3. Interprozedurale Datenflussanalyse

<u>Ziel</u>: Datenflussanalyse für rekursive Programme

<u>Problem</u>: Berücksichtigung der Call-Return-Beziehung bei Prozeduraufrufen

<u>Idee</u>: Berechne <u>JOVP-Lösung</u> (<u>join-over-all-valid-paths</u>)

↳ Return zum richtigen Call-Block.

<u>Zwei Techniken</u>:

<u>Procedure-Summaries</u> (<u>funktionaler Ansatz</u>)

Berechne den Effekt $f_p : D \to D$ einer Prozedur p

<u>Call-Strings</u>

Führe den Stack (eine abstrahierte Version) als Datenflussinformation mit.

3.1 Rekursive Programme:

<u>Definition (Rekursives Programm)</u>:

Ein <u>rekursives Programm</u> ist definiert als Folge von Prozeduren:

$$prog ::= \underline{proc}\ [main()]^{entry}\ \underline{begin}\ c\ [end]^{exit}$$
$$\mid prog;\ \underline{proc}\ [p()]^{entry}\ \underline{begin}\ c\ [end]^{exit}$$

$$c ::= \text{wie bisher}$$
$$\mid [p()]^{call}_{return}$$

↳ Es wird angenommen, dass alle Prozeduren verschieden heißen

↳ Entry- und Exit-Blöcke garantieren, dass es genau einen Anfangs- und Endblock gibt.

↳ Blöcke $[p()]^{call}_{return}$ haben zwei Label.

<u>Lokale und globale Variablen</u>

↳ Prozeduren nutzen <u>lokale</u> Variablen

↳ Die Variablen in main() sind <u>global</u>,
d.h. auch innerhalb der Prozeduren sichtbar.

↳ Mit globalen Variablen lassen sich Parameter von Prozeduren und Rückgabewerte nachbilden.

Auch rekursive Programme werden als <u>Kontrollflussgraphen</u> dargestellt:

↳ Gegeben eine Prozedur p in prog, sei

$$G_p := (B_p, E_p, F_p)$$

der Kontrollflussgraph, der wie bisher konstruiert wird.

↳ Der Kontrollflussgraph von prog ist

$$G_{prog} := \Big(\biguplus_{p \in prog} B_p,\ \biguplus_{p \in prog} E_p,\ \biguplus_{p \in prog} F_p,\ IF\Big).$$

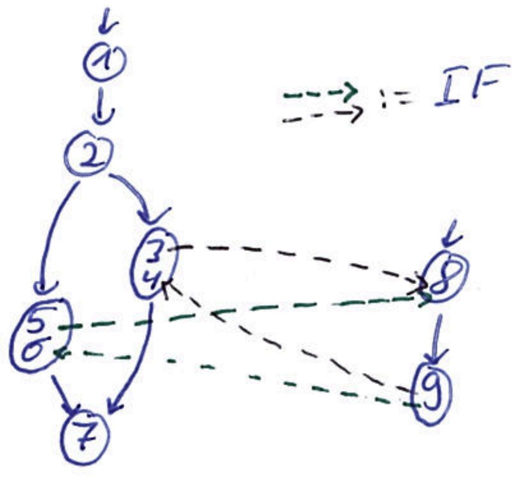
Dabei ist der <u>interprozedurale Flow</u> definiert als

$$IF := \{(call, entry, exit, return) \mid \text{eine Prozedur von prog enthält } [p()]^{call}_{return} \text{ mit } \underline{proc}\ [p()]^{entry}\ \underline{begin}\ c\ [end]^{exit}\}$$

<u>Beispiel:</u>

Warum wird IF als 4-Tupel gehalten?

Kontrollflussgraph



$\underline{proc}\ [main()]^1\ \underline{begin}$

$\quad \underline{if}\ [(*)]^2\ \underline{then}$

$\quad\quad [p()]^3_4$

$\quad \underline{else}$

$\quad\quad [p()]^5_6$

$\quad \underline{endif}$

$[end]^7$

$\underline{proc}\ [p()]^8\ \underline{begin}$

$[end]^9$

Nicht alle Pfade durch den Kontrollflussgraphen entsprechen auch gültigen Ausführungen des rekursiven Programms:

↳ 1 2 3 8 9 4 ist ein gültiger Pfad

↳ 1 2 3 8 9 6 ist kein gültiger Pfad

↳ Ähnlich ungültige Pfade treten bei Rücksprüngen aus geschachtelten Rekursionen auf:

<u>Definition (Gültige Pfade)</u>:

Sei $G = (B, E, F, IF)$ ein Kontrollflussgraph.

