CISPA Center for IT-Security, Privacy and Accountability

SAARLAND UNIVERSITY COMPUTER SCIENCE

Static Program Analysis

Reaching Definitions

Winter Term 2014/15

Advanced Lecture (9 CP)

Christian Hammer

Scale

```
while(TRUE) {
  if ((p_ab[CTRL2] & 0x10)==0) {
    u = ((p_ab[PB] & 0x0f) << 8) + p_ab[PA];
    u_kg = u * kal_kg;}
  if ((p_cd[CTRL2] & 0x01) != 0) {
    for (idx=0;idx<7;idx++) {
      e_puf[idx] = p_cd[PA];
      if ((p_cd[CTRL2] & 0x10) != 0) {
        switch(e_puf[idx]) {
          case '+': kal_kg *= 1.1; break;
          case '-': kal_kg *= 0.9; break; } } }
      e_puf[idx] = '\0'; }
  printf("Artikel: %07.7s\n",e_puf);
  printf(" %6.2f kg ",u_kg);
}
```

Scale

```
while(TRUE) {
  if ((p_ab[CTRL2] & 0x10)==0) {
    u = ((p_ab[PB] & 0x0f) << 8) + p_ab[PA];
    u_kg = u * kal_kg;}
  if ((p_cd[CTRL2] & 0x01) != 0) {
    for (idx=0;idx<7;idx++) {
      e_puf[idx] = p_cd[PA];
      if ((p_cd[CTRL2] & 0x10) != 0) {
        switch(e_puf[idx]) {
          case '+': kal_kg *= 1.1; break;
          case '-': kal_kg *= 0.9; break; } } }
      e_puf[idx] = '\0'; }
  printf("Artikel: %07.7s\n",e_puf);
  printf(" %6.2f kg ",u_kg);
}
```

Calibration factor can be changed!

Dataflow Analysis

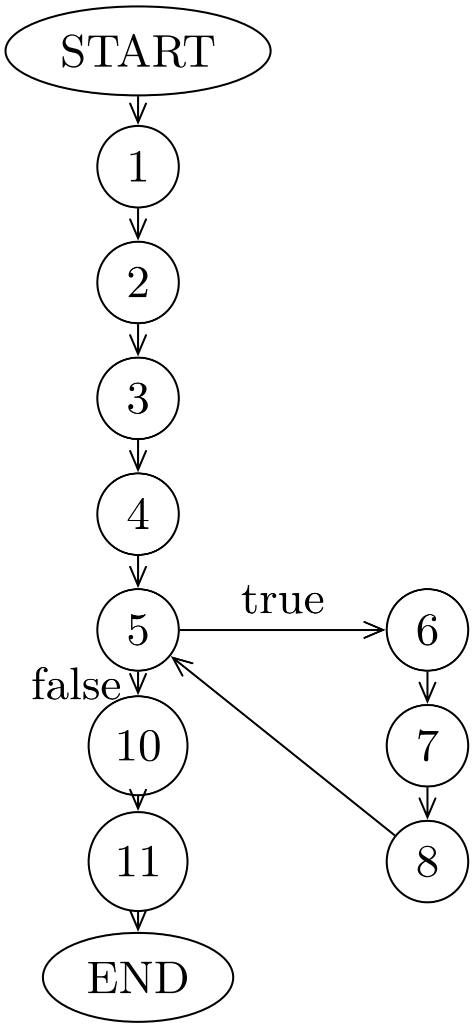
- Can a value computed at a certain statement flow to another given statement?
- usually represents program as a directed graph
- nodes are statements/predicates
- edges describe the control flow
- allows analysis of programs with
  - structured and
  - unstructured control flow
    (break, continue, try ... catch ... finally, goto)

