# Übersetzung und Analyse von Programmen

Vorlesung im Wintersemester 2025/26

Prof. Dr. Stephan Diehl
Informatik, Universität Trier

# Zeitplan bis zum Jahresende

| | | |
|---|---|---|
| 11.12.2025 | Abstrakte Interpretation (von TRIPLA) | |
| 16.12.2025 | Weihnachtsdings + Vortrag Wölwer | Abgabe Übungsblatt „Übersetzung von Java“ |
| 18.12.2025 | | Abgabe Projekt III |
| |  | |
| | | |
| 08.01.2026 | Datenflussanalyse | Übungsblatt „Datenflussanalyse“ |
| 13.01.2026 | Datenflussanalyse von TRIPLA | |
| 15.01.2026 | Projekt IV (Kontrollflussanalyse) | Abgabe Übungsblatt „Datenflussanalyse“ |
| 20.01.2026 | Code Clone Detection | |
| 22.01.2026 | Reverse Engineering, Erkennung von Entwurfsmustern | Abgabe Projekt IV |
| 27.01.2026 | Gastvortrag: „Programm Analysis and Transformation from a Practitioner‘s Point of View“ (Jonas Eckstein, itestra) | Projekt V (Datenflussanalyse) |

# Datenflussanalyse für TRIPLA

Bildquelle: mit KI erzeugt (Bing, DALL-E)

# Formale Beschreibung der **Kontrollflussanalyse**

Gegeben sei die Grammatik G:

```
E → let D in E
  | ID
  | ID ( A )
  | E AOP E
  | ( E )
  | CONST
  | ID = E
  | E ; E
  | if B then E else E
  | while B do { E }
A → E | A , E
D → ID ( V ) { E }
  | D D
V → ID | V , V
B → E | E RELOP E
```

| | |
|---|---|
| ID: | Bezeichner (fangen mit Buchstaben an und bestehen aus Buchstaben und Zahlen) |
| CONST: | ganze Zahlen |
| AOP: | arithm. Operatoren (+, -, *, /) |
| RELPOP: | Vergleichsoperatoren (==, !=, <, >) |

- Wir definieren die Analyse durch Induktion über die Struktur von Ausdrücken:

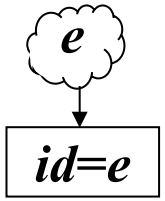
$$\text{cfg} : L_{G_{TRIPLA}}(\text{E}) \times \Psi \longrightarrow \Gamma$$

- In der Knotenumgebung werden für Funktionen deren **in** und **out** Knoten gespeichert:

$$\Psi : Identifier \longrightarrow V \times V$$

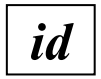
# **Intra**prozedurale Kontrollflussanalyse

*id=e*

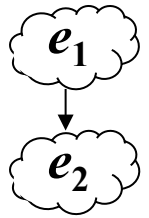



*const*

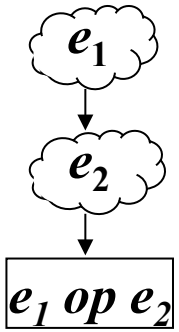

*id*



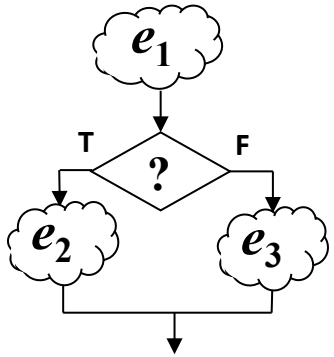

$e_1$ ; $e_2$



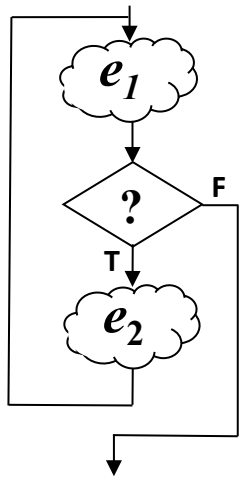

$e_1$ *op* $e_2$




if $e_1$ then $e_2$ else $e_2$




while $e_1$ do { $e_2$ }

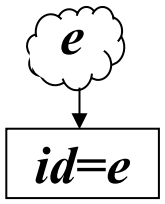


# Formale Definition eines Kontrollflussgraphen

A CFG is a tuple $(V, E, in, out)$ where

- $V$ is a set of nodes,
- $E \subseteq V \times L \times V$ a set of edges,
- $L = \{\epsilon, \mathtt{T}, \mathtt{F}\}$ a set of labels,
- and $in, out \in V$ are start and end nodes.

# CFG für einfache Ausdrücke: $\mathsf{cfg}(w) = (V, E, in, out)$, where



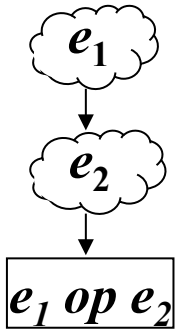

$$\text{if } w = id{=}e \text{ then } \begin{cases} V = V' \cup \{w\} \\ E = E' \cup \{(out', \epsilon, w)\} \\ in = in' \\ out = w \end{cases}$$

where $(V', E', in', out') = \mathsf{cfg}(e, \psi)$.

---


$$\text{if } w = id \text{ or } w = const \text{ then } \begin{cases} V = \{w\} \\ E = \emptyset \\ in = w \\ out = w \end{cases}$$

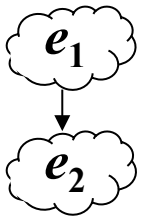
---




$$\text{if } w = e_1 \;\; op \;\; e_2 \text{ then } \begin{cases} V = V_1 \cup V_2 \cup \{w\} \\ E = E_1 \cup E_2 \cup \{(out_1, \epsilon, in_2), (out_2, \epsilon, w)\} \\ in = in_1 \\ out = w \end{cases}$$

where $(V_1, E_1, in_1, out_1) = \mathsf{cfg}(e_1, \psi)$ and $(V_2, E_2, in_2, out_2) = \mathsf{cfg}(e_2, \psi)$.

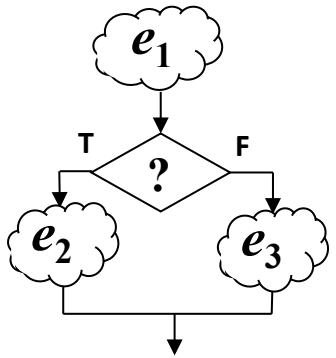
---




$$\text{if } w = e_1 \;\; ; \;\; e_2 \text{ then } \begin{cases} V = V_1 \cup V_2 \\ E = E_1 \cup E_2 \cup \{(out_1, \epsilon, in_2)\} \\ in = in_1 \\ out = out_2 \end{cases}$$

where $(V_1, E_1, in_1, out_1) = \mathsf{cfg}(e_1, \psi)$ and $(V_2, E_2, in_2, out_2) = \mathsf{cfg}(e_2, \psi)$.

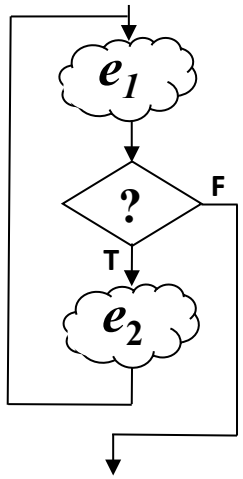
# CFGs für IF-THEN-ELSE und WHILE




if $w = \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3$ then

$$\begin{cases} V = V_1 \cup V_2 \cup V_3 \cup \{diamond, glue\} \\ E = \ E_1 \cup E_2 \cup E_3 \\ \qquad \cup \{(out_1, \epsilon, diamond), (diamond, T, in_2), (diamond, F, in_3), \\ \qquad\quad (out_2, \epsilon, glue), (out_3, \epsilon, glue)\} \\ in = in_1 \\ out = glue \end{cases}$$

where $(V_i, E_i, in_i, out_i) = \mathsf{cfg}(e_i, \psi)$ and $diamond$ and $glue$ are two new nodes.