Dann ist die <u>Menge der gültigen Pfade von l_1 zu l_2</u>, <u>validpaths(l_1, l_2)</u>, definiert durch die kontextfreie Grammatik

$$\Big(\underbrace{\{ N_{l,l'} \mid l, l' \in Labels \}}_{\text{nicht-Terminale}},\ \underbrace{Labels}_{\text{Terminale}},\ P,\ \underbrace{N_{l_1,l_2}}_{\text{Startsymbol}} \Big)$$

mit Produktionen

$$N_{l,l} \rightarrow l$$

$$N_{l,l''} \rightarrow l \cdot N_{l',l''} \quad \text{mit } (l,l') \in F$$

$$N_{call,l} \rightarrow call \cdot N_{entry,exit} \cdot N_{return,l} \quad \text{mit}$$

$$(call, entry, exit, return) \in IF.$$

Wir schreiben

$$validpaths^{-}(l_1, l_2) := \bigcup_{(l,l_2) \in F} validpaths(l_1, l)$$

für die <u>gültigen Pfade von l_1 bis zu (und ohne) l_2</u>.

Definition (JOVP-Lösung)

Sei $S = (G, (D, \leq), i, f)$ ein (rekursives) Datenflusssystem.

Die join-over-all-valid-paths (JOVP)-Lösung ist

$$JOVP(S) := (X_1^{JOVP}, \ldots, X_{|B|}^{JOVP}) \in D^{|B|}$$

mit

$$X_b^{JOVP} := \bigsqcup \{ f_\pi(i) \mid \pi \in validpaths^-(entry(main), b) \}$$

Korollar:

(1) $JOVP(S) \leq \underbrace{JOP(S)}$

Hier werden alle Pfade beachtet, Call-Return-Beziehungen werden nicht berücksichtigt.

(2) $JOVP(S)$ ist nicht berechenbar, da die intraprozedurale Analyse ein Spezialfall ist.

3.2 Der funktionale Ansatz:

Ziel: Fixpunktberechnung, die ungültige Pfade vermeidet.

Idee:
- Berechne Transferverhalten von Prozeduren (Procedure-Summary)
- Vermeide dann Analyse innerhalb von Prozedurrümpfen

Genauer:
- Jeder gewöhnliche Block b (z.B. $b = [x := a]^\ell$) hat eine Transferfunktion

$$f_b : D \to D,$$

die die Datenflussinformation ändert.
- Angenommen für Prozedur p hätten wir eine Transferfunktion

$$f_p : D \to D,$$

die das Verhalten von p zusammenfasst.

· Dann ließe sich ein Block

$$b = [p()]^{call}_{return}$$

durch die Transf.funktion

$$f_b(X) = f_{return}(X, f_p(f_{call}(X)))$$

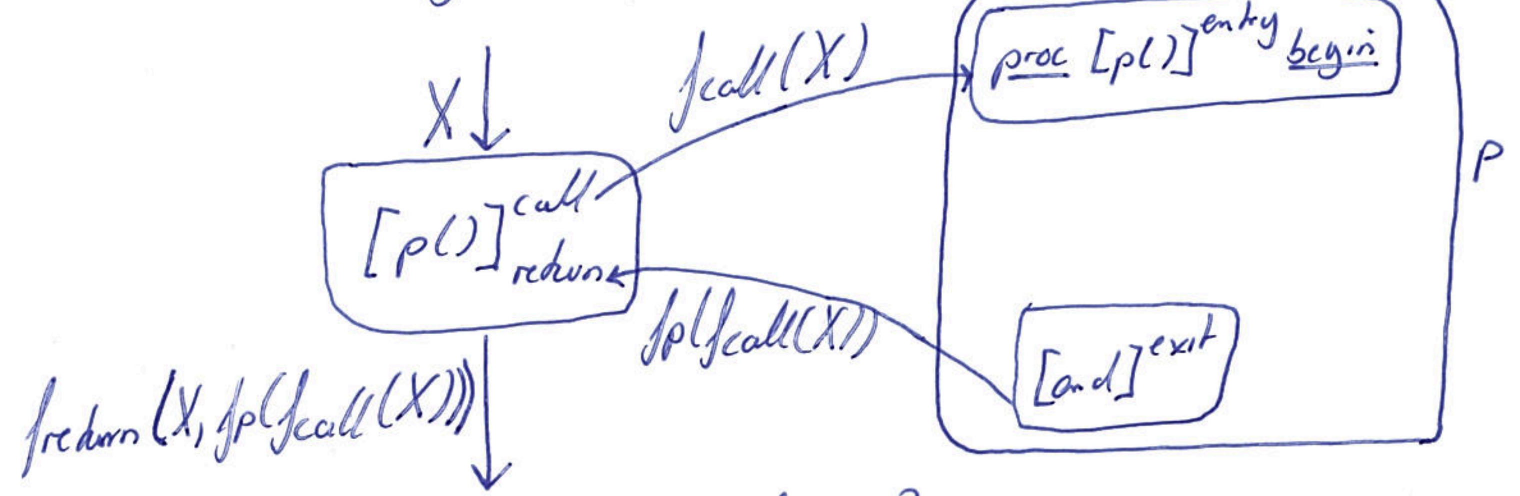
darstellen.

