







**omatische Semantik:**  
 Beschreibung eines Programms durch Angabe einer Vor- und einer Nachbedingung.  
 $\{A\} c \{B\}$   
 A: Vorbedingung  
 c  $\in$  Cmd  
 B: Nachbedingung  
**Bedeutung** von  $\{A\} c \{B\}$ :  
 Wenn vor Ausführung von c die Bedingung A erfüllt ist und c terminiert, dann ist nach Ausführung von c die Bedingung B erfüllt.  
**partielle Korrektheitsaussage:**  $\{A\} c \{B\}$  ist eine partielle Korrektheitsaussage da wir nur eine Aussage darüber abgeben ob c terminiert  
**totale Korrektheitsaussage:**  $\{A\} c \{B\}$  ist eine totale Korrektheitsaussage mit der Bedeutung: Wenn A erfüllt ist und c ausgeführt wird, dann terminiert c und anschließend ist B erfüllt.

Aexp	Arithmetische Ausdrücke	Assn	Zusicherungen (assertions)
$x := n \mid X \mid i \mid (a_0 + a_1) \mid (a_0 - a_1) \mid (a_0 * a_1)$	$A ::= \text{true} \mid \text{false} \mid a_1 = a_2 \mid a_1 < a_2 \mid a_1 > a_2$	$A ::= \text{true} \mid \text{false} \mid a_1 = a_2 \mid a_1 < a_2 \mid a_1 > a_2$	$A ::= \text{true} \mid \text{false} \mid a_1 = a_2 \mid a_1 < a_2 \mid a_1 > a_2$
$n \in \mathbb{N}, X \in \text{Loc}, i \in \text{Intvar}$	$A ::= a_1 \leq a_2 \mid \neg A \mid A_1 \wedge A_2 \mid A_1 \vee A_2$	$A ::= a_1 \leq a_2 \mid \neg A \mid A_1 \wedge A_2 \mid A_1 \vee A_2$	$A ::= a_1 \leq a_2 \mid \neg A \mid A_1 \wedge A_2 \mid A_1 \vee A_2$
Invar ist eine Menge von Integer- Variablen z.B. $\text{Intvar} = \{i_1, i_2, i_3, \dots\}$	$A ::= A_1 \wedge A_2 \mid \forall A \mid \exists A \mid A \neg A$	$A ::= A_1 \wedge A_2 \mid \forall A \mid \exists A \mid A \neg A$	$A ::= A_1 \wedge A_2 \mid \forall A \mid \exists A \mid A \neg A$

**Die Menge der Variablen:**  $V = \text{Loc} \cup \text{Intvar}$   
 Integer-Variablen können frei oder Gebunden vorkommen.  
**Auswertung von Zusicherungen:**  
 $A \in \text{Assn}, s \in \Sigma$  und es gelte  $FV(A) = \emptyset$ . Dann kann man A unter s auswerten.  
 $s(A) = \text{IB}$

bei freien Variablen:  
 diese müssen erst interpretiert werden, d.h. sie bekommen eine natürliche Zahl als Wert zugeordnet. (kurz:  $I: \text{Intvar} \rightarrow \mathbb{N}$ )  
 Schreibweise:  
 $s \models A$  wenn  $s(I(A)) = \text{true}$   
 und  $s \models A$  wenn es für alle Interpretationen gilt

**Beispiel:**  $g \in A \in \text{Assn}$  so dass  $s \models A \leftrightarrow (I(i) \text{ ist Primzahl})$   
 Lösung:  
 $A \equiv \forall j: ((j \leq 1) \vee (j \geq i) \vee (\forall k: j * k \text{ ungleich } i))$   
 ist nämlich  $I(i) = n \in \mathbb{N}$  dann gilt:  
 $I(A) \equiv \forall j: ((j \leq 1) \vee (j \geq n) \vee (\forall k: j * k \text{ ungleich } n))$

