Data Flow Analysis

CS2210
Lecture 14

Lattices

- **D = (S, ≤)**
  - S set of elements
  - ≤ induces a **partial order**
    - Reflexive, transitive & anti-symmetric
    - ∀ x,y∈S: meet ^*(x,y) (greatest lower bound)
    - join v(x,y) (least upper bound)
  - closure property
  - Unique Top (T) & Bottom (⊥) elements:
    - X^⊥ ⊥ = ⊥ and x v T = T
  - Meet and join are commutative and associative

- **Height of lattice**: longest path through partial order from top to bottom

Lattices in Data Flow Analysis

- Model information by elements of a lattice domain
  - Top = best case info
  - Bottom = worst case info
  - Initial info for optimistic analyses (at least back edges: top)
  - If a ≤ b then a is a conservative approximation of b
  - Merge function = meet (^*) : the most precise element that’s a conservative approximation of both input elements
  - Initial info for optimistic analyses (at least back edges: top)
Some Typical Lattice Domains

- Two point lattice: ⊥, T
  - Boolean property
  - A tuple of two point lattices = bit vector
- Lifted set: set of incomparable values and ⊥ and T
  - Example?
- Powerset lattice: set of all subsets of S, ordered somehow (often by ⊆)
  - T = {} or S, vice versa
  - Collecting analysis
    - Isomorphic to tuple of booleans indicating membership in subset of elements of S

Product (aka. Tuple) Lattices

- Often useful to break complex lattice into a tuple of lattices, one per variable analyzed
- \( D_T = <S_T, ≤_T> = <S, ≤>_N \)
  - \( S_T = S_1 \times S_2 \times ... \times S_N \)
  - \( ≤_T \) pointwise ordering
  - \( T_T = <T_1, ..., T_N> \), bottom tuple of bottoms
  - Height(\( D_T \)) = \( N \times \text{height}(D) \)
  - Example?

Analysis of Loops with Lattices

\[ F = \text{flow function for loop body} \]
\[ F(\text{info at loop head}) = \text{info at back edge:} \]
\[ F = d_{\text{dest}} \land \neg T \]
\[ F = d_{\text{dest}} \land \neg F(F(P)) = F(\text{info}) \]
\[ F = d_{\text{dest}} \land \neg F(F(P)) = F(\text{info}) \]
\[ F = d_{\text{dest}} \land \neg F(P(F(P))) = F(\text{info}) \]
Repeat until \( P < 1 \)
Termination of Iterative Analysis

- Sufficient conditions
  - Flow functions (F) are **monotonic**
    \[ d_1 \leq d_2 \implies F(d_1) \leq F(d_2) \]
  - Lattice is of finite height
    - Start at \( T \)
    - Each application of \( F \) goes down one level
    - Eventually hit fixed-point or bottom
      - At most \( \text{height}(D) - 1 \) applications

Examples

- Lattices for
  - Constant propagation
  - Live variables
  - Reaching definitions

Distributive Lattices

- A lattice is **distributive** if
  \[ \forall x, y, z \in D: \quad (x \land y) \lor z = (x \lor z) \land (y \lor z) \]
  and
  \[ (x \lor y) \land z = (x \land z) \lor (y \land z) \]
- Example:
  - Live variables, only elements \( T \) and \( \bot \) (easy exercise)
- Counterexample
  - Constant propagation:
    \[ (1 \lor 2) \land 3 = \]
    \[ (1 \land 3) \lor (2 \land 3) = \]
Meet-Over-All Paths Solution

Flow function for basic block $B$: $F_B$, for a path $p$ along
$B_1 \ldots B_n$: $F_p = F_{B_1} \ldots F_{B_n}$

