine *Folge notwendiger Arbeitsanweisungen bei gegebenen Basisoperationen*. Es legt somit fest, welche Basisoperationen in welcher Reihenfolge zur Ausführung des Algorithmus abzuarbeiten sind.

Problem $s = kgV(m, n) = m * n / ggT(m, n)$

Bestimmen des kleinsten gemeinsamen Vielfachen *s* zweier natürlicher Zahlen *m* und *n* unter Verwendung des euklidischen Algorithmus zur Berechnung des größten gemeinsamen Teiler $r = ggT(m, n)$:

Programm $s = kgV(m, n)$, drei Varianten:

1.
```
a = m;
b = n;
c = a % b;
while( c != 0)
{
  a = b;
  b = c;
  c = a % b;
}
```
} *ggT(m, n)*

```
s = m * n;
s = s / b;
```

Dieses Programm kann durch unseren Modellrechner verarbeitet werden.

2.
```
r = ggT ( m, n);
s = m * n;
s = s / r;
```

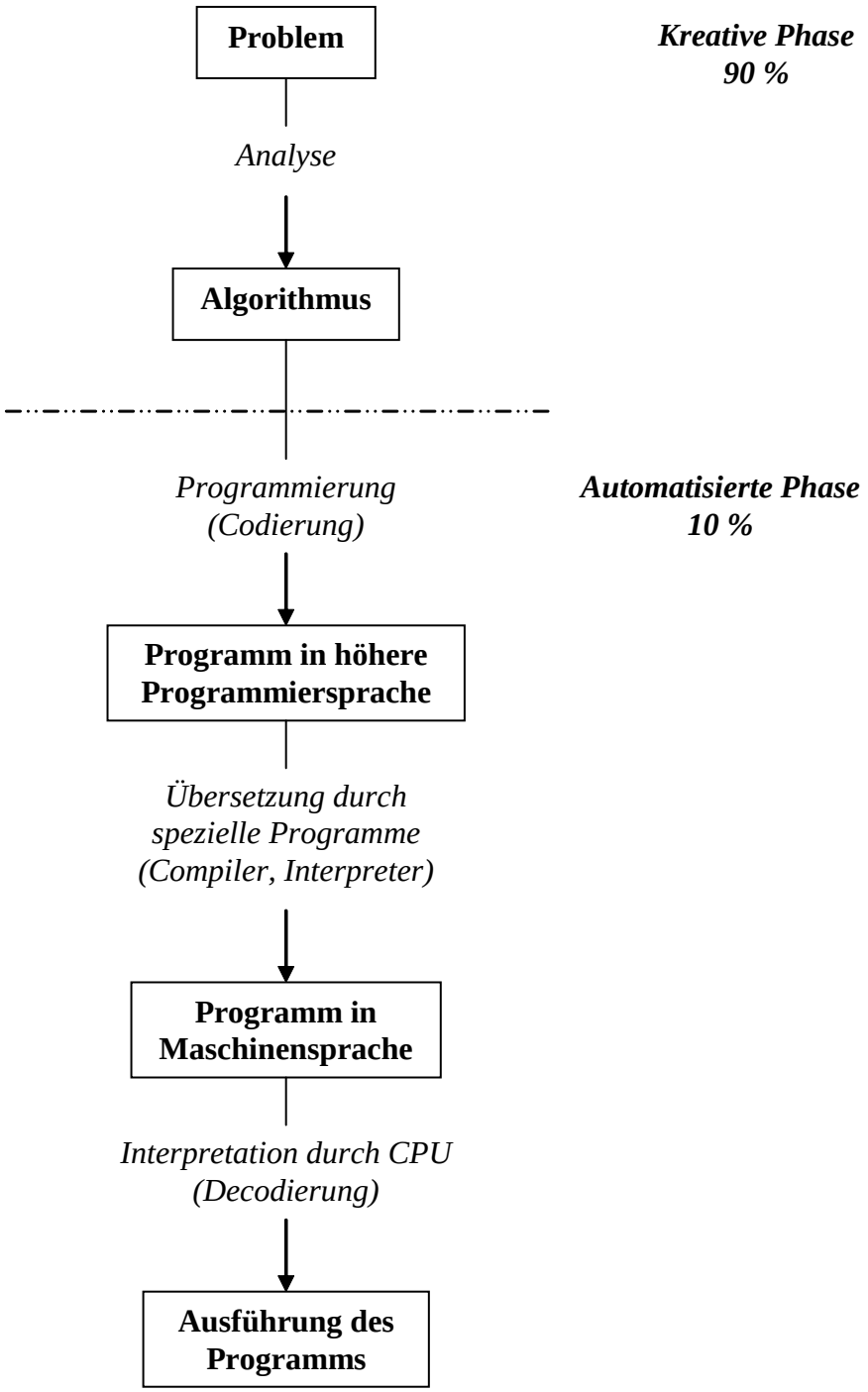
Ein *Aufruf eines anderen Programms* (als Unterprogramm oder Funktion) innerhalb eines Programms ist in den Basisoperationen unseres Modellrechners *nicht* vorgesehen!

