CS6013 - Modern Compilers: Theory and Practise

Data flow analysis

V. Krishna Nandivada

IIT Madras

Opening remarks

What have we done so far?

- Compiler overview.
- Scanning and parsing.
- JavaCC, visitors and JTB
- Semantic Analysis - specification, execution, attribute grammars.
- Type checking, Intermediate Representation, Intermediate code generation.
- Control flow analysis, interval analysis, structural analysis
- Data flow anaalysis, intra-procedural constant propagation.

Announcement:

- Assignment 4: Ten days to go.

Today:

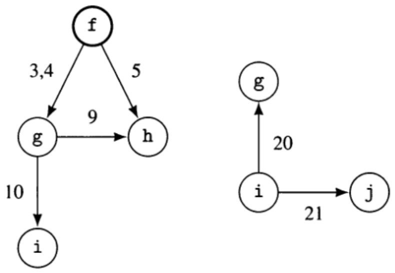
- Inter-procedural constant propagation.

Interprocedural CFA - Call graph

- Inter-procedural CFA constructs a static Call graph
  - A directed multigraph.
- Given a program P, consisting of procedures $p_1, p_2 \ldots p_n$, the call grpah $G = \langle N, S, E, r \rangle$
- N is the set of procedures.
- S is the set of call sites labels (e.g. line numbers in TAC).
- $E \subseteq N \times S \times N$: An edge from (p_1, s, n_2) indicates a call from p_1 to p_2 at site s.

Example call graph

```
1    procedure f( )
2    begin
3       call g( )
4       call g( )
5       call h( )
6    end    || f
7    procedure g( )
8    begin
9       call h( )
10      call i( )
11   end    || g
12   procedure h( )
13   begin
14   end    || h
15   procedure i( )
16      procedure j( )
17      begin
18      end    || j
19   begin
20      call g( )
21      call j( )
22   end    || i
```



Constructing the call graph

```
LabeledEdge = Procedure × integer × Procedure

procedure Build_Call_Graph(P,r,N,E,numinsts)
   P: in set of Procedure
   r: in Procedure
   N: out set of Procedure
   E: out set of LabeledEdge
   numinsts: in Procedure ⟶ integer
begin
   i: integer
   p, q: Procedure
   OldN := ∅: set of Procedure
   N := {r}
   E := ∅
   while OldN ≠ N do
      p := ♦(N - OldN)
      OldN := N
      for i := 1 to numinsts(p) do
         for each q ∈ callset(p,i) do
            N ∪= {q}
            E ∪= {⟨p,i,q⟩}
         od
      od
   od
end    || Build_Call_Graph
```


Challenges

- Separate compilation – we would not know the complete call graph; wait till the whole program is available.
- Function pointers.
- Overloaded functions and inheritance.

Read yourself.

Interprocedural constant propagation

Two flavors of inter-procedural constant propagation.

- Context insensitive (call site independent) constant propagation.
  - For each procedure in a program identify the subset of its parameters, such that each of the parameter will get a constant value, in every invocation.
  - The return value may be constant for every invocation or none.
- Context sensitive (call site dependent) constant propagation:
  - for each particular procedure called from each particular site, the subset of parameters that have the same constant value each time the procedure is called at that site.
  - For each call site, the return value may be constant or not.

Interprocedural constant propagation overview

Function constantProp()
begin
  worklist = {root};
  while worklist is not empty **do**
    p = $worklist.dequeue()$;
    foreach callsite s in p **do**
      compute the actuals of s using the formals of p;
      `// Intra-procedural constant propagation`
      Say the function being called at s is q;
      Compute the *meet* of the current values for the formals of q and the actuals at s;
      if constant values of q has changed **then**
        add q to the worklist;
    v = compute the meet of all the return values of p;
    Set the return value of p to v;
    foreach call function q that calls p **do**
      add q to the worklist
end

Initialization and modification to to CP algorithm

Initialization

- The return value of each function is initialized to $\top$.
- The constant value of each formal argument is initialized to $\top$.

Modification to the CP

- Constant value of a function call is given by the constant-return value of the function.
- If the statement is of the form $a = foo(\cdots)$, set the constant value of a to that of the function.

Definitions for a formal algorithm

- Jump function: $J(p,i,L,x)$
  - i - call site
  - p - caller procedure
  - L - formal arguments of caller
  - x - a formal parameter of the callee.
  - The jump function maps information about the actual arguments of the call at the call site i to x.
- Return-jump function: $R(p,L)$
  - p - procedure
  - L - formal parameters
  - Maps the return value of the function.
  - If the language admits call-by references: $R(p,L,x)$, where x - a formal parameter of the callee. Maps the value returned by the formal parameter x.

Algorithm

```
procedure Intpr_Const_Prop(P,r,Cval)
   P: in set of Procedure
   r: in Procedure
   Cval: out Var ⟶ ICP
begin
   WL := {r}: set of Procedure
   p, q: Procedure
   v: Var
   i, j: integer
   prev: ICP
   Pars: Procedure ⟶ set of Var
   ArgList: Procedure × integer × Procedure
      ⟶ sequence of (Var ∪ Const)
   Eval: Expr × ICP ⟶ ICP
   || construct sets of parameters and lists of arguments
   || and initialize Cval( ) for each parameter
   for each p ∈ P do
      Pars(p) := ∅
      for i := 1 to nparams(p) do
         Cval(param(p,i)) := ⊤
         Pars(p) ∪= {param(p,i)}
      od
      for i := 1 to numinsts(p) do
         for each q ∈ callset(p,i) do
            ArgList(p,i,q) := []
            for j := 1 to nparams(q) do
               ArgList(p,i,q) ⊕= [arg(p,i,j)]
            od
         od
      od
   od
```


Algorithm

```
   while WL ≠ ∅ do
      p := ♦WL; WL -= {p}
      for i := 1 to numinsts(p) do
         for each q ∈ callset(p,i) do
            for j := 1 to nparams(q) do
               || if q( )'s jth parameter can be evaluated using values that
               || are arguments of p( ), evaluate it and update its Cval( )
               if Jsupport(p,i,ArgList(p,i,q),param(q,j)) ⊆ Pars(p) then
                  prev := Cval(param(q,j))
                  Cval(param(q,j)) ⊓= Eval(J(p,i,
                     ArgList(p,i,q),param(q,j)),Cval)
                  if Cval(param(q,j)) ⊏ prev then
                     WL ∪= {q}
                  fi
               fi
            od
         od
      od
   od
end    || Intpr_Const_Prop
```

Discussion

- The function `J` can be thought of as
  1. a function that does all the computation required to compute the actual arguments to the callee in terms of the formal arguments of the caller. And `Eval` evaluates the return value of `J`.
  2. It is a simple function that just represents the argument text. And the `Eval` function does the actual constant propagation.
- the precision of the constant propagation will depend on the precision of `J` and `Eval`
  Examples (assuming scheme 1):
  - Literal constant: If the argument passed is a constant, then a constant, else $\perp$
  - Pass-through parameter: If a formal parameter is directly passed or a constant, then pass the constant value, else $\perp$
  - Constant if intra-procedural constant.
  - Do a full fledged analysis to determine its value.

Closing remarks

What have we done today?

- Call graphs.
- Inter-procedural constant propagation.

To read

- Muchnick - Ch 19.1, 19.3.

Next:

- Control tree based data flow analysis, du-chains.