**Regeln von Hoare:** mit Hoare Logik kann wir beweisen werden, dass ein Programm korrekt arbeitet wenn es terminiert, nicht das es terminiert (partielle Korrektheit)  
 1. Hoare Tripel:  $\{P\} S \{Q\}$  (stellt Booleschen Wert dar)  
 Beispiele:  
 $\{0 < i\} i := i - 1 \{0 \leq i\}$  (Der gesamte Ausdruck ist wahr, wenn P vor und Q nach der Ausführung gilt.)  
 $\{\text{true}\} i := 1 \{i = 1\}$  ist gültig  
 $\{\text{true}\} i := 1 \{i = 2\}$  ist nicht gültig

2. Zeitgleiche Zuweisungen:  $x, y, z := 2 * y, x + y, 3 * z$   
 $x, y := y, x$  vertauscht die Werte der Variablen x und y  $x, x := 1, 0$  ist verboten, da linke Seite muss paarweise disjunkt sein

3. Zuweisungs Axiom  
 $\{Q[x/e]\} x := e \{Q\}$   
 heißt so viel wie nimm die Anweisung, ersetze in der Anweisung den mit der Precondition übereinstimmenden Teil, und erhalte die Postcondition...  
 Beispiel:  
 $\{x = 23 * i - 2\} f := x + 2 \{f = 23 * i\}$   
 3.2 erweitertes Zuweisungsaxiom  
 Beispiel:  $\{i+j=C\} i := i+1, j-1 \{i+j=C\}$   
 ist eine Zusicherung die für simultane Zuweisungen gilt

4. Sequenzen  

$\{P\} S1 \{Q\}$ Beispiel: $\{x=1\} y:=3 \{x=1, y=3\}$	$\{Q\} S2 \{R\}$ $\{x=1, y=3\} z:=x+y \{z=4\}$
$\{P\} S1, S2 \{R\}$ $\{x=1\} y:=3, z:=x+y \{z=4\}$	

 5. IF - THEN - ELSE  

$\{P \wedge B\} S1$	$\{Q\}$ Beispiel:
$\{P \wedge \neg B\} S2$	$\{Q\}$
$\{P\}$ if B then S1 else S2	$\{Q\}$

 6. IF-THEN  

$\{P \wedge B\} S$	$\{Q\}$ Beispiel:
$\{P \wedge \neg B\} (P \wedge B) \rightarrow Q$	$\{Q\}$
$\{P\}$ if B then S1 else S2	$\{Q\}$

 7. WHILE-Schleife  

$\{P \wedge B\} S$	$\{P\}$ Beispiel:
$\{P\}$ while B do S	$\{P \wedge \neg B\}$

 8. Sequenz  

$\vdash (A \rightarrow A) \wedge (A \wedge B) \rightarrow (A \wedge B)$
---

Wir schreiben:  $\vdash \{A\} c \{B\}$  falls sich die Aussage  $\{A\} c \{B\}$  mit der Hoare-Logik ableiten lässt.  
**Beispiel:**  $w = \text{while true do skip od}$   
 damit gilt:  $\vdash \{\text{true}\} w \{\text{false}\}$   
 noch eins:  
 denn  $\vdash \{X=3\} X := 8 + X \{X = 11\}$   
 $\vdash \{(X=1)[8 + X/X]\} X := 8 + X \{X = 11\}$   
 also  $\vdash \{8 + X = 11\} X := 8 + X \{X = 11\}$   
 mit Konsequenz  $\vdash \{X=3\} X := 8 + X \{X = 11\}$   
**mehr Beispiel:**  $g \in \text{wp while } ! = x \text{ do } y := y + 1 \text{ od } \{y = x\}$

Um eine Vorbedingung zu finden kann man eine Methode ähnlich der Ermittlung der induktiven Hypothese in der mathematischen Induktion benutzen: Es wird die Bedingung für ein paar Fälle berechnet, mit der Hoffnung, dass man ein Muster erkennt.

