

# Theoretische Informatik: Berechenbarkeit und Komplexität

Prof. Dr. Sibylle Schwarz  
HTWK Leipzig, Fakultät IM  
Gustav-Freytag-Str. 42a, 04277 Leipzig  
Zimmer Z 411 (Zuse-Bau)  
<https://informatik.htwk-leipzig.de/schwarz>  
[sibylle.schwarz@htwk-leipzig.de](mailto:sibylle.schwarz@htwk-leipzig.de)

Wintersemester 2021/22

# Informatik

**Informatik** Lehre von der **symbolischen Darstellung** und **Verarbeitung** von **Information** durch **Algorithmen**

Teilgebiete der Informatik:

**theoretisch**

- ▶ Sprachen zur Formulierung von Information und Algorithmen,
- ▶ Möglichkeiten und Grenzen der Berechenbarkeit durch Algorithmen,
- ▶ Grundlagen für technische und praktische (und angewandte) Informatik

**technisch**

- ▶ maschinelle Darstellung von Information
- ▶ Mittel zur Ausführung von Algorithmen

(Rechnerarchitektur, Hardware-Entwurf, Netzwerk, ...)

**praktisch** Entwurf und Implementierung von Algorithmen  
(Betriebssysteme, Compilerbau, SE, ...)

**angewandt** Anwendung von Algorithmen  
(Text- und Bildverarbeitung, Datenbanken, KI, Medizin-, Bio-, Wirtschafts-, Medien-Informatik, ...)

# Teilgebiete der theoretischen Informatik

## Formale Sprachen

(bekannt aus INB-Modul)

- ▶ Repräsentation von Informationen und Aufgaben in maschinenlesbarer Form (Mensch-Maschine-Kommunikation)
- ▶ Ausdrucksstärke und Flexibilität von Programmiersprachen
- ▶ Übersetzung von Programmiersprachen (z.B. in ausführbaren Code)

## Maschinenmodelle (Automaten)

(bekannt aus INB-Modul)

- ▶ Möglichkeiten und Grenzen verschiedener Modelle zur (maschinellen) Ausführung von Algorithmen

## Berechenbarkeitstheorie

- ▶ Welche Aufgaben sind überhaupt algorithmisch (mit Hilfe verschiedener Maschinenmodelle) lösbar?  
Auch negative Antworten sind sehr hilfreich  
(sparen Aufwand für ungeeignete Lösungsansätze)

## Komplexitätstheorie

- ▶ Welche Aufgaben sind mit beschränkten Ressourcen (z.B. Zeit, Speicherplatz) lösbar?
- ▶ Für welche Aufgaben können schnelle Algorithmen existieren?

# Prinzipien der theoretischen Informatik

ältester Zweig der Informatik (lange vor dem ersten Computer)

Mathematische Prinzipien:

- ▶ Abstraktion
  - ▶ ermöglicht verallgemeinerte Aussagen und breit einsetzbare Verfahren,
  - ▶ Ergebnisse und Verfahren oft nicht sofort praktisch anwendbar, müssen auf spezielle Situationen angepasst werden.
- ▶ Beweisbarkeit
  - ▶ erfordert präzise Modellierung der Aufgabe
  - ▶ Nachweis der Korrektheit von Hard- und Software (Tests können dies nicht !)

Wissen aus der theoretischen Informatik

- ▶ veraltet über viele Jahre kaum
- ▶ Grundlage für Verständnis von (schnelllebigem) Spezialwissen, z.B. konkrete Programmiersprachen, Domain-spezifische Sprachen, Transformationen verschiedener Darstellungen

# Aus der Modulbeschreibung

C144 Theoretische Informatik: Berechenbarkeit und Komplexität

**Arbeitsaufwand:** Präsenzzeit 56 SWS (= 2 SWS V + 2 SWS S)  
Vor- und Nachbereitungszeit 94 h ( $\approx 6$  h je Woche)

**Voraussetzungen:** anwendungsbereite Kenntnisse auf den Gebieten  
Modellierung, Logik, Formale Sprachen,  
Maschinenmodelle, Algorithmen und Datenstrukturen,  
Aufwandsabschätzungen

**Lernziele:** Nach erfolgreichem Abschluss des Moduls sind die Studierenden in der Lage, fundiert die prinzipiellen Möglichkeiten und Grenzen der verschiedenen Berechenbarkeitsmodelle einzuschätzen.

Sie besitzen ein Grundverständnis grundlegender Komplexitätsklassen.

Sie können die Komplexität ausgewählter Problembeispiele beurteilen und algorithmisch unlösbar oder schwer handhabbare Probleme als solche erkennen.

# Inhalt der Lehrveranstaltung

- ▶ **Berechenbarkeit** von Funktionen
- ▶ Algorithmenmodelle und ihre Rolle bei der Untersuchung von Grenzen der Berechenbarkeit
  
- ▶ Codierung von Aufgaben (Problemen) als Mengen
- ▶ **Entscheidbarkeit** von Mengen
- ▶ Unentscheidbare Probleme
  
- ▶ Komplexitätsmaße
- ▶ Zusammenhang zwischen Ausdrucksstärke von Algorithmenmodellen und **Komplexität** von Problemen
- ▶ Komplexitätsklassen (P, NP, PSPACE, ...)

jeweils mit vielen Beispielen und praktischen Folgerungen

# Lehrveranstaltungen

Folien, Übungsserien, aktuelle Informationen unter  
<https://informatik.htwk-leipzig.de/schwarz/lehre/ws21/tim>

**Vorlesung** jeden Donnerstag (2 SWS)

**Selbststudium** (Hausaufgaben): (6 h / Woche)

**schriftliche** Übungsserien (zu jeder Vorlesung)

Besprechung im folgenden Seminar

**Autotool** jeweils nach der Vorlesung bis Montag

**Seminar** jeden Donnerstag direkt vor der Vorlesung: (2 SWS)

Besprechung der Übungsserien (incl. Selbsttest)

Präsentation der Lösungen (Vorrechnen) durch

Teilnehmer

Alle Folien, Aufgaben, Lösungen schnell auffindbar  
mitbringen !

# Prüfung

## Prüfungsvorleistungen:

- ▶  $\geq 50\%$  aller Punkte für Autotool-Pflichtaufgaben und
- ▶  $\geq 3$  Punkte für Vorrechnen im Seminar

**Prüfung:** Klausur 120 min

Aufgabentypen ähnlich Übungsaufgaben

(Hilfsmittel: beidseitig handbeschriebenes A4-Blatt)



# Entscheidbarkeit: Eigenschaften von Grammatiken

- ▶  $E_3$ : Sprach-Äquivalenz von Typ-3-Grammatiken
- ▶  $E_2$ : Sprach-Äquivalenz von Typ-2-Grammatiken

gesucht ist jeweils ein Algorithmus mit

Eingabe: Paar  $(G_1, G_2)$  von Grammatiken,

Ausgabe: 1, falls  $L(G_1) = L(G_2)$ , sonst 0

praktische Motivation: Test bzw. Verkleinerung von regulären Ausdrücken, von Grammatiken (automatische Bewertung von Übungsaufgaben zu formalen Sprachen)

- ▶  $E_3$  ist entscheidbar,  
bekannt aus INB-Modul Automaten und formale Sprachen
- ▶  $E_2$  nicht entscheidbar (Nachweis später im Semester hier)

Methode:

**Reduktion:** wenn  $E_2$  entscheidbar, dann auch ...

# Sind verschiedene Aufgaben gleich schwer?

(eine typische Frage der Komplexitätstheorie)

- ▶ Eine (zulässige)  $k$ -Knoten-Färbung eines Graphen  $G = (V, E)$  ist eine Funktion  $f : V \rightarrow \{1, 2, \dots, k\}$  mit  $\forall uv \in E : f(u) \neq f(v)$ .
- ▶  $k\text{COL}$  = die Menge der Graphen, die eine  $k$ -Knoten-Färbung besitzen.
- ▶ Aufgaben: Gilt für einen gegebenen Graphen  $G$ 
  - ▶  $G \in 2\text{COL}$  ?
  - ▶  $G \in 3\text{COL}$  ?
- ▶ beide Aufgaben sind entscheidbar (warum?)
- ▶ für  $2\text{COL}$  ist ein effizienter Algorithmus bekannt, für  $3\text{COL}$  nicht.

Methode:

**Reduktion:** wenn man  $3\text{COL}$  effizient lösen könnte, dann auch ...

# Praktische Problemstellungen

Berechenbarkeitsmodell = Programmierparadigma

- ▶ Registermaschine: imperative Programmierung
- ▶ Loop- und While-Programme:  
strukturierte (imperative) Programmierung
- ▶ primitiv/allgemein-rekursive Funktionen:  
funktionale Programmierung
- ▶ (uniforme) Schaltkreise: parallele Programmierung
- ▶ nichtdeterministische Maschinen: Suchverfahren

für jede dieser Definitionen:

- ▶ exakte Beschreibung (Spezifikation)  
von (abstrakter) Syntax und Semantik = Interpreter-Bau

zwischen diesen Definitionen:

- ▶ semantik-erhaltende Übersetzung = Compiler-Bau

# Geschichte des Algorithmenbegriffs

Suche nach **Lösungsverfahren** für mathematische Aufgaben  
(symbolische Differentiation, Integration, Gleichungssysteme)

Beispiele:

1. Allgemeingültigkeit gegebener prädikatenlogischer Formeln
2. das 10. Hilbertsche Problem (1900):  
Lösbarkeit gegebener Polynomgleichungen in ganzen Zahlen  
z.B.  $a^2 + 1 = 0$ ,  $a^2 + b^2 - 1 = 0$ ,  $a^2 + b^2 - 1 = 0$ ,  
 $a^2 - b^2 = 0$ ,  $a^2 + b^2 - c^2 = 0$ ,  $a^5 = 0$ ,  $a^5 + b^5 - c^5 = 0$ , ...

... bzw. nach **Beweisen für deren Nicht-Existenz**

1. Gödel, Church, Turing (1936,...): nicht entscheidbar
2. Matijasevič (1970): nicht entscheidbar

# Bedeutung des Algorithmenbegriffs

(nach K. Wagner: Theoretische Informatik, Springer 2003)

- ▶ Die Bedeutung des Algorithmenbegriffs für Mathematik und Informatik entspricht der Bedeutung des Begriffes der natürlichen Zahlen.
- ▶ Die mathematische Präzisierung des Algorithmenbegriffs und die Erkenntnis der Grenzen des algorithmisch Machbaren gehören zu den wichtigsten intellektuellen Leistungen des 20. Jahrhunderts.

# Literatur

- ▶ Juraj Hromkovic: *Theoretische Informatik – Formale Sprachen, Berechenbarkeit, Komplexitätstheorie, Algorithmik, Kommunikation und Kryptographie* Vieweg 2014
- ▶ Ingo Wegener: *Theoretische Informatik* Teubner 2005
- ▶ Klaus Wagner: *Theoretische Informatik* Springer 2003
- ▶ Gottfried Vossen, Kurt-Ulrich Witt: *Grundkurs Theoretische Informatik* Vieweg 2016
- ▶ Uwe Schöning: *Theoretische Informatik – kurzgefasst* Spektrum 2001
- ▶ Renate Winter: *Theoretische Informatik* Oldenbourg 2002
- ▶ John E. Hopcroft, Jeffrey D. Ullman *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie* Addison-Wesley 1990
- ▶ Hartley Rogers Jr.: *Theory of Recursive Functions and Effective Computability*, 1987
- ▶ Michael Garey, David S. Johnson: *Computers and Intractability* Freeman 1979

# Wiederholung Grundlagen

aus den INB-Modulen

- ▶ Theoretische Informatik: Automaten und formale Sprachen  
<http://informatik.htwk-leipzig.de/schwarz/lehre/ws20/tib>
- ▶ Modellierung  
<http://informatik.htwk-leipzig.de/schwarz/lehre/ws20/modellierung>

# WH: Alphabet, Wort, Sprache

Operationen auf

Wörtern:  $\circ, *, ^R$

Sprachen:  $\cup, \cap, \circ, *, ^R$ ,

ÜA: Zeige

1.  $\forall u, v \in A^* : (u^R \circ v^R) = (v \circ u)^R$
2.  $\forall L \subseteq A^* : L \circ \{\varepsilon\} = L$
3.  $\forall L \subseteq A^* : L \circ \emptyset = \emptyset$
4.  $\forall L, L' \subseteq A^* : L \circ (L' \circ L)^* = (L \circ L')^* \circ L$



# Repräsentation formaler Sprachen

- ▶ Reguläre Ausdrücke (algebraisch, Terme)
- ▶ Grammatiken, Ableitungen (konstruktiv)
- ▶ Maschinenmodelle mit verschiedenen Akzeptanzbedingungen
  - ▶ NFA, DFA
  - ▶ PDA
  - ▶ LBA
  - ▶ TM

# WH: Reguläre Ausdrücke

**Syntax:** Terme (Bäume)

Induktive Definition der Menge **RegExp(A)** aller (einfachen) **regulären Ausdrücke** über Alphabet  $A$ :

**IA:**  $A \subseteq \text{RegExp}(A)$ ,  $\varepsilon \in \text{RegExp}(A)$ ,  $\emptyset \in \text{RegExp}(A)$

**IS:** Für alle  $E, F \in \text{RegExp}(A)$  gilt auch

- ▶  $E + F \in \text{RegExp}(A)$
- ▶  $EF \in \text{RegExp}(A)$
- ▶  $E^* \in \text{RegExp}(A)$

**Semantik:**  $L(E) \subseteq A^*$ , Äquivalenz

Erweiterte reguläre Ausdrücke: mit zusätzlich  $\cap, \setminus, \overline{\phantom{x}}, +, ^n, ^R$

ÜA:

1. Zeige  $ab(bab)^* \equiv (abb)^*ab$
2. Zeige  $\forall E, F \in \text{RegExp}(A) : E(FE)^* \equiv (EF)^*E$
3. Finde äquivalenten RegExp zu  $\overline{(a + b)^*ab(a + b)^*}$   
(mit Nachweis)

**Achtung:** Es gibt Sprachen, die **nicht** durch einen regulären Ausdruck dargestellt werden können. (Warum?)

# WH: Mächtigkeit unendlicher Mengen – Abzählbarkeit

Eine Menge  $A$  heißt

abzählbar gdw.  $\exists f : \mathbb{N} \rightarrow A$  surjektiv

überabzählbar sonst

Abschlusseigenschaften der Menge aller abzählbaren Mengen:

Sind die Mengen  $A_1$  und  $A_2$  abzählbar, dann sind auch

- ▶ jede Teilmenge von  $A_1$  abzählbar
- ▶  $A_1 \cup A_2$  abzählbar
- ▶  $A_1 \times A_2$  abzählbar (nach Cantor 1)
- ▶  $A_1^*$  abzählbar

Überabzählbar sind z.B. (nach Cantor 2):

- ▶  $2^{\mathbb{N}}$ ,
- ▶  $\mathbb{R}$ ,
- ▶  $2^{A^*}$  für  $A \neq \emptyset$

# Wortersetzung und Grammatiken

**Syntax:** Grammatik  $G = (N, T, P, S)$  mit

- ▶ Alphabet  $N \cup T$  ( $N$  enthält Hilfssymbole)  
Startsymbol  $S \in N$
- ▶ Wortersetzungssystem  $P \subseteq (N \cup T)^+ \times (N \cup T)^*$

**Semantik:**

- ▶ Ableitung von Wörtern in Grammatiken
- ▶  $L(G) \subseteq T^*$
- ▶ Äquivalenz
- ▶ Palindrom-, Dyck-, Łukasiewicz-Sprachen

ÜA:

1. Ableitungen von  $\varepsilon, b, ab, ba, aba$  in  
 $G = (\{S, T\}, \{a, b\}, \{S \rightarrow T|aS a|bS a, T \rightarrow b|\varepsilon\}, S)$
2. Mengen-Darstellung von  $L(G)$
3. Finde Grammatik  $G$  mit  $L(G) = \{a^m b^n a^{m+n} \mid m, n \in \mathbb{N}\}$
4. Finde Grammatik  $G$  mit  $L(G) = L(b(a^*b)^*)$

# Chomsky-Hierarchie

Eine **Grammatik**  $G = (N, T, P, S)$  ist vom Chomsky-Typ

0 immer,

1, falls für jede Regel  $(l \rightarrow r) \in P$  gilt:  $|l| \leq |r|$   
(monoton, kontextsensitiv)

2, falls Typ 1 und für jede Regel  $(l \rightarrow r) \in P$  gilt:  $l \in N$   
(kontextfrei)

3, falls Typ 2 und für jede Regel  $(l \rightarrow r) \in P$  gilt:  
 $l \in N$  und  $r \in (T \cup (T \circ N))$   
(regulär)

Eine **Sprache**  $L \subseteq T^*$  heißt vom **(Chomsky-)Typ  $i$**  für  $i \in \{0, \dots, 3\}$ , falls  $i$  die größte Zahl ist, für die eine Grammatik  $G$  vom Typ  $i$  mit  $L \setminus \{\varepsilon\} = L(G)$  existiert.

$\mathcal{L}_i$  bezeichnet die Menge aller Sprachen vom Typ  $i$ .

ÜA: Einordnung von  $\{a^n b^m c^k \mid n, m, k \in \mathbb{N}\}$ ,  $\{a^{2n} \mid n \in \mathbb{N}\}$ ,  
 $\{a^{n^2} \mid n \in \mathbb{N}\}$ ,  $\{a^n b^m a^n b^m \mid n \in \mathbb{N}\}$ ,  $\{a^n b^n \mid n, m \in \mathbb{N}\}$ ,  
 $\{a^n b^n c^n \mid n \in \mathbb{N}\}$

**Achtung:** Es gibt Sprachen, die **nicht** durch eine Grammatik erzeugt werden können, also **keinen** Chomsky-Typ haben (Warum?)

# WH: Endliche Automaten

Syntax: NFA] (nondeterministic finite automaton)

$A = (X, Q, \delta, I, F)$  mit

$X$  endliches Alphabet,

$Q$  endliche Menge von Zuständen,

$\delta$  Übergangsrelationen  $\delta : X \rightarrow 2^{(Q \times Q)}$ ,

$I \subseteq Q$  Startzustände,

$F \subseteq Q$  akzeptierende Zustände.

- ▶ Eigenschaften: vollständig, deterministisch (DFA)
- ▶ Akzeptanz von Wörtern, akzeptierte Sprache
- ▶ Transformationen zu Sprachoperationen
- ▶ Transformation von / zu Typ-3-Grammatik, RegExp
- ▶ nicht NFA-akzeptierbare Sprachen

ÜA:

1.  $L(A)$  für NFA  $A = (\{p, q\}, \{a, b\}, \delta, \{p\}, \{q\})$  mit  $\delta(a) = \{(p, p), (p, q), (q, p)\}$ ,  $\delta(b) = \{(p, p), (q, q)\}$
2. Finde NFA  $B$  mit  $L(B) = \overline{L(A)}$
3. Zeige:  $\{a^n b^m \mid n < m\}$  nicht NFA-akzeptierbar.
4. Verfahren zum Nachweis von  $L(A) = L(B)$  für NFA  $A, B$

# Maschinenmodelle

Definition (endliche Beschreibung) durch

- ▶ interne Steuerung  
(z.B. endliche Menge von Zuständen)
- ▶ externen Speicher mit speziellen Zugriffsmöglichkeiten
  - ▶ Typ des Speicherinhaltes  
(z.B. endliches Wort über endlichem Alphabet)
  - ▶ Zugriffsmethode (z.B. Lesen / Schreiben,  
einmal / wiederholt, feste Reihenfolge)

Konfigurationen (Momentaufnahmen)

und endliche Menge zulässiger lokaler Übergänge zwischen Konfigurationen

schrittweise Berechnung : Folge von Konfigurationen von einer Startkonfiguration über zulässige Konfigurationsübergänge

Akzeptanz einer Eingabe durch akzeptierende Berechnung:  
endliche Konfigurationenfolge zu einer akzeptierenden Konfiguration

## WH: Kellerautomat (PDA)

Erweiterung von NFA um Stack als internen Speicher  
nichtdeterministischer **Kellerautomat** (PDA)

$A = (X, Q, \Gamma, \delta, q_0, F, \perp)$  mit

$X$  Alphabet (endlich, nichtleer)

$Q$  Menge von Zuständen (endlich, nichtleer)

$\Gamma$  Kelleralphabet (endlich, nichtleer)

$q_0 \in Q$  Startzustand

$F \subseteq Q$  Menge der akzeptierenden Zustände

$\delta : (X \cup \{\varepsilon\}) \rightarrow (Q \times Q \times \Gamma \times \Gamma^*)$

für jedes  $s \in X \cup \{\varepsilon\}$  eine Übergangsrelation  
(mit Änderung des Kellerinhaltes)

$\perp \in \Gamma$  Kellerboden-Symbol

- ▶ Akzeptanz durch Zustand / leeren Keller
- ▶ Äquivalenz
- ▶ Transformation von / zu Typ-2-Grammatik



# WH: Turing-Maschine – Idee

Definition (endliche Beschreibung) durch

- ▶ Steuereinheit: Lese/Schreib-Kopf
- ▶ interner Speicher:
  1. endliche Menge von Zuständen (Lesen / Schreiben)
  2. beidseitig unendliches Arbeitsband aus Zellen
    - ▶ Typ des Speicherinhaltes:  
endliches Wort über endlichem Alphabet (Arbeitsalphabet)
    - ▶ Zugriffsmethode: Lesen / Schreiben  
Bewegung des Lese/Schreib-Kopfes auf ein Nachbarfeld
- ▶ externer Speicher:
  3. beidseitig unendliches Eingabeband aus Zellen
    - ▶ Typ des Speicherinhaltes:  
endliches Wort über endlichem Alphabet (Eingabealphabet)
    - ▶ Zugriffsmethode: Lesen  
Bewegung des Lese-Kopfes auf ein Nachbarfeld

übliche Vereinfachung der Struktur:

Zusammenfassen der Bänder im internen und externen Speicher zu einem Eingabe- und Arbeitsband (2,3) mit Lese/Schreib-Zugriff

## WH: Turing-Maschine – Definition

Turing-Maschine (TM)  $M = (X, Q, \Gamma, \delta, q_0, \square)$  mit

$X$  endliches Eingabealphabet

$Q$  endliche Menge von Zuständen

$\Gamma \supset X$  endliches Arbeitsalphabet

$\delta \subseteq (\Gamma \times Q \times \Gamma \times Q \times \{L, R, N\})$   
Übergangsrelation

$q_0$  Startzustand

$\square \in \Gamma \setminus X$  Leere-Zelle-Symbol

TM  $M$  heißt **deterministisch** (DTM) gdw.  
für jedes  $a \in \Gamma$  und jedes  $q \in Q$  gilt

$$|\{(a, q, b, p, x) \mid p \in Q, b \in \Gamma, x \in \{L, R, N\}\} \cap \delta| \leq 1$$

## WH: Beispiele für Turing-Maschinen

- ▶ TM  $M_1 = (\{a, b\}, \{q_0, f\}, \{a, b, \square\}, \delta, q_0, \square)$  mit

$$\delta = \{(a, q_0, a, q_0, L), (b, q_0, b, q_0, L), (\square, q_0, \square, f, R)\}$$

- ▶ TM  $M_2 = (\{a, b\}, \{q_0, q_a, q_b, f\}, \{a, b, \square\}, \delta, q_0, \square)$  mit

$$\delta = \left\{ \begin{array}{l} (a, q_0, \square, q_a, R), (b, q_0, \square, q_b, R), \\ (a, q_a, a, q_a, R), (b, q_a, b, q_a, R), \\ (a, q_b, a, q_b, R), (b, q_b, b, q_b, R), \\ (\square, q_0, \square, f, R), (\square, q_a, a, f, R), (\square, q_b, b, f, R) \end{array} \right\}$$

- ▶ TM  $M_3 = (\{1\}, \{q_0, q_1, q_2, q_3, q_4, f\}, \{1, x, \square\}, \delta, q_0, \square)$  mit

$$\begin{aligned} \delta = & \{(1, q_0, 1, q_1, R), (1, q_1, x, q_2, R), (1, q_2, 1, q_3, R)\} \\ & \cup \{(1, q_3, x, q_2, R), (1, q_4, 1, q_4, L), (x, q_1, x, q_1, R)\} \\ & \cup \{(x, q_2, x, q_2, R), (x, q_3, x, q_3, R), (x, q_4, x, q_4, L)\} \\ & \cup \{(\square, q_1, \square, f, L), (\square, q_2, \square, q_4, L), (\square, q_4, \square, q_0, R)\} \end{aligned}$$

# WH: Konfigurationen von TM

TM  $M = (X, Q, \Gamma, \delta, q_0, \square)$

**Konfiguration**  $uqv \in (\Gamma^* \times q \times \Gamma^*)$  mit

- ▶ aktuellem Bandinhalt  $w = uv$
- ▶ aktuellem Zustand der TM  $q$
- ▶ Schreib-/Lesekopf der TM zeigt auf erstes Symbol von  $v$   
(Leere-Zelle-Symbol, falls  $v = \varepsilon$ )

**Startkonfigurationen**  $q_0w$  mit  $w \in X^*$

**Konfigurationsübergänge** von  $upv$  mit  $u = u'a$  und  $v = bv'$ :

für  $(b, p, c, q, R) \in \delta$  :  $upv \vdash ucqv'$

für  $(b, p, c, q, L) \in \delta$  :  $upv \vdash u'qacv'$

für  $(b, p, c, q, N) \in \delta$  :  $upv \vdash uqcv'$

## WH: Berechnung durch TM

TM  $M = (X, Q, \Gamma, \delta, q_0, \square)$

Eine Folge  $k_0, k_1, \dots$  von Konfigurationen  $k_i \in \Gamma^* \times Q \times \Gamma^*$  heißt **Berechnung** durch  $M$  für das Wort  $w \in X^*$  gdw.