Intraprocedural Control Flow Graph

- Control Flow Graph (CFG) $G = (N, E, n_s, n_e, \nu)$
- Set of nodes N (statements and predicates)
- Distinguished nodes n_s and n_e
- Set of control flow edges E, $(n, m) \in E$
  $n \rightarrow_{cf} m$ iff m may execute directly after n
- Total attribute function $\nu : E \rightarrow \{\text{true}, \text{false}, \varepsilon\} \cup \mathbb{Z}$
  condition under which control flows along an edge
- Functions *succ* and *pred* that map the successors and predecessors to each node.
- Reachability is undecidable, so conservatively assumed

Intraprocedural Control Flow Graph

```
(1)  read(n);
(2)  i = 1;
(3)  sum = 0;
(4)  prod = 1;
(5)  while (i <= n) {
(6)    sum = sum + i;
(7)    prod = prod * i;
(8)    i++;
(9)  }
(10) write(sum);
(11) write(prod);
```



Monotone Dataflow Analysis Framework

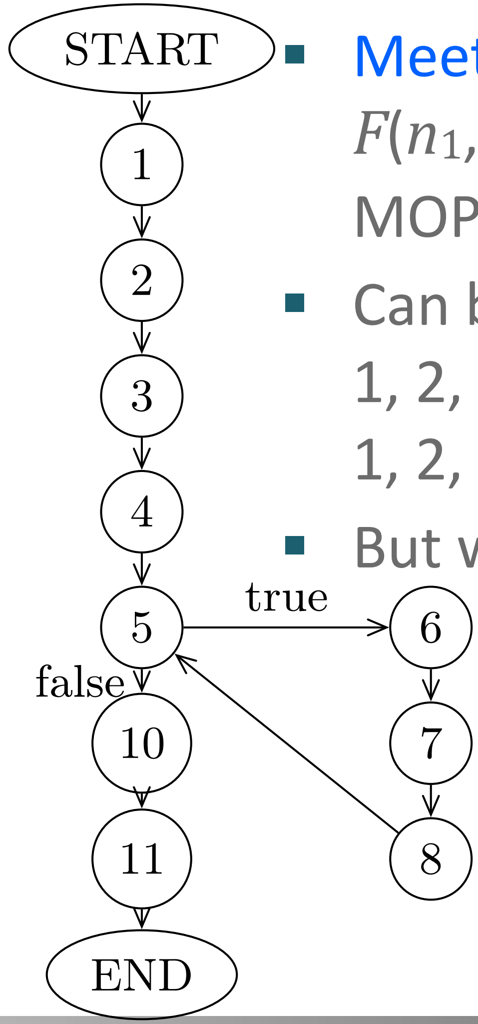
- most important class of dataflow analysis
- compute the most precise solution under certain conditions
- only takes a finite number of steps
  - even if loops have infinitely many paths
- Basic data structure: complete lattice $\mathscr{L} = (L, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$

Monotone Function Space

- Set of functions $\mathcal{F}$ on a meet semi-lattice $\mathscr{L} = (L, \sqsubseteq, \sqcap, \bot)$
- $\exists id_L \in \mathcal{F}: \forall x \in L: id_L(x) = x$ (identity function)
- $\forall F \in \mathcal{F}: \forall x, y \in L: x \sqsubseteq y \Rightarrow F(x) \sqsubseteq F(y)$ (monotonicity)
- $\forall F, G \in \mathcal{F}: F \circ G \in \mathcal{F}$ (closed under composition)
- $\forall F, G \in \mathcal{F}: F \sqcap G \in \mathcal{F}$ (pointwise infimum)

- Monotone Dataflow Analysis Framework
- Consists of a meet semi-lattice $\mathscr{L}$
- And a monotone function space $\mathcal{F}$
- distributive, if all functions are distributive over $\sqcap$:
  $\forall F \in \mathcal{F}: \forall x, y \in L: F(x \sqcap y) = F(x) \sqcap F(y)$

Computing Data Flow Analyses



- Meet-Over-All-Paths (MOP) solution desired
  $F(n_1, \ldots n_k): L \to L : F(n_1, \ldots n_k)(x) = (F{n_k} \circ F(n_1, \ldots n_{k-1}))(x)$
  MOP is $\sqcap_{\text{Path } P_i} F_{P_i}$
- Can be computed for 1, 2, 3, 4, 5, 10, 11;
  1, 2, 3, 4, 5, 6, 7, 8, 10, 11;
  1, 2, 3, 4, 5, 6, 7, 8, 6, 7, 8, 10, 11; etc.
- But when should we stop calculating?