**O-Notation:**  $f \in O(g)$  Pizza = komplett Ananas  
 $f \in O(g) \iff \exists c > 0 \exists x_0 \forall x > x_0: |f(x)| \leq c \cdot |g(x)|$  irgendwo auf Pizza inkl. Rand  
 $f \in o(g) \iff \forall c > 0 \exists x_0 \forall x > x_0: |f(x)| < c \cdot |g(x)|$  irgendwo auf Pizza ohne Rand  
 $f \in \Omega(g) \iff \exists c > 0 \exists x_0 \forall x > x_0: |f(x)| \geq c \cdot |g(x)|$  nirgends auf Pizza außer Rand  
 $f \in \omega(g) \iff \forall c > 0 \exists x_0 \forall x > x_0: |f(x)| > c \cdot |g(x)|$  nirg auf Pizza nichma auf Rand  
 $f \in \Theta(g) \iff \exists c_0 > 0 \exists c_1 > 0 \exists x_0 \forall x > x_0: c_0 \cdot |g(x)| \leq |f(x)| \leq c_1 \cdot |g(x)|$   
**Komplexitätstheorie:** Das Nichtdet  
**DSPACE** liefert eine grundsätzliche Aussage darüber, welchen Speicherbedarf ein Rechenverfahren auf einem idealisierten Modell eines Computers beansprucht.  
 Wenn der Speicherbedarf eines Rechenverfahrens in linearer Proportion mit der Länge des Eingabeworts wächst, so gehört das Verfahren zu  $DSPACE(n)$ .  
 Speicherbedarf wächst exponentiell  $\rightarrow DSPACE(exp(n))$ .

$H_1(s := s/2, s = 1)$	$= Q[s/2/s]$	$= (s/2 = 1)$	$= (s = 1)$
$H_2(s := s/2, s = 2 \text{ or } s = 3)$	$= Q[s/2/s]$	$= ((s/2 = 2) \text{ or } (s/2 = 3))$	$= ((s = 2) \text{ or } (s = 5) \text{ or } (s = 6) \text{ or } (s = 7))$
$H_3(s := s/2, (s = 4) \text{ or } (s = 5) \text{ or } (s = 6) \text{ or } (s = 7))$	$= Q[s/2/s]$	$= (s/2 = 4)$	$= ((s = 8) \text{ or } (s = 9) \text{ or } \dots \text{ or } (s = 15))$

Daraus kann gefolgert werden, dass  $H_i = (s >= 2^i)$  und  $(s < 2^{i+1})$   
 Im Vergleich zum ersten Schleifen-Beispiel hat man mit dieser Methode nicht die schwächste Vorbedingung gefunden. Die Vorbedingung  $(s > 1)$  ist z.B. wesentlich schwächer.

$\{Y < 20; X > 5\} Y := Y - X \{Y < 18\}$   
 $\{Y = 100 \wedge X = 8\} Y := Y - X \{Y = 92\}$   
 1.  $\{X > 18; Y - X/Y\} Y := Y - X \{Y < 18\}$   
 2.  $\{Y - X < 18\} Y := Y - X \{Y < 18\}$   
 3. da  $\{Y < 20 \wedge X > 5\} \Rightarrow \{Y - X < 18\}$  gültig können wir sagen:  
 $\{Y < 20; X > 5\} Y := Y - X \{Y < 18\}$   
 1. Zuweisungsregel  
 2. Einsetzen in die Zusicherung  
 3. Allgemeingültige Implikation mit Konsequenzregel

$\{X > 0\} X := X + 3 \{X > 2\}$   
 $\{X = 3\} X := X + 4 \{X = 4\}$   
 Diese Aussage ist nicht gültig, denn genau auf den Speicherbelegungen s mit  $s(X) = 3$  ist sie nicht erfüllt.  
 Nach Ausführung der Anweisung  $X := X + 4$  hat nämlich X den Wert 7, d.h. die Aussage  $X = 4$  ist nicht erfüllt.  
 Aufgrund der Korrektheit des Hoare-Kalküls kann die Aussage also nicht beweisbar sein (da ja nur gültige Aussagen beweisbar sind)