1.  $k_0 = q_0w$  ist die Startkonfiguration mit Eingabe  $w$ .
2. Für jedes  $i$  ist  $k_i \vdash k_{i+1}$  ein Konfigurationsübergang von  $M$ .

Ende von Berechnungen der TM  $M$  für ein Eingabewort  $w$ :

1. Berechnung durch  $M$  endet in  $k_i$  ( $M$  **hält**) gdw.  
aus einem  $k_i$  kein Konfigurationsübergang von  $M$  möglich ist.  
Berechnung von  $M$  endet in  $k_i$ .
2. Berechnung durch  $M$  endet nicht (unendliche Berechnung)  
gdw.  
für jedes  $i \in \mathbb{N}$  der Konfigurationsübergang  $k_i \vdash k_{i+1}$  existiert.

ÜA: Für welche Wörter halten die TM  $M_1, M_2, M_3$  auf Folie 26?

# Was bisher geschah

## Wiederholung Berechnungsmodell DTM (deterministische Turing-Maschine)

- ▶ Definition  $M = (X, Q, \Gamma, \delta, q_0, \square)$
- ▶ Konfigurationen
- ▶ Konfigurationenfolgen (Berechnungen)
- ▶ Akzeptanz durch Halt
- ▶ Nebenwirkung: Änderung des Bandinhaltes
- ▶ Berechnung von Funktionen  $X^* \rightarrow \Gamma^*$  durch DTM
- ▶ Berechnung von Funktionen  $\mathbb{N} \rightarrow \mathbb{N}$  durch DTM
- ▶ Beispiele
- ▶ Transformationen zwischen Darstellungen natürlicher Zahlen

# Darstellung von Daten

- ▶ DTM können Zeichenketten akzeptieren und manipulieren (Nebenwirkung der Ausführung)
- ▶ Zeichenketten über (endlichem) Alphabet  $A$  können als Zahlen zur Basis  $|A|$  oder  $|A| + 1$  interpretiert werden.

Besispiele für Bedeutungen von Zeichenketten:

- ▶ Wörter aus Sprachen
- ▶ Zahlen
- ▶ Darstellung strukturierter Daten, z.B.
  - ▶ Tupel,
  - ▶ Listen,
  - ▶ Bäume,
  - ▶ Graphen,
  - ▶ NFA, TM, ...
  - ▶ Konfigurationen, Konfigurationsfolgen
  - ▶ Programmtexte,
  - ▶ ...

# Darstellung von Zahlen als Zeichenketten

- ▶ natürliche Zahlen: Unär-, Binär-, ...-Darstellung
- ▶ ganze, gebrochene Zahlen: Paare natürlicher Zahlen mit um Trennzeichen erweitertem Alphabet, z.B.  $2/3$  codiert durch Paar  $(2, 3)$  in Unärdarstellung mit Trennzeichen 0 : 110111
- ▶ übliche Approximationen reeller Zahlen durch Maschinenzahlen



# Darstellung von Zeichenketten als natürliche Zahlen

Darstellung des Alphabets (endliche Menge):

1. (beliebige) Reihenfolge der Symbole in  $A$  festlegen:

$$A = \{a_1, \dots, a_{|A|}\}$$

2. Nummerierung der Symbole:

$$\text{Zuordnung } \forall i \in \{1, \dots, |A|\} : c(a_i) = i \in \mathbb{N}$$

3. zur Weiterverarbeitung: Unärdarstellung  $c' : A \rightarrow 1^*$  mit

$$\forall i \in \{1, \dots, |A|\} : c'(a_i) = 1^i$$

Beispiel:  $A = \{\clubsuit, \spadesuit, \heartsuit, \diamondsuit\}$ ,

Nummerierung  $c : A \rightarrow \mathbb{N}$  mit  $c(\clubsuit) = 1, c(\spadesuit) = 2, c(\heartsuit) = 3, c(\diamondsuit) = 4$

und  $c' : A \rightarrow 1^+$  mit  $c'(\clubsuit) = 1, c'(\spadesuit) = 11, c'(\heartsuit) = 111, c'(\diamondsuit) = 1111$

Darstellung von Wörtern  $w \in A^*$  über endlichem Alphabet  $A$ :

1. Nummerierung und Unärdarstellung der Alphabet-Symbole (s.o.),

2. Verkettung mit 0 als Trennsymbol

$$c'(w) = c'(w_1 \dots w_{|w|}) = c'(w_1)0c'(w_2)0 \dots 0c'(w_{|w|})$$

3. Durch Interpretation der Zeichenkette  $c'(w) \in \{0, 1\}^*$  als Binärzahl wird  $w$  die eindeutige Zahl  $c''(w) \in \mathbb{N}$  zugeordnet.

Beispiel:  $c'(\clubsuit\heartsuit\spadesuit) = 10111011, c''(\clubsuit\heartsuit\spadesuit) = 187$

# Darstellung strukturierter Daten

- ▶ direkte Darstellung  $c : D \rightarrow A^*$  der einfachen Daten  $d \in D$  (oder Numerierung in Unärdarstellung analog Alphabet)
- ▶ Codierung von Paaren, Tupeln, Listen  $(x_1, \dots, x_n)$  als Wörter z.B.  $(5, \heartsuit, 2, 1) \mapsto 11111101110111011$
- ▶ Schachtelungen z.B. durch neue oder mehrfache Trennzeichen z.B.  $(5, (\heartsuit, 2), 1) \mapsto 111111 \bullet 1110111 \bullet 11$  oder  $(5, (\heartsuit, 2), 1) \mapsto 1111110011101110011$

## Beispiel: Darstellung von Graphen

(endliche gerichtete) Graphen  $G = (V, E)$  mit  $E \subseteq V \times V$

Idee:

Betrachte  $V$  als Alphabet und  $E$  als Folgen der Länge 2 über  $V$

- ▶ (beliebige) Reihenfolge der Knoten festlegen  $V = \{v_1, \dots, v_{|V|}\}$
- ▶ Numerieren der Knoten, z.B. unär:  $c_V : V \rightarrow 1^*$  mit  $\forall i \in \{1, \dots, |V|\} : c_V(v_i) = 1^{i+1}$
- ▶ Darstellung der Kanten:  $c_E(u, v) = c_V(u)0c_V(v)$
- ▶  $c(G) = c_V(v_1)0 \cdots 0c_V(v_{|V|})000c_E(u_1, v_1)00 \cdots 00c_E(u_{|E|}, v_{|E|})$

oder

- ▶ Darstellung jedes Knotens  $v \in V$  als Paar  
( $c_V(v)$ , Darstellung der Liste der Namen aller Nachbarn)  
(ggf. als Tupel mit weiteren Daten, z.B. Schlüssel)
- ▶  $c(G) =$  Darstellung von  $G$  Graph als Codierung der Liste aller Knoten (Adjazenzliste)

oder ...

# Darstellung von TM

Darstellung jeder TM  $M = (X, Q, \Gamma, \delta, q_0, \square)$  als (endliches) Wort über dem endlichen Alphabet

$X' = X \cup \Gamma \cup Q \cup \{L, R, N\} \cup \text{Klammern} \cup \text{Trennzeichen}$  möglich

Die Menge aller solcher Darstellungen von TM ist eine Sprache über  $X'$ .

Darstellung von 0 TM als  $\{0, 1\}$ -Folge:

- ▶ Alphabet, Zustände, Richtungen in Unärcodierung (aus  $1^*$ )
- ▶ Trennzeichen als Folgen aus  $0^*$

ÜA: Hat die Sprache aller  $\{0, 1\}$ -Folgen, die in dieser Darstellung korrekte Codierungen von TM sind, einen Chomsky-Typ? Falls ja, welchen?

## Beispiel: Darstellung von TM

Darstellung von TM als Wort aus  $\{0, 1\}^*$

Beispiel: TM  $M = (X, Q, \Gamma, \delta, q_0, \square)$  in  $\{0, 1\}^*$  mit

- ▶  $X = \{1\}$ ,  $\Gamma = \{1, \square\}$  mit  $c(1) = 1$ ,  $c(\square) = 11$
- ▶  $Q = \{q_0, q_1\}$  mit Startzustand  $q_0$ :  $c(q_0) = 1$ ,  $c(q_1) = 11$
- ▶  $B = \{L, R, N\}$  mit  $c(L) = 1$ ,  $c(R) = 11$ ,  $c(N) = 111$
- ▶  $\delta = \{(\square, q_0, 1, q_1, R), (\square, q_1, 1, q_0, L), (1, q_0, 1, q_1, L)\}$

für jeden Übergang  $u = (a, p, b, q, r) \in \delta$ :

Darstellung:  $c(u) = c(a)0c(p)0c(b)0c(q)0c(r)$

$$c(\square, q_0, 1, q_1, R) = \underbrace{11}_{\square} 0 \underbrace{1}_{q_0} 0 \underbrace{1}_1 0 \underbrace{11}_{q_1} 0 \underbrace{11}_R$$

$$c(\square, q_1, 1, q_0, L) = 11011010101$$

$$c(1, q_0, 1, q_1, L) = 1010101101$$

Darstellung der TM  $M$  ( $\delta$  genügt):

$$\begin{aligned} c(M) &= c(\square, q_0, 1, q_1, R)00c(\square, q_1, 1, q_0, L)00c(1, q_0, 1, q_1, L) \\ &= 110101011011001101101010101001010101101 \end{aligned}$$

# Darstellung von TM-Konfigurationen

Konfigurationen einer TM  $M = (X, Q, \Gamma, \delta, q_0, \square)$ :

$upv \in \Gamma^* \times Q \times \Gamma^*$

- ▶ Darstellung von  $u$  und  $v$  als  $\{0, 1\}$ -Folgen (wie bisher)
- ▶ Darstellung von  $p$  als Unärzahl
- ▶ Markierung der Kopfposition (Position der als Zustand zu interpretierenden Zeichenkette)  
z.B. durch doppeltes Trennzeichen nach Zustand

Beispiel: TM von voriger Folie

Startkonfiguration  $\underbrace{\square}_u q_0 \underbrace{11}_v \mapsto \underbrace{11}_{c(u)} 0 \underbrace{1}_{c(q_0)} 00 \underbrace{1010}_{c(v)}$

# Gödelisierung

(Kurt Gödel 1906 - 1978)

- ▶ Die bisher vorgestellten Codierungen  $c$  ordnen jedem Element  $x$  der darzustellenden Menge (Graphen, TM, Konfigurationen, ...) ein **eindeutiges** Wort  $c(x) \in \{0, 1\}^*$  zu.
  - ▶ Interpretation der Zeichenkette  $c(x) \in \{0, 1\}^*$  als Binärzahl ordnet jedem Element  $x$  der darzustellenden Menge eine **eindeutige** natürliche Zahl  $c'(x) \in \mathbb{N}$  zu.
1. Beide Funktionen  $c$  und  $c'$  sind injektiv und algorithmisch berechenbar.
  2. Es lässt sich algorithmisch feststellen, ob ein Wort  $w \in \{0, 1\}^*$  (eine Zahl  $n \in \mathbb{N}$ ) die Darstellung eines Elementes ist. (ÜA: Wie?)
  3. Die Umkehrfunktionen  $c^{-1} : c(X) \rightarrow X$  und  $c'^{-1} : c'(X) \rightarrow X$  sind injektiv.  
Aus jeder korrekten Darstellung lässt sich das Element  $x$  eindeutig algorithmisch berechnen.

Solche Funktionen  $c$  und  $c'$  heißen **Gödelisierungen**.

Für jede Gödelisierung  $c$  heißt  $c(x)$  **Gödelnummer** des Elementes  $x$ .

# Darstellung von Problemen als Sprachen

Aufgaben (Probleme): Zuordnung zwischen

- ▶ Eingaben
- ▶ passenden Ausgaben

(oft strukturierte Daten)

Aufgabe: Abbildung = Menge von Paaren (Eingabe, Ausgabe)  
(übliche Codierungen)

Spezialfall Entscheidungsprobleme: Ausgabe  $\in \{0, 1\}$

- ▶ definiert eine Eigenschaft auf der Menge aller Eingaben
- ▶ repräsentiert durch Urbild der 1
- ▶ charakteristische Funktion der Menge der Eingaben mit dieser Eigenschaft



# Aufgaben und Instanzen – Beispiele

- ▶ Enthaltensein in einer Folge (von Elementen des Typs  $E$ )

**Aufgabe:**  $F = \{((x_1, \dots, x_n), y) \in E^* \times E \mid \exists k \in \{1, \dots, n\} : x_k = y\}$

Menge aller Paare  $((x_1, \dots, x_n), y)$  mit  
 $y \in \{x_1, \dots, x_n\}$

**Instanz:**  $((1, 3, 4, 6), 4)$  (Gilt  $((1, 3, 4, 6), 4) \in F$  ?)

Ja,  $((1, 3, 4, 6), 4) \in F$ , weil  $4 \in \{1, 3, 4, 6\}$

**Lösungsverfahren:** (beliebiges) Suchverfahren in Folgen

- ▶ SAT (Erfüllbarkeitsproblem der Aussagenlogik)

**Aufgabe:**  $\text{SAT} = \{\varphi \in \text{AL} \mid \text{Mod}(\varphi) \neq \emptyset\}$

Menge aller erfüllbaren aussagenlogischen Formeln

**Instanz:**  $\neg a \wedge (a \vee b \vee c) \wedge \neg c \wedge \neg b$

(Gilt  $(\neg a \wedge (a \vee b \vee c) \wedge \neg c \wedge \neg b) \in \text{SAT}$  ?)

Nein, weil  $\text{Mod}(\varphi) = \emptyset$

**Lösungsverfahren** (Entscheidungsverfahren):

- ▶ semantisch: Wahrheitstabellen
- ▶ syntaktisch: Kalküle, z.B. Resolution, äquivalente Umformungen von ... zu ...

## Beispiel: Graphen-Probleme

- ▶  $k$ -Färbung eines Graphen  $G = (V, E)$ :  
Abbildung  $f : V \rightarrow \{1, 2, \dots, k\}$
- ▶ Färbung  $f$  von  $G = (V, E)$  ist konfliktfrei  
gdw.  $\forall xy \in E : f(x) \neq f(y)$

$$k\text{COL} = \{G \mid \exists f : f \text{ ist konfliktfreie } k\text{-Färbung von } G \}$$

Hamilton-Kreis in  $G = (V, E)$ :

Kreis in  $G$  durch alle Knoten in  $V$

$\text{HC} = \{G \mid G \text{ ist ungerichteter Graph und } G \text{ enthält Hamilton-Kreis} \}$  (Hamiltonian Circuit)

$\text{DHC} = \{G \mid G \text{ ist gerichteter Graph und } G \text{ enthält gerichteten Kreis durch alle Knoten} \}$  (Directed HC)

## Codierung in natürlichen Zahlen

DTM akzeptieren / manipulieren Zeichenketten.

Die meisten Berechenbarkeitsmodelle (z.B. Programme, rekursive Funktionen) rechnen aber mit natürlichen Zahlen.

Satz: Das ist keine Einschränkung der Allgemeinheit, denn man kann jedes strukturierte Datum in eine einzige natürliche Zahl codieren.

Vorgehen wie in den vorangegangenen Beispielen:

1. Darstellung als Zeichenkette  $c(d) \in \{0, 1\}^*$
2. Interpretation von  $c(d)$  als Binärzahl  $c'(d) \in \mathbb{N}$

Bei Berechnungen mit Zahlen ist der Umweg über Zeichenketten oft zu aufwendig.

deshalb gesucht:

direkte Codierung und Decodierung strukturierter Daten in natürlichen Zahlen

# Codierung von Zahlenpaaren

gesucht sind (TM-)berechenbare Funktionen

Konstruktor:  $C : \mathbb{N}^2 \rightarrow \mathbb{N}$

Destruktoren:  $P_1, P_2 : \mathbb{N} \rightarrow \mathbb{N}$

Testfunktion:  $T : \mathbb{N} \rightarrow \{0, 1\}$

mit den folgenden Eigenschaften (Spezifikation):

$$\forall x_1, x_2 \in \mathbb{N} : P_1(C(x_1, x_2)) = x_1$$

$$\forall x_1, x_2 \in \mathbb{N} : P_2(C(x_1, x_2)) = x_2$$

$$\forall x \in \mathbb{N} : T(x) = 1 \iff \exists x_1, x_2 \in \mathbb{N} : x = C(x_1, x_2)$$

verschiedene Möglichkeiten, z.B.:

- ▶  $C(x_1, x_2) = (x_1 + x_2)(x_1 + x_2 + 1)/2 + x_1$
- ▶  $C(x_1, x_2) = 2^{x_1}(2x_2 + 1)$ ,
- ▶  $C(x_1, x_2) = 2^{x_1} \cdot 3^{x_2}$

ÜA: jeweils  $C(2, 3)$ ,  $C(3, 2)$ ,  $T(10)$ ,  $T(12)$ ,  $P_1(12)$ ,  $P_2(12)$ ,  
Algorithmen für  $T$ ,  $P_i$ .

# Codierung von Listen

Konstruktor  $L : \mathbb{N}^* \rightarrow \mathbb{N}$

Destruktoren  $D_i : \mathbb{N} \rightarrow \mathbb{N}$ .

zwei (von vielen) Möglichkeiten:

- ▶ mittels einer Paar-Codierung  $C$

IA:  $L([]) = 0$

IS:  $\forall a \in \mathbb{N} \forall w \in \mathbb{N}^* : L([a] \circ w) = C(a + 1, L(w))$

- ▶ direkte Kodierung als Produkt von Primzahlpotenzen

$$L([x_0, x_1, \dots, x_n]) = 2^{x_0+1} \cdot 3^{x_1+1} \cdot \dots \cdot p(n)^{x_n+1}.$$

ÜA: jeweils  $L([])$ ,  $L([5])$ ,  $L([2, 3, 0])$ , Algorithmus für  $D_i$

ÜA: die Funktion  $p : n \mapsto$  die  $n$ -te Primzahl, also  $p(0) = 2, p(1) = 3, p(2) = 5, \dots$  ist While-berechenbar.

# Codierung von Bäumen

## Motivation

- ▶ allgemein: Baum = Term in einer Signatur,
- ▶ Signatur: eine endlichen Menge von Funktionssymbolen mit zugeordneter Stelligkeit.
- ▶ Bsp:  $\Sigma = \{(f, 2), (g, 1), (a, 0)\}$ ,  
 $t = f(f(a, g(a)), a) \in \text{Term}(\Sigma)$ .  
Notation  $\text{root}(t) = f$ ,  $\text{args}(t) = [f(a, g(a)), a]$ .
- ▶ wird u.a. benötigt, um (prädikatenlogische) Formeln als Zahlen zu kodieren.

Realisierung:  $B(t) = C(\text{num}(\text{root}(t)), L([B(t_1), \dots, B(t_k)]))$   
mit  $\text{args}(t) = [t_1, \dots]$ , Paar-Kodierung  $C$ ,  
Listen-Kodierung  $L$ , sowie Symbol-Nummerierung

# Was bisher geschah

## Wiederholung TM / DTM

- ▶ Definition  $M = (X, Q, \Gamma, \delta, q_0, \square)$
- ▶ Konfigurationen
- ▶ Konfigurationsfolgen (Berechnungen)
- ▶ Akzeptanz durch Halt
- ▶ Nebenwirkung: Änderung des Bandinhaltes

## Wiederholung abzählbare / überabzählbare Mengen

### Gödelisierung:

- ▶ Darstellung durch endliche Zeichenketten:
  - ▶ Zahlen
  - ▶ strukturierte Daten: Paare, Tupel, Listen, Bäume
  - ▶ Graphen, TM, TM-Konfigurationen
- ▶ Codierung strukturierter Daten durch natürliche Zahlen, z.B.: Paare, Tupel, Listen, Bäume
- ▶ Repräsentation von Problemen als Sprachen / Mengen

## WH: Nichtdeterministische TM

TM  $M = (\{0, 1\}, \{q_0, q_1, f\}, \{0, 1, \square\}, \delta, q_0, \square)$  mit

$$\delta = \{ \begin{array}{l} (0, q_0, 1, q_0, R) \\ (1, q_0, 1, q_0, R) \\ (1, q_0, 1, q_1, R) \\ (1, q_1, 1, q_1, R) \\ (\square, q_1, \square, f, L) \end{array} \}$$

endet bei Eingabe des Wortes 11011 bei (z.B.) folgender  
Berechnung

$$\begin{array}{l} q_0 11011 \vdash 1q_0 1011 \vdash 11q_0 011 \vdash 111q_0 11 \vdash 1111q_1 1 \\ \vdash 11111q_1 \square \vdash 1111f 1 \end{array}$$



# Simulation nichtdeterministischer TM

## Satz

*Zu jeder nichtdeterministischen TM  $M$  existiert eine deterministische TM  $M'$  mit  $L(M) = L(M')$ .*

Beweisidee:

Menge aller möglichen Berechnungen von  $M$  bei Eingabe von  $w$  bilden einen Baum (evtl. mit unendlich langen Pfaden)

Knoten: Konfigurationen

Wurzel: Startkonfiguration (Startzustand, Eingabewort)

Blätter: Halt-Konfigurationen (ohne Folgekonfigurationen)

Kanten: zulässige Konfigurationsübergänge in  $M$

$M'$  führt Breitensuche in diesem Baum durch  
(simuliert parallele Berechnung aller Möglichkeiten, dovetailing),  
Bandinhalt von  $M'$  ist Liste von Konfigurationen aus  $M$   
 $M'$  hält, sobald eine Halt-Konfiguration von  $M$  auf dem  
Arbeitsband steht.

# DTM zur Berechnung von Funktionen

Beobachtung (der Nebenwirkung) der Berechnung von DTM:

Jede **deterministische** TM  $M$  berechnet eine Funktion

$$f_M : X^* \rightarrow \Gamma^*$$

**Eingabe**  $w \in X^*$ : Inhalt des Arbeitsbandes bei Start der Berechnung (Eingabewort)

**Ausgabe**  $v \in \Gamma^*$ : Inhalt des Arbeitsbandes nach Halt der TM

Falls  $M$  bei Eingabe von  $w$  nicht hält, ist  $f_M(w)$  nicht definiert.

TM  $M$  berechnet i.A. eine **partielle** Funktion

$$f_M : X^* \rightarrow \Gamma^*$$

da  $f_M$  nur für die Wörter  $w \in X^*$  definiert ist, bei deren Eingabe  $M$  hält.

# Turing-berechenbare Funktionen

Jede deterministische TM  $M$  definiert die (partielle) Funktion  $f_M : X^* \rightarrow \Gamma^*$ , wobei  $\forall w \in X^*$

$$f_M(w) = \begin{cases} v & \text{falls Bandinhalt } v, \text{ nachdem } M \text{ h\u00e4lt} \\ \text{nicht definiert} & \text{falls } M \text{ nicht h\u00e4lt} \end{cases}$$

Beispiel:  $M = (\{a, b\}, \{q_0, q_1, q_2, q_3\}, \{a, b, \square\}, \delta, q_0, \square)$  mit

$$\begin{aligned} \delta = & \{(a, q_0, \square, q_1, R), (b, q_0, \square, q_2, R), (\square, q_0, \square, q_0, N)\} \\ & \cup \{(a, q_1, a, q_1, R), (b, q_1, b, q_1, R), (\square, q_1, a, q_3, N)\} \\ & \cup \{(a, q_2, a, q_2, R), (b, q_2, b, q_2, R), (\square, q_2, b, q_3, N)\} \end{aligned}$$

definiert die Funktion

$$f_M(w) = \begin{cases} vx & \text{falls } w = xv \text{ mit } x \in \{a, b\} \text{ und } v \in \{a, b\}^* \\ \text{nicht definiert} & \text{falls } w = \varepsilon \end{cases}$$

Eine Funktion  $f : X^* \rightarrow X^*$  hei\u00dft **Turing-berechenbar** gdw. eine deterministische TM  $M$  mit  $f = f_M$  existiert.

