Automatentheorie und Formale Sprachen

Teil 1 - Einführung

Prof. Dr. Hans-Werner Sehring

NORDAKADEMIE
HOCHSCHULE DER WIRTSCHAFT

Agenda

Abschnitt 1

Organisatorisches

Organisatorisches

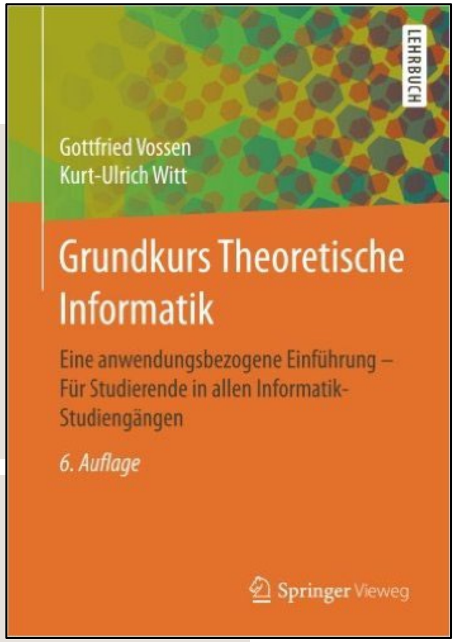
Stellung im Studienplan und Zeiten

- 3. Semester: 4 SWS
- 9 Veranstaltungen mit je 4 Stunden für Vorlesung und Übungen, jeweils 15 Minuten Pause
- Abschluss mit Klausur (Anfang Q2)

Moodle-Kurs für Materialien

- `https://moodle2.nordakademie.de/course/view.php?id=6687`,
  Einschreibeschlüssel: I140_AufS23a und I140_AufS23b
- Nachrichten- und Diskussionsforum
- Vorlesungsfolien, Übungsaufgaben, Lösungen, JFLAP-Automatenmodelle
- Prüfungsrelevant sind die in der VL behandelten Inhalte!

Literatur



Hauptempfehlung

Vossen, G. / Witt, K.-U. 2016. Grundkurs Theoretische Informatik (6. Auflage), Springer Vieweg

Online verfügbar unter
https://link.springer.com/content/pdf/10.1007%2F978-3-8348-2202-4.pdf

Weitere Literatur

- **Priese, L. und Erk, K.** Theoretische Informatik – Eine umfassende Einführung (4. Auflage), Springer Vieweg; online verfügbar unter https://link.springer.com/content/pdf/10.1007%2F978-3-662-57409-6.pdf
Alternative zu Vossen/Witt

- **Schöning, U. 2008** Theoretische Informatik – kurz gefasst (5. Auflage), Spektrum Akademischer Verlag.
Knappe Einführung ohne Schnickschnack

- **Hopcroft, J. / Motwani, R. / Ullman, J. 2011** Einführung in Automatentheorie, Formale Sprachen und Berechenbarkeit (3. Auflage), Pearson Studium.
Das Standardwerk. Wichtige Beweise, viele Übungsaufgaben. Bei ausreichenden Sprachkenntnissen lieber zum englischen Original greifen, falls verfügbar.

Kurze Abfrage

Haben Sie Vorkenntnisse in theoretischer Informatik?

Welche besonderen Erwartungen haben Sie an diese Veranstaltung?
Was haben Sie über diese Veranstaltung gehört?

Übersicht über die Veranstaltungsinhalte

Grundelemente: Alphabete, Wörter, Sprachen

Endliche Automaten

- Graphische Darstellung und formale Definition
- Deterministische Endliche Automaten
- Nichtdeterministische Endliche Automaten
- ε-Automaten
- Transduktoren (Mealy- und Moore-Maschinen)
- Eigenschaften, Umformungen, Minimierung

Kellerautomaten

Turing-Maschinen

Reguläre Ausdrücke

- Notation
- Rechenregeln
- Äquivalenz von Automaten und Ausdrücken

Grammatiken

- Formale Definition
- Syntaxbäume und Ableitungen
- Chomsky-Hierarchie
- Äquivalenz von Sprachen und Automaten
- Abschlusseigenschaften und Entscheidbarkeit