**Gödelscher Unvollständigkeitssatz:** Jedes hinreichend mächtige formale System ist entweder widersprüchlich oder unvollständig. & Ein System kann nicht zum Beweis seiner eigenen Widerspruchsfreiheit verwendet werden.  
 $\{(A, B, C) \vdash \{A\} c \{B\}\}$  ist nicht rekursiv aufzählbar.  
 Begründung:  $\{\text{true}\} c \{\text{false}\}$  ist eine Wahre Aussage wenn c auf allen Speicherbelegungen nicht terminiert. Könnte man dies aufzählen, so wäre das Halteproblem entscheidbar.

**Das Hoare-Kalkül unterliegt der Tatsache, dass wir keine formale Überprüfbarkeit der Gültigkeit von Aussagen  $\{A\} c \{B\}$  hierdurch bekommen, deshalb spricht man von relativer Vollständigkeit.**

**Transformationen:**  
 1. **Whileprogramm in Gotoprogramm**  
 M1: wenn Bedingung nicht erfüllt gehe zu Endmarker  
 M2: führe das was in der Schleife steht aus  
 ME: Ende  
 WHILE  $x < 0$  DO P OD  
 M1: If  $x < 0$  THEN GOTO M4  
 M2: P  
 M3: GOTO M1  
 M4: ...

2. **Gotoprogramm in Whileprogramm**  
 M1: A1; M2: A2; ... Mk: Ak  
 geht mit einer While-Schleife die die Marken kodiert  $\rightarrow \text{count} = 1$   
 Es gilt zusätzlich:  
 $A_i' = \begin{cases} x_i = x_i + c; \text{count} = \text{count} + 1 \text{ wenn } A_i: x_j = x_j + c \\ \text{count} = n \text{ wenn } A_i: \text{GOTO } M_n \\ \text{Anw}_{\text{alt}} \text{ wenn } A_i: \text{Anw}_{\text{alt}} \\ \text{count} = 0 \text{ wenn } A_i: \text{HALT} \end{cases}$   
 $\text{Anw}_{\text{alt}} = \text{IF } x_j = c \text{ THEN } \text{count} = n \text{ ELSE } \text{count} = \text{count} + 1$   
 $\text{Anw}_{\text{neu}} = \text{IF } x_j = c \text{ THEN GOTO } M_n$

3. **Whileprogramm in Turingmaschine**  
 4. **Turingmaschine in Gotoprogramm**  
 Kodierung der Zustände in 3 Variablen  
 Variable 1: Zustand  
 Variable 2: Inhalt des Bandes rechts vom/unter dem Lesekopf  
 Variable 3: Inhalt des Bandes links vom Lesekopf

5.  **$\mu$ -rekursiv in Whileprogramm**  
 Sei  $f(x)$  eine Funktion, welche ein WHILE-Programm  $y := f(x), \dots, xn$ ; berechnet.  
 Ein Programm für  $\mu f$  ist gesucht.  
 $x_0 := 0$ ;  
 $y := f(x_0, \dots, xn)$ ;  
 END

6. **Loop in Whileprogramm**  
 LOOP x DO P END  
 $\rightarrow y := x$  WHILE  $y != 0$  DO  $y := y - 1$ ; P END

