Info I

Was ist Informatik

Von–Neumann–Rechner

1. Rechnerkern
2. Arbeitspeicher
3. Peripherie
4. Bus–Systeme

Algorithmus

- Ein Algorithmus ist eine mit formalen Mitteln beschreibbare, mechanisch nachvollziehbare Verarbeitungsvorschrift zur Lösung einer Klasse von Problemen. Eine exakte (formale) Beschreibung ist notwendig, um Lösungsverfahren für die unterschiedlichen Probleme so zu formulieren, daß ihre Bearbeitung in Form eines Programms von einem Rechner übernommen werden kann: Sprache zur Abfassung von Algorithmen.
- Hat drei Eigenschaften:
  1. Endliches Verfahren zur Lösung von Problemen
  2. Sprachlich präzis und eindeutig formuliert
  3. Erfordern geeignete Methoden zur Strukturierung der von Algorithmen manipulierten Daten und Datenstrukturen

Information

- Syntaktischer, semantischer und pragmatischer Teil
- **Symbole**: Codierung, denen ein Empfänger eine Bedeutung zumessen kann
- **Nachricht**: Mitteilung vereinbarter Symbole
- **Shannonscher Informationsbegriff**: Maß für den mittleren Informationsgehalt, den man mit einer gegebenen Codierung übertragen kann

Vom Problem zum Algorithmus

Formale Grundlagen

- Eine **Menge** M ist eine Zusammenfassung von wohldefinierten und wohlunterschiedlichen Objekten unserer Anschauung oder unseres Denkens (welche *Elemente* von M genannt werden) zu einem Ganzen. Die Beschreibung kann sowohl **extensional** als auch **intensional** erfolgen
- Operationen s. S.14 – 18

Aussagenlogik

- Konjunktive Normalform (KNF), Disjunktive Normalform (DNF)
- Tautologie, Widerspruch
- Ein (logischer) *Kalkül* K ist ein algorithmisches Verfahren, mit dem man logische Schlüsse syntaktisch in endlich vielen Schritten ableiten kann, ohne auf die Interpretation einzugehen. (**Sammlung rein "mechanisch" anzuwendender syntaktischer Umformunsregeln**)
- *Modus Ponens* (Abtrennungsregel): [a AND (a¢b)] ¢ b
- *Resolutionsregel*: [(a OR b) AND (!a OR c)] ¢ (b OR c)

- *Implikation*: (a¢b) ↔ (!a OR b)
- *Hornformel*: Formel ist in KNF und jedes Disjunktionsglied hat höchstens ein positives Literal

Prädikatenlogik

- Symbole: Konstanten, Variablen, Funktionen, Prädikate, Operatoren und Quantoren
- Terme (induktiv definiert)
- Freie und gebundene Variablen
- Der Prädikatenkalkül ist unentscheidbar

Aspekte des Algorithmenentwurfs

- Ein Algorithmus ist keine Funktion, sondern ein Verfahren zur Realisierung einer Funktion
- *Strukturen der Ablaufkontrolle*: Sequenz, Auswahl (Selektion), Wiederholung(Iteration)
- *Rekursive Algorithmen*: A., die sich selber aufrufen
- *Datenstrukturen*: z.B. Array, queue, stack, trees
- *Datentyp*: Zusammenfassung von Wertebereichen und Operationen
- *Axiomatische Spezifikation*: Angabe von Syntax und Semantik
- Verifikation:
  - *Terminierung* (gemäß der Vorbedingung kommt der Algorithmus zum Ende; Nachweis über vollst. Induktion)
  - *partielle Korrektheit* (ein partiell korrekter A. muß nicht unbedingt enden oder ein Ergebnis erzeugen, wenn er aber eines erzeugt, so ist dieses auch korrekt)
  - *Totale Korrektheit* (Aus partieller Korrektheit und der Terminierung folgt die totale Korrektheit)
- Testen (dynamische Methode) vs. Beweisen (statische Methode)

Entwurfsprinzipien für Algorithmen

Divide and Conquer

- Teilen, Herrschen und Vereinigen
- **Türme von Hanoi**
  - Alle Steine mit Nummer x, x+1, ..., n auf Stapel i
    - Alle Steine x+1, ..., n auf Stapel k=6−(i+j)
    - Stein x von i auf j
    - Alle Steine x+1, ..., n von k=6−(i+j) auf Stapel j
      - Alle Steine befinden sich auf Stapel j
- **Binäre Suche**
  - Algorithmus zum Suchen in sortierten Listen
  - Mittlerer Suchaufwand: k = log2(len(X)+1)
- **Maximum−Contigous−Subvector Problem (maximum subarray Problem)**

- Maximale Summer aller Elemente einer zusammenhängenden Teilfolge
- Strategie: Folge in zwei Teilstücke aufteilen, wobei Randstücke an der Trennstelle extra betrachtet werden müssen (vielleicht können die ja verschmolzen werden) ¢ *Maximum Crossing*
- Dieser Vorgang des *mergens* kommt auch bei vielen anderen Algorithmen zum Tragen: Mergesort, dichtestes Punktepaar,

Greedy–Methode

Auf jeder Stufe wird die Entscheidung getroffen, ob ein spezieller Eingabewert zu einer optimalen Lösung gehört oder nicht.

- **Konvexe Hülle**
  - Package wrapping: Drehen im Uhrzeigersinn, Schnittpunkt wählen; wiederholen bis wieder beim Startpunkt
- **Minimum spanning tree**
  - Menge von Kanten, die alle Knoten eines Graphen beinalten, deren Kosten (Kanten bewertungen) aber minimal sind
  - Algorithmus nach *Kruskal*:
    - Kanten aufsteigend nach Kosten sortieren
    - Kante zur Menge hinzufügen, falls so kein Kreis entsteht (Bilde Mengen Ci der verbundenen Knoten, eine Kante e=(v1, v2) wird also nur dann hinzugefügt, wenn v1 aus Ci und v2 aus Cj und Ci nicht gleich Cj ist. Dann wird Ck=Ci u Cj gesetzt)
- **Single Source Shortest Path (SSSP)**
  - Dijkstra (S. 92):
    - Mit S={} anfangen und dann zu S immer die Knoten aus V\S hinzufügen, deren Abstand zu S minimal ist. S enthält dann

Dynamisches Programmieren

Die Menge der aufzuzählenden Entscheidungensfolgen läßt sich oft reduzieren, indem man einige Entscheidungsfolgen, die unmöglich sein können bei der Aufzählung wegläßt (*Optimalitätsprinzip*).
Die *optimale Entscheidungsfolge* hat die Eigenschaft, daß unabhängig vom Anfangszustand und der Anfangsentscheidung die übrigen Entscheidungen eine optimale Entscheidungsfolge bilden müssen unter Berücksichtigung der aus des ersten Entscheidung resultierenden Zustandes.

- **Travelling salesman problem**
  - g(i, S): Länge des kürzesten Weges, der bei Knoten i beginnt, durch alle Knoten in S verläuft und bei Knoten 1 endet; wobei i und 1 nicht aus S sein dürfen.
  - g(1,V–{1}): optimale Handlungsreisenden Tour
  - Optimalitätsprinzip:
    $g(1, V-\{1\}) = \min \{ c_{1k} + g(k, V-\{1,k\}) \}$, 2<=k<=n
    $g(i, S) = \min \{ cij + g(j, S-\{j\}) \}$, j aus S

Traversal

- Präorder, Inorder und Postorder
- **Tiefensuche**
  - Realisierung mittels Stack oder rekursiv
- **Breitensuch**
  - Realisierung mittel Schlange (queue)

- **Teifensuche auf Graphen (DFS)**
  - Wie normale Tiefensuche nur mit Markieren der bereits besuchten Kanten.
- **Breitensuche auf Graphen (BFS)**
  - Auch mit Markierung

Backtracking

Systematisches Durchsuchen des Lösungsraumes.

