Exercise Sheet Solutions 6

Compiler Design Lab
Prof. Dr. Sebastian Hack
Fabian Ritter, B.Sc.

This is a solution proposal. It does not necessarily describe complete solutions but gives the initial ideas to solve the theoretical exercises and presents the results.

Exercise 6.1 Dominance and Data Flow

Dominance is defined for directed graphs as follows:

Let (N, E, s) be a directed graph with nodes N, edges $E \subseteq N \times N$ and a unique starting node $s \in N$ with no incoming edges. A node $n_1 \in N$ *dominates* a node $n_2 \in N$ in (N, E, s), if for every path π from s to n_2, n_1 occurs on π.

Develop a data flow analysis to compute the set of dominators of each block in a control flow graph. The analysis is performed on a control flow graph (CFG), which only consists of basic blocks and control flow edges between them. (The actual instructions do not matter for dominance.) Further, a CFG has a unique start node. A basic block is a maximal sequence of instructions which starts with a label and ends in a conditional or unconditional branch and does not contain any other label or branch. Consider the following aspects:

- What is the domain, which is a complete lattice, of the analysis? In particular, describe $\bot$ and $\top$.
- What is the join operator $\sqcup$?
- If a block A is dominated by a block B (and $A \neq B$), then all direct predecessors of A are also dominated by B.
- Each block dominates itself.
- What does it mean that information is (un-)safe for this analysis?
- Is the analysis performed forwards (along the control flow) or backwards (against the control flow)?
- What is the initialization at each block?
- What is the set of dominators of an unreachable block?

Draw the CFG for the following program and perform dominance analysis. Label each basic block with an uppercase letter.

```
void f(void) {
  if (...) {
    while (...) {
      if (...)
        break;
    }
  } else {
  }
}
```

Solution: Let $\mathcal{B}$ be the set of all basic blocks in the control flow graph.

- As a lattice, we choose $\mathcal{B} \to \mathcal{P}(\mathcal{B})$ with $f \sqsubseteq g$ iff $\forall x \in \mathcal{B}.\ f(x) \supseteq g(x)$ (so we keep for each block a set of definitive dominators). We define $\bot := \lambda x.\ \mathcal{B}$ (meaning every block is dominated by every block) and $\top := \lambda x.\ \varnothing$ (meaning for every block we cannot find any block that dominates it).
- The join operator $\sqcup$ is the element-wise application of set intersection $\cap$:

  $$f \sqcup g := \lambda x.\ f(x) \cap g(x)$$

  Thus, we can combine the information from two locations by performing a set intersection operation.

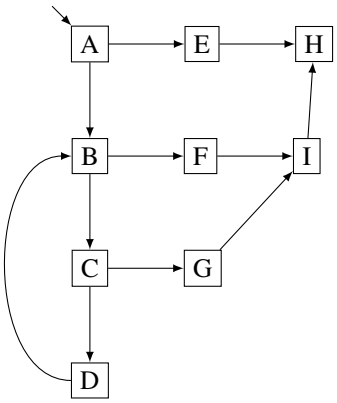
- When constructing a constraint system for this analysis, we have to add the following inequalities:

$$S[i] \subseteq \{i\} \cup \bigcap_{(j,i) \in E} S[j] \qquad \text{for all } i \in \mathcal{B}$$

  (i.e. every block is dominated by itself and by the common dominators of all of its predecessors.)

- Information is unsafe if it claims that a block A dominates a block B although it does not.
- We perform the analysis forwards (i.e the value for a block depends on the values of its predecessors in the CFG).
- We initialize the analysis with $\bot$, therefore we have for each block the set $\mathcal{B}$ of all blocks.
- When a block A is unreachable, there exists no path from the start node to A. Therefore, the dominance condition contains a $\forall$-quantification over the empty set which implies that every block B dominates A.

