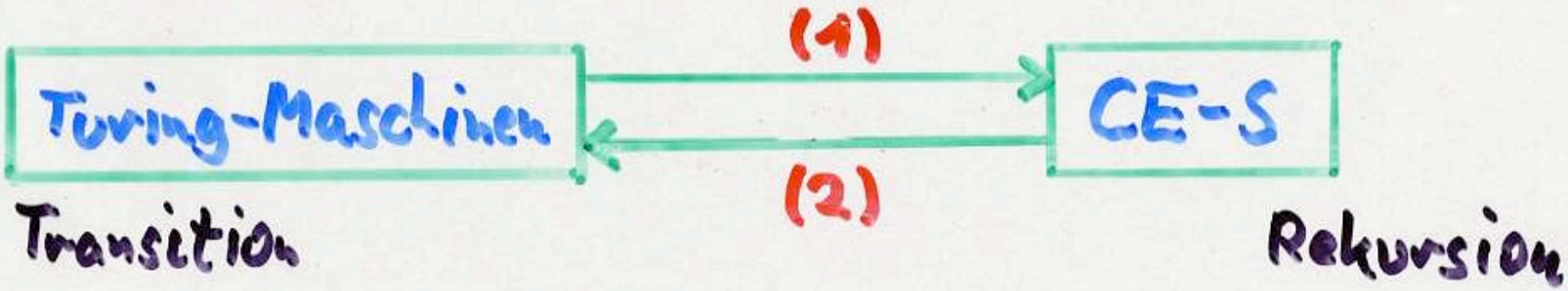
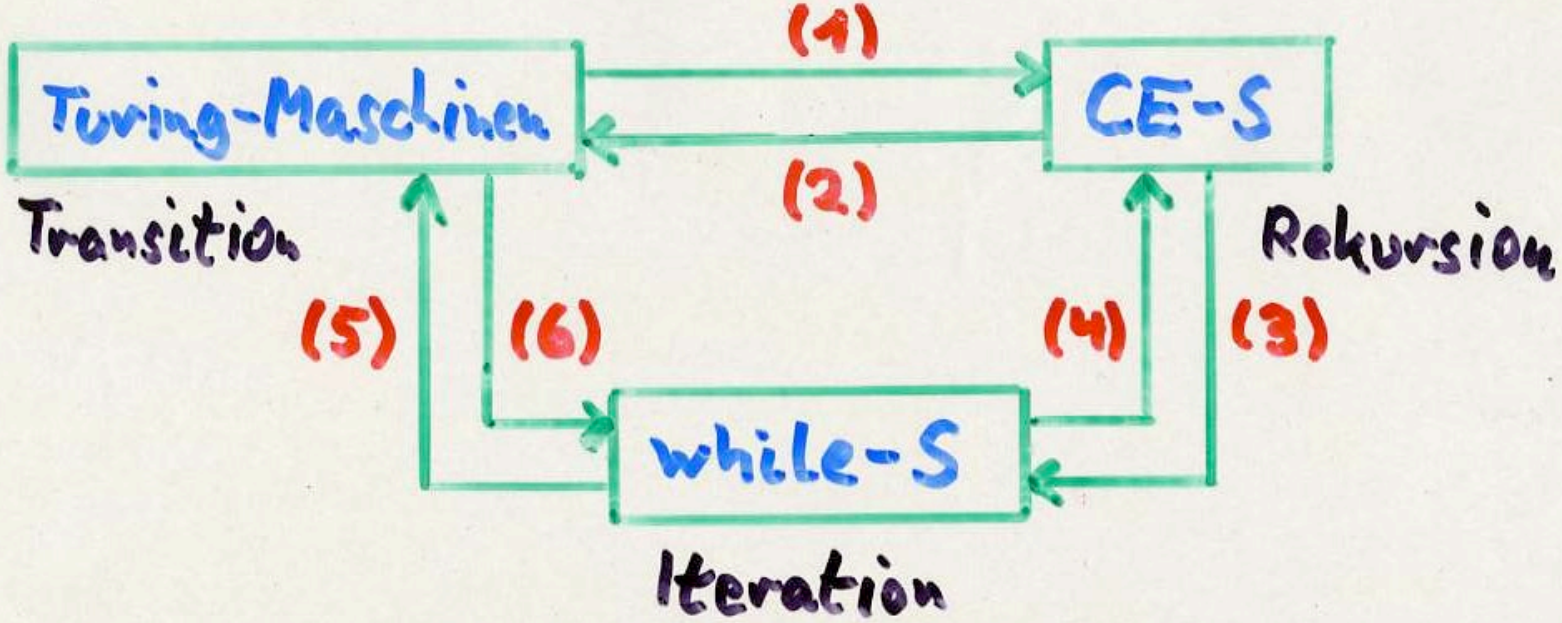
Vergleich von Berechenbarkeitsmodellen