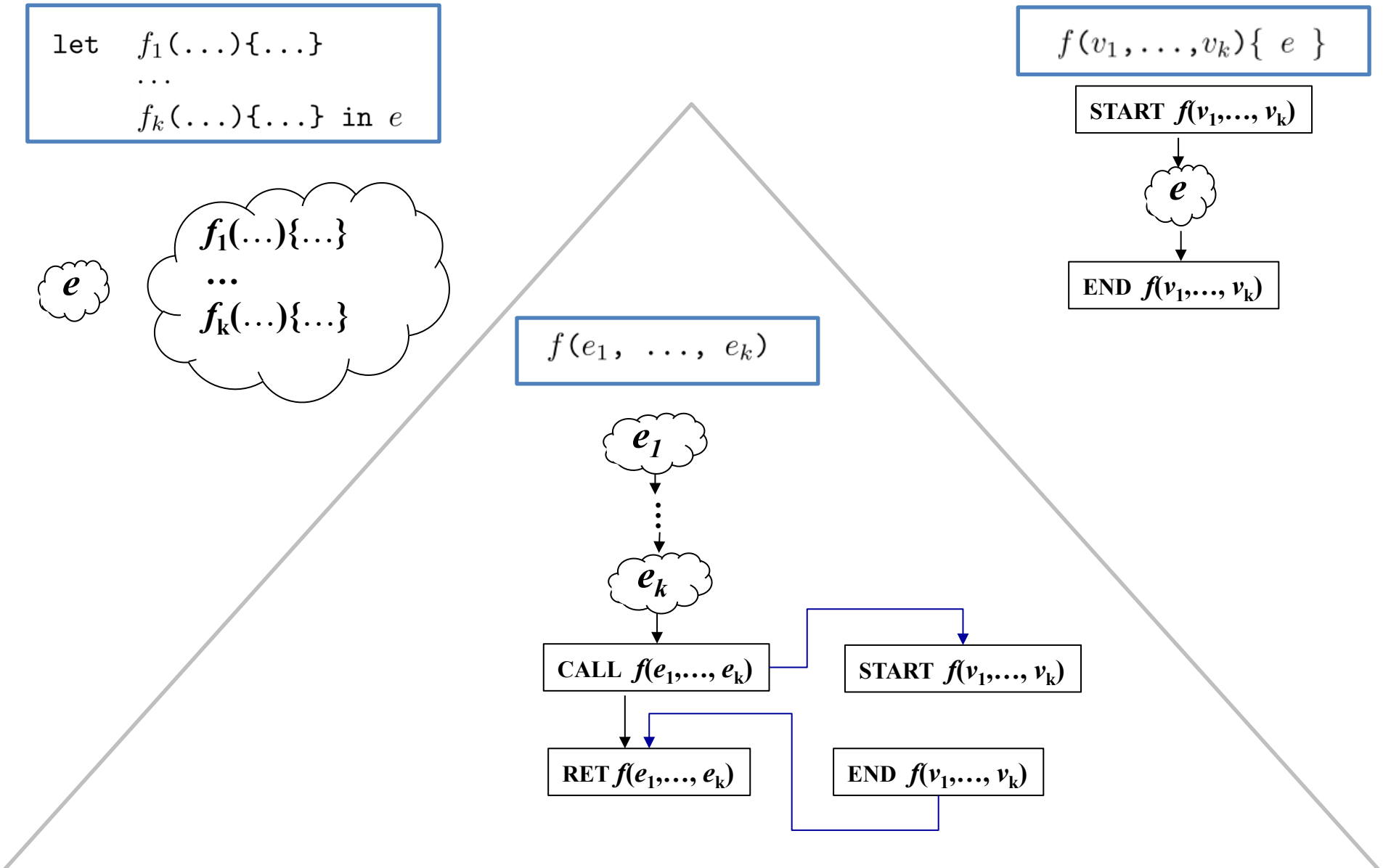



if $w = \texttt{while}\ e_1\ \texttt{do}\ \{\ e_2\ \}^p$ then

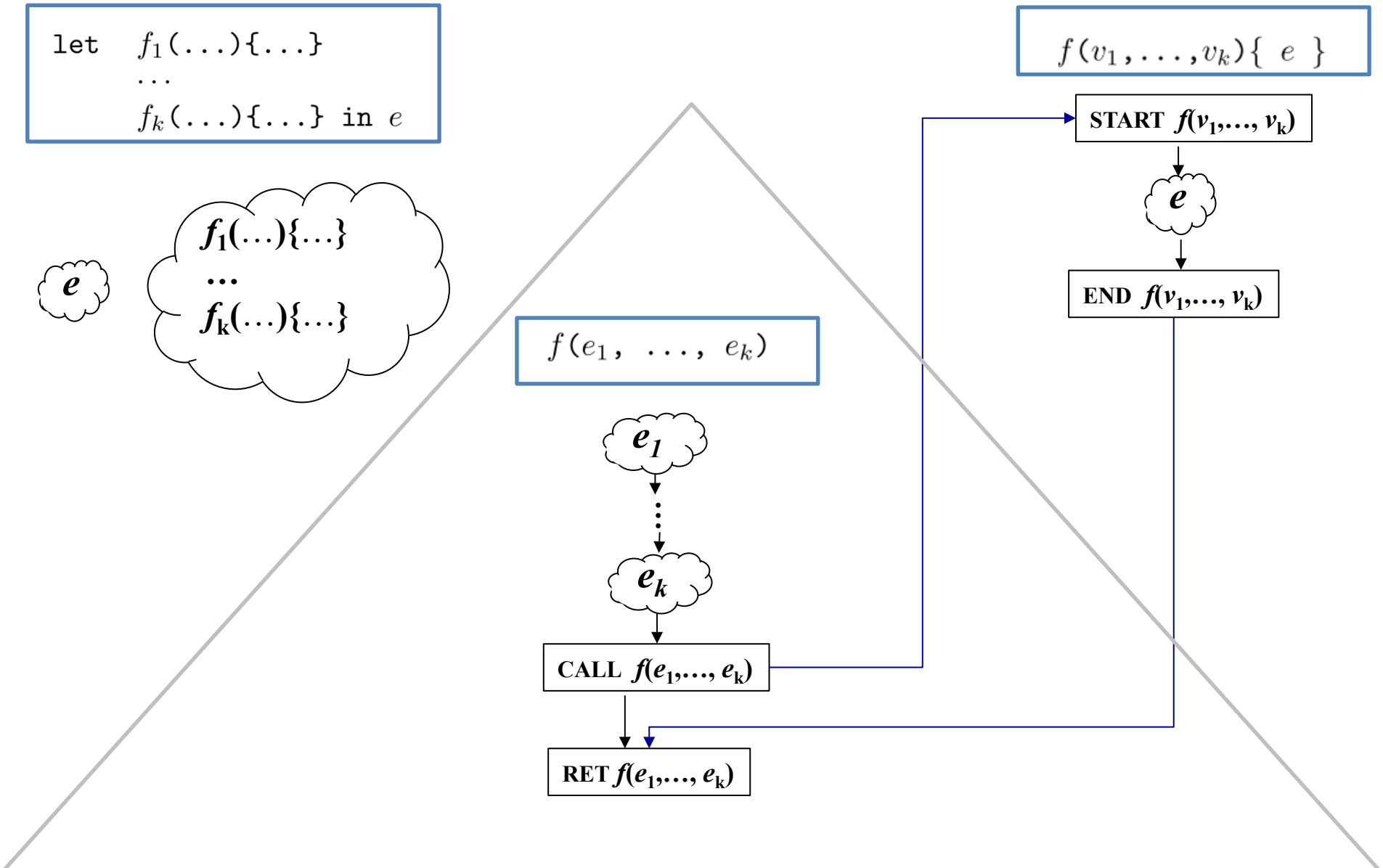
$$\begin{cases} V = V_1 \cup V_2 \cup \{diamond, glue\} \\ E = \ E_1 \cup E_2 \cup \{(out_1, \epsilon, diamond), (diamond, \texttt{T}, in_2), \\ \qquad (diamond, \texttt{F}, glue), (out_2, \epsilon, in_1\} \\ in = in_1 \\ out = glue \end{cases}$$

where $(V_1, E_1, in_1, out_1) = \mathsf{cfg}(e_1, \psi)$ and $(V_2, E_2, in_2, out_2) = \mathsf{cfg}(e_2, \psi)$ and $diamond$ and $glue$ are two new nodes.

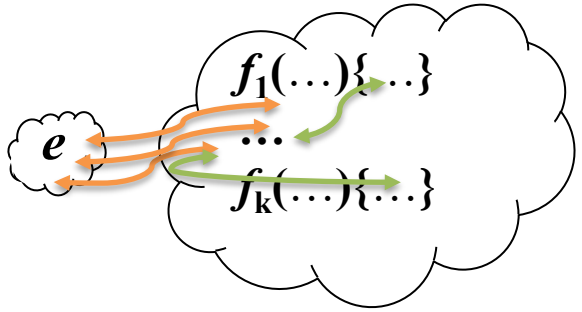
# **Inter**prozedurale Kontrollflussanalyse



# **Inter**prozedurale Kontrollflussanalyse



# CFG für LET-Ausdruck




$$\mathsf{cfg}(w, \psi) = (V, E, in, out), \text{ where}$$

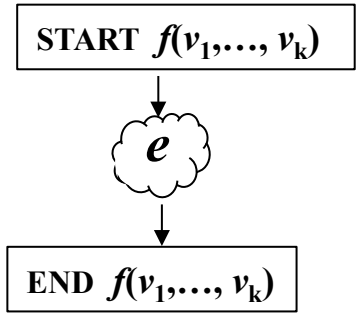
$$\text{if } w = \texttt{let } f_1(v_1^1, \ldots, v_{n_1}^1)\{e_1\} \cdots f_k(v_1^k, \ldots, v_{n_k}^k)\{e_k\} \texttt{ in } e \text{ then}$$

$$\begin{cases} V = V' \cup \bigcup V_i \\ E = E' \cup \bigcup E_i \\ in = in' \\ out = out' \end{cases}$$

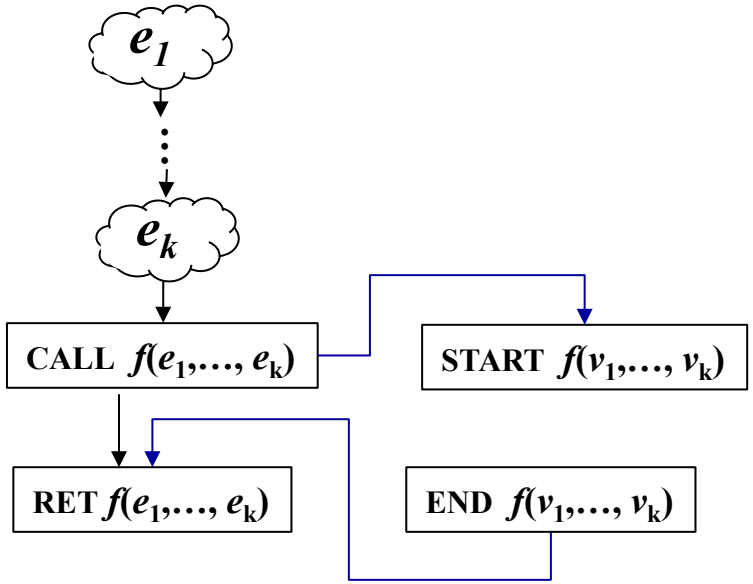
$$\text{where } \psi' = \psi[(in_1, out_1)/f_1] \cdots [(in_k, out_k)/f_k]$$
$$\text{with } in_i = (\mathsf{START}, f_i(v_1^i, \ldots, v_{n_i}^i)),\ out_i = (\mathsf{END}, f_i(v_1^i, \ldots, v_{n_i}^i)),$$
$$\text{and } (V', E', in', out') = \mathsf{cfg}(e, \psi')$$
$$\text{and } (V_i, E_i, in_i, out_i) = \mathsf{cfg}(f_i(v_1^i, \ldots, v_{n_i}^i)\{e_i\}, \psi')$$

Funktioniert auch für verschachtelte LETs

# CFGs für Methodendefinition und -aufruf




$$\text{if } w = f(v_1, \ldots, v_k)\{\ e\ \} \text{ then}$$
$$\begin{cases} V = V' \cup \{\quad in_f, out_f\} \\ E = E' \cup \{(in_f, \epsilon, in'), (out', \epsilon, out_f)\} \\ in = in_f \\ out = out_f \end{cases}$$
$$\text{where } (V', E', in', out') = \mathsf{cfg}(e, \psi) \text{ and } (in_f, out_f) = \psi(f).$$




$$\text{if } w = f(e_1, \ \ldots, \ e_k) \text{ then}$$
$$\begin{cases} V = \{in_f, out_f, out\} \cup \bigcup V_i \\ E = \{(out_1, \epsilon, in_2), \ldots (out_{k-1}, \epsilon, in_k), \\ \qquad (out_k, \epsilon, call), (call, \epsilon, in_f), (out_f, \epsilon, ret), \\ \qquad (call, \epsilon, ret)\} \cup \bigcup E_i \\ in = in_1 \\ out = ret \end{cases}$$
$$\text{where } (V_i, E_i, in_i, out_i) = \mathsf{cfg}(e_i, \psi),$$
$$(in_f, out_f) = \psi(f) \text{ and } call = (\mathsf{CALL}, w),\ ret = (\mathsf{RET}, w).$$

```
let ggT(a,b) {
   if (a==b) then
         a
      else
         if (a>b) then
            ggT(a-b,b)
          else
            ggT (b-a,a) }
 in ggT(28, 49)
```

TRIPLA Sourc Code

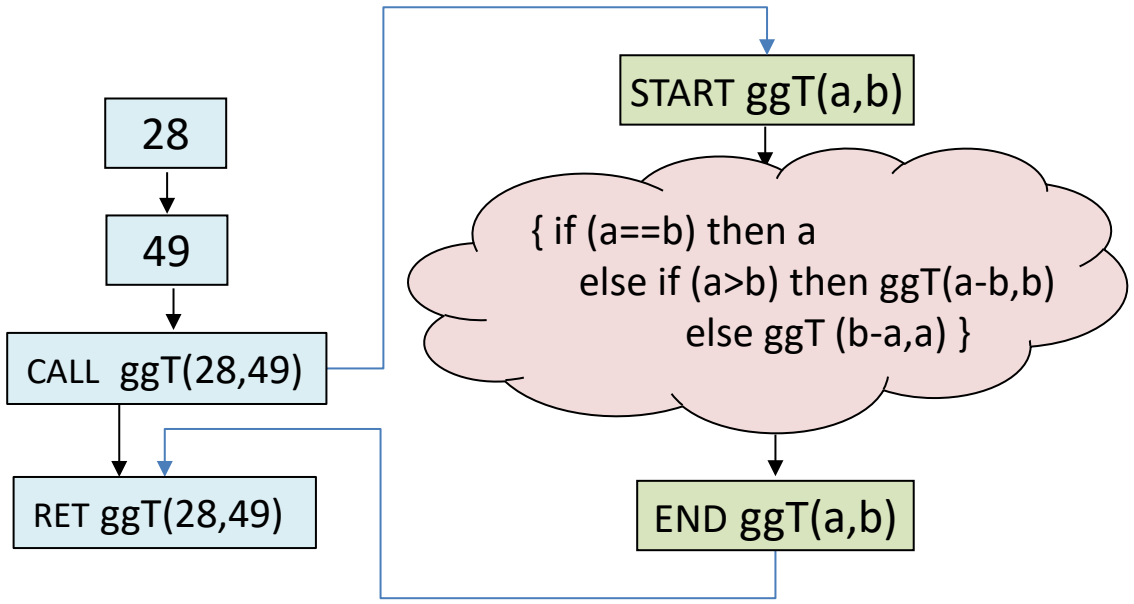
```
let ggT(a,b) { if (a==b) then a
                 else if (a>b) then ggT(a-b,b)
                      else ggT (b-a,a) }
 in ggT(28, 49)
```