The CFG looks as follows:



We get the following system of inequations:

$$\begin{aligned}
S[A] &\supseteq \{A\} \\
S[B] &\supseteq \{B\} \cup (S[A] \cap S[D]) \\
S[C] &\supseteq \{C\} \cup S[B] \\
S[D] &\supseteq \{D\} \cup S[C] \\
S[E] &\supseteq \{E\} \cup S[A] \\
S[F] &\supseteq \{F\} \cup S[B] \\
S[G] &\supseteq \{G\} \cup S[C] \\
S[H] &\supseteq \{H\} \cup (S[E] \cap S[I]) \\
S[I] &\supseteq \{I\} \cup (S[F] \cap S[G])
\end{aligned}$$

We solve this system in a round-robin fashion:

#	$S[A]$	$S[B]$	$S[C]$	$S[D]$	$S[E]$	$S[F]$	$S[G]$	$S[H]$	$S[I]$
0	$\mathcal{B}$	$\mathcal{B}$	$\mathcal{B}$	$\mathcal{B}$	$\mathcal{B}$	$\mathcal{B}$	$\mathcal{B}$	$\mathcal{B}$	$\mathcal{B}$
1	$\{A\}$	$\{A,B\}$	$\{A,B,C\}$	$\{A,B,C,D\}$	$\{A,E\}$	$\{A,B,F\}$	$\{A,B,C,G\}$	$\{A,H\}$	$\{A,B,I\}$
2	$\{A\}$	$\{A,B\}$	$\{A,B,C\}$	$\{A,B,C,D\}$	$\{A,E\}$	$\{A,B,F\}$	$\{A,B,C,G\}$	$\{A,H\}$	$\{A,B,I\}$

As the values did not change in the second iteration, we obtained a sound result.

Exercise 6.2 Chaotic Iteration

For computing the least fixed-point of the abstract semantics of a program, we have shown the correctness of the Kleene iteration scheme. Following this scheme, if we have an analysis that computes information for each program point, we have to apply the transformers for all program locations in parallel in each step (starting from $\bot^n$).

In the lecture, we claimed that we can instead apply transformers for individual program locations independently in an arbitrary order and still obtain the same least fixed-point as Kleene iteration. This exercise proves the validity of this claim.

First some definitions and notations:
Let $(L, \sqsubseteq)$ be a complete lattice with no infinite ascending chains. We extend the domain of $\sqsubseteq$ to n-tuples of elements of L in a straight-forward way: $\underline{a} \sqsubseteq \underline{b} \equiv \forall i \in \{1, \ldots, n\}.\ a_i \sqsubseteq b_i$.

Let $F : L^n \to L^n$ with $F = (f_1, \ldots, f_n)$. Let further all $f_i : L^n \to L$ be monotone. (From this also follows that F is monotone.) For convenience, we lift the component functions f_i to return n-tuples:

$$\widetilde{f_i} : L^n \to L^n,\ \widetilde{f_i}(\underline{x}) = (x_1, \ldots, x_{i-1}, f_i(\underline{x}), x_{i+1}, \ldots, x_n)$$

Note that the monotonicity of the $\widetilde{f_i}$ follows from the monotonicity of the f_i.

We add a shorthand notation for compositions of these functions: Let $\sigma \in \{1, \ldots, n\}^*$ be a sequence of indices. We define:

$$\widetilde{f_\sigma} = \begin{cases} \lambda x.\ x & \text{if } \sigma = \varepsilon \\ \widetilde{f_k} \circ \widetilde{f_{\sigma'}} & \text{if } \sigma = k\sigma' \end{cases}$$

Proceed as follows to prove the correctness of the chaotic iteration scheme:

1. Prove that for any sequence ρ of indices, the intermediate results of $\widetilde{f_\rho}$ form an ascending chain:

   $$\forall \sigma \in \{1, \ldots, n\}^*.\ \forall k \in \{1, \ldots, n\}.\ \widetilde{f_\sigma}(\bot^n) \sqsubseteq \widetilde{f_{k\sigma}}(\bot^n)$$

   Hint: Use an induction over the length of σ.

