# 9b
# WHILE- und GOTO-Programme

Prof. Dr. Jasmin Blanchette

Lehr- und Forschungseinheit für
Theoretische Informatik und Theorembeweisen

Stand: 4. April 2024
Basiert auf Folien von PD Dr. David Sabel

# Syntax von WHILE-Programmen

WHILE-Programme werden durch die kontextfreie Grammatik $(V, \Sigma, P, Prg)$ erzeugt, wobei:

$$
\begin{aligned}
V &= \{Prg, Var, Id, Const\} \\
\Sigma &= \{\textbf{WHILE}, \textbf{LOOP}, \textbf{DO}, \textbf{END}, x, 0, \ldots, 9, ;, :=, +, -\} \\
P &= \{Prg \quad \to \quad \textbf{WHILE}\ Var \neq 0\ \textbf{DO}\ Prg\ \textbf{END} \\
&\qquad\quad \mid \textbf{LOOP}\ Var\ \textbf{DO}\ Prg\ \textbf{END} \\
&\qquad\quad \mid Prg; Prg \\
&\qquad\quad \mid Var := Var + Const \\
&\qquad\quad \mid Var := Var - Const, \\
&\quad Var \quad \to x_{Id}, \\
&\quad Const \to Id, \\
&\quad Id \quad \to 0 \mid 1 \mid \ldots \mid 9 \mid 1Id \mid 2Id \mid \ldots\ 9Id\}
\end{aligned}
$$

Beachte: WHILE-Programme erweitern LOOP-Programme um das **WHILE**-Konstrukt.

# Semantik von WHILE-Programmen

**Definition**

Die Berechnungsschritte $(\rho, P) \xrightarrow[\text{WHILE}]{} (\rho', P')$, wobei $\rho, \rho'$ Variablenbelegungen und $P, P'$ WHILE-Programme oder das leere Programm $\varepsilon$ sind, sind durch folgende Regeln definiert:

- $(\rho, x_i := x_j + c) \xrightarrow[\text{WHILE}]{} (\rho', \varepsilon)$, wobei $\rho' = \rho\{x_i \mapsto \rho(x_j) + c\}$
- $(\rho, x_i := x_j - c) \xrightarrow[\text{WHILE}]{} (\rho', \varepsilon)$, wobei $\rho' = \rho\{x_i \mapsto \max(0, \rho(x_j) - c)\}$
- $(\rho, P_1; P_2) \xrightarrow[\text{WHILE}]{} (\rho', P_2)$ wenn $(\rho, P_1) \xrightarrow[\text{WHILE}]{} (\rho', \varepsilon)$
- $(\rho, P_1; P_2) \xrightarrow[\text{WHILE}]{} (\rho', P_1'; P_2)$ wenn $(\rho, P_1) \xrightarrow[\text{WHILE}]{} (\rho', P_1')$ und $P_1' \neq \varepsilon$
- $(\rho, \textbf{LOOP } x_i \textbf{ DO } P \textbf{ END}) \xrightarrow[\text{WHILE}]{} (\rho, \underbrace{P; \ldots; P}_{\rho(x_i)\text{-mal}})$

# Semantik von WHILE-Programmen

**Definition**

- $(\rho, \textbf{WHILE } x_i \neq 0 \textbf{ DO } P \textbf{ END}) \xrightarrow[\text{WHILE}]{} (\rho, \varepsilon)$ wenn $\rho(x_i) = 0$
- $(\rho, \textbf{WHILE } x_i \neq 0 \textbf{ DO } P \textbf{ END}) \xrightarrow[\text{WHILE}]{} (\rho, P; \textbf{WHILE } x_i \neq 0 \textbf{ DO } P \textbf{ END})$ wenn $\rho(x_i) \neq 0$

# Semantik von WHILE-Programmen

**Definition**

- $(\rho, \textbf{WHILE } x_i \neq 0 \textbf{ DO } P \textbf{ END}) \xrightarrow[\text{WHILE}]{} (\rho, \varepsilon)$ wenn $\rho(x_i) = 0$
- $(\rho, \textbf{WHILE } x_i \neq 0 \textbf{ DO } P \textbf{ END}) \xrightarrow[\text{WHILE}]{} (\rho, P; \textbf{WHILE } x_i \neq 0 \textbf{ DO } P \textbf{ END})$ wenn $\rho(x_i) \neq 0$

Wir schreiben $\xrightarrow[\text{WHILE}]{}^{i}$ für $i$ Schritte und $\xrightarrow[\text{WHILE}]{}^{*}$ für 0 oder beliebig viele Schritte.

# WHILE-Berechenbarkeit

**Definition**

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt WHILE-berechenbar, wenn es ein WHILE-Programm $P$ gibt, das die folgenden Bedingungen erfüllt:

- Für alle $n_1, \ldots, n_k \in \mathbb{N}$, sodass $f(n_1, \ldots, n_k)$ definiert ist, gilt $(\rho, P) \xrightarrow[\text{WHILE}]{}^{*} (\rho', \varepsilon)$, wobei $\rho = \{x_1 \mapsto n_1, \ldots, x_k \mapsto n_k\}$ und $\rho'(x_0) = f(n_1, \ldots, n_k)$.
- Falls $f(n_1, \ldots, n_k)$ nicht definiert ist, stoppt das Programm $P$ nicht, d.h. für alle $i \in \mathbb{N}$ gibt es $\rho'$ und $P'$, sodass $(\rho, P) \xrightarrow[\text{WHILE}]{}^{i} (\rho', P')$.