**DSPACE(f)** oder auch **SPACE(f)** steht für die Menge der Raumkomplexitätsklassen in Bezug auf eine deterministische Turingmaschine.  
 Wird eine konkrete Funktion f angegeben, so ist die Klasse derjenigen Entscheidungsprobleme gemeint, die auf einer deterministischen Turingmaschine mit O(f) Speicherplatz lösbar sind.  
 Anstatt O(f) schreibt man dies oft ohne die Konstante also als  $DSPACE(f) = DSPACE(O(f))$   
**NSPACE** steht für die Platzkomplexitätsklasse der Entscheidungsprobleme, die von einer Nichtdeterministischen Turingmaschine gelöst werden können.  
**PSPACE**  $PSPACE = \bigcup_{k \geq 1} DSPACE(n^k)$   
 Klasse der Entscheidungsprobleme, die von einer deterministischen Turingmaschine mit polynomiellen Platz entschieden werden können.  
**PSPACE-Vollständig** Sind Probleme in PSPACE, auf die sich ALLE anderen PSPACE-Probleme in Polynomzeit reduzieren lassen.  
 Es wird angenommen, dass die Klasse der vollständigen Probleme nicht in NP liegen.  
 Bsp.: QBF (quantified boolean formula) die Auswertung quantifizierter aussagenlogischer Formeln.  
 Kann ein gegebenes Wort von einer gegebenen kontextsensitiven Grammatik erzeugt werden?  
**NPSPACE** Klasse der auf polynomiellen Platz von einer nichtdeterministischen Turingmaschine entscheidbaren Probleme.  
**DTIME** oder auch **TIME** die Menge der Zeitkomplexitätsklassen in Bezug auf eine deterministische Turingmaschine.  
 Wird eine konkrete Funktion f angegeben, so bedeutet dies: DTIME(f) ist die Klasse derjenigen Entscheidungsprobleme, die auf einer deterministischen Turingmaschine in O(f) Zeit lösbar sind.  
**EXPTIME** oder auch **EXP** Komplexitätsklasse der Entscheidungsprobleme die von einer deterministischen Turingmaschine in durch  $O(2^{p(n)})$  beschränkter Zeit entschieden werden können.  $p(n)$  ist dabei ein beliebiges Polynom von der Eingabelänge n.  
 $EXPTIME = \bigcup_{k \in \mathbb{N}} DTIME((2^n)^k)$   
 $PSPACE \subseteq EXPTIME$   
**P** in Polynomzeit für deterministische Turingmaschinen lösbare Probleme. Diese Problemklasse wird allgemein als die Klasse der "praktisch lösbaren" Probleme betrachtet.  
 $P = \bigcup_{k \geq 1} DTIME(n^k)$   
**NP** Probleme für die in polynomialer Zeit nichtdeterministisch positive Antworten produziert werden können.  
 oder anders:  
 die Menge aller Probleme, bei denen es mit einer deterministischen Turingmaschine in Polynomzeit möglich ist, zu verifizieren, ob eine geratene Lösung tatsächlich eine Lösung der Aufgabenstellung darstellt  
**Q** nur linearer Zeitbedarf auf nichtd. M. (Quasi Realzeitsprachen)  
 $Q \subseteq NP$   
**E** Klasse der Sprachen, die sich von einer deterministischen Turingmaschine in exponentieller Zeit mit linearem Exponenten lösen lassen.  
 $PSPACE \neq E$   
**L** einer deterministischen Turingmaschine mit logarithmischem Platzverbrauch  
 $L = DSPACE(\log n)$   
**NL** einer nichtdeterministischen Turingmaschine mit logarithmischem Platzverbrauch  
 $NL = NSPACE(\log n)$   
**Co-** Beispiel an NP: In ihr sind genau die Sprachen enthalten, deren Komplement zu NP gehören.  $CoNP := \{L \mid L^c \in NP\}$   
 Intuitiv gesprochen besteht Co-NP aus der Klasse der Sprachen, in denen der Beweis, dass ein Wort nicht zur Sprache gehört, nichtdeterministisch polynomiel ist.  
 Ob NP = CoNP ist nicht geklärt.  
 P ist sowohl Teilmenge von NP als auch Co-NP