$MOP(B) = \bigwedge_{p \in \text{Path}(B)} F_p(\text{Init})$

Init is initial info at entry block

$MOP$ computation is NP for monotone flow functions

I.e. there is no algorithm that is guaranteed to work for all
flow graphs

Use approximation: maximum fixed-point solution

$MFP$ is most precise solution we can hope for

$MOP$ computation is NP for monotone flow functions

I.e. there is no algorithm that is guaranteed to work for all
flow graphs

Important Results

- Monotone lattices
  - Iterative algorithm guaranteed to produce the MFP
    solution

- Distributive monotone lattices
  - $MFP = MOP$
  - Functions over lattices of bit vectors are
distributive, i.e., all functions $f: \text{BV}^n \rightarrow \text{BV}^n$ are
distributive

Important Data Flow Problems

- Reaching definitions
  - Which definitions of a
    variable $v$ reach a particular
    use of $v$

- Available expressions
  - What expressions are
    available at a particular
    program point (e.g., $x+y$ is
    available in variable $z$)

- Live variables
  - For a given program point,
    is there a use of the
    variable along some path to
    exit

- Upwards exposed uses
  - What uses of variables at
    particular points are
    reached by particular
    definitions

- Copy propagation
  - For $x := y$ to a use of $x$ no
    assignments to $y$?

- Constant propagation

- Partial redundancy
  elimination

- Original formulation forward
  & backward flow problem
Worklist Algorithm for IDFA

```
procedure Worklist_Iterate(N, Entry, FP, dfin, Init)
N: in set of Node
entry: in Node
FP: in Node -> L
dfin: out Node -> L
Init: in L
begin
B,P: Node
Worklist: set of Node
effect, totaleffect: L
dfin(entry) := Init
Worklist := N - {entry}
for each B in N do
dfin(B) := TOP
od
repeat
B := choose(Worklist)
Worklist -={B}
totaleffect := r
for each P in Pred(B) do
  effect := F(P, dfin(P))
totaleffect := MEET= effect
if dfin(B) != totaleffect then
dfin(B) := totaleffect
  Worklist U= {B}
fi
od
until Worklist = {}
end
```

Lattices of Flow Functions (1)

- Can define lattice of monotone flow functions over lattices:
  - L lattice, define \( L^2 \) set of monotone functions from \( L \rightarrow L \), i.e., \( f \in L^2 \Rightarrow \forall x,y \in L \ x \leq y \Rightarrow f(x) \leq f(y) \)
  - Meet defined as: \( \forall f,g \in L^2, \forall x \in L : (f \land g)(x) = f(x) \land g(x) \)
  - Top: \( \forall x \in L : T(x) = T \)
  - Bottom: \( \forall x \in L : \bot(x) = \bot \)

Lattices of Flow Functions (2)

- Identify function: \( id(x) = x, \forall x \in L \)
- Function composition: \((f \circ g)(x) = f(g(x)) \)
  - \( L^2 \) is closed under composition
  - \( F^f := id, F^{f \circ g} = F^f \circ F^g \) for \( n = 1 \)
- Kleene closure \( F^* \)
  - \( \forall x \in L: F^*(x) := lim_{n \to \infty} (id \circ f)^n(x) \)
  - \( L^2 \) is closed under Kleene closure (for finite height lattices)
  - Sufficient to have finite effective height (relative to function \( f \))
    - longest strictly descending chain of the form \( f(x), f(f(x)), f(f(f(x))), \ldots \)
Control-Tree-Based Data Flow Analysis

- Recall two approaches for control flow analysis
- Interval analysis
- Structural analysis
- Control-tree-based data flow analysis uses the intervals / control structures identified to perform data flow analysis

Structural Data Flow Analysis

Example: if-then

\[ F_{\text{if-then}} = (F_{\text{then}} \circ F_{\text{if/N}}) \wedge F_{\text{if/Y}} \]

- Propagate info to substructures:
  - \( \text{in} \text{(if)} = \text{in}(\text{if-then}) \)
  - \( \text{in} \text{(then)} = F_{\text{if/Y}} \text{(in}(\text{if})) \)