2. When performing our chaotic iteration, we apply component transformers $\widetilde{f_i}$ to $\bot^n$ until no application of any $\widetilde{f_i}$ can change the the result anymore. Argue that chaotic iteration terminates and computes a fixed-point of F.

3. Prove that the fixed-point x of F that we compute via chaotic iteration is the least fixed-point of F.
   Hint: Compare the intermediate values of applying $\widetilde{f_\sigma}(\bot^n)$ to those of applying $F^{|\sigma|}(\bot^n)$.

Solution:

1. **Induction base** $\sigma = \varepsilon$.
   Let $k \in \{1, \ldots, n\}$. By the definition of $\bot$, we know:

   $$\widetilde{f_\varepsilon}(\bot^n) = \bot^n \sqsubseteq \widetilde{f_{k\varepsilon}}(\bot^n)$$

   Induction hypothesis Let $\forall k \in \{1, \ldots, n\}.\ \widetilde{f_\sigma}(\bot^n) \sqsubseteq \widetilde{f_{k\sigma}}(\bot^n)$ hold for all index sequences $\sigma \in \{1, \ldots, n\}^{\leq s}$ with a length of at most s for some $s \in \mathbb{N}$.

   Induction step Let $\sigma \in \{1, \ldots, n\}^{s+1}$ and $k \in \{1, \ldots, n\}$.
   We need to show that $\widetilde{f_\sigma}(\bot^n) \sqsubseteq \widetilde{f_{k\sigma}}(\bot^n)$. Destruct $l\sigma' = \sigma$ with $l \in \{1, \ldots, n\}$ and $|\sigma'| = s$.
   We distinguish two cases:

   Case 1 $l = k$
   By the induction hypothesis, we know that $\widetilde{f_{\sigma'}}(\bot^n) \sqsubseteq \widetilde{f_{l\sigma'}}(\bot^n)$.
   By the monotonicity of $\widetilde{f_l}$, we can conclude that $\widetilde{f_l}(\widetilde{f_{\sigma'}}(\bot^n)) \sqsubseteq \widetilde{f_l}(\widetilde{f_{l\sigma'}}(\bot^n)) = \widetilde{f_k}(\widetilde{f_{l\sigma'}}(\bot^n))$.
   Hence, $\widetilde{f_{\sigma'}}(\bot^n) \sqsubseteq \widetilde{f_{l\sigma'}}(\bot^n) \sqsubseteq \widetilde{f_{kl\sigma'}}(\bot^n)$, which proves our claim.

Case 2 $l \neq k$

Let $\underline{v} := \widetilde{f}_{\sigma'}(\bot^n)$. We need to prove that

$$\widetilde{f}_l(\underline{v}) \sqsubseteq \widetilde{f}_k(\widetilde{f}_l(\underline{v}))$$

which we can expand to

$$(v_1, \ldots, v_{l-1}, f_l(\underline{v}), v_{l+1}, \ldots, v_n) \sqsubseteq (v_1, \ldots, v_{l-1}, f_l(\underline{v}), v_{l+1}, \ldots, v_{k-1}, f_k(\widetilde{f}_l(\underline{v})), v_{k+1}, \ldots, v_n)$$

We see that the only component which differs between the two sides of the above inequality is the kth, hence we only need to prove that $v_k \sqsubseteq f_k(\widetilde{f}_l(\underline{v}))$. We prove this as follows:
From the induction hypothesis, we know that

$$\underline{v} \sqsubseteq \widetilde{f}_l(\underline{v})$$

As f_k is monotone, we can conclude that

$$f_k(\underline{v}) \sqsubseteq f_k(\widetilde{f}_l(\underline{v}))$$

The induction hypothesis also gives us

$$\underline{v} \sqsubseteq \widetilde{f}_k(\underline{v})$$

which implies

$$v_k \sqsubseteq f_k(\underline{v})$$

As a result, we find

$$v_k \sqsubseteq f_k(\underline{v}) \sqsubseteq f_k(\widetilde{f}_l(\underline{v}))$$

which proves our claim.

