Dataflow Analysis 2

**Data-flow Analysis**

- **Algorithm**
  1. Find basic blocks, build CFG
  2. Find propagation functions for basic blocks
  3. Propagate information around CFG
  4. Propagate information into basic blocks

- **Example**

  ![Control flow graph (CFG)](image)

<table>
<thead>
<tr>
<th>Code</th>
<th>Control flow graph (CFG)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = 1 if (b) then c = a+b else b = 1 c = a+b</td>
<td></td>
</tr>
</tbody>
</table>

**Live Variables**

- **Universe of facts?**
  - All possible subsets of variables in the program

- **GEN and KILL sets for each basic block?**
  - GEN = variables used in basic block AND not killed before reaching beginning of basic block
  - KILL = variables defined in basic block

- **Initial values of LV(B) before propagation starts?**
  - All variables in the program
    - Assumes variable is live until proven otherwise
    - The conservative assumption

**Available Expressions**

- An expression e is **defined** if its value is computed.
- An expression e is **killed** if the values of any of its operands may have been changed.

  ![Available expressions](image)

<table>
<thead>
<tr>
<th>Basic Block</th>
<th>GEN</th>
<th>KILL</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

  \[
  AVAIL(B) = \bigcap_{B_i \in \text{PRED}(B)} (GEN(B_i) \cup [AVAIL(B_i) - KILL(B_i)])
  \]

  GEN(Bi): All definitions of expressions in Bi that are not subsequently killed in Bi

  KILL(Bi): All expressions with operand variables defined in Bi
Available Expressions

- Universe of facts?
  → All possible subsets of expressions in the program

- GEN and KILL sets for each basic block?
  → GEN = expressions defined in basic block AND operands not defined before reaching end of basic block
  → KILL = expressions whose operands are defined in basic block

- Initial values of AVAIL(B) before propagation starts?
  → ∅
  - Assumes expression is not available until proven otherwise
  - The conservative assumption

Available Expressions Example

- Control flow graph

- Local information

Very Busy Expressions

An expression e is defined if its value is used before any definition of its operands.

An expression e is killed if the values of any of its operands may have been changed before it is used

\[ VBE(B) = \bigcap_{B_i \in \text{SUCC}(B)} \left( \text{GEN}(B_i) \cup \left( VBE(B_i) - \text{KILL}(B_i) \right) \right) \]

GEN(Bi): All definitions of expressions in Bi that are used before they are killed in Bi

KILL(Bi): All expressions with operand variables defined in Bi

Very Busy Expressions Example

<table>
<thead>
<tr>
<th>Basic Block</th>
<th>GEN</th>
<th>KILL</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B2</td>
<td>a = b</td>
<td></td>
</tr>
<tr>
<td>B3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ VBE(B_4) = \bigcap_{B_i \in \text{SUCC}(B_4)} \left( \text{GEN}(B_i) \cup \left( VBE(B_i) - \text{KILL}(B_i) \right) \right) \]

Implementation Issues: Bit-vector Problems

The set of facts (universe of facts) can often be expressed as finite subsets of a finite base set. Such sets can be represented as bit-vectors.

Implementation Issues: Bit-vector Problems

The set of facts (universe of facts) can often be expressed as finite subsets of a finite base set. Such sets can be represented as bit-vectors.


\[
\begin{align*}
\text{RD} & \subseteq \{ B1:a, B2:c, B3:d \} \\
\text{B1:a, B2:c, B3:d} & = 1, 1, 1 \\
\text{B1:a, B2:c} & = 1, 1 \\
\text{B1:a, B3:d} & = 1, 0 \\
\text{B2:c, B3:d} & = 0, 1 \\
\text{B1:a} & = 1 \\
\text{B2:c} & = 0 \\
\text{B3:d} & = 0 \\
\text{B1:a, B2:c, B3:d} & = 1, 1, 0 \\
\text{B1:a, B2:c, B3:d, B3:d} & = 1, 1, 0, 1 \\
\text{B1:a, B2:c, B3:d, B2:c, B3:d} & = 1, 1, 0, 1, 0 \\
\end{align*}
\]

Implementation Issues: Bit-vector Problems

Meet operation is either bit-wise logical AND or bit-wise logical OR.