While Loops

\[ F_{\text{while}} = (F_{\text{body}} \circ F_{\text{while/N}})^* \]

\[ F_{\text{while-loop}} = F_{\text{while/N}} \wedge F_{\text{loop}} \]

- Propagate info to substructures:
  - \( \text{in} \text{(while)} = F_{\text{loop}} \text{(in}(\text{while-loop})) \)
  - \( \text{in} \text{(body)} = F_{\text{loop}} \text{(in}(\text{while})) \)
Structural Analysis for Backward Flow Problems

- Problem
  - Single-entry but multiple-exit control structures
    - Have multiple entries in backward flow
    - Have to meet (*) possible exits
      - Details in the book
  - Single-entry, single-exit structures
    - Can turn equations around

Static Single Assignment Form

Reading

- Rest of chapter 8
- Chapter 9
Variable Webs
- Web = maximal union of intersecting du-chains

---

Static Single Assignment Form
- THE standard IR in optimizing compilers
- Properties
  - Every use of a variable has at most one reaching definition
  - Separates values from the locations
  - Makes many optimizations more effective
    - Constant propagation, value numbering, invariant code motion & removal, strength reduction, etc.
- Mechanism:
  - Create new target names for definitions by subscripting variables
  - Introduce Φ-functions at merge points as pseudo-definitions
  - Adjust uses
  - Want to minimize # Φ-functions
  - Minimizes translation overhead for code generation

---

Dominance Frontier
- DF(x) = \{y | \exists z \in \text{pred}(y) \text{ so that } x \text{ dom } z \text{ and not } x \text{ sdom } y\}
- Set of all CFG nodes which for which x dominates a predecessor but not the node itself
- Direct computation is quadratic in number of CFG nodes
- Use recursive equations and solve iteratively
Dominance Frontier

\[ \text{DF}_{\text{local}}(x) = \{ y \in \text{succ}(x) \mid \text{idom}(y) \neq x \} \]

\[ \text{DF}_{\text{up}}(x, z) = \{ y \in \text{DF}(z) \mid \text{idom}(y) \neq x \} \]

\[ \text{DF}(x) = \text{DF}_{\text{local}}(x) \cup \{ z \in N \mid \text{idom}(z) = x \ \text{DF}_{\text{up}}(x, z) \} \]

Example

\[
\begin{align*}
\text{Dom}(5) &= \{5, 6, 7, 8\} \\
\text{DF}(5) &= \{4, 15, 12, 13\}
\end{align*}
\]

Dominance Frontier Algorithm

Procedure ComputeDF(n)
begin
S := {} 
for each node y in succ(n) do 
if idom(y) != n
\[ S \cup= \{y\} \]
end /*this loop computes DF_{local}(n)*/
for each child c of n in the dominator tree do 
compute DF[c] 
for each element w of DF[c] do 
if n does not dominate w or if n = w
\[ S \cup= \{w\} \]
end /*for each element w of DF[c]*/
end /*compute DF[n]*/
end

Algorithm is \(O(E + \text{sizeof}(DFs))\)

In practice linear in size of graph
Dominance Frontier Criterion

- Whenever node x contains a definition of some variable a, then any node in the dominance frontier of x needs a $\Phi$-function.
- Dominance frontier earliest points where definition is not guaranteed to be unique.
- Since $\Phi$-functions are definitions themselves have to iterate.

Iterated Dominance Frontier

- Define DF for set of nodes: $DF(S) = \cup_{x \in S} DF(x)$
- **Iterated Dominance Frontier:** $DF_i(S) = \lim_{i \to \infty} DF_i(S)$, where $DF_{i+1}(S) = DF_i(S \cup DF_i(S))$, and $DF_1(S) = DF(S)$.
- If S is set of nodes that assign to variable x (including the entry node) then $DF(S)$ is the set of nodes that need $\Phi$-functions for x.

Example

- On board