DFA foundation

Simone Campanoni
simone.campanoni@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 “correctness,” “conservativeness,” and “optimality”
  • Correctness proof for DFAs
  • Place bounds on time complexity of iterative DFAs
  • This is not to drive the implementation, but to reason about data-flow analyses
Lattice

- 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 (\( \land \)) and join (\( \lor \))
  - \( b \lor c \): least upper bound
  - \( b \land c \): greater lower bound
  - An useful property: if \( e \leq b \) and \( e \leq c \), then \( e \leq b \land c \)
Lattice

• 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 ($\land$) and join ($\lor$)
  • $b \lor c$: least upper bound
  • $b \land c$: greater lower bound

• An useful property: if $e \leq b$ and $e \leq c$, then $e \leq b \land c$
Lattice

• 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
    • An unique greatest lower bound (a.k.a. meet) and
    • An 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

\[
\begin{array}{ccc}
  & a & \\
b & & c \\
c & d & \\
\end{array}
\]

If you know nothing, this is still a correct, but conservative, solution
Lattice 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 one “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--(\( \bot \))
  • Liveness analysis: variables that might be used after a given point in the CFG
    \( T = \) no variable is alive = \{ \}
    \( \bot = \) all variables are alive = \( V \)

Why? \( \bot = \) all variables are alive = \( V \)

• To solve a data-flow analysis: 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
  \{ \text{apple} \} \leq \{ \text{apple}, \text{apple} \}
• T = (best case) = all apples
• \bot = (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
  \[
  \{\text{apple, apple}\} \leq \{\text{apple}\}
  \]
• \( T = \) no apples (empty set)
• \( \bot = \) (most conservative) all apples
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

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

\[
\bot = \{ \}\n\]
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
- \( \perp \) “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 ≤ 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

How many apples I must have?

We found out there is no guarantee we have the green apple
We traverse the lattice

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

- New information discovered while computing the IN/OUT sets will bring us down in the lattice
- New information is merged into the current knowledge/state/current-point-in-the-lattice

...let’s see how
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}
We traverse the lattice

We discover:
- a new definition, $d_1$, reaches our point in the CFG

New fact = \{d_1\}
\{ \} \land \{d_1\} = \{d_1\}

As long as we know how to get new information, then we know how to traverse the lattice to converge to the final solution

\[
\begin{align*}
T &= \{ \} \\
\{ d_1 \} &\quad \{ d_2 \} & \quad \{ d_3 \} \\
\{d_1,d_2\} &\quad \{d_1,d_3\} & \quad \{d_2,d_3\} \\
\bot &= \{d_1,d_2,d_3\}
\end{align*}
\]
How can we find new facts/information?
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
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-A, 1-2-B
  - 0-1-B, 1-2-A
  - 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)

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

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

• Lattice and data-flow analysis

• DFA correctness

• DFA precision

• DFA complexity
Correctness

$V_{entry}$

$T = \emptyset$

$V_{correct} \leq V_{MOP}$
Correctness \( fs \) is monotonic => 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 \( V_{MFP} \leq V_{MOP} \)

Let us compare

An useful property: if \( e \leq b \) and \( e \leq c \), then \( e \leq b \land c \)
Monotonicity

• $X \leq Y$ then $f_s(X) \leq f_s(Y)$

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

• Iterative DFA is guaranteed to terminate if the $f$ 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?
  • (did having performed $\land$ earlier change anything?)

\[ * \text{ is distributive over } + \]
\[
4 \times (2 + 3) = 4 \times (5) = 20 \\
(4 \times 2) + (4 \times 3) = 8 + 12 = 20
\]

\[
\begin{array}{c}
i:v1 = 3 \\
j:v2 = 4 \\
k:v3 = v1 + v2
\end{array}
\]

i and j reach this point
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
    • KILL[n] = {(x,k) | ∀k}
    • GEN[n] = {(x,c)}
  • Effect of node n: x = y + z
    • KILL[n] = {(x,k) | ∀k}
    • GEN[n] = {(x,c) | c=valy+valz, (y, valy) ∈ IN[n], (z, valz) ∈ IN[n]}

\[
\begin{align*}
v_1 &= 3 \\
v_2 &= 4 \\
v_3 &= v_1 + v_2 \\
v_3 &= 7
\end{align*}
\]
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

\[
\text{OUT[ENTRY]} = \{ \}; \\
\text{for (each instruction } i \text{ other than ENTRY)} \quad \text{OUT}[i] = \{ \}; \quad \text{do } \{ \\
\quad \text{for (each instruction } i \text{ other than ENTRY)} \{ \\
\quad \quad \text{IN}[i] = \bigcup_p \text{ a predecessor of } i \quad \text{OUT}[p]; \\
\quad \quad \text{OUT}[i] = \text{GEN}[i] \cup (\text{IN}[i] \ominus \text{KILL}[i]); \\
\quad \} \\
\} \quad \text{while (changes to any OUT occur)};
\]
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 complexity of a single invocation of for
    • constant # of set operations per node
    • O(N) nodes ⇒ O(N^2) time for the loop
  • Each invocation of the for loop can only make the IN/OUT sets larger
  • Each invocation modifies in the worst case only one set ⇒ O(N^3)
  • N invocations to reach the fixed point at most

• Worst case: O(N^4)

• Typical case: 2 to 3 invocations 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 ENTRY) \hspace{1em} $\text{OUT}[B] = \{ \}$;
\hspace{1em} do \{ \hspace{1em} \\
\hspace{2em} for (each basic block $B$ other than ENTRY) \{ \hspace{1em} \\
\hspace{3em} $\text{IN}[B] = \bigcup_{\text{a predecessor of } B} \text{OUT}[p]$; \hspace{1em} \\
\hspace{3em} $\text{OUT}[B] = \text{GEN}[B] \cup (\text{IN}[B] - \text{KILL}[B])$; \hspace{1em} \\
\hspace{2em} \} \hspace{1em} \\
\hspace{1em} \} \hspace{1em} while (changes to any OUT occur);
Optimization: work list

\[
\begin{align*}
  \text{OUT}[\text{ENTRY}] &= \{ \}; \\
  \text{for (each basic block } B \text{ other than ENTRY) } & \quad \text{OUT}[B] = \{ \}; \\
  \text{workList} &= \text{all basic blocks} \\
  \text{while (workList isn’t empty)} & \\
  \quad B &= \text{pick and remove a block from workList} \\
  \quad \text{oldOUT} &= \text{OUT}[B] \\
  \quad \text{IN}[B] &= \bigcup \text{p a predecessor of } B \text{ OUT}[p]; \\
  \quad \text{OUT}[B] &= \text{GEN}[B] \cup (\text{IN}[B] \setminus \text{KILL}[B]); \\
  \text{if (oldOut !=} \text{OUT}[B]) & \quad \text{workList} = \text{workList U \{all successors of } B \} \\
\end{align*}
\]