1

ggT(28,49)

ggT(a,b) { if (a==b) then a
else if (a>b) then ggT(a-b,b)
else ggT (b-a,a) }

2



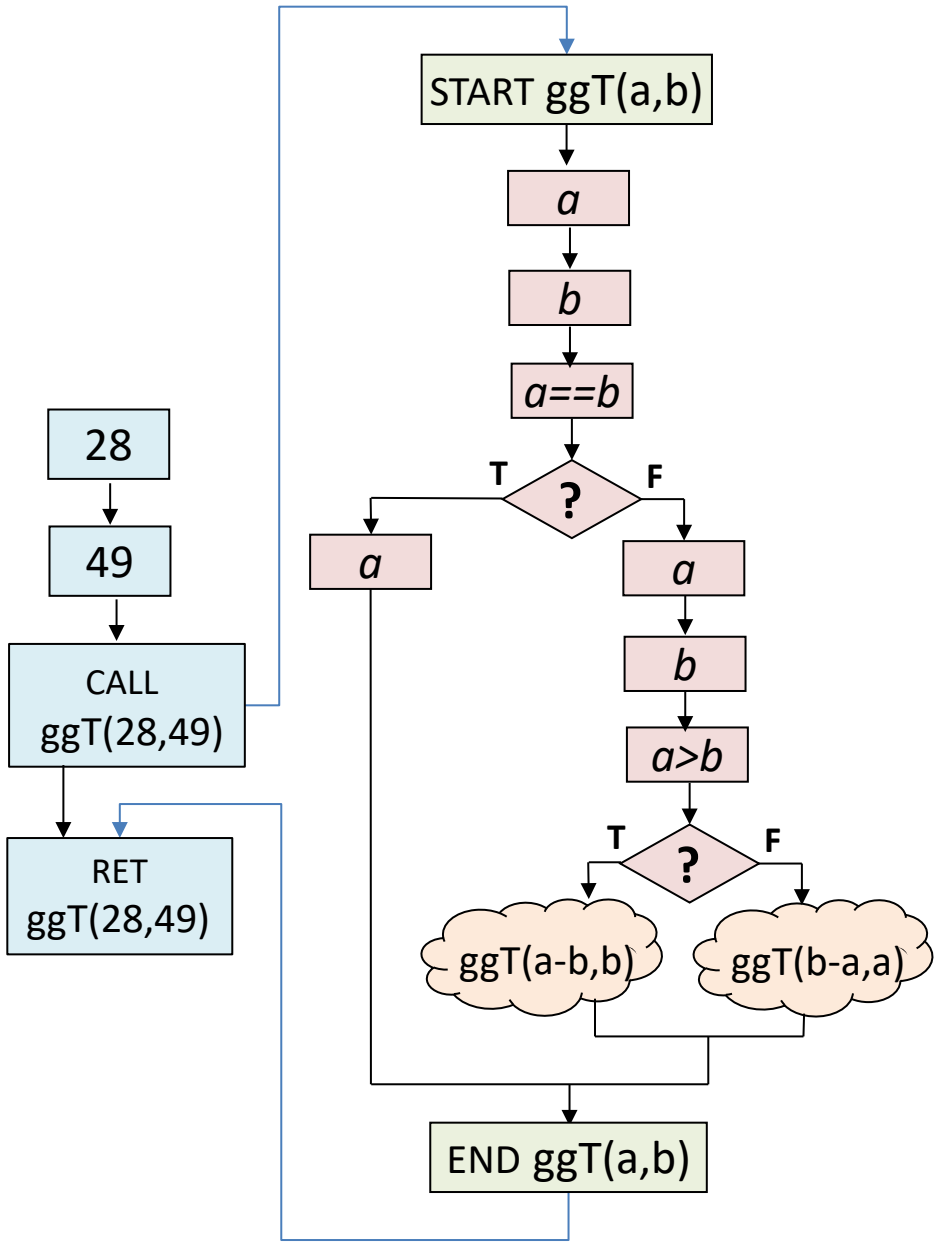

Abstract Syntax Tree

```
LET
└ FUNDEF
    ├ ggT(a,b)
    └ IF
        └ ==
            ├ a
            └ b
        ├ a
        └ IF
            └ >
                ├ a
                └ b
            └ FUNCALL
                ├ ggT
                └ -
                    ├ a
                    └ b
                └ b
            └ FUNCALL
                ├ ggT
                └ -
                    ├ b
                    └ a
                └ a
└ FUNCALL
    └ ggT
        ├ 28
        └ 49
```

TRIPLA Quellkode

```
let ggT(a,b) { if (a==b) then a
               else if (a>b) then ggT(a-b,b)
                    else ggT (b-a,a) }
 in ggT(28, 49)
```

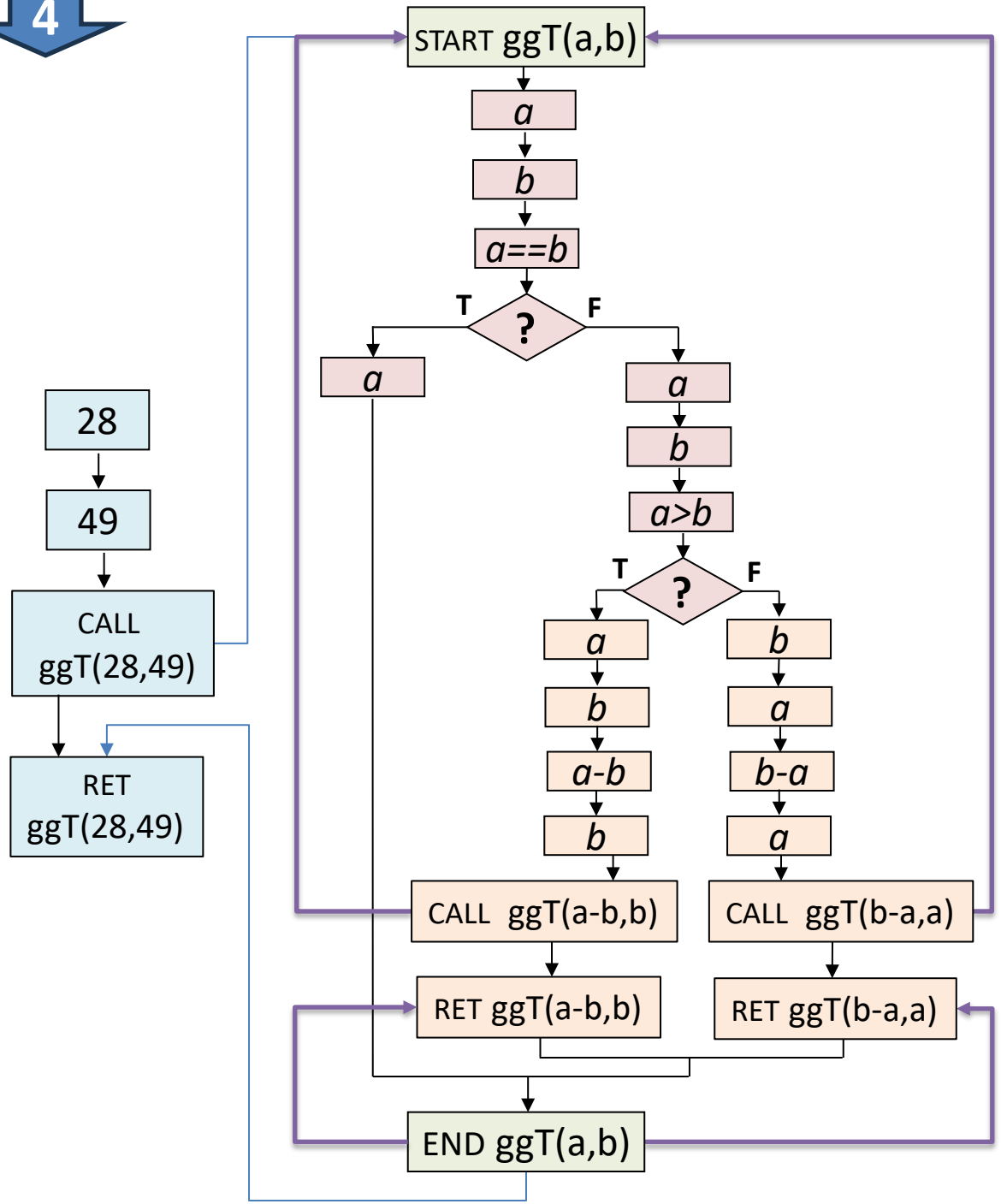
**endrekursiv**



TRIPLA Quellkode

```
let ggT(a,b) { if (a==b) then a
                    else if (a>b) then ggT(a-b,b)
                                  else ggT (b-a,a) }
 in ggT(28, 49)
```

4

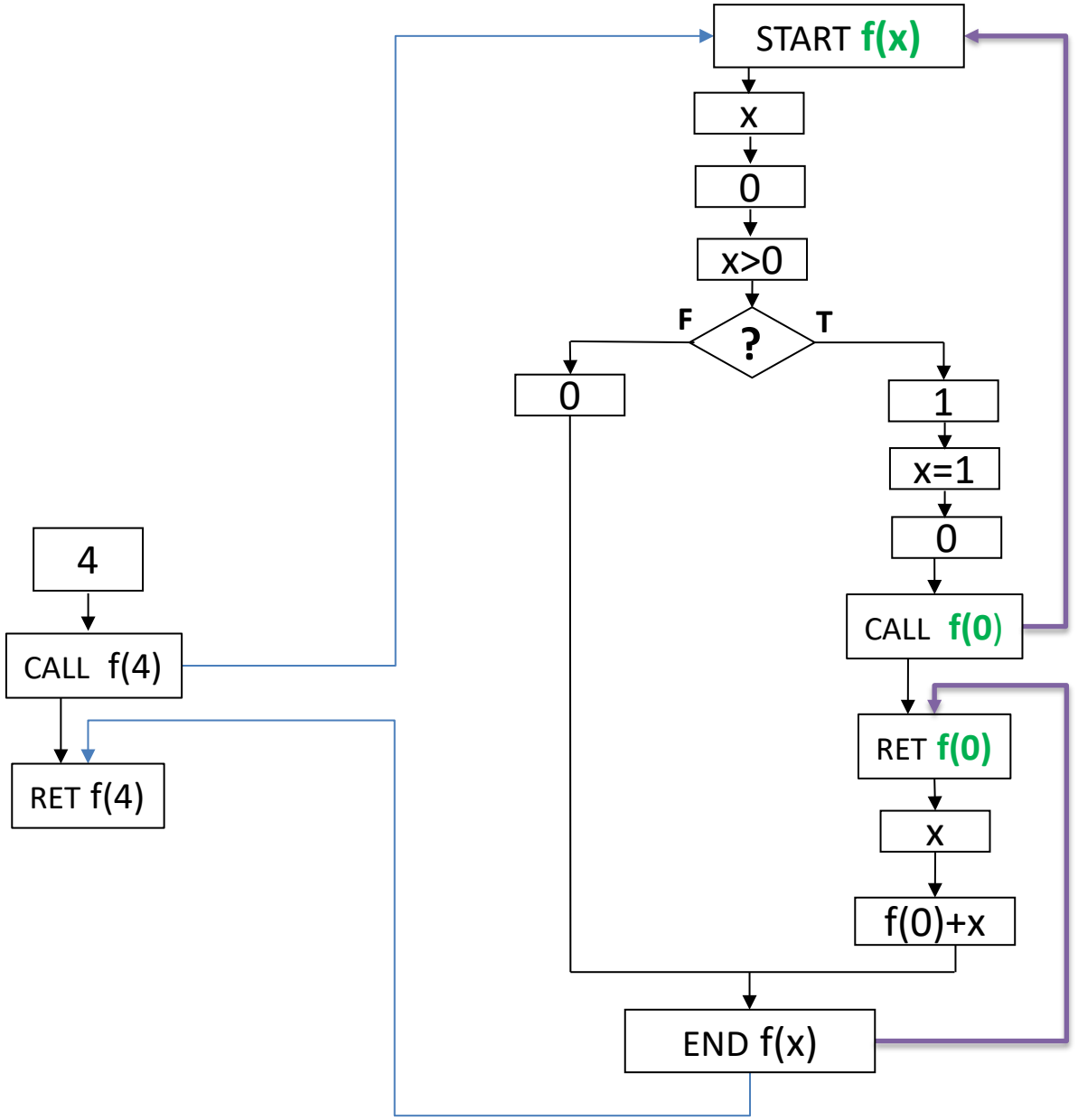


# Rekursion

TRIPLA Quellkode

```
let f(x) { if (x>0) then
             x=1; f(0)+x
           else 0 }
 in f(4)
```

