Lecture 5. Data Flow Analysis

Wei Le

2014.10
Abstraction-based Analysis

- **dataflow analysis**: combines model checking’s fix point engine with abstract interpretation of data values.
- **abstract interpretation**: interpretation of program executions over an abstract domain; the semantics of program statements is lifted to operate on a set of states, an element of an abstract domain, rather than a single concrete statements.

note: credit to Matt Dwyer
Dataflow Analysis

What is dataflow analysis?

- The flow of data values
- The global properties of data at a program point
Reaching Definitions
Available Expressions
Live Variable Analysis
Generalizing Dataflow Analysis

An instance of dataflow problem:

- What is the problem?
- The properties of dataflow problems:
  - backward or forward (determine the direction of the dataflow)
  - may or must (determine how to merge dataflow)
- Dataflow equation: Gen, Kill, In, Out (local information)
Generalizing Dataflow Analysis

- **Goal**: solving dataflow equations — determine dataflow for the whole program
  - Reaching definitions: for each program point, what are the definitions can reach
  - Available expressions: for each program point, what are the expressions available
  - Live variables: for each program point, what are the variables available

- **Framework**: a set of data propagation algorithms for a set of dataflow problems

- **Key of the algorithm**: Stabilize in presence of loops — fixpoint, using worklist algorithm

- **Data structure for efficiency**: bit vector to represent the global information
  - Reaching definitions: each definition is a bit, each program point has a bitvector
  - Available expressions: each expression is a bit, each program point has a bitvector
  - Live variable analysis: each variable is a bit, each program point has a bitvector
Dataflow Algorithms

- Exhaustive (Dragon Book)
- Demand-driven [1]

for forward problem, considering all possible paths from the entry to the given point, compute flow value for all the paths, and then meet the flow value at the end of the points
Conservative Analysis
Formal Model to Describe Dataflow Analysis [3, 2]

- **Lattice**: define domain of program properties computed by dataflow analysis (flow value) (the set of elements plus the order of these elements)
- Define *flow functions* (*transfer functions: how each node affects the flow value*) and *merge functions* over this domain using standard lattice operations
Lattice

Concrete lattice
$(L, \sqsubseteq)$

Abstract lattice
$(I, \sqsubseteq)$

Example (Sets of Values)
For a variable ranging over a domain $\mathbb{D}$:
$(\mathcal{P}(\mathbb{D}), \subseteq)$

Example (Sign Lattice)

```
        ⊤
       /   \
      /     \
     /       \
    /         \
   /           \
  /             \
 0               0+
  \\
  \\
  \\
  \\
⊥
```

-0
Lattice models in data flow analysis

Data flow info at a prog. pt. modeled by an element of a lattice

- our convention: if \( a < b \), then \( a \) is **less precise** than \( b \)
  - i.e., \( a \) is a conservative approximation to \( b \)
- top = most precise, best case info
- bottom = least precise, worst case info
- merge function = (meet) on lattice elements
  (the most precise element that’s a conservative approximation to both input elements)
- initial info for optimistic analysis (at least back edges): top
Remember what these flow values represent

- At each program point a lattice element represents an in[] set or an out[] set

Initially for liveness

\[
\begin{align*}
\text{print}(x) & \quad \text{print}(y) \\
\{x \leq x, y \leq y\} & \quad \{x \leq x, y \leq y\}
\end{align*}
\]

Finally

\[
\begin{align*}
\text{print}(x) & \quad \text{print}(y) \\
\{x \leq x, y \leq y\} & \quad \{x \leq x, y \leq y\}
\end{align*}
\]
Example: Liveness analysis with 3 variables
\[ S = \{v1, v2, v3\} \]

- **V:** \(2^S = \{\{v1,v2,v3\}, \{v1,v2\}, \{v1,v3\}, \{v2,v3\}, \{v1\}, \{v2\}, \{v3\}, \emptyset\}\)
- **Meet (\(^\wedge\):** \(\cup\)
  - \(\leq:\) \(\supseteq\)
  - Top(\(T\)): \(\emptyset\)
  - Bottom (\(\perp\)): \(\mathbb{V}\)
- **F:** \(\{f_n(X) = \text{Gen}_n \cup (X - \text{Kill}_n), \forall n\}\)
Transfer Functions