$L = DSPACE(\log n) = \bigcup_{e \in O(\log(n))} DSPACE(f)$   $P = \bigcup_{k \geq 1} DTIME(n^k)$   
 $NL = NSPACE(\log n) = \bigcup_{e \in O(\log(n))} NSPACE(f)$   $NP = \bigcup_{k \geq 1} NTIME(n^k)$   
 $PSPACE = \bigcup_{k \geq 1} DSPACE(n^k) = \bigcup_{k \geq 1} NSPACE(n^k)$  (PSPACE ist wohldefiniert)  
 $DTIME(t(n)) \subseteq DSPACE(x(n))$   $NPSPACE = \bigcup_{e \in O(n^k)} NSPACE(f)$   $k \in \mathbb{N}$

**Satz von Savitch:** Ein von einer nichtdeterministischen Turingmaschine mit einer bestimmten Platzkomplexität lösbares Problem kann auf einer deterministischen Turingmaschine mit einer quadratisch höheren Platzschränke gelöst werden.

**NSPACE(s(n))  $\subseteq$  DSPACE((s(n))^2)**  
**Zeitkomplexität** (auch asymptotische Laufzeit): die Anzahl der Rechenschritte, die ein Algorithmus zur Lösung dieses Problems benötigt, in Abhängigkeit von der Länge der Eingabe.  
**Platzkomplexität...** eines Problems ist der (minimalen) Bedarf an Speicherplatz eines Algorithmus zur Lösung dieses Problems, in Abhängigkeit von der Länge der Eingabe.  
**Zeitreduktionssatz:** 1.  $NTIME(O(f)) = NTIME(f)$   
 2. Sei  $\epsilon > 0$  &  $\forall n: f(n) \geq (1 + \epsilon)n$ , dann gilt:  
 $DTIME(O(f)) = DTIME(f)$

**Bandreduktionssatz:**  
 $DSPACE(O(f)) = DSPACE(E_{1-Band}(f))$   
 $NSPACE(O(f)) = NSPACE(E_{1-Band}(f))$   
 $DTIME(f) \subseteq DTIME(E_{1-Band}(f))$   
**Satz von Hennie-Stearns:**  
 Sei  $\epsilon > 0, k \geq 1$  und  $n \forall n: f(n) \geq (1 + \epsilon)n$ , dann gilt:  
 $DTIME_{k-Band}(f) \subseteq DTIME_{2-Band}(f \cdot \log(f))$

**Zeithierarchiesatz:**  
 Seien  $t_1, t_2: \mathbb{N} \rightarrow \mathbb{N}$  Funktionen,  $t_1 \cdot \log(t_1) \notin \Omega(t_2)$ ,  $t_2 \in \Omega(n \log n)$  und  $t_2$  zeitkonstruierbar. Dann gilt:  
 $DTIME(t_1) \not\subseteq DTIME(t_2)$   
**Platzhierarchiesatz:**  
 Seien  $s_1$  und  $s_2: \mathbb{N} \rightarrow \mathbb{N}$  Funktionen,  $s_1 \notin \Omega(s_2)$  (bedeutet:  $\forall \epsilon > 0$  unendlich viele n mit  $s_1(n) < \epsilon \cdot s_2(n)$ ),  $s_2 \in \Omega(\log n)$  und sei  $s_2$  platzkonstruierbar. Dann gilt:  
 $DSPACE(s_2) \not\subseteq DSPACE(s_1) \not\subseteq \emptyset$

**Sonstige geltende Gesetzmäßigkeiten:**  
 $PSPACE = IP \rightarrow NP \rightarrow NP \cap coNP \rightarrow P \rightarrow NL \rightarrow L$   
 $PSPACE = IP \neq NSPACE(n) = coNSPACE(n) \rightarrow (1.LBA-Problem) \rightarrow DSPACE(n) \neq \bigcup_{k \geq 1} DSPACE(\log^k(n)) = \bigcup_{k \geq 1} NSPACE(\log^k(n)) \neq NL \rightarrow L$   
 $L \neq PSPACE$  (Platzhierarchiesatz)  
 $DSPACE(n) \neq P$   $DSPACE(n) \neq NP$  (beides Abschlusseigenschaft)  
 $DTIME(n) \neq DTIME(O(n))$   
 (Zeitred.-satz, Linearzeit nicht auf Realzeit komprimierbar)  
 $DTIME(f) \subseteq NTIME(f) \subseteq DSPACE(f)$   $f(n) \geq n \forall n$