## Beispiele

- ▶  $g : \{1\}^* \rightarrow \{1\}^*$  mit  $g(w)$  für alle  $w \in \{1\}^*$  undefiniert  
ist berechenbar durch die TM  $M = (\{1\}, \{q_0\}, \{1, \square\}, \delta, q_0, \square)$   
mit  $\delta = \{(q_0, 1, q_0, 1, N), (q_0, \square, q_0, \square, N)\}$
- ▶  $f : \{a, b\}^* \rightarrow \{a, b\}^*$  mit  
für alle  $w \in \{a, b\}^*$  gilt

$$f(w) = \begin{cases} a^{m+n} & \text{falls } w = a^m b a^n \\ \text{nicht definiert} & \text{sonst} \end{cases}$$

ist berechenbar durch die TM

$M = (\{a, b\}, \{q_0, q_1, q_2, f\}, \{a, b, \square\}, \delta, q_0, \square)$  mit

$$\begin{aligned} \delta = & \{(a, q_0, a, q_0, R), (b, q_0, a, q_1, R), (\square, q_0, \square, q_0, N)\} \\ & \cup \{(a, q_1, a, q_1, R), (b, q_1, b, q_1, N), (\square, q_1, \square, q_2, L)\} \\ & \cup \{(a, q_2, \square, f, N)\} \end{aligned}$$

hält genau bei jedem Eingabewort aus  $a^* b a^*$ ,

**verschiebt** dabei den Teil recht von  $b$  um eine Zelle nach links  
(mit  $a \mapsto 1$  und  $b \mapsto \bullet$  : Addition in Unärdarstellung)

## Komposition (Nacheinanderausführung) von TM

Für DTM  $M_1 = (X, Q_1, \Gamma_1, \delta_1, q_1, \square)$  und  
 $M_2 = (X, Q_2, \Gamma_2, \delta_2, q_2, \square)$  mit  $Q_1 \cap Q_2 = \emptyset$   
berechnet  $M = (X, Q_1 \cup Q_2 \cup Z, \Gamma_1 \cup \Gamma_2, \delta, q_1, \square)$  mit

$$\begin{aligned} \delta = & \delta_1 \cup \delta_2 \\ & \cup \{(x, q, x, z_0, N) \mid x \in \Gamma_1 \wedge q \in Q_1 \wedge (x, q, \cdot, \cdot, \cdot) \notin \delta_1\} \\ & \cup \delta_Z \cup \{(x, z_f, x, q_2, N) \mid x \in \Gamma_1\} \end{aligned}$$

die Nacheinanderausführung von  $f_{M_1}$  und  $f_{M_2}: x \mapsto f_{M_2}(f_{M_1}(x))$   
wobei  $\delta_Z$  alle nötigen Übergänge zwischen  $z_0 \in Z$  und  $z_f \in Z$   
enthält zur

- ▶ Prüfung, ob Ausgabe  $\in X^*$  (also korrekte Eingabe für  $M_2$ ) und
- ▶ Bewegung des Kopfes an Anfang der Ausgabe

Beispiel:

Addition einer festen gegebenen Zahl durch mehrmalige  
Nacheinanderausführung von  $M_{+1}$

## Beispiel

TM  $M_{=0} = (X, Q, \Gamma, \delta, q_0, \square)$  mit

$$X = \{0, 1\}$$

$$Q = \{q_0, p, f_1, f_2\}$$

$$\Gamma = \{0, 1, \square\}$$

$$\delta = \left\{ \begin{array}{l} (1, q_0, 1, f_2, N), (\square, q_0, \square, f_2, N), \\ (0, q_0, 0, p, R), (1, p, 1, f_2, L), \\ (0, p, 0, f_2, L), (\square, p, \square, f_1, L), \end{array} \right\}$$

hält in Konfiguration

- ▶  $f_10$  für Eingabe 0 und
- ▶  $f_2w$  für jede andere Eingabe  $w$

$M_{=0}$  testet, ob Bandinhalt = 0

# Fallunterscheidung

Für TM

$$M_{=0} = (X, Q_{=0}, \Gamma_{=0}, \delta_{=0}, q_0, \square)$$

$$M_1 = (X, Q_1, \Gamma_1, \delta_1, q_1, \square)$$

$$M_2 = (X, Q_2, \Gamma_2, \delta_2, q_2, \square)$$

mit  $Q_{=0}, Q_1, Q_2$  paarweise disjunkt

berechnet die TM

$$M = (X, Q_{=0} \cup Q_1 \cup Q_2, \Gamma_{=0} \cup \Gamma_1 \cup \Gamma_2, \delta, q_1, \square)$$

mit

$$\delta = \delta_{=0} \cup \delta_1 \cup \delta_2 \cup \bigcup_{x \in \Gamma_1} \{(x, f_1, x, q_1, N), (x, f_2, x, q_2, N)\}$$

die Funktion  $f_{M_1}$ , falls 0 auf dem Eingabeband steht, sonst  $f_{M_2}$

# Existenz nicht Turing-berechenbarer Funktionen

Ist jede Funktion  $f : \mathbb{N} \rightarrow \mathbb{N}$  (binärcodiert) Turing-berechenbar?  
(analog  $f : X^* \rightarrow X^*$ ,  $f : X^* \rightarrow Y^*$ ,  $f : \mathbb{N}^n \rightarrow \mathbb{N}$ )

**Nein** (Gegenbeispiel später)

Begründung:

1. Wieviele **Turing-Maschinen** über dem Alphabet  $\{0, 1\}$  gibt es?  
**abzählbar** viele  
Jede TM kann endlich codiert und die Menge aller dieser Codierungen kann (z.B. quasi-lexikographisch) angeordnet werden.
2. Wieviele **Funktionen**  $f : \mathbb{N} \rightarrow \mathbb{N}$  gibt es?  
**überabzählbar** viele  
(zweites Diagonalverfahren von Cantor)

Damit existieren sogar sehr viel mehr (überabzählbar viele) Funktionen  $f : \mathbb{N} \rightarrow \mathbb{N}$   
( $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ ,  $f : X^* \rightarrow X^*$ ,  $f : X^* \rightarrow Y^*$ ,  $f : \mathbb{N}^n \rightarrow \mathbb{N}$ ),  
die nicht von TM berechnet werden können.



# Mehrband-Turingmaschinen

Idee:

- ▶ mehrere beidseitig unendliche Arbeitsbänder
- ▶ je ein Schreib-/Lesekopf je Band arbeiten unabhängig voneinander

$k$ -Band-Turing-Maschine  $M = (X, Q, \Gamma, \delta, q_0, \square)$  mit

$X$  endliches Eingabealphabet

$Q$  endliche Menge von Zuständen

$\Gamma \supset X$  endliches Arbeitsalphabet

$\delta \subseteq (\Gamma^k \times Q \times \Gamma^k \times Q \times \{L, R, N\}^k)$   
Übergangsrelation

$q_0$  Startzustand

$\square \in \Gamma \setminus X$  Leere-Zelle-Symbol

## Beispiel für Mehrband-TM

2-Band-Turing-Maschine  $M = (X, Q, \Gamma, \delta, q_0, \square)$  mit

$$X = \{0, 1\}$$

$$Q = \{q_0, p, f\}$$

$$\Gamma = \{0, 1, \square\}$$

$$\delta = \left\{ \begin{array}{l} ( (0, \square), q_0, (0, 0), q_0, (R, R) ), \\ ( (1, \square), q_0, (1, 1), q_0, (R, R) ), \\ ( (\square, \square), q_0, (\square, \square), p, (L, L) ), \\ ( (0, 0), p, (0, 0), p, (L, L) ), \\ ( (1, 1), p, (1, 1), p, (L, L) ), \\ ( (\square, \square), p, (\square, \square), f, (R, R) ) \end{array} \right\}$$

kopiert Inhalt des Bandes 1 auf (leeres) Band 2

allgemein: Projektion  $(x_1, \dots, x_n) \mapsto x_k$

# Kombination von TM

Idee:

Kombination verschiedener Einband-TM  $M_i$  zu einer Mehrband-TM  $M$

Einband-TM zur Berechnung von (binärcodiert)

- ▶  $M_{i,0} = (\{0, 1\}, \{q_0, p\}, \{0, 1, \square\}, \delta, q_0, \square)$  mit  
 $\delta = \{(0, q_0, \square, q_0, R), (1, q_0, \square, q_0, R), (\square, q_0, 0, p, N)\}$   
schreibt Konstante 0 auf Band  $i$
- ▶ Nachfolger vom Inhalt des Bandes  $i$  auf Band  $i$   
 $M_{i,+1} = \dots$
- ▶ Vorgänger vom Inhalt des Bandes  $i$  auf Band  $i$   
 $M_{i,-1} = \dots$
- ▶ Kopie des Inhalts des Bandes  $i$  auf Band  $j$

# Simulation von Mehrband-TM

## Satz

Zu jeder  $k$ -Band-TM  $M$  existiert eine TM  $N$  (mit einem Band) mit  $f_M = f_N$ .

Beweisidee: Zusammenfügen aller  $k$  Bänder zu einem Band (mit mehreren Spuren)

- ▶ Eingabe- und Arbeitsalphabet von  $N$  besteht aus  $k$ -Tupeln von Symbolen aus  $M$   
(simuliert Spuren,  $i$ -te Komponente ist Symbol auf Band  $i$ )
- ▶ zusätzliche Zustände zur Verschiebung der Inhalte einzelner Komponenten (Spuren)
- ▶ nach jedem Übergang:  
Synchronisierung der Positionen aller Schreib/Lese-Köpfe durch Verschiebung der Bandinhalte (in mehreren Schritten)
  - ▶ um eine Zelle nach links bei Bewegung des Kopfes nach rechts
  - ▶ um eine Zelle nach rechts bei Bewegung des Kopfes nach links

# Elementare Turing-berechenbare Funktionen

Turing-berechenbar sind z.B. die folgenden Funktionen:

- ▶ jede Konstante  $x \in \mathbb{N}$  ( $w \in X^*$ )
- ▶ identische Funktion  $x \mapsto x$
- ▶ Projektion  $(x_1, \dots, x_n) \mapsto x_k$  (z.B. mit Mehrband-TM )
- ▶ Nachfolger  $x \mapsto x + 1$
- ▶ Vorgänger  $x \mapsto \max(0, x - 1)$
- ▶ Addition
- ▶ schwache Subtraktion  $(x_1, x_2) \mapsto \max(0, x_1 - x_2)$ ,  
Notation:  $x_1 \dot{-} x_2$

# Abschluss-Eigenschaften

Die Menge aller Turing-berechenbaren Funktionen ist abgeschlossen unter:

**Nacheinanderausführung** Sind  $f : \mathbb{N} \rightarrow \mathbb{N}$  und  $g : \mathbb{N} \rightarrow \mathbb{N}$  Turing-berechenbar, dann ist auch  $x \mapsto f(g(x))$  Turing-berechenbar. (Nacheinanderausführung der TM für  $g$  und  $f$ )

**Einsetzen** (Substitution)  
Sind  $f : \mathbb{N}^2 \rightarrow \mathbb{N}$ ,  $g_1, g_2 : \mathbb{N} \rightarrow \mathbb{N}$  Turing-berechenbar, dann auch  $x \mapsto f(g_1(x), g_2(x))$

## Was bisher geschah

- ▶ Turing-Maschine  $(X, Q, \Gamma, \delta, q_0, \square)$
- ▶ Simulation nichtdeterministischer durch deterministische TM (DTM)
- ▶ Mehr-Band-TM, Simulation durch Ein-Band-TM
- ▶ DTM berechnet (partielle) Funktion  $f_M : X^* \rightarrow \Gamma^*$
- ▶ Turing-berechenbare Funktionen
- ▶ **Turing** = Menge aller durch eine DTM berechenbaren Funktionen
- ▶ Existenz nicht Turing-berechenbarer Funktionen (indirekter Nachweis, Kardinalität)
- ▶ TM für Basisfunktionen:  
Konstanten, Nachfolger, Vorgänger, Test auf Bandinhalt = 0, Identität (Kopie), Projektionen
- ▶ Komposition von TM:  
Nacheinanderausführung, Einsetzen, Verzweigung

# Strukturierte Programmierung

Formalisierung des **imperativen** Programmierens

- ▶ Programm: Folge von Befehlen
- ▶ Programmausführung: Folge von Zustandsänderungen
- ▶ Zustand: Speicherbelegung und Befehlsnummer

Eigenschaften dieses Modells:

- ▶ ist einfach in Hardware realisierbar  
(seit Jahrzehnten werden Rechner so gebaut)
- ▶ ist softwaretechnisch unzweckmäßig  
(der Beweis für die Korrektheit eines Programms sieht ganz anders aus als das Programm selbst)



# Semantik: Speicher

- ▶ Speicher der Maschine besteht aus Registern (Zellen),
- ▶ Registerinhalte  $\in \mathbb{N}$ ,
- ▶ Register sind numeriert durch  $\mathbb{N}$ ,
- ▶ nur endlich viele Register werden verwendet (Rest bleibt 0).

Menge der möglichen Speicherbelegungen:

$$S := \left\{ s \in \mathbb{N}^{\mathbb{N}} \mid \{x \mid s(x) \neq 0\} \text{ ist endlich} \right\}$$

Beispiel:  $s(0) = 42, s(1) = 10, \forall x \in \mathbb{N} : x \geq 2 \rightarrow s(x) = 0$

Notation für Speicher-Änderungen:  $s[x := y]$   
ist die Funktion  $z \mapsto (\text{if } z = x \text{ then } y \text{ else } s(z))$ .

Beispiel:  $s[1 := 8](1) = \dots, s[1 := 8](2) = \dots$

ÜA: Gilt  $s[a := b][c := d] = s[c := d][a := b]$  ?

# Syntax: Befehle und Programme

Menge  $B$  der **Befehle**:

- ▶ arithmetische Befehle:
  - ▶ Inc  $\mathbb{N}$
  - ▶ Dec  $\mathbb{N}$
- ▶ Sprungbefehle:
  - ▶ Goto  $\mathbb{N}$
  - ▶ GotoZ  $\mathbb{N} \times \mathbb{N}$
- ▶ Stop

Menge  $P$  der **goto-Programme** =  $B^*$  (Folgen von Befehlen)

Beispiel für ein goto-Programm:

```
[ GotoZ 1 5, Dec 1, Inc 0, Inc 0, Goto 0, Stop ]
```

# Semantik: Befehle

Menge der Konfigurationen  $C \subseteq \mathbb{N} \times S$  mit

- ▶ **Befehlszähler**  $c \in \mathbb{N}$   
(enthält die Nr. des nächsten auszuführenden Befehls)
- ▶ **Speicherbelegung**  $s \in S$

Übergangsrelation des goto-Programms  $p$ :

$\text{step}_p \subseteq C \times C$  mit  $((c, s), (c', s')) \in \text{step}_p$ ,

falls  $c < |p|$  und ... und  $p_c =$

- ▶ Inc  $i$ , dann  $c' = c + 1, s' = s[i := s(i) + 1]$   
( $\text{step}_{\text{Inc } i} = \{((c, s), (c + 1, s[i := s(i) + 1])) \mid c \in \mathbb{N}, s \in S, i \in \mathbb{N}\}$ )
- ▶ Dec  $i$ , dann ...
- ▶ Goto  $k$ , dann ...
- ▶ GotoZ  $i$   $k$ , dann: wenn  $s(i) = 0$ , dann ... sonst ...

Satz: Die Relation  $\text{step}_p$  ist eine partielle Funktion.

(ÜA)

## Semantik: Programme

- ▶ initiale Konfiguration  $I(x)$  mit Eingabe  $x = (x_1, \dots, x_n) \in \mathbb{N}^n$ :  
 $(0, s)$  mit  $s(1) = x_1, \dots, s(n) = x_n, s(i) = 0$  sonst
- ▶ finale Konfiguration  $(c, s)$ , wobei  $c < |p|$  und  $p_c = \text{Stop}$
- ▶ die Ausgabe  $O(c, s)$  einer Konfiguration ist  $s(0)$

Jedes goto-Programm  $p$  berechnet eine partielle Funktion

$f_p : \mathbb{N}^n \rightarrow \mathbb{N}$  mit

$(x, y) \in f_p$  gdw.

- ▶  $(I(x), F) \in \text{step}_p^*$  und  $F$  ist final und  $y = O(F)$

Beispiel:  $p =$

[GotoZ 1 4, Dec 1, Goto 0, Inc 1, Inc 3, Inc 0, Stop]

berechnet  $f_p : \mathbb{N}^2 \rightarrow \mathbb{N}$  mit  $f_p(2, 1) = \dots$

# Goto-berechenbare Funktionen

Partielle Funktion  $f : \mathbb{N}^n \rightarrow \mathbb{N}$  heißt **goto-berechenbar** gdw. ein goto-Programm  $p$  mit  $f = f_p$  existiert.

**GOTO** = Menge aller goto-berechenbaren Funktionen

Beispiel: Programm  $p$ , welches die Funktion  $f_p(x_1) = 3$  berechnet?

Semantik des leeren Programms (mit  $|p| = 0$ ) ?

## goto-Programm für $x \mapsto 2x$

$p = [ \text{GotoZ } 1 \ 5 , \text{ Dec } 1 , \text{ Inc } 0 , \text{ Inc } 0 , \text{ Goto } 0 , \text{ Stop } ]$

Nachweis für  $\forall x \in \mathbb{N} : f_p(x) = 2x$  in zwei Teilen

- ▶ das goto-Programm **hält** für jede Eingabe (vgl. Def.  $\text{step}_p^*$ )
- ▶ die Ausgabe ist **korrekt**

jeder Konfiguration  $(c, s)$  wird zugeordnet:

**Invariante**  $s(0) + 2 \cdot s(1)$

**Schranke**  $s(1)$

und gezeigt (für die Teilfolge aller Konfigurationen mit  $c = 0$ ):

- ▶ die Invariante ist
  1. anfangs wahr,
  2. invariant
- ▶ die Schranke nimmt ab (um wieviel?) und bleibt  $\geq 0$ .

# Elementare goto-berechenbare Funktionen

goto-berechenbar sind z.B. die folgenden Funktionen:

- ▶ Konstante 0
- ▶ jede konstante Funktion
- ▶ identische Funktion
- ▶ jede Projektion  $(x_1, \dots, x_n) \mapsto x_k$
- ▶ Addition
- ▶ schwache Subtraktion  $(x_1, x_2) \mapsto x_1 \dot{-} x_2 (= \max(0, x_1 - x_2))$

# Abschluss-Eigenschaften

**Nacheinanderausführung:** Sind  $f : \mathbb{N} \rightarrow \mathbb{N}$  und  $g : \mathbb{N} \rightarrow \mathbb{N}$  goto-berechenbar, dann ist auch  $x \mapsto f(g(x))$  goto-berechenbar.

Beweis:

[ Prog. für  $g$  ;  $s(1) := s(0)$ ;  $s(0) := 0$ ; Prog. für  $f$  ]

Warum  $s(0) := 0$ ?

**Einsetzen:** Sind  $f : \mathbb{N}^2 \rightarrow \mathbb{N}$ ,  $g_1, g_2 : \mathbb{N} \rightarrow \mathbb{N}$  goto-berechenbar, dann auch  $x \mapsto f(g_1(x), g_2(x))$ .

einfach [ Programm für  $g_1$ , Programm für  $g_2, \dots$  ] ?



## GOTO $\subseteq$ Turing

Satz:  $\forall f : \mathbb{N}^k \rightarrow \mathbb{N}$ :  $f$  goto-berechenbar  $\rightarrow f$  Turing-berechenbar.

Beweis (Idee):

Ausführung des goto-Programmes  $p$  zur Berechnung von  $f = f_p$   
auf einer Mehrband-TM  $M$

- ▶ in  $p$  werden  $k$  Register verwendet –  $M$  hat  $k$  Arbeits-Bänder
- ▶ Register  $i$  mit Inhalt  $x$  – Arbeits-Band  $i$  hat Inhalt  $1^x$
- ▶ Ausführung von `Inc i`, `Dec i` durch  $M$  auf Band  $i$   
(Unterprogramme)
- ▶ Programmablaufsteuerung:  
Befehlsnummer (PC) wird im Zustand der TM verwaltet
- ▶ für jede Programmzeile eine Zustandsmenge zur Ausführung  
des entsprechenden Unterprogrammes der TM
- ▶ Realisierung der Sprungbefehle durch Zustandsübergänge

ÜA: Ergänzung der Details zur Herstellung der Initialkonfiguration,  
Ablesen des Resultates aus Finalkonfiguration

# Turing $\subseteq$ GOTO

Satz:  $\forall f : \mathbb{N}^k \rightarrow \mathbb{N}$ :  $f$  TM-berechenbar  $\rightarrow f$  goto-berechenbar.

Beweis (Idee):

- ▶ Jedes Band der  $k$ -Band-TM wird simuliert durch zwei Register  $(l, r) \in \mathbb{N}^2$
- ▶ Jede Zahl  $n \in \mathbb{N}$  ist Wort zur Basis  $b = 1 + |X|$  (oder  $= |X|$ ).
- ▶  $l$  enthält den Bandinhalt links vom Kopf,  $r$  enthält den Bandinhalt rechts vom Kopf (gespiegelt).  
Kopfposition = erste Stelle in  $r^R$  = letzte Stelle in  $l$   
Beispiel: Für  $\underbrace{1010}_l p \underbrace{011}_{r^R}$  (mit  $b = 2$ ) ist  $l = 10, r = 6$
- ▶ Zeichen am Kopf lesen:  $r \bmod b$ , Beispiel:  $r \bmod b = 0$
- ▶ Zeichen  $x$  schreiben und Kopf nach rechts bewegen  $((a, p, x, q, R))$  :  
 $l' := b \cdot l + x$ ,  $r' = \lfloor r/b \rfloor$   
Beispiel:  $(0, p, 1, q, R) 1010p011 \vdash 10101q11$  mit  
 $l' := b \cdot l + x = 2 \cdot 10 + 1 = 21$  (10101),  $r' = \lfloor r/b \rfloor = \lfloor 6/2 \rfloor = 3$  (11)  
analog für Übergänge mit Kopfbewegung  $L, N$  (ÜA)
- ▶ diese Rechnungen werden in einem zusätzlichen Register ausgeführt  
insgesamt:  $k$  Bänder  $\mapsto 2k + 1$  Register

# Zusammenfassung GOTO

- ▶ formalisiert maschinennahe imperative Programmierung (Programmausführung als Folge von Speicherzustandsänderungen)
- ▶ der Befehlssatz ist klein, die Ausdruckskraft scheint gering, aber
- ▶ einfache Basis-Funktionen sind in GOTO (Konstanten, Verschieben von Registerinhalten, arithmetische Operationen)
- ▶ GOTO ist abgeschlossen bzgl.
  - ▶ Nacheinanderausführung,
  - ▶ Einsetzen
- ▶ GOTO = Turing (beide Inklusionen, jeweils Beweisidee)

# Was bisher geschah

**Turing** = Menge aller durch eine DTM berechenbaren Funktionen

goto-Programme:

- ▶ Syntax:
  - ▶ Befehle  $B$ : Inc  $i$ , Dec  $i$ , Goto  $k$ , GotoZ  $i$   $k$ , Stop
  - ▶ goto-Programm  $p \in B^*$  (endliche Folge von Befehlen)
- ▶ Semantik:
  - ▶ Konfigurationen  $C \subset \mathbb{N} \times S$   
(Befehlszähler, Speicherbelegung)
  - ▶ Übergangsrelation  $\text{step}_p \subseteq C \times C$
  - ▶ Start-, Finalkonfigurationen

goto-Programm  $p$  berechnet partielle Funktion  $f_p : \mathbb{N}^n \rightarrow \mathbb{N}$

- ▶ goto-Programme für Basisfunktionen:  
Konstanten, Projektionen, Nachfolger, Vorgänger,  
Test auf Speicherinhalt  $s(i) = 0$
- ▶ Verknüpfung von goto-Programmen durch Operationen:  
Verkettung, Verzweigung

**GOTO** = Menge aller durch goto-Programm berechenbaren Funktionen

Satz: **GOTO** = **Turing**

# Strukturierte Programme – Motivation

Goto-Programme sind flach (Listen von Befehlen),  
haben keine sichtbare Struktur.  
Das ist gut für die Hardware, schlecht für den Programmierer.

Struktur = Hierarchie = Baum

Programme sind ab jetzt Bäume. (entspricht etwa  
dem Schritt von Assembler/Fortran zu Algol,  $\approx$  1960)

Bemerkung:

Das ist immer noch imperative Programmierung,  
also immer noch unübersichtlich für den Programmierer  
(Semantikdefinition verwendet Maschinenzustand,  
dieser ist im Programm nicht sichtbar).

Ausweg: funktionale Programmierung (kein Zustand).

# Syntax

Menge der While-Programme  $P$ :

- ▶ elementare: **Inc**  $i$ , **Dec**  $i$  mit  $i \in \mathbb{N}$ ,  
leeres Programm: **Skip**
- ▶ zusammengesetzte:
  - ▶ Nacheinanderausführung: **Seq**( $p, q$ ) mit  $p, q \in P$
  - ▶ Verzweigung: **IfZ**( $i, p, q$ ) mit  $i \in \mathbb{N}, p, q \in P$
  - ▶ Wiederholung: **While**( $i, p$ ) mit  $i \in \mathbb{N}, p \in P$
- ▶ (kein Stop, kein Goto)

Beispiel:

- ▶ **While**(1, **Seq**(**Dec**(1), **Inc**(0))).
- ▶ autotool-Syntax: **While** 1 (**Seq** (**Dec** 1) (**Inc** 0))

# Semantik (Prinzip)

Semantik eines Programms  $p \in P$   
ist eine Relation (genauer: partielle Funktion)  
 $\text{sem}_p \subseteq S \times S$  auf Speicherbelegungen.

*big step semantics* (ein Schritt!)

beachte:

es gibt keinen *program counter*, diese Rolle übernimmt der Index  $p$ .

# Semantik von While-Programmen

elementare Programme:

$$\begin{aligned}\text{sem}_{\text{Skip}} &= I_{(\mathbb{N}^{\mathbb{N}})} = \{(s, s) \mid s : \mathbb{N} \rightarrow \mathbb{N}\} \\ \text{sem}_{\text{Inc}(i)} &= \{(s, s[i := s(i) + 1]) \mid s : \mathbb{N} \rightarrow \mathbb{N}\} \\ \text{sem}_{\text{Dec}(i)} &= \{(s, s[i := \max(0, s(i) - 1)]) \mid s : \mathbb{N} \rightarrow \mathbb{N}\}\end{aligned}$$

zusammengesetzte Programme  $p$ :

$$\begin{aligned}\text{sem}_{\text{Seq}(p_1, p_2)} &= \text{sem}_{p_1} \circ \text{sem}_{p_2} \\ &= \{(s_1, s_2) \mid \exists s' : (s_1, s') \in \text{sem}_{p_1} \wedge (s', s_2) \in \text{sem}_{p_2}\} \\ \text{sem}_{\text{IfZ}(i, p_1, p_2)} &= \left\{ (s_1, s_2) \mid \begin{array}{l} (s_1(i) = 0 \wedge (s_1, s_2) \in \text{sem}_{p_1}) \\ \vee (s_1(i) > 0 \wedge (s_1, s_2) \in \text{sem}_{p_2}) \end{array} \right\} \\ \text{sem}_{\text{While}(i, q)} &= \left\{ (s_1, s_2) \mid \begin{array}{l} (s_1(i) = 0 \wedge (s_1 = s_2)) \\ \vee (s_1(i) > 0 \wedge (s_1, s_2) \in \text{sem}_{\text{Seq}(q, p)}(s_1, s_2)) \end{array} \right\}\end{aligned}$$



# Äquivalenz von Programmen

Programme  $p$  und  $q$  sind äquivalent ( $p \equiv q$ ) gdw.  $\text{sem}_p = \text{sem}_q$ .