□

2. Since the intermediate results of our chaotic iteration form an ascending chain and our complete lattice does not contain any infinite ascending chains, we will eventually reach a value for which no continuation of the sequence so far can changes the value anymore. Thus, the algorithm terminates. Assume its result value $\underline{x}^*$ was not a fixed-point of F. This means that $F(\underline{x}^*) \neq \underline{x}^*$. Therefore the two values have to differ in at least one component i which would imply that $f_i(\underline{x}) \neq x_i$ which contradicts our assumption.

3. We show that $\forall i \in \mathbb{N}.\ \forall \sigma \in \{1, \ldots, n\}^i.\ \widetilde{f}_\sigma(\bot^n) \sqsubseteq F^i(\bot^n)$ by induction over i.

   Induction base $i = 0$, $\sigma = \varepsilon$

   $$\widetilde{f}_\varepsilon(\bot^n) = \bot^n \sqsubseteq F^0(\bot^n)$$

   Induction hypothesis Let $\forall \sigma \in \{1, \ldots, n\}^i.\ \widetilde{f}_\sigma(\bot^n) \sqsubseteq F^i(\bot^n)$ hold for some $i \in \mathbb{N}$.

   Induction step Let $i' = i + 1$, $\sigma' = l\sigma$.

   By the induction hypothesis, we know that

   $$\widetilde{f}_\sigma(\bot^n) \sqsubseteq F^i(\bot^n)$$

   As $\widetilde{f}_k$ is monotone, we can conclude that

   $$\widetilde{f}_{k\sigma}(\bot^n) \sqsubseteq \widetilde{f}_k(F^i(\bot^n))$$

   The proof of Kleene's theorem provides that $F^i(\bot^n) \sqsubseteq F^{i+1}(\bot^n)$. As $\widetilde{f}_k(F^i(\bot^n))$ only has components that also occur in $F^i(\bot^n)$ or in $F^{i+1}(\bot^n)$, we know that $\widetilde{f}_k(F^i(\bot^n)) \sqsubseteq F^{i+1}(\bot^n)$. Therefore, we can conclude that $\widetilde{f}_{k\sigma}(\bot^n) \sqsubseteq F^{i+1}(\bot^n)$, which proves our claim.

   So we know that after s steps, chaotic iteration will reach a fixed-point of F. By Kleene's theorem, Kleene iteration will reach the least fixed-point of F after t steps. Therefore, after $\max(s, t)$ steps, each algorithm will have reached a fixed-point of F, and the one produced by chaotic iteration is smaller than or equal to the least fixed-point produced by Kleene iteration, hence the two fixed-points are equal. □

Project task E Semantic Analysis

Implement semantic analysis.

- You can perform this either during parsing and AST construction or as a separate phase.
- Semantic analysis augments `--parse` and `--print-ast`.
- Major parts are name and type analysis. Name analysis associates identifiers with declarations. Type analysis associates expressions with types. It encompasses the Constraints and Semantics clauses.
- If you delayed certain syntactic checks (e.g. rejecting `a || b = c`), perform them now.
- The only *null pointer constant*, which you need to support, is literal `0`.
- The previous restriction and the restricted language subset ensure that it is not necessary to evaluate the value of any expressions during semantic analysis.
- It is not necessary to accept programs which contain functions that return a struct or have one as parameter. Similarily, it is not necessary to accept programs which contain assignments of struct type.
- It is not necessary to accept programs which contain anonymous structs.
- You do not need to handle `__func__`.
- Use `int` for the types `ptrdiff_t` and `size_t`.
- Due to the restricted language subset, type compatibility degenerates to equality.
- For the error location use the location of the (first) terminal of the syntactic construct where the error was detected. E.g. for adding two pointers, show the location of the `+`. For an `if` whose condition is not scalar, show the location of the keyword `if`.
- If you are uncertain about some aspect, ask!
- The soft deadline for this milestone is 2017-12-22.
- Keep it simple!