Dataflow Analysis

- Dataflow analysis
  - sets up system of equations
  - iteratively computes MFP
  - Terminates because transfer functions are monotonic and lattice has finite height

- Other possible solutions: FP, MOP, IDEAL
- All are safe solutions, but some are more precise:
  - $FP \subseteq MFP \subseteq MOP \subseteq IDEAL$
  - $MFP = MOP$ if distributive transfer functions
- MOP and IDEAL are intractable
- Compilers use dataflow analysis and MFP

Dataflow Analysis Instances

- Apply dataflow framework to several analysis problems:
  - Live variable analysis
  - Available expressions
  - Reaching definitions
  - Constant folding

- Discuss:
  - Implementation issues
  - Classification of dataflow analyses

Problem 1: Live Variables

- Compute live variables at each program point
- Live variable = variable whose value may be used later, in some execution of the program
- Dataflow information: sets of live variables
- Example: variables $(x, z)$ may be live at program point $p$

LV: Dataflow Equations

- Equations:
  $in[n] = (out[n] = def[n]) \cup use[n]$, for all $n$
  $out[n] = \delta[n] \cup in[n]' \cap \cup\text{succ}(n)$, for all $n$
  $out[n] = \emptyset$

- Where:
  - $def[n] = \text{set of variables defined (written) by } n$
  - $use[n] = \text{set of variables used (read) by } n$

- Meaning of transfer function:
  "A variable is live before a node if the node uses it, or if the variable is live after and the node doesn’t define it”

- Meaning of union operator:
  "A variable is live at the end of node $n$ if it is live at the beginning of one of its successor nodes"

LV: Monotonicity

- Are transfer functions $F[X] = (X - def[n]) \cup use[n]$ monotonic?

- Observation: $def[n]$ and $use[n]$ are constant: they do not depend on the current dataflow information $X$.

- Because $def[n]$ is constant, $G(X) = X - def[n]$ is monotonic:
  - $X1 \subseteq X2$ implies $X1 - def[n] \supseteq X2 - def[n]$

- Because $use[n]$ is constant, $H(Y) = Y \cup use[n]$ is monotonic:
  - $Y1 \subseteq Y2$ implies $Y1 \cup use[n] \subseteq Y2 \cup use[n]$

- Put pieces together: $F_0[X] = \text{monotonic}$
  - $(X1 - def[n]) \cup use[n] \supseteq (X2 - def[n]) \cup use[n]$
  - $X1 \subseteq X2$ implies
    - $(X1 - def[n]) \cup use[n] \subseteq (X2 - def[n]) \cup use[n]$
LV: Distributivity

- Are transfer functions: $F_n(X) = (X - \text{def}[n]) \cup \text{use}[n]$ distributive?
- Since $\text{def}[n]$ is constant: $G(X) = X - \text{def}[n]$ is distributive:
  $(X_1 \cup X_2) - \text{def}[n] = (X_1 - \text{def}[n]) \cup (X_2 - \text{def}[n])$
  because: $(a \cup b) - c = (a - c) \cup (b - c)$
- Since $\text{use}[n]$ is constant: $H(Y) = Y \cup \text{use}[n]$ is distributive:
  $(Y_1 \cup Y_2) \cup \text{use}[n] = (Y_1 \cup \text{use}[n]) \cup (Y_2 \cup \text{use}[n])$
  because: $(a \cup b) \cup c = (a \cup c) \cup (b \cup c)$
- Put pieces together: $F_n(X)$ is distributive
  $F_n(X_1 \cup X_2) = F_n(X_1) \cup F_n(X_2)$

Live Variables: Summary

- Lattice: $(\mathbb{Z}, \leq, \cup, \emptyset)$, has finite height
- Meet is set union, top is the empty set
- Is a backward dataflow analysis
- Dataflow equations:
  $\text{in}[a] = (\text{out}[a] \setminus \text{def}[a]) \cup \text{use}[a]$, for all $n$
  $\text{out}[a] = \bigcup \{\text{out}[n] \mid \text{next}(n)\}$, for all $n$
- Transfer functions are monotonic and distributive
- Iterative solution to dataflow equations:
  - terminates
  - computes MOP solution