# LOOP-berechenbar impliziert WHILE-berechenbar

**Satz**

Jede LOOP-berechenbare Funktion ist auch WHILE-berechenbar.

# LOOP-berechenbar impliziert WHILE-berechenbar

**Satz**

Jede LOOP-berechenbare Funktion ist auch WHILE-berechenbar.

**Beweis** Das ist offensichtlich, da jedes LOOP-Programm auch ein WHILE-Programm mit derselben Semantik ist. □

# **LOOP**-Schleife durch **WHILE**-Schleife ersetzen

Das **LOOP**-Konstrukt ist überflüssig in WHILE: Ersetze

**LOOP** $x_i$ **DO** $P$ **END**

durch

$x_m := x_i$;
**WHILE** $x_m \neq 0$ **DO** $x_m := x_m - 1$; $P$ **END**

wobei $x_m$ nirgends sonst vorkommt.

# WHILE-berechenbar impliziert turingberechenbar

**Theorem**

Jede WHILE-berechenbare Funktion ist auch turingberechenbar.

# WHILE-berechenbar impliziert turingberechenbar

**Theorem**

Jede WHILE-berechenbare Funktion ist auch turingberechenbar.

**Beweis** Wir haben bereits gezeigt, dass Zuweisung, Hintereinanderschaltung und **WHILE** durch Turingmaschinen darstellbar sind.

# WHILE-berechenbar impliziert turingberechenbar

**Theorem**

Jede WHILE-berechenbare Funktion ist auch turingberechenbar.

**Beweis** Wir haben bereits gezeigt, dass Zuweisung, Hintereinanderschaltung und **WHILE** durch Turingmaschinen darstellbar sind.

Dabei liegen die Variablen $x_i$ jeweils auf Band $i$ einer Mehrband-Turingmaschine.

# WHILE-berechenbar impliziert turingberechenbar

**Theorem**

Jede WHILE-berechenbare Funktion ist auch turingberechenbar.

**Beweis** Wir haben bereits gezeigt, dass Zuweisung, Hintereinanderschaltung und **WHILE** durch Turingmaschinen darstellbar sind.

Dabei liegen die Variablen $x_i$ jeweils auf Band $i$ einer Mehrband-Turingmaschine.

**LOOP** wird vorher in **LOOP**-freies WHILE-Programm transformiert. □

# Syntax von GOTO-Programmen

GOTO-Programme werden durch die kontextfreie Grammatik $(V, \Sigma, P, Prg)$ erzeugt, wobei:

$$
\begin{aligned}
V &= \{Prg, Var, Id, Const\} \\
\Sigma &= \{\textbf{GOTO}, \textbf{HALT}, \textbf{IF}, \textbf{THEN}, x, 0, \ldots, 9, ;, :=, +, -\} \\
P = \{Prg \quad &\to \quad M_{Id}\text{: } \textbf{GOTO } M_{Id} \\
&\quad\;\; \mid M_{Id}\text{: } \textbf{IF } x_i = 0 \textbf{ THEN GOTO } M_{Id} \\
&\quad\;\; \mid M_{Id}\text{: } \textbf{HALT} \\
&\quad\;\; \mid M_{Id}\text{: } Var := Var + Const \\
&\quad\;\; \mid M_{Id}\text{: } Var := Var - Const \\
&\quad\;\; \mid Prg; Prg, \\
Var \quad &\to x_{Id}, \\
Const \quad &\to Id, \\
Id \quad &\to 0 \mid 1 \mid \ldots \mid 9 \mid 1Id \mid 2Id \mid \ldots \; 9Id\}
\end{aligned}
$$

Befehle sind mit verschiedenen $M_i$: markiert. Wir lassen sie manchmal weg.

# Semantik von GOTO-Programmen

**Definition**

Seien $P_0$ ein normalisiertes GOTO-Programm, $P$ ein GOTO-Programm und $\rho$ eine Variablenbelegung. Dann sind die Berechnungsschritte $\xrightarrow[\text{GOTO}]{P_0}$ durch folgende Regeln definiert:

- $(\rho, M_i\colon x_j := x_k + c; P) \xrightarrow[\text{GOTO}]{P_0} (\rho', P)$, wobei $\rho' = \rho\{x_j \mapsto \rho(x_k) + c\}$
- $(\rho, M_i\colon x_j := x_k + c) \xrightarrow[\text{GOTO}]{P_0} (\rho', \varepsilon)$, wobei $\rho' = \rho\{x_j \mapsto \rho(x_k) + c\}$
- $(\rho, M_i\colon x_j := x_k - c; P) \xrightarrow[\text{GOTO}]{P_0} (\rho', P)$, wobei $\rho' = \rho\{x_j \mapsto \max(0, \rho(x_k) - c)\}$
- $(\rho, M_i\colon x_j := x_k - c) \xrightarrow[\text{GOTO}]{P_0} (\rho', \varepsilon)$, wobei $\rho' = \rho\{x_j \mapsto \max(0, \rho(x_k) - c)\}$
- $(\rho, M_i\colon \textbf{GOTO } M_k; P) \xrightarrow[\text{GOTO}]{M_1\colon A_1; \ldots; M_n\colon A_n} (\rho, M_k\colon A_k; \ldots; M_n\colon A_n)$ wenn $1 \leq k \leq n$
- $(\rho, M_i\colon \textbf{GOTO } M_k) \xrightarrow[\text{GOTO}]{M_1\colon A_1; \ldots; M_n\colon A_n} (\rho, M_k\colon A_k; \ldots; M_n\colon A_n)$ wenn $1 \leq k \leq n$