Abschnitt 2

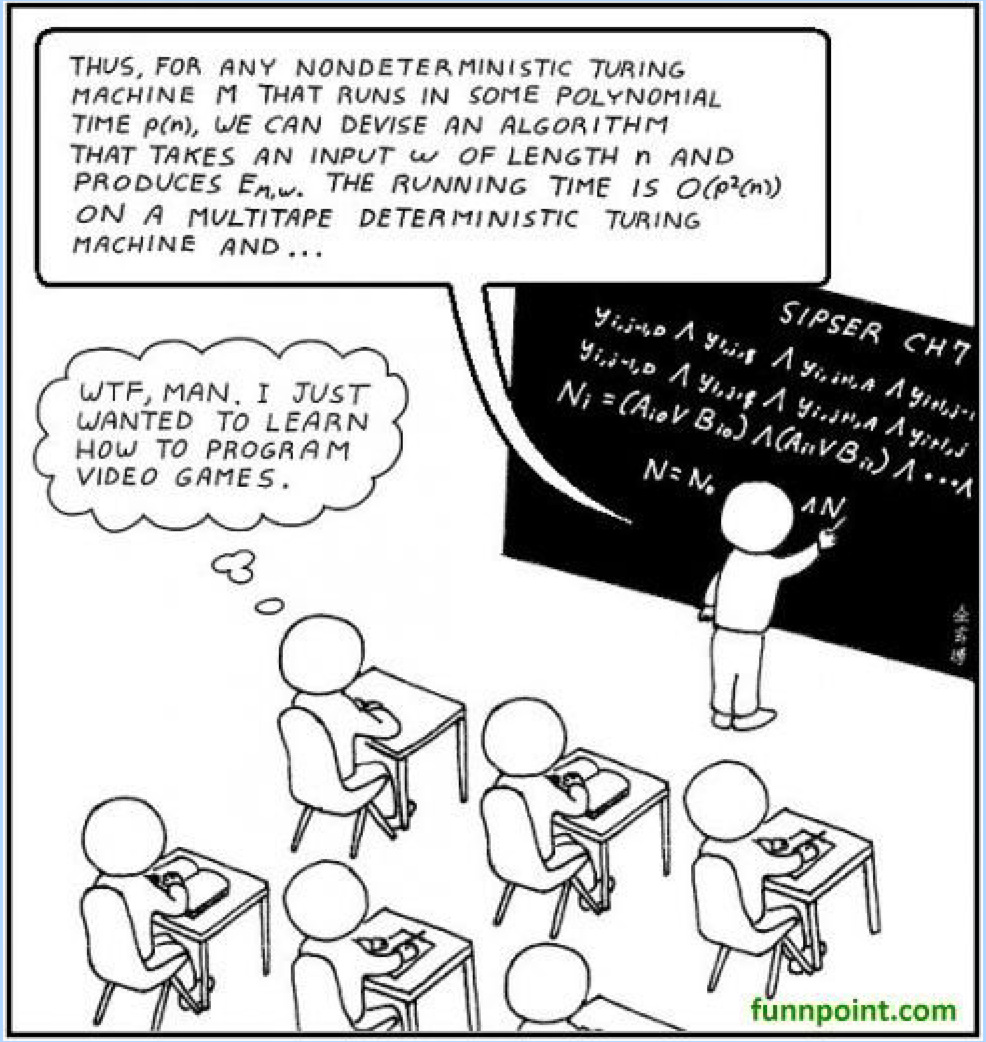
Motivation theoretische Informatik

Nutzen der theoretischen Informatik für Praktiker

Wofür Theorie?

Sie haben Programmieren gelernt, z.B. in *Einführung in die Programmierung*.

- Können Sie jetzt zu jedem Problem eine Lösung programmieren?
- Können Sie das beste Programm zu einem Problem schreiben?



Für jedes Problem das passende Programm? (1)

Können Sie jede Funktion programmieren?

Church-Turing-These: Intuitiv berechenbare Funktionen sind die Turing-berechenbaren Funktionen.
„Intuitiv berechenbar": das, was sie auch gerade im Sinn hatten.