Problem 2: Available Expressions

- Compute available expressions at each program point
- Available expression = expression evaluated in all program
  executions, and its value would be the same if re-evaluated
- Similar to available copies for constant propagation
- Dataflow information: sets of available expressions
- Example: expressions $(x+y, y-z)$ are available at point $p$
- Is a forward analysis

AE: Dataflow Equations

- Equations:
  $\text{out}[n] = F_i(\text{in}[n])$, for all $n$
  $\text{in}[n] = \bigcap \{\text{out}[n] \mid \text{next}(n)\}$, for all $n$
  $\text{in}[n] = d_i$
- Meaning of intersection meet operator:
  “An expression is available at entry of node $n$ if it is
  available at the exit of all predecessors”

AE: Transfer Functions

- General form of transfer functions:
  $F_n(X) = (X - \text{kill}[n]) \cup \text{gen}[n]$
  where:
  $\text{kill}[n]$ = expressions “killed” by $n$
  $\text{gen}[n]$ = new expressions “generated” by $n$
- Meaning of transfer functions: “Expressions available after node $n$
  include: 1) expressions available before $n$, not killed by $n$, and 2) expressions
  generated by $n$”

Available Expressions: Summary

- Lattice: $(\mathbb{Z}, \subseteq, \cap, \emptyset)$; has finite height
- Meet is set intersection, top element is entire set
- Is a forward dataflow analysis
- Dataflow equations:
  $\text{out}[n] = F_i(\text{in}[n])$, for all $n$
  $\text{in}[n] = \bigcap \{\text{out}[n] \mid \text{next}(n)\}$, for all $n$
  $\text{in}[n] = d_i$
- Transfer functions: $F_n(X) = (X - \text{kill}[n]) \cup \text{gen}[n]$
  - are monotonic and distributive
- Iterative solving of dataflow equation:
  - terminates
  - computes MOP solution
Problem 3: Reaching Definitions

- Compute reaching definitions for each program point
- Reaching definition = definition of a variable whose assigned value may be observed at current program point in some execution of the program
- Dataflow information: sets of reaching definitions
- Example: definitions \((d_2, d_7)\) may reach program point \(p\)
- Is a forward analysis

RD: Dataflow Equations

- Equations:
  \[
  \text{out}[n] = \left( \text{in}[n] - \text{kill}[n] \right) \cup \text{gen}[n], \text{ for all } n
  \]
  \[
  \text{in}[n] = \bigcup \{ \text{out}[n] \mid n' \text{ pred}(n) \}, \text{ for all } n
  \]
  \[
  \text{in}[n] = d_i
  \]
- Meaning of intersection meet operator:
  “A definition reaches the entry of node \(n\) if it reaches the exit of at least one of its predecessor nodes”
- Meaning of transfer functions: “Reaching definitions after node \(n\) include: 1) reaching definitions before \(n\), not killed by \(n\), and 2) reaching definitions generated by \(n\)”

Reaching Definitions: Summary

- Lattice: \((2^\mathbb{N}, \subseteq, \cup, \emptyset)\); has finite height
- Meet is set union, top element is \(\emptyset\)
- Is a forward dataflow analysis
- Transfer functions are monotonic and distributive
- Iterative solving of dataflow equation:
  - terminates
  - computes MOP solution

Efficient Implementation

- Lattices in these analyses = power sets
- Information in these analyses = subsets of a set
- How to implement subsets?
  1. Set implementation
     - Data structure with as many elements as the subset has
     - Usually list implementation
  2. Bitvectors:
     - Use a bit for each element in the overall set
     - Bit for element \(x\): 1 if \(x\) is in subset, 0 otherwise
     - Example: \(S = \{a, b, c\}\), use 3 bits
     - Subset \(\{a, c\}\) is 101, subset \(\{b\}\) is 010, etc.

