CS 516 Compilers and Programming Languages II

Approximation 2
New homework is posted on data flow analysis frameworks / approximation

First homework submission extension until Friday 21, 11:59pm

Will post first papers for in-class discussion soon,
Underlying assumption: There is a ground truth consisting of the perfect/correct/optimal answer. This answer may not be effectively computable. An approximate answer comes “close” to the ground truth, where “closeness” is an application-specific metric.

Model approximation (solving a real-world problem through an algorithm):
- use a simpler view of the world (ground truth) that still works for a particular application
  (example: assume the world is flat for a car-navigation application)
- even optimal solution of the model is approximation to ground truth (e.g.: simplified physical interactions)

Data approximations:
- less precise data (e.g.: single vs. double)
- use a subset of data (e.g.: statistical representation, subset probing)
- probabilistic data values

Computation approximation:
- solve a model non-optimally (e.g.: use a heuristic)
  + relaxed convergence criteria
  + imprecise solution strategies (e.g.: simpler stencil for PDE solvers)
  + probabilistic solution strategies
Example: Data flow analysis

How to represent information?

Answer: Represent information as “elements” in a semi-lattice

A meet semi-lattice \((\text{Info}, \land)\)
- domain of values \(\text{Info}\)
- meet operator \(\land: \text{Info} \times \text{Info} \rightarrow \text{Info}\)

With binary operator \(\land\) is
  - commutative: \(a \land b = b \land a\)
  - associative: \(a \land (b \land c) = (a \land b) \land c\)
  - idempotent: \(a \land a = a\)

The meet operator \(\land\) can be used to define a partial order over pieces of information:

\[ a \land b = a \iff a \leq b \]
A lattice \((\text{Info}, \land, \lor)\) has two binary operators
- domain of values \text{Info}
- meet operator \(\land: \text{Info} \times \text{Info} \to \text{Info}\)
- join operator : \(\lor: \text{Info} \times \text{Info} \to \text{Info}\)

with binary operators are basically “duals” of each other that model information “intersection” (meet) and “union” (join). Both \((\text{Info}, \land)\) and \((\text{Info}, \lor)\) are semi-lattices.

The operators \(\land\) and \(\lor\) define a partial order over \text{Info} (pieces of information):

\[
a \lor b = b \quad \text{IFF} \quad a \land b = a \quad \text{IFF} \quad a \leq b
\]
You are given a rectangular, 2-dimensional array / data regions.

Build a lattice that models the “intersection” and the “union” of two rectangular regions “a” and “b”. Note: the regions may or may not overlap.

\[ a \land b = \]

\[ a \lor b = \]

Do we have to use approximation here?
Exercise

You are given a rectangular, 2-dimensional array / data regions.

Build a lattice that models the “intersection” and the “union” of two rectangular regions “a” and “b”. Note: the regions may or may not overlap.

Do we have to use approximation here?
You are given a rectangular, 2-dimensional array / data regions.

Build a lattice that models the “intersection” and the “union” of two rectangular regions “a” and “b”.
Note: the regions may or may not overlap.
• \((x < y) \equiv (x \leq y) \land (x \neq y)\)

• A semi-lattice diagram:
- Set of nodes: set of values/information units
- Set of edges \(\{(y, x): x < y \text{ and } \neg \exists z \text{ s.t. } (x < z) \land (z < y)\}\)

\[
\begin{align*}
\text{a} & \quad \text{b} \\
\text{a} & \quad \text{b} \\
\text{a} & \quad \text{b}
\end{align*}
\]

\(a < b\), but no edge from \(a\) to \(b\)

Avoid transitive edges
Definition

The **height** of a semi-lattice is the largest number of > relations that will fit in a descending chain:
\[ x_0 > x_1 > \ldots \]

**Descending chain property:**
All chains are of finite height

Example:

What is the maximal height of a chain for the \( \{a, b, c\}, \cap \) semi-lattice?

What about the \( \{a, b, c\}, \cup \) semi-lattice?
One classical application for semi-lattices

Data-flow frameworks \((F, \text{Info}, \land)\) are defined by
- A semi-lattice
  - domain of values \text{Info}
  - operator \(\land : \text{Info} \times \text{Info} \rightarrow \text{Info}\)
- A family of transfer/propagation functions \(F : \text{Info} \rightarrow \text{Info}\)

Basic Properties \(f : \text{Info} \rightarrow \text{Info}\)

- There is an identity function
  - \(\exists f \in F\) such that \(f(x) = x\) for all \(x\).
- Closed under composition
  - \(\text{if } f_1, f_2 \in F, f_1 \cdot f_2 \in F\)
A framework \((F, \text{Info}, \land)\) is monotone iff
- \(x \leq y\) implies \(f(x) \leq f(y)\)

Equivalently,
a framework \((F, \text{Info}, \land)\) is monotone iff
- \(f(x \land y) \leq f(x) \land f(y)\),
- meet inputs, then apply \(f\) is \(\leq\) to apply \(f\) individually to inputs, then meet results

Lesson: Merging information “early” may lead to information loss, and summarizing “paths” may lead to information loss
Control Flow Graph (CFG)

Directed graph representing a single program procedures.

