Final Exam

COSE312 Compilers, Fall 2015

Instructor: Hakjoo Oh

Problem 1 (Program Analysis, 40pts) The central technology of modern optimizing compilers is static program analysis, which aims to reason about the program's run-time behavior at compile-time. The key idea is that compilers execute the program *abstractly* (with so-called *abstract values*) rather than concretely. The purpose of this problem is to explore this idea by going through a series of static analyses for a simple programming language.

Consider the following expression language:

$$e \rightarrow n \mid -e \mid e_1 + e_2 \mid e_1 \diamond e_2 \mid e^* \tag{1}$$

An expression is an integer ($n \in \mathbb{Z}$), a negation ($-e$), an addition ($e_1 + e_2$), a non-deterministic choice ($e_1 \diamond e_2$), or a closure (e^*). Evaluating an expression e produces an integer. The choice expression $e_1 \diamond e_2$ evaluates to the value of either e_1 or e_2. The closure e^* evaluates to one of the values of $e^1, e^2, e^3, \cdots$. For example,

- $2 + 4$ evaluates to 6.
- $-1 \diamond 1$ (randomly) evaluates to one of $\{-1, 1\}$.
- $2 + (-2 \diamond 2)$ evaluates to one of $\{0, 4\}$.
- $(2 \diamond 3)^*$ evaluates to one of $\{2, 3, 4, 8, 9, 16, 27, \cdots\}$.

Formally, the possible outcomes of an expression are defined by the evaluation function $\mathcal{V} : e \rightarrow \mathcal{P}(\mathbb{Z})$ as follows:

$$\begin{array}{rcl} \mathcal{V}(n) & = & \{n\} \\ \mathcal{V}(-e) & = & \{-n \mid n \in \mathcal{V}(e)\} \\ \mathcal{V}(e_1 + e_2) & = & \{n_1 + n_2 \mid n_1 \in \mathcal{V}(e_1) \wedge n_2 \in \mathcal{V}(e_2)\} \\ \mathcal{V}(e_1 \diamond e_2) & = & \mathcal{V}(e_1) \cup \mathcal{V}(e_2) \\ \mathcal{V}(e^*) & = & \{n^k \mid n \in \mathcal{V}(e) \ \wedge \ k \geq 1\} \end{array}$$

Let us call $\mathcal{V}$ the *concrete semantics* of the expression language.

A static analysis is obtained by abstracting the concrete semantics. We go through three instances of such approximations below.

1. (10pts) The first analysis aggressively approximates the concrete semantics and focuses only on finding out even numbers. It evaluates a given expression in terms of the following abstract values:

   $$\hat{\mathbb{Z}} = \{\mathsf{e}, \top\}$$

   where each abstract value denotes a set of integers:

   $$\gamma(\mathsf{e}) = \{n \in \mathbb{Z} \mid n \text{ is even}\}, \qquad \gamma(\top) = \mathbb{Z}$$

   The analysis can be defined by the abstract version $\hat{\mathcal{V}} : e \rightarrow \hat{\mathbb{Z}}$ of the concrete evaluation function. For example,

   - $\hat{\mathcal{V}}(2 + 4) = \mathsf{e}$
   - $\hat{\mathcal{V}}(-1 \diamond 1) = \top$
   - $\hat{\mathcal{V}}(2 + (-2 \diamond 2)) = \mathsf{e}$
   - $\hat{\mathcal{V}}((2 \diamond 3)^*) = \top$

   Define $\hat{\mathcal{V}}$ for this even-number analysis. Note that your analysis must be safe and must terminate for all expressions. An analysis $\hat{\mathcal{V}}$ is safe iff it over-approximates the concrete semantics $\mathcal{V}$, i.e., for all e, $\mathcal{V}(e) \subseteq \gamma(\hat{\mathcal{V}}(e))$.

   $$\begin{array}{rcl} \hat{\mathcal{V}}(n) & = & \begin{cases} \mathsf{e} & n \text{ is even} \\ \top & \text{o.w.} \end{cases} \\ \hat{\mathcal{V}}(-e) & = & \begin{cases} \mathsf{e} & \hat{\mathcal{V}}(e) = \mathsf{e} \\ \top & \text{o.w.} \end{cases} \\ \hat{\mathcal{V}}(e_1 + e_2) & = & \begin{cases} \mathsf{e} & \hat{\mathcal{V}}(e_1) = \mathsf{e} \wedge \hat{\mathcal{V}}(e_2) = \mathsf{e} \\ \top & \text{o.w.} \end{cases} \\ \hat{\mathcal{V}}(e_1 - e_2) & = & \begin{cases} \mathsf{e} & \hat{\mathcal{V}}(e_1) = \mathsf{e} \wedge \hat{\mathcal{V}}(e_2) = \mathsf{e} \\ \top & \text{o.w.} \end{cases} \\ \hat{\mathcal{V}}(e_1 \diamond e_2) & = & \begin{cases} \mathsf{e} & \hat{\mathcal{V}}(e_1) = \mathsf{e} \wedge \hat{\mathcal{V}}(e_2) = \mathsf{e} \\ \top & \text{o.w.} \end{cases} \\ \hat{\mathcal{V}}(e^*) & = & \begin{cases} \mathsf{e} & \hat{\mathcal{V}}(e) = \mathsf{e} \\ \top & \text{o.w.} \end{cases} \end{array}$$