Analysis time not linear with program size!

Fixed points

- Fixed point of an operator f on a semi-lattice L is an element $x \in L$ such that $f(x) = x$.
- Maximal fixed point (MFP) x if $\forall\, y \in L\colon f(y) = y \Rightarrow y \sqsubseteq x$.

CISPA Center for IT-Security, Privacy and Accountability

Computing the MFP

$$
\begin{array}{l}
\textbf{foreach } n \in CFG \textbf{ do} \\
\quad A[n] = \bot \\
\textbf{od} \\
\textbf{do} \\
\quad change = false \\
\quad \textbf{foreach } n \in CFG \textbf{ do} \\
\qquad temp = \bigsqcap_{q \in pred(n)} F_q(A[q]) \\
\qquad \textbf{if } temp \neq A[n] \\
\qquad\quad change = true \\
\qquad\quad A[n] = temp \\
\qquad \textbf{fi} \\
\quad \textbf{od} \\
\textbf{until } !change
\end{array}
$$

- $fix(s) := A[s]$ after loop (Kildall '77)
- Coincience Theorem (Kam, Ullman '77)

$$fix(s) = \bigsqcap_{\text{Path } P_i \text{ ending in } s} F_{P_i}(\bot)$$

if $\mathcal{F}$ is distributive

- If $\mathcal{F}$ is not distributive, the equality becomes $\sqsubseteq$, i.e., the fixed point is only a conservative approximation

Reaching definitions

- Reaching definitions is classical compiler problem
- Also prerequisite for computing dependence graphs
- Does a definition of a variable reach the use at a statement?
- Without being redefined underway?
- $\mathrm{Def}(n)$ set of variables defined at n
- $\mathrm{Use}(n)$ set of variables used at n
- Definition d, variable $v := \mathrm{var}(d)$ where $v \in \mathrm{Def}(n)$
- d reaches node n' if $\exists$Path $P = (n{=}n_0, \ldots, n_k{=}n')$ in G, $\mathrm{k} > 0$ $\forall i \in 1, \ldots, k - 1$: $v \notin \mathrm{Def}(n_i)$

Reaching definitions as a DMDFF

- $\mathscr{L} = (\mathcal{P}(D), \supseteq, \cup, \emptyset)$ (powerset of all definitions)
- Transfer functions derived from abstract semantics of variable assignment
- $F_n(X) = X \setminus \text{kill}(n) \cup \text{gen}(n)$
- Statement n defines variables in $\text{Def}(n)$, so all definitions in D defining the same variable can no longer be visible (killed)
  $$kill(n) := \bigcup_{v \in \text{Def}(n)} D_v, \text{ where } D_v := \{d \in D \mid v = var(d)\}$$
- All elements of $\text{Def}(n)$ are generated by n, i.e. visible after its execution
  $\text{gen}(n) := \text{Def}(n)$
- This MDFF is distributive (MOP and MFP coincide)

