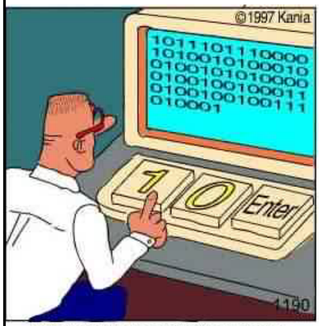
Software Visualization



Real programmers code in binary.

Lecture
WS 02/03

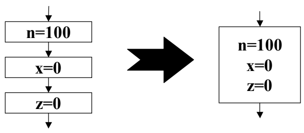
Visualizing the Results of Program Analyses

Static Program Analyses

- Control Flow Analysis
- Data Flow Analysis
- Visualizing Analysis Results

Control Flow Graphs

- Remark:
  - Often sequences of statements are combined into a single node called *basic block*.



Static Program Analysis

- Computes properties of a program, which hold for all executions of a program.
  - Not every property can be computed because of the Halting Problem
- Dynamic vs. Static properties
  - How often is a program point executed for a given input (dynamic)
  - Is a program point not executed for any input. (static)

Control Flow Analysis

- Computes the control flow graph of a program
  - Computation was very easy for the sample language we used before!
  - Problems for real programming languages:
    - Function/procedure calls → interprocedural CFG
    - Function pointers (in higher-order functional languages, but also in C)
    - Dynamic dispatch in object oriented languages like Java

Data Flow Analysis

- Computes information for each program point (node in the CFG) about the data that will reach this program point during execution.
- In general two kinds of information are computed for each node *v*:
  - IN(*v*): information about the state before the program point is executed.
  - OUT(*v*): information about the state after the program point is executed.
- Method: Locally available information is propagated over the paths in the control flow graph.

Data Flow Analysis

- Forward Flow Problem: what can happen before control reaches this program point, e.g.:
  - Reaching definitions
  - Available expressions
- Backward Flow Problem: what can happen after control leaves this program point, e.g.:
  - Live variables
  - Very busy expressions
  - Reached uses

Available Expressions

- A binary expression e_1 *op* e_2 is **available** at program point p if it has been computed before, i.e. along each path by which p can be reached.
- → program optimization: create temporary variable to avoid recomputation.
- Forward Flow Problem

Data Flow Analysis: Available Expressions

Given a control flow graph (V, E, in, out). Let $\mathcal{E}$ be the set of all binary expressions which occur in the program.

$GEN, KILL : V \to L_G(E)$

$$GEN(v) = \begin{cases} \{e' \mid e' \text{ is a binary subexpression of } e\} & \text{if } v = e \\ \{e' \mid e' \text{ is a binary subexpression of } e \text{ and } x \text{ does not occur in it}\} & \text{if } v = x{=}e \\ \emptyset & \text{otherwise} \end{cases}$$

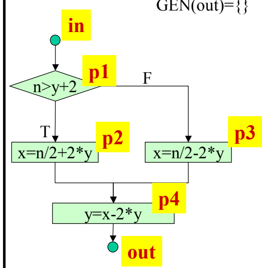
$$KILL(v) = \begin{cases} \{e' \mid e' \in \mathcal{E} \text{ and } x \text{ occurs in } e'\} & \text{if } v = x{=}e \\ \emptyset & \text{otherwise} \end{cases}$$

Algorithm:
$IN(in) = \emptyset$
$OUT(in) = \emptyset$
for all $v \in (V - \{in\})$ do
  $IN(v) = \mathcal{E}$
  $OUT(v) = (IN(v) - KILL(v)) \cup GEN(v)$
while there are changes do
  for all $v \in (V - \{in\})$ do
    $IN(v) = \bigcap_{(p,l,v) \in E} OUT(p)$
    $OUT(v) = (IN(v) - KILL(v)) \cup GEN(v)$

Example: Available Expressions

GEN(in)={}
GEN(p1)={y+2}
GEN(p2)={n/2, 2*y, n/2+2*y}
GEN(p3)={n/2, 2*y, n/2-2*y}
GEN(p4)={}
GEN(out)={}

KILL(in)={}
KILL(p1)={}
KILL(p2)={x-2*y}
KILL(p3)={x-2*y}
KILL(p4)={y+2, 2*y, n/2+2*y, n/2-2*y, x-2*y}
KILL(out)={}