- **Graham Scan zur Ermittlung der konvexen Hülle einer Punktmenge**
  - Ankerpunkt: kleinste Y–Koordinate, größte X–Koordinate
  - Sortierung aller Knoten derart, daß ein Polygonzug entsteht, der alle Punkte umfaßt und sich selbst nicht schneidet
  - Die ersten beiden Punkte sind fest, danach wir dann für jeden Punkt überprüft, ob der Vorgänger auf der konvexe Hülle liegt
- **Das n–Damenproblem**
  - n Damen auf einem nxn großen Spielfeld unterbringen, so daß sie sich nicht gegenseitig schlagen können
  - Rekursiv: für einen Wert Xi, der eine Lösung werden könnte, werden alle möglichen Kombinationen überprüft

Korrektheit und Effizienz

Verifikation mittels Zusicherungskalkül

- **Methode von Hoare**
  - Regeln bestehen aus Axiomen und Ableitungsregeln

    1.) Leere Anweiung

    2.) Zuweisung:
    - Textuelle Substitution:

      $$\frac{B}{B[X := t]}$$
    - Interferenzregel (Leibniz):

      $$\frac{X = Y}{B[z := X] = B[x := Y]}$$
    - Wertzuweisung:

      $$\frac{P \rightarrow Q[x := t]}{\{P\} X := t \{Q\}}$$

    3.) Sequenz von Anweiungen

    $$\frac{\{P_1\} A_1 \{Q_1\}, \{P_2\} A_2 \{Q_2\}, Q_1 \rightarrow P_2}{\{P_1\} A_1 ; A_2 \{Q_2\}}$$

    4.) Selektion

    $$\frac{\{P \wedge B\} a_1 \{Q\}, \{P \wedge \overline{B}\} a_2 \{Q\}}{\{P\} ifBthena_1 elsea_2 \{Q\}}$$

    5.) Schleifen

    $$\frac{\{P \wedge B\} A \{P\}}{\{P\} whileBdoA \{P \wedge \overline{B}\}}$$
- **Methode von Dijkstra**

- Schwächste Vorbedingung *wp[1](S, Q)*
- Gilt *wp(S, Q)* vor der Ausführung von S, so wird
  - S terminiert und
  - Die Nachbedingung Q erfüllt sein
- Es gibt wieder 5 Regeln:
  1.) Leere Anweisung: skip
  2.) Zuweisung: x:=E (dabei muß sowohl E als auch Q(E) definiert sein)
  3.) Sequenz von Anweisungen: S_1 ; S_2 (wp(S_1;S_2,Q) = wp(S_1, wp(S_2,Q))
  4.) Selektion
  5.) Iteration: while B do S:
  wp(while B do s, Q) =
  wp(skip, Q n !B) v
  B n wp(S, Q n !B) v
  B n wp(S, B) n wp(S; S, Q n !B) v ...
- Beweis der totalen Korrektheit (Terminierung garantiert)
- Problem der Vereinfachung der Formeln bei Iteration)

Aufwandsabschätzung

- **O–Kalkül:** Der Zeitaufwand wird nicht in Zeiteinheiten berechnet, sondern in Teilschritten des Algorithmus. Das eigentliche Problem ist, die aufwandbestimmende Variabilität in Abhängigkeit von der Eingabeklasse zu bestimmen.
- **Definition:**
  f, g sind Funktionen in R^+. f=O(g) bedeutet: f wächst asymptotisch höchstens so wie g:

  $$\exists x_0 \exists c \left[x_0 \in \Re \wedge c \in \Re^+ \wedge \forall x \left[(x \in \Re \wedge x > x_0) \rightarrow f(x) \leq c \cdot g(x)\right]\right]$$
- **Ω–Kalkül:** f=Ω(g) heißt, das f mindestens so stark wächst wie g.
- **Θ–Kalkül:** f=Θ(g) heißt, daß f genauso stark wächst wie g.
- **Rekurrenzrelation:** Zeitaufwandsabschätzung rekursiver Algorithmen

  Allgemeiner Fall: $$t(n) = \begin{cases} b & n = 1 \\ a \cdot t\left(\frac{n}{c}\right) + b \cdot n & n > 1, c = c^k \end{cases}$$

  Lösung: $$t(n) = \begin{cases} O(n) & a < c \\ O(n \cdot lob_c n) & a = c \\ O\left(n^{\log_c a}\right) & a > c \end{cases}$$

Testen

- **Hauptansätze:** statistische Programmanalyse, ablaufbezogenes / datenbezogenes / funktionsbezogenes Testen, Back–to–Back Testen
  - **Ablaufbezogenes Testen (structured testing):** Programm wird gegen sich selbst getestet, das Objekt ist der Ablaufplan des Programms (¢Ablaufprotokoll: Anweisungsüberdeckung, Ablaufzweigüberdeckung und Ablaufpfadüberdeckung). Es läßt sich nur eine bestimmte Klasse von Fehlern auffinden (z.B. grobe Abbruchfehler, unerreichbare Zweige, Irrpfade, endlose Schleifen.
    Nicht erkannt werden: Tippfehler, inkonsistente Schnittstellen, Abweichungen von der Spezifikation ¢ "nose rubbing test"
  - **Datenbezogenes Testen:** Test mit zufällig erzeugten Werten

  - **Funktionsbezogenes Testen:** Code−Abschnitte einzelnen Funktionen zuordnen und deren Verhalten gezielt testen
  - **Back−to−Back Testen:** Zwei Versionen werden gegeneinander getestet
  - **Nach Fehlerlokalisierung / Erkennung:** Regressionstest
- **Methoden zur Fehlerlokalisierung:**
  - Hauruck−Verfahren: Dump, Kontrollausgaben
  - Analytische Methoden: Vergleich Input/Output, Fehler genau auf Ursache untersuchen, Hyptothesen aufstellen
- **Testvorgang**
  - **Planung:** Aufruffolge, Datenfluß
  - **Testvorbereitung:** Testumgebung schaffen (Testbett, Dummies, Testdaten)
  - **Testdurchführung und −auswertung:** Schreibtischtest, Testlauf, Fehlerlokalisierung, schrittweises Testen mit/dann ohne Dummies, Erkenntnisse nutzen

Programmentwicklung

- **Programmanforderungen:**
  - Korrektheit } absolut
  - Wartbarkeit } absolut
  - Adaptierbarkeit
  - Portabilität
  - Kompatibilität
  - Zuverlässigkeit
  - Verfügbarkeit
  - Benutzerfreundlichkeit
  - Leistung und Effizienz
  - Wiederverwendbarkeit
  - Verteilbarkeit
- **Korrektheit:**
  Spezifikation, Externe Spezifikation, Interne Spezifikation
- **Wartbarkeit:**
  Modular, gut dokumentiert, Dokumentation, Benutzerhandbuch
- **Adabtierbarkeit:**
  modular, funktionale hierarchische Gliederung
- **Portabilität:**
  Vors: Adabtierbarkeit
- **Zuverlässigkeit:**
  Sicherheit gegen interne Fehler, Ausfallsicherheit, Fehlertoleranz, Robustheit (Kompromiß zu Effizienz)
- **Entwicklungsmethoden:**
  Verifikation, Entwurf, Strukturierungsmethoden, Fehlertoleranz, Zuverlässigkeit
- **Programmentwurf:**
  Hierarchische Abstraktionsebenen, Strukturierte Programmierung
- **UML**
- **Softwarelebenszyklus:**
  Anforderungsanalyse/Spezifikation ¢ Entwurf / Spezifikation ¢ Programmierung / Modultest ¢ Integration / Systemtest ¢ Auslieferung / Wartung