3.
```
s = m * n / ggT( m, n);
```

Komplexe Anweisungen können mittels der angegebenen Basisoperationen unseres Modellrechners *nicht* verarbeitet werden!

Moderne Programmiersprachen ermöglichen eine *komplexe* Syntax analog 2. und 3. Solche Programme nennt man **problemorientiert**. Einzelne Arbeitsanweisungen sind nicht sofort ausführbar, eine *Zerlegung* in die vorhandenen Basisoperationen ist notwendig. Diese wird in der Praxis automatisch durch spezielle Programme (**Compiler**, **Interpreter**) erledigt. Das 1. Programm könnte Ergebnis einer solchen **Übersetzung** sein. Es besteht nur aus ausführbaren Arbeitsanweisungen unseres Modellrechners. Man nennt es ein **maschinenorientiertes** Programm.

In den folgenden Beispielen erlauben wir für unseren Modellrechner auch problemorientierte Programme, d.h. sowohl ein Aufruf eines anderen Programms (2.) als auch komplexe Ausdrücke (3.) sind erlaubt.



Ein *Programm* ist ein codierter Algorithmus.
Ein *Algorithmus* wird durch ein Programm interpretiert.

1.2.3 Rekursion

Schließlich wird noch ein weiteres problemorientiertes Programm für den euklidischen Algorithmus betrachtet. Dieses wurde als **Rekursion** entwickelt, *es ruft sich selbst auf.*

Programm *r = ggT(m, n)* **mit Rekursion**

```
a = m;
b = n;
```

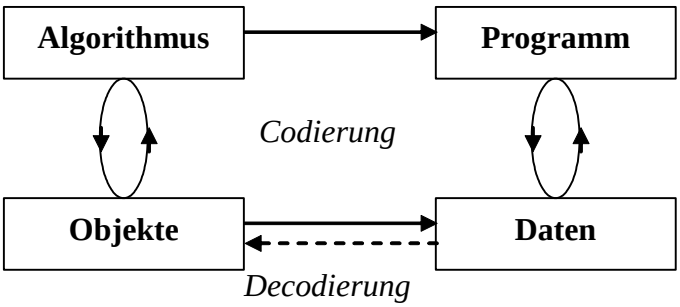
```
if( b != 0)
{
  r = ggT( b, a % b);
}
else
{
  r = a;
}
```

Rekursion, problemorientiert

Zu *einem* Algorithmus kann man *verschiedene* Programme formulieren, zum einen in Abhängigkeit von den vorhandenen Basisoperationen, zum anderen aber auch in Abhängigkeit von den verwendeten Arbeitsanweisungen.

1.2.4 Codierung

Algorithmen dienen i.d.R. der Manipulation mathematischer *Objekte*. Sowohl der Algorithmus als auch die Objekte sind dem Rechner zugänglich zu machen. Sie sind zu **codieren**.