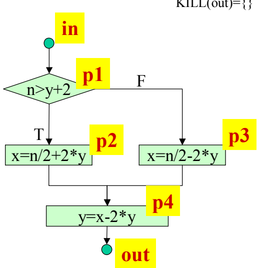
Initialization:
$\mathcal{E}$ ={y+2, n/2, 2*y, n/2+2*y, n/2-2*y, x-2*y}
IN(in)={} OUT(in)={}
IN(p1)= $\mathcal{E}$ OUT(p1)= $\mathcal{E}$
IN(p2)= $\mathcal{E}$ OUT(p2)={y+2, n/2, 2*y, n/2+2*y, n/2-2*y}
IN(p3)= $\mathcal{E}$ OUT(p3)={y+2, n/2, 2*y, n/2+2*y, n/2-2*y}
IN(p4)= $\mathcal{E}$ OUT(p4)={n/2}
IN(out)= $\mathcal{E}$ OUT(out)=$\mathcal{E}$

Example: Available Expressions

GEN(in)={}
GEN(p1)={y+2}
GEN(p2)={n/2, 2*y, n/2+2*y}
GEN(p3)={n/2, 2*y, n/2-2*y}
GEN(p4)={}
GEN(out)={}

KILL(in)={}
KILL(p1)={}
KILL(p2)={x-2*y}
KILL(p3)={x-2*y}
KILL(p4)={y+2, 2*y, n/2+2*y, n/2-2*y, x-2*y}
KILL(out)={}

Initialization:
$\mathcal{E}$ ={y+2, n/2, 2*y, n/2+2*y, n/2-2*y, x-2*y}
IN(in)={} OUT(in)={}
IN(p1)= $\mathcal{E}$ OUT(p1)= $\mathcal{E}$
IN(p2)= $\mathcal{E}$ OUT(p2)={y+2, n/2, 2*y, n/2+2*y, n/2-2*y}
IN(p3)= $\mathcal{E}$ OUT(p3)={y+2, n/2, 2*y, n/2+2*y, n/2-2*y}
IN(p4)= $\mathcal{E}$ OUT(p4)={n/2}
IN(out)= $\mathcal{E}$ OUT(out)=$\mathcal{E}$



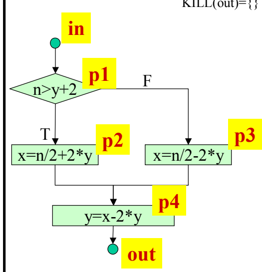
First Iteration:
IN(p1)= {} OUT(p1)={y+2}
IN(p2)= {y+2} OUT(p2)={y+2, n/2, 2*y, n/2+2*y}
IN(p3)= {y+2} OUT(p3)={y+2, n/2, 2*y, n/2-2*y}
IN(p4)= {y+2,n/2, 2*y }
OUT(p4)={n/2}
IN(out)= {n/2}
OUT(out)={n/2}

Example: Available Expressions

GEN(in)={}
GEN(p1)={y+2}
GEN(p2)={n/2, 2*y, n/2+2*y}
GEN(p3)={n/2, 2*y, n/2-2*y}
GEN(p4)={}
GEN(out)={}

KILL(in)={}
KILL(p1)={}
KILL(p2)={x-2*y}
KILL(p3)={x-2*y}
KILL(p4)={y+2, 2*y, n/2+2*y, n/2-2*y, x-2*y}
KILL(out)={}

First Iteration:
IN(p1)= {} OUT(p1)= {y+2}
IN(p2)= {y+2} OUT(p2)={y+2, n/2, 2*y, n/2+2*y}
IN(p3)= {y+2} OUT(p3)={y+2, n/2, 2*y, n/2-2*y}
IN(p4)= {y+2, n/2, 2*y }
OUT(p4)={n/2}
IN(out)= {n/2}
OUT(out)={n/2}



Second Iteration:
IN(p1)= {} OUT(p1)= {y+2}
IN(p2)= {y+2} OUT(p2)={y+2, n/2, 2*y, n/2+2*y}
IN(p3)= {y+2} OUT(p3)={y+2, n/2, 2*y, n/2-2*y}
IN(p4)= {y+2,n/2, 2*y }
OUT(p4)={n/2}
IN(out)= {n/2}
OUT(out)={n/2}

No Changes → Fixpoint

Live Variables

- A variable x is **live** at program point p if there is a path from p to p' and
  - x is used at p' (occurs in an expression)
  - There is no redefinition (assignment to x) of x along that path.
- Backward Flow Problem

Data Flow Analysis: Live Variables

$GEN, KILL : V \to L_G(V)$

$$KILL(v) = \begin{cases} \{x\} & \text{if } v = x{=}e \\ \emptyset & \text{otherwise} \end{cases}$$

In particular x' can be x.

