Data-flow analysis

Data-flow analysis

- *compile-time* reasoning about the *run-time* flow of values in the program
- represent facts about run-time behavior
- represent effect of executing each basic block
- propagate facts around control flow graph

Formulated as a set of simultaneous equations

- sets attached to the nodes and edges
- lattice to describe relation between values
- usually represented as bit or bit vector

Solve equations using iterative framework

- start with initial guess of facts at each node
- propagate until stabilizes at *maximal fixed point*
- desire *meet over all paths* solution (MOP)
Data-flow analysis

Limitations

1. *precision* — up to symbolic execution
   - no knowledge about control flow decisions
   - assume all paths are taken

2. *solution* — cannot afford MOP solution
   - class of problems where MOP = MFP
   - not all problems fit this category

3. *arrays* and *pointers* — difficult to analyze
   - precise techniques are *expensive*
   - imprecision rapidly adds up

Summary

*For scalar values, we can quickly compute solutions to simple problems*
Control-flow graph

Example flow graph

<table>
<thead>
<tr>
<th>Node Table</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Edges</td>
<td>Info</td>
</tr>
<tr>
<td>Name</td>
<td>In</td>
<td>Out</td>
</tr>
<tr>
<td>a</td>
<td>—</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>c</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Edge Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

Predecessors & successors have a natural (cheap) representation.
Available expressions

Definition

- An expression is *defined* at point $p$ if its value is computed at $p$.
- An expression is *killed* at a point $p$ if one of its argument variables is defined at $p$.
- An expression $e$ is *available* at a point $p$ in a procedure if every path leading to $p$ contains a prior definition of $e$ that is not killed between its definition and $p$.

Global common subexpression elimination

- If, at some definition point for $p \leftarrow e$, $e$ is available with name $x$, we can replace the evaluation with a reference to $x$.
- requires a global naming scheme
- natural analog to parts of value numbering
Available expressions

For a block $b$

- let $\text{AVAIL}(b)$ be the set of expressions available on entry to $b$.
- let $\text{KILL}(b)$ be the set of expressions killed in $b$.
- let $\text{GEN}(b)$ be the set of expressions defined in $b$ and not subsequently killed in $b$.

**Note:** $\text{GEN}(b)$ is $\text{AVAILTAB}$ from value numbering. $\text{KILL}(b)$ is harder to construct.

Now, AVAIL can be defined as:

$$\text{AVAIL}(b) = \bigcap_{x \in \text{pred}(b)} (\text{GEN}(x) \cup (\text{AVAIL}(x) \setminus \text{KILL}(x)))$$

**Note:** initializations must be conservative.
Available expressions example

\[
\text{AVAIL}(A) = \emptyset
\]
\[
\text{AVAIL}(B) = \text{GEN}(A) \cup (\text{AVAIL}(A) - \text{KILL}(A))
\]
\[
= \emptyset \cup (\emptyset - \{a+b\}) = \emptyset
\]
\[
\text{AVAIL}(C) = \text{GEN}(A) \cup (\text{AVAIL}(A) - \text{KILL}(A))
\]
\[
= \emptyset \cup (\emptyset - \{a+b\}) = \emptyset
\]
\[
\text{AVAIL}(D) = (\text{GEN}(B) \cup (\text{AVAIL}(B) - \text{KILL}(B))) \cap
\]
\[
(\text{GEN}(C) \cup (\text{AVAIL}(C) - \text{KILL}(C)))
\]
\[
= (\{a+b\} \cup (\emptyset - \emptyset)) \cap
\]
\[
(\{a+b\} \cup (\emptyset - \{a+b\}))
\]
\[
= \{a+b\}
\]
High-level view

Algorithm

1. build control flow graph \((cfg)\)
2. initial (local) data gathering
3. propagate information around the graph
4. post-processing \((if needed)\)

Example

<table>
<thead>
<tr>
<th>Facts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
</tr>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>c</td>
</tr>
</tbody>
</table>
Iterative data-flow framework

A worklist iterative algorithm

\[
\text{worklist} \leftarrow \text{the set of all nodes} \\
\text{while}(\ \text{worklist} \neq \emptyset) \\
\quad \text{pick a node } n \text{ from worklist} \\
\quad \text{remove } n \text{ from worklist} \\
\quad \text{recompute } \text{AVAIL}(n) \\
\quad \text{if } \text{AVAIL}(n) \text{ changed then} \\
\quad \quad \text{worklist} \leftarrow \text{worklist} \cup \text{successor}(n)
\]

Questions

- does this terminate?
- what answer does it compute?
- how fast (or slow) is it?


Data-flow lattices

Definitions

1. A *semilattice* is a set $L$ and a meet operation $\land$ such that, $\forall a, b, c \in L$

   (a) $a \land a = a$

   (b) $a \land b = b \land a$

   (c) $a \land (b \land c) = (a \land b) \land c$

2. $\land$ imposes an order on $L$, $\forall a, b \in L$

   (a) $a \geq b \iff a \land b = b$

   (b) $a > b \iff a \geq b \text{ and } a \neq b$

3. A lattice may have a *bottom* element, denoted $-$

   (a) $\forall a \in L, - \land a = -$

   (b) $\forall a \in L, a \geq -$ 

4. A lattice may have a *top* element, denoted $\top$

   (a) $\forall a \in L, \top \land a = a$

   (b) $\forall a \in L, \top \geq a$
Data-flow lattices

How does this relate to data-flow analysis?

- choose a semilattice $L$ to represent facts
- attach to each element of $L$ a meaning
  each $a \in L$ is a distinct set of known facts
- with each node $n$, associate a function
  $f_n : L \rightarrow L$ to model behavior of $n$

Example – AVAIL

- semilattice is $2^E$, where $E$ is the set of all
  expressions computed in the procedure, and $\wedge$ is $\cap$
  $-$ is $\emptyset$, $\top$ is $E$
- for a node $n$, $f_n$ has the form
  $f_n(x) = D_n \cup (x - N_n)$
  where $D_n = \text{GEN}_n$ and $N_n = \text{KILL}_n$
- the underlying graph is the flow graph
  $G = (N, E, n_0)$
  $n_0$ is the entry node
Data-flow analysis framework

Termination

- lattice of finite height
- node appears on worklist finite times
- iterative algorithm terminates

Correctness

- data-flow analysis results conservative
- iterative algorithm computes maximum fixed point
- equal to meet over paths for distributive frameworks

Speed

- propagate facts in reverse postorder
- define $d(G)$ as loop connectedness
  - maximum # of back edges in acyclic path in G
- stabilize in $d(G) + 3$ iterations
- in practice, $d(G')$ is less than 3     [Knuth]
Data-flow analysis

Representation

- represent a fact as a bit
- represent sets of facts as bit vectors
- meet function may be union or intersection
- operations on bit vectors

Key things to look for in a data-flow framework

- the domain and its size
- size of a single fact
- forward or backward problem
- model of characteristic function

Complexity

- distinguish bit-vector steps from logical steps
- watch out for complex mappings (GEN→KILL)