Chapter 10:
Interprocedural Analysis and Optimization

by Dr. Ken Kennedy

April 18, 1995

Presented by

Vijay Pai

More complicated languages

More intricate programs

Greater opportunity for parallelism

- Systems
- Applications

Gather information about program

Find less conservative restrictions

Use opportunities for parallelism

Transform using interprocedural analysis

Typically modifies multiple procedures

Example: Inlining

Interprocedural Problems

Modification & Reference

Aliasing

Live variables

Constant Propagation

Kill Analysis

```
      COMMON X, Y
      ...
      DO 100 I = 1, N
S0      CALL F
50      X(I) = X(I) + Y(I)
100   CONTINUE
```

F modifies Y?

F modifies or references X?

```
     SUBROUTINE S(A,X,N)
       REAL A(*), X, Y
       COMMON Y
       DO 100 I=1,N
S0       X = X + Y*A(I)
100    CONTINUE
     END
```

Keep X and Y in registers

```
CALL S(A,Y,10)
```

```
   SUBROUTINE S(A,B,N,IS,I1)
     REAL A(*), B(*)
L    DO I I=0, N-1
S0     A(IS*I + I1) = A(IS*I + I1) + B(I+1)
     ENDDO
   END
```

`IS` ever zero?

If not, vectorize!

May vs. Must

- May: ALIAS, MOD, REF
- Must: CONST

Flow-sensitive vs. Flow-insensitive

- Sensitive: CONST
- Insensitive: ALIAS, MOD, REF

Side-effect vs. Propagation

- Side-effect: MOD, REF
- Propagation: ALIAS, CONST

Assumptions for Talk

No procedure nesting

Call by reference only

No pointers

Maximum number of parameters μ

Flow-insensitive side-effects

```
   SUBROUTINE P(...)
   ...
 S    CALL Q(...)
   ...
   END
```

$MOD(s)$

set of variables that *may* be modified

$REF(s)$

set of variables that *may* be referenced

Examples show MOD

Computing REF analogous

Alias definition

$ALIAS(p, x)$

set of variables that *may* be

aliased to x in procedure p

Needed for computing MOD

Treat it separately

Problem notation

$z \xrightarrow{s} w$

call site s, actual parameter z

bound to formal parameter w

$s = (p, q)$

call site s, in procedure p

calls procedure q

```
   SUBROUTINE P(A,B)    SUBROUTINE Q(C)
   ...                  ...
S    CALL Q(A)          END
   ...
   END
```

$$a \xrightarrow{s} c$$

MOD Problem components

$DMOD(s)$

- "direct modification set"
- ignores alias effects

$GMOD(p)$

- procedure "general modification set"
- includes side-effects of calls

$IMOD(p)$

- procedure "immediate modification set"
- modifications *explicitly* in p

Problem decomposition

$$
\begin{aligned}
MOD(s) &= DMOD(s) \cup \bigcup_{x \in DMOD(s)} ALIAS(p, x) \\
DMOD(s) &= \{z \mid s = (f, p), z \xrightarrow{s} w, w \in GMOD(p)\} \\
GMOD(p) &= IMOD(p) \cup \\
&\quad \bigcup_{s=(p,q)} \{z \mid z \xrightarrow{s} w, w \in GMOD(q)\}
\end{aligned}
$$

Split globals & formals

Start with IMOD and work up

Define new subproblems as needed

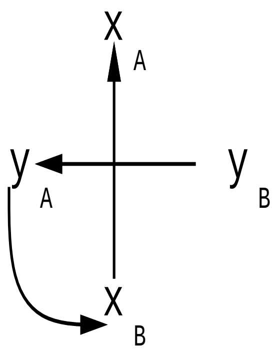
Binding graph

Vertex for each formal parameter

Edge (f_1, f_2)

- f_1 formal of p, f_2 formal of q
- $\exists s = (p, q)[f_1 \xrightarrow{s} f_2]$

```
A(x,y)
{
  B(y,z);
  z=0;
}
```

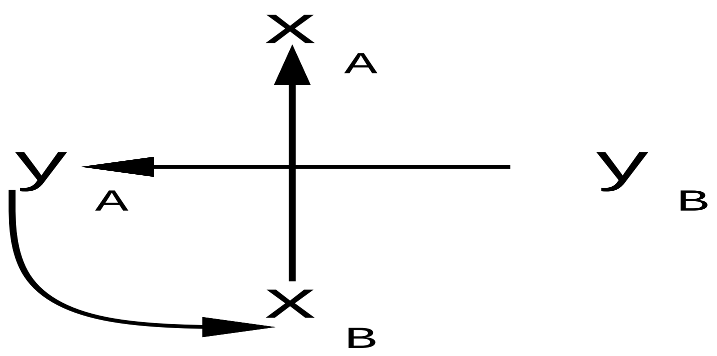



```
B(x,y)
{
  x += z;
  A(x,y);
}
```