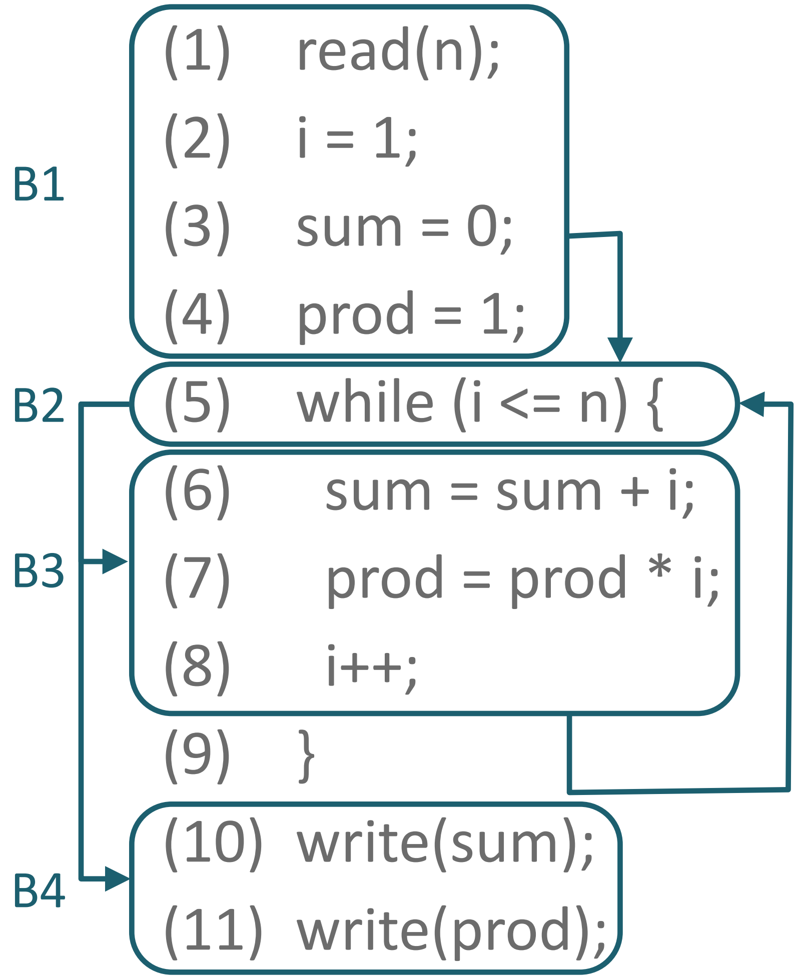
Basic Blocks

- Exactly one entry point
  no code within is the destination of a jump instruction
- Exactly one exit point
  only the last instruction can be a predicate

- Whenever the first instruction in a basic block is executed, the rest of the instructions are necessarily executed exactly once, in order