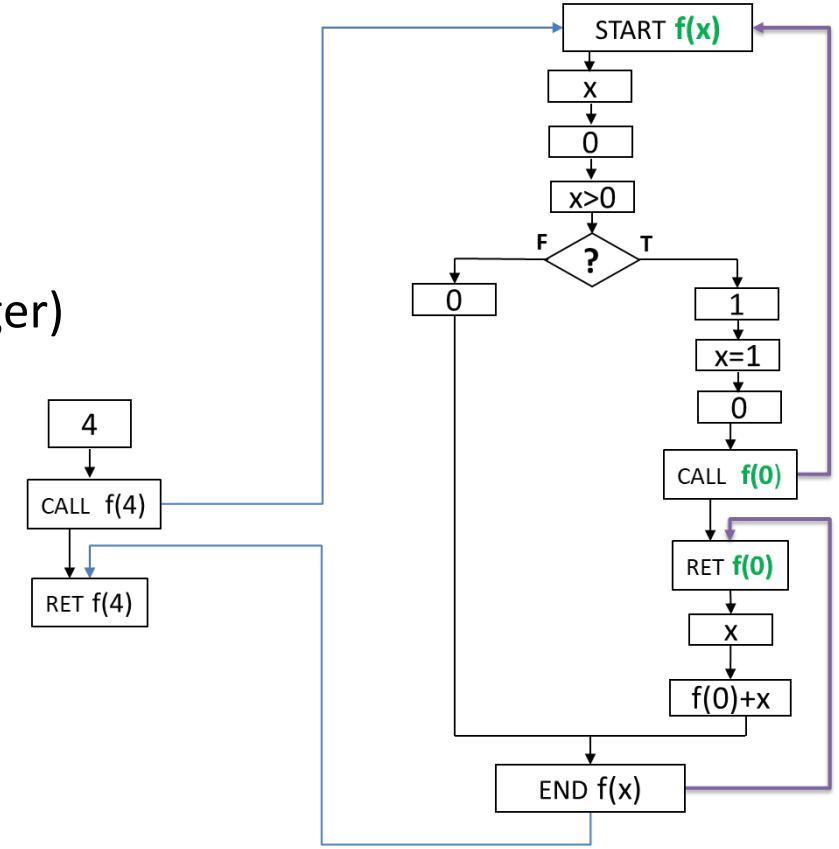
**rekursiv, aber nicht endrekursiv**



# Interprozedurale Datenflussanalyse

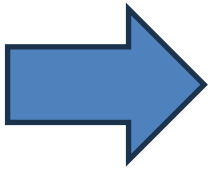
- Knotentypen
  - x (lesender Zugriff auf x)
  - c (Konstante)
  - e1 op e2 (binärer Ausdruck)
  - x=e (Zuweisung)
  - <?> („if true“, Bedingung steht in Vorgänger)
  - CALL
  - RET
  - START
  - END

**Implementierungshinweis:**
Statt Mengen von Kanten verwendet man besser Pointer.



# Konstantenpropagation: Beispiel

- a=0
- b=1
- c=2
- if (d>0) a=5 else b=a
- d=a*b+c



- a=0
- b=1
- c=2
- if (d>0) a=5 else b=0
- d=a*b+2

# Vorwärtsflussproblem: Konstantenpropagation

$$
\begin{array}{l}
IN(in) = \emptyset \qquad\qquad \mathcal{E} = L_G(ID) \times L_G(CONST) \\
OUT(in) = \emptyset \\
\textit{for all } v \in (V - \{in\}) \textit{ do} \\
\quad IN(v) = \mathcal{E} \\
\quad OUT(v) = (IN(v) - KILL(v)) \cup GEN(v) \\
\textit{while there are changes do} \\
\quad \textit{for all } v \in (V - \{in\}) \textit{ do} \\
\qquad IN(v) = \bigcap_{(p,l,v) \in E} OUT(p) \\
\qquad OUT(v) = (IN(v) - KILL(v)) \cup GEN(v)
\end{array}
$$

**Implementierungshinweis:**
$\mathcal{E}$ nicht erzeugen, sondern als speziellen Wert darstellen und als Sonderfall bei Mengenoperationen behandeln.

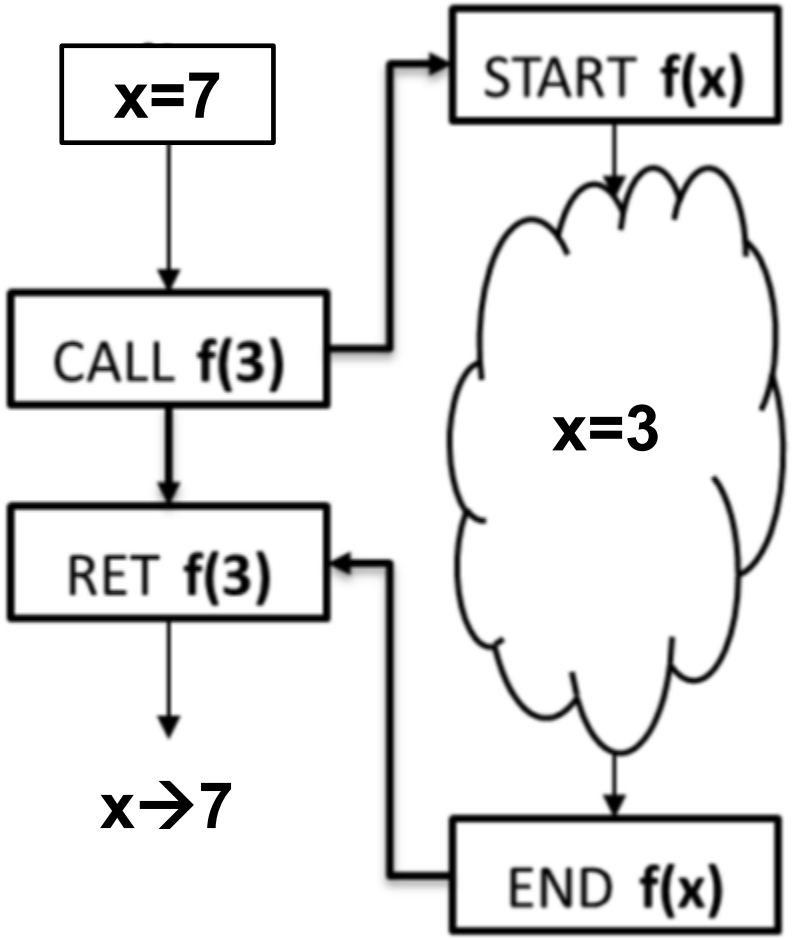
$$\grave{OUT}(v) := \{(id, c) | (id, c) \in IN(v) \text{ and } id \notin KILLVARS(v)\} \cup GEN(v)$$

$$
KILLVARS(v) = \begin{cases} \{id\} & \text{if } v = id\text{=}e \\ \{v_1 \ldots, v_k\} & \text{if } v = (\mathsf{START}, f(v_1, \ldots, v_k)) \\ \{v_1, \ldots, v_k\} & \text{if } v = (\mathsf{END}, f(v_1, \ldots, v_k)) \\ \emptyset & \text{otherwise} \end{cases}
$$

$$
KILL(v) = \{(id, c) \mid id \in KILLVARS(v) \text{ and } c \text{ is a constant}\}
$$

$$
GEN(v) = \begin{cases} \{(id, c)\} & \text{if } v = id\text{=}c \\ \emptyset & \text{otherwise} \end{cases}
$$

**Analog zu Erreichbaren Ausdrücken**


x=7
START f(x)
CALL f(3)
x=3
RET f(3)
x→7
END f(x)

# Rückwärtsflussproblem: Lebendige Variablen

$$KILLVARS(v) = \begin{cases} \{id\} & \text{if } v = id{=}e \\ \{v_1 \ldots, v_k\} & \text{if } v = (\mathsf{START}, f(v_1, \ldots, v_k)) \\ \{v_1, \ldots, v_k\} & \text{if } v = (\mathsf{END}, f(v_1, \ldots, v_k)) \\ \emptyset & \text{otherwise} \end{cases}$$

```
for all v ∈ V do
    IN(v) := GEN(v)
while there are changes do
  for all v ∈ V do
       if v == (CALL, w) then
           ∃!ret ∈ succ(v) with ret = (RET, w)
           ∃!start ∈ succ(v) with start = (START, w')
           ∃!end ∈ pred(ret) with end = (END, w')
           Δ := (OUT(end) − OUT(start)) − KILL(start)
           IN(v) := (OUT(ret) ∪ IN(start)) − Δ
       else
          OUT(v) := ⋃_{s∈succ(v)} IN(s)
          IN(v) := (OUT(v) − KILL(v)) ∪ GEN(v)
```

$$KILL(v) = KILLVARS(v)$$

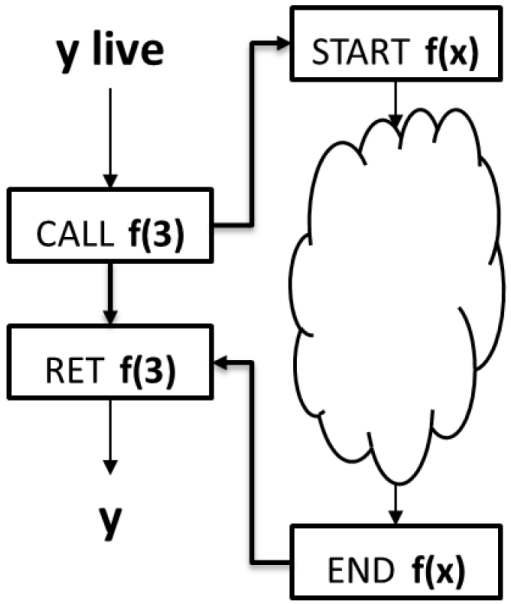
$$GEN(v) = \begin{cases} \{id\} & \text{if } v = id \\ \emptyset & \text{otherwise} \end{cases}$$

**Hinweis:**
In der vorherigen Vorlesung wurde USE und DEF statt GEN und KILL verwendet, um die Mengen zu benennen..

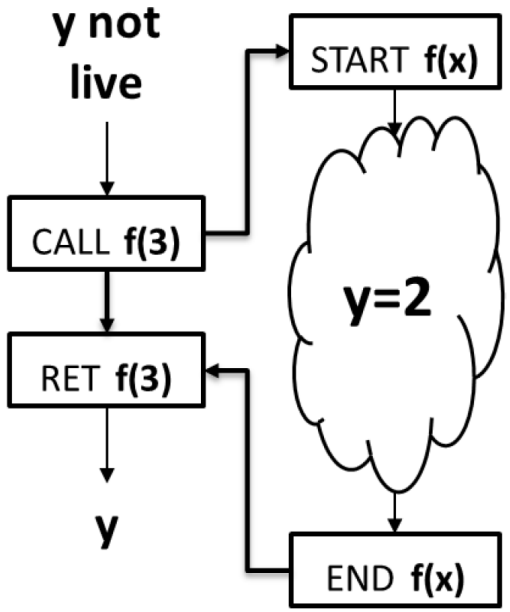
# Interprozeduale DFA: Lebendige Variablen