$IMOD(a) = \{z\}$ $\quad$ $IMOD(b) = \{x\}$

$RMOD(p)$

formals of p modified either directly in p or through a call



Mark IMOD set true

Mark all vertices connected to it

IMOD+ subproblem

$RMOD$ plus $IMOD$

$IMOD(a) = \{z\}$ $\qquad$ $IMOD(b) = \{x\}$

$RMOD(a) = \{y\}$ $\qquad$ $RMOD(b) = \{x, y\}$

$y_a \xrightarrow{s_a} x_b \qquad z \xrightarrow{s_a} y_b \qquad x_b \xrightarrow{s_b} x_a \qquad y_b \xrightarrow{s_b} y_a$

$$\begin{aligned} IMOD^+(p) &= IMOD(p) \cup \\ & \bigcup_{s=(p,q)} \{z \mid z \xrightarrow{s} w, w \in RMOD(q)\} \\ GMOD(p) &= IMOD^+(p) \cup \\ & \bigcup_{s=(p,q)} \left[GMOD(q) \cap GLOB\right] \end{aligned}$$

$IMOD^+(a) = \{y, z\}$ $\qquad$ $IMOD^+(b) = \{x, y\}$

Use depth-first search on call graph

Initial value is $IMOD^+$

Non-back edges

$\forall$ edges $p \rightarrow q$

$GMOD(p) \leftarrow GMOD(p) \cup [GMOD(q) \cap GLOB]$

Strongly-connected

$\forall u$ in cycle headed by p

$GMOD(u) \leftarrow GMOD(u) \cup [GMOD(p) \cap GLOB]$

Computing MOD

$$
\begin{aligned}
GMOD(a) &= \{y, z\} \\
GMOD(b) &= \{x, y, z\}
\end{aligned}
$$

$$
\begin{aligned}
DMOD(s) &= \{z \mid s = (f, p), z \xrightarrow{s} w, w \in GMOD(p)\} \\
MOD(s) &= DMOD(s) \cup \bigcup_{x \in DMOD(s)} ALIAS(p, x)
\end{aligned}
$$

Naive algorithm

- Iterate over call sites
- Initially $MOD \leftarrow DMOD$
- $\forall x \in DMOD, MOD \leftarrow MOD \cup ALIAS(p, x)$
- Potentially $O(EV^2)$ time

Alias properties

2 globals never aliases

Globals only aliased to formals

\# of formals bounded by μ

$\forall$ call site s

$\quad t \leftarrow \emptyset$

$\quad \forall$ global $x \in DMOD(s)$ $\qquad O(V)$

$\quad\quad t \leftarrow t \cup \{x\}$

$\quad\quad t \leftarrow t \cup ALIAS(p, x)$ $\qquad O(\mu)$

$\quad MOD(s) \leftarrow MOD(s) \cup t$

$\quad \forall$ formal $f \in DMOD(s)$ $\qquad O(\mu)$

$\quad\quad MOD(s) \leftarrow MOD(s) \cup \{f\}$

$\quad\quad MOD(s) \leftarrow MOD(s) \cup ALIAS(p, f)$ $\qquad O(V)$

$O(V)$ per call site

$O(EV)$ total

Split up globals and formals

First approximation

- $A(f)$ for each formal f
- globals that may be aliased to f
- $g \rightarrow f_0 \rightarrow f_1 \rightarrow \ldots \rightarrow f_n = f$

```
A(x,y)                  B(x,y)
{                       {
  B(y,z);                 x += z;
  z=0;                    A(x,y);
}                       }
```

Literal Phase

$\forall$ call site s

$\quad\forall$ global g, $g \xrightarrow{s} f$

$\quad A(f) \leftarrow A(f) \cup \{g\}$

(e.g. $A(y_b) = \{z\}$)

```
A(x,y)
{
  B(y,z);
  ...
```

Binding Graph to DAG

- cycle $\rightarrow$ single node
- topologically sort it
- keep ancestors before children

$$y_B \longrightarrow y_A \longrightarrow x_B \longrightarrow x_A$$

A(f) ***z***

Transitive Phase

$\forall f$ in topological order

$$A(f) \leftarrow A(f) \cup \bigcup_{(f_0, f) \in E_b} A(f_0)$$

Cyclic Phase

$\forall$ cycle represented by c

$\quad \forall f \in c, A(f) \leftarrow A(c)$

For globals

$\forall$ formal f

$p =$ procedure of f

$\forall g \in A(f)$

$ALIAS(p, g) \leftarrow ALIAS(p, g) \cup \{f\}$

$$\begin{array}{lccccccc} & y_B & \longrightarrow & y_A & \longrightarrow & x_B & \longrightarrow & x_A \\ A(f) & z & & z & & z & & z \end{array}$$

$$\begin{aligned} ALIAS(a, z) &= \{x, y\} \\ ALIAS(b, z) &= \{x, y\} \end{aligned}$$