Beispiele:

- ▶  $\text{Seq}(\text{Skip}, p) \equiv p \equiv \text{Seq}(p, \text{Skip})$

für erste Äquivalenz z.z.:  $\text{sem}_{\text{Seq}(\text{Skip}, p)} = \text{sem}_p$

$$\begin{array}{ccc} \text{sem}_{\text{Seq}(\text{Skip}, p)} & \stackrel{(\text{Seq})}{=} & \text{sem}_{\text{Skip}} \circ \text{sem}_p \\ & \stackrel{(\text{Skip})}{=} & I_{\mathbb{N}^{\mathbb{N}}} \circ \text{sem}_p \\ & \stackrel{I_{\mathbb{N}^{\mathbb{N}}} \text{ bzgl. } \circ \text{ neutral}}{=} & \text{sem}_p \end{array}$$

- ▶  $\text{Seq}(\text{Seq}(p, q), r) \equiv \text{Seq}(p, \text{Seq}(q, r))$
- ▶  $\text{While}(i, p) \equiv \text{IfZ}(i, \text{Skip}, \text{Seq}(p, \text{While}(i, p)))$

ÜA: IfZ wird nicht benötigt, lässt sich simulieren

# While-berechenbare Funktionen

- ▶ initiale Speicherbelegung  $I(x)$  für Eingabe  $x \in \mathbb{N}^n$ :  
 $s(i) =$  wenn  $1 \leq i \leq n$ , dann  $x_i$ , sonst 0.
- ▶ Programm  $p$  berechnet partielle Funktion  $f : \mathbb{N}^n \rightarrow \mathbb{N}$  :  
 $\forall x \in \mathbb{N}^n : f(x) = y$  gdw.  $\exists s : \text{sem}_p(I(x), s) \wedge y = s(0)$ .
- ▶ jede so berechenbare partielle Fkt. heißt While-berechenbar.

WHILE = Menge aller While-berechenbaren Funktionen

- ▶ Die üblichen elementaren Funktionen sind  $\in$  WHILE.  
(Autotool)
- ▶ WHILE ist abgeschlossen unter Substitution, z.B.  
 $(f \in \text{WHILE} \wedge g \in \text{WHILE}) \rightarrow (x \mapsto f(g(x))) \in \text{WHILE}$  (ÜA)

# Beziehungen zwischen Goto- und While-Berechenbarkeit

Ziel: WHILE = GOTO

äquivalente Aussage:

Für jede partielle Funktion  $f$  gilt:

$f$  ist While-berechenbar gdw.  $f$  ist Goto-berechenbar.

praktisches Argument für „ $\subseteq$ “:

ist genau die Aufgaben von Compilern

(etwa von C nach Assembler/Maschinsprache)

(siehe Modul „Compilerbau“)

# WHILE $\subseteq$ GOTO (Prinzip)

Übersetzer:

$\text{compile} : \mathbb{N} \times \text{WHILE} \rightarrow \mathbb{N} \times \text{GOTO}$

wobei  $\text{compile}(a, p) = (e, q)$  bedeutet:

- ▶ das While-Programm  $p$  wird übersetzt
- ▶ in ein äquivalentes Goto-Programm  $q$ ,
- ▶ das auf Adresse  $a$  beginnt
- ▶ und auf  $e - 1$  endet (d.h.  $e = a + |q|$ )

dabei bedeutet „Äquivalenz“:

$\forall p \forall a \in \mathbb{N}$  mit  $(e, q) = \text{compile}(a, p)$ :

$\forall s_1, s_2 : (s_1, s_2) \in \text{sem}_p \leftrightarrow ((a, s_1), (e, s_2)) \in \text{step}_q^*$

# WHILE $\subseteq$ GOTO (elementar, Seq)

einfache Programme:

- ▶  $\text{compile}(a, \text{Skip}) = (a, [])$
- ▶  $\text{compile}(a, \text{Inc}(i)) = (a + 1, [\text{Inc}(i)])$ .
- ▶  $\text{compile}(a, \text{Dec}(i)) = (a + 1, [\text{Dec}(i)])$ .

zusammengesetzte Programme:

- ▶  $\text{compile}(a, \text{Seq}(p, q)) = (e, p' \circ q')$ , wobei  
 $\text{compile}(a, p) = (m, p')$  und  $\text{compile}(m, q) = (e, q')$

## WHILE $\subseteq$ GOTO (While, IfZ)

IfZ:

```
compile (_, IfZ i p1 p2) =>
  A: GotoZ i M
    compile (_, p2) ;
    Goto E ;
  M: compile (_, p1);
  E: ...
```

While:

```
compile (_, While i p) =>
  A: GotoZ i E
    compile (A+1, p) ;
    Goto A ;
  E: ...
```

## WHILE $\subseteq$ GOTO – Korrektheit Inc $i$

Diese Definition von  $\text{compile}(a, p)$  erfüllt die Spezifikation:

$\forall p \in \text{While}, a \in \mathbb{N}$  gilt für  $(e, q) = \text{compile}(a, p)$ :

$\forall s, s' : (s, s') \in \text{sem}_p$  gdw.  $((a, s), (e, s')) \in \text{step}_q^*$

Beweis der Korrektheit für  $p = \text{Inc } i$ :

$\text{compile}(a, p) = (e, q)$  mit  $e = a + 1$  und  $q = [\text{Inc } i]$

$$\begin{aligned} \forall s, s' : & \quad (s, s') \in \text{sem}_p \\ \text{gdw.} & \quad (s, s') \in \text{sem}_{\text{Inc } i} \\ \text{gdw.} & \quad s' = s[i := s(i) + 1] \\ \text{gdw.} & \quad (a, s) = (\underbrace{a + 1}_e, \underbrace{s[i := s(i) + 1]}_{s'}) \in \text{step}_{[\text{Inc } i]}^* \\ \text{gdw.} & \quad (a, s) = (e, s') \in \text{step}_q^* \end{aligned}$$

## WHILE $\subseteq$ GOTO – Korrektheit While $i$ $p$

Beweis der Korrektheit für  $p = \text{While } i \ p'$ :

$\text{compile}(a, p) = (e, q)$  mit  $q = [\text{GotoZ } i \ e] \circ q' \circ [\text{Goto } a]$ ,

wobei  $\text{compile}(a + 1, p') = (e', q')$

- ▶  $\forall s, s'$  mit  $s(i) = 0$

$$\begin{aligned} & (s, s') \in \text{sem}_p \text{ gdw. } (s, s') \in \text{sem}_{\text{While } i \ p'} \\ \text{gdw. } & \text{gdw. } s' = s \text{ gdw. } (a, s) = (e, s) \in \text{step}_{[\text{GotoZ } i \ e]} \\ \text{gdw. } & (a, s) = (e, s) \in \text{step}_{[\text{GotoZ } i \ e] \circ q' \circ [\text{Goto } a]}^* \\ \text{gdw. } & (a, s) = (e, s) \in \text{step}_q^* \end{aligned}$$

- ▶  $\forall s, s'$  mit  $s(i) > 0$ :

$$\begin{aligned} & (s, s') \in \text{sem}_p \text{ gdw. } (s, s') \in \text{sem}_{\text{While } i \ p'} \\ \text{gdw. } & (s, s') \in \text{sem}_{\text{Seq}(p', p)} \\ \text{gdw. } & (s, s') \in \text{sem}_{p'} \circ \text{sem}_p \\ \text{gdw. } & \exists s'' : (s, s'') \in \text{sem}_{p'} \wedge (s'', s') \in \text{sem}_p \\ \text{gdw. } & \exists (e', s'') : ((a, s), (e', s'')) \in \text{step}_{q'}^* \wedge ((e' + 1, s''), (e, s')) \in \text{step}_q^* \\ \text{gdw. } & \dots \\ \text{gdw. } & ((a, s), (e, s)) \in \text{step}_q^* \end{aligned}$$



## WHILE $\subseteq$ GOTO (insgesamt)

Satz:

Für jedes While-Programm  $p$  existiert ein Goto-Programm  $q$ ,  
welches dieselbe partielle Funktion berechnet wie  $p$ .

Beweis(-plan):

- ▶  $q = \text{compile}(0, p) \circ [\text{Stop}]$
- ▶ Aussage folgt aus Korrektheit bzgl. der Spezifikation  
 $\forall p \in \text{WHILE}, a \in \mathbb{N}$  gilt für  $(e, q) = \text{compile}(a, p)$ :

$$\forall s_1, s_2 : (s_1, s_2) \in \text{sem}_p \quad \leftrightarrow \quad ((a, s_1), (e, s_2)) \in \text{step}_q^* \quad (\ddot{\text{U}}\text{A})$$

## Was bisher geschah

Turing = Menge aller durch eine DTM berechenbaren Funktionen

GOTO = Menge aller durch Goto-Programme berechenbaren Funktionen

- ▶ Syntax: Goto-Program = Liste von Befehlen aus  
 $B = \{\text{Inc } i, \text{Dec } i, \text{Goto } k, \text{GotoZ } i \ k, \text{Stop}\}$
- ▶ Semantik: small step  
Konfigurationen:  $(c, s)$  mit  $c \in \mathbb{N}, s \in \mathbb{N}^{\mathbb{N}}$  (PC, Speicherbelegung)  
Konfigurationsübergänge:  $\text{step}_p$  für Befehl,  $\text{step}_p^*$  für Programm

Satz: Turing = GOTO

WHILE = Menge aller durch While-Programme berechenbaren Funktionen

- ▶ Syntax: While-Program = Baum mit  
Blättern:  $\text{Inc } i, \text{Dec } i, \text{Skip}$   
inneren Knoten:  $\text{While } i \ p, \text{Seq } p \ q, \text{IfZ } i \ p \ q$
- ▶ Semantik: big step  
Konfigurationen  $s \in \mathbb{N}^{\mathbb{N}}$  (Speicherbelegung)  
Konfigurationsübergänge  $\text{sem}_p \subseteq \mathbb{N}^{\mathbb{N}} \times \mathbb{N}^{\mathbb{N}}$  für Programm

(halber) Satz: WHILE  $\subseteq$  GOTO

# GOTO $\subseteq$ WHILE

das scheint schwieriger:

- ▶ goto-Programm = Spaghetti-Code,
- ▶ while-Programm = strukturierter Code.

Es ist aber möglich und das erzeugte While-Programm hat eine besondere (einfache) Struktur, die später noch ausgenutzt wird (Kleene-Normalform-Theorem)

## GOTO $\subseteq$ WHILE (Ansatz)

Eingabe: Goto-Programm  $p$ ,

Ausgabe: äquivalentes While-programm  $q$

PC  $c$  = erstes in  $p$  nicht benutzte Register,

Das nächste Register  $h$  verwenden wir zum Anhalten.

Struktur von  $q$  ist:

```
Inc h;
```

```
While (h) {
```

```
  if (c == 0) { ... } else {
```

```
    if (c == 1) { ... } else {
```

```
      if (c == 2) { ... } else {
```

```
        ..                               else Skip
```

```
}
```

## GOTO $\subseteq$ WHILE (Einzelheiten)

für Befehl  $p_i$  erzeuge:  $\text{if } (c == i) \text{ } q_i$  else mit  $q_i =$

- ▶ wenn  $p_i \in \{\text{Inc } r, \text{Dec } r\}$ , dann  $\text{Seq}(p_i, \text{Inc } c)$
- ▶ wenn  $p_i = \text{Stop}$ , dann  $\text{Dec } h$
- ▶ wenn  $p_i = \text{Goto}(l)$ , dann  $c := l$ ,
- ▶ wenn  $p_i = \text{GotoZ}(r, l)$ , dann  $\text{IfZ } r \text{ } (c := 1) \text{ } (\text{Inc } c)$

$p$  erreicht Stop gdw.  $q$  hält. (ÜA)

Man beachte dabei auch den Fall  $\text{Goto } l$  mit  $l \geq |p|$ .

Hier wird  $\text{if } (c == i)$  und  $c := 1$  benutzt,  
Implementierung durch While-Programme (ÜA)  
(hier auch ohne Schleife möglich, warum?)

# Normalform-Theorem für While

Vorige Konstruktion zeigt den  
Satz:

- ▶ Zu jedem Goto-Programm existiert ein äquivalentes While-Programm ( $GOTO \subseteq WHILE$ )
- ▶ mit **genau einem** While.

zusammen mit  $WHILE \subseteq GOTO$  folgt

Satz:  $WHILE = GOTO$

Satz (Kleene-Normalform für While-Programme):  
Zu jedem While-Programm gibt es ein äquivalentes  
While-Programm mit genau einem While.

„äquivalent“ = berechnet dieselbe partielle Funktion.

## Loop-Programme – Motivation

- ▶  $\text{While}(i, q)$  bedeutet: solange  $s(i) > 0$  ist,  $q$  ausführen
- ▶ es gibt While-Programme, die nicht für jede Eingabe terminieren (es gibt  $f \in \text{WHILE}$  mit  $f$  nicht total)
- ▶  $\text{Loop}(i, q)$  bedeutet:  $q$  genau  $s(i)$  mal ausführen (der Wert von  $i$  vor Beginn der Schleife)
- ▶ Loop-Programme terminieren (jede  $f \in \text{LOOP}$  ist total)  
Das ist softwaretechnisch nützlich.
- ▶ aber auch eine Einschränkung:  
es gibt  $f \in (\text{WHILE} \cap \text{TOTAL}) \setminus \text{LOOP}$

# Loop-Programme

Syntax und Semantik wie While-Programme, außer:

**Syntax**  $\text{Loop}(i, q)$  statt  $\text{While}(i, q)$ ,

**Semantik** wenn  $p = \text{Loop}(i, q)$ ,

dann  $\text{sem}_p(s, s') = \text{sem}_q^{s(i)}(s, s')$

Befehl  $q$  wird genau  $s(i)$  mal ausgeführt

(der Wert von  $i$ , wenn die Schleife zu erstmal  
betreten wird — egal, was später mit  $i$  passiert)

Beispiel:  $\text{Seq}(\text{Loop}(1, \text{Inc } 1), \text{Loop}(2, \text{Inc } 0))$

Jede so berechenbare Funktion heißt Loop-berechenbar.

LOOP = Menge aller Loop-berechenbaren Funktionen

Beispiele: Addition, Subtraktion, Multiplikation, Potenz,  
 $n \mapsto n$  ist gerade,  $n \mapsto n$  ist Quadratzahl,  $n \mapsto n$  ist prim,...



## Beispiel: Semantik von Loop-Programmen

$$p = \text{Seq}(\text{Loop } 1 \text{ (Inc } 1))(\text{Loop } 2 \text{ (Inc } 0))$$

$$\begin{aligned} \text{sem}_p &= \text{sem}_{\text{Loop } 1 \text{ (Inc } 1)} \circ \text{sem}_{\text{Loop } 2 \text{ (Inc } 0)} \\ &= \text{sem}_{\text{Inc } 1}^{s(1)} \circ \text{sem}_{\text{Inc } 0}^{s(2)} \\ &= \{(s, s[1 := s(1) + 1]) \mid s \in \mathbb{N}^{\mathbb{N}}\}^{s(1)} \\ &\quad \circ \{(s', s'[0 := s'(0) + 1]) \mid s' \in \mathbb{N}^{\mathbb{N}}\}^{s(2)} \\ &= \{(s, s[1 := s(1) + s(1)]) \mid s \in \mathbb{N}^{\mathbb{N}}\} \\ &\quad \circ \{(s', s'[0 := s'(0) + s(2)]) \mid s' \in \mathbb{N}^{\mathbb{N}}\} \\ &= \{(s, s[1 := s(1) + s(1)][0 := s(0) + s(2)]) \mid s \in \mathbb{N}^{\mathbb{N}}\} \end{aligned}$$

$p$  berechnet  $f : \mathbb{N}^2 \rightarrow \mathbb{N}$  mit  $\forall(x_1, x_2) : f(x_1, x_2) = x_2$   
(mit Nebenwirkung  $s(1) \mapsto 2s(1)$ )

Modifikation:

$$q = \text{Seq}(\text{Loop } 1 \text{ (Inc } 0))(\text{Loop } 2 \text{ (Inc } 0))$$

$q$  berechnet  $f : \mathbb{N}^2 \rightarrow n$  mit  $\forall(x_1, x_2) : f(x_1, x_2) = x_1 + x_2$

# LOOP $\subseteq$ WHILE

$$\text{Loop}(i, p) \mapsto q = \text{Seq}(\text{Seq}(\text{While}(i, \text{Seq}(\text{Seq}(\text{Inc } j, \text{Inc } k), \text{Dec } i)), \text{While}(k, \text{Seq}(\text{Inc } i, \text{Dec } k))), \text{While}(j, \text{Seq}(p, \text{Dec } j))))$$

mit in  $p$  nicht verwendeten Registern  $j, k$

$$\begin{aligned} \text{sem}_q &= \text{sem}_{\text{While}(i, \text{Seq}(\text{Seq}(\text{Inc } j, \text{Inc } k), \text{Dec } i))} \\ &\quad \circ \text{sem}_{\text{While}(k, \text{Seq}(\text{Inc } i, \text{Dec } k))} \circ \text{sem}_{\text{While}(j, \text{Seq}(p, \text{Dec } j))} \\ &= \{(s, s[j := s(i), k := s(i), i := 0]) \mid s \in \mathbb{N}^{\mathbb{N}}\} \\ &\quad \circ \{(s, s[i := s(k), k := 0]) \mid s \in \mathbb{N}^{\mathbb{N}}\} \\ &\quad \circ \text{sem}_{\text{While}(j, \text{Seq}(p, \text{Dec } j))} \\ &= \{(s, s[j := s(i)]) \mid s \in \mathbb{N}^{\mathbb{N}}\} \circ \text{sem}_{\text{While}(j, \text{Seq}(p, \text{Dec } j))} \\ &= \{(s, s[j := s(i)]) \mid s \in \mathbb{N}^{\mathbb{N}}\} \circ \text{sem}_{\text{Seq}(p, \text{Dec } j)}^{s(j)} \quad (j \text{ kommt nicht in } p \text{ vor}) \\ &= \{(s, s[j := s(i)]) \mid s \in \mathbb{N}^{\mathbb{N}}\} \circ (\text{sem}_p \circ \text{sem}_{\text{Dec } j})^{s(j)} \\ &= \{(s, s[j := s(i)]) \mid s \in \mathbb{N}^{\mathbb{N}}\} \circ \text{sem}_p^{s(i)} \circ \{(s', s'[j := 0]) \mid s \in \mathbb{N}^{\mathbb{N}}\} \\ &= \text{sem}_p^{s(i)} = \text{sem}_{\text{Loop}(i, p)} \end{aligned}$$

# LOOP und WHILE

schon gezeigt:  $\text{LOOP} \subseteq \text{WHILE} \cap \text{TOTAL}$

$\text{WHILE} \cap \text{TOTAL} \not\subseteq \text{LOOP}$

Beweis: Diagonalisierung

- ▶  $p_0, p_1, \dots$  (z.B. quasi-lexikografische) Aufzählung aller LOOP-Programme, die einstellige Funktionen berechnen,  $f_0, f_1, \dots$
- ▶ für  $d : x \mapsto f_x(x) + 1$  gilt:  $d \in \text{WHILE} \cap \text{TOTAL}$   
Begründung: Interpreter für LOOP-Programme (universelle TM / While-Programm)
- ▶ Diese Funktion  $d$  kommt in der Aufzählung  $f_0, f_1, \dots$  nicht vor (also  $d \notin \text{LOOP}$ )

Begründung (indirekt):

- ▶ Annahme  $d \in \text{LOOP}$
- ▶ damit existiert ein  $k \in \mathbb{N}$  mit  $d = f_k$
- ▶ Widerspruch bei  $d(k) = f_k(k)$  (nach Def. von  $f_k$ ) und  $d(k) = f_k(k) + 1$  (nach Def. von  $d$ )
- ▶ also ist die Annahme falsch und  $d \notin \text{LOOP}$

# Was bisher geschah

- ▶ Goto-Programme
  - ▶ Syntax: = Liste von Befehlen aus  
 $B = \{\text{Inc } i, \text{Dec } i, \text{Goto } k, \text{GotoZ } i \ k, \text{Stop}\}$
  - ▶ Semantik: small step  
Konfigurationen  $(l, s)$  mit  $l \in \mathbb{N}, s \in \mathbb{N}^{\mathbb{N}}$  (PC, Speicherbeleg.)  
Konfigurationsübergänge  $\text{step}_p$  für Befehl,  $\text{step}_p^*$  für Programm
- ▶ While/Loop-Programme
  - ▶ Syntax: Baum mit Blättern aus  $\{\text{Inc } i, \text{Dec } i, \text{Skip}\}$   
inneren Knoten  $\text{While } i \ p / \text{Loop } i \ p, \text{Seq } p \ q, \text{IfZ } i \ p \ q$
  - ▶ Semantik: big step,  
Konfigurationen  $s \in \mathbb{N}^{\mathbb{N}}$  (Speicherbelegung)  
Konfigurationsübergänge  $\text{sem}_p \subseteq (\mathbb{N}^{\mathbb{N}})^2$  für Programm

Turing	=	Menge aller durch eine DTM berechenbaren Funktionen
GOTO	=	Menge aller durch ein Goto-Programm berechenbaren Fkt.
WHILE	=	Menge aller durch ein While-Programm berechenbaren Fkt.
LOOP	=	Menge aller durch ein Loop-Programm berechenbaren Fkt.
TOTAL	=	Menge aller für jede Eingabe ( $\in \mathbb{N}^{(k)}$ ) definierter Fkt.

Sätze: Turing = GOTO = WHILE, LOOP  $\subset$  WHILE  $\cap$  TOTAL

# Ackermann-Funktion

(Wilhelm Friedrich Ackermann, 1928)

$A : \mathbb{N}^2 \rightarrow \mathbb{N}$  mit  $\forall (x, y) \in \mathbb{N}^2 :$

$$A(0, y) = y + 1$$

$$A(x + 1, 0) = A(x, 1)$$

$$A(x + 1, y + 1) = A(x, A(x + 1, y))$$

Beispiele (Tafel):  $A(0, 0)$ ,  $A(0, 1)$ ,  $A(0, 2)$ ,  
 $A(1, 0)$ ,  $A(1, 1)$ ,  $A(1, 2)$ ,  $A(2, 1)$ ,  $A(2, 2)$

- ▶ bestimme  $A(2, 4)$ ,  $A(3, 3)$ ,  $A(4, 2)$

(ÜA)

# Eigenschaften der Ackermann-Funktion

1.  $\forall x, y \in \mathbb{N} : A(x, y) > y$   
Beweis durch Induktion nach  $x$ , innen Induktion nach  $y$
2.  $\forall x, y \in \mathbb{N} : A(x, y + 1) > A(x, y)$   
(Monotonie im zweiten Argument)  
Beweis (ohne Induktion) Fallunterscheidung  $x = 0$ ,  $x > 0$
3.  $\forall x, y \in \mathbb{N} : A(x + 1, y) \geq A(x, y + 1)$   
Beweis durch Induktion nach  $y$  (ÜA)
4.  $\forall x, y \in \mathbb{N} : A(x + 1, y) > A(x, y)$   
(Monotonie im ersten Argument)
5.  $\forall x, x', y, y' \in \mathbb{N} : ((x \leq x' \wedge y \leq y') \rightarrow (A(x, y) \leq A(x', y')))$   
(Monotonie in beiden Argumenten)

# Eigenschaften der Ackermann-Funktion (1)

1.  $\forall x, y \in \mathbb{N} : A(x, y) > y$

Beweis durch Induktion nach  $x$ , innen Induktion nach  $y$ :

**IAx:**  $x = 0: \forall y \in \mathbb{N} : A(0, y) = y + 1 > y$

**ISx:** IH  $x: \forall y \in \mathbb{N} : A(x, y) > y$

IB  $x: \forall y \in \mathbb{N} : A(x + 1, y) > y$

Beweis  $x: \forall y \in \mathbb{N} : A(x + 1, y) > y$  durch Induktion über  $y$

**IAy:**  $y = 0: A(x + 1, 0) = A(x, 1) \stackrel{(IHx)}{>} 1 > 0$

**ISy:** IH  $y: A(x + 1, y) > y$

IB  $y: A(x + 1, y + 1) > y + 1$

Beweis  $y$ :

$$A(x + 1, y + 1) = A(x, A(x + 1, y)) \stackrel{(IHx)}{>} A(x + 1, y) \stackrel{(IHy)}{>} y$$

und damit  $A(x + 1, y + 1) > y + 1$

## Eigenschaften der Ackermann-Funktion (2)

$$2. \forall x, y \in \mathbb{N} : A(x, y + 1) > A(x, y)$$

Beweis (ohne Induktion) Fallunterscheidung  $x = 0$  ,  $x > 0$

$$\text{a } x = 0: A(0, y + 1) = y + 2 > y + 1 = A(0, y)$$

$$\text{b } x > 0: A(x, y + 1) = A(x - 1, A(x, y)) \stackrel{1.}{>} A(x, y)$$



## Eigenschaften der Ackermann-Funktion (2)

$$\forall x, y \in \mathbb{N} : A(x + 1, y) > A(x, y)$$

Beweis:

$$A(x + 1, y) \stackrel{3}{\geq} A(x, y + 1) \stackrel{2}{\geq} A(x, y)$$

1-4 ergeben zusammengefasst:

Monotonie in beiden Argumenten:

$$\forall x, x', y, y' \in \mathbb{N} : ((x \leq x' \wedge y \leq y') \rightarrow (A(x, y) \leq A(x', y')))$$

# Wachstum von Funktionen in LOOP

Loop-Programm  $p$  (berechnet Funktion  $\mathbb{N} \rightarrow \mathbb{N}$ )

Menge aller in  $p$  vorkommenden Register (Variablen):  $\text{var}(p)$

$\forall x \in \text{var}(p)$ :

- ▶  $x_a$  Wert im Register  $x$  zu Beginn der Ausführung von  $p$
- ▶  $x_e$  Wert im Register  $x$  nach Ende der Ausführung von  $p$   
(Warum ist  $x_e$  immer definiert?)