# Semantik von GOTO-Programmen

**Definition**

- $(\rho, M_i\colon \textbf{IF } x_r = 0 \textbf{ THEN GOTO } M_k; P) \xrightarrow[\text{GOTO}]{M_1\colon A_1;\ldots;M_n\colon A_n} (\rho, M_k\colon A_k; \ldots; M_n\colon A_n)$ wenn $1 \leq k \leq n$ und $\rho(x_r) = 0$
- $(\rho, M_i\colon \textbf{IF } x_r = 0 \textbf{ THEN GOTO } M_k) \xrightarrow[\text{GOTO}]{M_1\colon A_1;\ldots;M_n\colon A_n} (\rho, M_k\colon A_k; \ldots; M_n\colon A_n)$ wenn $1 \leq k \leq n$ und $\rho(x_r) = 0$
- $(\rho, M_i\colon \textbf{IF } x_r = 0 \textbf{ THEN GOTO } M_k; P) \xrightarrow[\text{GOTO}]{P_0} (\rho, P)$ wenn $\rho(x_r) \neq 0$
- $(\rho, M_i\colon \textbf{IF } x_r = 0 \textbf{ THEN GOTO } M_k) \xrightarrow[\text{GOTO}]{P_0} (\rho, \varepsilon)$ wenn $\rho(x_r) \neq 0$
- $(\rho, M_i\colon \textbf{HALT}; P) \xrightarrow[\text{GOTO}]{P_0} (\rho, M_i\colon \textbf{HALT})$
- $(\rho, \varepsilon) \xrightarrow[\text{GOTO}]{P_0} (\rho, \varepsilon)$

# Semantik von GOTO-Programmen

**Definition**

- $(\rho, M_i\colon \textbf{IF } x_r = 0 \textbf{ THEN GOTO } M_k; P) \xrightarrow[\text{GOTO}]{M_1\colon A_1;\ldots;M_n\colon A_n} (\rho, M_k\colon A_k; \ldots; M_n\colon A_n)$ wenn $1 \leq k \leq n$ und $\rho(x_r) = 0$
- $(\rho, M_i\colon \textbf{IF } x_r = 0 \textbf{ THEN GOTO } M_k) \xrightarrow[\text{GOTO}]{M_1\colon A_1;\ldots;M_n\colon A_n} (\rho, M_k\colon A_k; \ldots; M_n\colon A_n)$ wenn $1 \leq k \leq n$ und $\rho(x_r) = 0$
- $(\rho, M_i\colon \textbf{IF } x_r = 0 \textbf{ THEN GOTO } M_k; P) \xrightarrow[\text{GOTO}]{P_0} (\rho, P)$ wenn $\rho(x_r) \neq 0$
- $(\rho, M_i\colon \textbf{IF } x_r = 0 \textbf{ THEN GOTO } M_k) \xrightarrow[\text{GOTO}]{P_0} (\rho, \varepsilon)$ wenn $\rho(x_r) \neq 0$
- $(\rho, M_i\colon \textbf{HALT}; P) \xrightarrow[\text{GOTO}]{P_0} (\rho, M_i\colon \textbf{HALT})$
- $(\rho, \varepsilon) \xrightarrow[\text{GOTO}]{P_0} (\rho, \varepsilon)$

Wir schreiben $\xrightarrow[\text{GOTO}]{P_0}^{i}$ für $i$ Schritte und $\xrightarrow[\text{GOTO}]{P_0}^{*}$ für 0 oder beliebig viele Schritte.

# GOTO-Berechenbarkeit

**Definition**

Ein GOTO-Programme ist normalisiert, wenn es von der Form $M_1 : A_1; \ldots; M_n : A_n$ ist. Diese Form ist durch konsistentes Umbenennen der Markierungen herstellbar.

# GOTO-Berechenbarkeit

**Definition**

Ein GOTO-Programme ist normalisiert, wenn es von der Form $M_1 : A_1; \ldots; M_n : A_n$ ist. Diese Form ist durch konsistentes Umbenennen der Markierungen herstellbar.

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt GOTO-berechenbar, wenn es ein normalisiertes GOTO-Programm $P$ gibt, das die folgenden Bedingungen erfüllt:

- Für alle $n_1, \ldots, n_k \in \mathbb{N}$, sodass $f(n_1, \ldots, n_k)$ definiert ist, gilt $(\rho, P) \xrightarrow[\text{GOTO}]{P}{}^{*} (\rho', \textbf{HALT})$, wobei $\rho = \{x_1 \mapsto n_1, \ldots, x_k \mapsto n_k\}$ und $\rho'(x_0) = f(n_1, \ldots, n_k)$.
- Falls $f(n_1, \ldots, n_k)$ nicht definiert ist, stoppt das Programm $P$ nicht, d.h. für alle $i \in \mathbb{N}$ gibt es $\rho'$ und $P'$, sodass $(\rho, P) \xrightarrow[\text{GOTO}]{P}{}^{i} (\rho', P')$.

# WHILE-berechenbar impliziert GOTO-berechenbar

**Satz**

Jede WHILE-berechenbare Funktion ist auch GOTO-berechenbar.

# WHILE-berechenbar impliziert GOTO-berechenbar

**Satz**

Jede WHILE-berechenbare Funktion ist auch GOTO-berechenbar.