Info II

Rechnerarchitektur

Rechneraufbau

- Rechenwerk (Register, ALU, Shifter, Status−Register) } CPU
- Leitwerk } CPU
- Speicher
- Ein−/Ausgabe
- Verbindung über Daten−, Adreß−, Steuerleitungen und Leitungen zum Übertragen der Befehle
- Maschinenbefehl: Opcode+Adreßfeld
- Verschiedene Architekturen: SISD, SIMD, MISD, MIMD

Codierung

- Codierung (Übertragung, Zeichenvorrat, Kompression, Fehlerkorrektur, Verschlüsselung)
- Codier−Funktion muß injektiv sein
- Zahlkodierungen:
  - **BCD−Code:**
    Keine Zahlkodierung sonder Ziffernkodierung (jede Ziffer einer Zahl wird durch 4 Bit dargestellt)
    Einstz bei wenig rechenintensiven Anwendungen und bei speziellen Instruktionen (¢BCD−Arithmetik)
  - **Vorzeichen/Betragsdarstellung:**
    Erstes Bit zeigt an ob positiv oder negativ
    Fallunterscheidung bei Addition/Subtraktion
  - **Einerkomplement (Stellenkomplement):**
    Um eine Zahl mit (−1) zu multiplizieren wird einfach das Stellenkomplement gebildet
    Bei der Addition muß ein Übertrag neu aufaddiert werden
  - **Zweierkomplement (Echtes Komplement):**
    Erhält man durch Stellenkomplement und Addition von 1
    Die Null hat eine eindeutige Darstellung
    Bei Addition kann der Übertrag ignoriert werden

Rationale und reelle Zahlen

- **Festkommadarstellung:**
  Längenangabe (n, m) mit n für ganzzahligen Anteil, m für gebrochenzahligen Anteil
  + Einfaches Format, einfach zu realisierende arithmet. Operationen
  − Kommata der Operanden müssen an gleicher Stelle stehen, gefährliche Rundungen, ineffiziente Darstellung
- **Gleitkommadarstellung:**
  $z = \pm m \cdot b^{\pm \exp}$ (m *Mantisse*, b *Basis*, exp *Exponent*)
  keine eindeutige Darstellung
  - Halbalgorithmische Darstellung: nicht eindeutig
  - Normalisierte halbalgorithmisch Darstellung: $1/b \leq |m| < 1$ für $z \neq c$ (0,x...)
  - Normalisierte Darstellung: $1 \leq |m| < b$ oder z=0 (x,...)
    VZ exp | exp (r Bits) | VZ m | m (p Bits)

- **Gleitpunktdarstellung nach ANSI/IEE754:**
  Ausnahmen bei Div0, Overflow, underflow, ungenau
  - **Arithmetik:**
    Es können nichtnormalisierte Zwischenergebnisse auftreten ¢ nachträgliche Normalisierung ist i.a. erforderlich
    Zahlendichte unterschiedlich (zwischen 0,1 und 1 genauso viele Zahlen wie zwischen 10.000 und 100.000 bei Basis 10)
    U.U. können Verletzungen der Assoziativität bei Adition und Subtraktion auftreten
    Exakte Gleichheit durch Mindestgrenze bei Subtraktion ersetzen
    Subtraktion fast gleich großer Zahlen vermeiden
    Fortpflanzung von Rechenfehlern bei großer Anzahl von Rechenschritten
    Division durch kleine Werte vermeiden

Kodierung / Informationstheorie

- **Fano–Bedingung:** kein Codewort ist Präfix eines anderen Codewortes (**prefix code**) ¢ Darstellung des Codes als Baum
- **Huffmancodierung:** Kodierung ergibt sich aus Wahrscheinlichkeiten des Auftretens der Zeichen.
  Die Coderwörter der beiden Symbole, die am seltensten auftreten, haben gleiche Länge und unterscheiden sich nur in der letzten Bit–Stelle
  Verfahren: Symbole absteigend nach Wahrscheinlichkeiten sortieren, dann von hinten nach vorne die beiden unwahrscheinlichsten jeweils zusammenfassen.Anordnung in einem Baum merken

  E(l) $= \sum_{i=1}^{n} p(a_i) \cdot l(a_i)$

  Var(l) $= \sum_{i=1}^{n} p(a_i) \cdot \left(l(a_i) - E(l)\right)^2$

  Einsortierung kann so vorgenommen werden, daß neue "vor" welchen mit gleicher Ws stehen. Dann ändert sich die Varianz, nicht der Erwartungswert
- **Informationsgehalt I eines Zeichens z:** $I_b(z) = \log_b\left(\frac{1}{p(z)}\right)$
  mit b als Maßeinheit der Information (Bit)
- **Entropie:** $H = -\sum_{z \in A} p(z) \cdot \log_b\left(p(z)\right)$
  Effizienz läßt sich über Redundanz messen., die die Differenz zwischen der mittleren Codewortlänge und der Entropie darstellt. "Maß für Informationsgehalt einer Meldung, bezogen auf eine Wahrscheinlichkeitsverteilung"

Fehlererkennung und –korrektur

- Minimierung der Redundanz; sind alle Bitkombinationen zulässige Codewörter, so können Fehler nicht erkannt werden.
- **Kanalkodierung:** Hinzufügen von Redundanz (¢Prüfstellen); Wahl der Prüfbits hängt von Störeigenschaften des Kanals ab
- **Ein–Fehler–prüfbare Codes:** Parität; jedes Codewort unterscheidet sich in mindestens 2 Binärstellen (¢Hamming–Distanz d=2); Relative Redundanz= $\frac{\#Anzahl\,\text{Pr}\ddot{u}fbits}{\#L\ddot{a}ngeCodewort}$
- **Korrigierbare Codes:** Fehler können ohne Rückfrage korrigiert werden

- **Blocksicherung:** Blockweises Übertragen von ein–Fehler–prüfbaren Codewörtern; am Ende wird ein Prüfwort angefügt; so kann die Fehlerstelle ausgemacht werden
- **Allgemeiner Ansatz:** in jedem einzelnen Codewort kann ein Fehler korrigiert werden; für jedes Codewort gilt: n=m+k mit m (#Informationstragenden Stellen) und k (#Prüfstellen).
  Hamming: mit k Stellen lassen sich 2^k Zustände beschreiben; bei der Codeprüfung müssen n+1 Zustände festgestellt werden können ¢ $2^k \geq m+k+1$
  Auch ein Ein–Fehler–korrigierbarer Code kann nicht für jedes beliebiges m mit gleicher Redundanz aufgestellt werden.
- **Ein Ein–Fehler–korrigierbarer Code:** Jede Stelle wird zweimal überprüft: dadurch wird gewährleistet, daß erkannt wird, ob ein Fehler in des Nutzdaten oder in den Paritätsbits vorliegt (geht auch mittels Vergleich aller zulässigen Codewörter)
- **Hammingdistanz:** Anzahl der Binärstellen, in denen sich zwei Codewörter unterscheiden (Mindestabstand ist dann die Hamming–Distanz des Codes)

- **Linearcode:** Codetabelle hat Gruppeneigenschaft ¢ LA; Konstruktion weniger linear unabhängiger Codewörter und Zusammenfassung in einer *Generatormatrix G* zur Berechnung der restlichen Codewörter: $G=[E_m P]$ (Einheitsmatrix mxm und Prüfmatrix mxk)
  Kontrollmatrix $H=[P^T E_k]$ (Transponierte Prüfmatrix und Einheitsmatrix kxk)
  Die Multiplikation eines Codewortes mit H^T ($s = g \cdot H^T$) sollte immer die Null–Matrix ergeben. Bei Fehler gibt es einen Vektor (Syndrom), der Hinweise auf Fehler gibt.
  Das falsche Codewort c' und das Fehlerwort e (mit $c' = c \oplus e$) haben dasselbe Syndrom.
- **Fehlerkorrigierende zyklische Codes:**
  Bei diesen Blockcodes entsteht bei zyklischen Vertauschen der Bits wieder ein zulässoges Codewort. Die empfangsseitige Fehlerkorrektur läßt sich mittels rückgekoppelter Schieberegister realisieren. Jedes Codewort läßt sich als Summe von Zeilen der Generatormatrix darstellen. Ist vollständig definiert durch Generatorpolynom g(x), so daß sich jedes Codewort duch g(x) ohne Rest teilen läßt.
  Bei Realisierung werden Bits um eins nach links geschoben. Wird eine 1 herausgeschoben, so wird das an den 1er–Stellen des Polynoms addiert.