Jedes Loop-Programm  $p$  definiert eine Funktion  $g_p : \mathbb{N} \rightarrow \mathbb{N}$  durch

$$\forall n \in \mathbb{N} : g_p(n) = \max \left\{ \sum_{x \in \text{var}(p)} x_e \mid \sum_{x \in \text{var}(p)} x_a \leq n \right\}$$

# Wachstum von Funktionen in LOOP (IA für Inc)

Satz:  $\forall p \in \text{Loop} \exists k \in \mathbb{N} \forall n \in \mathbb{N} : g_p(n) < A(k, n)$

Beweis durch Induktion über Struktur von  $p$ :

IA: Skip, Inc, Dec

$p = \text{Inc } i$ : Ansatz  $k = 1$

$$\text{sem}_p = \{(s, s[i := s(i) + 1]) \mid s \in \mathbb{N}^{\mathbb{N}}\}$$

$$n = \sum_{x \in \text{var}(p)} s(x) = s(i) + \sum_{x \in \text{var}(p) \setminus \{i\}} s(x)$$

$$\begin{aligned} g_p(n) &= \sum_{x \in \text{var}(p)} s[i := s(i) + 1](x) = 1 + s(i) + \sum_{x \in \text{var}(p) \setminus \{i\}} s(x) \\ &= 1 + \sum_{x \in \text{var}(p)} s(x) = 1 + n < n + 2 = A(1, n) \end{aligned}$$

Skip, Dec analog mit  $k = 0$

IS: Seq( $p, q$ ) (mit  $k = \max(k_p - 1, k_q) + 2$ )

Loop i p (mit  $k = k_p + 1$ )

# Wachstum von Funktionen in LOOP (IS für Seq)

z.z: Programm Seq( $p, q$ )

Ansatz:  $k = \max(k_p - 1, k_q) + 2$

Beweis:

- ▶ IH:  $\exists k_p \forall n \in \mathbb{N} : g_p(n) < A(k_p, n)$  und  
 $\exists k_q \forall n \in \mathbb{N} : g_q(n) < A(k_q, n)$
- ▶ IB:  $\exists k \forall n \in \mathbb{N} : g_{\text{Seq}(p,q)}(n) \leq A(k, n)$
- ▶ Beweis: für  $k' = \max(k_p - 1, k_q)$  gilt

$$\begin{aligned} g_{\text{Seq}(p,q)}(n) &\leq f_q(g_p(n)) \\ &\stackrel{(IH)}{<} A(k_q, A(k_p, n)) \\ &\stackrel{(5)}{\leq} A(k', A(k' + 1, n)) \\ &\stackrel{(Def.A)}{=} A(k' + 1, n + 1) \stackrel{(2)}{\leq} A(k' + 2, n) \end{aligned}$$

Für  $k = k' + 2 = \max(k_p - 1, k_q) + 2$  gilt

$\forall n \in \mathbb{N} : g_{\text{Seq}(p,q)}(n) < A(k, n)$ .

# Ackermann-Funktion $\notin$ LOOP

Satz:  $A \in \text{WHILE} \setminus \text{LOOP}$

Beweis:

- ▶  $A \notin \text{LOOP}$  (indirekt)

Annahme:  $A \in \text{LOOP}$

Dann gilt auch für die Funktion

$d : \mathbb{N} \rightarrow \mathbb{N}$  mit  $\forall n \in \mathbb{N} : d(n) = A(n, n)$ :

$d \in \text{LOOP}$ .

(ÜA: Warum?)

Also existiert ein Loop-Programm  $p$  mit  $d(n) \leq g_p(n)$

und nach Satz auf F. 102 gilt  $\exists k \forall n \in \mathbb{N} : g_p(n) < A(k, n)$

Widerspruch bei  $n = k$ :  $d(k) \leq g_p(k) < A(k, k) = d(k)$

- ▶  $A \in \text{WHILE}$

# Was bisher geschah

- ▶ Goto-Programme
  - ▶ Syntax: = Liste von Befehlen aus  
 $B = \{\text{Inc } i, \text{Dec } i, \text{Goto } k, \text{GotoZ } i \ k, \text{Stop}\}$
  - ▶ Semantik: small step  
Konfigurationen  $(l, s)$  mit  $l \in \mathbb{N}, s \in \mathbb{N}^*$  (PC, Speicherbelegung)  
Konfigurationsübergänge  $\text{step}_\rho$  für Befehl,  $\text{step}_\rho^*$  für Programm
- ▶ While/Loop-Programme
  - ▶ Syntax: Baum mit Blättern aus  $\{\text{Inc } i, \text{Dec } i, \text{Skip}\}$   
inneren Knoten  $\text{While } i \ p$  ( $\text{Loop } i \ p$ ),  $\text{Seq } p \ q$ ,  $\text{IfZ } i \ p \ q$
  - ▶ Semantik: big step  
Konfigurationen  $s \in \mathbb{N}^*$  (Speicherbelegung)  
Konfigurationsübergänge  $\text{sem}_\rho \subseteq \mathbb{N}^* \times \mathbb{N}^*$  für Programm
- ▶ Ackermann-Funktion

LOOP  $\subset$  WHILE = Turing = GOTO

# Rekursive Funktionen

(induktive) Definition der Menge aller **rekursiven Funktionen**:

IA: elementare Funktionen

- ▶ Konstante 0:  $\text{Zero}^n = (x_1, \dots, x_n) \mapsto 0$
- ▶ Nachfolger:  $\text{Succ} = (x_1) \mapsto 1 + x_1$
- ▶ Projektionen :  $\text{Proj}_k^n = (x_1, \dots, x_n) \mapsto x_k$

IS: Operatoren

- ▶ Substitution (Sub)
- ▶ primitive Rekursion (PR)
- ▶ unbeschränkte Rekursion (MIN,  $\mu$ )

# Operator: Substitution

Typ:

$$\text{Sub}_k^n : (\mathbb{N}^k \rightarrow \mathbb{N}) \times (\mathbb{N}^n \rightarrow \mathbb{N})^k \rightarrow (\mathbb{N}^n \rightarrow \mathbb{N})$$

Argumente: eine  $k$ -stellige Funktion und  $k$   $n$ -stellige Funktionen  
(evtl. partiell)

Resultat: eine  $n$ -stellige Funktion

Wert:

wenn  $f = \text{Sub}_k^n(g, h_1, \dots, h_k)$ ,

dann  $f(x_1, \dots, x_n) = g(h_1(x_1, \dots, x_n), \dots, h_k(x_1, \dots, x_n))$

Beispiele:

- ▶  $\underline{\text{Sub}}_1^0(\text{Succ}, \text{Zero}^0) = \text{Succ}(\underline{\text{Zero}}^0) = \text{Succ}(0) = 1$
- ▶  $\underline{\text{Sub}}_1^0(\text{Succ}, \underline{\text{Sub}}_1^0(\text{Succ}, \text{Zero}^0)) =$   
 $\underline{\text{Sub}}_1^0(\text{Succ}, (\text{Succ}(\underline{\text{Zero}}^0))) = \underline{\text{Sub}}_1^0(\text{Succ}, 1) = \text{Succ}(1) = 2$
- ▶  $\underline{\text{Sub}}_1^2(\text{Succ}, \text{Proj}_2^2)$   
 $f(x_1, x_2) = \text{Succ}(\text{Proj}_2^2(x_1, x_2)) = \text{Succ}(x_2) = x_2 + 1$



# Operator: Primitive Rekursion

Typ (für  $n > 0$ ):

$$\text{PR}^n : (\mathbb{N}^{n-1} \rightarrow \mathbb{N}) \times (\mathbb{N}^{n+1} \rightarrow \mathbb{N}) \rightarrow (\mathbb{N}^n \rightarrow \mathbb{N})$$

Wert:

wenn  $f = \text{PR}^n(g, h)$ , dann

$$f(x_1, \dots, x_{n-1}, 0) = g(x_1, \dots, x_{n-1})$$

$$f(x_1, \dots, x_{n-1}, 1 + y) = h(x_1, \dots, x_{n-1}, y, f(x_1, \dots, x_{n-1}, y))$$

Beispiele:

- ▶ für  $f = \text{PR}^2(\text{Proj}_1^1, \text{Sub}_1^3(\text{Succ}, \text{Proj}_3^3))$ :  
 $f(2, 0), f(0, 1), f(2, 3)$
- ▶ für  $f = \text{PR}^1(\text{Zero}^0, \text{Proj}_1^2)$ :  $f(0), f(1), f(5)$
- ▶ für  $f = \text{PR}^1(\text{Zero}^0, \text{Sub}_1^2(\text{Succ}, \text{Zero}^2))$ :  $f(0), f(1), f(5)$

# Primitive Rekursion (Beispiele)

Durch Zero, Succ, Proj, Sub, PR lassen sich u.A. darstellen:

- ▶ Addition, Multiplikation, Potenzieren, Wurzelziehen
- ▶ Vorgänger, Subtraktion, Division
- ▶  $x \mapsto$  wenn  $x$  ist prim, dann 1, sonst 0;
- ▶  $x \mapsto$  die  $x$ -te Primzahl

## $\mu$ -Operator

Typ (für  $n \geq 0$ ):

$$\mu^n : (\mathbb{N}^{n+1} \rightarrow \mathbb{N}) \rightarrow (\mathbb{N}^n \rightarrow \mathbb{N})$$

Wert:

wenn  $f = \mu^n(g)$ , dann  $f(x_1, \dots, x_n) = z$  falls

$$g(x_1, \dots, x_n, z) = 0 \text{ und } \forall y < z : g(x_1, \dots, x_n, y) \in \mathbb{N} \setminus \{0\}$$

(sonst nicht definiert)

das kleinste letzte Argument, für das der Wert 0 ist,

aber nur, falls alle vorigen Werte definiert (und  $\neq 0$ ) sind.

$$\mu^n(g)(x_1, \dots, x_n) = \min \left\{ z \in \mathbb{N} \mid \begin{array}{l} g(x_1, \dots, x_n, z) = 0 \\ \wedge \forall y < z : g(x_1, \dots, x_n, y) \text{ definiert} \end{array} \right\}$$

Beispiele:

- ▶ für  $f = \mu(g) = \mu^1(g)$  mit  $g(x_1, y) = x_1 \div (2y + 1)$  ist  
 $f(3) = \dots$
- ▶ für  $f = \mu(g) = \mu^2(g)$  mit  $g(x_1, x_2, y) = x_1 \div x_2 \cdot y$ , sind  
 $f(12, 5), f(3, 0)$

(ÜA)

# Was bisher geschah

- ▶ (deterministische) Turing-Maschinen
- ▶ Goto-Programme
  - ▶ Syntax: = Liste von Befehlen aus  
 $B = \{Inc\ i, Dec\ i, Goto\ k, GotoZ\ i\ k, Stop\}$
  - ▶ Semantik: small step  
Konfigurationen  $(l, s)$  mit  $l \in \mathbb{N}, s \in \mathbb{N}^*$  (PC, Speicherbelegung)  
Konfigurationsübergänge  $step_p$  für Befehl,  $step_p^*$  für Programm
- ▶ While/Loop-Programme
  - ▶ Syntax: Baum mit Blättern aus  $\{Inc\ i, Dec\ i, Skip\}$   
inneren Knoten  $While\ i\ p$  ( $Loop\ i\ p$ ),  $Seq\ p\ q$ ,  $IfZ\ i\ p\ q$
  - ▶ Semantik: big step  
Konfigurationen  $s \in \mathbb{N}^*$  (Speicherbelegung)  
Konfigurationsübergänge  $sem_p \subseteq \mathbb{N}^* \times \mathbb{N}^*$  für Programm
- ▶ Ackermann-Funktion  $\in (WHILE \cap TOTAL) \setminus LOOP$

$LOOP \subset (WHILE \cap TOTAL) \subset WHILE = GOTO = Turing$

# WH: Rekursive Funktionen

(induktive) Definition der Menge aller **rekursiven Funktionen**:

IA: elementare Funktionen

- ▶ Konstante 0:  $\text{Zero}^n = (x_1, \dots, x_n) \mapsto 0$
- ▶ Nachfolger:  $\text{Succ} = (x_1) \mapsto 1 + x_1$
- ▶ Projektionen :  $\text{Proj}_k^n = (x_1, \dots, x_n) \mapsto x_k$

IS: Operatoren

- ▶ Substitution  $\text{Sub}_k^n : (\mathbb{N}^k \rightarrow \mathbb{N}) \times (\mathbb{N}^n \rightarrow \mathbb{N})^k \rightarrow (\mathbb{N}^n \rightarrow \mathbb{N})$   
 $(\text{Sub}_k^n(g, h_1, \dots, h_k))(x_1, \dots, x_n) =$   
 $g(h_1(x_1, \dots, x_n), \dots, h_k(x_1, \dots, x_n))$
- ▶ primitive Rekursion  $\text{PR}^n : (\mathbb{N}^{n-1} \rightarrow \mathbb{N}) \times (\mathbb{N}^{n+1} \rightarrow \mathbb{N}) \rightarrow (\mathbb{N}^n \rightarrow \mathbb{N})$   
 $(\text{PR}^n(g, h))(x_1, \dots, x_{n-1}, 0) = g(x_1, \dots, x_{n-1})$   
 $(\text{PR}^n(g, h))(x_1, \dots, x_{n-1}, 1 + y) =$   
 $h(x_1, \dots, x_{n-1}, y, f(x_1, \dots, x_{n-1}, y))$
- ▶ unbeschränkte Rekursion  $\mu^n : (\mathbb{N}^{n+1} \rightarrow \mathbb{N}) \rightarrow (\mathbb{N}^n \rightarrow \mathbb{N})$   
 $(\mu^n(g))(x_1, \dots, x_n) = z$  falls  
 $g(x_1, \dots, x_n, z) = 0$  und  $\forall y < z : g(x_1, \dots, x_n, y) \in \mathbb{N} \setminus \{0\}$   
(sonst nicht definiert)

## Codierungen von Funktionstermen

- ▶ (Funktions-)Terme über einer bestimmten Signatur sind Bäume.
- ▶ WH:  
Codierungen von Bäumen / Termen  $t = f[t_1, \dots, t_n]$  mit
  - ▶  $\text{root}(t) = f$
  - ▶  $\text{args}(t) = [t_1, \dots, t_n]$

$$B(t) = C(\text{num}(\text{root}(t)), L([B(t_1), \dots, B(t_n)]))$$

aufbauend auf den Codierungen:

- ▶ Nummerierung num aller Symbole der Signatur (endlich),
- ▶ Paar-Kodierung C für (Symbol, Argumentliste),
- ▶ Listen-Kodierung L für  $\text{args}(t) = [t_1, \dots, t_n]$ ,

Alle früher besprochenen Codierungen und Decodierungen sind Loop-berechenbar.

Im Folgenden: Bezeichnungen encode und decode für Codierung / Decodierung von Funktionstermen, Tupeln, ...

# Klassen rekursiver Funktionen

Die Menge **Prim** der **primitiv rekursiven** Funktionen ist die kleinste Menge  $M$ , die  $\text{Zero}^n$ ,  $\text{Succ}$ ,  $\text{Proj}_k^n$  enthält und **abgeschlossen** ist **unter Sub und PR**

1.  $g \in M \wedge h_1 \in M \wedge \dots \wedge h_k \in M \rightarrow \text{Sub}_k^n(f, h_1, \dots, h_k) \in M$ ,
2.  $g \in M \wedge h \in M \rightarrow \text{PR}^n(g, h) \in M$

Die Menge **Part** der **partiell rekursiven** Funktionen ist die kleinste Menge  $M$ , die  $\text{Zero}^n$ ,  $\text{Succ}$ ,  $\text{Proj}_k^n$  enthält und **abgeschlossen** ist **unter Sub, PR und  $\mu$**

1. + 2. + 3.  $g \in M \rightarrow \mu^n(g) \in M$

Die Menge **Allg** der **allgemein rekursiven** Funktionen ist  $\text{Allg} = \text{Part} \cap \text{TOTAL} = \{f \mid f \in \text{Part} \wedge f \text{ ist total}\}$

Es gilt  $\text{Prim} \subseteq \text{Allg}$ .

(folgt direkt aus den Definitionen)

Noch zu zeigen (nichttrivial):  $\text{Prim} \subset \text{Allg}$  (Inklusion ist echt)

# Prim $\subset$ Allg

Existenz einer Funktion in Allg  $\setminus$  Prim:

- ▶ Gödelisierung aller primitiv-rekursiven Funktionsterme liefert eine Aufzählung  $f_0, f_1, \dots$  von Prim
- ▶ Für die Funktion  $h : \mathbb{N} \rightarrow \mathbb{N}$  mit  $\forall x \in \mathbb{N} : h(x) = f_x(x) + 1$  gilt:
  1.  $h$  ist total,
  2.  $h \in \text{Part}$ , (Warum?)
  3.  $h \in \text{Allg} \setminus \text{Prim}$ .  
(indirekt) Annahme:  $h \in \text{Prim}$ ,  
dann existiert ein  $k$  mit  $h = f_k$ , und  $h(k) = \dots$

Explizite Konstruktion (ohne Gödelisierung):

$A \in \text{Allg} \setminus \text{Prim}$



# Syntaktisch und semantisch definierte Klassen

Definitionen von Prim und Part sind **syntaktisch**:

$f \in \text{Part}$  gdw.

$f$  durch einen Funktionsterm (Ausdruck, Programm) für  $f$  aus bestimmten Basisfunktionen und Operatoren darstellbar ist.

Definition von  $\text{TOTAL} = \{f : \mathbb{N} \rightarrow \mathbb{N} \mid f \text{ ist total} \}$

ist **semantisch**.

Diese semantische Bedingung („ist total“, d.h.  $\forall n \in \mathbb{N} : f(n) \in \mathbb{N}$ ) lässt sich nicht durch eine syntaktische Bedingung ersetzen.  
(Nachweis später)

Definition von Allg ist damit auch semantisch.

$f \in \text{Allg} \leftrightarrow f \in \text{Part} \wedge f \text{ ist total}$

# LOOP $\subseteq$ Prim

Satz: Jede Loop-berechenbare Funktion ist primitiv rekursiv.

Beweis: Transformation

Loop-Programm  $\mapsto$  primitiv rekursiver Funktionsterm

Induktion über Loop-Programm  $p$

IA:  $p = \text{Inc}(i)$

$$f_{\text{Inc}(i)} = \text{Sub}(\text{encode}, \text{decode}_1, \dots, \text{Sub}(\text{Succ}, \text{decode}_i), \dots)$$

Dec  $i$ , Skip analog (ÜA)

- IS:
- ▶ Annahme: Die von den Loop-Programmen  $q, q'$  berechneten Funktionen werden durch die primitiv rekursive Funktionsterme  $f_q$  und  $f_{q'}$  repräsentiert.
  - ▶ Konstruktion von primitiv rekursive Funktionstermen für die Loop-Programmen  $p = \text{Seq}(q, q')$  und  $p = \text{Loop}(i, q)$

▶  $p = \text{Seq}(q, q')$ :

$$f_{\text{Seq}(q, q')} = \text{Sub}(f_{q'}, f_q)$$

▶  $p = \text{Loop}(i, q)$ :  $f_{\text{Loop}(i, q)} = \dots$  (ÜA)

# Prim $\subseteq$ LOOP

Satz: Jede primitiv rekursive Funktion ist Loop-berechenbar.

Beweis: Transformation

primitiv rekursiver Funktionsterm  $\mapsto$  Loop-Programm

Induktion über Funktionsterm (abstrakten Syntaxbaum):

IA: (Loop-Programme für)

elementare Funktionen  $\text{Zero}^n, \text{Succ}, \text{Proj}_k^n$

IS:  $\blacktriangleright$  Substitutions-Operator  $\text{Sub}_k^n(g, h_1, \dots, h_k)$  (ÜA)

$\blacktriangleright$  Primitive Rekursion  $\text{PR}^n(g, h)$ :

wenn  $g : \mathbb{N}^{n-1} \rightarrow \mathbb{N}$  berechnet durch  $p$

und  $h : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$  berechnet durch  $q$ ,

dann wird  $\text{PR}^n(g, h)(x, y)$  berechnet durch

$\text{Seq} ( \text{Seq} ( a := g(x), k := y, z := 0 ),$

$\text{Loop } k ( \text{Seq} ( a := h(x, z, a), \text{Inc } z ) ) )$

(Ergebnis in  $a$ )

zusammen mit  $\text{LOOP} \subseteq \text{Prim}$  ergibt das

$$\text{Prim} = \text{LOOP}$$

## Part $\subseteq$ WHILE

Satz: Jede partiell rekursive Funktion ist While-berechenbar.

Beweis: Induktion über Funktionsterm (abstrakten Syntaxbaum)

IA: elementare Funktionen

- IS:   ▶ Substitutions-Operator  $\text{Sub}_k^n(g, h_1, \dots, h_k)$    (ÜA)
- ▶ Primitive Rekursion  $\text{PR}(g, h)$ :  
      wenn  $g : \mathbb{N}^{n-1} \rightarrow \mathbb{N}$  berechnet durch  $p$   
      und  $h : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$  berechnet durch  $q$ ,  
      dann  $\text{PR}^n(g, h)(x, y)$  berechnet durch  
      Seq ( Seq ( a := g(x), z := 0 ),  
          While (...) ( Seq ( a := ..., Inc z ) ) )  
      (Ergebnis in a)
- ▶  $\mu$ -Operator:  $\mu(g)$   
      wenn  $g : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$  berechnet durch  
      While-Programm  $p$ , dann  $\mu(g)(x)$  durch  
      Seq ( z := 0, While (...) ( Inc z ) )  
      (Ergebnis in z)

# WHILE $\subseteq$ Part (Plan)

Satz: Jede while-berechenbare Funktion ist partiell rekursiv.

- ▶ diese Aussage betrifft Funktionen nach  $\mathbb{N}^1$ , im Beweis werden aber Funktionen  $\mathbb{N}^n \rightarrow \mathbb{N}^n$  benötigt (Zustands-Übergänge der Maschine, alle Register).
- ▶ Für jedes  $p \in \text{WHILE}$ , welches die Registermenge  $\text{var}(p)$  benutzt:

$f_p : \mathbb{N} \rightarrow \mathbb{N}$  mit  $f_p(x) = y \leftrightarrow (\text{decode}(x), \text{decode}(y)) \in \text{sem}_p$ .  
mit (primitiv-rekursiver) Tupel-Codierung und -Dekodierung

- ▶  $\text{encode} : \mathbb{N}^{|\text{var}(p)|} \rightarrow \mathbb{N}$ ,
- ▶  $\text{decode} : \mathbb{N} \rightarrow \mathbb{N}^{|\text{var}(p)|}$
- ▶ und Projektionen auf die Elemente der Tupel  
 $\text{decode}_i : \mathbb{N} \rightarrow \mathbb{N}$  mit  $\text{decode}_i(x) = \text{Proj}_i^{|\text{var}(p)|}(\text{decode}(x))$

# WHILE $\subseteq$ Part

z.z.: Für jedes While-Programm  $p$  ist  $f_p \in \text{Part}$ .

$f_p : \mathbb{N} \rightarrow \mathbb{N}$  mit  $f_p(x) = y \leftrightarrow (\text{decode}(x), \text{decode}(y)) \in \text{sem}_p$ .

Beweis durch Induktion über Programmtext von  $p \in \text{WHILE}$ :

- ▶ einfache Programme: Inc, Dec, Skip

Für  $p = \text{Inc}(i)$  ist

$$f_p = \text{Sub}(\text{encode}, \text{decode}_1, \dots, \text{Sub}(\text{Succ}, \text{decode}_i), \dots)$$

- ▶ zusammengesetzte Programme:

- ▶  $p = \text{Seq}(q, q')$ :  $f_{\text{Seq}(p,q)} = \text{Sub}(f_{q'}, f_q)$

- ▶  $p = \text{IfZ}(i, p, q)$ : (ÜA)

- ▶  $p = \text{While}(i, q)$ :

- ▶ Für die Funktionen  $g(x, k) = \text{decode}_i(f_q^k(x))$  und  $h = \mu(g)$  ist  $h(x)$  das kleinste  $k$ , so dass nach  $k$ -maliger Ausführung des Schleifenkörpers  $q$  gilt  $\text{decode}_i(f_q^k(x)) = 0$

- ▶ Dann gilt  $\forall x \in \mathbb{N} : f_p(x) = f_{\text{While}(i,q)}(x) = f_q^{h(x)}(x)$

... durch PR und Sub darstellbar

(ÜA)

## WHILE $\subseteq$ Part

Zusammen mit  $\text{Part} \subseteq \text{WHILE}$  ergibt das

$$\text{Part} = \text{WHILE}$$

Beobachtungen:

- ▶ Wenn  $g \in \text{Prim}$ ,  
dann sind auch die Funktionen  $(g^k(x)) (i) \in \text{Prim}$ .
- ▶ In den Übersetzungen wird ein While in genau ein  $\mu$   
transformiert und umgekehrt.

