

9b

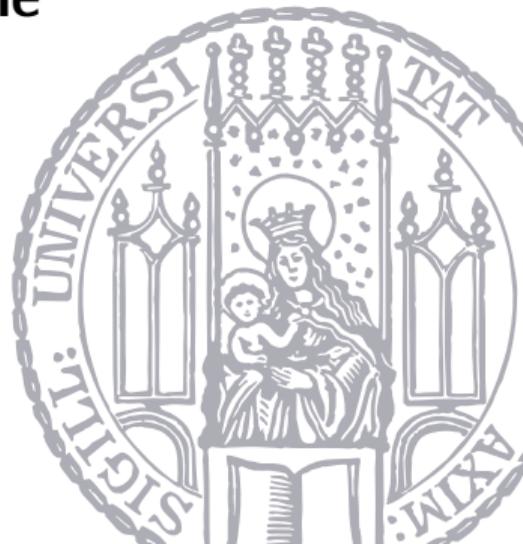
WHILE- und GOTO-Programme

Prof. Dr. Jasmin Blanchette

Lehr- und Forschungseinheit für
Theoretische Informatik und Theorembeweisen

Stand: 18. Juni 2024

Basierend auf Folien von PD Dr. David Sabel



Syntax von WHILE-Programmen

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

$$\begin{aligned} V &= \{Prg, Var, Id, Const\} \\ \Sigma &= \{\mathbf{WHILE}, \mathbf{LOOP}, \mathbf{DO}, \mathbf{END}, x, 0, \dots, 9, ,, :=, +, -\} \\ P &= \{Prg \rightarrow \mathbf{WHILE} \text{ } Var \neq 0 \mathbf{DO} \text{ } Prg \mathbf{END} \\ &\quad | \mathbf{LOOP} \text{ } Var \mathbf{DO} \text{ } Prg \mathbf{END} \\ &\quad | Prg; Prg \\ &\quad | Var := Var + Const \\ &\quad | Var := Var - Const, \\ Var &\rightarrow xId, \\ Const &\rightarrow Id, \\ Id &\rightarrow 0 | 1 | \dots | 9 | 1Id | 2Id | \dots | 9Id\} \end{aligned}$$

Beachte: WHILE-Programme [erweitern](#) LOOP-Programme um das **WHILE**-Konstrukt.

Definition

Die Berechnungsschritte $(\rho, P) \xrightarrow{\text{WHILE}} (\rho', P')$, wobei ρ, ρ' Variablenbelegungen und P, P' WHILE-Programme oder das leere Programm ε 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 $P'_1 \neq \varepsilon$
- ▶ $(\rho, \mathbf{LOOP} \ x_i \ \mathbf{DO} \ P \ \mathbf{END}) \xrightarrow{\text{WHILE}} (\rho, \underbrace{P; \dots; P}_{\rho(x_i)\text{-mal}})$

Definition

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

Semantik von WHILE-Programmen

Definition

- ▶ $(\rho, \mathbf{WHILE } x_i \neq 0 \mathbf{ DO } P \mathbf{ END}) \xrightarrow{\text{WHILE}} (\rho, \varepsilon)$ wenn $\rho(x_i) = 0$
- ▶ $(\rho, \mathbf{WHILE } x_i \neq 0 \mathbf{ DO } P \mathbf{ END}) \xrightarrow{\text{WHILE}} (\rho, P; \mathbf{WHILE } x_i \neq 0 \mathbf{ DO } P \mathbf{ END})$ 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.