Die *Daten* und *Programme* sollten weitestgehend mit den *Objekten* und den *Algorithmen* **verträglich** sein. Im Idealfall heißt das

$$x + y = z \xrightarrow{\varphi} \varphi(x) \oplus \varphi(y) = \omega \text{ mit } \varphi^{-1}(\omega) = z,$$

wobei x, y, z die Objekte, φ die Codierungsfunktion, + für eine beliebige Operation und $\oplus$ für deren **Interpretation** (Basisoperation oder Programm) im Rechner steht.

Durch die **Endlichkeit** des Rechners bestehen Einschränkungen im Wertebereich der darstellbaren Objekte, mathematische Gesetze können nicht vollständig übertragen werden!

- ⇒ Eingabefehler, Verfahrensfehler, Rechenfehler
- ⇒ Fehlerfortpflanzung
- ⇒ fehlerbehaftete Resultate
- ⇒ **Numerik**

***Numerik* beschäftigt sich u.a. mit Fragen der rechentechnischen Umsetzung mathematischer Probleme, wie weit sich Fehler und deren Fortpflanzung auf Resultate auswirken und Computerergebnisse brauchbar sind.**

1.3 Programmierparadigmen

Zur Programmentwicklung wurden im Verlaufe der letzten 60 Jahre zahlreiche Programmiersprachen entwickelt, die im Wesentlichen in vier Programmierparadigmen unterteilt werden.

Imperative Programmierung

Ein Programm besteht aus einer *Folge von Wertzuweisungen* (s. Modellrechner).

Dabei ist das *Variablen-Konzept* und damit die *Wertzuweisung* die wichtigste Anweisungsform. *Eingabewerte und Zwischenergebnisse* werden in *Variablen* gespeichert (**John-von-Neumann-Rechner**, **1947**).

Algol ... ***Algo****rithmic* ***L****anguage*, Fortran, Pascal, C

Funktionale Programmierung

Ein Programm ist eine *mathematische Funktion*. Der Funktionswert liefert das Ergebnis des Programms bei gegebenen Argumenten.

Lisp ... ***Lis****t* ***P****rogramming*, Haskell

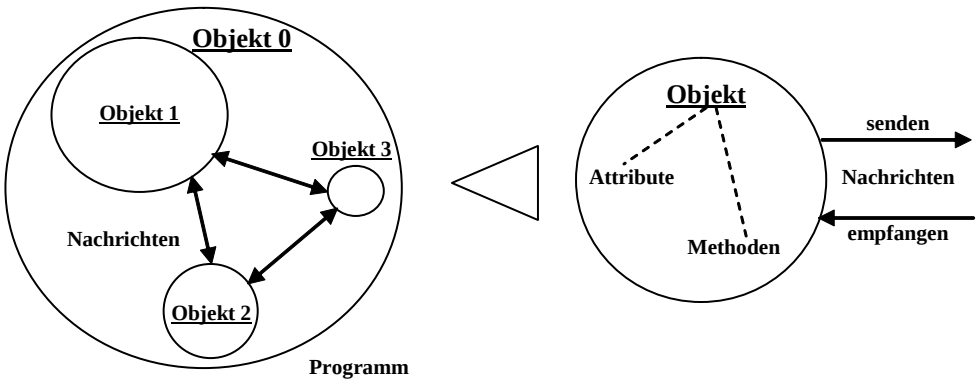
Logische Programmierung

Ein Programm ist eine *Frage*. Aus *Axiomen* (wahre Aussage) wird mittels *Regel* die Frage mit „yes“ bei Korrektheit und „no“ bei Fehlschlag beantworten. Die Programmierung basiert auf *Logik*.

Prolog ... ***Pro****gramming in* ***Log****ik*

0bjektorientierte Programmierung

Ein Programm ist ein *Objekt*, welches selbst aus Objekten aufgebaut ist. Der Datenaustausch zwischen den Objekten erfolgt über *Nachrichten*.