- Simplifies analysis when performed on basic blocks instead of statements

Example



- gen(B1) = {d1, d2, d3, d4}
- kill(B1) = {d6, d7, d8}
- use(B1) = ∅ //maybe {n}

- gen(B2) = ∅ = kill(B3);
- use(B2) = {i,n}

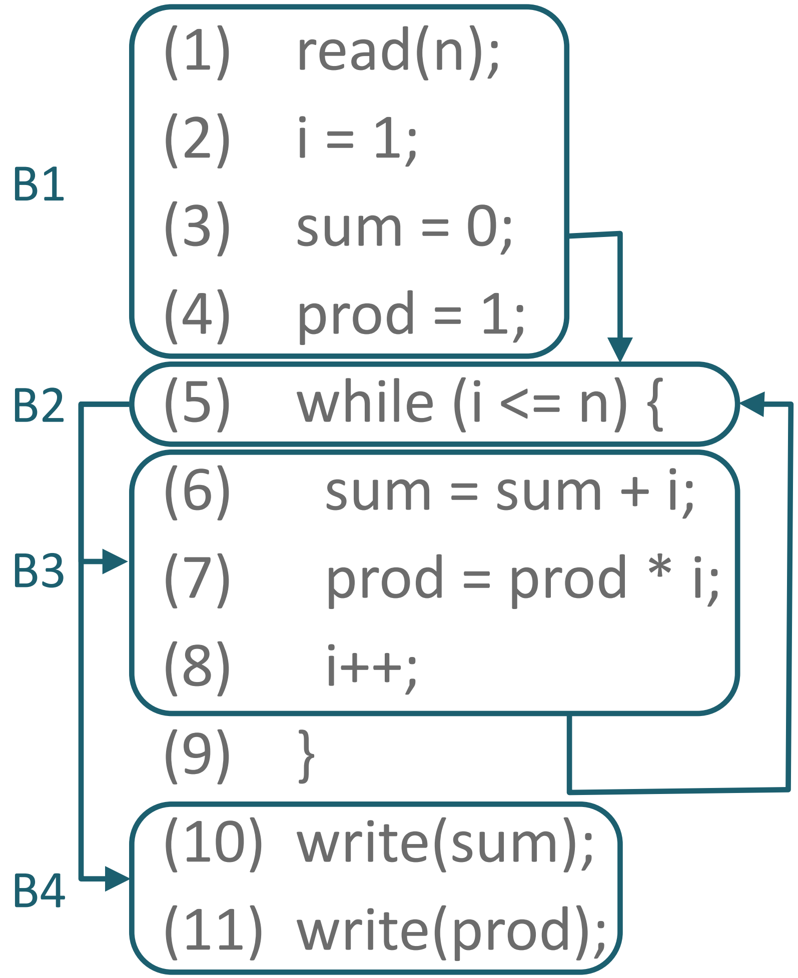
- gen(B3) = {d6, d7, d8}
- kill(B3) = {d2, d3, d4}
- use(B3) = {i, sum, prod}