(1) Zustände als rekursive Funktionen auf dem Band
(2) CE-S-Vorwärtsinterpreter als Turing-Maschine

Vergleich von Berechenbarkeitsmodellen



(1) Zustände als rekursive Funktionen auf dem Band

(2) CE-S-Vorwärtsinterpreter als Turing-Maschine

while-S

- kleine imperative Programmiersprache mit Zuweisung, Reihung, Wiederholung & Fallunterscheidung
- Datentypen wie CE-S
  - BOOL mit $T, F, \neg, \wedge, \vee, \ldots$
  - $\mathbb{N}$ mit $0, 1, 2, \ldots$ & $+, -, \cdot, \ldots$
  - A (Alphabet, Aufzählungstyp) mit $a, b, c, \ldots$ & $\equiv, \leq$
  - A^* mit λ, Linksaddition, Konkatenation, length, count, is-in, $\equiv$, ...
    head & tail

Syntax von while-S

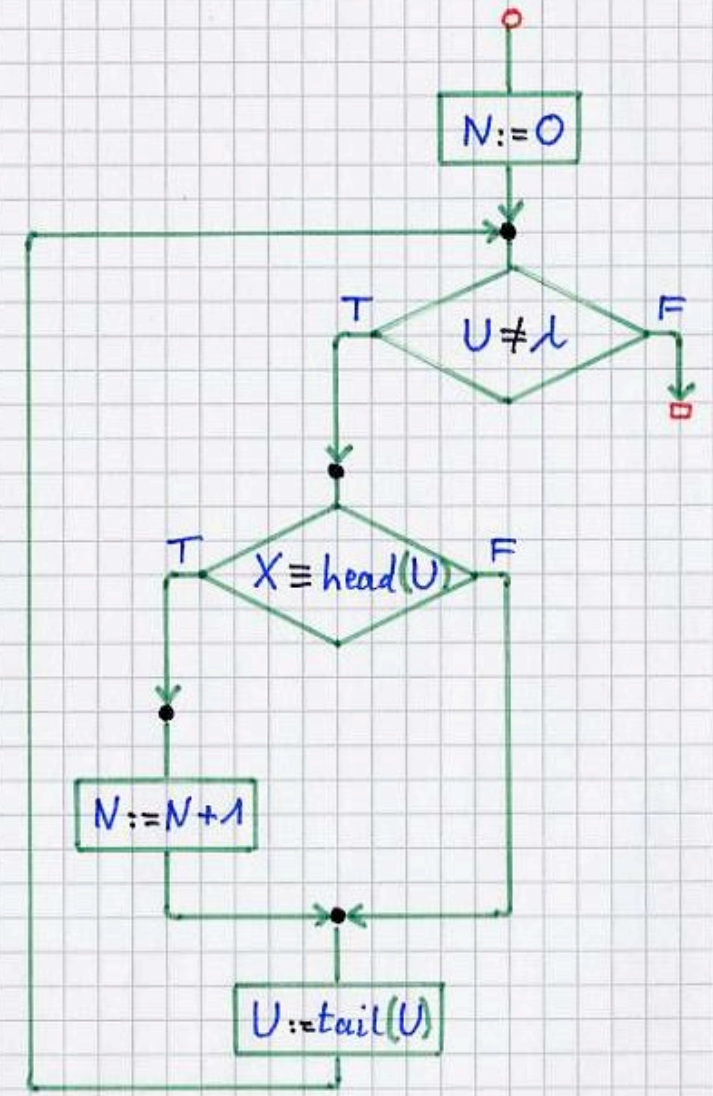
- Programmform : name
  vars: $X_1 \in D_1, \ldots, X_k \in D_k$
  <compound>