Definition

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

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

durch

$x_m := x_j;$

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, Σ, P, Prg) erzeugt, wobei:

$$\begin{aligned} V &= \{Prg, Var, Id, Const\} \\ \Sigma &= \{\mathbf{GOTO}, \mathbf{HALT}, \mathbf{IF}, \mathbf{THEN}, x, 0, \dots, 9, ;;, :=, +, -\} \\ P &= \{Prg \rightarrow M_{Id}: \mathbf{GOTO} M_{Id} \\ &\quad | M_{Id}: \mathbf{IF} x_i = 0 \mathbf{THEN GOTO} M_{Id} \\ &\quad | M_{Id}: \mathbf{HALT} \\ &\quad | M_{Id}: Var := Var + Const \\ &\quad | M_{Id}: Var := Var - Const \\ &\quad | Prg; Prg, \\ Var &\rightarrow x_{Id}, \\ Const &\rightarrow Id, \\ Id &\rightarrow 0 | 1 | \dots | 9 | 1Id | 2Id | \dots 9Id\} \end{aligned}$$

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 ρ eine Variablenbelegung. Dann sind die Berechnungsschritte $\frac{P_0}{\text{GOTO}}$ durch folgende Regeln definiert:

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

Definition

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

Semantik von GOTO-Programmen

Definition

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

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

Definition

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

Definition

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

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

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

```
 $M_j$ :   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.

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 $(\rho, P') \xrightarrow{\text{GOTO}}^* (\rho'', \mathbf{HALT})$
mit $\rho''(x_0) = \rho'(x_0)$.
- ▶ Falls $(\rho, P') \xrightarrow{\text{GOTO}}^* (\rho', \mathbf{HALT})$, 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 $M_1: A_1; \dots; M_n: A_n$ 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:

- ▶ Falls $A_i = x_j := x_k \pm c$:
 $x_j := x_k \pm c; x_M := x_M + 1$
- ▶ Falls $A_i = \mathbf{GOTO} M_j$:
 $x_M := j$
- ▶ Falls $A_i = \mathbf{IF} x_k = 0 \mathbf{THEN GOTO} M_j$:
 $\mathbf{IF} x_k = 0 \mathbf{THEN} x_M := j \mathbf{ELSE} x_M := x_M + 1 \mathbf{END}$
- ▶ Falls $A_i = \mathbf{HALT}$:
 $x_M := 0$.

GOTO-berechenbar impliziert WHILE-berechenbar

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

Dann gilt:

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

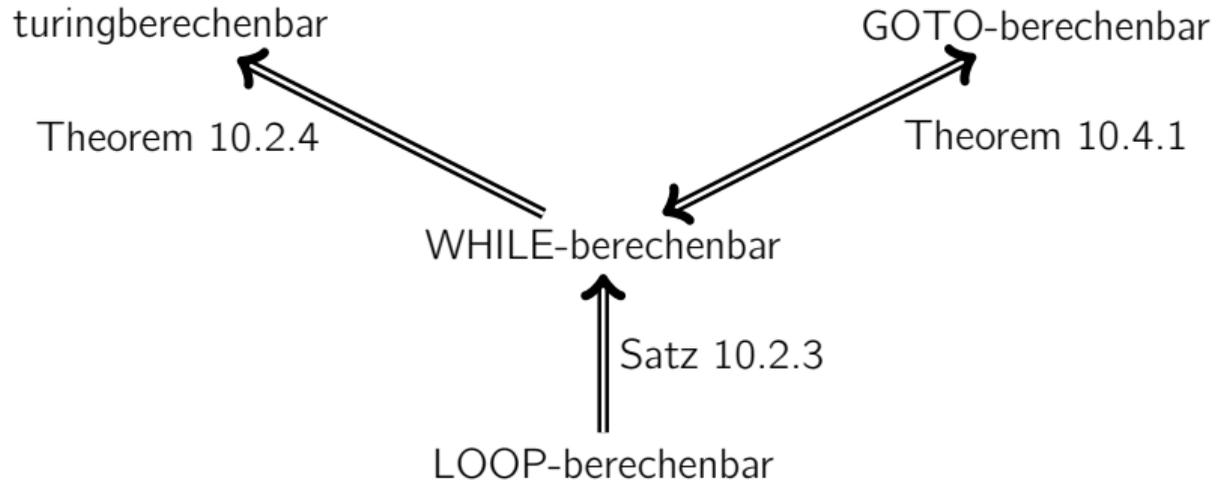
WHILE- und GOTO-Berechenbarkeit sind äquivalente Begriffe. D.h. jede WHILE-berechenbare Funktion ist auch GOTO-berechenbar und umgekehrt ist jede GOTO-berechenbare Funktion auch WHILE-berechenbar.

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

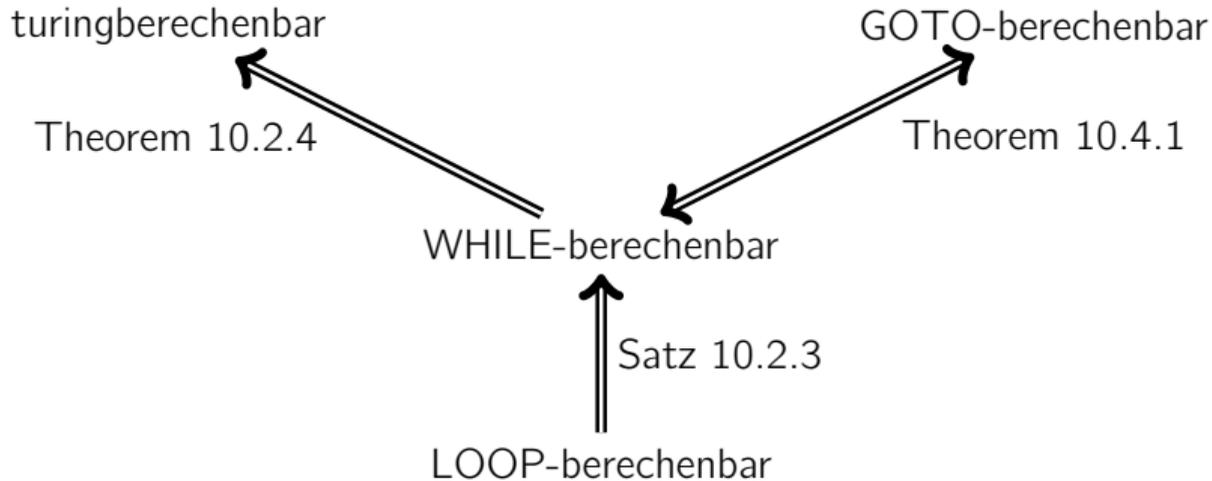
Satz

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

Fazit bisher



Fazit bisher

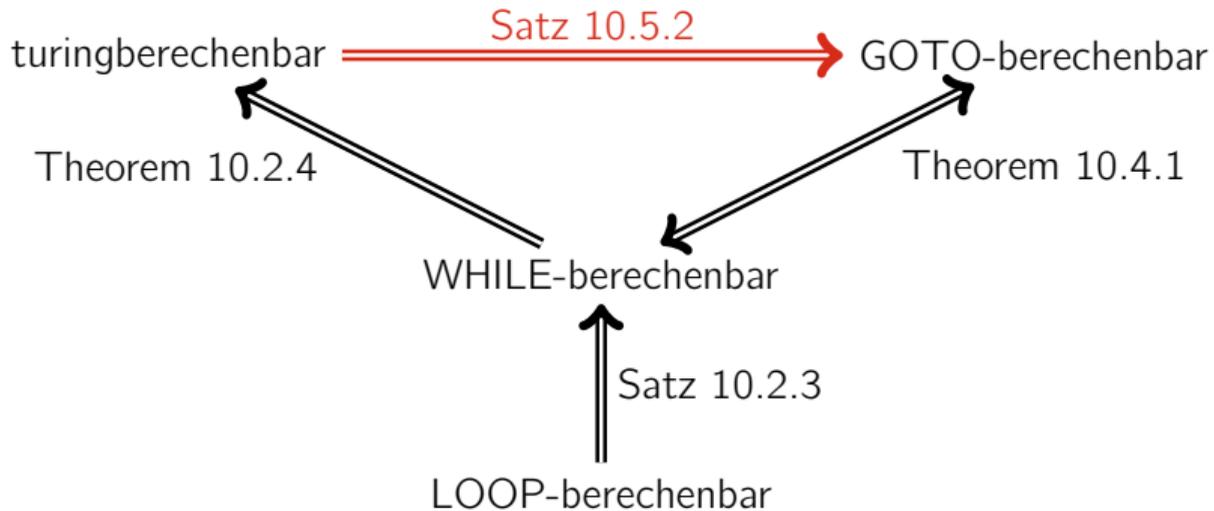


Unser Ziel:

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 $\Gamma = \{a_1, \dots, a_m\}$ 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 $(i_1 \dots i_n)$.
- ▶ Sei $b > m$.

Darstellung von Wörtern als natürliche Zahlen

- ▶ Sei $\Gamma = \{a_1, \dots, a_m\}$ 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 $(i_1 \dots i_n)$.
- ▶ Sei $b > m$.

Kodierung: Das Wort $a_{i_1} \dots a_{i_n}$ 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 $\Gamma = \{a_1, \dots, a_m\}$ 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 $(i_1 \dots i_n)$.
- ▶ Sei $b > m$.

Kodierung: Das Wort $a_{i_1} \dots a_{i_n}$ 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, \dots, 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