**Beweis** Wir simulieren ein WHILE-Programm durch ein GOTO-Programm.

- $x_j := x_k \pm c$ und $P_1; P_2$ bleiben ungeändert.

# WHILE-berechenbar impliziert GOTO-berechenbar

**Satz**

Jede WHILE-berechenbare Funktion ist auch GOTO-berechenbar.

**Beweis** Wir simulieren ein WHILE-Programm durch ein GOTO-Programm.

- $x_j := x_k \pm c$ und $P_1; P_2$ bleiben ungeändert.
- Ersetze **LOOP**-Schleifen durch **WHILE**-Schleifen.

# WHILE-berechenbar impliziert GOTO-berechenbar

**Satz**

Jede WHILE-berechenbare Funktion ist auch GOTO-berechenbar.

**Beweis** Wir simulieren ein WHILE-Programm durch ein GOTO-Programm.

- $x_j := x_k \pm c$ und $P_1; P_2$ bleiben ungeändert.
- Ersetze **LOOP**-Schleifen durch **WHILE**-Schleifen.
- Ersetze **WHILE** $x_i \neq 0$ **DO** $P$ **END** durch

$$
\begin{array}{ll}
M_j: & \textbf{IF } x_i = 0 \textbf{ THEN GOTO } M_k; \\
\vdots & P'; \\
M_{k-1}: & \textbf{GOTO } M_j; \\
M_k: &
\end{array}
$$

wobei $P'$ das GOTO-Programm zu $P$ ist.

# WHILE-berechenbar impliziert GOTO-berechenbar

**Satz**

Jede WHILE-berechenbare Funktion ist auch GOTO-berechenbar.

**Beweis** Wir simulieren ein WHILE-Programm durch ein GOTO-Programm.

- $x_j := x_k \pm c$ und $P_1; P_2$ bleiben ungeändert.
- Ersetze **LOOP**-Schleifen durch **WHILE**-Schleifen.
- Ersetze **WHILE** $x_i \neq 0$ **DO** $P$ **END** durch

  $M_j$: **IF** $x_i = 0$ **THEN GOTO** $M_k$;
  $\vdots$ $P'$;
  $M_{k-1}$: **GOTO** $M_j$;
  $M_k$:

  wobei $P'$ das GOTO-Programm zu $P$ ist.
- Füge **HALT** am Ende ein.

# WHILE-berechenbar impliziert GOTO-berechenbar

**Beweis** (Fortsetzung) Sei $P$ das WHILE-Programm und $P'$ das GOTO-Programm.

Dann gilt:

- Falls $(\rho, P) \xrightarrow[\text{WHILE}]{}^{*} (\rho', \varepsilon)$, dann $(\rho, P') \xrightarrow[\text{GOTO}]{P'}^{*} (\rho'', \textbf{HALT})$ mit $\rho''(x_0) = \rho'(x_0)$.
- Falls $(\rho, P') \xrightarrow[\text{GOTO}]{P'}^{*} (\rho', \textbf{HALT})$, dann $(\rho, P) \xrightarrow[\text{WHILE}]{}^{*} (\rho'', \varepsilon)$ mit $\rho''(x_0) = \rho'(x_0)$. □

# GOTO-berechenbar impliziert WHILE-berechenbar

**Satz**

Jede GOTO-berechenbare Funktion ist auch WHILE-berechenbar.

**Beweis** Sei $M_1\colon A_1; \ldots; M_n\colon A_n$ ein normalisiertes GOTO-Programm, sodass $x_M$ nicht darin vorkommt. Definiere folgendes passendes WHILE-Programm:

$x_M := 1;$
**WHILE** $x_M \neq 0$ **DO**
  **IF** $x_M = 1$ **THEN** $P_1$ **END**;
  $\vdots$
  **IF** $x_M = n$ **THEN** $P_n$ **END**;
**END**

wobei $P_i$ auf der nächsten Folie definiert ist.

# GOTO-berechenbar impliziert WHILE-berechenbar

**Beweis** (Fortsetzung) $P_i$ ist definiert wie folgt:

- $x_j := x_k \pm c; x_M := x_M + 1$,
  falls $A_i = x_j := x_k \pm c$
- $x_M := j$,
  falls $A_i =$ **GOTO** $M_j$
- **IF** $x_k = 0$ **THEN** $x_M := j$ **ELSE** $x_M := x_M + 1$ **END**,
  falls $A_i =$ **IF** $x_k = 0$ **THEN GOTO** $M_j$
- $x_M := 0$,
  falls $A_i =$ **HALT**.

# GOTO-berechenbar impliziert WHILE-berechenbar

**Beweis** (Fortsetzung) Sei $P$ das GOTO-Programm und $P'$ das WHILE-Programm. Dann gilt:

- Falls $(\rho, P) \xrightarrow[\text{GOTO}]{P}{}^{*} (\rho', \textbf{HALT})$, dann $(\rho, P') \xrightarrow[\text{WHILE}]{}{}^{*} (\rho'', \varepsilon)$ mit $\rho''(x_0) = \rho'(x_0)$.
- Falls $(\rho, P') \xrightarrow[\text{WHILE}]{}{}^{*} (\rho', \varepsilon)$, dann $(\rho, P) \xrightarrow[\text{GOTO}]{P}{}^{*} (\rho'', \textbf{HALT})$ mit $\rho''(x_0) = \rho'(x_0)$.