- mit folgenden Syntaxregeln (BNF):

  <compound> ::= begin <stmtList> end
  <stmtList> ::= <stmt> | <stmt> ; <stmtList>
  <stmt> ::= <assign> | <compound> | <loop> | <case>
  <assign> ::= $X_i := t$ { Term des Typs D_i }
  <loop> ::= while b do <stmt> { BOOL-Term }
  <case> ::= if b then <stmt> [else <stmt>]
  { optional }

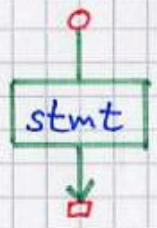
z. B. Zeichenzählen

```
count
  vars: X∈A, U∈A*; N∈ℕ
  begin
    N := 0;
    while U ≠ λ do
      begin
        if X ≡ head(U) then N := N+1;
        U := tail(U)
      end
  end
```



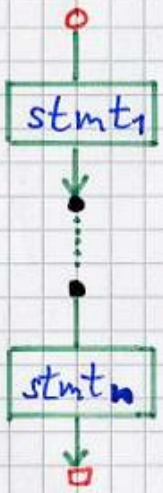
graphische Darstellung von while-S-Programmen

- jede Anweisung (<stmt>) wird graphisch dargestellt durch

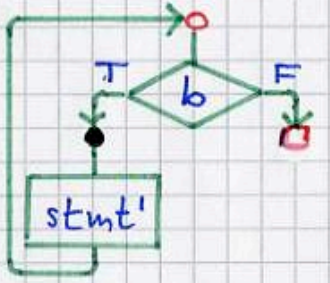


- dabei können folgende Ersetzungen vorgenommen werden:

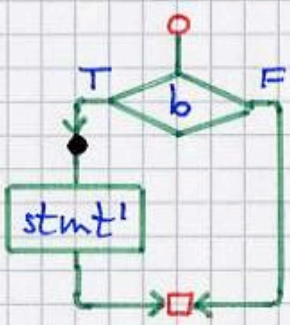
falls $stmt = \text{begin } stmt_1; \ \ldots\ ; stmt_n \text{ end}$