$$(13321)_4 = 1 \cdot 4^4 + 3 \cdot 4^3 + 3 \cdot 4^2 + 2 \cdot 4^1 + 1 \cdot 4^0 = 1 \cdot 256 + 3 \cdot 64 + 3 \cdot 16 + 2 \cdot 4 + 1 \cdot 1 = 505.$$

Umgekehrt ergibt

$$505/4 = 126 \quad \text{Rest } 1$$

$$126/4 = 31 \quad \text{Rest } 2$$

$$31/4 = 7 \quad \text{Rest } 3$$

$$7/4 = 1 \quad \text{Rest } 3$$

$$1/4 = 0 \quad \text{Rest } 1$$

$$0/4 = 0 \quad \text{Rest } 0$$

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 $\Gamma = \{a_1, \dots, a_m\}$ 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 $\Gamma = \{a_1, \dots, a_m\}$ 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:

$$x_z = k$$

$$x_p = (i_1 \cdots i_n)_b$$

$$x_s = (j_m \cdots j_1)_b$$

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 ℓ 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 ℓ aktualisiert werden.
- ▶ x_p bleibt unverändert.
- ▶ x_s muss von $(j_m \cdots j_1)_b$ zu $(j_m \cdots j_2 j')$ 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 ℓ 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 ℓ 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

$$x_z := \ell;$$

$$x_p := b * x_p + j'; \quad \text{wobei } \square = a_r.$$

$$x_s := r$$

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 ℓ 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 ℓ 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 := ℓ; // neuer Zustand z_ℓ
x_s := x_s div b; // Abschneiden der letzten Stelle: (j_m ⋯ j_1) → (j_m ⋯ j_2)
x_s := b * x_s + j'; // j' als letzte Stelle hinzufügen: (j_m ⋯ j_2) → (j_m ⋯ j_2 j')
x_s := b * x_s + (x_p mod b); // i_n = x_p mod b am Ende hinzufügen:
// (j_m ⋯ j_2 j') → (j_m ⋯ j_2 j' i_n)
x_p := x_p div b // Abschneiden der letzten Stelle: (i_1 ⋯ i_n) → (i_1 ⋯ 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, \dots, 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, \dots, 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:

```
 $M_2$  :    $x_a := x_s \bmod b$ ;  
         IF  $x_z = 1$  and  $x_a = 1$  THEN GOTO  $M_{1,1}$ ;  
         IF  $x_z = 1$  and  $x_a = 2$  THEN GOTO  $M_{1,2}$ ;  
          $\vdots$   
         IF  $x_z = q$  and  $x_a = m$  THEN GOTO  $M_{q,m}$ ;  
 $M_{1,1}$  :  $P_{1,1}$ ;  
         GOTO  $M_2$ ;  
 $M_{1,2}$  :  $P_{1,2}$ ;  
         GOTO  $M_2$ ;  
          $\vdots$   
 $M_{q,m}$  :  $P_{q,m}$   
         GOTO  $M_2$ 
```

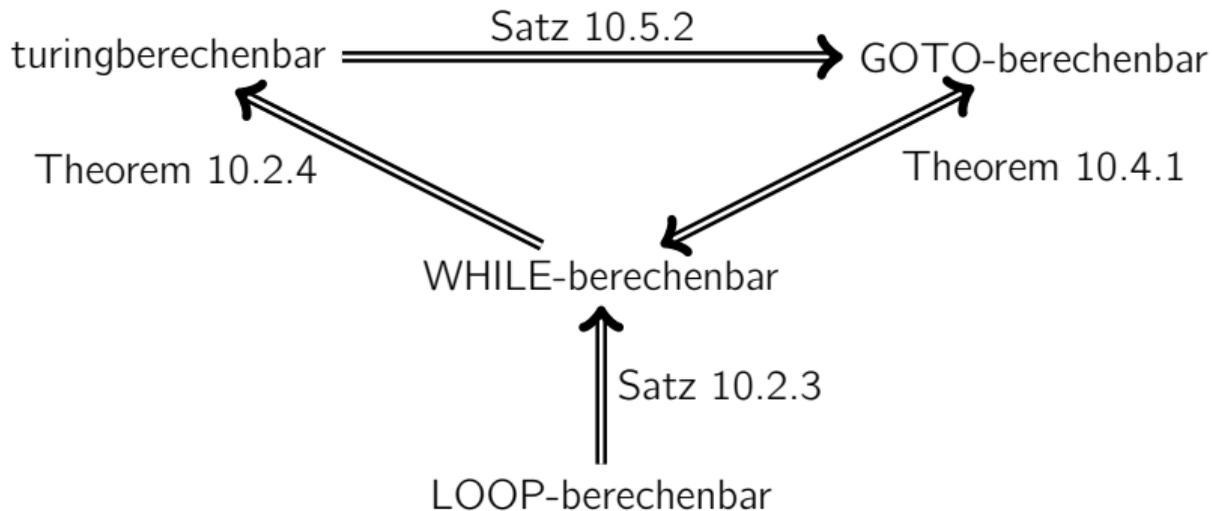
Programm $P_{i,j}$ ist **GOTO** M_3 , wenn $z_i \in E$ und $DELTA(i,j)$ sonst. □

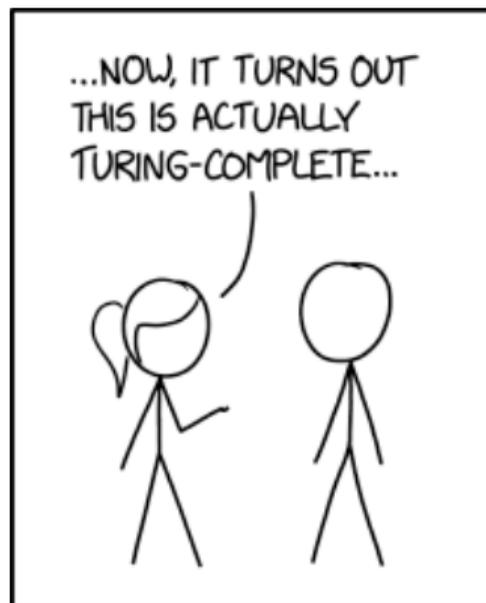
Theorem

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

Theorem

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





THIS PHRASE EITHER MEANS
SOMEONE SPENT SIX MONTHS
GETTING A DISHWASHER TO
PLAY MARIO OR YOU'RE UNDER
ATTACK BY A NATION-STATE.

xkcd.com/2556/