6.820: FOUNDATIONS OF PROGRAM ANALYSIS
ABSTRACT INTERPRETATION

JEAN YANG

1. ABSTRACT INTERPRETATION

Sound approximation of semantics basic on monotonic functions over ordered sets (lattices).

1.1. How it works. Map a concrete set \( L \) to an abstract set \( L' \).

- Abstraction function \( \alpha \) for mapping \( x \in L \) to \( \alpha(x) \in L' \). Use this to perform interpretation.
- Concretization function \( \gamma \) mapping \( x' \in L' \) to \( \gamma(x') \in L \). Need this to determine if abstraction is sound.

Also useful in practice for getting concrete counterexamples, etc.

Partially execute the program to project out a set of values we care about. Iterate until the set of values converges.

Suppose we have ordered sets \( L_1, L_2, L'_1 \), and \( L'_2 \). A concrete semantics \( f \) is a monotonic function from \( L_1 \) to \( L_2 \). A function \( f' \) is a valid abstraction of \( f \) if for all \( x' \in L'_1 \), \( (f \circ \gamma)(x') \leq (\gamma \circ f')(x') \).

1.2. Examples of abstract domains.

- Integer intervals.
- Relational integer abstract domains: congruence on relations on integers, convex polyhedra, difference-bound matrices, linear equalities, etc.

1.3. Applications. Static analysis tools for bug finding.

2. INTRODUCTION TO OCAML

2.1. Basics. Functional language with Hindley-Milner polymorphism and type inference. Additionally has:

- A module system—lets you package and export a namespace of values (variables and functions).
- Imperative aspects: references and loops.
- Optional whitespace sensitivity.

2.2. Hello world. The entry point is the \( \text{main : unit } \rightarrow \text{unit} \) function.

\begin{verbatim}
let main () = print_string "Hello world!\n"
\end{verbatim}

3. INTRODUCTION TO CIL

George Necula’s C Intermediate Language for analyzing C programs. It is a highly structured, “clean” subset of C that organized C’s imperative features into expressions, instructions, and statements based on the presence of side effects and control flow.

One of the most useful tools is the visitor pattern implementation. The visiting engine the CIL program depth-first and at each node queries the user-provided function whether to do ignore, descend, and/or replace the subtree.

3.1. AST of intermediate language. This is in \text{cil.mli}, which contains the definitions for the AST.

Expressions:

\begin{verbatim}
and exp =
  Const of constant
| Lval of lval
| SizeOf of typ
| SizeOfE of exp
| SizeOfStr of string
| AlignOf of typ
| AlignOfE of exp
\end{verbatim}
UnOp of unop * exp * typ
BinOp of binop * exp * exp * typ
CastE of typ * exp
AddrOf of lval
StartOf of lval

Statements:

and stmt = {
    mutable labels: label list;
    mutable skind: stmtkind;
    mutable sid: int;
    mutable succs: stmt list;
    mutable preds: stmt list;
}

and stmtkind =
    I n s t r of instr list
    Return of exp option * location
    Goto of stmt ref * location
    Break of location
    Continue of location
    If of exp * block * block * location
    Switch of exp * block * (stmt list) * location
    Loop of block * location * (stmt option) * (stmt option)
    Block of block
    TryFinally of block * block * location
    TryExcept of block * (instr list * exp) * block * location

Instructions:

and instr =
    Set of lval * exp * location
    Call of lval option * exp * exp list * location
    Asm of attributes *
        string list *
        (string option * string * lval) list *
        (string option * string * exp) list *
        string list
        location