2. (10pts) The previous analysis is too imprecise to find out even some trivial properties. For example, even though expression $1 + 3$ definitely evaluates to an even number (4), the analysis fails to capture this property, i.e., $\hat{\mathcal{V}}(1+3) = \top$. Propose a new set of abstract values and design the corresponding analysis that fix this problem. Use the abstract values:

   $$\hat{\mathbb{Z}} = \{\mathsf{e}, \mathsf{o}, \top\}$$

   where e, o, $\top$ denote even, odd, all integers, respectively. It is easy to define the corresponding $\hat{\mathcal{V}}$ for this domain.

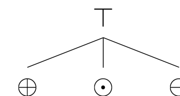
3. (10pts) The next analysis focuses on finding out possible signs of expressions. This *sign analysis* evaluates the expressions in terms of their signs. The analysis uses four abstract values:

   $$\hat{\mathbb{Z}} = \{\oplus, \odot, \ominus, \top\}$$

   where $\oplus$, $\odot$, $\ominus$, $\top$ denote the integer sets:

   $$\begin{array}{ll} \gamma(\oplus) = \{n \in \mathbb{Z} \mid n > 0\} & \gamma(\odot) = \{0\} \\ \gamma(\ominus) = \{n \in \mathbb{Z} \mid n < 0\} & \gamma(\top) = \mathbb{Z} \end{array}$$

   Note that this set is partially ordered as depicted as follows:

   Define $\hat{\mathcal{V}} : e \rightarrow \hat{\mathbb{Z}}$ for the sign analysis. For example,

   - $\hat{\mathcal{V}}(1 + 3) = \oplus$
   - $\hat{\mathcal{V}}(-1 \diamond -2) = \ominus$
   - $\hat{\mathcal{V}}(2 + (-2 \diamond 2)) = \top$
   - $\hat{\mathcal{V}}((2 \diamond 3)^*) = \oplus$.

$$
\begin{aligned}
\mathcal{V}(n) &= \begin{cases} \oplus & n > 0 \\ \ominus & n < 0 \\ \odot & n = 0 \end{cases} \\
\mathcal{V}(-e) &= \begin{cases} \oplus & \mathcal{V}(e) = \ominus \\ \ominus & \mathcal{V}(e) = \oplus \\ \odot & \mathcal{V}(e) = \odot \\ \top & \mathcal{V}(e) = \top \end{cases} \\
\mathcal{V}(e_1 + e_2) &= \begin{cases} \oplus & \hat{\mathcal{V}}(e_1) = \oplus \wedge \hat{\mathcal{V}}(e_2) = \oplus \\ \oplus & \hat{\mathcal{V}}(e_1) = \oplus \wedge \hat{\mathcal{V}}(e_2) = \odot \\ \oplus & \hat{\mathcal{V}}(e_1) = \odot \wedge \hat{\mathcal{V}}(e_2) = \oplus \\ \ominus & \hat{\mathcal{V}}(e_1) = \ominus \wedge \hat{\mathcal{V}}(e_2) = \ominus \\ \ominus & \hat{\mathcal{V}}(e_1) = \ominus \wedge \hat{\mathcal{V}}(e_2) = \odot \\ \ominus & \hat{\mathcal{V}}(e_1) = \odot \wedge \hat{\mathcal{V}}(e_2) = \ominus \\ \odot & \hat{\mathcal{V}}(e_1) = \odot \wedge \hat{\mathcal{V}}(e_2) = \odot \\ \top & \text{otherwise} \end{cases} \\
\hat{\mathcal{V}}(e_1 \diamond e_2) &= \begin{cases} \oplus & \hat{\mathcal{V}}(e_1) = \hat{\mathcal{V}}(e_2) = \oplus \\ \ominus & \hat{\mathcal{V}}(e_1) = \hat{\mathcal{V}}(e_2) = \ominus \\ \odot & \hat{\mathcal{V}}(e_1) = \hat{\mathcal{V}}(e_2) = \odot \\ \top & \text{otherwise} \end{cases} \\
\hat{\mathcal{V}}(e^*) &= \begin{cases} \oplus & \hat{\mathcal{V}}(e) = \oplus \\ \odot & \hat{\mathcal{V}}(e) = \odot \\ \top & \text{otherwise} \end{cases}
\end{aligned}
$$