□

# GOTO-berechenbar ist äquivalent zu WHILE-berechenbar

**Theorem**

WHILE- und GOTO-Berechenbarkeit sind äquivalente Begriffe. D.h. jede WHILE-berechenbare Funktion ist auch GOTO-berechenbar und umgekehrt ist jede GOTO-berechenbare Funktion auch WHILE-berechenbar.

# GOTO-berechenbar ist äquivalent zu WHILE-berechenbar

**Theorem**

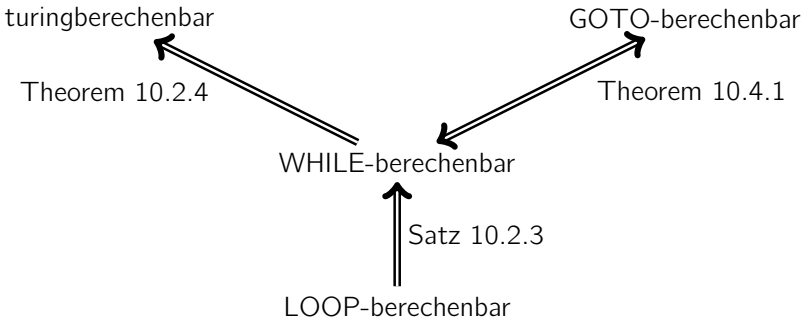
WHILE- und GOTO-Berechenbarkeit sind äquivalente Begriffe. D.h. jede WHILE-berechenbare Funktion ist auch GOTO-berechenbar und umgekehrt ist jede GOTO-berechenbare Funktion auch WHILE-berechenbar.

Die Übersetzung von GOTO-Programmen in WHILE-Programme zeigen auch:

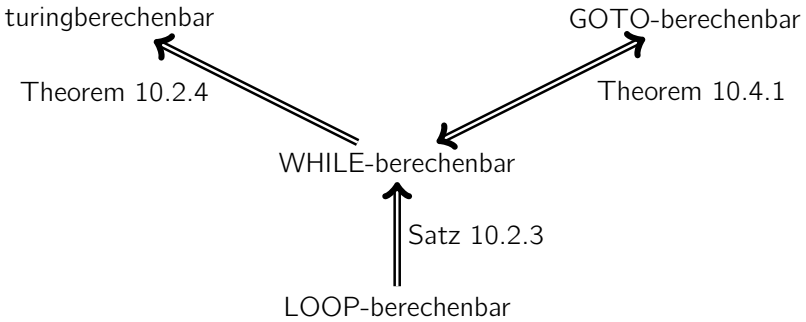
**Satz**

WHILE-Programme benötigen nur eine **WHILE**-Schleife.

# Fazit bisher



# Fazit bisher

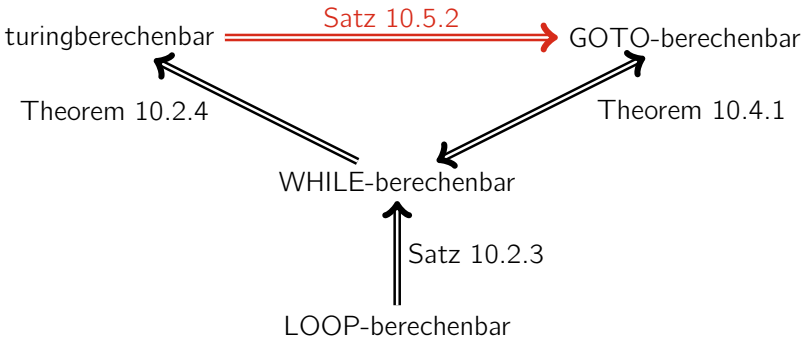



Unser Ziel:

**Theorem**

Turingberechenbarkeit, WHILE-Berechenbarkeit und GOTO-Berechenbarkeit sind äquivalente Begriffe.

# Fazit bisher




Unser Ziel:

**Theorem**

Turingberechenbarkeit, WHILE-Berechenbarkeit und GOTO-Berechenbarkeit sind äquivalente Begriffe.

# Turingberechenbar impliziert GOTO-berechenbar

**Satz**

Jede turingberechenbare Funktion ist auch GOTO-berechenbar.

**Beweis** Grundgedanke:

- Stelle TM-Konfiguration durch 3 Zahlen dar.
- Lege Zahlen in Programmvariablen ab.

# Darstellung von Wörtern als natürliche Zahlen

- Sei $\Gamma = \{a_1, \ldots, a_m\}$ das Bandalphabet.
- Zeichen $a_i$ wird mit Index $i$ identifiziert. (Beachte: $i \neq 0$.)
  D.h. $a_{i_1} \cdots a_{i_n} \in \Gamma^*$ ist äquivalent zur Folge $(i_1 \ldots i_n)$.
- Sei $b > m$.

# Darstellung von Wörtern als natürliche Zahlen

- Sei $\Gamma = \{a_1, \ldots, a_m\}$ das Bandalphabet.
- Zeichen $a_i$ wird mit Index $i$ identifiziert. (Beachte: $i \neq 0$.)
  D.h. $a_{i_1} \cdots a_{i_n} \in \Gamma^*$ ist äquivalent zur Folge $(i_1 \ldots i_n)$.
- Sei $b > m$.