- gen(B4) = ∅ = kill(B4)
- use(B4) = {sum, prod}

CISPA Center for IT-Security, Privacy and Accountability

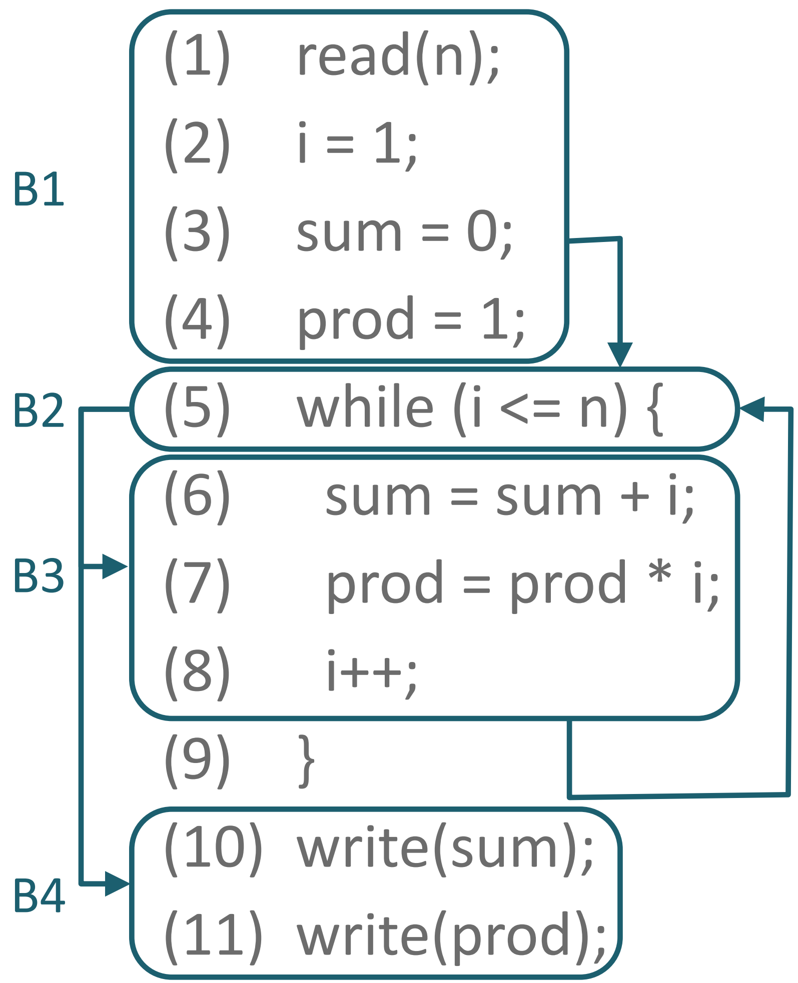
Example

in is called A in the algorithm



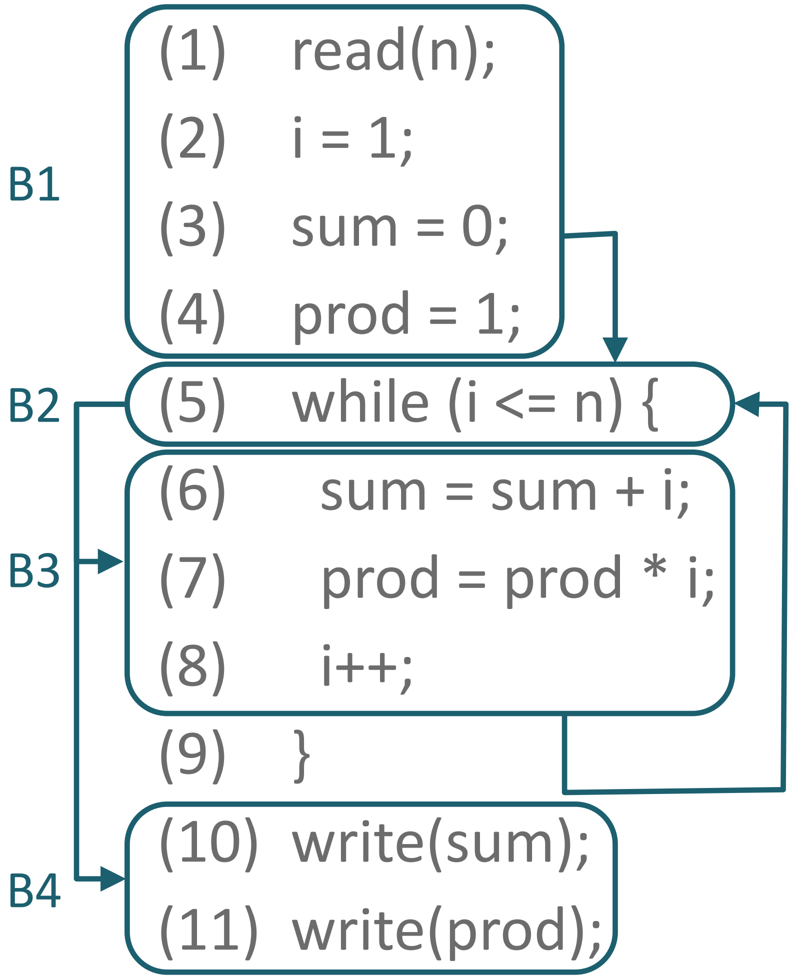
- Initialization: in = [∅, ∅, ∅, ∅]
- in(B1) = ∅, change = F
- in(B2) = out(B1) ∪ out(B3) = F_{B1}(in(B1)) ∪ F_{B3}(in(B3)) = (∅ \ {d6, d7, d8} ∪ {d1, d2, d3, d4}) ∪ (∅ \ {d2, d3, d4} ∪ {d6, d7, d8}) = {d1, d2, d3, d4, d6, d7, d8}
  change = T
- in(B3) = out(B2) = F_{B2}(in(B2)) = {d1, d2, d3, d4, d6, d7, d8} \ ∅ ∪ ∅ = {d1, d2, d3, d4, d6, d7, d8},
  change = T
- …