4. (10pts) The last instance is the interval analysis that finds out the possible ranges of values that a given expression produces. The analysis uses the abstract values:

   $$\hat{\mathbb{Z}} = \{[l, u] \mid l \in \mathbb{Z} \cup \{-\infty\} \ \wedge \ u \in \mathbb{Z} \cup \{+\infty\} \ \wedge \ l \leq u\}$$

   Define $\hat{\mathcal{V}} : e \to \hat{\mathbb{Z}}$ for the interval analysis. For example,

   - $\hat{\mathcal{V}}(1 + 3) = [4, 4]$
   - $\hat{\mathcal{V}}(-1 \diamond -2) = [-2, -1]$
   - $\hat{\mathcal{V}}(2 + (-2 \diamond 2)) = [0, 4]$
   - $\hat{\mathcal{V}}((2 \diamond 3)^*) = [2, +\infty]$
   - $\hat{\mathcal{V}}((-2 \diamond 2)^*) = [-\infty, +\infty]$

$$
\begin{aligned}
\hat{\mathcal{V}}(n) &= [n, n] & & \\
\hat{\mathcal{V}}(-e) &= [-u, -l] & & \hat{\mathcal{V}}(e) = [l, u] \\
\hat{\mathcal{V}}(e_1 + e_2) &= [l_1 + l_2, u_1 + u_2] & & \hat{\mathcal{V}}(e_i) = [l_i, u_i] \ (i = 1, 2) \\
\hat{\mathcal{V}}(e_1 \diamond e_2) &= [min(l_1, l_2), max(u_1, u_2)] & & \hat{\mathcal{V}}(e_i) = [l_i, u_i] \ (i = 1, 2) \\
\hat{\mathcal{V}}(e^*) &= [l < 0 ? -\infty, l, +\infty] & & \hat{\mathcal{V}}(e) = [l, u]
\end{aligned}
$$

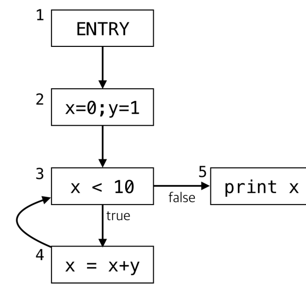
Problem 2 (Data-Flow Analysis, 15pts) Illustrate the following data-flow analyses and their applications to compiler optimization.

1. Reaching definitions analysis
2. Available expressions analysis
3. Constant propagation analysis

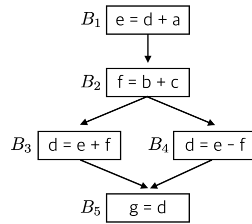
Problem 3 (Use of Liveness Analysis, 10pts) Once the compiler has computed the liveness information for each block in the procedure's control-flow graph, we can find uses of variables that are potentially uninitialized.

1. (5pts) Explain how to detect uninitialized variables using the liveness information.
2. (5pts) This approach may yield false positives; the compiler may report a use of a variable as uninitialized even when it is always initialized at run-time. Discuss when/where false positives may occur.

Problem 4 (Interval Analysis, 15pts) Illustrate the interval analysis for the program below. How does the analysis work? How does the fixed point get computed? What is the final outcome of the analysis? etc.



Problem 5 (Register Allocation, 10pts) Consider the program:



1. (5pts) Compute the live-in and live-out sets for each basic block:

   $\text{live-in}(B_1) = \{a, b, c, d\}$ $\text{live-out}(B_1) = \{b, c, e\}$
   $\text{live-in}(B_2) = \{b, c, e\}$ $\text{live-out}(B_2) = \{e, f\}$
   $\text{live-in}(B_3) = \{e, f\}$ $\text{live-out}(B_3) = \{d\}$
   $\text{live-in}(B_4) = \{e, f\}$ $\text{live-out}(B_4) = \{d\}$
   $\text{live-in}(B_5) = \{d\}$ $\text{live-out}(B_5) = \{g\}$

2. (5pts) Construct the register interference graph and find an assignment of 4 registers to variables.

Problem 6 (10pts) What is a compiler? Illustrate compilers from the perspectives of both language translators and program synthesizers.