Kodierung: Das Wort $a_{i_1} \ldots a_{i_n}$ lässt sich als Zahl $(i_1 \cdots i_n)_b$ im Stellenwertsystem zur Basis $b$ darstellen. Der Wert von $a_{i_1} \cdots a_{i_n}$ ist

$$(i_1 \cdots i_n)_b = \sum_{j=1}^{n} i_j \cdot b^{n-j}$$

# Darstellung von Wörtern als natürliche Zahlen

- Sei $\Gamma = \{a_1, \ldots, a_m\}$ das Bandalphabet.
- Zeichen $a_i$ wird mit Index $i$ identifiziert. (Beachte: $i \neq 0$.)
  D.h. $a_{i_1} \cdots a_{i_n} \in \Gamma^*$ ist äquivalent zur Folge $(i_1 \ldots i_n)$.
- Sei $b > m$.

Kodierung: Das Wort $a_{i_1} \ldots a_{i_n}$ lässt sich als Zahl $(i_1 \cdots i_n)_b$ im Stellenwertsystem zur Basis $b$ darstellen. Der Wert von $a_{i_1} \cdots a_{i_n}$ ist

$$(i_1 \cdots i_n)_b = \sum_{j=1}^{n} i_j \cdot b^{n-j}$$

Umkehrung: Berechne $(i_1 \cdots i_n)$ aus einer natürlichen Zahl.

- Teile wiederholt mit Rest durch Basis $b$ bis kein Rest mehr entsteht.
- Liste der Reste: $r_0, \ldots, r_m$.
- Ergibt $(r_m \cdots r_0)_b$ zur Basis $b$ und repräsentiert daher das Wort $a_{r_m} \cdots a_{r_0}$.

# Beispiel für die Darstellung von Wörtern als natürliche Zahlen

Sei $\Gamma = \{\square, 0, 1\}$, sodass $a_1 = \square$, $a_2 = 0$, $a_3 = 1$.

# Beispiel für die Darstellung von Wörtern als natürliche Zahlen

Sei $\Gamma = \{\square, 0, 1\}$, sodass $a_1 = \square$, $a_2 = 0$, $a_3 = 1$.

Betrachte $\square 110 \square$, d.h. $a_1 a_3 a_3 a_2 a_1$. Sei $b = 4$.

# Beispiel für die Darstellung von Wörtern als natürliche Zahlen

Sei $\Gamma = \{\square, 0, 1\}$, sodass $a_1 = \square$, $a_2 = 0$, $a_3 = 1$.

Betrachte $\square 110 \square$, d.h. $a_1 a_3 a_3 a_2 a_1$. Sei $b = 4$.

Identifiziere $\square 110 \square$ mit $(13321)$ und
$(13321)_4 = 1*4^4+3*4^3+3*4^2+2*4^1+1*4^0 = 1*256+3*64+3*16+2*4+1*1 = 505.$

# Beispiel für die Darstellung von Wörtern als natürliche Zahlen

Sei $\Gamma = \{\square, 0, 1\}$, sodass $a_1 = \square$, $a_2 = 0$, $a_3 = 1$.

Betrachte $\square 110 \square$, d.h. $a_1 a_3 a_3 a_2 a_1$. Sei $b = 4$.

Identifiziere $\square 110 \square$ mit $(13321)$ und
$(13321)_4 = 1*4^4+3*4^3+3*4^2+2*4^1+1*4^0 = 1*256+3*64+3*16+2*4+1*1 = 505$.

Umgekehrt ergibt

$$\begin{aligned} 505/4 &= 126 \quad \text{Rest } 1 \\ 126/4 &= 31 \quad \text{Rest } 2 \\ 31/4 &= 7 \quad \text{Rest } 3 \\ 7/4 &= 1 \quad \text{Rest } 3 \\ 1/4 &= 0 \quad \text{Rest } 1 \\ 0/4 &= 0 \quad \text{Rest } 0 \end{aligned}$$

Dies ergibt die Zahl $(13321)_4$, welche das Wort $a_1 a_3 a_3 a_2 a_1 = \square 110 \square$ darstellt.

# Kodierung von Turingmaschinen-Konfiguration im GOTO-Programm

Eine beliebige Konfiguration einer Turingmaschine mit $\Gamma = \{a_1, \ldots, a_m\}$ ist von der Form

$$a_{i_1} \cdots a_{i_n} z_k a_{j_1} \cdots a_{j_m}$$

# Kodierung von Turingmaschinen-Konfiguration im GOTO-Programm

Eine beliebige Konfiguration einer Turingmaschine mit $\Gamma = \{a_1, \ldots, a_m\}$ ist von der Form

$$a_{i_1} \cdots a_{i_n} z_k a_{j_1} \cdots a_{j_m}$$

Sie wird im GOTO-Programm durch 3 Variablen $x_z, x_p, x_s$ dargestellt:

$$\begin{aligned} x_z &= k \\ x_p &= (i_1 \cdots i_n)_b \\ x_s &= (j_m \cdots j_1)_b \end{aligned}$$

Beachte: $x_s$ verwendet umgekehrte Reihenfolge der Ziffern.

# GOTO-Programme für Übergänge

Für $\delta(z_k, a_{j_1}) = (z_\ell, a_{j'}, x)$ mit $x \in \{N, L, R\}$ sei $DELTA(k, j_1)$ das zugehörige GOTO-Programm. Wir gehen die verschiedenen Fälle durch.

# GOTO-Programme für Übergänge

Für $\delta(z_k, a_{j_1}) = (z_\ell, a_{j'}, x)$ mit $x \in \{N, L, R\}$ sei $DELTA(k, j_1)$ das zugehörige GOTO-Programm. Wir gehen die verschiedenen Fälle durch.