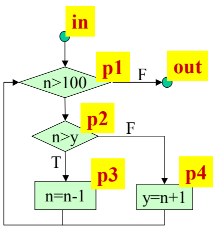
$$GEN(v) = \begin{cases} \{x' \mid x' \text{ occurs in } e\} & \text{if } v \in \{x{=}e, e\} \\ \emptyset & \text{otherwise} \end{cases}$$

Algorithm:

for all $v \in V$ do
  $IN(v) = GEN(v)$
while there are changes do
  for all $v \in V$ do
    $OUT(v) = \bigcup_{(v,l,s) \in E} IN(s)$
    $IN(v) = (OUT(v) - KILL(v)) \cup GEN(v)$

Example: Live Variables

GEN(in)={}	KILL(in)={}
GEN(p1)={n}	KILL(p1)={}
GEN(p2)={n,y}	KILL(p2)={}
GEN(p3)={n}	KILL(p3)={n}
GEN(p4)={n}	KILL(p4)={y}
GEN(out)={}	KILL(out)={}



Initialization:
IN=GEN

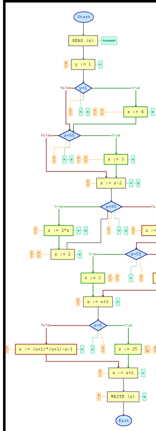
First Iteration:

OUT(in)={n}	IN(in)={n}
OUT(p1)={n,y}	IN(p1)={n,y}
OUT(p2)={n}	IN(p2)={n,y}
OUT(p3)={n,y}	IN(p3)={y,n}
OUT(p4)={n,y}	IN(p4)={n}
OUT(out)={}	IN(out)={}

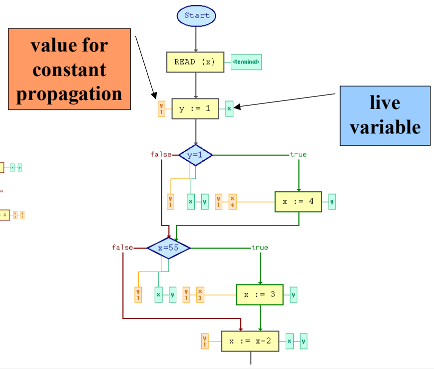
Second Iteration:

OUT(in)={n}	IN(in)={n}
OUT(p1)={n,y}	IN(p1)={n,y}
OUT(p2)={y,n}	IN(p2)={n,y}
OUT(p3)={n,y}	IN(p3)={y,n}
OUT(p4)={n,y}	IN(p4)={n}
OUT(out)={}	IN(out)={}

Third Iteration: No Changes → Fixpoint



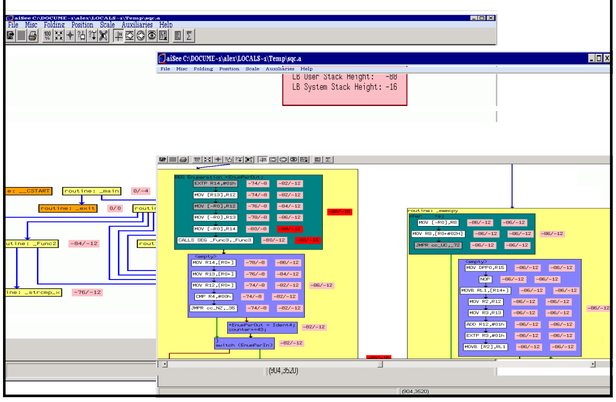
Live Variables with aiSee



Other Static Analyses

- Stack usage
- Types
- Worst Case Execution Times

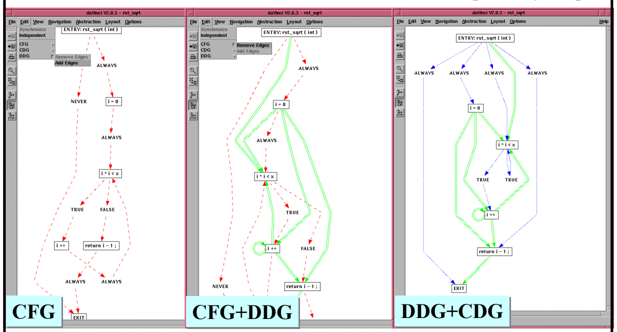
Stack Usage Visualization



VISTA

http://www.cigitallabs.com/research/demos/vista/

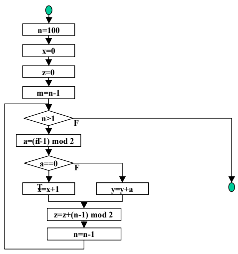
CFG: Control Flow Graph
DDG: Data Dependency Graph
CDG: Control Dependency Graph



No application-specific graph drawing algorithm!

- Control Dependence Graph:
  - statements are only dependent on their preceding condition or the entry node
  - similar to Jackson Diagrams

Assignments



- We will collect the assignments at the start of next weeks lecture.
- http://www.cs.uni-sb.de/~diehl/SoftVisVorles
  - User:
  - Password:

Example