Ist aber jede Funktion Turing-berechenbar?

„Einfaches" Beispiel: terminierende und nicht terminierende Programme

- Beispiele:
  `(define fac (lambda [n] (if (= n 0) 1 (* n (fac (- n 1))))))` terminiert (für `n` >= 0)
  `(define fac (lambda [n]                 (* n (fac (- n 1))) ))` ☹
- Frage: Können wir uns eine Funktion „halt" schreiben, die terminierende Funktionen erkennt?
  (Das wäre doch praktisch bei der Qualitätssicherung.)

Für jedes Problem das passende Programm? (2)

Das berühmte **Halteproblem**: Kann ein Programm entscheiden, ob ein (anderes) Programm terminiert?

Nein. (nicht für allgemeine Programme)

Skizze des Widerspruchsbeweises:

- Angenommen, es existiert Funktion `halt` so, dass `(halt f)` = `#true` gdw. `f` terminiert
- Definiere `(nothalt f)` als Funktion, die terminiert, gdw. `f` nicht terminiert:

```
(define nothalt
  (lambda [f] (cond [(halt f) (nothalt f)]
                    [else     #false     ])))
```

- Was ergibt `(nothalt nothalt)`?
  „`nothalt` terminiert genau dann, wenn `nothalt` nicht terminiert.“ ↯

Es gibt also nicht- (Turing-) berechenbare Funktionen!

Die besten Programmierer schreiben die besten Programme! (1)

Finden Sie zu jedem Problem das beste Programm?

Gegenfrage: Was ist denn „das beste“?

- Welche Kategorien von „gut“ gibt es?
- Wie gut ist die Aufgabe lösbar?

Theoretische Informatik: Fragen der **Komplexitätsbetrachtung**

- *Ist mein Programm „schnell“? Wie schnell geht es denn?* 🤔
- *Verschwendet mein Programm Hauptspeicher? Mit wie wenig Speicher geht es denn?* 🤔
- *Kann ich mein Programm weiter parallelisieren? Wie viel Nebenläufigkeit steckt in meinem Programm?* 🤔

Die besten Programmierer schreiben die besten Programme! (2)

Beispiel: Matrixmultiplikation $R^{l \times m} \times R^{m \times n} = R^{l \times n}, (A, B) \to C = A \cdot B$

- Schulalgorithmus: $c_{ik} = \sum_{j=1}^{m} a_{ij} \cdot b_{jk}, 1 \leq i \leq l, 1 \leq k \leq n$
- Braucht offenbar $l \cdot n \cdot m$ Rechenschritte, also „ca.“ n^3.
- Aber: die Ergebnismatrix hat $l \cdot n$ Elemente, könnte doch also m-mal schneller in „ca.“ n^2 berechnet werden.
- Es könnte also ein Programm geben, das so schnell Matrizen multipliziert. (Und kein Schnelleres.)

Und diese Fragen sollen (in der theoretischen Informatik) nicht durch „best effort“, Herumprobieren oder gar ein Bauchgefühl beantwortet werden, sondern durch einen (mathematischen) Beweis!

Dazu schaffen wir in dieser Vorlesung zumindest das grundlegende Werkzeug.

Gründe für die Beschäftigung mit Theorie

Praktische Bedeutsamkeit außerdem (als Grundlage) für

- Gefühl für die Grenzen des (mit Software) machbaren
- Bewusstsein für Datenstrukturen (Einfluss der Datenmodellierung auf Algorithmen)
- Bewusstsein für Datenformate (eindeutige Darstellung? wie schwierig zu lesen und zu schreiben?)
- Einfache Mustererkennung, z.B. `regex (7)`.

Abschnitt 3

Einführendes Beispiel

Einführendes Beispiel

Kontrolle des Eintritts in ein Schwimmbad.

Eintrittspreis: 2 €, bezahlbar in 50 Cent-, 1 Euro- und 2 Euro-Stücken.