Folgerung: Jede  $\mu$ -rekursive Funktion lässt sich durch einen  
Funktionsterm mit nur einem  $\mu$  darstellen.

Beweis: Transformationen  $\mu \rightarrow \text{While} \rightarrow \text{ein While} \rightarrow \text{ein } \mu$

# These von Church und Turing

gezeigt:  $\text{Prim} = \text{LOOP} \subset \text{WHILE} = \text{GOTO} = \text{Turing} = \text{Part}$

und für verschiedene weitere Berechnungsmodelle  $M$ ,

z.B.  $\lambda$ -Kalkül, SRS, Markov-Algorithmen, ...

lässt sich  $M = \text{Turing}$  (oder äquivalente Klasse) zeigen.

These von Church und Turing:

„Alle intuitiv vernünftigen Berechenbarkeitsmodelle definieren die gleiche Klasse von partiellen Funktionen.“

(Diese Aussage ist sinnvoll, aber wegen des Bezugs auf den ungenauen Begriff „intuitiv“ nicht beweisbar.)

Die These von Church und Turing kann man auffassen als empirische Aussage oder als Definition

$M$  ist vernünftig  $\leftrightarrow$  (es gibt beide Compiler  $M \leftrightarrow \text{WHILE}$ )



# Was bisher geschah

## Berechenbarkeit

- ▶ Berechnungsmodelle
  - ▶ Goto-, While-, Loop-Programme
  - ▶ Goto-, While-, Loop-**berechenbare Funktionen**
  - ▶ primitiv, allgemein, partiell rekursive Funktionen
- ▶ WHILE = GOTO = Turing = Part  
(Hauptsatz der Algorithmentheorie)  
gezeigt durch (primitiv rekursive) Übersetzungen zwischen den Berechnungsmodellen
- ▶ These von Church
- ▶ Prim = LOOP  $\subset$  WHILE  
Ackermann-Funktion  $\in$  WHILE  $\setminus$  LOOP

## Hilfsmittel / Werkzeuge:

- ▶ Übersetzung  $X^* \leftrightarrow \mathbb{N}$
- ▶ Codierungen von Paaren, Tupeln, Listen, Bäumen, Termen, Programmen  
(Gödelisierung, Gödelnummern)
- ▶ Darstellung von Aufgaben (Problemen) als Mengen

## WH: Gödelisierung

- ▶ partiell rekursiver Funktionsterm  $t \in (\Sigma_{PR})$
- ▶  $t$  repräsentiert die partiell rekursive Funktion  $f_t : \mathbb{N}^n \rightarrow \mathbb{N}$
- ▶  $B(t) \in \mathbb{N}$  ist Gödelnummer (Baumcodierung) von  $t$

Gesucht:

Zuordnung von  $B(t)$  zu  $f_t$  durch partiell rekursive Funktionen

$\varphi^n : \mathbb{N} \rightarrow (\mathbb{N}^n \rightarrow \mathbb{N})$ , so dass  $\varphi^n(B(t)) = f_t$ ,

also  $\forall B(t) \in \mathbb{N} \forall x \in \mathbb{N}^n : (\varphi^n(B(t)))(x) = f_t(x)$

- ▶ Wie erhält man solche  $\varphi^n$  ?
- ▶ Geht das primitiv / partiell rekursiv?

# Parametrisierung

Satz: Zu jeder partiell rekursiven Funktion  $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$  und jedem  $x \in \mathbb{N}$  existiert eine partiell rekursive Funktion  $g : \mathbb{N} \rightarrow \mathbb{N}$  mit  $g = f(x, \cdot)$ .  
(Currying)

Es gibt eine primitiv rekursive Funktion  $s : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ , welche die Gödelnummer von  $g$  aus der für  $f$  und  $x$  berechnet.

Beweisidee:

- ▶  $f \in \text{Part}$ , also repräsentiert durch Term  $t$  mit Gödelnummer  $B(t)$
- ▶ Konstruktion Term  $t'$  durch Modifikation von  $t$  abhängig von  $x$ :  
Einsetzen eines Terms für Konstante  $x$  an allen Stellen in  $t$ ,  
an denen auf das erste Argument zugegriffen wird (Proj, Succ)
- ▶  $t'$  repräsentiert  $g : \mathbb{N} \rightarrow \mathbb{N}$  mit  $\forall y \in \mathbb{N} : g(y) = f(x, y)$
- ▶  $s(B(t), x)$  definiert durch
  1. Decodierung von  $B(t) \in \mathbb{N}$  zu Funktionsterm  $t$
  2. Transformation  $t \mapsto t'$
  3. Codierung des entstandenen Funktionsterms  $t'$  ergibt  
 $s(B(t), x) = B(t')$

Alle 3 Schritte sind primitiv rekursiv, also  $s$  auch

- ▶ Gödelnummer  $s(B(t), x) \in \mathbb{N}$  repräsentiert  $g : \mathbb{N} \rightarrow \mathbb{N}$  mit  
 $\forall y \in \mathbb{N} : f(x, y) = (\varphi^2(B(t)))(x, y) = (\varphi^1(s(B(t), x)))(y) = g(y)$

# Parametrisierung (smn-Theorem)

analog für mehrstellige Funktionen:

Satz (smn-Theorem):

Für jedes Paar  $(m, n) \in \mathbb{N}^2$  existiert eine primitiv rekursive Funktion  $s_{(m,n)} : \mathbb{N} \times \mathbb{N}^m \rightarrow \mathbb{N}$  mit

$$\forall f : \mathbb{N}^{m+n} \rightarrow \mathbb{N} \forall x \in \mathbb{N}^m : \varphi^{m+n}(x, \cdot) = \varphi^n(s_{(m,n)}(t, x), x)$$

also

$$\forall f : \mathbb{N}^{m+n} \rightarrow \mathbb{N} \forall x \in \mathbb{N}^m \forall y \in \mathbb{N}^n : \varphi^{m+n}(x, y) = (\varphi^n(s_{(m,n)}(t, x), x))(y)$$

Beweisidee: Induktion über  $n$

# Universelle Funktionen

universelle Funktionen:  $\psi_u^n : \mathbb{N} \times \mathbb{N}^n \rightarrow \mathbb{N}$

mit  $\forall B(t) \in \mathbb{N} : \psi_u^n(B(t), \cdot) = \varphi^n(B(t)) = f_t$ ,

(d.h.  $\forall B(t) \in \mathbb{N} \forall x \in \mathbb{N}^n : \psi_u^n(B(t), x) = (\varphi^n(B(t)))(x) = f_t(x)$ )

Die Funktion  $\psi_u^n(B(t), x)$  simuliert die Anwendung von  $\varphi^n(B(t)) : \mathbb{N}^n \rightarrow \mathbb{N}$  (also  $f_t : \mathbb{N}^n \rightarrow \mathbb{N}$ ) auf Eingabe  $x \in \mathbb{N}^n$ :

- ▶  $\psi_u^n(B(t), x)$  ist das Ergebnis der Anwendung von  $f_t : \mathbb{N}^n \rightarrow \mathbb{N}$  auf Eingabe  $x \in \mathbb{N}^n$ .
- ▶  $\psi_u^n(B(t), \cdot) = \varphi^n(B(t)) = f_t$  ist ein Interpreter für  $f_t$ .
- ▶  $\psi_u^n$  interpretiert beliebige  $n$ -stellige partiell rekursive Funktionen.

Satz:

Für jedes  $n \in \mathbb{N}$  existieren solche  $\psi_u^n$  und es gilt  $\psi_u^n \in \text{Part}$ .

Wegen  $\text{Part} = \text{WHILE} = \text{GOTO} = \text{Turing}$  existieren also auch universelle While-, Goto-Programme und TM

häufige Notation:  $\psi_u$  für alle  $\psi_u^n$ ,  $t$  für  $B(t)$

# WH: Codierung von Problemen als Mengen

Aufgabe (Problem): Zuordnung zwischen

- ▶ Eingaben
- ▶ passenden Ausgaben

(oft strukturierte Daten)

Beispiele: Addieren zweier Zahlen, Sortieren von Listen

Repräsentation der Aufgabe als

Zuordnung = Menge von Paaren (Eingabe, Ausgabe)

Transformation in  $P \subseteq \mathbb{N}$  (durch übliche Codierungen)

Spezialfall Entscheidungsprobleme: Ausgabe  $\in \{0, 1\}$

- ▶ definiert eine Eigenschaft auf der Menge aller Eingaben
- ▶ repräsentiert durch Urbild der 1
- ▶ charakteristische Funktion der Menge aller Eingaben mit dieser Eigenschaft

Beispiele:  $\{n \in \mathbb{N} \mid \exists k \in \mathbb{N} : n = 2^k\}$ ,

$\{(x, y, z) \in \mathbb{N}^3 \mid x^2 + y^2 = z^2\}$ ,

HC =  $\{G = (V, E) \mid G \text{ enthält Hamiltonkreis}\}$

# Entscheidbarkeit von Mengen

**charakteristische Funktion** einer Menge  $L \subseteq \mathbb{N}$  (Sprache  $L \subseteq X^*$ ):

$$\chi_L : \mathbb{N} \rightarrow \{0, 1\}, \text{ wobei } \forall n \in \mathbb{N} : \chi_L(n) = \begin{cases} 1 & \text{falls } n \in L \\ 0 & \text{sonst} \end{cases}$$

Menge  $L \subseteq \mathbb{N}$  heißt rekursiv **entscheidbar** (rekursiv)

gdw. die (totale) charakteristische Funktion  $\chi_L$  berechenbar ist.

**REC**: Menge aller rekursiv entscheidbaren Mengen

Menge  $L \subseteq \mathbb{N}$  heißt **semi-entscheidbar**

gdw. die „halbe“ charakteristische Funktion  $\chi'_L : \mathbb{N} \rightarrow \{0, 1\}$

berechenbar ist:

$$\forall n \in \mathbb{N} : \chi'_L(n) = \begin{cases} 1 & \text{falls } n \in L \\ \text{nicht definiert} & \text{sonst} \end{cases}$$

## Fakt

*Eine Menge  $L$  ist genau dann semi-entscheidbar, wenn sie Turing-akzeptierbar ist.*

(ÜA)

# Beispiele entscheidbarer Mengen

- ▶ jede endliche Menge  $L = \{k_1, \dots, k_n\}$   
(für jedes  $i$  nacheinander Test, ob Eingabe =  $k_i$ )
- ▶  $\{2^n \mid n \in \mathbb{N}\}$ ,
- ▶ Menge aller Primzahlen,
- ▶ Menge aller Primzahlzwillinge, d.h.  
Paare  $(p, p + 2)$  mit Primzahlen  $p$  und  $p + 2$
- ▶ jede reguläre, kontextfreie, kontextsensitive Sprache,
- ▶ Menge aller Codierungen (bei gegebener Codierungsfunktion)  
von TM, Goto-Programmen, ...



# Entscheidbare Mengen

## Satz

*Eine Sprache (Menge)  $L$  ist genau dann entscheidbar, wenn  $L$  und  $\bar{L}$  Turing-akzeptierbar sind.*

Beweis (Idee):

( $\rightarrow$ ) gegeben: TM  $M$ , die  $L$  entscheidet

Konstruktion von TM  $M_1, M_2$ , die  $L$  und  $\bar{L}$  akzeptieren:

Transformation entscheidene  $\rightarrow$  akzeptierende TM für  $L$ , (ÜA)  
für  $\bar{L}$  ... (ÜA)

( $\leftarrow$ ) gegeben: TM  $M_1$  akzeptiert  $L$ , TM  $M_2$  akzeptiert  $\bar{L}$

Konstruktion einer TM  $M$ , die  $L$  entscheidet ( $\chi_L$  berechnet):

TM  $M$  mit Eingabe  $w$  simuliert abwechselnd je einen

Berechnungsschritt in  $M_1$  und  $M_2$ .

Einer der folgenden Fälle tritt nach endlich vielen Schritten ein:

- ▶  $M_1$  akzeptiert  $w$ , dann Ausgabe  $M$ : 1 (falls  $w \in L$ )
- ▶  $M_2$  akzeptiert  $w$ , dann Ausgabe  $M$ : 0 (falls  $w \notin L$ )

# Entscheidbare Mengen – Abschlusseigenschaften

Die Menge aller **entscheidbaren** Sprachen ist abgeschlossen unter

**Vereinigung** wegen

$$\chi_{(L \cup L')}(w) = \max(\chi_L(w), \chi_{L'}(w)) \text{ berechenbar}$$

**Schnitt** wegen

$$\chi_{(L \cap L')}(w) = \min(\chi_L(w), \chi_{L'}(w)) \text{ berechenbar}$$

**Komplement** wegen

$$\chi_{\bar{L}}(w) = 1 - \chi_L(w) \text{ berechenbar}$$

Achtung (Unterschied zu TM-akzeptierbaren Sprachen):  
Aus der TM-Akzeptierbarkeit von  $L$  folgt i.A. **nicht** die  
TM-Akzeptierbarkeit von  $\bar{L}$ .

# Rekursiv aufzählbare Mengen

$L \subseteq \mathbb{N}$  ( $L \subseteq X^*$ ) heißt (rekursiv) aufzählbar gdw.

- ▶  $L = \emptyset$  oder
- ▶ eine totale berechenbare Funktion  $f : \mathbb{N} \rightarrow \mathbb{N}$  ( $f : \mathbb{N} \rightarrow X^*$ ) mit  $L = f(\mathbb{N})$  existiert.

also  $L = \emptyset$  oder  $L = \{f(0), f(1), f(2), \dots\}$ , wobei

- ▶ jedes Element von  $L$  kommt wenigstens einmal vor,
- ▶  $f$  ist nicht notwendig injektiv (wiederholungsfrei),
- ▶  $f$  ist nicht notwendig monoton.

**RE:** Menge aller rekursiv aufzählbaren Mengen

Achtung:

- ▶ Nicht jede abzählbare Menge ist rekursiv aufzählbar.
- ▶ Jede rekursiv aufzählbare Menge ist abzählbar.

(ÜA)

# Rekursiv aufzählbare Mengen

## Satz

Für jede Sprache  $L \subseteq \mathbb{N}$  ( $L \subseteq X^*$ ) sind folgende Aussagen äquivalent:

1.  $L$  ist rekursiv aufzählbar.
2.  $L$  ist TM-akzeptierbar.
3.  $L$  ist semi-entscheidbar. (d.h.  $\chi'_L$  berechenbar).

Beweis (Idee):

1  $\rightarrow$  2    ÜA

2  $\rightarrow$  3    ÜA

3  $\rightarrow$  1     $L$  semi-entscheidbar gdw. ex. TM  $M$ , die  $\chi'_L$  berechnet

gesucht: totale berechenbare Funktion  $f : \mathbb{N} \rightarrow \mathbb{N}$

( $f : \mathbb{N} \rightarrow X^*$ ) mit  $L = \{f(0), f(1), \dots\}$

Algorithmus zur Berechnung von  $f : \mathbb{N} \rightarrow \mathbb{N}$  :

für  $L = \emptyset$ :  $f = \perp$ , sonst  $\exists y \in L$  und dove-tailing:

$n \in \mathbb{N}$  als Codierung von  $(x, i) \in \mathbb{N}^2$  (Eingabe, Schritt)

$$f(n) = \begin{cases} x & \text{falls } n = C(x, i) \text{ und } M(x) \text{ nach } i \text{ Schritten hält} \\ y & \text{sonst} \end{cases}$$

# Abschlusseigenschaften von RE

Falls  $A, B \in \text{RE}$ , dann gilt auch

- ▶  $(A \cup B) \in \text{RE}$ .

Beweis:  $f_A$  aufzählende Funktion für  $A \neq \emptyset$ ,  $f_B$  die für  $B \neq \emptyset$

Dann wird  $A \cup B$  aufgezählt von

z.B.  $h(2n) = f_A(n)$ ,  $h(2n+1) = f_B(n)$

- ▶  $(A \cap B) \in \text{RE}$ .

Beweis: dovetailing, 2-dim. unendliche Tabelle,

In Zeile  $x$ , Spalte  $y$  steht Zahlenpaar  $(f_A(x), f_B(y))$ .

Wenn  $\dots$ , gibt  $h(n) = \dots$  aus.

Falls für  $L, L' \subseteq X^*$  gilt  $L, L' \in \text{RE}$ , dann gilt auch

- ▶  $L^R \in \text{RE}$ . (ÜA)
- ▶  $(L \circ L') \in \text{RE}$ . (ÜA)
- ▶  $L^* \in \text{RE}$ . (ÜA)

# Aufgaben (Probleme) als Mengen (Sprachen)

**Entscheidungsproblem** (Aufgabe) =  
Menge der Codierungen aller korrekten Instanzen

Beispiel:

Aufgabe : Gilt für gegebene Zahlen  $x, y \in \mathbb{N} : x < y$  ?

Darstellung als Sprache  $P_{<} = \{C(x, y) \in \mathbb{N} \mid x < y\}$

**Parameter** (Eingaben für Lösungsverfahren) :  $(x, y) \in \mathbb{N}^2$ ,

**Frage:** Gilt  $x < y$  ? (Gilt  $(x, y) \in P_{<}$  ?)

**Antwort** (mögliche Ausgaben des Lösungsverfahrens):

- ▶ **ja** (1), falls  $x < y$
- ▶ **nein** (0), sonst

**Instanzen** z.B.

- ▶  $C(3, 5) \in P_{<}$  (korrekte Instanz),
- ▶  $C(5, 3) \notin P_{<}$

# Aufgaben (Probleme) als Mengen (Sprachen)

Beispiel: Wortproblem für Typ-3-Grammatiken

Wortproblem  $WP_{G_3} = \{C(C(G), C(w)) \mid w \in L(G)\}$

Parameter (Eingaben für Lösungsverfahren) :  $(G, w)$ ,

- ▶ Typ-3-Grammatik  $G = (N, T, P, S)$  und
- ▶  $w \in T^*$

Frage: Gilt  $w \in L(G)$  ? (Gilt  $n \in WP_{G_3}$  ?)

Antwort (mögliche Ausgaben des Lösungsverfahrens):

- ▶ **ja** (1), falls  $w \in L(G)$   
(falls  $n = C(C(G), C(w))$ ) für eine Grammatik  $G$   
und ein Wort  $W$  und  $w \in L(G)$ )
- ▶ **nein** (0), sonst  
(falls  $n$  keine Codierung oder  $w \notin L(G)$ )

Instanzen z.B. mit  $G = (\{S\}, \{a, b\}, \{S \rightarrow aS|b\}, S)$ :

- ▶  $C(C(G), C(aab)) \in WP_{G_3}$  (korrekte Instanz),
- ▶  $C(C(G), C(aba)) \notin WP_{G_3}$

(im Folgenden meist ohne explizite Codierungen)

# Aufgaben und Instanzen

**Aufgabe** (Problem): **Sprache**  $P \subseteq X^*$  (Menge  $P \subseteq \mathbb{N}$ )  
(Codierungen einer Klasse von verwandten Aufgaben)

**Instanz** eines Problems: **Wort**  $w \in X^*$  ( $w \in \mathbb{N}$ )  
(Codierung einer speziellen Aufgabe aus dieser Klasse)

**Lösungsverfahren** für das Problem  $P$ :

Entscheidungsverfahren für  $P$ , d.h. Verfahren, welches für  
**jede Instanz**  $w \in X^*$  ( $n \in \mathbb{N}$ ) bestimmt, ob  $w \in P$  ( $n \in P$ )

Beispiel: Wortproblem für reguläre Sprachen

**Problem:**  $WP\text{-REG} = \{(C(L), w) \mid w \in L\}$  mit  
endlicher Darstellung  $C(L)$  der Sprache  $L$   
z.B.  $C(L) = A$  für einen NFA  $A$  mit  $L = L(A)$

**Instanz:** z.B.  $(C(A), aba)$  mit  $A = (\{a, b\}, \{p, q\}, \delta, \{p\}, \{p, q\})$  und  
 $\delta(a) = \{(p, p)\}$ ,  $\delta(b) = \{(p, q), (q, q)\}$   
Frage: Gilt  $(A, aba) \in WP\text{-REG}$ ?  
wahr gdw.  $aba \in L(A)$

**Lösungsverfahren** (Entscheidungsverfahren),  
z.B. Pfadsuche im Automatengraphen



# Aufgaben und Entscheidungsverfahren

Problem (als Menge)  $P$  ist

**unentscheidbar** gdw. **kein** Entscheidungsverfahren (TM, Programm,...) für  $P$  existiert,

**entscheidbar** gdw. **(wenigstens) ein** Entscheidungsverfahren für  $P$  existiert (i.A. existieren mehrere)  
z.B. SAT, Suche in Folgen, Sortieren von Folgen  
durch verschiedene Such- und Sortierverfahren

**Entscheidbarkeit** eines Problems (Sprache)  $P$  wird i.A. durch Angabe eines Entscheidungsverfahrens für  $P$  gezeigt.

**Unentscheidbarkeit** eines Problems  $P$  wird oft durch **Reduktion** bekannter unentscheidbarer Probleme (oft HP) auf  $P$  gezeigt.

## Spezielles Halteproblem für TM

WH:  $C(M) \in \mathbb{N}$ : Codierung der (endlichen Beschreibung der) TM  $M$   
( $C(M)$  heißt Gödelnummer der TM  $M$ )

Abhängig von der Codierung muss nicht zu jedem  $n \in \mathbb{N}$  eine TM  $M$  mit  $C(M) = n$  existieren.

Mit fester TM  $M'$  lässt sich jedem  $n \in \mathbb{N}$  eine TM  $M_n$  zuordnen

$$M_n = \begin{cases} M & \text{falls } n = C(M) \\ M' & \text{sonst} \end{cases}$$

Das **spezielle Halteproblem** ist die Menge

$$H_S = \{n \mid M_n \text{ hält bei Eingabe } n\} \subseteq \mathbb{N}$$

### Satz

Die Menge (Sprache)  $H_S$  ist Turing-akzeptierbar.

(ÜA)

(und damit auch: semi-entscheidbar, rekursiv aufzählbar)

Frage: Ist die Menge  $H_S$  entscheidbar?

# Unentscheidbarkeit des speziellen Halteproblems

## Satz

Es existiert keine TM, welche das spezielle Halteproblem (die Menge  $H_S$ ) entscheidet, d.h. die folgende Funktion berechnet:

$$\forall n \in \mathbb{N} : \chi_S(n) = \begin{cases} 1 & \text{falls TM } M_n \text{ bei Eingabe von } n \text{ h\"alt} \\ 0 & \text{sonst} \end{cases}$$

indirekter Beweis: (analog Barbier-Problem)

Annahme:  $H_S$  entscheidbar, also existiert eine TM  $N$  mit  $f_N = \chi_{H_S}$

Konstruktion einer TM  $N'$  aus  $N$ , so dass f\"ur jedes  $n \in \mathbb{N}$  gilt:

$$f_{N'}(n) = \begin{cases} \text{nicht definiert} & \text{falls } M_n \text{ bei Eingabe von } n \text{ h\"alt} \\ 1 & \text{falls } M_n \text{ bei Eingabe von } n \text{ nicht h\"alt} \end{cases}$$

Ist  $f_{N'}(C(N'))$  definiert?

nein gdw.  $M_{C(N')} = N'$  bei Eingabe von  $C(N')$  h\"alt

ja gdw. ...

Widerspruch in beiden F\"allen

Eine solche TM  $N$  kann also nicht existieren.

# Komplement des speziellen Halteproblems

Spezielles HP:

$$H_S = \{n \mid M_n \text{ hält bei Eingabe } n\}$$

Komplement des speziellen HP:

$$\overline{H_S} = \{n \mid M_n \text{ hält bei Eingabe } n \text{ nicht}\}$$

## Folgerung

*Die Sprache  $\overline{H_S}$  ist nicht Turing-akzeptierbar.*

Beweisidee (indirekt):

- ▶ bekannt:
  1.  $H_S$  ist TM-akzeptierbar.
  2.  $H_S$  ist nicht entscheidbar.
  3.  $H_S$  TM-akz.  $\wedge$   $\overline{H_S}$  TM-akz.  $\rightarrow H_S$  entscheidbar
- ▶ Annahme:  $\overline{H_S}$  ist TM-akzeptierbar.  
dann ist  $H_S$  entscheidbar wegen 1. und 3.  
Widerspruch zu 2., also Annahme falsch,  
also  $\overline{H_S}$  nicht TM-akzeptierbar.