## 1. Fall: Äußere Variable nicht durch Parameter überdeckt

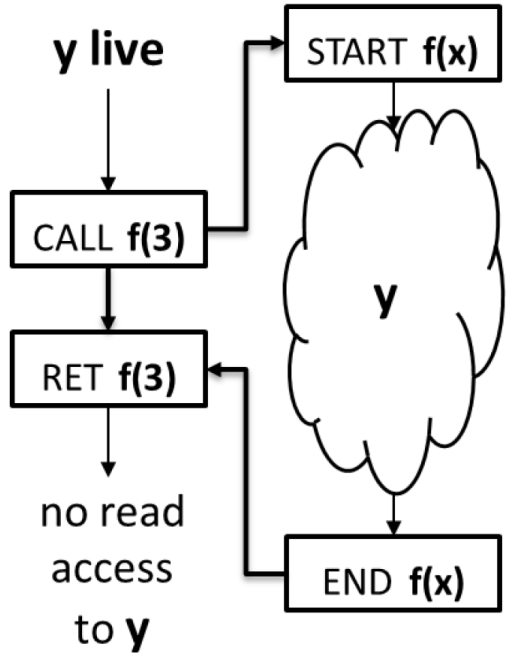
Rückwärtsflussanalyse




(a) The variable y is read after the function call, but neither written nor read in f().
$\Delta = (\{y\} - \{y\}) - \{x\} = \emptyset$
$IN = (\{y\} \cup \emptyset) - \emptyset = \{y\}$




(b) The variable y is read after the function call, but written along all paths in f().
$\Delta = (\{y\} - \emptyset) - \{x\} = \{y\}$
$IN = (\{y\} \cup \{y\}) - \{y\} = \emptyset$

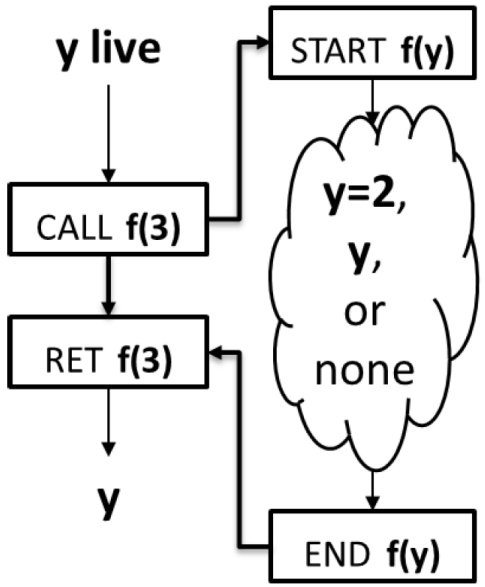



(c) There is at least one path along which the variable y is not written before it is read in f().
$\Delta = (\emptyset - \{y\}) - \{x\} = \emptyset$
$IN = (\emptyset \cup \{y\}) - \emptyset = \{y\}$

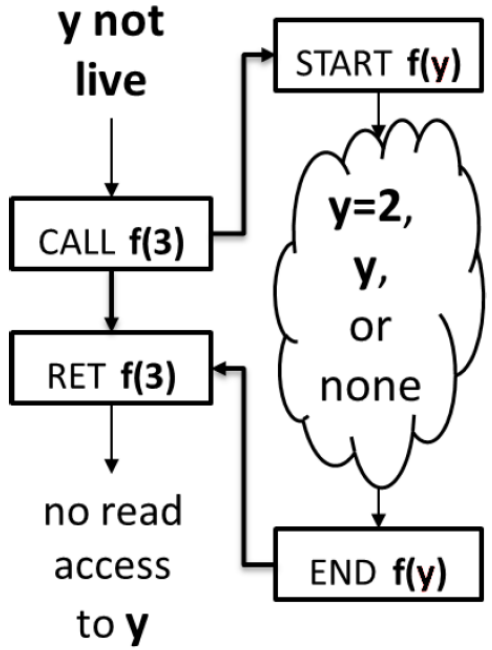
# Interprozeduale DFA: Lebendige Variablen

## 2. Fall: Äußere Variable durch gleichnamigen Parameter überdeckt

Rückwärtsflussanalyse




(a) The variable y is read after the function call. All occurrences of y in f() refer to the parameter with the same name, not to the outer variable y.
$\Delta = (\{y\} - M) - \{y\} = \emptyset$
$IN = (\{y\} \cup \emptyset) - \emptyset = \{y\}$

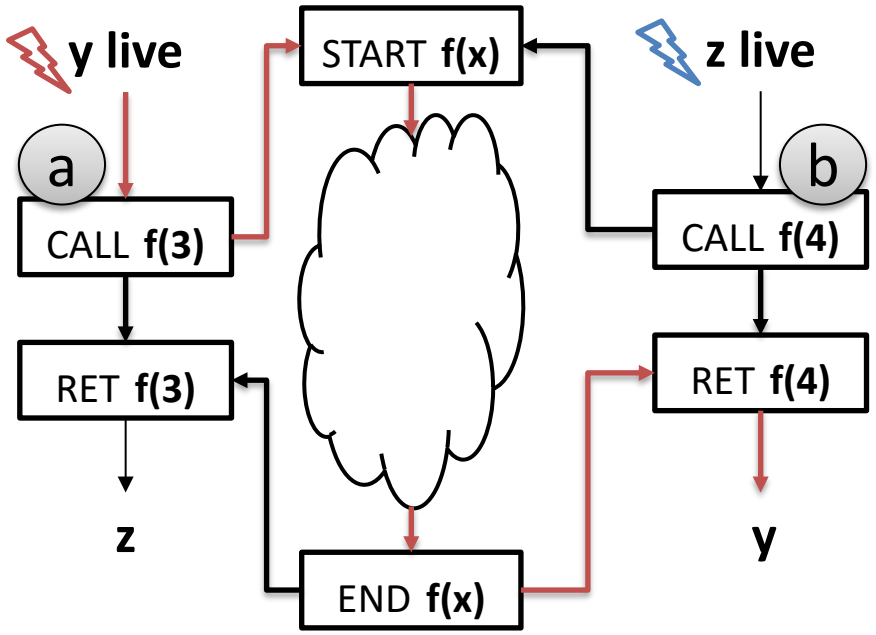



(b) The variable y is not read after the function call. All occurrences of y in f() refer to the parameter with the same name, not to the outer variable y.
$\Delta = (\emptyset - M) - \{y\} = \emptyset$
$IN = (\emptyset \cup \emptyset) - \emptyset = \emptyset$

For the computation: $OUT(start) = M$ where $M = \{\mathbf{y}\}$ if **y** is read and not written along at least one path in **f** and $M = \emptyset$ otherwise.

