DFA foundations

Simone Campanoni
simonec@eecs.northwestern.edu
We have seen several examples of DFAs

• Are they correct?
• Are they precise?
• Will they always terminate?
• How long will they take to converge?
Outline

• Lattice and data-flow analysis

• DFA correctness

• DFA precision

• DFA complexity
Understanding DFAs

• We need to understand all of them
  • Liveness analysis: is it correct? Precision? Convergence?
  • Reaching definitions: is it correct? Precision? Convergence?
  • ...

• Idea: create a framework to help reasoning about them
  • Provide a single formal model that describes all data-flow analyses
  • Formalize the notions of “safe,” “conservative,” and “optimal”
  • Correctness proof for DFAs
  • Place bounds on time complexity of iterative DFAs
Lattices

- Lattice $L = (V, \leq)$:
  - $V$ is a (possible infinite) set of elements
  - $\leq$ is a binary relation over elements of $V$
- Properties of $\leq$:
  - $\leq$ is a partial order (reflexive, transitive, anti-symmetric)
  - Every pair of elements in $V$ has
    - A unique greatest lower bound (a.k.a. meet) and
    - A unique least upper bound (a.k.a. join)
- Top ($T$) = unique greatest element of $V$ (if it exists)
- Bottom ($\bot$) = unique least element of $V$ (if it exists)
- Height of $L$: longest path from $T$ to $\bot$
  - Infinite large lattice can still have finite height
Lattices and DFA

• A lattice \( L = (V, \leq) \) describes all possible solutions of a given DFA
  • A lattice for reaching definitions
  • Another lattice for liveness analysis
  • ...

• The relation \( \leq \) connects all solutions of its related DFA from the best one (\( T \)) to the worst one (\( \perp \))
  • Liveness analysis:
    \( T \) = no variable is alive = \{ \}
    \( \perp \) = all variables are alive = \( V \)

• We traverse the lattice of a given DFA to find the correct solution in a given point of the CFG
  • We repeat it for every point in the CFG
Lattice example

• How many apples I must have?
• \( V = \text{sets of apples} \)
• \( \leq = \text{set inclusion} \)
  \[
  \{\text{apple}\} \leq \{\text{apple, apple}\}
  \]
• \( T = \text{(best case)} = \text{all apples} \)
• \( \bot = \text{(worst case)} \text{ no apples (empty set)} \)

Apples, definitions, variables, expressions ...
How can we use this mathematical framework, lattice, to study a DFA?
Use of lattice for DFA

• Define domain of program properties (flow values --- apple sets) computed by data-flow analysis, and organize the domain of elements as a lattice

• Define how to traverse this domain to compute the final solution using lattice operations

• Exploit lattice theory in achieving goals
Data-flow analysis and lattice

- Elements of the lattice (V) represent flow values (e.g., an IN[] set)
  - *e.g.*, Sets of apples
Data-flow analysis and lattice

• Elements of the lattice (V) represent flow values (e.g., an IN[] set)
  • *e.g.*, Sets of live variables for liveness

• ⊥ “worst-case” information
  • *e.g.*, Universal set

• T “best-case” information
  • *e.g.*, Empty set

• If \( x \leq y \), then \( x \) is a conservative approximation of \( y \)
  • *e.g.*, Superset

\[
\begin{align*}
T &= \{ \} \\
\{v1\} &\quad \{v2\} & \quad \{v3\} \\
\{v1,v2\} & \quad \{v1,v3\} & \quad \{v2,v3\} \\
\perp &= \{v1,v2,v3\}
\end{align*}
\]
Data-flow analysis and lattice (reaching defs)

• Elements of the lattice (V) represent flow values (IN[], OUT[])
  • *e.g.*, Sets of definitions

• $T$ represents “best-case” information
  • *e.g.*, Empty set

• $\bot$ represents “worst-case” information
  • *e.g.*, Universal set

• If $x \leq y$, then $x$ is a conservative approximation of $y$
  • *e.g.*, Superset
How do we choose which element in our lattice is the data-flow value of a given point of the input program?
We traverse the lattice

\[ T = \{ \{ \}, \{ \}, \{ \}, \{ \}, \{ , , \} \} \]
Merging information

• New information is found
  • e.g., a new definition (d1) reaches a given point in the CFG

• New information is described as a point in the lattice
  • e.g. \{d1\}

• We use the "meet" operator (\wedge) of the lattice to merge the new information with the current one
  • e.g., set union
  • Current information: \{d2\}
  • New information: \{d1\}
  • Result: \{d1\} U \{d2\} = \{d1, d2\}
How can we find new facts/information to iterate over the lattice?
Computing a data-flow value (ideal)

• For a forward problem, consider all possible paths from the entry to a given program point, compute the flow values at the end of each path, and then meet these values together

• Meet-over-all-paths (MOP) solution at each program point
  • It’s a correct solution
Computing MOP solution for reaching definitions

$V_{\text{entry}}$

Entry

d1

d2

d3

$T = \{ \{d1\} \{d1,d2\} \{d1,d2,d3\} \}$
From ideal to practical solution

• **Problem**: all preceding paths must be analyzed
  • Exponential blow-up

• **Solution**: compute meets early (at merge points) rather than at the end
  • Maximum fixed-point (MFP)

