Exercise 1: Monotonicity and Distributivity

Recall the lattice $\mathbb{Z}^\top = \mathbb{Z} \cup \{\bot, \top\}$ with the ordering "='. Determine for each of the following functions $D \rightarrow D$, whether it is monotonic, distributive, and strict.

\[
\begin{align*}
\text{bot}(x) &= \bot \\
\text{top}(x) &= \top \\
\text{zero}(x) &= \begin{cases} 
\top & x = \top \\
1 & x = 0 \\
0 & \text{otherwise} 
\end{cases} \\
\text{exp}(x) &= \begin{cases} 
\top & x = \top \\
2^x & x \geq 0 \\
\bot & \text{otherwise} 
\end{cases} \\
\text{inv}(x) &= \begin{cases} 
\top & x = \top \\
-x & x \in \mathbb{Z} \\
\bot & x = \bot 
\end{cases} \\
\text{sq}(x) &= \begin{cases} 
\top & x = \top \\
x^2 & x \in \mathbb{Z} \\
\bot & x = \bot 
\end{cases}
\end{align*}
\]

Solution: By definition the function $f: D_1 \rightarrow D_2$ is:

- monotonic if $\forall x, y \in D$ such that $x \sqsubseteq y f(x) \sqsubseteq f(y)$.
- strict if $f(\bot) = \bot$
- distributive if $f(\sqcup X) = \sqcup \{f(x) | x \in X\}$ for all $\emptyset \neq X \subseteq D$

Applying the definitions to the functions we obtain:

1. $\text{bot}(x)$ is strict: $\text{bot}(\bot) = \bot$, monotonic and distributive, as it is a constant function
2. $\text{top}(x)$ is not strict: $\text{top}(\bot) = \top$, is monotonic and distributive analogously to $\text{bot}(x)$
3. $\text{zero}(x)$ is
   - not strict, $\text{zero}(\bot) = 0 \neq \bot$
   - not monotonic, counterexample: $\bot \sqsubseteq 0$, but $\text{zero}(\bot) = 0 \not\sqsubseteq 1 = \text{zero}(0)$
   - not distributive, counterexample: $\text{zero}(1) \sqcup \text{zero}(2) = 0 \sqcup 0 = 0$, but $\text{zero}(1 \sqcup 2) = \text{zero}(\top) = \top, 0 \neq \top$
4. $\text{exp}(x)$ is
• strict: \(\exp(\bot) = \bot\)

• monotonic, we check relations between the following subsets: \(\bot, \top\) and one of these \(Neg = \{x \in \mathbb{Z} | x < 0\}, Pos = \{x \in \mathbb{Z} | x \geq 0\}\).

\(\forall x \in Neg: \bot \subseteq x\) and \(\exp(\bot) = \bot \subseteq \bot = \exp(x), x \subseteq \top\) and \(\exp(x) = \bot \subseteq \top = \exp(\top),\)

\(\forall x, y = 2^x \in Pos: \bot \subseteq x\) and \(\exp(\bot) = \bot \subseteq y = \exp(x), x \subseteq \top\) and \(\exp(x) = y \subseteq \top = \exp(\top),\)

• not distributive, counterexample: \(\exp(-1) \sqcup \exp(0) = \bot \sqcup 1 = 1\), but \(\exp(-1 \sqcup 0) = \exp(\top) = \top, 1 \neq \top\)

5. \(inv(x)\) is

• strict: \(inv(\bot) = \bot\)

• monotonic, \(\forall x \in \mathbb{Z}\) holds \(\bot \subseteq x\) and \(inv(\bot) = \bot \subseteq -x = inv(x); x \subseteq \top\) and \(inv(x) = -x \subseteq \top = inv(\top)\)

• distributive, \(\forall x, y \in \mathbb{Z}\) holds:

\(inv(\bot) \sqcup inv(x) = \bot \sqcup -x = -x\) and \(inv(\bot \sqcup x) = inv(x) = -x, -x = -x\)

\(inv(x) \sqcup inv(y) = -x \sqcup -y = \top\) and \(inv(x \sqcup y) = inv(\top) = \top, \top = \top\)

\(inv(x) \sqcup inv(\top) = -x \sqcup \top = \top\) and \(inv(x \sqcup \top) = inv(\top) = \top, \top = \top\)

6. \(sq(x)\) is

• strict: \(sq(\bot) = \bot\)

• monotonic, \(\forall x, y = x^2 \in \mathbb{Z}\) holds \(\bot \subseteq x\) and \(sq(\bot) = \bot \subseteq y = sq(x); x \subseteq \top\) and \(sq(x) = y \subseteq \top = sq(\top)\)

• not distributive, counterexample: \(sq(-1) \sqcup sq(1) = 1 \sqcup 1 = 1\), but \(sq(-1 \sqcup 1) = sq(\top) = \top, 1 \neq \top\)

Exercise 2: Liveness vs. True Liveness

Consider the following program fragment:

```plaintext
x = 42;
r = x + 99;
i = 0;
y = 0;
r = 1;
while (i < 3) {
    r = r*n;
y = y + 5;
i = i + 1;
}
```

The corresponding control flow graph looks as follows:
The program fragment computes $n^3$ for a given $n$ and returns the result in the variable $r$.

1. Give the constraint system for calculating the live variable sets. Assume the variable $r$ to be live at the end of the program since it contains the result of the computation and may be read afterwards.