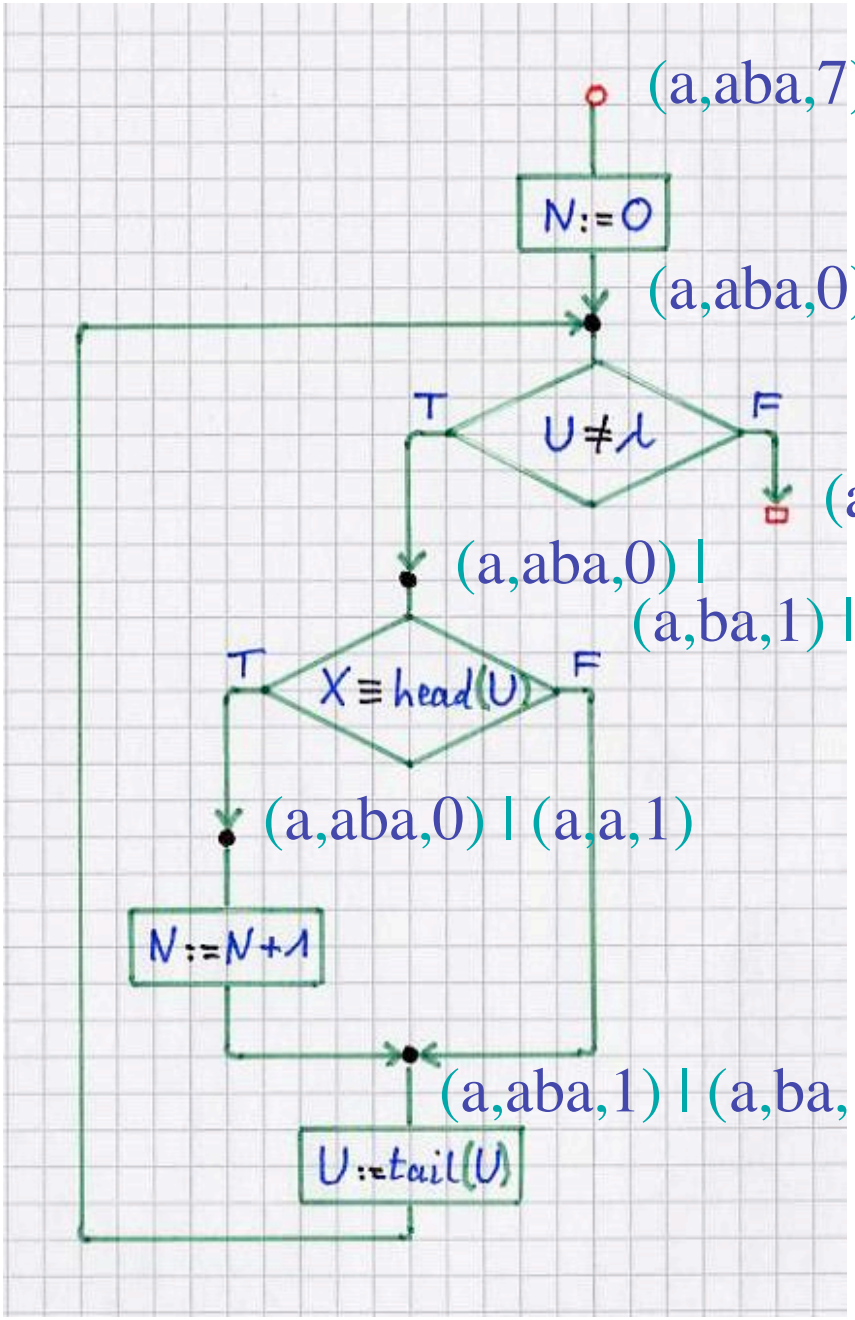
falls $stmt = \text{while } b \text{ do } stmt'$



falls $stmt =$ if b then $stmt'$



- genau ein Eingang ○ und genau ein Ausgang □
- abgesehen von Ausgang, dem nichts folgt, liegt hinter jedem Knoten genau eine Instruktion (d.h. Zuweisung oder Test)



(a,aba,7)

(a,aba,0) | (a,ba,1) | (a,a,1) | (a,λ,2)

(a,λ,2)

(a,aba,0) |
(a,ba,1) | (a,a,1)

(a,aba,0) | (a,a,1)

(a,aba,1) | (a,ba,1) | (a,a,2)

Zustandstransformationssemantik

- sei P ein while-S-Programm in graphischer Darstellung
- Berechnungszustand: $a = (x_1, \ldots, x_k) \in D_1 \times \cdots \times D_k$
  (aktueller Wert für jede Variable)
- Berechnung: $a_0 A_1 a_1 A_2 a_2 \cdots a_{i-1} A_i a_i A_{i+1} a_{i+1} \cdots$
  - a_i jeweils Berechnungszustand an einem Knoten
  - A_{i+1} jeweils folgende Instruktion (soweit vorhanden)
  - a_0 Anfangszustand am Eingang, frei wählbar (I.A.)
  - sei Berechnung bis a_i bereits konstruiert (I.V.)
  - 1. Fall: a_i am Ausgang; dann terminiert Berechnung
    2. Fall: a_i nicht am Ausgang und A_{i+1} Test b;
    dann $a_{i+1} = a_i$ am T-Ausgang, falls $a_i[b] = T$,
    und am F-Ausgang sonst
    3. Fall: a_i nicht am Ausgang und A_{i+1} Zuweisung $X_j := t$,
    dann $a_{i+1} = (x_1, \ldots, x_{j-1}, a_i[t], x_{j+1}, \ldots, x_k)$ am
    Ausgang von A_{i+1} ($a_i = (x_1, \ldots, x_k)$)

Eigenschaften von count

(1) jede Berechnung terminiert

(2) sei (x_m, u_m, n_m) Endzustand zum Anfangszustand (x_0, u_0, n_0);

dann gilt: $n_m = count(x_0, u_0)$ (mit count aus CE-S)