GEN and KILL sets can be expressed as single bit-vectors.

Bit-wise logical - is bit-wise negation followed by bit-wise AND.

Implementation steps:
1. - bit-vector construction/interpretation
2. - bit-vector CFG initialization (RD, GEN, and KILL vectors)
3. - bit-vector CFG propagation
4. - information post-processing (e.g.: DU/UD chains)

Bit-vector Problem?

\[
\begin{align*}
\text{B1:} & \quad i := a+b \\
\text{B2:} & \quad j := a+b \\
\text{B3:} & \quad k := a+b \\
\end{align*}
\]

- What does lattice look like?
- GEN and KILL functions?
- Confluence (meet) operator \( \wedge \)?
- Initial value AVAIL(Bi)
  
  \[ \text{AVAIL}(B_i) = \text{ENTIRE SET } (a+b) \]

Constant Propagation

A constant for a variable \( x \) is generated if it is assigned a constant value

A constant for variable \( x \) is killed if \( x \) is reassigned a different value, even if it is a constant value

\[
\text{CONST}(B) = \bigcap (\text{GEN}(B_i) \cup \text{KILL}(B_i) - \text{CONST}(B_i)) \\
B_i \in \text{PRED}(B)
\]

GEN(Bi) : Set of immediate constants (static) and dynamically encountered constants (dynamic)

KILL(Bi): All definitions of variables \( x \) defined in \( B_i \)

Very Busy Expressions

- Universe of facts?
  - All possible subsets of variable * constant in the program

- GEN and KILL sets for each basic block?
  - GEN = variable assigned constant value in basic block AND variable not redefined before reaching end of basic block
  - KILL = variable defined to non-constant value in basic block

- Initial values of \( \text{CONST}(B) \) before propagation starts?
  - \( \emptyset \)
    - Assumes variable is not constant until proven otherwise
    - The conservative assumption

Constant Propagation Example

\[
\begin{align*}
\text{Basic Block} & \quad \text{GEN} & \quad \text{KILL} \\
\text{B1} & \quad & \text{B1} \\
\text{B2} & \quad & \text{B2} \\
\text{B3} & \quad & \text{B3} \\
\text{B4} & \quad & \text{B4} \\
\end{align*}
\]

\[
\text{CONST}(B_4) = \bigcap (\text{GEN}(B) \cup (\text{CONST}(B) - \text{KILL}(B))) \\
B_i \in \text{PRED}(B)
\]
The set of facts (universe of facts) are set of pairs 
(variable, value) 
where value is from the following lattice:

```
1 2 3 . . .
```

Examples: (a, T), (b, 5)

Note: Typically, constant propagation is only performed 
on integral values, not floating point values

Non Bit-vector Problem

The confluence (meet) operator \( \land \) for the second component 
of a (variable, value) pair is defined as follows:

```
\| | \ T | \ e_2 | 1  \\
\| | \ T | \ T | 1  \\
\| | \ e_1 | \ e_2 | \ \ \ 1  \\
\| | \ 1 | \ 1 | 1  \\
```

Non Bit-vector Problem

Remarks:

• optimistic constant propagation
• constant propagation typically done on 
UD/DU chains, but similar principles apply
• pessimistic constant propagation does not find any constants in our example
• intermediate results are not safe in 
optimistic approach

Dead Code Elimination

How to use DU/UD chains for dead code elimination?

Termiology:

- UD(S, v): set of statements that contain definitions for variable 
v that reach the use of v in statement S
- DU(S, v): set of statements that contain used of variable v that 
are reached by the definition of v in statement S

Dead Code Elimination

Two flavors of dead code: A statement can be considered 
dead code if

1. it will never be executed, or
2. it may be executed, but the result will never be used 
   (including side effects)

The discussed algorithm uses UD chains.
Dead Code Elimination

Report all statements that are not on any path from the entry to exit node as dead code, and remove them.

Initialize all statement as not useful.

Initialize worklist with critical statements (print statements).

While worklist is not empty Do

Remove a statement from worklist, call it $S$; mark $S$ useful.

Mark all $S'$ in CONTROL($S$) useful and add them to worklist.

For all variables $v$ in USES($S$) add each $S'$ in UD($S,v$) to worklist EndForall

EndWhile.

Report all unmarked statements as dead code.