Example



- $in(B1) = \emptyset$
- $in(B2) = \{d1, d2, d3, d4, d6, d7, d8\}$
- $in(B3) = \{d1, d2, d3, d4, d6, d7, d8\}$
- $in(B4) = out(B2) = F_{B2}(in(B2)) =$
  $\{d1, d2, d3, d4, d6, d7, d8\} \setminus \emptyset \cup \emptyset$
  $= \{d1, d2, d3, d4, d6, d7, d8\}$
  change = T
- …

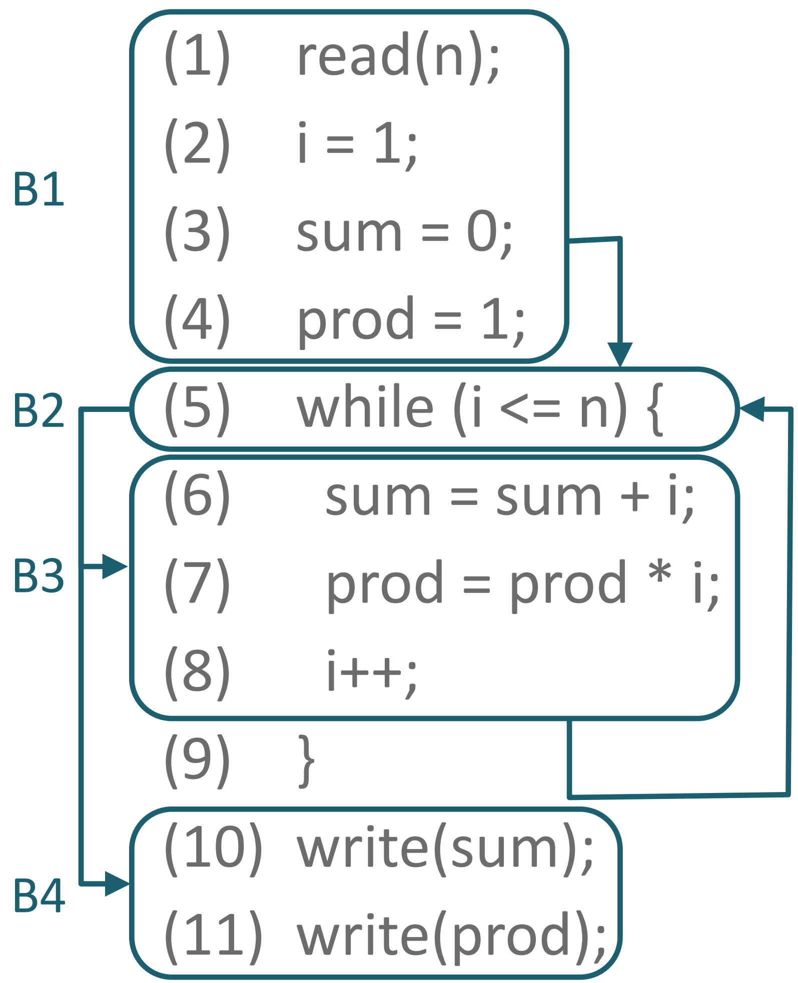
Example



- in(B1) = ∅, change = F
- in(B2) = out(B1) ∪ out(B2) = {d1, d2, d3, d4, d6, d7, d8}, change = F
- in(B3) = out(B2) = {d1, d2, d3, d4, d6, d7, d8}, change = F
- in(B4) = out(B2) = {d1, d2, d3, d4, d6, d7, d8}, change = F
- Algorithm terminates here

- need to use transfer functions in the block to determine reaching definitions for statements

Example



- in(B1) = ∅
- in(B2) = {d1, d2, d3, d4, d6, d7, d8}
- in(B3) = {d1, d2, d3, d4, d6, d7, d8}
- in(B4) = {d1, d2, d3, d4, d6, d7, d8}
- at beginning of B2 all definitions are visible, in particular both definitions of i, sum, and prod
- But:
  in(7) = F_6(in(B3)) =
  {d1, d2, d4, d6, d7, d8}