Keine Rückgabe von überzahltem Geld. [Vossen/Witt]

Einführendes Beispiel – imperative Sichtweise

Pseudo-Code:

```
integer akk := 0
wiederhole
    integer einwurf := leseEingabe
    falls einwurf = 50|100|200
        dann
            akk := akk + einwurf
        sonst
            Ausgabe Fehlermeldung
bis akk >= 200
```

Erinnerung: so ähnlich haben Sie einen Fahrscheinautoamten in EidOOP programmiert

Einführendes Beispiel – funktionale Sichtweise

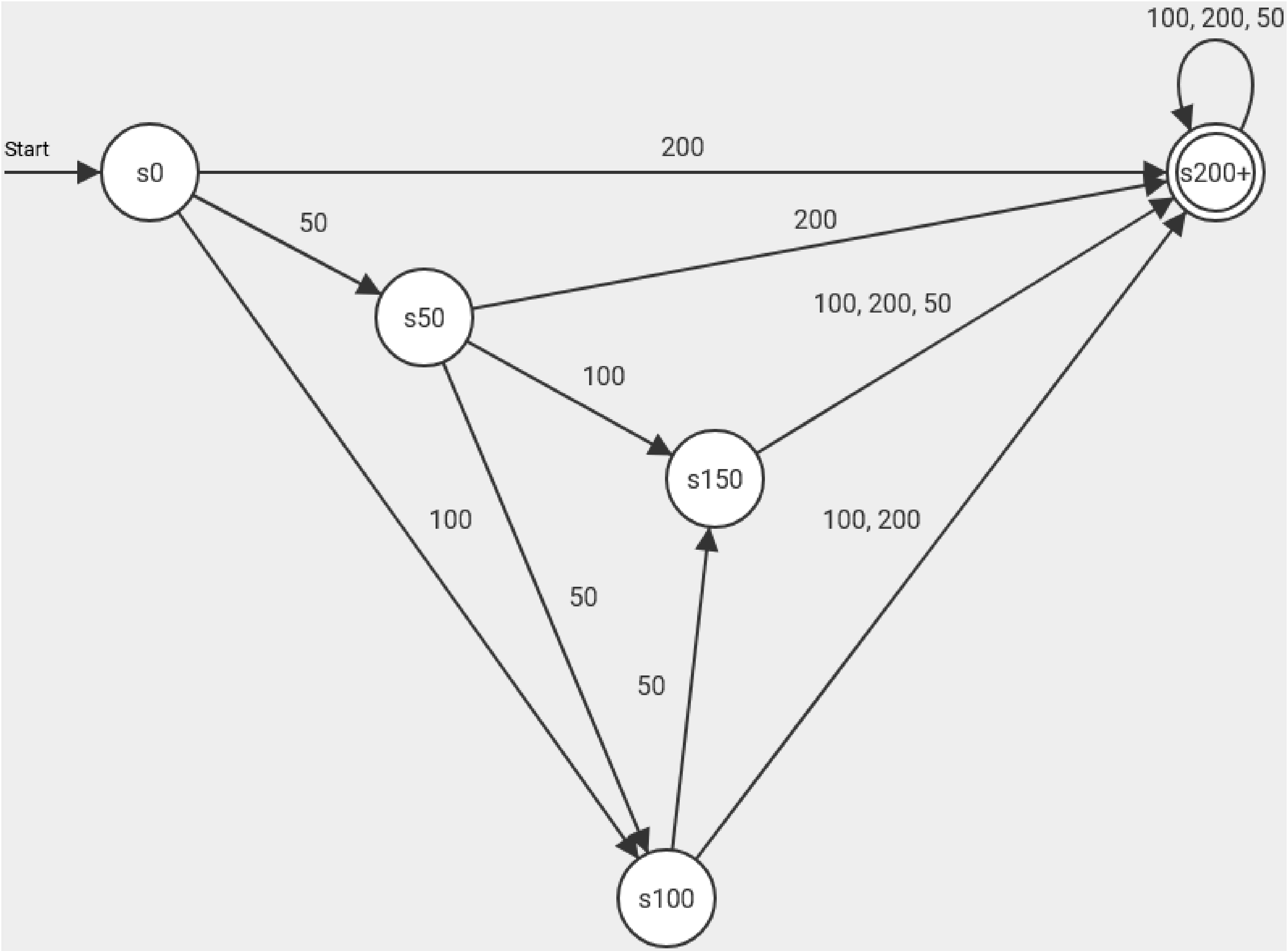
Pseudo-Code:

```
(define eintrittMoeglich?
  (lambda [lvm]
          (<= 200 (foldl + 0 lvm))))
```