Boolsche Funktionen und Schaltnetze

- **Antivalenz / XOR:** $x \oplus y = \overline{x}y + x\overline{y} = \overline{(x \leftrightarrow y)}$
- **Bijunktion / Äquivalenz:** $x \leftrightarrow y = xy + \overline{xy} = \overline{(x \oplus y)}$
- **Minterm:** Und–Klausel mit allen n Variablen
- **DNF:** Disjunktion von Konjunktionen
- **KNF:** Konjunktion von Disjunktionen
- **KDNF:** Disjunktion von Mintermen (eindeutige Darstellung boolscher Funktionen)
- **Kosten der Realisierung einer DNF:** #Literale + #Vorkommen des Oder–Symbols + 1
- **Regeln:**
  - **De Morgan:** $\overline{a \cdot b} = \overline{a} + \overline{b}$; $\overline{a + b} = \overline{a} \cdot \overline{b}$
  - **Absorption:** $a \cdot (a + b) = a$; $a + (a \cdot b)$
- **Kombinatorische Schaltnetze:** eine gegebene Eingangsbelegung bewirkt stets die gleiche Ausgangsbelegung
  - **Komparator:** Tabelle aufstellen und daraus DNF ableiten
  - **Halbaddierer:** $s = \overline{x_1} x_0 + x_1 \overline{x_0}$; $ü = x_1 \cdot x_0$

- **Volladierer:**

- **Kodewandler:** Entspricht de r Umwandlung anhand einer Tabelle
- **Sequenzielle Schaltungen:** "Die Vorgeschichte veranlaßt die Schaltung bzw. das System zum gegenwärtigen Zeitpunkt einen definierten Zustand einzunehmen" ¢ Rückkopplung
  Es kann zu zeitlichen Verzögerungen kommen. Das Verzögerungsglied τ stellt einen Speicher dar, der den Wert $q(t-\tau)$ für einer Zeit von τ aufbewahrt
  Synchrone Schaltungen: Zustände und Ausgaben ändern sich zu bestimmten Takten
  Asynchrone Schaltungen: kein Takt ¢ abhängig von Eingaben
  Formal wir ein Automat A durch (X, Z, f, a_z) beschrieben. Bei k möglichen Eingaben und n möglichen Zuständen ergibt sich eine Schaltung mit k+n Eingängen und n Ausgängen
- **Moore–Atomat:** $A=(X, Z, f, a_z, Y, g)$
  Grafische Aaaaaasdfdsa Darstellung mittels Knoten, in denen die Ausgabe vermerkt ist
  Automatentabelle: Spalten¢Eingabeelemente, Zeilen¢Zustandsmenge, Inhalt¢Folgezustand, Werte der Ausgabefunktion in zusätzlicher Spalte anordnen
- **Mealy–Automat:** $A=(X, Z, f, a_z, Y, h)$
  Grafische Darstellung indem die Ausgabe an die Kanten geschrieben wird (neben die Eingabewerte)
  In der Automatentabelle wird der Ausgabewert zu den Folgezuständen geschrieben
- **Flip–Flop:** Sequentielle Schaltung mit zwei stabilen Zuständen; besitzen Informationseingängen, die Typ spezifisieren; Komplementäre Ausgänge Q und $\overline{Q}$
- **D–Flip–Flop:** arbeitet getaktet; Verzögerungs–FF; Verzögerungszeit t_D ist die Differenz zwischen dem Moment des interessierenden "0–1"–Zustandwechsels des D–Signals am Informationseingang und seinem Erscheinen am Ausgang Q des DFF infolge des Wirksamwerdens des Taktsignals C
- **Arithmetische Operationen:**
  - **Schieberegister:** Realisierung z.B. über Automat
  - **Serien–Addierwerk:** Addiert zwei Dualzahlen mit einem Volladdierer, verwendet Schieberegister dafür; arbeitet getaktet
  - **Parallel–Addierwerk:** Additionen werden parallel ausgeführt ¢ Übertrag muß erst noch durchsickern
  - **Multiplikation:** $x = x_0...x_{n-1}$; $y = y_0...y_{n-1}$; $s_{i+1} = s_i + wert(x_{n-1}...x_0)\cdot y_i\cdot 2^i$
    Realisierung mit 1 Addierer, 2 Schieberegister
  - **Division:** Divisor solange vom Dividenden subtrahieren, bis Different negativ wird. Letze Subtraktion war dann zuvie ¢ Korrekturaddition; Rest
    Realisierung: 2er–Komplement, Addierer, Zähler, Komparator, 2 Register für Ergebnis

Maschinennahe Programmierung

Programme im Maschinencode können vom Prozessor ausgeführt werden. Ein Assembler verwendet mnemonische Codes als symbolische Bezeichner für Maschinenbefehle.

- **Mikrocode:** elementare Befehle des Prozessors, die intern durch "Mikroprogramme" ausgeführt werden, die einzelne Leitungen in den Ausführungseinheiten (ALU, Register) ansteuern. Alternativ können Prozessorbefehle auch als festverdrahtete Logik auf

Gatterebene ausgeführt werden (**random logic**); langsam aber leicht zu implementieren/erweitern und braucht wenig Platz

- **Befehlssatz:** Datentransportbefehle, Arithmetisch–logische Befehle, Programmsteuerbefehle, Systemsteuerbefehle
- **Architekturen:** CISC, RISC
- **Adreßregister:**
  - 7 x 32Bit breit
  - Dienen zum Adressieren von Operanden
  - Können nur mit Wort– und Langwortdaten arbeiten, nicht mit Bytes
  - Wortdaten werden beim Lesen vorzeichenrichtig zu Langwortdaten
  - Durch Operationen auf Datenregistern werden Flags gesetzt, nicht bei Adreßregistern.
  - Adressierung: direkt / indirekt
- **Condition Code Register (CCR):** 5 sog. "Flags": C–/Carry, V–/Overflow, Z–/Zero, N–/Negativ, X–/Extended (fast wie Carry Flag)
- **Stapel:** Stackpointer SP zeigt auf oberstes Element
- **Adreßregister indirekt mit Verschiebung:** Eignet sich gut für Listen, Tabellen mit Anfangsadresse und relativen Verschiebungen.: `MOVE.W 14(A0), D1` mit A0 als Startadresse
- **Adreßregister indirekt mit Index:** `MOVE.W  VALVE(A0, D0.W), D1`
- **Subroutinenaufrufe:** JSR, BSR sichern Rückehradresse auf Stapel; JSR hol Rückkehradressen vom Stapel und legt sie auf PC, RTR mach gleiches speichert aebr noch Bedingungscodes zurück
- **Parameterübergabe:** Direkt auf Stapel
- **Reentrant:** Subroutinen sind reentrant, wennsie nach Unterbrechung einfach wieder fortgeführt werden können.
- **Parameterübergabe an Unterprogramme:** Register / Stapel
- **Relokierbare Programme:** Programme können nach Assemblierung an beliebiger Stele im Speicher ausgeführt werden.
  - **PC–relative Adressierung:** Versatz von Programmierer/Assembler berechnet (LEA)
- **Supervisor– und Anwenderebene:** TRAP (Fehlerbehandlungsvektor)

Compilerbau

Überstzen eines Programms in einer Quellsprache A in ein äquivalentes Programm in einer Zielsprache B.