Implementation Tradeoffs

- Advantages of bitvectors:
  - Efficient implementation of set union/intersection: set union is bitwise “or” of bitvectors
  - Set intersection is bitwise “and” of bitvectors
  - Drawback: inefficient for sparse subsets
- In general, bitvectors work well if the size of the (original) set is linear in the program size

Problem 4: Constant Folding

- Compute constant variables at each program point
- Constant variable = variable having a constant value on all program executions
- Dataflow information: sets of constant values
- Example: \(\{x=2, y=3\}\) at program point \(p\)
- Is a forward analysis
  - Let \(V\) = set of all variables in the program
  - Let \(N\) = set of integer numbers
  - The lattice is a map from \(V\) to \(N\)
  - Construct the lattice starting from a lattice for \(N\)
Constant Folding Lattice

- **Second try:** lattice \( (N \cup \{T, \bot\}, \preceq) \)
  - Where \( \bot \preceq m \), for all \( m \in N \)
  - And \( m \preceq T \), for all \( m \in N \)
  - Is complete!

- **Meaning:**
  - \( \psi = T \): don’t know if \( \psi \) is constant
  - \( \psi = \bot \): \( \psi \) is not constant

- **Note:** meet of any two distinct numbers is \( \bot \)

---

**Constant Folding Lattice**

- **Solution:** flat lattice \( L = (N \cup \{T, \bot\}, \subseteq) \)
  - Where \( \bot \subseteq m \), for all \( m \in N \)
  - And \( m \subseteq T \), for all \( m \in N \)
  - And distinct integer constants are not comparable

- **Diagram:**

```
  T 1
  \ldots
  \vdots
  -2 -1 0 1 2 \ldots
  \bot
```

**CF: Transfer Functions**

- **Transfer function for node** \( n \):
  \[ F_n(X) = (X - \text{kill}[n]) \cup \text{gen}[n] \]

- Dataflow information \( X \) is a map from \( V \) to \( N \cup \{T, \bot\} \)
  - Represent it as a set of pairs \( (\text{var} \rightarrow x) \)
  - Denote by \( X[\text{var}] = x \) the value of var in this mapping

- If \( n \) is \( v = c \) (constant):
  \[ \text{gen}[n] = \{v\rightarrow c\} \]

- If \( n \) is \( v = u + v \):
  \[ \text{gen}[n] = \{v\rightarrow u, v\rightarrow v\} \]

  where \( e = X[u] + X[v] \), if \( X[u] \) and \( X[v] \) are not \( T, \bot \)

  - \( e = \bot \), if \( X[u] = \bot \) or \( X[v] = \bot \)
  - \( e = T \), if \( X[u] = T \) or \( X[v] = T \)

---

**CF: Distributivity**

- **Example:**

```
  x = 2
  y = 3
  z = x + y
```

- **MFP and MOP yield different solutions**
Classification of Analyses

- **Forward analyses**: information flows from
  - CFG entry block to CFG exit block
  - Input of each block to its output
  - Output of each block to input of its successor blocks
  - Examples: available expressions, reaching definitions, constant folding

- **Backward analyses**: information flows from
  - CFG exit block to entry block
  - Output of each block to its input
  - Input of each block to output of its predecessor blocks
  - Example: live variable analysis

Another Classification

- "may" analyses:
  - information describes a property that MAY hold in SOME executions of the program
  - Usually: \( P = U \), \( T = \emptyset \)
  - Hence, initialize info to empty sets
  - Examples: live variable analysis, reaching definitions

- "must" analyses:
  - information describes a property that MUST hold in ALL executions of the program
  - Usually: \( P = \cap \), \( T = S \)
  - Hence, initialize info to the whole set
  - Examples: available expressions