<u>$f_{call}: D \to D$:</u> (Gegeben als Teil des Datenflusssystems)

· Initialisiere Datenflusswert bei Eintritt in die Prozedur,
· abhängig vom aktuellen Datenflusswert.

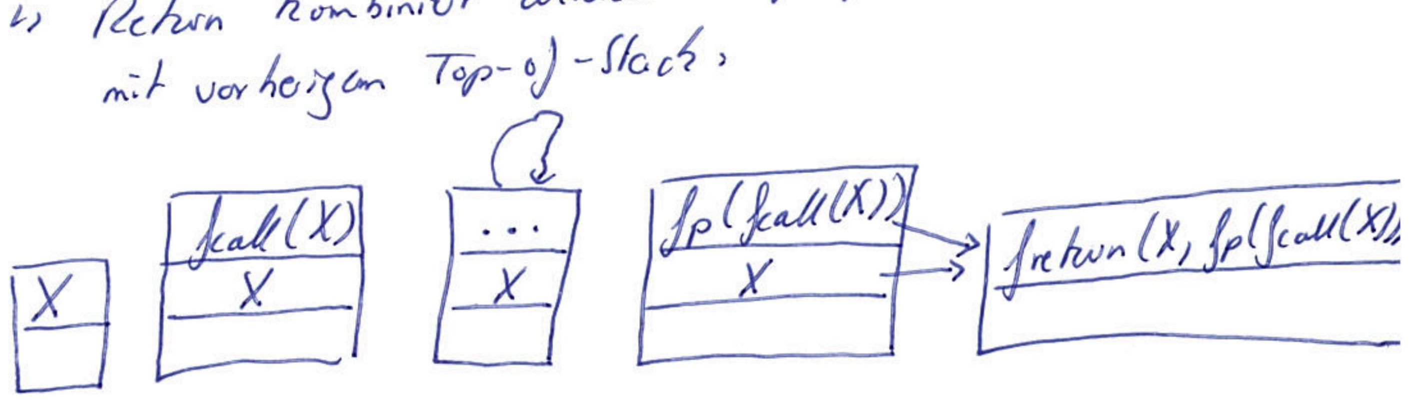
<u>$f_{return}: D \times D \to D$</u> (Gegeben als Teil des Datenflusssystems)

· Kombiniere Datenflusswert am Ende der Prozedur (2. Parameter)
· mit Datenflusswert bei Prozedureintritt.



<u>Warum ist X Parameter von f_{return}?</u>

↳ Call berechnet neuen Activation-Record (Top-of-Stack)
↳ Übliche Operationen ändern nur den Top-of-Stack
↳ Return kombiniert aktuellen Top-of-Stack
mit vorherigem Top-of-Stack.



Problem: · f_p ist nicht vor der Analyse bekannt.

· Bestimme f_p so, dass

$$f_p \geq f_\pi \quad \text{für alle } \pi \in validpaths(entry(p), exit(p))$$

Ansatz: Fixpunktberechnung auf dem vollständigen Verband der monotonen Funktionen in $D \to D$:

$$(MonFun(D \to D), \leq)$$

mit

$$MonFun(D \to D) := \{ f : D \to D \mid f \text{ ist monoton} \}$$

$$f \leq g, \text{ falls } f(d) \leq g(d) \text{ für alle } d \in D.$$

Definition (Summary-Gleichungssystem):

Sei $S = (G, (D, \leq), \iota, f)$ ein rekursives Datenflussystem.

Dann induziert S das Summary-Gleichungssystem

$$Y_{entry} = id$$

$$Y_b = \bigsqcup \{ f_{b'} \circ Y_{b'} \mid (b', b) \in F \}$$

mit

$$f_b := callret(Y_q), \text{ falls } b = [q()]^{call}_{return}$$

$$f_b := \text{wie in } f \text{ angegeben, sonst.}$$

Dabei ist

$$callret(Y_q) : D \to D \text{ mit}$$

$$callret(Y_q)(d) = f_{return}(d, Y_q(f_{call}(d)))$$

$$Y_p = f_{exit} \circ Y_{exit}$$

Man könnte eine Variable Y_b auch mit $Y_{entry,b}$ bezeichnen, um anzudeuten, dass sie den Effekt der Prozedur von entry bis zu b fasst.

Satz:

Sei $\psi_1, \ldots, \psi_{|D|}$ die kleinste Lösung des Summary-Gleichungssystems

(a) Sei b ein Block der Prozedur p.

Dann gilt

$$\psi_b \geq f_\pi \quad \text{für alle } \pi \in \text{validpaths}^-(\text{entry}(p), b).$$

(b) $\psi_p \geq f_\pi$ für alle $\pi \in \text{validpaths}(\text{entry}(p), \text{exit}(p))$.

Problem:

Monotone Funktionen in $D \to D$ müssen effektiv dargestellt werden.

↳ Falls D endlich, nutze Wertetabelle

↳ Falls D unendlich, problematisch.