Analyse–Synthese Modell

Überetzung in 2 Hauptphasen:

- **Analyse:** Prüfung Syntax/Semantik, Ausgabe Fehlermeldungen, Erzeugung von Zwischencode, Codeoptimierung (FRONTEND)
- **Synthese:** Erzeugt Maschinencode, Optimierung auf Basis der Kenntnis des Zielcodes (BACKEND)
  - **Präprozessor:** Einbindung referenzierter Dateien, Ersetzung von Zeichenketten/Makros, Kommentarentferung etc.
  - **Assembler:** Übersetzung von Assemblercoe in Maschinencode
  - **Loader/Linker:** Zusammenbinden getrennt übersetzter Programme, Einbinden von Bibliotheken

- **Phasen der Übersetzung:**
  Source ¢ lexikalische Analyse ¢ Syntax–Analyse ¢ Semantische Analyse ¢ Zwischencode ¢ Optimierung ¢ Code
  - **Lexikalische Analyse:** Zerlegung der Eingabe in einzelne Tokens (Scannen), Speichern von Bezeichnern in einer Symboltabelle
  - **Syntax–Analyse:** Token werden zu korrekten Phrasen der Quellsprache zugeordnet; Syntax besitzt i.A. hierarchische Struktur, die durch (rekursive) Regeln beschrieben ist ¢Parse–Baum (Operatoren als interne knoten, Operanden als Blätter), Syntax–Baum (nur Terminalzeichen)
  - **Semantische Analyse:** Es werden Typinformationen gesammelt und auf semantische Fehler hin geprüft
  - **Zwischencode–Erzeugung:** Programm für abstrakte Maschine, einfach zu erzeugen einfach in Zielsprache zu übertragen, unabh. Von Quellsprache
- **Reguläre Ausdrücke:** Spezifizierung von Mustern, bestimmt Sprache L(r):
  rs, r|s, r*, (r); Abkürzung: [a–z], [abc], r^+, r*

Tokenerkennung

Regeln werden zum Beispiel regulär definiert und können in einem Transitions–/Übergangs oder Zustandsdiagramm dargestellt werden.

- **Übergangsdiagramm:**
  - Zustand als Kringel
  - Transition mit Beschriftung/Markierung
  - Doppelkreis als akzeptierender Endzustand
  - * zeigt an, daß eEingabe um ein Zeichen zurückgesetzt werden muß

Syntaxanalyse

- **3 Parser–Typen:** Cocke–Younger–Kasami, Earley's Alg. ¢ sehr ineffizient, deshalb:
- **Klasifikation:** Top–Down / Bottom–Up

Kontextfreie Grammatiken

- Viele Programmiersprachen haben inhärant rekursive Struktur ¢ können also durch kontextfreie Grammatik beschrieben werden.
- Besteht aus:
  - **Terminalen:** Kleinbuchstaben, Operatorsymbole, Ziffern, unterstrichene Folgen
  - **Nichtterminalen:** Großbuchstaben, Startsymbol S, kursive Namen
  - **Startsymbol S**
  - **Produktions–/Ersetzungregeln**
  - **Grammatische Symbole ¢ griechische Buchstaben**
- **Ableitung (derivation):** Produktion als Ersetzungsregel; => als Ableitung in einem Schrit – mit * oder + noch möglich
- Zwei Grammatiken heißen äquivalent, falls sie die gleiche Sprache erzeugen
- S =*> α, mit α **Satzform**, falls α Nichtterminale enthält; **Satz** ist Satzform ohne Nichtterminale
- Bei jedem Ableitungsschritt muß bestimmt werden, 1.) welches Nichtterminal ersetzt werden soll und 2.) durch was es ersetzt werden soll
- **Linksableitungen:** hier wird stets das am weitesten links stehende Nichttemrinal ersetzt

- **Ableitungs–/Parsebaum:** Innere Knoten bestehen nur aus Nichtterminalen; Blätter sind dann Terminale; Nicht jeder Satz hat notwendigerweise nur einen Parse–Baum, genau wie er auc nicht nur eine Links– oder eine Rechts–Ableitung hat.

Grammatiken

- Nicht jede Struktur läßt sich durch einen regulären Ausdruck spezifizieren (z.B. ineinander geschachtelte Klammerausdrücke)
- Mehrdeutigkeiten vermeiden!
- **Links–rekursiv:** Es gibt Ableitung der Art $A =^{+}> A\alpha$
- **Algorithmus zum Vermeiden von Links–Rekursionen:**
  Vors: G hat keine Zyklen und keine ε–Produktionen (A¢ε)
  1.) Ordne Nichtterminal in beliebiger Reihenfolge $A_1, ..., A_n$ an
  2.) for (i=1; i<=n; i++)
  for(j=1; j<= i–1; j++)
  ersetze jede Produktion der Form A_i¢$A_j\gamma$ durch
  A_i ¢ $\delta_1\gamma \mid \delta_2\gamma \mid ... \mid \delta_k\gamma$, mit
  A_j ¢ $\delta_1 \mid \delta_2 \mid ... \mid \delta_k$
  eliminiere diekte Links–Rekursionen in den A_i–Produktionen
- **Links–Faktorisierung:**
  Bei der Expandierung kann es sein, daß man nicht genau weiß, wie man expandieren soll. Man schreibt nun die Produktionen so um, daß die anzuwendende Produktion stets eindeutig ist: A¢$\alpha\beta_1 \mid \alpha\beta_2$ wird zu A¢αA' und A'¢$\beta_1 \mid \beta_2$
  Algorithmus zur Transformation einer Grammatik G in eine äquivalente links–faktorisierte Grammatik:
  - Bestimme für jedes Nichtterminal A das α, das längstes gemeinsames Präfix zweier oder mehrerer Alternativen in A ist
  - Wenn $\alpha \neq \varepsilon$, dann setze alle A–Produktionen
    A¢$\alpha\beta_1 \mid \alpha\beta_2 \mid ... \mid \alpha\beta_n \mid \gamma$, mit γ für alle nicht mit α beginnenden Alternativen, durch:
    A ¢ αA' | γ
    A' ¢ $\beta_1 \mid \beta_2 \mid ... \mid \beta_n$
  - Wiederhole Transformation solange, bis es für kein Nichtterminal mehr Alternativen mit gemeinsamem Präfix gibt

Top–Down–Syntaxanalyse

- Versuch, Linksableitung für eine Eingabe zu konstruieren = Parse–Baum für die Eingabe aufstellen mit Knoten in Präorder
  Grammatiken, die einer Top–Down–Syntaxanalyse zugrundeliegen dürfen nicht linksrekursiv sein.
- **Methoden:** Rekursiver Abstieg, Prädikative Parser (Vors. keine Linksrekursion, linksfaktorisiert)
- **Implementierung:** rekursive Prozeduren, Parse–Tabelle
- **Transitionsdiagramme:**
  Ein Diagramm pro Nichtterminal der Grammatik, Kanten sind mit Token (Transition gilt, falls das nächste Eingabesymbol dem Token entspricht) oder Nichtterminalen (Übergang in Startzustand des zu dem Nichtterminal gehörenden Transitionsdiagramms, falls akzeptierender Zustand erreicht wird¢ursprüngliche Transition) beschriftet

Bottom–Up–Syntaxanalyse

Ausgehend von einem Eingabestring soll ein Parsebaum konstruiert werden, wobei man allerdings bei den Blättern beginnt.