# Was bisher geschah

## Berechenbare Funktionen

- ▶ Berechnungsmodelle WHILE = GOTO = Turing = Part
- ▶ Codierungen von Paaren, Listen, Bäumen, Programmen (Gödelisierung, Gödelnummern), Übersetzung  $X^* \rightarrow \mathbb{N}$
- ▶ Universelle Funktion (TM, Programm) partiell, berechenbar  $\psi_u : \mathbb{N}^2 \rightarrow \mathbb{N}$  mit  $\psi_u(x, y) = f_{M_x}(y)$  für TM  $M_x = C^{-1}(x)$

## Entscheidbare Mengen REC (von recursive = entscheidbar)

- ▶  $L \subseteq \mathbb{N}$  ( $L \subseteq X^*$ ) entscheidbar  
gdw. charakteristische Funktion  $\chi_L$  berechenbar  
gdw. sowohl  $L$  als auch  $\bar{L}$  TM-akzeptierbar
- ▶ abgeschlossen unter  $\cup, \cap, -, \setminus, \circ, *, R$

## Rekursiv aufzählbare Mengen RE (von recursively enumerable)

- ▶  $L \subseteq \mathbb{N}$  ( $L \subseteq X^*$ ) rekursiv aufzählbar  
gdw.  $L = \emptyset$  oder  $\exists f : \mathbb{N} \rightarrow \mathbb{N}$  total berechenbar mit  $L = f(\mathbb{N})$   
gdw. „halbe“ charakteristische Funktion  $\chi'_L$  berechenbar  
gdw.  $L$  TM-akzeptierbar
- ▶ abgeschlossen unter  $\cup, \cap, \circ, *, R$ , aber nicht unter  $-$

spezielles HP  $H_S = \{n \in \mathbb{N} \mid \psi_u(n, n) \text{ definiert}\} \in \text{RE} \setminus \text{REC}, \overline{H_S} \notin \text{RE}$

## Reduktion $\leq_m$

Zum Vergleich der algorithmischen Schwierigkeit von Problemen definiert man für  $P \subseteq \mathbb{N}, Q \subseteq \mathbb{N}$ :

$P \leq_m Q$  („ $P$  ist reduzierbar auf  $Q$ “) durch:

es existiert eine berechenbare totale Funktion  $f : \mathbb{N} \rightarrow \mathbb{N}$  mit  $\forall x \in \mathbb{N} : x \in P \leftrightarrow f(x) \in Q$ .

- ▶ beachte die Richtung:  $P$  ist **höchstens so schwierig** wie  $Q$
- ▶ „reduzieren“ bedeutet: ein Entscheidungsverfahren für  $P$  auf ein Verfahren für  $Q$  zurückführen.
- ▶ Index  $m$  kommt von „many-one“-Reduktion

Beispiele:

- ▶  $2\mathbb{N} \leq_m 3\mathbb{N}$  mit  $\forall x \in \mathbb{N} : f(x) = 3x/2$ , falls  $x \in 2\mathbb{N}$ , sonst 1
- ▶  $\{C(M) \mid \text{TM } M\} \leq_m \{C(M) \mid \text{DTM } M\}$  mit Transformation nichtdeterministischer in (äquivalente) deterministische TM

Satz:  $\leq_m$  ist transitiv.

(ÜA)

## Reduktion zum Nachweis

Satz:

$$\forall P, Q \subseteq \mathbb{N} : ((P \leq_m Q \wedge Q \in \text{RE}) \rightarrow P \in \text{RE}) \quad (1)$$

$$\forall P, Q \subseteq \mathbb{N} : ((P \leq_m Q \wedge Q \in \text{REC}) \rightarrow P \in \text{REC}) \quad (2)$$

Beweis: Für Reduktion  $f : \mathbb{N} \rightarrow \mathbb{N}$

gilt  $\forall n \in \mathbb{N} : \chi'_P(n) = \chi'_Q(f(n))$  und  $\forall n \in \mathbb{N} : \chi_P(n) = \chi_Q(f(n))$

Häufiger benutzt wird die Umkehrung, z.B.

$$\forall P, Q \subseteq \mathbb{N} : ((P \leq_m Q \wedge P \notin \text{REC}) \rightarrow Q \notin \text{REC}) \quad (\text{ÜA})$$

zum Nachweis der Unentscheidbarkeit von  $Q$  mit Hilfe der (bekannten) Unentscheidbarkeit von  $P$

# Folgerungen aus der Unentscheidbarkeit von $H_S$

(Anwendungen der Reduktion)

Keine der folgenden Probleme (Mengen) ist entscheidbar:

$$H = \{(n, m) \in \mathbb{N}^2 \mid \psi_u(n, m) \text{ definiert}\}$$

**allgemeines Halteproblem**

Beweis durch Reduktion  $H_S \leq_m H$  mit  $f = n \mapsto (n, n)$

$$H_0 = \{n \in \mathbb{N} \mid \psi_u(n, 0) \text{ definiert}\}$$

**Halteproblem auf leerem Band** (Unärcodierung von  $\mathbb{N}$ )

Beweis durch Reduktion  $H \leq_m H_0$

$$T = \{n \in \mathbb{N} \mid \forall m \in \mathbb{N} : \psi_u(n, m) \text{ definiert}\}$$

**Totalitätsproblem**

$$I = \{n \in \mathbb{N} \mid \exists m \in \mathbb{N} : \psi_u(n, m) \text{ definiert}\}$$

$$N = \{n \in \mathbb{N} \mid \forall m \in \mathbb{N} : \psi_u(n, m) \text{ nicht definiert}\} = \bar{I}$$

**Leerheitsproblem**



# Gleichheitsproblem

$$G = \{(n, m) \in \mathbb{N}^2 \mid \forall x \in \mathbb{N} : \psi_u(n, x) = \psi_u(m, x)\}$$

alternative Formulierung als Gleichheitsproblem für DTM

$$G = \{(M, N) \mid M \text{ akzeptiert dieselbe Sprache wie } N\}$$

Satz:  $G$  ist nicht entscheidbar.

Beweis:

Wähle DTM  $N$ , die jede Eingabe akzeptiert (durch Halt).

Reduktion  $H_0 \leq_m G$

mit Reduktionsfunktion  $f = M \mapsto (M', N)$ , wobei DTM  $M'$

1. löscht Eingabe und schreibt 0
2. führt dann  $M$  aus, akzeptiert durch Halt

$(M', N) \in G$  gdw.  $M'$  und  $N$  äquivalent

gdw.  $M'$  hält auf jeder Eingabe gdw.  $M$  hält auf Eingabe 0

gdw.  $M \in H_0$

# Praktische Bedeutung

nach These von Church:

Das Halteproblem ist für kein intuitives Berechnungsmodell entscheidbar.

Es existiert kein Programm, welches für jedes beliebige Programm  $P$  feststellt, ob  $P$  immer (für jede Eingabe) hält.

(Unentscheidbarkeit des Totalitätsproblems)

Es existiert kein Programm, welches für jedes beliebige Paar  $(P, v)$  feststellt, ob das Programm  $P$  bei Eingabe von  $v$  anhält.

(Unentscheidbarkeit des allgemeinen Halteproblems)

Es existiert kein Programm, welches für jedes beliebige Paar  $(P, Q)$  von Programmen feststellt, ob  $P$  und  $Q$  dieselbe Funktion berechnen.

(Unentscheidbarkeit des Gleichheitsproblems)

also: Verifikation während der Entwicklung der Programme notwendig

Diese negativen Aussagen sind nützlich,

z.B. um Verschwendung von Ressourcen zu vermeiden.

# Satz von Rice

Satz [Henry Gordon Rice, 1953]:

Jede nichttriviale Menge  $E \subseteq \mathbb{N}$  ist unentscheidbar.

(Jede nichttriviale semantische Eigenschaft von Programmen ist unentscheidbar.)

- ▶ Eigenschaft: Menge  $E \subseteq \mathbb{N}$  von Gödelnummern (codierten TM oder Programmen)
- ▶ nichttrivial:  $E \neq \emptyset \wedge E \neq \mathbb{N}$
- ▶ semantisch:  $\forall i, j \in \mathbb{N} : (\psi_u(i) = \psi_u(j)) \rightarrow (i \in E \leftrightarrow j \in E)$

(Es kann keinen Algorithmus geben, der bei Eingabe eines Programms  $p$  in endlicher Zeit feststellt, ob die von  $p$  berechnete partielle Funktion  $f_p$  zur Menge  $E$  gehört.)

Beispiele (semantisch oder nicht?)

- ▶ das Programm berechnet eine totale Funktion
- ▶ die Länge des Programmtextes ist gerade
- ▶ das Programm hält bei Eingabe 27
- ▶ zwei Programme berechnen dieselbe Funktion
- ▶ die Gödelnummer ist gerade ( $2\mathbb{N}$ )
- ▶ die berechnete Funktion ist konstant

# Das Postsche Korrespondenzproblem

Emil Post (1897 – 1954)

Ziel: einfach zu definierendes kombinatorisches Problem, das trotzdem schwierig ist.

- ▶ Problem: Eigenschaft einer Folge von Paaren von Wörtern
- ▶ einfach: Definition PCP ist kürzer als Definition TM
- ▶ schwierig:  $PCP \notin REC$

PCP ist Hilfsmittel (für Reduktionen  $\leq_m$ ) für Beweise der Unentscheidbarkeit von Eigenschaften formale Sprachen, logischer Formeln, usw.

## PCP – Definition

**Instanz** des Postschen Korrespondenzproblems  $P \in ((X^+)^2)^+$ :

endliche Folge  $[(x_1, y_1), \dots, (x_k, y_k)]$  von Paaren endlicher Wörter über einem endlichen Alphabet  $X$  ( $X = \{0, 1\}$  genügt)

Beispiel:  $P = [(1, 101), (10, 00), (011, 11)]$

Wort  $w \in \{1, \dots, k\}^+$  heißt **Lösung** der Instanz, wenn

$$x_{w_1} \circ \dots \circ x_{w_{|w|}} = y_{w_1} \circ \dots \circ y_{w_{|w|}}$$

Beispiel:  $w = 1323$  ist Lösung von  $P$ , weil

$$x_1 \circ x_3 \circ x_2 \circ x_3 = 1 \circ 011 \circ 10 \circ 011 = 101110011 = 101 \circ 11 \circ 00 \circ 11 = y_1 \circ y_3 \circ y_2 \circ y_3$$

**PCP** =  $\{P \in ((X^+)^2)^+ \mid \exists w \in \{1, \dots, |P|\}^* : x_{w_1} \circ \dots \circ x_{w_{|w|}} = y_{w_1} \circ \dots \circ y_{w_{|w|}}\}$

(PCP ist die Menge aller lösbaren PCP-Instanzen)

Beispiele:

- ▶  $P = [(1, 111), (10111, 10), (10, 0)]$  hat eine Lösung
- ▶ PCP  $P' = [(001, 0), (01, 011), (01, 101), (10, 001)]$  hat eine Lösung der Länge 66
- ▶  $P'' = [(10, 101), (011, 11), (101, 011)]$  hat keine Lösung

# Modifiziertes PCP

## Spezialfall MPCP

- ▶ Instanzen: endliche Liste  $[(x_1, y_1), \dots, (x_k, y_k)]$  von Paaren endlicher Wörter über einem endlichen Alphabet  $X$
- ▶  $w$  ist Lösung des MPCP gdw.  
 $w$  Lösung des PCP  $[(x_1, y_1), \dots, (x_k, y_k)]$  und  $w_1 = 1$

Satz: MPCP  $\leq_m$  PCP

Beweis:  $X' = X \cup \{\bullet\}$

Reduktion  $f = ((x_1, y_1), (x_2, y_2), \dots, (x_k, y_k)) \mapsto$

$$\left\{ \begin{array}{l} (\bullet x_{11} \bullet x_{12} \bullet \dots \bullet x_{1|x_1|} \bullet, \bullet y_{11} \bullet y_{12} \bullet \dots \bullet y_{1|y_1|} \bullet), \\ (x_{11} \bullet x_{12} \bullet \dots \bullet x_{1|x_1|} \bullet, \bullet y_{11} \bullet y_{12} \bullet \dots \bullet y_{1|y_1|} \bullet), \\ (x_{21} \bullet x_{22} \bullet \dots \bullet x_{2|x_2|} \bullet, \bullet y_{21} \bullet y_{22} \bullet \dots \bullet y_{2|y_2|} \bullet), \\ \vdots \\ (x_{k1} \bullet x_{k2} \bullet \dots \bullet x_{k|x_k|} \bullet, \bullet y_{k1} \bullet y_{k2} \bullet \dots \bullet y_{k|y_k|} \bullet), (\bullet, \bullet\bullet) \end{array} \right\}$$

$f$  ist berechenbar und total

MPCP  $P$  hat eine Lösung gdw. PCP  $f(P)$  eine Lösung hat

Beweis:  $w = [w_1 \dots w_m]$  Lösung für  $P = [(x_1, y_1), \dots, (x_k, y_k)]$

gdw.  $w' = [1, w_2 + 1, \dots, w_m + 1, k + 2]$  Lösung für  $f(P)$

# Unentscheidbarkeit des MPCP

Satz:  $H \leq_m \text{MPCP}$

Beweis: Reduktion durch Konstruktion

eines MPCP  $P = [(x_1, y_1), \dots, (x_n, y_n)]$  aus H-Instanz  $(u, v)$ , so dass:  
 $P$  hat Lösung gdw.  $M_u = (X, Q, \Gamma, \Delta, q_0, F, \square)$  bei Eingabe von  $v$  hält  
( $P$  simuliert die Konfigurationenfolgen der Berechnung von  $M_u$  auf  $v$ )

- ▶  $(x_1, y_1) = (\bullet, \bullet q_0 v \bullet) \in P$  (Startkonfiguration)
- ▶ Kopierregeln:  $\{(a, a) \mid a \in \Gamma \cup \{\bullet\}\} \subseteq P$
- ▶ Überführungsregeln zur Simulation von  $\delta$ , z.B.  
 $(p, a, q, b, R) \in \delta \mapsto (pa, bq) \in P,$   
 $(p, \bullet, q, b, R) \in \delta \mapsto (p\bullet, bq\bullet) \in P,$   
 $(p, a, q, b, L) \in \delta \mapsto \{(xpa, qxb) \mid x \in \Gamma\} \cup \{(\bullet pa, \bullet q \square b)\} \subseteq P$
- ▶ Löschrregeln  $\{(af, f) \mid a \in \Gamma \wedge f \in F\} \cup \{(fa, f) \mid a \in \Gamma \wedge f \in F\} \subseteq P$
- ▶ Abschlussregeln:  $\{(f \bullet \bullet, \bullet) \mid f \in F\} \subseteq P$

$(u, v) \in H$  gdw.  $M_u$  hält bei Eingabe  $v$

gdw.  $\exists$  zulässige Berechnung (Konfigurationenfolge)  $(k_0, \dots, k_t)$  für  $M_u$   
mit  $k_0 = q_0 v$  und Endkonfiguration  $k_t$

gdw.  $\bullet k_0 \bullet k_1 \bullet \dots \bullet k_t \bullet \dots \bullet f \bullet \bullet$  (nach  $k_t$  löschen) ist Lösungswort für  $P$

gdw  $P \in \text{MPCP}$

# Unentscheidbarkeit des PCP

Wir haben schon gezeigt:  $H_S \leq_m H \leq_m \text{MPCP} \leq_m \text{PCP}$

aus  $H_S \notin \text{REC}$  folgt also  $\text{PCP} \notin \text{REC}$

Damit lässt sich die Unentscheidbarkeit weiterer Probleme zeigen,  
z.B. aus den Bereichen

- ▶ formale Grammatiken
- ▶ Logik



# Prädikatenlogik

$\text{FOL}(\Sigma, \mathbb{X})$ : Menge aller prädikatenlogischen Formeln mit

- ▶ Signatur  $\Sigma = (\Sigma_F, \Sigma_R)$  (Funktions- und Relationsymbolen)
- ▶ Variablenmenge  $\mathbb{X}$

Satz:

$$\text{FOL} = \{ \varphi \in \text{FOL}(\Sigma, \mathbb{X}) \mid \varphi \text{ allgemeingültig} \} \in \text{RE}$$

Beweis: Beweiskalküle (natürliches Schließen, Gentzen, Sequenzen)

$\varphi \in \text{FOL}(\Sigma, \mathbb{X})$  allgemeingültig gdw.  $\neg\varphi$  unerfüllbar (ÜA)

Folgerung:

$$U = \{ \varphi \in \text{FOL}(\Sigma, \mathbb{X}) \mid \varphi \text{ unerfüllbar} \} \in \text{RE}$$

# Unentscheidbarkeit der Prädikatenlogik

Satz [Alan Turing und Alonzo Church, 1953]:

$$\text{FOL} = \{\varphi \in \text{FOL}(\Sigma, \mathbb{X}) \mid \varphi \text{ allgemeingültig}\} \notin \text{REC}$$

Beweis: Reduktion  $\text{PCP} \leq_m \text{FOL}$

Idee: Zuordnung von Formeln  $\varphi_P \in \text{FOL}(\Sigma, \mathbb{X})$  zu PCP-Instanzen  $P$  über Alphabet  $\{0, 1\}$  mit  $\varphi_P$  allgemeingültig gdw.  $P$  hat Lösung

geeignete Signatur:  $\Sigma = (\Sigma_F, \Sigma_R)$  mit  $\Sigma_F = \{(c, 0), (f_0, 1), (f_1, 1)\}$ ,  $\Sigma_R = \{(G, 2)\}$

Jedes Wort  $w \in \{0, 1\}^*$  wird repräsentiert durch den Term  $t_w(c) = t_{w_1 \dots w_{|w|}}(c) = f_{w_{|w|}}(\dots(f_{w_1}(c))\dots) \in \text{Term}(\Sigma_F, \emptyset)$

Bsp.:  $w = 01101 \mapsto t_w = t_{01101}(c) = f_1(f_0(f_1(f_1(f_0(c))))))$

# Unentscheidbarkeit der Prädikatenlogik

Zu PCP  $P = [(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)]$

Konstruktion der Formel  $\varphi_P = (\psi_1 \wedge \psi_2) \rightarrow \psi_3$  mit

$$\psi_1 = \bigwedge_{i \in \{1, \dots, n\}} G(t_{x_i}(c), t_{y_i}(c))$$

(Repräsentation der Paare des PCP  $P$ )

$$\psi_2 = \forall u \forall v \left( G(u, v) \rightarrow \bigwedge_{i \in \{1, \dots, n\}} G(t_{x_i}(u), t_{y_i}(v)) \right)$$

(Repräsentation der simultanen Verkettung)

$$\psi_3 = \exists x G(x, x)$$

(Repräsentation der Gleichheit der erzeugten Wörter)

Satz:  $\varphi_P$  allgemeingültig gdw.  $P$  hat Lösung

Also gilt  $\text{PCP} \leq_m \text{FOL}$  (mit Reduktion  $f = P \mapsto \varphi_P$ )

und damit wegen  $\text{PCP} \notin \text{REC}$  auch  $\text{FOL} \notin \text{REC}$

# Was bisher geschah

Codierungen von Paaren, Listen, Bäumen, Programmen

Sprachen und Aufgaben (Probleme) als Mengen

Eigenschaften von **Funktionen**:

**berechenbar** verschiedene Berechnungsmodelle  
Äquivalenz von Berechnungsmodellen

Eigenschaften von **Mengen**:

**entscheidbar** gdw.  $\chi_M$  berechenbar  
gdw.  $M$  und  $\overline{M}$  semi-entscheidbar  
abgeschlossen unter  $\cup, \cap, \overline{\phantom{x}}, \setminus, \circ, *, R$

**semi-entscheidbar** gdw.  $\chi'_M$  berechenbar  
gdw. Turing-akzeptierbar  
gdw. (rekursiv) aufzählbar  
abgeschlossen unter  $\cup, \cap, \circ, *, R$ , aber nicht unter  $\overline{\phantom{x}}$

Reduktion  $\leq_m$  zwischen Problemen (Mengen, Sprachen)

damit auch Unentscheidbarkeitsnachweise

Unentscheidbare Probleme:

- ▶ Halteprobleme (für TM, Programme)
- ▶ PCP
- ▶ Allgemeingültigkeit von FOL-Formeln

# Motivation Komplexitätstheorie

- ▶ bis jetzt: Berechenbarkeit als **qualitativer** Begriff  
(eine Funktion ist berechenbar oder nicht,  
eine Menge ist entscheidbar oder nicht)
- ▶ ab jetzt: **quantitative** Untersuchung:  
für berechenbare Funktionen/entscheidbare Mengen:  
mit welchem Aufwand lässt sich Rechnung durchführen?
- ▶ Komplexität sowohl für deterministische als auch für  
nichtdeterministische Berechnungsmodelle (Suchprobleme)
- ▶ verwendet Methoden der Berechenbarkeitstheorie  
(insbesondere Reduktion, Vollständigkeit)

# Definition

Ressourcen:

z.B. Rechenzeit, Speicherplatz, Kommunikationsaufwand

Komplexität von

**Algorithmen** (Programmen, Maschinen):

Ressourcenverbrauch als **Funktion** der Eingabegröße

Bsp.: die Komplexität von Bubblesort ist quadratisch.

**Problemen** (Aufgaben):

Komplexität für „besten“ Algorithmus,  
der das Problem löst.

Bsp: die Komplexität des Sortierens ist

$\in \Theta(n \mapsto n \cdot \log n)$

Komplexitätstheorie ist die Lehre von der Schwierigkeit von

**Problemen** (Aufgaben).

# Eigenschaften von Zahlen

- ▶ Zahlen sind hier:  $\in \mathbb{N}$ , binärkodiert
- ▶ zusammengesetzte Zahlen (Produkte)  
$$\text{COMP} = \{x \in \mathbb{N} \mid \exists y, z \in \mathbb{N} : y \geq 2 \wedge z \geq 2 \wedge x = y \cdot z\}$$
- ▶ Primzahlen  
$$\text{PRIMES} = \{x \in \mathbb{N} \mid x \geq 2\} \setminus \text{COMP}$$
- ▶ Quadratzahlen  
$$\text{SQUARES} = \{x \in \mathbb{N} \mid \exists y \in \mathbb{N} : x = y^2\}$$
- ▶ Motivation: u.a. Kryptographie  
RSA-Schlüssel  $\in \text{COMP}$ ,  
aber die Ursache (die Faktoren) soll geheim bleiben

# Aussagenlogisches Erfüllbarkeitsproblem

Menge  $AL(P)$  aller aussagenlogischen Formeln mit Aussagenvariablen aus der (endlichen) Menge  $P$

Belegung  $W : P \rightarrow \{0, 1\}$

Belegung  $W : P \rightarrow \{0, 1\}$  erfüllt die Formel  $\varphi$  gdw.  $W(\varphi) = 1$

Modellmenge  $\text{Mod}(\varphi) = \{W : P \rightarrow \{0, 1\} \mid W(\varphi) = 1\}$

**SAT** =  $\{\varphi \in AL(P) \mid \exists W : W(\varphi) = 1\} = \{\varphi \in AL(P) \mid \text{Mod}(\varphi) \neq \emptyset\}$

Beispiele:

- ▶  $((x \rightarrow \neg y) \wedge (x \leftrightarrow y)) \in \text{SAT}$
- ▶  $(x \wedge \neg x) \notin \text{SAT}$

Spezialfälle (durch syntaktische Einschränkungen)

- ▶ CNF-SAT: wie oben für  $\varphi$  in konjunktiver Normalform
- ▶  $k$ -CNF-SAT: ... mit  $\leq k$  Literalen je Klausel

Anwendung / Motivation:

Entwurf und Überprüfung digitaler Schaltungen



# Färbungsproblem

- ▶  $k$ -Färbung eines Graphen  $G = (V, E)$ :  
Abbildung  $f : V \rightarrow \{1, 2, \dots, k\}$
- ▶ Die Färbung  $f$  von  $G = (V, E)$  ist konfliktfrei  
gdw.  $\forall xy \in E : f(x) \neq f(y)$
- ▶  $k\text{COL} := \{G \mid \exists f : f \text{ ist konfliktfreie } k\text{-Färbung von } G \}$

Beispiele:

- ▶  $K_3 \notin 2\text{COL}$
- ▶  $C_5 \notin 2\text{COL}, C_5 \in 3\text{COL}$
- ▶  $K_5 \notin 3\text{COL}, K_5 \in 5\text{COL}$

ÜA:

- ▶ finde  $G$  mit:  $G$  enthält keinen  $K_3$  und  $G \notin 3\text{COL}$
- ▶ jeder Knoten von  $G$  hat  $< k$  Nachbarn  $\Rightarrow G \in k\text{COL}$ .