Dazu werden *Objekte der „realen Welt" und deren Verhalten* auf entsprechende *Objekte des Programms* wiedergespiegelt. Der Ursprung dieser Idee liegt in der **Simulationstechnik**.
Solche Objekte besitzen *individuelle Informationen* (**Attribute**) und *Verhaltensweisen* (**Methoden**). **Ein Objekt verwaltet seine Attribute und Methoden selbst.** *Kein* Objekt hat das Recht, direkt auf die Attribute anderer Objekte zuzugreifen (**information hiding**). Objekte treten durch **Nachrichten** untereinander in Verbindung. Ein Objekt kann Nachrichten *versenden*, *empfangen* und auf Nachrichten *reagieren*.

Klassen fassen Objekte gleichen Aufbaus zusammen, sie dienen der *Datenabstraktio*n in der objektorientierten Programmierung.

Smalltalk *rein objektorientiert*

Beispiel Hund und Herrchen
Eine *Klasse* **`Hund`** fasst alle Hunde verschiedener Rassen, Größe und Farbe zusammen. Jeder einzelne Hund selbst ist *Objekt* der Klasse und hat seine individuellen *Attribute*, wie Rasse: Riesenschnauzer, Farbe: schwarz, Größe: sehr groß. Hat ein Hund ein Herrchen, so ist dieses ein Objekt einer *anderen* Klasse, nennen wir sie **`Mensch`**. Das Herrschen gibt dem Hund den Befehl „Sitz". Es schickt also eine *Nachricht* an den Hund. Der Hund reagiert darauf oder auch nicht. Er allein entscheidet, wie er sich auf den Befehl „Sitz" verhält.

Hybridsprachen

Sprachen, die zunächst das *imperative* Paradigma verfolgten, erhielten nachträglich eine *objektorientierte Erweiterung*. Sie besitzen demzufolge sowohl imperative als auch objektorientierte Sprachbestandteile und gestatten eine *nicht* streng objektorientierte Programmierung.

Objekt Pascal, C++, Java *Hybridsprachen, sowohl imperativ als auch objektorientiert*