- Falls mehr als eine Regel angewendet werden kann, so wählen wir die Regel aus, deren rechte Seite den linkesten Substring des Ausgangswortes ersetzt.

Semantische Analyse

Verbindung von Informationen mit einem Programmiersprachenkonstrukt.
Wird realisiert durch Heften von Attributen an Grammatiksymbole. Dafür gibt es folgende Möglichkeiten:

- **Syntaxgesteuerte Definition**
  - Jedes Grammatiksymbol besitzt eine ihm zugehörige Attributmenge
  - 2 Attributmengen: synthetisierte (Attribut wird errechnet und ergibt sich aus den Attributen der Nachfolger) und ererbte Attribute (wird aus Wert der Geschwister und Vorgänger errechnet)
  - Semantikregeln erzeugen Abhängigkeiten zwischen Attributen
- **Übersetzungsschema**
  - Kontextfreie Grammatik, wo Attribute mit mit Grammatiksymbolen assoziiert werden
  - Reihenfolge der Auswertung der semantischen Regeln ist festgelegt
  - Semantische Aktionen werden zwischen den Klammern {} eingeschlossen und innerhalb der rechten Seite der Produktion eingefügt
- **Beispiele:** Typüberprüfung, Kontrollfluß, Eindeutigkeit,

Eine Sprache heißt **streng typisiert**, wenn Compiler nur Programme akzeptieren, die ohne Typfehler ausgeführt werden.

Code–Erzeugung

Verwendung von (Adreß– und Register–)Deskriptoren, die Anzeigen, was sich z.B. im Register befindet, oder wo sich ein Wert befindet (¢Symboltabelle)

- **Peehole–Optimierung:** versucht in kurzen Abschnitten den Code zu verbessern (z.B. durch Entfernen überflüssiger Anweisungen, Kontrollfluß–Optimierung, algebraische Vereinfachung, Ausnutzen von Eigenheiten der Maschine)

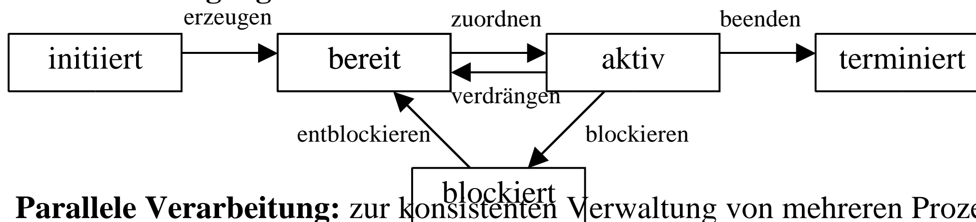
Betriebssysteme

"Programme eines digitalen Rechensystems, die zusammen mir den Eigenschaften dieser Rechenanlage die Basis der möglichen Betriebsarten des digitalen Rechensystems bilden und die insbesondere die Abwicklung von Programmen steuern und überwachen."

- **3 Aufgabenbereiche**:
  - Abbildung der Benutzerwelt auf die Maschinenwelt
  - Organisation und Koordination des Betriebsablaufs
  - Steuerung und Protokollierung des Programmablaufs
- "eine Sammlung von Programmen zur geregelten Verwaltung und Benutzung der Betriebsmittel" (Zuordnung und Betrieb)
- **Betriebsmittel:** Aktive Betriebsmittel (zeitlich aufteilbar), passive Betriebsmittel (exklusiv benutzt), Passive Betriebsmittel (räumlich aufteilbar)
- **Betriebsarten:** Batchbetrieb, Dialogbetrieb, Realzeitbetrieb, Multi–User–Betrieb
- **Virtuelle Maschinen:** Verdecken der gegebenen Maschine im Interesse einer komfortablen Benutzung und reibungslosen Betriebsabwicklung.

Prozessorverwaltung

- Wird übernommen durch **scheduler** und **dispatcher**
- Zuteilung des Prozessors wird durch Prioritäten gesteuert ¢ Zuteilungsverfahren
- Zeitspanne wird vom Scheduler berechnet neue Priorität aufgrund vom Verhalten des Prozesses während der letzten Zeitscheibe
- Vermeidung von Deadlocks, optimale Auslastung des Prozessors
- **Prozeß:** "zeitlich sich abspielender Vorgang, zeitlicher Ablauf einer Folge von Einzelereignissen in einem System: Zeitfunktion g:T¢M mit Zeitmenge T und Zustandsmenge M eines Systems" "der Vorgang einer algorithmisch ablaufenden Informationsverarbeitung" P=(S, f, s) mit Zustandsraum S, Aktionsfunktion f und Anfangszustände scS
- **Zustandsübergänge:**



- **Parallele Verarbeitung:** zur konsistenten Verwaltung von mehreren Prozessen in einem 1– oder n–Prozessor–System ist nötig:
  - Halten von Prozessen in Wartelisten bzgl. Betriebsmittelzuteilung
  - Retten von Zustandsinformation über Prozeß bei Umschaltzeitpunkt
  - **Prozeßleitblock:** Repräsentation eines Prozesses als manipulierbares Objekt mittels Attributen
  - **Zeitmultiplexbetrieb:** Verzahnung in der Zeit auch multiprogramming oder processor sharing
  - **Interrupts:** Unterbricht die Verarbeitung eines Prozesses
    - **Synchron:** vom Programm ausgelöst (auch durch Programmierfehler)
    - **Asynchron:** werden von externen Geräten, nebenläufigen Prozessen oder Maschinenfehlern ausgelöst
    - **Mechanismus:** Eintritt und Erkennung (in Hol–, Dekodier– und Ausführungsphase), Feststellung, Sichern des Zustandes, Start des Unterbrechungsprogramms, Beendigung des Unterbrechungsprogramms

Prozeßumschaltung

- Zwei Ziele: benutzerorientiert, systemorientiert
- Preemptive / non preemptive Multitasking
- **Strategien:** FIFO ¢ besser: Zeitscheibenverfahren, vielleicht noch mit Mehrschichtenstrategie, so daß noch Dringlichkeiten mit berücksichtigt werden
- **Leerlaufproblem:** wenn alle Prozesse im Zustand blockiert sind ¢ Lösung durch Leerlaufprozeß
- **Kontext eines Prozesses:** die im Prozeßleitblock enthaltene Information
- **Kooperation von Prozessen:**
  - Synchronisierung, Vermeidung von Deadlocks ¢ kritische Abschnitte (mutual exclusion)
  - Semaphorkonzept
  - Erzeuger–Verbraucher–Problem: Verbraucher kann nur agieren, wenn was da ist, Erzeuger kann nur erzeugen, wenn noch Platz da ist. (Puffer nicht voll)

Speicherverwaltung

"Speicher ist ein passives, räumlich aufteilbares Betriebsmittel. Teile können die Zustände frei und belegt annehmen. Speicherverwaltung hat als Aufgabe die Zuweisung und Überwachung des gesamten Speichers"