Anwendung: Ressourcenzuordnungsprobleme,

z.B. Speicherplatz zu Variablen, Frequenzbereiche zu Funkzellen

# Bemerkung zur Genauigkeit

bisher:

- ▶ wegen These von Church und Turing war das Berechnungsmodell beliebig
- ▶ wegen Codierungen (Gödelisierung) war die Art der Eingabe (Zahlen, Wörter usw.) beliebig

jetzt:

- ▶ wegen exakter Ressourcenmessung muss ein Modell fixiert werden (Turingmaschine)
- ▶ wegen Komplexität als Funktion der Eingabegröße muss „Größe“ exakt definiert werden (Länge des Wortes auf dem Eingabeband, Länge der Binärcodierung einer Zahl)

## Deterministische Zeit – P

- ▶ für  $f : \mathbb{N} \rightarrow \mathbb{N}$  bezeichnet  $\text{DTIME}_{\text{Alg}}(f)$   
die Menge aller deterministischen TM (DTM)  $M$  mit  
 $\forall x \in L(M) : M$  akzeptiert  $x$  nach  $\leq f(|x|)$  Schritten.
- ▶ für  $f : \mathbb{N} \rightarrow \mathbb{N}$  bezeichnet  $\text{DTIME}_{\text{Prob}}(f)$   
die Menge aller Sprachen (Probleme)  $L$  mit  
 $\exists M \in \text{DTIME}_{\text{Alg}}(f) : L = L(M)$ .
- ▶ Abkürzung  $\text{DTIME} = \text{DTIME}_{\text{Prob}}$
- ▶ für Menge  $F$  von Funktionen:  $\text{DTIME}(F) = \bigcup_{f \in F} \text{DTIME}(f)$
- ▶ Abkürzung:  $P = \text{PTIME} = \text{DTIME}(\text{Menge aller Polynome})$
- ▶  $\{w \mid w \in \{0,1\}^* \wedge w = w^R\} \in P, 2\text{COL} \in P, 2\text{SAT} \in P$

# Nichtdeterminismus und Orakel

bisher: deterministische Rechnungen

- ▶ Rechnung ist Pfad (Folge von Konfigurationsübergängen)
- ▶ ist erfolgreich, falls finaler Zustand erreicht wird

jetzt: nichtdeterministische Rechnungen (Such-Aufgaben)

- ▶ Rechnung ist Baum (evtl. mehrfach verzweigt)  
Knoten: Konfigurationen
- ▶ erfolgreich, falls ein finales Blatt existiert  
(akzeptierende Konfiguration, Halt)

alternative Sichtweise: Orakel-Berechnung

- ▶ ein Orakel beschreibt den Weg zum finalen Blatt,
- ▶ deterministische Maschine verifiziert diesen Weg

# Orakel-Maschinen

Definition:

**Orakel-TM**  $A = \text{DTM}$  mit Eingabeband (Alphabet  $X$ ), Orakelband (Alphabet  $\Gamma$ ) und evtl. Arbeitsbändern

Definition:

von  $A$  akzeptierte Sprache  $L(A) \subseteq \Sigma^*$ :

Menge aller Eingabewörter  $x$ , für die ein Orakelwort  $y$  existiert, so dass die Rechnung  $M(x, y)$  erfolgreich hält.

nach dieser Definition:

1. Orakel sieht  $x$  und schreibt  $y$ ,
2. danach rechnet (*verifiziert*)  $A$  (deterministisch)

alternative Sichtweise:

$y$  ist unbekannt, bei jedem Lesen eines unbekanntes Zeichens von  $y$  verzweigt die Rechnung:  $A$  ist *nichtdeterministische* TM

die Rechnung  $A(x)$  ist ein Baum  $T$ ,

Eingabe wird akzeptiert, wenn  $T$  ein erfolgreiches Blatt enthält

## Nichtdeterministische Zeit – NP

- ▶ für  $f : \mathbb{N} \rightarrow \mathbb{N}$  bezeichnet  $\text{NTIME}_{\text{Alg}}(f)$  die Menge der Orakel-TM  $M$  mit  $\forall x : x \in L(M)$  gdw. es gibt ein Orakelwort  $y$  mit:  $M$  akzeptiert  $(x, y)$  in  $\leq f(|x|)$  Schritten.  
(dabei werden  $\leq f(|x|)$  Zeichen von  $y$  gelesen)
- ▶ entsprechend  $\text{NTIME}_{\text{Prob}}(f)$ ,  $\text{NTIME}(f)$ ,  $\text{NTIME}(F)$
- ▶ Abkürzung  $\text{NP} = \text{NPTIME} = \text{NTIME}(\text{Polynome})$
- ▶ Bsp.:  $\text{COMP} \in \text{NP}$ ,  $\text{SAT} \in \text{NP}$ ,  $\forall k : k\text{COL} \in \text{NP}$
- ▶ Satz:  $A \in \text{NP} \wedge B \in \text{NP} \Rightarrow (A \cup B) \in \text{NP}$
- ▶ Satz:  $\text{P} \subseteq \text{NP}$
- ▶ Frage:  $\text{P} = \text{NP}$  ?  
(schwierig:  $10^6$  \$, <http://www.claymath.org/millennium-problems/p-vs-np-problem>)
- ▶ für Sprachklasse  $C$ :  $\text{co}C = \{L \mid \bar{L} \in C\}$ , z.B.  $\text{coNP}$ ,  $\text{coRE}$

# Was bisher geschah

## Berechenbarkeitstheorie:

- ▶ Funktionen: total, berechenbar
- ▶ Mengen: entscheidbar, aufzählbar
- ▶ Reduktion  $\leq_m$ :  $\forall L, L' \subseteq \mathbb{N}$ :  
 $L \leq_m L'$  gdw.  $\exists (f : \mathbb{N} \rightarrow \mathbb{N}) \forall x \in \mathbb{N} : (x \in L \leftrightarrow f(x) \in L')$   
wobei  $f$  total und berechenbar
- ▶ Einordnung verschiedener Probleme,  
z.B. Halteprobleme für TM, PCP, FOL (allgemeingültige Formeln)

## Komplexitätstheorie:

- ▶ Funktionen: mit Ressourcen ... berechenbar
- ▶ Mengen: mit Ressourcen ... entscheidbar
- ▶ Probleme: SQUARES, PRIMES, SAT, kSAT, k-COL
- ▶  $\text{DTIME}(f)$ ,  $\text{P} = \text{DTIME}(\text{Menge aller Polynome})$ , 2-Col, 2SAT  $\in \text{P}$
- ▶ Orakelmaschinen,  $\text{NTIME}(f)$
- ▶  $\text{NTIME}(\text{Polynome}) = \text{NPTIME} = \text{NP}$
- ▶ Beispiele für Probleme in NP: SAT, CNFSAT, kCOL, HC

## Die Klasse coNP

- ▶ Def:  $\text{coNP} = \{L \mid \bar{L} \in \text{NP}\} = \{L \mid (A^* \setminus L) \in \text{NP}\}$
- ▶ Bsp:  $\text{PRIMES} \in \text{coNP}$  wegen  $\text{COMP} \in \text{NP}$
- ▶ es gilt auch  $\text{PRIMES} \in \text{NP}$ , Beweis benötigt etwas Zahlentheorie
- ▶  $\text{PRIMES} \in \text{P}$ , Manindra Agrawal, Neeraj Kayal, Nitin Saxena, 2004.  
<http://annals.math.princeton.edu/2004/160-2/p12>



# Vergleich: Berechenbarkeit/Komplexität

Analogien zwischen:

- ▶ Berechenbarkeit: REC, RE, coRE
- ▶ Komplexität: P, NP, coNP

Unterschied:

$REC \neq RE$ ,  $RE \neq coRE$  und  $RE \cap coRE = REC$  kann man beweisen,  
die entsprechenden Komplexitätsaussagen bisher nicht

Werkzeuge:

- ▶ Reduktion  
(zum Vergleich der Komplexität von Problemen)
- ▶ Vollständigkeit  
(zur Definition der schwersten Probleme einer Klasse)

# NP-vollständige Probleme

nach bisherigen Definitionen/Beispielen:

- ▶ P: Klasse der (auf sequentiellen Maschinen) effizient lösbaren Probleme
- ▶ NP: enthält häufig vorkommende Suchprobleme

Frage für (neues) Problem  $L$ :  $L \in P$  oder  $L \notin P$ ?

- ▶ bis heute ist kein  $L \in NP \setminus P$  bekannt
- ▶ Bekannt ist jedoch eine Charakterisierung der **schwersten** Probleme in NP:  
die Klasse **NP<sub>c</sub>** der **NP-vollständigen Probleme**

## Polynomialzeit-Reduktion $\leq_P$

Def:  $\text{FTIME}(f)$  = die in Zeit  $\leq f(|x|)$  auf DTM berechenbaren Funktionen,  $\text{FP} = \text{FTIME}(\text{poly})$ .

Def:  $A \leq_P B$  gdw.  $\exists f \in \text{FP} : \forall x \in X^* : x \in A \leftrightarrow f(x) \in B$ .

Beispiel:  $\text{HC} \leq_P \text{DHC}$ .

$\text{HC} = \{G \mid G \text{ ist ungerichteter Graph und } G \text{ enth\u00e4lt Kreis durch alle Knoten}\}$  (Hamiltonian Circuit)

$\text{DHC} = \{G \mid G \text{ ist gerichteter Graph und } G \text{ enth\u00e4lt gerichteten Kreis durch alle Knoten}\}$  (Directed HC)

Beweis: Reduktion (gerichteter  $\mapsto$  ungerichteter Graph)

$f(V, E) = (V \times \{1, 2, 3\}, E')$  mit

$E' = \{ \{(v, 1), (v, 2)\} \mid v \in V \} \cup \{ \{(v, 2), (v, 3)\} \mid v \in V \}$   
 $\cup \{ \{(u, 3), (v, 1)\} \mid (u, v) \in E \}$ .

zu zeigen sind f\u00fcr Reduktion  $f$ : Korrektheit, Laufzeit

Bemerkung: Eulerkreis  $\in P$  (Warum?)

Satz:  $\leq_P$  ist transitiv

## Abschluss-Eigenschaften von $\leq_P$

Satz:  $A \leq_P B \wedge B \in P \rightarrow A \in P$

Beweis: wegen Abschluss von  $P$  bzgl. Komposition für

- ▶ Reduktionsfunktion  $f$  von  $A$  nach  $B$
- ▶ charakteristische Fkt.  $\chi_B$  von  $B$

Satz:  $A \leq_P B \wedge B \in NP \rightarrow A \in NP$

Beweis: sei  $f$  die Reduktionsfunktion von  $A$  nach  $B$ ,

$\chi_B$  wird Orakel-berechnet  $M : (x, y) \mapsto \{0, 1\}$ .

Dann wird  $\chi_A$  Orakel-berechnet durch  $(x, y) \mapsto M(f(x), y)$   
zu betrachten sind: Korrektheit, Laufzeit, Orakelgröße.

vgl.  $A \leq_m B \wedge B \in REC \rightarrow A \in REC$ ,

$A \leq_m B \wedge B \in RE \rightarrow A \in RE$ .

# NPc und Satz von Cook

Definition: Problem (Sprache)  $B$  heißt

**NP-schwer** gdw.  $\forall A \in \text{NP} : A \leq_p B$

**NP-vollständig** gdw.  $B \in \text{NP}$  und  $B$  NP-schwer

**NPc** =  $\{B \mid B \text{ ist NP-vollständig}\}$

Satz (Steven Cook, 1971):  $\text{SAT} \in \text{NPc}$

Beweis:

1.  $\text{SAT} \in \text{NP}$ : Eingabe  $\varphi \in \text{AL}$  auf Band (codiert)  
Orakelmaschine  $M$ 
  - ▶  $M$  liest  $\varphi$  und stellt Variablenmenge  $\text{var}(\varphi) = \{x_1, \dots, x_n\}$  fest
  - ▶ Orakel rät eine der  $2^n$  Belegungen  $W : \{x_1, \dots, x_n\} \rightarrow \{0, 1\}$
  - ▶  $M$  berechnet Wert von  $W(\varphi)$ ,
  - ▶  $M$  akzeptiert gdw.  $W(\varphi) = 1$

Korrektheit:  $\varphi \in \text{SAT}$  gdw.  $M$  akzeptiert  $\varphi$

Laufzeit polynomiell

2.  $\text{SAT}$  ist NP-schwer ( noch zu zeigen )

# SAT $\in$ NP<sub>C</sub>

Satz (Steven Cook, 1971): SAT  $\in$  NP<sub>C</sub>

1. SAT  $\in$  NP ( schon gezeigt )
2. SAT ist NP-schwer
  - ▶ Wähle  $L \in$  NP beliebig
  - ▶ Dann existieren nichtdet. TM  $M = (X, Q, \Gamma, \delta, q_0, F, \square)$  mit  $L(M) = L$  und Polynom  $p$ ,  
so dass Laufzeit von  $M$  auf  $x$  durch  $p(|x|)$  beschränkt ist
  - ▶ Idee: Konstruktion einer Formel  $\psi_x \in$  AL zu Eingabewort  $x$ ,  
welche die Konfigurationenfolge einer erfolgreichen Rechnung  
von  $M$  mit  $\leq p(|x|)$  Schritten bei Eingabe  $x$  beschreibt.  
( $x \in L$  gdw.  $\psi_x$  erfüllbar)

# Satz von Cook (Einzelheiten)

Aussagenlogische Formel  $\psi_x$ , welche die Konfigurationsfolge einer erfolgreichen Rechnung von  $M = (X, Q, \Gamma, \delta, q_0, F, \square)$  mit  $\leq p(|x|) = T$  Schritten bei Eingabe  $x$  beschreibt.

- ▶ Variablen (mit  $t \in \{0, \dots, T\}, i \in \{-T, \dots, T\}, j \in X \cup \Gamma, k \in Q$ )

$b_{t,i,j}$  zur Zeit  $t$  steht in Zelle  $i$  das Zeichen  $j$

$p_{t,i}$  zur Zeit  $t$  steht Kopf an Position  $i$

$z_{t,k}$  zur Zeit  $t$  ist Maschine in Zustand  $k$

- ▶  $\psi_x = \psi_b \wedge \psi_k \wedge \psi_z \wedge \psi_s \wedge \psi_e \wedge \psi_t$  mit Teilformeln zur Beschreibung

- ▶ der Konfigurationen der TM  $M = (X, Q, \Gamma, \delta, q_0, F, \square)$

$\psi_b$  : in jeder Zelle steht immer genau ein Symbol aus  $X \cup \Gamma$ ,

$\psi_k$  : Kopf immer an genau einer Position,

$\psi_z$  : Maschine immer in genau einem Zustand

- ▶ der Berechnung (Konfigurationsfolge) von  $M$  auf  $x$ :

$\psi_s$  : Anfang in Startkonfiguration mit Eingabe  $x$

$\psi_e$  : Ende in einer akzeptierenden Konfiguration

$\psi_t$  : nur zulässige Konfigurationenübergänge

## Satz von Cook (Einzelheiten)

Abkürzung:  $\text{exactlyone}(x_1, \dots, x_n)$  wahr gdw.  $x_i$  für genau ein  $i$  wahr

$$\psi_k = \bigwedge_{t \in \{0, T\}} \text{exactlyone}(p_{t,-T}, \dots, p_{t,T}) \quad (\psi_b, \psi_z \text{ analog})$$

$$\psi_s = z_{0,q_0} \wedge p_{0,0} \wedge \bigwedge_{i \in \{1, \dots, |x|\}} b_{0,i-1,x_i} \wedge \bigwedge_{i \in \{-T, \dots, -1\} \cup \{|x|, \dots, T\}} b_{0,i,\square}$$

$$\psi'_t = \bigwedge_{\substack{t \in \{0, \dots, T\} \\ k \in Q \\ i \in \{-T, \dots, T\} \\ a \in X \cup \Gamma}} \left( \begin{array}{c} (z_{t,k} \wedge p_{t,i} \wedge b_{t,i,a}) \\ \rightarrow \\ \bigvee_{(a,k,b,k',m) \in \delta} (z_{t+1,k'} \wedge p_{t+1,i+m} \wedge b_{t+1,i,b}) \end{array} \right)$$

$$\psi_x = \psi_b \wedge \psi_k \wedge \psi_z \wedge \psi_s \wedge \psi_e \wedge \psi'_t \wedge \dots \quad (\text{ÜA: Ergänzung})$$

Es gilt:

1.  $\psi_x$  erfüllbar gdw. akzeptierende Berechnung von  $M$  auf  $x$  existiert
2.  $\psi_x$  aus  $x$  in polynomieller Zeit berechenbar

Beobachtung: alle  $\psi_*$  in CNF, also auch  $\psi_x$  in CNF

Diese Reduktion ist also zugleich eine Reduktion auf CNFSAT



# CNFSAT $\leq_p$ 3SAT

Satz: 3SAT  $\in$  NP<sub>c</sub>

Beweis:

1. 3SAT  $\in$  NP.

weil CNFSAT  $\in$  NP und 3SAT  $\subseteq$  CNFSAT

2. 3SAT ist NP-schwer.

weil CNFSAT NP-schwer und Reduktion CNFSAT  $\leq_p$  3SAT

Konstruktion von 3CNF  $\varphi'$  aus CNF  $\varphi$  durch Transformation der Klauseln in  $\varphi$

$$x \mapsto x \vee x \vee x$$

$$x \vee y \mapsto \dots$$

$$x \vee y \vee z \mapsto \dots$$

$$x \vee y \vee z \vee u \mapsto (x \vee y \vee h) \wedge (\neg h \vee z \vee u)$$

$$x \vee y \vee z \vee u \vee v \mapsto (x \vee y \vee h) \wedge (\neg h \vee z \vee h') \wedge (\neg h' \vee u \vee v)$$

...

## Weitere Komplexitätsklassen

- ▶  $\text{coNP} = \{L \mid \bar{L} \in \text{NP}\}$   
Bsp.:  $\overline{\text{SAT}} = \{\varphi \in \text{AL} \mid \varphi \text{ unerfüllbar}\}$   
 $= \{\varphi \in \text{AL} \mid \neg\varphi \text{ allgemeingültig}\}$
- ▶ für jede deterministische Zeitklasse  $C$  gilt  $\text{co}C = C$   
(Akzeptanz / Ablehnung durch dieselbe Funktion)
- ▶ PSPACE
- ▶ EXPTIME / EXPSPACE

# Deterministischer Platz

- ▶ TM mit Eingabe-, Ausgabe- und Arbeitsband
- ▶ Platz wird **nur auf dem Arbeitsband** gemessen (auf Ein- und Ausgabeband nicht)
- ▶ für  $f : \mathbb{N} \rightarrow \mathbb{N}$  bezeichnet  $\text{DSPACE}_{\text{Alg}}(f)$  die Menge aller DTM  $M$  mit  $\forall x \in X^*$ :  
 $x \in L(M)$  gdw.  $M$  akzeptiert  $x$  mit Arbeitsband der Breite  $\leq f(|x|)$
- ▶ für  $f : \mathbb{N} \rightarrow \mathbb{N}$  bezeichnet  $\text{DSPACE}_{\text{Prob}}(f)$  die Menge der Sprachen  $L$  mit  $\exists M \in \text{DSPACE}_{\text{Alg}}(f) : L = L(M)$
- ▶ Abkürzung  $\text{DSPACE} = \text{DSPACE}_{\text{Prob}}$
- ▶ für Menge  $F$  von Funktionen:  $\text{DSPACE}(F) = \bigcup_{f \in F} \text{DSPACE}(f)$
- ▶ Abkürzung  $\text{PSPACE} = \text{DSPACE}(\text{poly})$

## Beispiele für DSPACE: DSPACE(0)

- ▶ Eingabe ist read-only, Ausgabe ist write-only,
- ▶ Arbeitsband darf nicht benutzt werden (Breite 0)  
einzige Speichermöglichkeit ist der Zustand
- ▶ das sind endliche Automaten (mit Ein- und Ausgabe)
- ▶ Bsp: Berechnung des Nachfolgers in Binärdarstellung

## Beispiele für DSPACE: QBF $\in$ PSPACE

- ▶ QBF = Menge aller wahren aussagenlogischen Formeln mit Quantoren ohne freie Variablen
- ▶ Syntax:  $t|f|p|\neg\varphi|\varphi * \psi|Qp\varphi$   
mit Aussagenvariablen  $p$ ,  $*$  zweistelliger Junktor,  $Q \in \{\forall, \exists\}$   
Bsp:  $\varphi_1 = \forall x \exists y ((x \vee y) \wedge (\neg x \vee \neg y))$ ,  
 $\varphi_2 = \exists x \forall y ((x \vee y) \wedge (\neg x \vee \neg y))$
- ▶ Semantik: Auswertung der Formel rekursiv:  
 $\text{eval}(\forall x \varphi) = \text{eval}(\varphi[x := 0]) \wedge \text{eval}(\varphi[x := 1])$   
 $\text{eval}(\exists x \varphi) = \text{eval}(\varphi[x := 0]) \vee \text{eval}(\varphi[x := 1])$   
Bsp:  $\text{eval}(\varphi_1) = \dots$

Satz: QBF  $\in$  PSPACE

Beweis:

- ▶ Auswertung mit linearem Platzbedarf möglich
- ▶ benutze Keller (auf Arbeitsband) für die Verwaltung der aktuell untersuchten Variablenbelegungen, Kellertiefe ist  $\leq |\varphi|$
- ▶ die Ressource Platz kann man **nachnutzen**  
... und dabei sehr viel mehr Zeit verbrauchen

## Beispiele: Spiele in PSPACE

- ▶ QBF für Formeln der Gestalt  
 $\forall x_1 \exists y_1 \forall x_2 \exists y_2 \cdots \forall x_n \exists y_n : P(x_1, y_1, x_2, y_2, \dots, x_n, y_n)$   
als Zweipersonenspiel: für den ersten Zug von  $x$  gibt es einen Antwortzug für  $y$ , so dass für jeden ...  
und  $P(\dots)$  beschreibt die Gewinnbedingung
- ▶ viele andere Zweipersonenspiele  $\in$  PSPACE  
genauer, das Entscheidungsproblem  
 $\{s \mid \text{Position } s \text{ ist sicherer Gewinn für Spieler 2}\}$   
wenn die Länge des Spiels polynomiell beschränkt ist

Beispiele:

- ▶ Stefan Reisch: *HEX ist PSPACE-vollständig*,  
Acta Inf. 15(2)1981, 167–191
- ▶ Othello / Reversi: S. Iwata and T. Kasai:  
*The Othello game on an  $n \times n$  board is PSPACE-complete*,  
Theor. Comp. Sci. 123 (1994) 329-340.

# Beziehungen zwischen Platz und Zeit

Satz:  $\text{DTIME}(f) \subseteq \text{DSPACE}(f)$ . (Warum?)

Satz:  $\text{DSPACE}(f) \subseteq \text{DTIME}(2^{O(f)})$   
(dabei  $2^{O(f)} := \bigcup_{c>0} (n \mapsto 2^{c \cdot f(n)})$ )

Beweis: sei  $M \in \text{DSPACE}_{\text{Alg}}(f)$  (mit Alphabet  $\{0, 1\}$ ).  
Wenn  $M(x)$  mit  $f(|x|)$  Platz akzeptiert,  
dann sind alle Konfigurationen verschieden (Warum?)  
Es gibt  $\leq 2^{f(|x|)}$  Konfigurationen,  
also höchstens so viele Schritte.

Folgerung:  $f$  berechenbar  $\rightarrow \text{DSPACE}(f) \subseteq \text{REC}$

Beweis:  $\text{DSPACE}(f) \subseteq \text{DTIME}(2^{O(f)}) \subseteq \text{REC}$

# Reduktion, Vollständigkeit

- ▶ Def.  $\leq_P$  wie bisher
- ▶ Satz: PSPACE ist abgeschlossen bzgl.  $\leq_P$ :  
 $\forall L_1, L_2 : L_1 \leq_P L_2 \wedge L_2 \in \text{PSPACE} \Rightarrow L_1 \in \text{PSPACE}$   
Beweis: Reduktionsfunktion  $f$  ist in polynomieller Zeit, also in polynomiellem Platz berechenbar.
- ▶ Def:  $L$  ist PSPACE-vollständig, gdw.  
 $L \in \text{PSPACE} \wedge \forall L' \in \text{PSPACE} : L' \leq_P L$



# QBF ist ein „schwerstes“ Problem in PSPACE

Satz: QBF ist PSPACE-vollständig.

Beweis:

1. QBF  $\in$  PSPACE schon gezeigt

2. QBF ist PSPACE-schwer, d.h.  $\forall L \in \text{PSPACE} : L \leq_P \text{QBF}$

Idee: Reduktionsfunktion  $f$  transformiert PSPACE-DTM  $M$  in QBF-Formel  $\varphi$ , so dass  $\varphi$  wahr gdw.  $M$  hält bei Eingabe  $x$ , d.h.  $\exists$  Berechnung (Konf.-Folge) von Start- zu einer akz. Konf. (enthält  $\leq 2^{|x|}$  Konfigurationen)

$\varphi = r(c_s, c_a)$  wahr gdw. akz. Konf.  $c_a$  aus Start  $c_s$  erreichbar

Idee:  $r_{|x|}(c, c')$  für Erreichbarkeit in  $\leq 2^{|x|}$  Schritten rekursiv:

$r_{i+1}(c, c') =$

$\exists c_i \forall d_i \forall d'_i (((d = c \wedge d' = c_i) \vee (d_i = c_i \wedge d'_i = c')) \rightarrow r_i(d_i, d'_i))$

$r_0(c, c')$  repräsentiert  $M$ , insbes.  $\delta$  (analog Satz von Cook)

in jedem der  $|x|$  Schritte (poly): zusätzliche Variablen  $c_i, d_i, d'_i$ ,  
 $|r_i(c, c')| \leq |r_{i-1}(c, c')| + O(|x|) \dots$ , also  $|r_{|x|}(c, c')| \leq O(|x|^2)$

(Warum nicht einfach  $\exists c_0 \exists c_1 \dots \exists c_k (\delta(c_s, c_1) \wedge \dots \wedge \delta(c_k, c_a))$ ?)

# PSPACE-vollständige Einpersonenspiele

- ▶  $M \in \text{PSPACE}$  akzeptiert  $x$   
     $\iff$  es gibt eine passende Konfigurationsfolge.
- ▶ Platz für jede Konf.  $\leq f(|x|)$  mit  $f \in \text{poly}$ , Länge  $\leq 2^{O(f)}$
- ▶ Konfiguration kann auch die Anordnung von Spielsteinen auf einem (beschränkten) Brett sein
- ▶ Übergang = Spielzug = (z.B.) Verschieben von Steinen
- ▶ Joseph C. Culberson: *Sokoban is PSPACE-complete*, 1997 ,  
<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.52.41>
- ▶ Rush Hour ist PSPACE-vollständig.

# Nichtdeterministischer Platz

- ▶ Def:  $\text{NSPACE}(f)$  : Probleme lösbar durch nichtdeterministische TM mit Arbeitsband  $\leq f(|\text{Eingabe}|)$
- ▶ Satz  $\text{DSPACE}(f) \subseteq \text{NSPACE}(f)$ . (Warum ?)
- ▶ Satz (Savitch)  $\text{NSPACE}(f) \subseteq \text{DSPACE}(f^2)$ .
- ▶ Folgerung:  $\text{DSPACE}(\text{poly}) = \text{NSPACE}(\text{poly}) = \text{PSPACE}$