• **Questions**:
  • Is MFP correct?
  • What’s the precision of MFP?
Outline

• Lattice and data-flow analysis

• DFA correctness

• DFA precision

• DFA complexity
Correctness

\[ V_{\text{correct}} \leq V_{\text{MOP}} \]
Correctness

• Key idea:
  • “Is MFP correct?” iff $V_{MFP} \leq V_{MOP}$

• Focus on merges:
  • $V_{MOP} = fs(V_{p1}) \land fs(V_{p2})$
  • $V_{MFP} = fs(V_{p1} \land V_{p2})$
  • $V_{MFP} \leq V_{MOP}$ iff $fs(V_{p1} \land V_{p2}) \leq fs(V_{p1}) \land fs(V_{p2})$

• If $fs$ is monotonic: $X \leq Y$ then $fs(X) \leq fs(Y)$
  • So $fs(V_{p1} \land V_{p2}) \leq fs(V_{p1}) \land fs(V_{p2})$
  • And therefore $V_{MFP} \leq V_{MOP}$

$fs$ is monotonic => MFP is correct!
Monotonicity

- $X \leq Y$ then $fs(X) \leq fs(Y)$

- If the flow function $f$ is applied to two members of $V$, the result of applying $f$ to the “lesser” of the two members will be under the result of applying $f$ to the “greater” of the two

- More conservative inputs leads to more conservative outputs (never more optimistic outputs)
Convergence

• **From lattice theory**
  If $fs$ is monotonic,
  then the maximum number of times $fs$ can be applied
  w/o reaching a fixed point is $\text{Height}(V) - 1$

• Iterative DFA is guaranteed to terminate
  if the $fs$ is monotonic and
  the lattice has finite height
Outline

• Lattice and data-flow analysis

• DFA correctness

• DFA precision

• DFA complexity
Precision

• $V_{MOP}$: the best solution

• $V_{MFP} \leq V_{MOP}$
  • $fs(V_{p1} \land V_{p2}) \leq fs(V_{p1}) \land fs(V_{p2})$

• Distributive $fs$ over $\land$
  • $fs(V_{p1} \land V_{p2}) = fs(V_{p1}) \land fs(V_{p2})$
  • $V_{MFP} = V_{MOP}$

• Is reaching definition $fs$ distributive?
A new DFA example: reaching constants

• Goal
  • Compute the value that a variable must have at a program point

• Flow values (V)
  • Set of (variable,constant) pairs

• Merge function
  • Intersection

• Data-flow equations
  • Effect of node n x = c
    • $\text{KILL}[n] = \{(x,k) | \forall k\}$
    • $\text{GEN}[n] = \{(x,c)\}$
  • Effect of node n x = y + z
    • $\text{KILL}[n] = \{(x,k) | \forall k\}$
    • $\text{GEN}[n] = \{(x,c) | c=\text{valy}+\text{valz}, (y, \text{valy}) \in \text{in}[n], (z, \text{valz}) \in \text{in}[n]\}$
Reaching constants: characteristics

• $\bot = ? \{ \}$

• IN = ?

• OUT = ?

• Let’s study this analysis
  • Correctness?
  • Convergence?
    • is $fs$ monotonic? Has the lattice a finite height?
  • Precision?
    • is $fs$ distributive?
Outline

• Lattice and data-flow analysis

• DFA correctness

• DFA precision

• DFA complexity
Complexity

\[ \text{OUT}[\text{ENTRY}] = \{ \} \];

for (each instruction \( i \) other than ENTRY) \( \text{OUT}[i] = \{ \} \);

while (changes to any \( \text{OUT} \) occur){
    for (each instruction \( i \) other than ENTRY) {
        \( \text{IN}[i] = \bigcup p \) a predecessor of \( i \) \( \text{OUT}[p] \);
        \( \text{OUT}[i] = \text{GEN}[i] \cup (\text{IN}[i] - \text{KILL}[i]) \);
    }
}

Complexity

• N instructions (N definitions at most)
  • Each IN/OUT set has at most N elements
  • Each set-union operation takes O(N) time
  • The for loop body
    • constant # of set operations per node
    • O(N) nodes ⇒ O(N^2) time for the loop
  • Each iteration of the repeat loop can only make the set larger
  • Each iteration modifies in the worst case only one set ⇒ O(N^3)
  • N iterations to reach the fixed point at most

• Worst case: O(N^4)

• Typical case: 2 to 3 iterations with good ordering & sparse sets
  • O(N) to O(N^2)

N=500
Worst case: 62,500,000,000
Optimized average case: 500 – 250,000
Optimization: work list

OUT[ENTRY] = \{ \};
for (each basic block B other than ENTRY) OUT[B] = \{ \};
workList = all basic blocks
while (workList isn’t empty)
    B = pick and remove a block from workList
    oldOUT = OUT[B]
    IN[B] = \bigcup p \text{ a predecessor of } B \text{ OUT}[p];
    if (oldOut \neq OUT[B]) workList = workList \bigcup \{ all successors of B \}
}