Project 2 – Interval analysis by abstract interpretation

Static Program Analysis and Constraint Solving
Master's Degree in Formal Methods in Computer Science
Year 2019/20

This project on static analysis involves implementing an analysis in a programming language of your choice and proving some theoretical results related to this analysis. You can assume that the abstract syntax tree (AST) of the program to be analysed is available. You are not required to implement a parser.

Grading: The project accounts for 45% of the overall score of the course. It will be graded from 0 to 10 on the basis of the work done (each section in this document contains a maximum score to be awarded) and its defence after submission. 30% of the project score will be given on the basis of the latter.

Submission instructions: Students are required to submit two files:

- A `.zip` or `.tgz` file with the implementation of the analysis and a `README` file containing instructions on how to compile and run the implementation.
- A `.pdf` file with the theoretical results (rules, definitions, proofs) required in each section. Students may write these in Spanish, but handwritten submissions are **not** allowed this time.

This project has two goals: (1) implement the abstract interpretation-based interval analysis explained in Lesson 4, and (2) establish how interval analysis is related to constant propagation analysis. We shall work, as usual, with our *While* language:

$$
\begin{array}{lcl}
e & ::= & n \mid x \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \\
b & ::= & e_1 = e_2 \mid e_1 \leq e_2 \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \\
S & ::= & \texttt{skip} \mid x := e \mid S_1; S_2 \mid \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2 \mid \texttt{while } b \texttt{ do } S
\end{array}
$$

1 Obtaining a control-flow graph (CFG) [2 points]

Since our analysis is data flow-based, our first step is to transform the abstract syntax tree of our program into a control-flow graph. Each vertex in this graph is a block that may contain a `skip` statement, an assignment or a conditional expression.

Implement a method or function that, given a statement, computes its CFG. It is highly recommended that your implementation does not only return the graph, but also a reference to the initial block in that graph, and a set of references to its (possibly many) final blocks. Thus you will be able to build the CFG in a compositional way (i.e. following the recursive structure of the statement being transformed).

2 Interval analysis [4 points]

Once we have obtained a CFG from our input program, we can apply the interval analysis explained in Lesson 4. This analysis determines, at each program point, an abstract state in $\mathbf{State}^\sharp = \mathbf{Var} \to \mathbf{Interval}$ that maps every program variable to an interval containing the possible values that it may take at runtime. Implementing this analysis involves the following steps:

1. Implement an **Interval** abstract data type, including its associated $\sqcup$ and $\sqcap$ operators.
2. Implement an abstract interpreter $[\![e]\!]^\sharp : \mathbf{State}^\sharp \to \mathbf{Interval}$ that yields the interval of values which an expression may evaluate to, given the interval of each variable occurring in e.
3. Implement an abstract transfer function for each kind of CFG block (`skip`, assignment, and conditional).
4. With the help of the functions implemented in the previous step, implement the procedure that computes Kleene's ascending chain until a fixed point is reached, or until a given number of iterations have been performed.

   Note: In this step you do not have to take boolean conditions into account, as we did in the lectures (that is, you do not have to apply $\sqcap$ operator in the transfer functions). If a CFG block n contains a boolean condition, we can assume that its transfer function is $f_n^\sharp(\sigma^\sharp) = \sigma^\sharp$.
5. Introduce the widening operator $\diamond$ on intervals explained in Lesson 4. Assume that K is the set of constants appearing in the program. Modify Kleene's ascending chain with this operator to ensure stabilization of the chain.

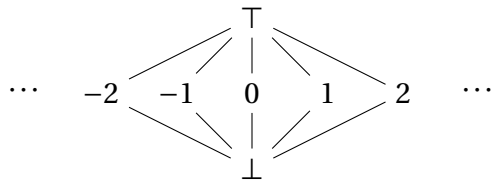
3 Constant propagation analysis [4 points]

It is not difficult to become convinced that a constant propagation analysis (as we did in Assignment 3) is a particularization of an interval analysis. Indeed, if our interval analysis determines at a given point that the value of a variable x lies within an interval $[k, k]$ for some $k \in \mathbb{Z}$, we know that x is constant at that point. In this section we use the abstract interpretation framework in order to derive a constant propagation analysis from the interval analysis.

In this section, **Interval** will be our concrete domain, whereas $\mathbb{Z}_\bot^\top$ will be our abstract domain. The order relation $\sqsubseteq$ on $\mathbb{Z}_\bot^\top$ is defined as in Assignment 3:

$$\forall\, t_1, t_2 \in \mathbb{Z}_\bot^\top. \quad t_1 \sqsubseteq t_2 \iff t_1 = \bot \vee t_2 = \top \vee t_1 = t_2$$

This relation can be depicted by the following Hasse diagram:



1. Define an abstraction function and a concretization function between **Interval** and $\mathbb{Z}_{\perp}^{\top}$,

$$\begin{aligned} \alpha &: \quad \textbf{Interval} \to \mathbb{Z}_{\perp}^{\top} \\ \gamma &: \quad \mathbb{Z}_{\perp}^{\top} \to \textbf{Interval} \end{aligned}$$

   and prove that $(\textbf{Interval}, \alpha, \gamma, \mathbb{Z}_{\perp}^{\top})$ is a Galois connection.

2. Extend this Galois connection to functions on program variables,

$$(\underbrace{\textbf{Var} \to \textbf{Interval}}_{\textbf{State}^{\sharp}}, \alpha', \gamma', \underbrace{\textbf{Var} \to \mathbb{Z}_{\perp}^{\top}}_{\textbf{State}^{\sharp\sharp}})$$

   and define α' and γ' in the standard way.

3. Define an abstract interpreter $[\![e]\!]^{\sharp\sharp} : \textbf{State}^{\sharp\sharp} \to \mathbb{Z}_{\perp}^{\top}$ that determines whether the result of an expression *must* be constant at runtime.

4. Finally, show that the interpreter is correct by proving that $[\![e]\!]^{\sharp\sharp} \sqsupseteq \alpha \circ [\![e]\!]^{\sharp} \circ \gamma'$, where $[\![e]\!]^{\sharp}$ is the abstract interpretation in the lattice of intervals.