Nodes: Basic blocks in a program
Edges: Possible control flow between basic blocks

Example:
```
S0
if (x < 100) then
  S1
else
  S2
end
if (x >= 100) then
  S3
else
  S4
end
S5
```
Homework 2: Directed graph representing a single program procedures.

Nodes: Basic blocks in a program
Edges: Possible control flow between basic blocks

procedure foo( )
    S0: a = 1
        b = 2
        read(x)
    C6: if (x < 100) then
        S1: c = 5
        else
        S2: a = 2
            b = 10
            end
    C7: if (x >= 100) then
        S3: a = 3
            b = b + 1
            else
        S4: b = c + 2
            end
    S5: print(a + b + c)
Instance of a data flow problem

- (Forward) monotone data-flow framework \((F, \text{Info, } \wedge)\)
- \(\text{CFG } G = (N, E)\)
- Assignment of transfer functions in \(F\) to nodes: \(N \rightarrow F\)
- Initialization of all node values to \(T\) element

Solution: All equations

\[\text{IN}_x = \bigwedge_{p \in \text{PRED}(x)} f_p(\text{IN}_p)\]

\[f_x(\text{IN}_x) = \text{OUT}_x\]

are simultaneously satisfied.

\[\text{IN}_x \quad f_x \quad \text{OUT}_x\]
Instance of a data flow problem

- Can compute a solution using an iterative algorithm
- Algorithm is guaranteed to terminate with MFP (maximal fixed-point) solution if
  - Info has descending chain property
  - Monotone data flow framework

Solution: All equations

\[ \text{IN}_x = \bigwedge_{p \in \text{PRED}(x)} f_p(\text{IN}_p) \]

\[ f_x(\text{IN}_x) = \text{OUT}_x \]

are simultaneously satisfied.
Example data flow problem

Reaching Definitions (RD)

$$RD_x = \bigcup_{p \in \text{PRED}(x)} f_p(RD_p)$$

Each transfer/propagation function $f$ can be described by two sets, namely Gen and Kill:

$$f(x) = \text{Gen} \cup (x - \text{Kill}),$$

**Gen** = {set of assignments to variables in node}  
**Kill** = {set of assignments to same variables in other nodes}

Semi-lattice based on union $\land = \cup$

**top element** $T$ is the empty set
Is RD a monoton data flow framework?

\[ f(x \cup y) \subseteq f(x) \cup f(y) \]

Transfer/propagation function:
\[ f(x) = \text{Gen } U \ (x - \text{Kill}) \]

Proof:
\[ f(x_1 \cup x_2) = \text{Gen } U \ ((x_1 \cup x_2) - \text{Kill}) = \]
\[ \text{Gen } U \ ((x_1 - \text{Kill}) \cup (x_2 - \text{Kill})) = \]
\[ \text{Gen } U \ (x_1 - \text{Kill}) \cup \text{Gen } U \ (x_2 - \text{Kill}) = f(x_1) \cup f(x_2) \]

In fact, RD is not only monotone, but also **distributive**:
\[ f(x \cup y) = f(x) \cup f(y) \quad -- \text{no information loss} \]
**Input:** CFG (N, E)  
forward data flow framework with \( f(x) = \text{Gen} \cup (x - \text{Kill}) \)

**Output:** Maximal fixed point solution represented as IN(n),

\[
\begin{align*}
\text{for } n \in N \\
\text{IN}(n) &= \emptyset \quad // \text{init to T} \\
\text{OUT}(n) &= \text{GEN}(n)
\end{align*}
\]

endfor

worklist ← n ∈ N

while ( worklist ≠ \emptyset )

pick and remove a node n from worklist

oldout(n) = OUT(n)

\[
\begin{align*}
\text{IN}(n) &= \bigcup (\text{OUT}(p)), \quad p \in \text{PRED}(n) \\
\text{OUT}(n) &= \text{GEN}(n) \cup (\text{IN}(n) - \text{KILL}(n))
\end{align*}
\]

if oldout(n) ≠ OUT(n) then

worklist ← worklist \cup \text{SUCC}(n)

\]

endwhile
Example data flow problem

\begin{equation}
\begin{array}{c}
f_1 \\
\downarrow \\
f_2 \\
\downarrow \\
f_3 \quad f_4 \\
\downarrow \\
f_5 \\
\downarrow \\
f_6 \quad f_7 \\
\downarrow \\
f_8
\end{array}
\end{equation}

\begin{align*}
a &= 1 \\
a &= 2 \\
a &= 3
\end{align*}
Approximation: CFG

Assumption: No assignment to x in the two branches.

There are only two possible (feasible) execution path in this program. CFG assumes that all paths are feasible!

Other examples of approximations:
- computed GOTOs
- “inter-procedural” CFGs

Model Approximation
Our example join (v) semi-lattice.

Key issue: how much information to encode in every element in the “info” set of the semi-lattice.

Data Approximation
In a monotone data flow problem, information may be lost due to merging information (join or meet).

**MOP** (meet over all path solution)
enumerate all possible execution paths and compute information for these paths without merging. This is the best you can do!

\[
\text{MOP}(n) = \bigwedge_{p \in \text{PATH}(n)} f_p(T)
\]

**MFP** (maximal fixed-point solution)
initialize information with top element, and then propagate and merge information.

**FP** (fixed-point solution)
any solution

\[ \text{FP} \leq \text{MFP} \leq \text{MOP} \]