DFA foundation

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 $V$

• Lower bound
  • $z$ is a lower bound of $x$ and $y$ iff $z \leq x$ and $z \leq y$

• Upper bound
  • $z$ is a upper bound of $x$ and $y$ iff $x \leq z$ and $y \leq z$

• Operations: meet ($\wedge$) and join ($\vee$)
  • $b \vee c$: least upper bound
  • $b \wedge c$: greater lower bound
  • An useful property: if $e \leq b$ and $e \leq c$, then $e \leq b \wedge c$
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 ($\perp$) = unique least element of $V$ (if it exists)

• Height of $L$: longest path from $T$ to $\perp$
  • 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
  • ...
  • For DFAs that look for solutions per point in the CFG, then
    1 “lattice instance” per point

• The relation $\leq$ connects all solutions of its related DFA from the best one ($T$) to the worst one --most conservative one--($\perp$)
  • Liveness analysis: *variables that might be used after a given point in the CFG*
    $T = \text{no variable is alive} = \{ \}$
    $\perp = \text{all variables are alive} = V$

• 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?
- \( V = \) sets of apples
- \( \leq = \) set inclusion
- \( T = (\text{best case}) = \) all apples
- \( \perp = (\text{worst case}) \) no apples (empty set)

Apples, definitions, variables, expressions ...
Another lattice example

- How many apples I may have?
- $V =$ sets of apples
- $\leq =$ set inclusion
  \[
  \{\text{🍎, 🍏} \} \leq \{\text{🍎, 🍏, 🍐} \}
  \]
- $T =$ no apples (empty set)
- $\bot =$ (most conservative) all apples

Conservativeness

Precision
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

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

\[
\bot = \{\}
\]
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
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

• ⊥ 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

for (each instruction $i$ other than ENTRY) $\text{OUT}[i] = \{ \}$;
We traverse the lattice

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

$T = \{ \}$

$\{ \text{d1} \} \quad \{ \text{d2} \} \quad \{ \text{d3} \}$

$\{ \text{d1, d2} \} \quad \{ \text{d1, d3} \} \quad \{ \text{d2, d3} \}$

$\bot = \{ \text{d1, d2, d3} \}$
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

Entry

$V_{\text{entry}}$

d1

d2

d3

T={ }

{d1}

{d1,d2}

{d1,d2,d3}
The problem of ideal solution

• **Problem**: all preceding paths must be analyzed
  • Exponential blow-up
• To compute the MOP solution in BB2: 0-1-A, 1-2-A
  0-1-B, 1-2-A
  0-1-A, 1-2-B
  0-1-B, 1-2-B
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)

\[ \text{IN}[i] = \bigcup \text{p a predecessor of } i \text{ OUT}[p]; \]

- **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 \( fs \) is monotonic \( \Rightarrow \) MFP is correct!

• 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 \( fs(X) \leq fs(Y) \)
  • \( (V_{p1} \land V_{p2}) \leq V_{p1} \) by definition of meet
  • \( (V_{p1} \land V_{p2}) \leq V_{p2} \) by definition of meet
  • So \( fs(V_{p1} \land V_{p2}) \leq fs(V_{p1}) \) and \( fs(V_{p1} \land V_{p2}) \leq fs(V_{p2}) \)
  • Therefore \( fs(V_{p1} \land V_{p2}) \leq fs(V_{p1}) \land fs(V_{p2}) \)
  • And therefore \( V_{MFP} \leq V_{MOP} \)
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 $f_s$ is monotonic, then the maximum number of times $f_s$ can be applied w/o reaching a fixed point is $\text{Height}(V) - 1$

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

• Lattice and data-flow analysis

• DFA correctness

• DFA precision

• DFA complexity
Precision

• $V_{\text{MOP}}$: the best solution
• $V_{\text{MFP}} \leq V_{\text{MOP}}$
  • $fs(V_{p1} \land V_{p2}) \leq fs(V_{p1}) \land fs(V_{p2})$
• Distributive $fs$ over $\land$
  • $fs(V_{p1} \land V_{p2}) = fs(V_{p1}) \land fs(V_{p2})$
  • $V_{\text{MFP}} = V_{\text{MOP}}$
• Is reaching definition $fs$ distributive?
  • (did having performed $\land$ earlier change anything?)

* is distributive over +
$4 \times (2 + 3) = 4 \times (5) = 20$
$(4 \times 2) + (4 \times 3) = 8 + 12 = 20$

i:v1 = 3
j:v2 = 4

... i and j reach this point
k:v3 = v1 + v2
A new DFA example: reaching constants

• **Goal**
  • Compute the value that a variable must have at a program point (no SSA)

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

• **Merge function**
  • Intersection

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

\[ v_1 = 3 \]
\[ v_2 = 4 \]
\[ v_3 = v_1 + v_2 \]
\[ v_3 \text{ is 7} \]
Reaching constants: characteristics

• \( \bot = ? \)
• IN = ?
• OUT = ?
• Let’s study this analysis
  • Does it convergence?
    • is \( fs \) monotonic? Has the lattice a finite height?
  • What is the precision of the solution?
    • is \( fs \) distributive?
Outline

• Lattice and data-flow analysis

• DFA correctness

• DFA precision

• DFA complexity
Complexity

OUT[ENTRY] = { };  
for (each instruction \( i \) other than ENTRY)  OUT[\( i \)] = { };  
do {  
  for (each instruction \( i \) other than ENTRY) {  
    IN[\( i \)] = \( \bigcup \)  \( p \) a predecessor of \( i \) OUT[\( p \)];  
    OUT[\( i \)] = GEN[\( i \)] \( \cup \) (IN[\( i \)] \( \setminus \) KILL[\( i \)]);  
  }  
} while (changes to any OUT occur);
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
  • Between N and N^2

N=500
Worst case: 62,500,000,000
Optimized average case: 500 – 250,000
Optimization: basic blocks

\[
\text{OUT}[\text{ENTRY}] = \{ \};
\]
for (each basic block \(B\) other than \(\text{ENTRY}\)) \(\text{OUT}[B] = \{ \}\);
do {
  for (each basic block \(B\) other than \(\text{ENTRY}\)) {
    \(\text{IN}[B] = \bigcup \{\text{OUT}[p] : p\ \text{is a predecessor of } B\}\);
    \(\text{OUT}[B] = \text{GEN}[B] \cup (\text{IN}[B] - \text{KILL}[B])\);
  }
} while (changes to any \(\text{OUT}\) occur);
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] = ∪ \textit{p}, a predecessor of B \textit{OUT}[p];
  if (oldOut != OUT[B]) workList = workList U \{all successors of B\} 
}