$DTIME(cn) = DTIME(3n) \forall c > 3$  (deterministische Zeitreduktion)  
 $DSPACE(f) \subseteq NSPACE(f) \subseteq DTIME(2^{\Omega(f)})$   
 wobei  $f(n) \geq \log(n) \forall n$

$L \subseteq NL \subseteq DSPACE(\log^2(n)) \subseteq DTIME(n^{\Omega(\log(n))})$   
 $NL \subseteq P$   $CS = LBA = NSPACE(n) \subseteq DTIME(2^{\Omega(n)})$   
 $DSPACE(n^2) \subseteq DTIME(2^{\Omega(n^2)})$   
 $DSPACE(\log n) \subsetneq DSPACE(\log^2 n) \subsetneq DSPACE(n) \subseteq NSPACE(n) \subsetneq DSPACE(n^{2.1}) \subsetneq PSPACE$   
 (folgt alles aus Platzhierarchiesatz)

$DTIME(O(n)) \subsetneq DTIME(O(n^2)) \subsetneq P \subsetneq DTIME(O(2^n)) \subsetneq DTIME(O((2 + \epsilon)^n))$   
 (folgt alles aus dem Zeithierarchiesatz)

**Es existieren Funktionen für die gilt (laut Lückensatz von Borodin):**  
 $DTIME(s) = NSPACE(s) = DTIME(s \cdot O(s)) = NSPACE(2^{2^{2^{2^s}}})$   
**Lückensatz von Borodin (1972):** Es sei  $r$  eine totale, berechenbare Funktion,  $r(n) \geq n$  für alle n. Dann gibt es effektiv eine totale, berechenbare Funktion  $s: \mathbb{N} \rightarrow \mathbb{N}$  mit der Eigenschaft:  
 $DTIME(s) = DTIME(r \circ s)$

**Starke Komplexitätsklassen:** sind Klassen, bei denen die Einhaltung der entsprechenden Schranken (Zeit- oder Platzschranken) bei JEDER Berechnung verbindlich ist. (Also nicht nur bei Akzeptierenden Berechnungen)  
**Zeitkonstruierbar:** Eine Funktion  $f: \mathbb{N} \rightarrow \mathbb{N}$  ist zeitkonstruierbar, falls es eine deterministische Turingmaschine gibt, die auf Eingabe  $1^n$  (d.h. n in unärer Kodierung) nach genau f(n) Schritten anhält.  
**Platzkonstruierbar:** Eine Funktion  $f: \mathbb{N} \rightarrow \mathbb{N}$  heißt platzkonstruierbar, falls es eine deterministische Turingmaschine gibt, die auf Eingabe  $1^n$  genau f(n) Felder auf den Arbeitsbändern markiert und dann anhält. Darüberhinaus darf sie den markierten Platz bei der Berechnung nicht verlassen.

Alle Zeitkonstruierbaren Funktionen sind auch Platzkonstruierbar  
 Es gibt Platzkonstruierbare Funktionen die nicht Zeitkonstruierbar sind.  
**Entscheidungsvariante:** Existiert eine Lösung für ein gegebenes Problem?  
**Berechnungsvariante:** Berechne eine Lösung von dem gegebenen Problem!  
**Optimierungsvariante:** Berechne eine kostenoptimale Lösung des Problems!  
**mittlere Suchdauer im binären Baum mit n Knoten:**  
 Summe von 1 bis n über  $(p_i * t_i) = MSD(B)$   $t_i =$  Knotentiefe,  $p_i =$  Wahrscheinlichkeit

**WIR WERDEN ALLE STERBEN!!**