Alias introduction

- `CALL P(X,X)`
- Two actual parameters get same variable
- $x \xrightarrow{s} f_1, x \xrightarrow{s} f_2$

Find pairs of aliased formals

Introduction Phase

$\forall$ alias introduction $s = (p, q)$

$FPAIRS(q) \leftarrow FPAIRS(q) \cup \langle f_1, f_2 \rangle$

$worklist \leftarrow worklist \cup \langle f_1, f_2 \rangle$

Transitive Phase

$\forall \langle f_1, f_2 \rangle \in worklist$

$\forall s$ passing both f_1 and f_2

q is procedure invoked

$f_1 \xrightarrow{s} f_3, f_2 \xrightarrow{s} f_4$

if $\langle f_3, f_4 \rangle$ not $\in FPAIRS(q)$

$FPAIRS(q) \leftarrow FPAIRS(q) \cup \{\langle f_3, f_4 \rangle\}$

$worklist \leftarrow worklist \cup \langle f_3, f_4 \rangle$

Computing $ALIAS$ for formals

$\forall$ procedures p

Get $\langle f_1, f_2 \rangle$ from $FPAIRS(p)$

$ALIAS(p, f_1) \leftarrow ALIAS(p, f_1) \cup f_2$

$ALIAS(p, f_2) \leftarrow ALIAS(p, f_2) \cup f_1$

Repeat until $FPAIRS(p)$ empty

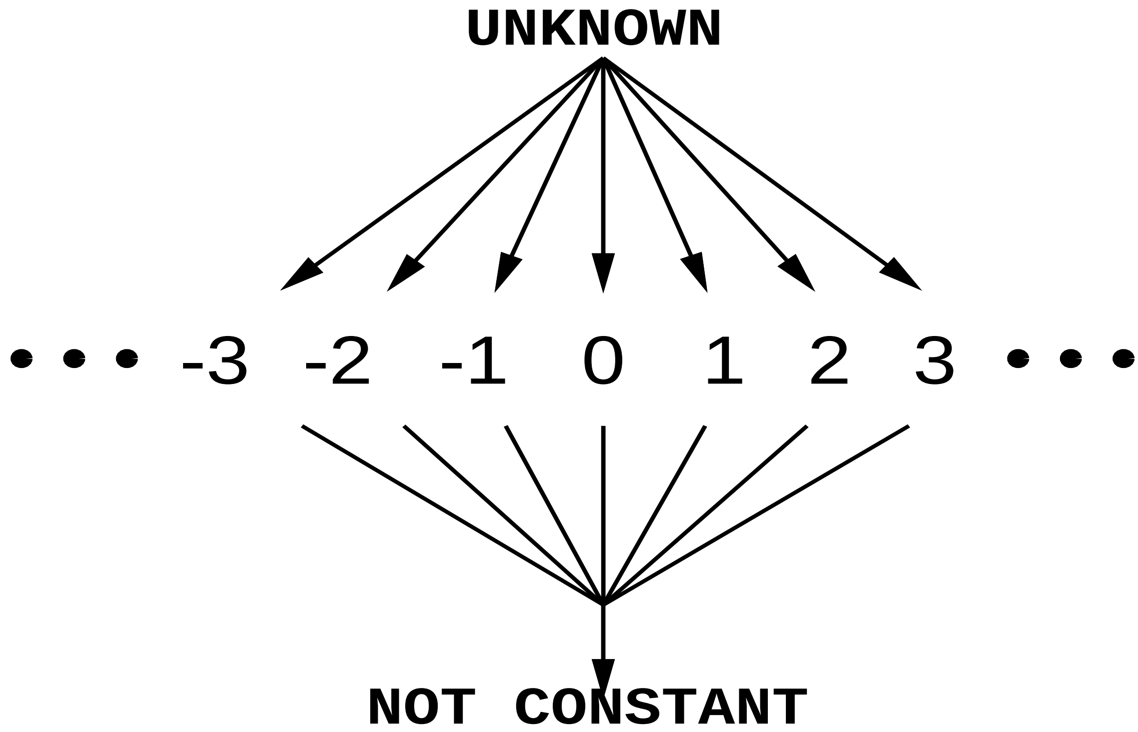
```
SUBROUTINE PHASE(N)
  INTEGER N, A, B
  CALL INIT(A,B,N)
  CALL PROCESS(A,B)
END
```

```
SUBR INIT(A,B,N)
  INTEGER A, B, N
  A = N + 1
  B = (N * A) / 2
END
```

Is `PHASE` only called with 10?