- Anwendung: Zentralspeicher, Peripheriespeicher, Anwenderprogramme
- Aufteilung des Speichers:
  - Konstante Einheiten der Länge e
  - Vielfache einer konstanten Einheit der Länge e
  - Variabel lange Stücke
- Darstellung der Speicherverwaltung: Vektor oder Tabelle
  - **Vektor:** Vergabe in konstanten Einheiten; diese bilden dann zusammen den Belegungsvektor
  - **Tabelle:** bei großer Anzahl an kleinen Einheiten, Konzentration auf Lage und Länge
  - Belegungsdarstellung kann aber auch in den verwalteten Stücken selbst untergebracht werden (¢ Verkettung; Freispeicherliste)
- Strategien zur Befriedigung von Speicheranforderungen: first fit, best fit, random fit
- Verschnitt: intern¢im Schnitt eine halbe Einheit, extern¢"Garbage" entsteht dadurch, daß kein genügend großes Stück mehr verfügbar ist
- Problem der Wiedereingliederung freier Speicherteile:
  - Vektor: First–Fit
  - Tabelle: **Boundary tag system**, Vergabe nach best–fit–Strategie; Speicherblöcke tragen in WF bzw. WB Verwaltungsinformationen über Länge und Zustand mit sich, dadurch wird das Verschmelzen freier Teile vereinfacht
  - **Buddy–System (Halbierungsverfahren):**
    Speicher der Länge 2^{Km}, Vergabe in Stücken zu 2^{Ki} Einheiten. Für alle Stückgrößen werden Freispeicherlisten FL_1, ..., FL_m unterhalten, die in Komponenten des Ankervektors A verankert sind. Zu Anfang ist nur die Liste FL_m mit höchster Potenz k_m enthalten.
    Wenn eine Belegungsanforderung auf eine leere Liste FL_i trifft, dann wird bei der Liste FL_{i+1} ein Stück entliehen. Diese wird halbiert, mit der einen Hälfte die Anforderung befriedigt und die andere Hälfte in die Liste FL_i aufgenommen. Das Ausleihen aknn sich über mehrere Listen hinweg bis zu FL_m fortsetzen (¢rekursiver Algorithmus).
    Beim freigeben des Speicherplatzes muß nun überprüft werden, ob der Nachbar (Buddy) vor dem Teilen auch frei ist, damit die Teile verschmolzen werden können.
    Gut: Schnell durch relative Adressierung mit Dualadressen.
    Die Adresse P eines Stückes läßt auf die Adresse Q des Partners schließen durch:
    $Q = P + 2^{Ki}$, wenn $P \bmod 2^{(Ki+1)} = 0$ bzw.
    $Q = P - 2^{Ki}$, wenn $P \bmod 2^{(Ki+1)} = 2^{Ki}$ (entspricht der Komplementierung der Bits)
    Zum Prüfen, ob bei Q wirklich der Partner von P ist, wird der Exponent, der die Länge beschreibt als Statusinformation mit abgespeichert.
    Schlecht: recht hoher rrecht hoher interner Verschnitt (es gehen im Schnitt 25% des Speichers verloren)
  - **Haldenverwaltung:** Verwaltung frei gewordenen Speichers (¢garbage collection: zusammengesetzte Freispeicherverwaltung / Speicherbereinigung)
    Strategien:
    - **explizite Freigabe des Speichers:** Anforderung an Programmiersprache, Halde ist anfänglich abgegrenzter Speicherbereich, der zu Beginn Freispeicherliste bildet.

Problem der losen Referenzen nach Speicherfreigabe, wenn noch Referenzen auf bereits gelöschte Objekte bestehen

- **automatische Speicherbereinigung mit Referenzzählern:** Für Objekt wird Zählvariable mitgeführt (+1 bei Zuweisung, –1 bei Löschen). Bei 0 wird Speicherberich freigegeben. Vorteil: bei geringer Anzahl von Referenten auf ein Objekt bleibt das System nicht plötzlich wg. Speicherbereinigung stehen
- **automatische Speicherbereinigung unter Kennzeichnung aller noch zugänglicher Objekte und Freigabe des restlichen Speichers:** Erlaubt dynamisches Wachsen der Halde und kann periodisch durchgeführt werden, auf jeden Fall aber, wenn Keller mit Halde zusammenstößt ¢ belegte Teile werden am Anfang zusammengelegt, freie zu einem großen Stück zusammengefaßt: Markieren noch zugänglicher Objekte, Adressierung verlegter Objekte, Freigabe

Virtueller Speicher

"Der Speicherbedarf eines Prozesses ist häufig größer als der real vorhandene; man möchte einen hohen Parallelitätsgrad erzielen"

- **Preplanned paging:** Nicht durchführbar, da im Prinzip der Programmablauf schon im vorneherein bekannt sein müßte.
- **Demand paging:** Weil Programmablauf datenabhängig, vor allem weil Trennung von betriebstechnischem und Problemlösung unabhängig. Laufende Programmausführung muß kurzzeitig unterbrochen werden.
- **Prinzip:** Zentralspeicher in Kacheln aufgeteilt; Seitentabelle (Seitenadresse und Wort– bzw. Byteadresse¢relative Adresse); Transformationstabelle muß nur Zuordnung von Programmseitennummer zu Speicherkachelnummer leisten.
- **Präsenzindikator:** anwesend / abwesend
- **Referenzindikator:** referiert (seit letztem Bezugspunkt mindestens einmal zugegriffen), sonst nichtreferiert
- **Modifikationsindikator:** modifiziert (seit letztem Ablegen in Kachel verändert), sonst nicht modifiziert
- **Seitenfehler:** Seite, die benötigt wird ist gerade ausgelagert¢Seitentausch
  Auswahl kann lokal (nur Anwendungsbezogen) oder global erfolgen
- **Strategien zur Seitenauswahl:** LFU, LRU, RNU; am besten mit "gleitendem Fenster"
- **Relatives Speicherangebot r:** r=m/s mit s Seiten für den Prozess und m verfügbaren Kacheln Ws(eine Seite muß beschafft werden)=(s–m)/s=1–r "Seitentauschgrad"
- **Working set:** bevorzugt referierte Teilmenge der Seiten eines Prozesses
- **Thrashing:** ständiges Auslagern
- **Zweistufige Adreßumsetzung:** im Gegensatz zu einstufiger günstig bei zweidimensionaler Adressierung und großen Segmenten; Segment(+Segmenttabellenbasisadresse)+Segmenttabelle+Seitentabelle+Wort; Damit nicht unnötig langsam werden die letzten 4,8,16 Zuordnungen in schnellen Assoziativregistern gehalten
- **Dreistufig:** Gliederung der Segmente in Blöcken

Dateiverwaltung

- Dateiorganisation:
  - **Sequentiell:** Ale Sätze in linearer Reihenfolge angeordnet (seq. Lesen, am Ende Hinzufügen, Positionieren am Anf/Ende, ersetzen des Satzes an akt. Position, Löschen von Sätzen ab aktueller Pos)

- **Direkt (indiziert):**
  Satznummernbasis: Datei als array; numerierte Plätze werden reserviert und können beliebig gefüllt oder geleert werden
  Schlüsselbasis: Datei als klassische Menge, Elemente werden durch Schlüssel identifiziert
  Direkte Organisation: Sätze in aufeinanderfolgenden Blöcken ablegen; Sätze müssen gleiche Länge haben. +Effizient, einfach, schnell (¢Datenbank), –Indexlücken¢Speicherverschnitt, Benutzer muß zusätzlich gewünschte Dateiorganisation programmieren
- **Indexsequentiell:** Alle Sätze nach Schlüssel sortiert angeordnet; Zugriff sequentiell (lesen seq., hinzufügen am Ende eines Satzes oder an entspr. Pos, Positionieren am Anf/Ende oder an durch Schlüssel best. Pos, löschen/ersetzen eines Satzes)
  f: I¢S mit S: Menge der Sätze, I: Menge der Schlüssel in O(log|S|)
  + viele Operatoren¢flexibler Umgang mit Dateien; 2 Organisationskriterien perfekt vereinigt; günstig für Sätze mit Schlüssel–Lücken
  – Aufwendige Realisierung; Maschinenbelastung oft unterschätzt
  Realisierung oft durch B*–Baum mit Ordnung n
- **Swapping:** Prozesse werden ein–/ausgelagert
- **Paging:** Kacheln werden ein–/ausgelagert