# Interprozeduale DFA: Lebendige Variablen

## Problem: Propagation entlang ungültiger Pfade

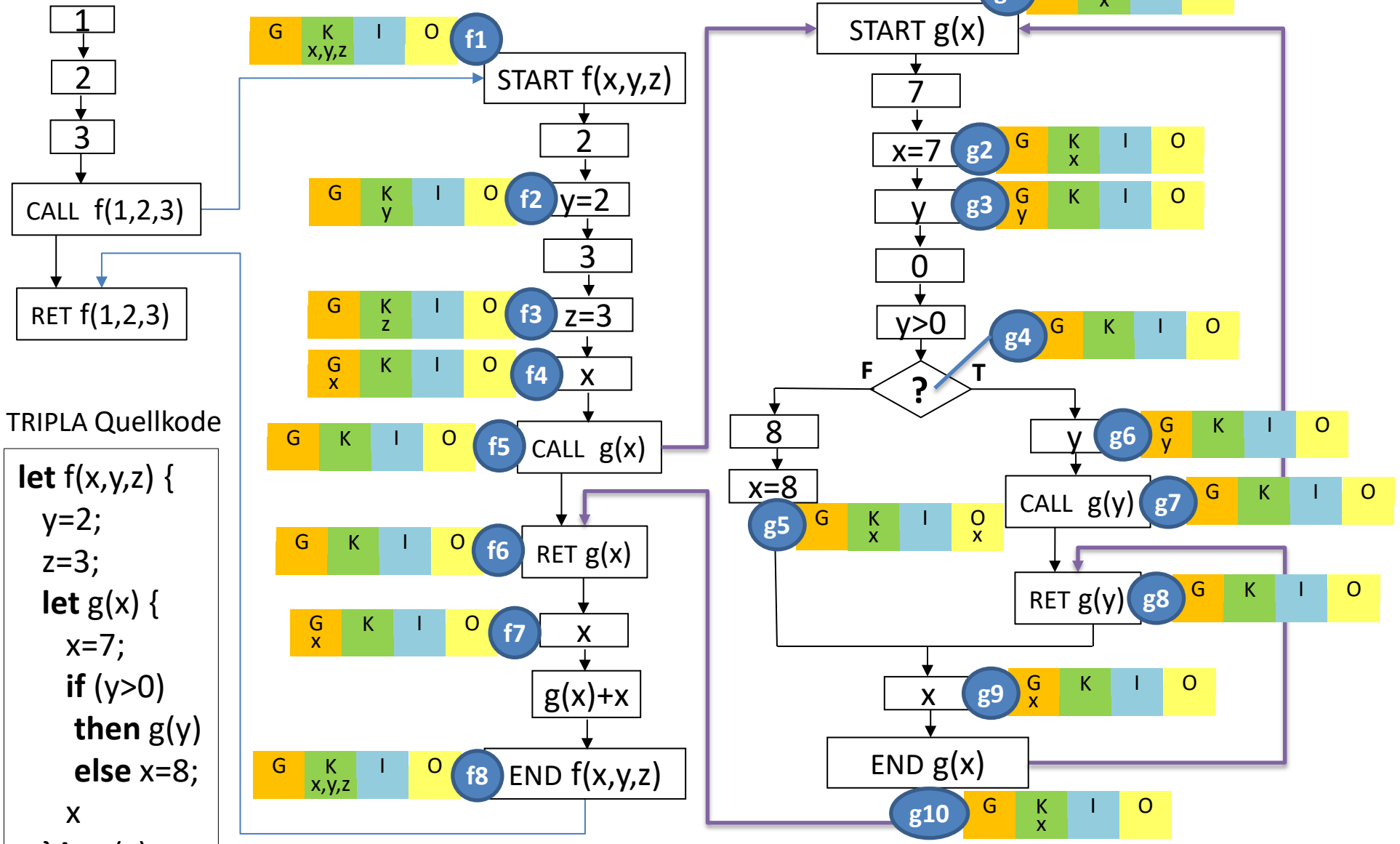



$$\Delta = (\{y, z\} - \{y, z\}) - \{x\} = \emptyset$$

a) $IN = (\{z\} \cup \{y, z\}) - \emptyset = \{y, z\}$

b) $IN = (\{y\} \cup \{y, z\}) - \emptyset = \{y, z\}$

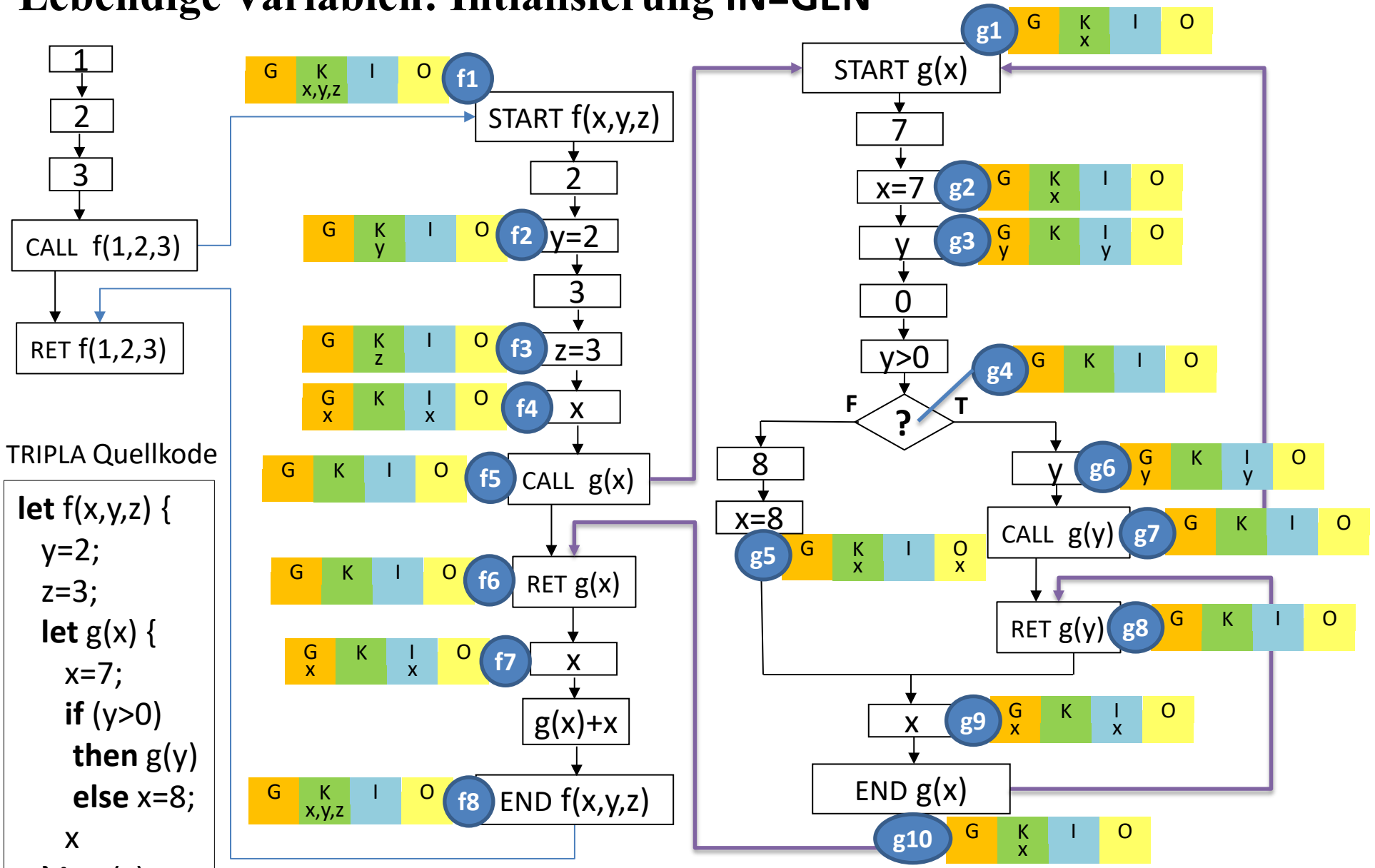
# Lebendige Variablen: KILL und GEN




TRIPLA Quellkode

```
let f(x,y,z) {
  y=2;
  z=3;
  let g(x) {
    x=7;
    if (y>0)
     then g(y)
     else x=8;
    x
  } in g(x)+x
} in f(2,3,0)
```

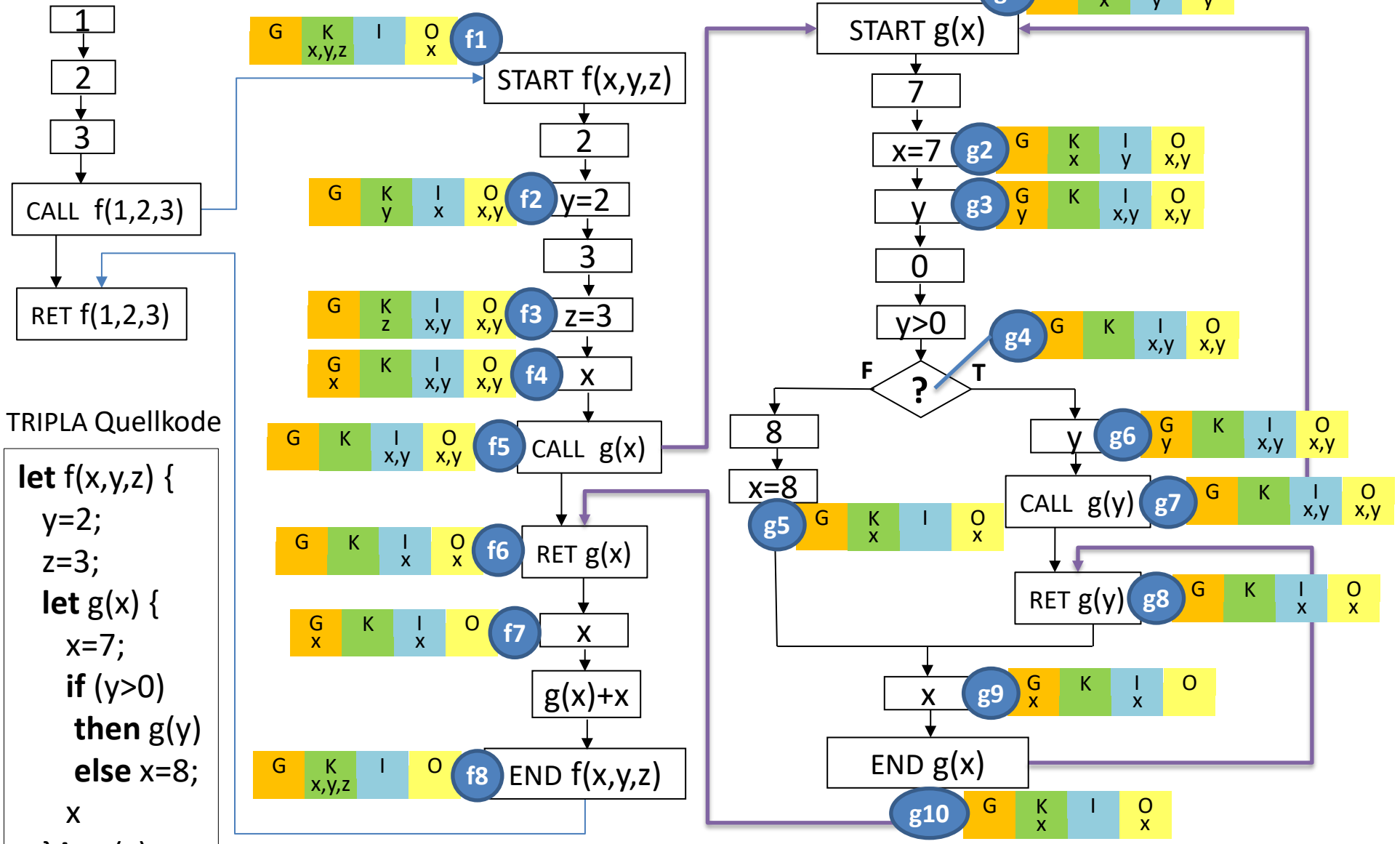
# Lebendige Variablen: Intialisierung IN=GEN




TRIPLA Quellkode

```
let f(x,y,z) {
  y=2;
  z=3;
  let g(x) {
    x=7;
    if (y>0)
     then g(y)
     else x=8;
    x
  } in g(x)+x
} in f(2,3,0)
```

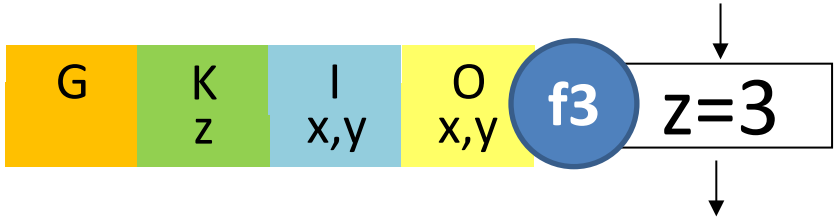
# Lebendige Variablen: Ergebnis




TRIPLA Quellkode

```
let f(x,y,z) {
  y=2;
  z=3;
  let g(x) {
    x=7;
    if (y>0)
     then g(y)
     else x=8;
    x
  } in g(x)+x
} in f(2,3,0)
```

# Programmoptimierung




- z ist hier nicht lebendig, d.h. es gibt keine zukünftige Verwendung von z. Daher kann auf die Zuweisung verzichtet werden, im Allgemeinen jedoch wegen möglicher Seiteneffekte nicht auf die Auswertung der rechten Seite der Zuweisung.
- Im Programmcode kann $id = e$ durch $e$ ersetzt werden, wenn für den entsprechenden Knoten $v$ im Kontrollflussgraphen $id \notin OUT(v)$ gilt.

# Rückwärtsflussproblem: Reached Uses
## (Erreichte Verwendungen)

$$KILL(v) = \begin{cases} \{(p', id) \mid p' \in P\} & \text{if } v = id{=}e^{\,p} \\ \{(p', id_1) \ldots, (p', id_k) \mid p' \in P\} & \text{if } v = (\mathsf{START}, f(id_1, \ldots, id_k)^p) \\ \{(p', id_1), \ldots, (p', id_k) \mid p' \in P\} & \text{if } v = (\mathsf{END}, f(id_1, \ldots, id_k)^p) \\ \emptyset & \text{otherwise} \end{cases}$$

$$GEN(v) = \begin{cases} \{(p, id)\} & \text{if } v = id^{\,p} \\ \emptyset & \text{otherwise} \end{cases}$$

```
for all v ∈ V do
    IN(v) := GEN(v)
while there are changes do
  for all v ∈ V do
      if v == (CALL, w) then
          ∃!ret ∈ succ(v) with ret = (RET, w)
          ∃!start ∈ succ(v) with start = (START, w')
          ∃!end ∈ pred(ret) with end = (END, w')
          Δ := (OUT(end) − OUT(start)) − KILL(start)
          IN(v) := (OUT(ret) ∪ IN(start)) − Δ
      else
          OUT(v) := ⋃_{s∈succ(v)} IN(s)
          IN(v) := (OUT(v) − KILL(v)) ∪ GEN(v)
```

**Analog zu Lebendige Variablen**

# Rückwärtsflussproblem: Reached Uses
## (Erreichte Verwendungen)

$$KILL(v) = \begin{cases} \{id\} & \text{if } v = id\text{=}e^{\,p} \\ \{id_1, \ldots, id_k\} & \text{if } v = (\mathsf{START}, f(id_1, \ldots, id_k)^p) \\ \{id_1, \ldots, id_k\} & \text{if } v = (\mathsf{END}, f(id_1, \ldots, id_k)^p) \\ \emptyset & \text{otherwise} \end{cases}$$