If so, `PROCESS(11,55)`

Analogous to single procedure



Constant propagation lattice

J_s^x $\qquad s = (p, q)$

Determines value of x, a formal of q

Support of J_s^x

inputs used in evaluation of J_s^x

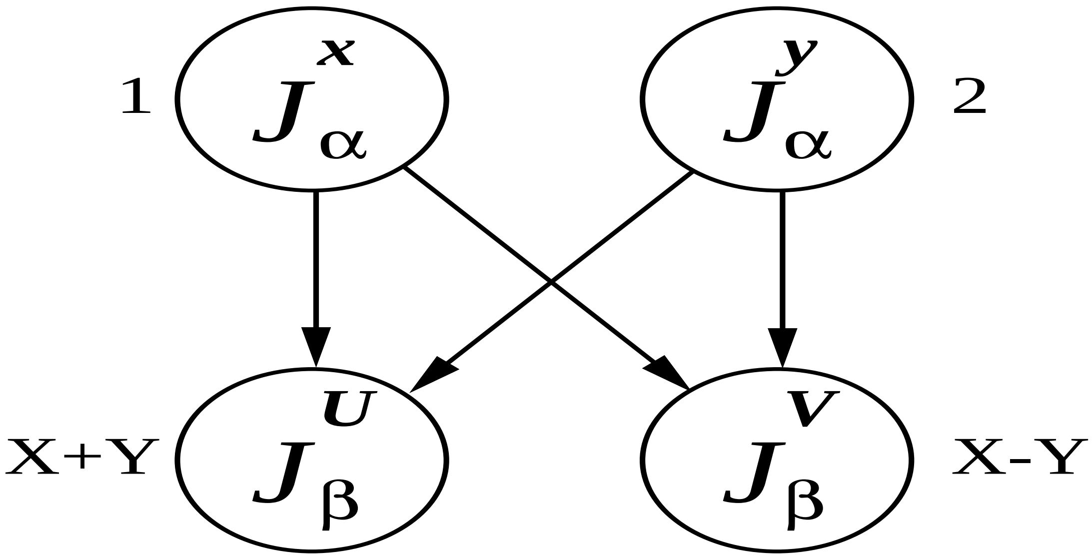
<pre>   PROGRAM MAIN
     INTEGER A, B
     A = 1
     B = 2
α   CALL S(A,B)
   END</pre>
	<pre> SUBROUTINE S(X,Y)
	     INTEGER X,Y,Z,W
	     Z = X + Y
	     W = X - Y
	β CALL T(Z,W)
	   END</pre>

$J_\alpha^x = \{1\}$	$J_\beta^U = \{X + Y\}$
$J_\alpha^y = \{2\}$	$J_\beta^V = \{X - Y\}$

Vertices are jump functions

Edge $J_s^x \rightarrow J_t^y$ if

- $s = (p, q) \wedge t = (q, f)$
- $x \in$ support (J_t^y)



Conservative

- Account for all control paths
- How much effort to put in?

What about function calls?

R_p^x "return jump function"

- Value of x on return from p
- Based on inputs of p
- x a formal of p

Return jump functions

```
  PROGRAM MAIN                   SUBR PROCESS(N,B)
    INTEGER A                      INTEGER N,B,I
α   CALL PROCESS(15,A)       β     CALL INIT(I,N)
    RETURN A                 γ     CALL SOLVE(B,I)
  END                            END

 SUBROUTINE INIT(X,Y)           SUBROUTINE SOLVE(C,T)
   INTEGER X, Y                   INTEGER C, T
   X = 2 * Y                      C = T * 10
 END                            END
```

$R^X_{INIT} = \{2 * Y\} \quad R^C_{SOLVE} = \{T * 10\} \quad J^N_\alpha = \{15\}$

$J^T_\gamma = \{ \text{ if } I \in MOD(\beta) \text{ then } R^X_{INIT}(N) \text{ else } undef\}$

$$R^B_{PROCESS} = \{ \text{ if } B \in MOD(\gamma) \\ \text{then } R^C_{SOLVE}(J^T_\gamma(N)) \text{ else } B\}$$

Compute MOD before jump functions

Single jump functions get most constants

Reverse jumps rarely useful, but valuable

Next session

Kill analysis

Call graphs

Live / use analysis

Further optimizations