▷ betrachte Berechnung $\underset{\|}{a_0}\ \underset{\|}{A_1}\ \underset{\|}{a_1}\ \underset{\|}{A_2}\ \underset{\|}{a_2} \dots a_i A_{i+1} a_{i+1} \dots$

$(x_0, u_0, n_0)(N:=0)(x_0, u_0, 0)(U \neq \lambda)(x_0, u_0, 0)$

und $loop(i) = \#\{ j \mid j \leq i+1, A_j = (U \neq \lambda) \}$ (Eintritte in Schleife bis i)

dann gilt für alle i mit $A_{i+1} = (U \neq \lambda)$: $loop(i) \geq (i+3)/4$ &

$x_i = x_0$, $u_0 = v_i u_i$ mit $length(v_i) = loop(i) - 1$, $n_i = count(x_0, v_i)$

- Letzteres alles vorausgesetzt folgt:

  $length(u_0) \geq length(v_i) = loop(i) - 1 \geq ((i+3)/4) - 1 \geq length(u_0)$

  und damit $length(u_0) = length(v_i)$ falls $i \geq 4\,length(u_0) + 1$

- d.h. es ex. $a_{m-1} = (x_0, \ell, n_{m-1})$ mit $A_m = (U \neq L)$, so dass $a_m = a_{m-1}$ Endzustand ist mit

  $n_m = n_{m-1} = count(x_0, v_{m-1}) = count(x_0, u_0)$

- d.h. (1) und (2) gelten

- Rest mit vollständiger Induktion über $loop(i)$:

  I.A.: $loop(1) = \#\{2\} = 1$ und es gilt:

  $x_n = x_0$, $u_0 = \ell u_0 = \ell u_n = v_n u_n$, $n_n = 0 = count(x_0, v_n)$

  mit $length(v_n) = length(\ell) = 0 = 1 - 1 = loop(1) - 1$

I.S.: $loop(i) = l+1$ mit $A_{i+1} \equiv (U \neq l)$

betrachte Berechnung von a_{i-4} bis a_{i+1}

a_{i-4} A_{i-3} a_{i-3} A_{i-2} a_{i-2} A_{i-1} a_{i-1} A_i a_i A_{i+1} a_{i+1}

I.V. $\|$ $(U \neq l)$ $\|$ $(x \equiv head(U))$ $\|$ $(N := N+1)$ $\|$ $(U := tail(U))$

1. Fall

(x_0, u_{i-4}, n_{i-4})

mit

$u_0 = v_{i-4}\, u_{i-4}$

$length(v_{i-4}) \geq l-1$

$n_{i-4} = count(x_0, v_{i-4})$

$loop(i-4) = l$

$\geq (i-1)/4$

I.S.: $loop(i) = l+1$ mit $A_{i+1} \equiv (U \neq \lambda)$

betrachte Berechnung von a_{i-4} bis a_{i+1}

a_{i-4} A_{i-3} a_{i-3} A_{i-2} a_{i-2} A_{i-1} a_{i-1} A_i a_i A_{i+1} a_{i+1}

I.V. $\|$ $(U \neq \lambda)$ $\|$ $(X \equiv head(U))$ $\|$ $(N := N+1)$ $\|$ $(U := tail(U))$

1. Fall

(x_0, u_{i-4}, n_{i-4}) a_{i-4} a_{i-4} $(x_0, u_{i-4}, n_{i-4}+1)$

mit

$u_0 = v_{i-4}\, u_{i-4}$

$length(v_{i-4}) \geq l-1$

$n_{i-4} = count(x_0, v_{i-4})$

$loop(i-4) = l \geq (i-1)/4$

$x_0 \equiv head(u_{i-4})$

1. Fall

und damit

$x_i = x_0$

$u_0 = v_{i-4}\, u_{i-4} =$

$= v_{i-4}\, head(u_{i-4})\, tail(u_{i-4})$

$= v_i\, u_i$

$length(v_i) = l$

$n_i = n_{i-4} + 1 = count(x_0, v_i)$

$loop(i) = l+1 \geq (i+3)/4$

2. Fall analog