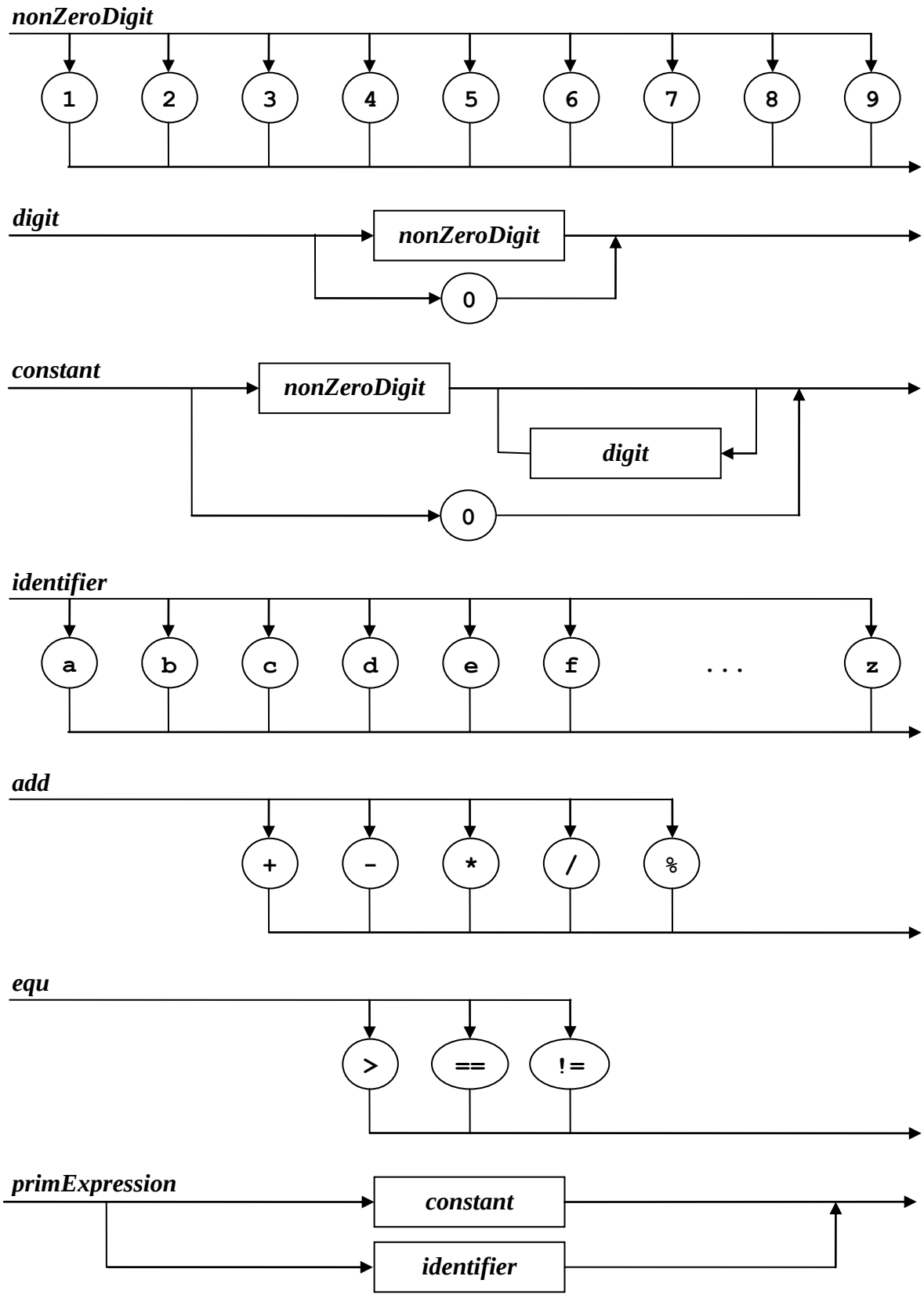
1.4 Anhang zum Modellrechner

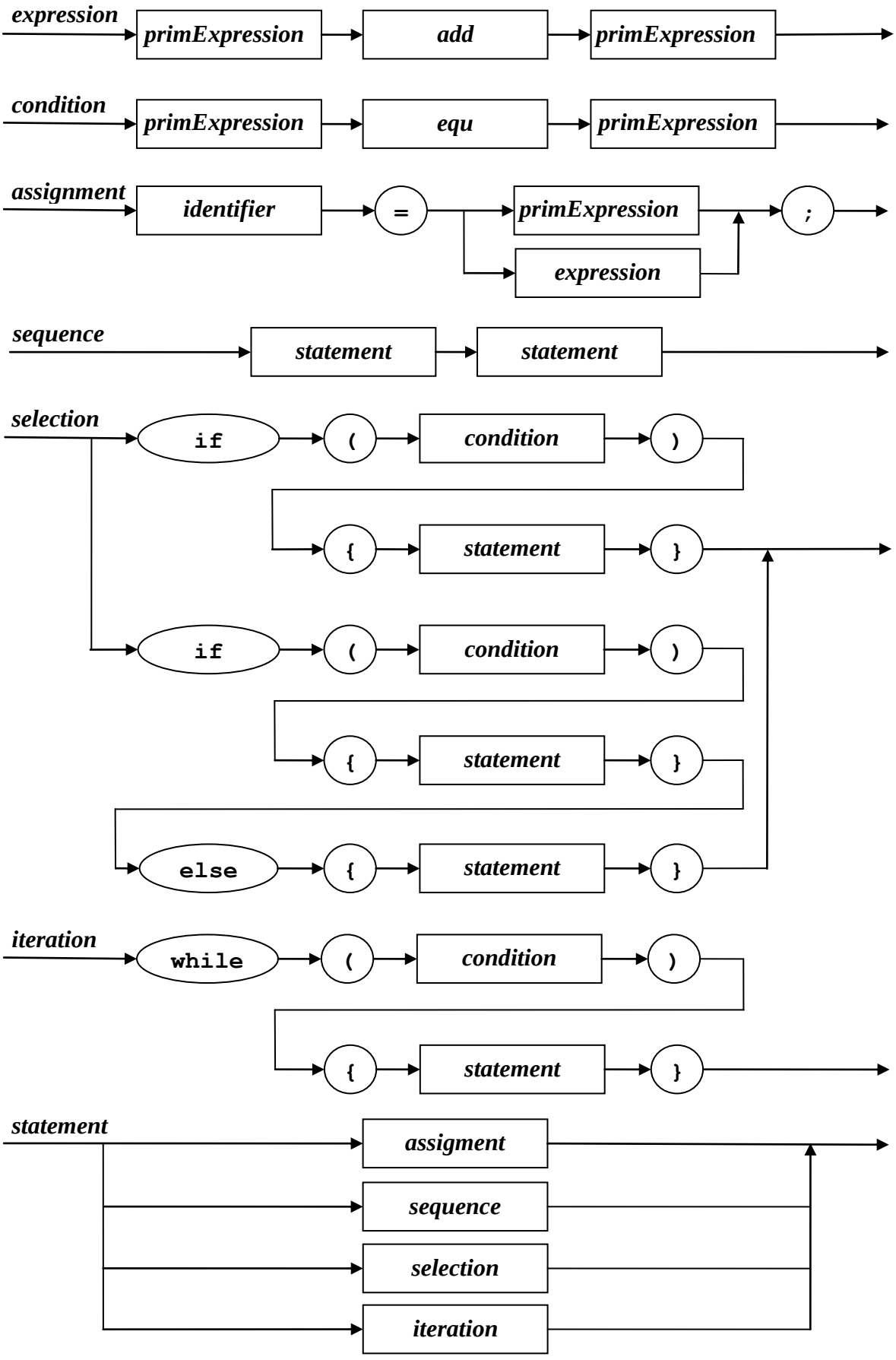
Definition der Sprache des Modellrechners, entspricht dem Dijkstra-Diagramm.

1.4.1 Backus-Naur-Form (BNF)

nonZeroDigit	::=	'1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'	
digit	::=	'0' | ***nonZeroDigit***	Ziffern
constant	::=	'0' | ***nonZeroDigit { digit }***	Konstante
identifier	::=	'a' | 'b' | 'c' | 'd' | 'e' | 'f' | ... | 'z'	Variable
add	::=	'+' | '-' | '*' | '/' | '%'	Rechenoperator
equ	::=	'>' | '==' | '!='	Vergleichsoperator
primExpression	::=	***constant*** | ***identifier***	einfache Ausdrücke
expression	::=	***primExpression add primExpression***	Rechenausdruck
condition	::=	***primExpression equ primExpression***	Vergleichsausdruck
assignment	::=	***identifier*** '=' ***primExpression*** ';'| ***identifier*** '=' ***expression*** ';'	Wertzuweisung
sequence	::=	***statement statement***	Hintereinanderausführung
selection	::=	'if' '(' ***condition*** ')' '{' ***statement*** '}'| 'if' '(' ***condition*** ')' '{' ***statement*** '}''else' '{' ***statement*** '}'	Auswahlanweisung
iteration	::=	'while' '(' ***condition*** ')' '{' ***statement*** '}'	Schleifenanweisung
statement	::=	***assignment*** | ***sequence*** | ***selection*** | ***iteration***	Programm

1.4.2 Syntaxdiagramm




expression
primExpression
add
primExpression
condition
primExpression
equ
primExpression
assignment
identifier
=
primExpression
;
expression
sequence
statement
statement
selection
if
(
condition
)
{
statement
}
if
(
condition
)
{
statement
}
else
{
statement
}
iteration
while
(
condition
)
{
statement
}
statement
assigment
sequence
selection
iteration