- Let $L = \text{dataflow information lattice}$

- **Transfer function** $F_I : L \to L$ for each instruction $I$
  - Describes how $I$ modifies the information in the lattice
  - If $\text{in}[I]$ is info before $I$ and $\text{out}[I]$ is info after $I$, then
    - Forward analysis: $\text{out}[I] = F_I(\text{in}[I])$
    - Backward analysis: $\text{in}[I] = F_I(\text{out}[I])$

- **Transfer function** $F_B : L \to L$ for each basic block $B$
  - Is composition of transfer functions of instructions in $B$
  - If $\text{in}[B]$ is info before $B$ and $\text{out}[B]$ is info after $B$, then
    - Forward analysis: $\text{out}[B] = F_B(\text{in}[B])$
    - Backward analysis: $\text{in}[B] = F_B(\text{out}[B])$
Transfer Functions

Monotonicity and Distributivity

- Two important properties of transfer functions
- **Monotonicity:** function $F : L \rightarrow L$ is monotonic if $x \sqsubseteq y$ implies $F(x) \sqsubseteq F(y)$
- **Distributivity:** function $F : L \rightarrow L$ is distributive if $F(x \sqcap y) = F(x) \sqcap F(y)$
- **Property:** $F$ is monotonic iff $F(x \sqcap y) \sqsubseteq F(x) \sqcap F(y)$
  - any distributive function is monotonic!

- Give a flow function, more conservative input leads to more conservative output
- IDFA is guaranteed to terminate if the flow function is monotonic and the lattice has a finite height
Flow Functions

Meet operation models how to combine information at split/join points in the control flow
- If in[B] is info before B and out[B] is info after B, then:
  Forward analysis: \( \text{in}[B] = \sqcap \{ \text{out}[B'] \mid B' \in \text{pred}(B) \} \)
  Backward analysis: \( \text{out}[B] = \sqcap \{ \text{in}[B'] \mid B' \in \text{succ}(B) \} \)
WorkList Algorithm

\[
\begin{align*}
\text{in}[B_S] &= X_0 \\
\text{out}[B] &= T, \text{ for all } B \\
\text{worklist} &= \text{set of all basic blocks } B \\
\text{Repeat} \\
\quad \text{Remove a node } B \text{ from the worklist} \\
\quad \text{in}[B] &= \cap \{\text{out}[B'] \mid B' \in \text{pred}(B)\} \\
\quad \text{out}[B] &= F_B(\text{in}[B]) \\
\quad \text{if out}[B] \text{ has changed, then} \\
\quad \quad \text{worklist} &= \text{worklist} \cup \text{succ}(B) \\
\text{Until} \text{ worklist} = \emptyset
\end{align*}
\]
Correctness

Worklist algorithm is correct
– Maintains the invariant that
  \[ \text{in}[B] = \bigcap \{ \text{out}[B'] \mid B' \in \text{pred}(B) \} \]
  \[ \text{out}[B] = F_B(\text{in}[B]) \]
  for all the blocks B not in the worklist
– At the end, worklist is empty
Termination

- **Key observation**: at each iteration, information decreases in the lattice
  \[ \text{in}_{k+1}[B] \sqsubseteq \text{in}_k[B] \text{ and } \text{out}_{k+1}[B] \sqsubseteq \text{out}_k[B] \]
  where \( \text{in}_k[B] \) is info before \( B \) at iteration \( k \) and \( \text{out}_k[B] \) is info after \( B \) at iteration \( k \)

- **Proof by induction**:
  - Induction basis: true, because we start with top element, which is greater than everything
  - Induction step: use monotonicity of transfer functions and meet operation

- **Information forms a chain**: \( \text{in}_1[B] \sqsupseteq \text{in}_2[B] \sqsupseteq \text{in}_3[B] \ldots \)
Evelyn Duesterwald, Rajiv Gupta, and Mary Lou Soffa.  
A practical framework for demand-driven interprocedural data flow analysis.  

John B. Kam and Jeffrey D. Ullman.  
Global data flow analysis and iterative algorithms.  

Gary A. Kildall.  
A unified approach to global program optimization.  