2. Give a solution to the constraint system. Use your solution to apply transformation 2 to the program and give either the resulting CFG or the source code.

3. Give the constraint system for calculating the truly live variable sets. Again assume $r$ to be live at the end of the program.

4. Give a solution to the constraint system for true liveness. Again use your solution to apply transformation 2 to the program and give either the resulting CFG or the source code.

5. Compare the three programs by counting the number of operations executed. One operation is a binary operation ($+$, $-$, $\ast$, $/$), an assignment to a variable, or an comparison ($=$, $\neq$, $<$, $\leq$, $>$, $\geq$). The statement $x = y + 1$ counts as two operations.

6. Instead of using the constraint system for true liveness, one could run the standard liveness analysis multiple times. Would that approach yield the same result as the true liveness for the given program? If not, which of the assignments would not be removed? Reason your answer.
Solution:

1. Constraint system for liveness:

\[
\begin{align*}
\mathcal{L}[0] & \supseteq (\mathcal{L}[1] \setminus \{x\}) \cup \emptyset \\
\mathcal{L}[1] & \supseteq (\mathcal{L}[2] \setminus \{r\}) \cup \{x\} \\
\mathcal{L}[2] & \supseteq (\mathcal{L}[3] \setminus \{i\}) \cup \emptyset \\
\mathcal{L}[3] & \supseteq (\mathcal{L}[4] \setminus \{y\}) \cup \emptyset \\
\mathcal{L}[4] & \supseteq (\mathcal{L}[5] \setminus \{r\}) \cup \emptyset \\
\mathcal{L}[5] & \supseteq \mathcal{L}[9] \cup \{i\} \\
\mathcal{L}[6] & \supseteq \mathcal{L}[6] \cup \{i\} \\
\mathcal{L}[7] & \supseteq (\mathcal{L}[7] \setminus \{r\}) \cup \{r, n\} \\
\mathcal{L}[8] & \supseteq (\mathcal{L}[5] \setminus \{i\}) \cup \{i\} \\
\mathcal{L}[9] & \supseteq \{r\}
\end{align*}
\] 

2. Solution:

\[
\begin{align*}
\mathcal{L}[0] & = \{n\} \\
\mathcal{L}[1] & = \{n, x\} \\
\mathcal{L}[2] & = \{n\} \\
\mathcal{L}[3] & = \{i, n\} \\
\mathcal{L}[4] & = \{i, y, n\} \\
\mathcal{L}[5] & = \{i, r, y, n\} \\
\mathcal{L}[6] & = \{i, r, y, n\} \\
\mathcal{L}[7] & = \{i, r, y, n\} \\
\mathcal{L}[8] & = \{i, r, y, n\} \\
\mathcal{L}[9] & = \{r\}
\end{align*}
\]

Transformed CFG:
Optimized program:

```c
x = 42;
i = 0;
y = 0;
r = 1;
while (i < 3) {
    r = r * n;
    y = y + 5;
    i = i + 1;
}
```
3. Constraint system for true liveness:

\[
\begin{align*}
L[0] & \supseteq (L[1] \setminus \{x\}) \cup (x \in L[1]) \setminus \emptyset : \emptyset \\
L[1] & \supseteq (L[2] \setminus \{r\}) \cup (r \in L[2]) \setminus \emptyset : \emptyset \\
L[2] & \supseteq (L[3] \setminus \{i\}) \cup (i \in L[3]) \setminus \emptyset : \emptyset \\
L[3] & \supseteq (L[4] \setminus \{y\}) \cup (y \in L[4]) \setminus \emptyset : \emptyset \\
L[4] & \supseteq (L[5] \setminus \{r\}) \cup (r \in L[5]) \setminus \emptyset : \emptyset \\
L[5] & \supseteq \{i\} \\
L[5] & \supseteq \{i\} \\
L[6] & \supseteq (L[7] \setminus \{r\}) \cup (r \in L[7]) \setminus \emptyset : \emptyset \\
L[7] & \supseteq (L[8] \setminus \{y\}) \cup (y \in L[8]) \setminus \emptyset : \emptyset \\
L[8] & \supseteq (L[5] \setminus \{i\}) \cup (i \in L[5]) \setminus \emptyset : \emptyset \\
L[9] & \supseteq \{r\}
\end{align*}
\]

4. Solution:

\[
\begin{align*}
L[0] &= \{n\} \\
L[1] &= \{n\} \\
L[2] &= \{n\} \\
L[3] &= \{i, n\} \\
L[4] &= \{i, n\} \\
L[5] &= \{i, r, n\} \\
L[6] &= \{i, r, n\} \\
L[7] &= \{i, r, n\} \\
L[8] &= \{i, r, n\} \\
L[9] &= \{r\}
\end{align*}
\]

Transformed CFG:
Optimized program:

```java
i = 0;
r = 1;
while (i < 3) {
    r = r * n;
    i = i + 1;
}
```

5. • Original program: $6 + 3 \times 7 + 1 = 28$ operations
   • Optimized program (liveness): $4 + 3 \times 7 + 1 = 26$ operations
   • Optimized program (true liveness): $2 + 3 \times 5 + 1 = 18$ operations

6. The approach would not yield the same result. The repeated liveness is able to additionally remove the assignment to $x$, but would keep all code for $y$. This is because the liveness of $y$ is cyclically dependent on itself and, thus, $y$ does not die of incrementally removing dead variables.