Auswertung:

```
> (eintrittMoeglich? '(200))
#true
> (eintrittMoeglich? '(50 50 50 50))
#true
> (eintrittMoeglich? '(100 50))
#false
```

Einführendes Beispiel – Automat



Formale Betrachtung des Automaten (1)

Automaten (genauer: **Deterministische Endliche Zustandsautomaten**) sind ein Quintupel:

$$(\boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{\delta}, \boldsymbol{s_o}, \boldsymbol{F})$$

$\boldsymbol{\Sigma}$ bezeichnet das **Eingabealphabet**.

$$\Sigma = \{50, 100, 200\}$$

$\boldsymbol{S}$ ist die **Zustandsmenge**.

$$S = \{s_0, s_{50}, s_{100}, s_{150}, s_{200+}\}$$

$\boldsymbol{\delta}: \boldsymbol{S} \times \boldsymbol{\Sigma} \rightarrow \boldsymbol{S}$ ist die **Zustandsüberführungsfunktion**.

Siehe Diagramm und nächste Folie

$\boldsymbol{s_o}$ ist der **Startzustand**.

$\boldsymbol{F} \subseteq \boldsymbol{S}$ ist die Menge der **Endzustände**.

$$F = \{s_{200+}\}$$

Automaten (genauer: **Deterministische Endliche Zustandsautomaten**) sind ein Quintupel:
$(\Sigma, S, \delta, s_o, F)$
Σ bezeichnet das **Eingabealphabet**.
$\Sigma = \{50, 100, 200\}$
S ist die **Zustandsmenge**.
$S = \{s_0, s_{50}, s_{100}, s_{150}, s_{200+}\}$
$\delta: S \times \Sigma \rightarrow S$ ist die **Zustandsüberführungsfunktion**.
Siehe Diagramm und nächste Folie
s_o ist der **Startzustand**.
$F \subseteq S$ ist die Menge der **Endzustände**.
$F = \{s_{200+}\}$

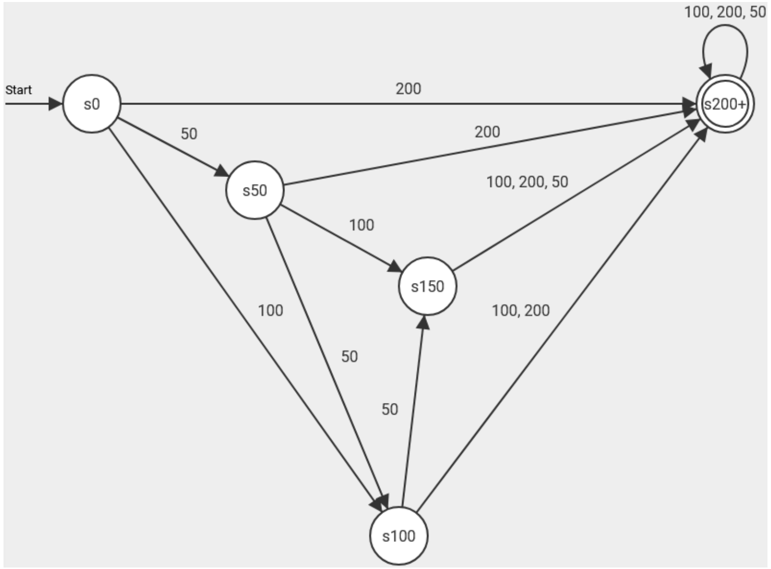
Formale Betrachtung des Automaten (2)

Automaten (genauer: **Deterministische Endliche Zustandsautomaten**) sind ein Quintupel:

$$(\boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{\delta}, \boldsymbol{s_o}, \boldsymbol{F})$$