- Fall $\delta(z_k, a_{j_1}) = (z_\ell, a_{j'}, N)$:

  Der Übergang ist

  $$a_{i_1} \cdots a_{i_n} z_k a_{j_1} \cdots a_{j_m} \vdash a_{i_1} \cdots a_{i_n} z_\ell a_{j'} a_{j_2} \cdots a_{j_m}$$

  Daher:

  - $x_z$ muss auf den Wert $\ell$ aktualisiert werden.
  - $x_p$ bleibt unverändert.
  - $x_s$ muss von $(j_m \cdots j_1)_b$ zu $(j_m \cdots j_2 j')_b$ aktualisiert werden.

# GOTO-Programme für Übergänge

Für $\delta(z_k, a_{j_1}) = (z_\ell, a_{j'}, x)$ mit $x \in \{N, L, R\}$ sei $DELTA(k, j_1)$ das zugehörige GOTO-Programm. Wir gehen die verschiedenen Fälle durch.

- Fall $\delta(z_k, a_{j_1}) = (z_\ell, a_{j'}, N)$:

  Der Übergang ist

  $$a_{i_1} \cdots a_{i_n} z_k a_{j_1} \cdots a_{j_m} \vdash a_{i_1} \cdots a_{i_n} z_\ell a_{j'} a_{j_2} \cdots a_{j_m}$$

  Daher:

  - $x_z$ muss auf den Wert $\ell$ aktualisiert werden.
  - $x_p$ bleibt unverändert.
  - $x_s$ muss von $(j_m \cdots j_1)_b$ zu $(j_m \cdots j_2 j')_b$ aktualisiert werden.

  Das Programm $DELTA(k, j_1)$ sei dann

  $x_z := \ell;$
  $x_s := x_s \text{ div } b;$
  $x_s := b * x_s + j'$

# GOTO-Programme für Übergänge

- Fall $\delta(z_k, a_{j_1}) = (z_\ell, a_{j'}, R)$:

  Der Übergang ist

$$a_{i_1} \cdots a_{i_n} z_k a_{j_1} \cdots a_{j_m} \vdash a_{i_1} \cdots a_{i_n} a_{j'} z_\ell a_{j_2} \cdots a_{j_m}$$

  falls $m > 1$. Daher:

  - $x_z$ muss auf den Wert $\ell$ aktualisiert werden.
  - $x_p$ muss von $(i_1 \cdots i_n)_b$ zu $(i_1 \cdots i_n j')_b$ aktualisiert werden.
  - $x_s$ muss von $(j_m \cdots j_1)_b$ zu $(j_m \cdots j_2)_b$ aktualisiert werden.

# GOTO-Programme für Übergänge

- Fall $\delta(z_k, a_{j_1}) = (z_\ell, a_{j'}, R)$:

  Der Übergang ist

  $$a_{i_1} \cdots a_{i_n} z_k a_{j_1} \cdots a_{j_m} \vdash a_{i_1} \cdots a_{i_n} a_{j'} z_\ell a_{j_2} \cdots a_{j_m}$$

  falls $m > 1$. Daher:

  - $x_z$ muss auf den Wert $\ell$ aktualisiert werden.
  - $x_p$ muss von $(i_1 \cdots i_n)_b$ zu $(i_1 \cdots i_n j')_b$ aktualisiert werden.
  - $x_s$ muss von $(j_m \cdots j_1)_b$ zu $(j_m \cdots j_2)_b$ aktualisiert werden.

  Das Programm $DELTA(k, j_1)$ sei dann

  $x_z := \ell;$
  $x_p := b * x_p + j';$
  $x_s := x_s \text{ div } b$

# GOTO-Programme für Übergänge

Für den Fall $m = 1$ gilt

$$a_{i_1} \cdots a_{i_n} z_k a_{j_1} \vdash a_{i_1} \cdots a_{i_n} a_{j'} z_\ell \square$$

Das Programm $DELTA(k, j_1)$ sei dann

$x_z := \ell;$
$x_p := b * x_p + j';$ wobei $\square = a_r.$
$x_s := r$

# GOTO-Programme für Übergänge

- Fall $\delta(z_k, a_{j_1}) = (z_\ell, a_{j'}, L)$:

# GOTO-Programme für Übergänge

- Fall $\delta(z_k, a_{j_1}) = (z_\ell, a_{j'}, L)$:

  Der Übergang ist

  $$a_{i_1} \cdots a_{i_n} z_k a_{j_1} \cdots a_{j_m} \vdash a_{i_1} \cdots a_{i_{n-1}} z_\ell a_{i_n} a_{j'} a_{j_2} \cdots a_{j_m}$$

  falls $n > 0$. Daher:
  - $x_z$ muss auf den Wert $\ell$ aktualisiert werden.
  - $x_p$ muss von $(i_1 \cdots i_n)_b$ zu $(i_1 \cdots i_{n-1})_b$ aktualisiert werden.
  - $x_s$ muss von $(j_m \cdots j_1)_b$ zu $(j_m \cdots j_2 j' i_n)_b$ aktualisiert werden.