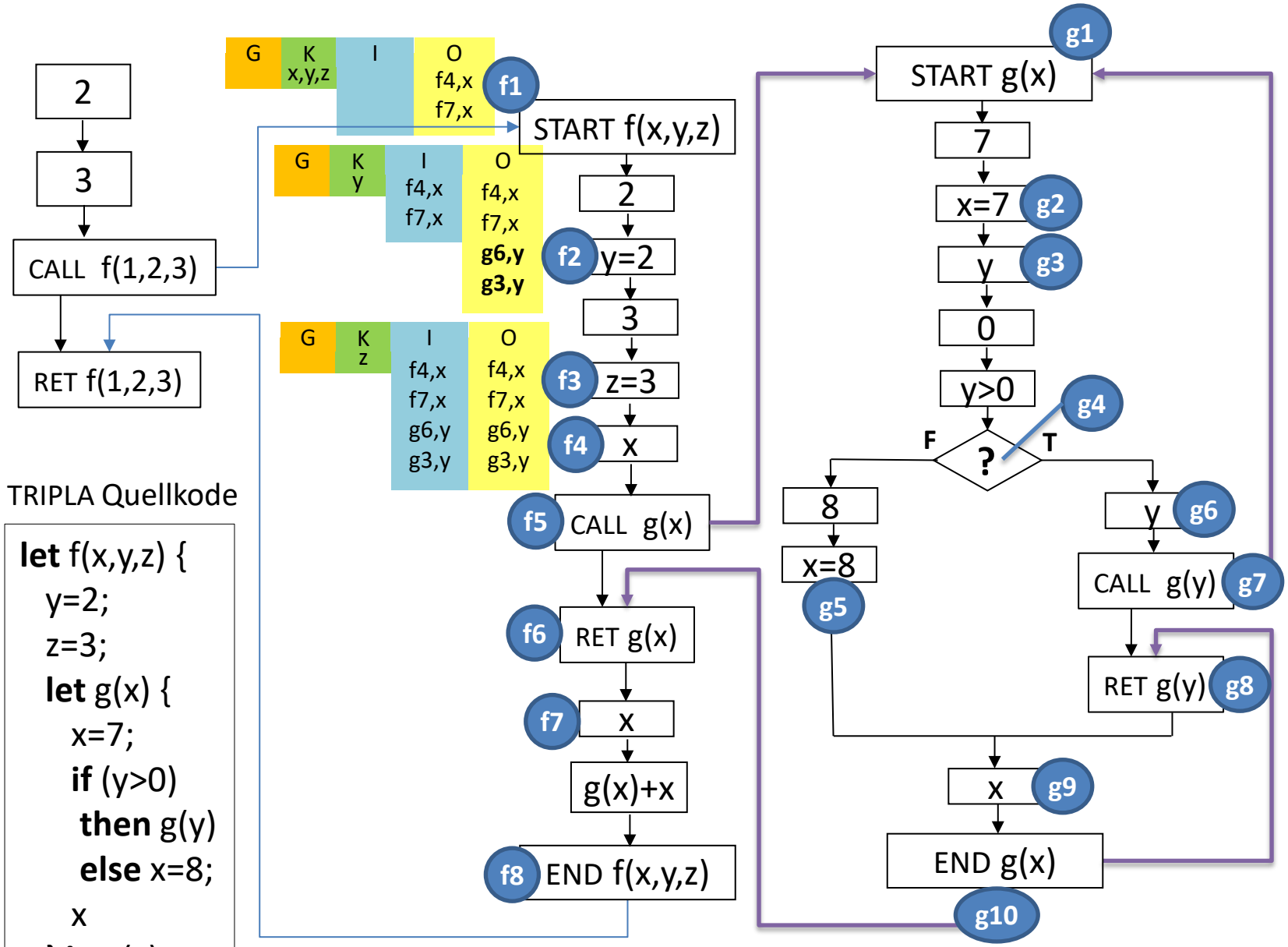
$$GEN(v) = \begin{cases} \{(p, id)\} & \text{if } v = id^{\,p} \\ \emptyset & \text{otherwise} \end{cases}$$

$$M \ominus K = \{(p, id) \mid (p, id) \in M \text{ and } id \notin K\}$$

```
for all v ∈ V do
    IN(v) := GEN(v)
while there are changes do
  for all v ∈ V do
      if v == (CALL, w) then
          ∃!ret ∈ succ(v) with ret = (RET, w)
          ∃!start ∈ succ(v) with start = (START, w')
          ∃!end ∈ pred(ret) with end = (END, w')
          Δ := (OUT(end) − OUT(start)) ⊖ KILL(start)
          IN(v) := (OUT(ret) ∪ IN(start)) − Δ
      else
          OUT(v) := ⋃_{s∈succ(v)} IN(s)
          IN(v) := (OUT(v) ⊖ KILL(v)) ∪ GEN(v)
```

**Analog zu Lebendige Variablen**

# Reached Uses




TRIPLA Quellkode

```
let f(x,y,z) {
  y=2;
  z=3;
  let g(x) {
    x=7;
    if (y>0)
     then g(y)
     else x=8;
    x
  } in g(x)+x
} in f(2,3)
```

$$
\begin{aligned}
&\texttt{if } v == (\text{CALL}, w) \texttt{ then}\\
&\quad \exists! ret \in \text{succ}(v) \texttt{ with } ret = (\text{RET}, w)\\
&\quad \exists! start \in \text{succ}(v) \texttt{ with } start = (\text{START}, w')\\
&\quad \exists! end \in \text{pred}(ret) \texttt{ with } end = (\text{END}, w')\\
&\quad \Delta := (OUT(end) - OUT(start)) \ominus KILL(start)\\
&\quad IN(v) := (OUT(ret) \cup IN(start)) - \Delta\\
&\texttt{else}\\
&\quad OUT(v) := \bigcup_{s \in \text{succ}(v)} IN(s)\\
&\quad IN(v) := (OUT(v) \ominus KILL(v)) \cup GEN(v)
\end{aligned}
$$

| Nodes (in order of visit) | GEN | KILL | $OUT_0$={}<br>$OUT_i$ = Union of IN of succcessors<br><br>$OUT_1$ | $IN_0$=GEN<br><br>$IN_i$=(OUT-KILL) + GEN<br><br>$IN_1$ | $OUT_2$ | $IN_2$ | |
|---|---|---|---|---|---|---|---|
| f8 | | | | | | | |
| f7 | (f7,x) | | | (f7,x) | | | |
| f6 | | | (f7,x) | (f7,x) | | | |
| g10 | | (_,x) | (f7,x) | | (f7,x),(g9,x) | | no changes to propagate |
| g9 | (g9,x) | | | (g9,x) | | | |
| g8 | | | (g9,x) | (g9,x) | | | |
| g7 | | | (g9,x) | (g9,x) | (g9,x),(g6,y),(g3,y) | (g9,x),(g6,y),(g3,y) | |
| g6 | (g6,y) | | (g9,x) | (g9,x),(g6,y) | (g9,x),(g6,y),(g3,y) | (g9,x),(g6,y),(g3,y) | |
| g5 | | (_,x) | (g9,x) | | | | |
| g4 | | | (g9,x),(g6,y) | (g9,x),(g6,y) | (g9,x),(g6,y),(g3,y) | (g9,x),(g6,y),(g3,y) | |
| g3 | (g3,y) | | (g9,x),(g6,y) | (g9,x),(g6,y),(g3,y) | (g9,x),(g6,y),(g3,y) | (g9,x),(g6,y),(g3,y) | no change to propagate |
| g2 | | (_,x) | (g9,x),(g6,y),(g3,y) | (g6,y),(g3,y) | | | |
| g1 | | (_,x) | (g6,y),(g3,y) | (g6,y),(g3,y) | | | |
| f5 | | | (f7,x),(g6,y),(g3,y) | (f7,x),(g6,y),(g3,y) | **(f7,x),(g6,y),(g3,y)** | | |
| f4 | (f4,x) | | (f7,x),(g6,y),(g3,y) | (f4,x),(f7,x),(g6,y),(g3,y) | | | |
| f3 | | (_,z) | (f4,x),(f7,x),(g6,y),(g3,y) | (f4,x),(f7,x),(g6,y),(g3,y) | | | |
| f2 | | (_,y) | (f4,x),(f7,x),(g6,y),(g3,y) | (f4,x),(f7,x) | | | |
| f1 | | (_,x),(_,y),(_,z) | (f4,x),(f7,x) | | | | |

Tabelle nicht auf dem neusten Stand !!!

2nd Iteration: Where did the IN-value of a successor change? => When was a node visited before one of its predecessors? => Where are cylces? => Where are backward edges? => g10->g8 and g7->g1

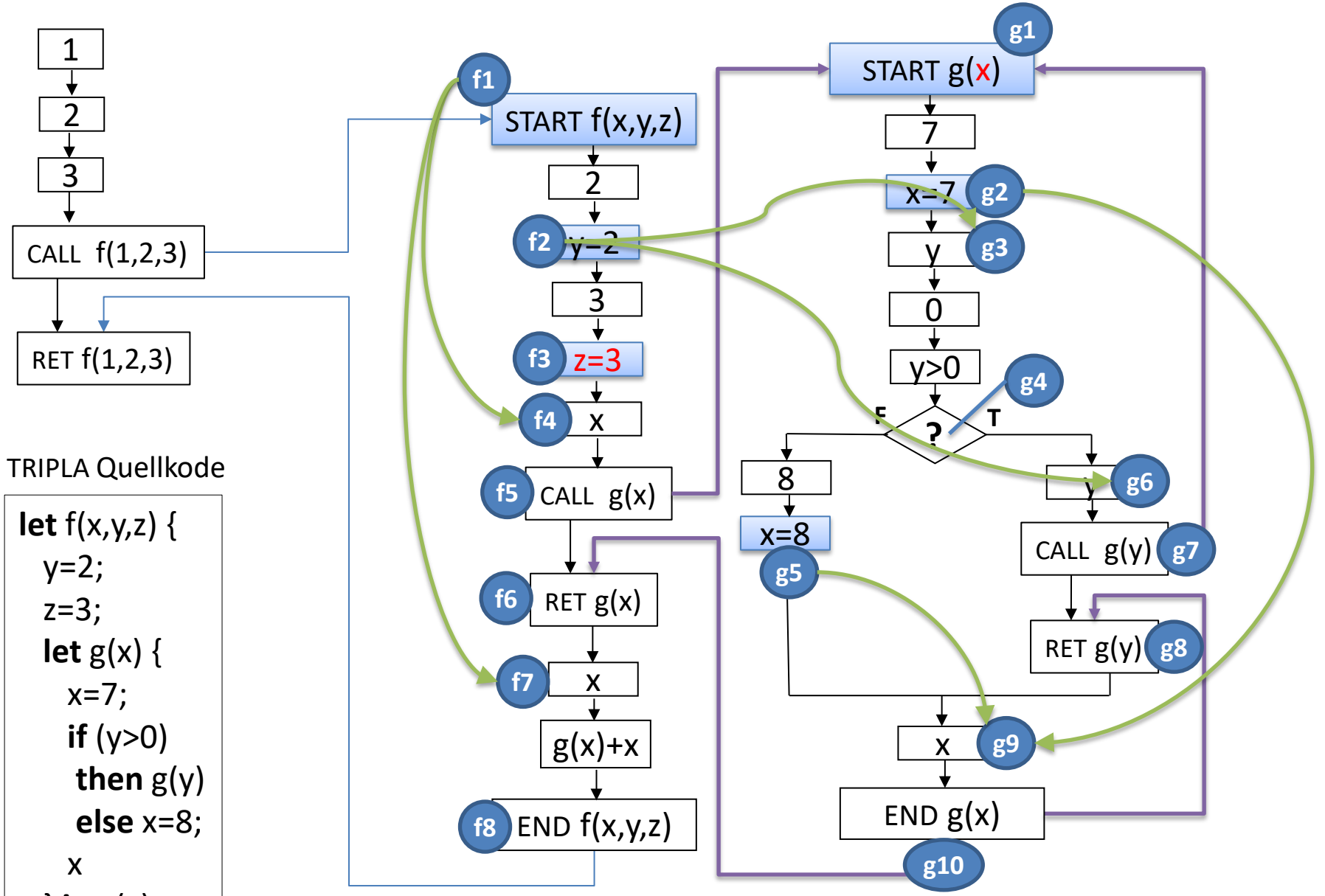
# Datenflussgraph

Mit Hilfe der erreichbaren Verwendungen (Reached Uses) kann der Datenflussgraph konstruiert werden:

$$G_D = (V, E_D)$$

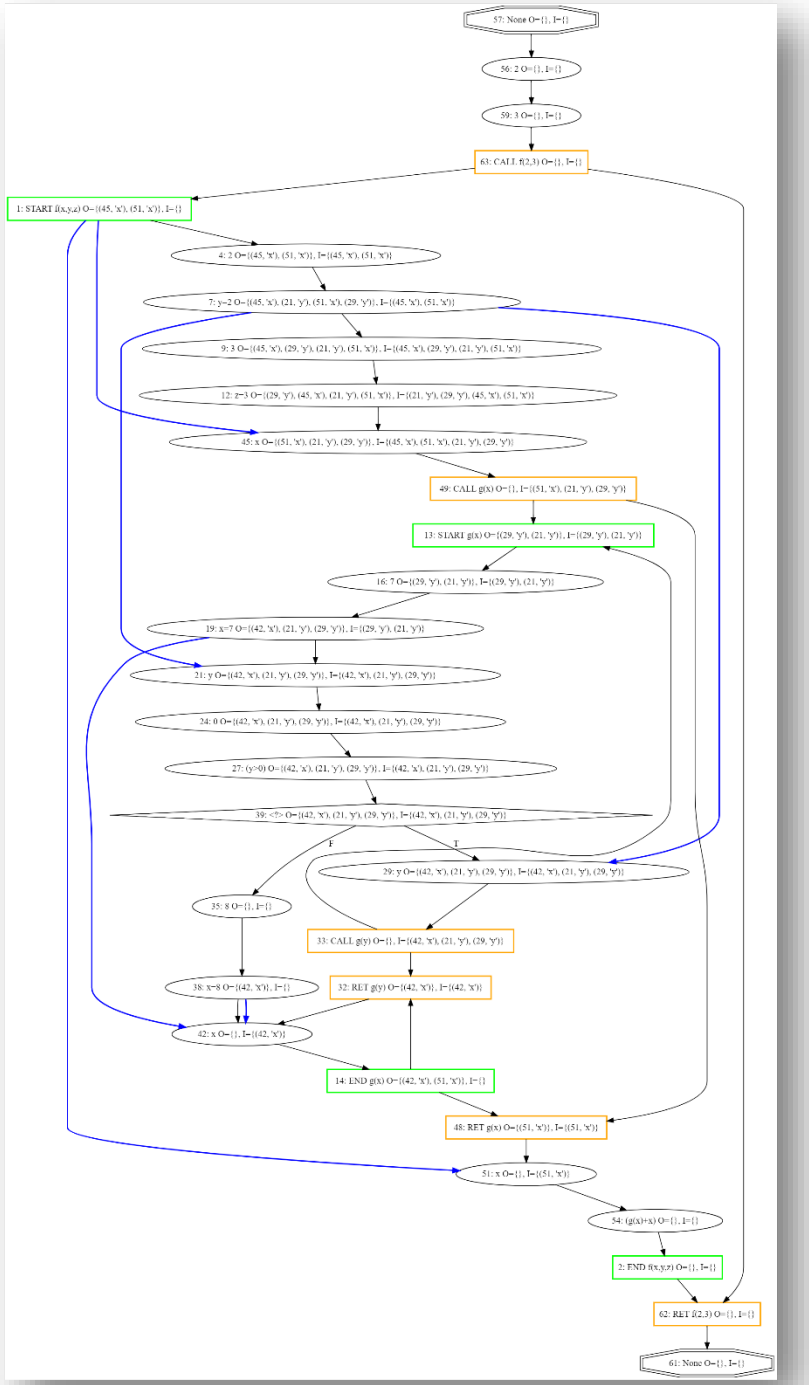
$$\begin{aligned} E_D \quad = \quad & \{(v, id^p) \mid v \in V, (p, id) \in OUT(v) \\ & \text{and } v = (\mathsf{START}, f(id_1, \ldots, id_k)^p) \text{ where } \exists i : id_i = id \\ & \text{or } v = id{=}e^{\,p}\} \cup E \end{aligned}$$

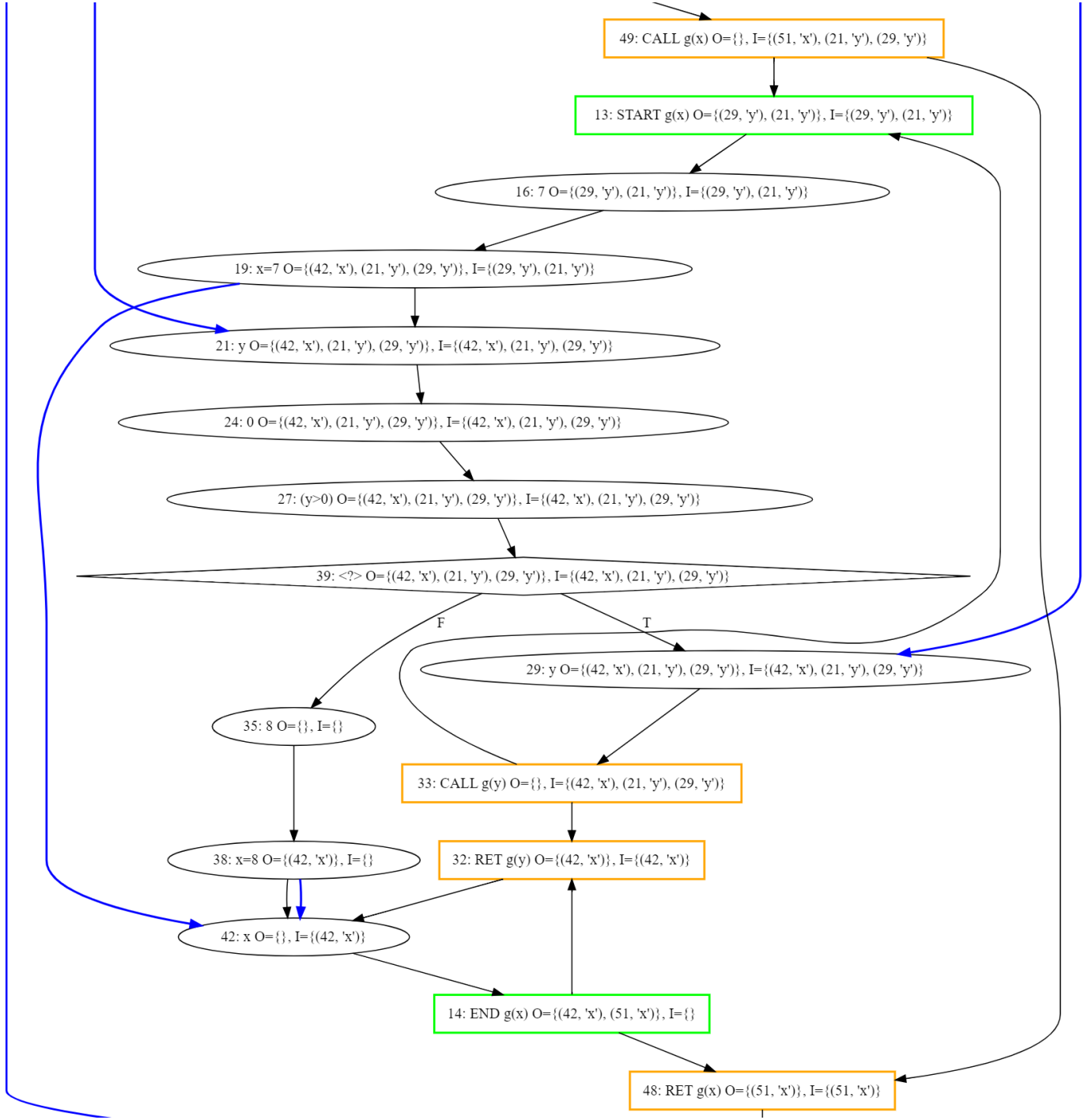
Es wird eine Datenflusskante von jeder Zuweisung x=e zu den Verwendungen (lesenden Zugriffen) der Variablen x erzeugt, sowie vom Eingangsknoten des CFGs jeder Funktion zu den Verwendungen ihrer Parameter.




TRIPLA Quellkode

```
let f(x,y,z) {
  y=2;
  z=3;
  let g(x) {
    x=7;
    if (y>0)
     then g(y)
     else x=8;
    x
  } in g(x)+x
} in f(2,3)
```


57: None O={}, I={}
56: 2 O={}, I={}
59: 3 O={}, I={}
63: CALL f(2,3) O={}, I={}
1: START f(x,y,z) O={(45, 'x'), (51, 'x')}, I={}
4: 2 O={(45, 'x'), (51, 'x')}, I={(45, 'x'), (51, 'x')}
7: y=2 O={(45, 'x'), (21, 'y'), (51, 'x'), (29, 'y')}, I={(45, 'x'), (51, 'x')}
9: 3 O={(45, 'x'), (29, 'y'), (21, 'y'), (51, 'x')}, I={(45, 'x'), (29, 'y'), (21, 'y'), (51, 'x')}
12: z=3 O={(29, 'y'), (45, 'x'), (21, 'y'), (51, 'x')}, I={(21, 'y'), (29, 'y'), (45, 'x'), (51, 'x')}
45: x O={(51, 'x'), (21, 'y'), (29, 'y')}, I={(45, 'x'), (51, 'x'), (21, 'y'), (29, 'y')}
49: CALL g(x) O={}, I={(51, 'x'), (21, 'y'), (29, 'y')}
13: START g(x) O={(29, 'y'), (21, 'y')}, I={(29, 'y'), (21, 'y')}
16: 7 O={(29, 'y'), (21, 'y')}, I={(29, 'y'), (21, 'y')}
19: x=7 O={(42, 'x'), (21, 'y'), (29, 'y')}, I={(29, 'y'), (21, 'y')}
21: y O={(42, 'x'), (21, 'y'), (29, 'y')}, I={(42, 'x'), (21, 'y'), (29, 'y')}
24: 0 O={(42, 'x'), (21, 'y'), (29, 'y')}, I={(42, 'x'), (21, 'y'), (29, 'y')}
27: (y>0) O={(42, 'x'), (21, 'y'), (29, 'y')}, I={(42, 'x'), (21, 'y'), (29, 'y')}
39: <?> O={(42, 'x'), (21, 'y'), (29, 'y')}, I={(42, 'x'), (21, 'y'), (29, 'y')}
F
T
29: y O={(42, 'x'), (21, 'y'), (29, 'y')}, I={(42, 'x'), (21, 'y'), (29, 'y')}
35: 8 O={}, I={}
33: CALL g(y) O={}, I={(42, 'x'), (21, 'y'), (29, 'y')}
38: x=8 O={(42, 'x')}, I={}
32: RET g(y) O={(42, 'x')}, I={(42, 'x')}
42: x O={}, I={(42, 'x')}
14: END g(x) O={(42, 'x'), (51, 'x')}, I={}
48: RET g(x) O={(51, 'x')}, I={(51, 'x')}
51: x O={}, I={(51, 'x')}
54: (g(x)+x) O={}, I={}
2: END f(x,y,z) O={}, I={}
62: RET f(2,3) O={}, I={}
61: None O={}, I={}


49: CALL g(x) O={}, I={(51, 'x'), (21, 'y'), (29, 'y')}
13: START g(x) O={(29, 'y'), (21, 'y')}, I={(29, 'y'), (21, 'y')}
16: 7 O={(29, 'y'), (21, 'y')}, I={(29, 'y'), (21, 'y')}
19: x=7 O={(42, 'x'), (21, 'y'), (29, 'y')}, I={(29, 'y'), (21, 'y')}
21: y O={(42, 'x'), (21, 'y'), (29, 'y')}, I={(42, 'x'), (21, 'y'), (29, 'y')}
24: 0 O={(42, 'x'), (21, 'y'), (29, 'y')}, I={(42, 'x'), (21, 'y'), (29, 'y')}
27: (y>0) O={(42, 'x'), (21, 'y'), (29, 'y')}, I={(42, 'x'), (21, 'y'), (29, 'y')}
39: <?> O={(42, 'x'), (21, 'y'), (29, 'y')}, I={(42, 'x'), (21, 'y'), (29, 'y')}
F
T
29: y O={(42, 'x'), (21, 'y'), (29, 'y')}, I={(42, 'x'), (21, 'y'), (29, 'y')}
35: 8 O={}, I={}
33: CALL g(y) O={}, I={(42, 'x'), (21, 'y'), (29, 'y')}
38: x=8 O={(42, 'x')}, I={}
32: RET g(y) O={(42, 'x')}, I={(42, 'x')}
42: x O={}, I={(42, 'x')}
14: END g(x) O={(42, 'x'), (51, 'x')}, I={}
48: RET g(x) O={(51, 'x')}, I={(51, 'x')}