Die Zustandsüberführungsfunktion $\boldsymbol{\delta}: \boldsymbol{S} \times \boldsymbol{\Sigma} \rightarrow \boldsymbol{S}$ lässt sich auch in Form einer Zustandstabelle darstellen:

δ	50	100	200
s_0	s_{50}	s_{100}	s_{200+}
s_{50}	s_{100}	s_{150}	s_{200+}
s_{100}	s_{150}	s_{200+}	s_{200+}
s_{150}	s_{200+}	s_{200+}	s_{200+}
s_{200+}	s_{200+}	s_{200+}	s_{200+}



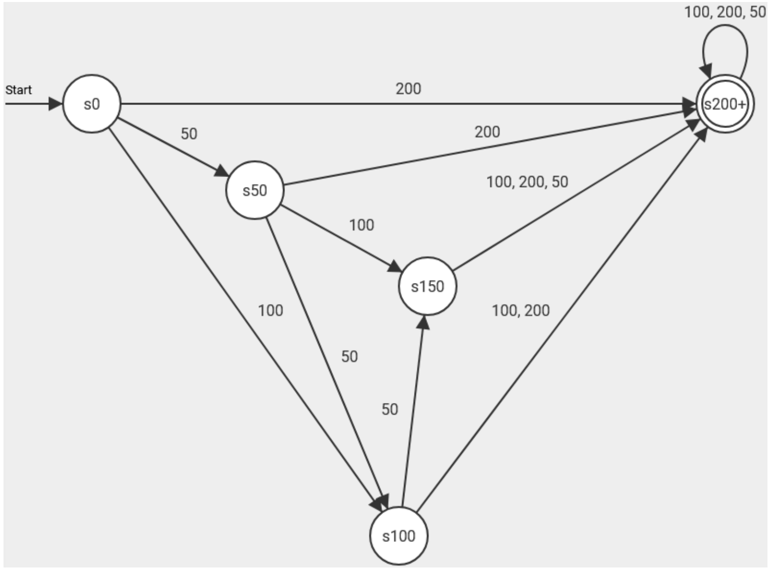
Formale Betrachtung des Automaten (3)

Wörter sind endlich lange Zeichenfolgen über dem Alphabet.

Beispiele: **50 50 100**, **100 50**

Ein Automat **akzeptiert** Wörter, wenn er sich nach ihrer vollständigen Abarbeitung in einem Endzustand befindet.

Beispiel: **50 50 100**



Die **Sprache** $\boldsymbol{L}$ des Automaten umfasst alle von ihm akzeptierten Wörter.

$$\boldsymbol{L} = \{\mathbf{50\ 50\ 50\ 50}, \mathbf{50\ 50\ 50\ 100}, \mathbf{50\ 50\ 50\ 200}, \mathbf{50\ 50\ 100}, \dots\}$$

Eine **Konfiguration** $\boldsymbol{k}$ beschreibt den aktuellen Stand der Verarbeitung eines Wortes: aktueller Zustand und noch zu verarbeitende Eingabe.

Beispiel: Wort **50 50 100** nach Einwurf der ersten 50 Cent: $\boldsymbol{k} = (\boldsymbol{s_{50}}, \mathbf{50\ 100})$

Entscheidungsprobleme

Folgende Probleme sind für Endliche Automaten immer durch Algorithmen eindeutig lösbar:

- **Wortproblem**
  „Kann ich mit meinen Münzen die Karte bezahlen?“
- **Leerheitsproblem**
  „Ist es überhaupt möglich, eine Karte zu erwerben?“
- **Endlichkeitsproblem**
  „Gibt es unendlich viele Möglichkeiten, eine Karte zu bezahlen?“
- **Äquivalenzproblem**
  „Sind zwei Eintrittsautomaten identisch?“

NORDAKADEMIE
HOCHSCHULE DER WIRTSCHAFT

NORDAKADEMIE gAG Hochschule der Wirtschaft

Köllner Chaussee 11 · 25337 Elmshorn · Tel.: +49 (0) 4121 4090-0 · E-Mail: info@nordakademie.de · Web: www.nordakademie.de