# GOTO-Programme für Übergänge

- Fall $\delta(z_k, a_{j_1}) = (z_\ell, a_{j'}, L)$:

  Der Übergang ist

  $$a_{i_1} \cdots a_{i_n} z_k a_{j_1} \cdots a_{j_m} \vdash a_{i_1} \cdots a_{i_{n-1}} z_\ell a_{i_n} a_{j'} a_{j_2} \cdots a_{j_m}$$

  falls $n > 0$. Daher:
  - $x_z$ muss auf den Wert $\ell$ aktualisiert werden.
  - $x_p$ muss von $(i_1 \cdots i_n)_b$ zu $(i_1 \cdots i_{n-1})_b$ aktualisiert werden.
  - $x_s$ muss von $(j_m \cdots j_1)_b$ zu $(j_m \cdots j_2 j' i_n)_b$ aktualisiert werden.

  Das Programm $DELTA(k, j_1)$ dazu:

  $x_z := \ell;$ // neuer Zustand $z_\ell$

  $x_s := x_s \text{ div } b;$ // Abschneiden der letzten Stelle: $(j_m \cdots j_1) \to (j_m \cdots j_2)$

  $x_s := b * x_s + j';$ // $j'$ als letzte Stelle hinzufügen: $(j_m \cdots j_2) \to (j_m \cdots j_2 j')$

  $x_s := b * x_s + (x_p \text{ mod } b);$ // $i_n = x_p \text{ mod } b$ am Ende hinzufügen:
  // $(j_m \cdots j_2 j') \to (j_m \cdots j_2 j' i_n)$

  $x_p := x_p \text{ div } b$ // Abschneiden der letzten Stelle: $(i_1 \cdots i_n) \to (i_1 \cdots i_{n-1})$

# GOTO-Programme für Übergänge

Für den Fall $n = 0$ gilt

$$z_k a_{j_1} \cdots a_{j_m} \vdash z_\ell \square a_{j_1} \cdots a_{j_m}$$

Setze zuerst $x_p := r$, wobei $\square = a_r$ und führe dann das Programm für den allgemeinen Fall aus.

# Turingberechenbar impliziert GOTO-berechenbar

Die Turingmaschine wird durch das folgende GOTO-Programm simuliert (mit Eingaben in $x_1, \ldots, x_k$ und Ausgabe in $x_0$):

$M_1 : P_1;$
$M_2 : P_2;$
$M_3 : P_3$

- $P_1$ erzeugt für die Eingabe in $x_1, \ldots, x_k$ die Binärzahldarstellung und dann die Darstellung der TM-Konfiguration in $x_p, x_z, x_s$.
- $P_2$: Siehe nächste Folie.
- $P_3$ verwendet die Darstellung der Endkonfiguration, um die Ausgabe in der Ausgabevariablen $x_0$ zu erzeugen.

# Turingberechenbar impliziert GOTO-berechenbar

Programmteil $P_2$ hat die Form:

$$
\begin{array}{ll}
M_2: & x_a := x_s \bmod b; \\
& \textbf{IF } x_z = 1 \textbf{ and } x_a = 1 \textbf{ THEN GOTO } M_{1,1}; \\
& \textbf{IF } x_z = 1 \textbf{ and } x_a = 2 \textbf{ THEN GOTO } M_{1,2}; \\
& \vdots \\
& \textbf{IF } x_z = q \textbf{ and } x_a = m \textbf{ THEN GOTO } M_{q,m}; \\
M_{1,1}: & P_{1,1}; \\
& \textbf{GOTO } M_2; \\
M_{1,2}: & P_{1,2}; \\
& \textbf{GOTO } M_2; \\
\vdots & \\
M_{q,m}: & P_{q,m} \\
& \textbf{GOTO } M_2
\end{array}
$$

Programm $P_{i,j}$ ist **GOTO** $M_3$, wenn $z_i \in E$ und $DELTA(i,j)$ sonst. □

# Fazit

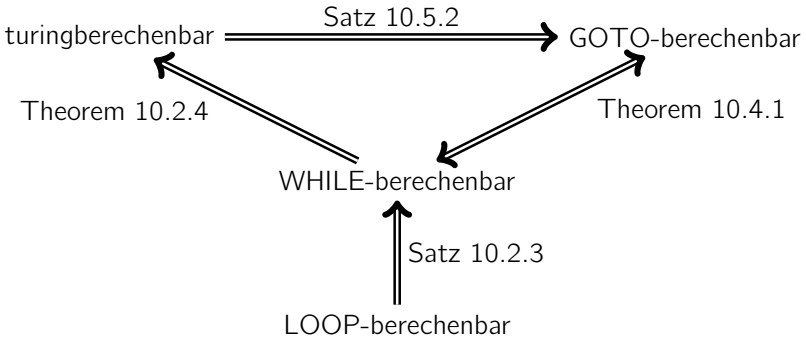
**Theorem**

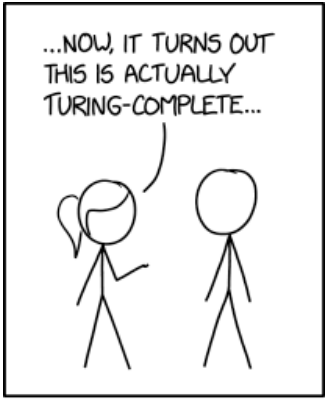
Turingberechenbarkeit, WHILE-Berechenbarkeit und GOTO-Berechenbarkeit sind äquivalente Begriffe.

# Fazit

**Theorem**

Turingberechenbarkeit, WHILE-Berechenbarkeit und GOTO-Berechenbarkeit sind äquivalente Begriffe.






xkcd.com/2556/