Prüfungsfragen

- **Was ist ein Algorithmus?**
  Ein Algorithmus ist eine mit formalen Mitteln beschreibbare, mechanisch nachvollziehbare Verarbeitungsvorschrift zur Lösung einer *Klasse* von Problemen. Eine *exakte (formale) Beschreibung* ist notwendig, um Lösungsverfahren für die unterschiedlichen Probleme so zu formulieren, daß ihre Bearbeitung in Form eines Programms von einem Rechner übernommen werden kann: Sprache zur Abfassung von Algorithmen.
  Hat drei Eigenschaften:
  1. *Endliches* Verfahren zur Lösung von Problemen
  2. Sprachlich *präzis und eindeutig formuliert*
  3. Erfordern geeignete Methoden zur *Strukturierung* der von Algorithmen manipulierten Daten und Datenstrukturen
- **Was ist Korrektheit?**
  Funktionale Korrektheit: Ein A. ist korrekt, wenn er den Anforderungen, die an ihn gestellt sind, entspricht (interne Spez. ¢ Ausgaben demgemäß)
  partielle Korrektheit (ein partiell korrekter A. muß nicht unbedingt enden oder ein Ergebnis erzeugen, wenn er aber eines erzeugt, so ist dieses auch korrekt)
  Totale Korrektheit (Aus partieller Korrektheit und der Terminierung folgt die totale Korrektheit)
- **Was ist ein Abstrakter Datentyp?**
  Unter einem Datentyp versteht an die Zusammenfassung von Wertebereichen und Operationen zu einer Einheit. Liegt der Schwerpunkt auf den Eigenschaften, die die Operationen und Wertebereiche besitzen, so spricht man von einem ADT.
  Syntax: Beschreibung der Funtkionen (Datentyp¢Datentyp)
  Axiome: Regeln, die Abhängigkeiten der Funktionen beschreiben
- **Buddy–Systeme!**
  Speicher der Länge 2^{Km}, Vergabe in Stücken zu 2^{Ki} Einheiten. Für alle Stückgrößen werden Freispeicherlisten FL_1, ..., FL_m unterhalten, die in Komponenten des Ankervektors A verankert sind. Zu Anfang ist nur die Liste FL_m mit höchster Potenz k_m enthalten.
  Wenn eine Belegungsanforderung auf eine leere Liste FL_i trifft, dann wird bei der Liste FL_{i+1} ein Stück entliehen. Diese wird halbiert, mit der einen Hälfte die Anforderung befriedigt und die andere Hälfte in die Liste FL_i aufgenommen. Das Ausleihen aknn sich über mehrere Listen hinweg bis zu FL_m fortsetzen (¢rekursiver Algorithmus).
  Beim freigeben des Speicherplatzes muß nun überprüft werden, ob der Nachbar (Buddy) vor dem Teilen auch frei ist, damit die Teile verschmolzen werden können.
  Gut: Schnell durch relative Adressierung mit Dualadressen.
  Die Adresse P eines Stückes läßt auf die Adresse Q des Partners schließen durch:
  $Q = P + 2^{Ki}$, wenn $P \bmod 2^{(Ki+1)} = 0$ bzw.
  $Q = P - 2^{Ki}$, wenn $P \bmod 2^{(Ki+1)} = 2^{Ki}$ (entspricht der Komplementierung der Bits)
  Zum Prüfen, ob bei Q wirklich der Partner von P ist, wird der Exponent, der die Länge beschreibt als Statusinformation mit abgespeichert.
  Schlecht: recht hoher rrecht hoher interner Verschnitt (es gehen im Schnitt 25% des Speichers verloren)
- **Wann ist ein Algorithmus effizient?**
  Wenn er die Betriebsmittel bestmöglichst ausnutzt.
- **Prozeßverwaltung mit Semaphoren (¢kritischer Abschnitt)!**
  Variable empty, full, mutex
- **Was ist syntaktisches Testen?**
  ???

- **Was ist ein Syntaxdiagramm?**
  Technik zur graphischen Darstellung kontextfreier Grammatiken. Jedes Nichtterminalsymbol bekommt einen gerichteten Graph mit einer zusätzlichen Eingangs– und Ausgangskante zugeordnet. Knoten repräsentieren Grammatiksymbole mit Terminalsymbolen als Kreise und Nichtterminalen als Rechtecke.
- **Was ist ein Syntaxbaum?**
  Hilfsmittel zur Darstellung der Syntax einer Sprache – eine allgemeinere interne Darstellung des Parse–Baumes mit Operatoren als innere Knoten und Operanden als Blätter.
- **Was ist Korrektheit?**
  Die Korrektheit eines Programmes hängt von der Spezifikation also der Anforderung an das Programm ab. Erfüllt es diese (interne Spezifikation als Ableitung von der externen Spezifikation), so ist es auch korrekt.
- **Modus Ponens (Anwendung und Beweis):**
  ???
- **Schleifeninvariante:**
  Prädikat, das wahr ist:
  - Unmittelbar vor Eintritt in die Schleife
  - Unmittelbar nach Eintritt in die Schleife
  - Nach jedem Iterationsschritt
  - Unmittelbar nach Verlassen der Schleife

  Sie kann während der Ausführung von Anweisungen im Schleifenrumpf nicht gelten.
- **Laufzeit bei rekursiven Programme?**
  $$t(n) = \begin{cases} O(n) & a < c \\ O(n \cdot lob_c n) & a = c \\ O\left(n^{\log_c a}\right) & a > c \end{cases}$$
- **Dynamische / Statische Zeiger?**
  ???
- **Phasen eines Compilers:**
  Source ¢ lexikalische Analyse ¢ Syntax–Analyse ¢ Semantische Analyse ¢ Zwischencode ¢ Optimierung ¢ Code
- **Elimination von Linksrekursionen:**
  s.o.
- **Semaphoren (P– und V–Operationen):**
  Mit der P–Operation wird in einen kritischen Abschnitt gegangen und mit der V–Operation selbiger wieder verlassen.
- **Hoare – while Schleife**
- **Quicksort:**
  O(nlogn); rekursiv
- **Hashing:**
  Speicher– und Suchverfahren. Sehr schnell, da Hash–Funktion einfach berechenbar. Berechnet Adressen über Schlüssel der Datensätze. Besser als B*–Baum, wenn es nur um Einfügeoperationen geht
- **Index–sequentielle Dateiverwaltung:**
  Alle Sätze nach Schlüssel sortiert angeordnet; Zugriff sequentiell (lesen seq., hinzufügen am Ende eines Satzes oder an entspr. Pos, Positionieren am Anf/Ende oder an durch Schlüssel best. Pos, löschen/ersetzen eines Satzes)
  f: I¢S mit S: Menge der Sätze, I: Menge der Schlüssel in O(log|S|)

+ viele Operatoren¢flexibler Umgang mit Dateien; 2 Organisationskriterien perfekt vereinigt; günstig für Sätze mit Schlüssel−Lücken
− Aufwendige Realisierung; Maschinenbelastung oft unterschätzt
Realisierung oft durch B*−Baum mit Ordnung n

- **Prädikatenlogik:**
  Aussagenlogik baut auf atomaren Aussagen auf, die als Ganzes wahr oder falsch sind. Die Prädikatenlogik differenziert diese Aussagen weiter (Konstanten, Variablen, Funktionen, Prädikate, Operatoren, Quantoren, Terme, Normalformen
- **4 Bänder−Sortieren:**
  Mergesort ??
- **Laufzeitstapel:**
  ???
- **Haldenverwaltung:**
  ???
- **Virtueller Speicher:**
  LRU, LFU, working−set, Konzept
- **Mehrprozeßsystem:**

- **Aussagenlogik:**
  Untersuchung einfacher Verknüpfungen zwischen atomaren sprachlichen Gebilden
- **Was ist beim Testen einer while−Schleife besonders zu beachten?**
  Die Schleifeninvariante
- **Prinzip der virtuellen Adressierung:**
  Kacheln?
- **Resolutionsregel**
  (avb) n (!avc) ¢ bvc