**MS Patch Tuesday**
- “eEye Digital Security has reported a vulnerability in Windows Media Player ... due to a boundary error within the processing of bitmap files (.bmp) and can be exploited to cause a heap-based buffer overflow via a specially crafted bitmap file that declares its size as 0 ... exploitation allows execution of arbitrary code”
- Six of seven “critical” or “important” bugs were found by people outside of Microsoft

**Apologies to Ralph Macchio**
- Daniel: You’re supposed to teach and I’m supposed to learn. Four homeworks I’ve been working on IMP, I haven’t learned a thing.
- Miyagi: You learn plenty.
- Daniel: I learn plenty, yeah. I learned how to analyze IMP, maybe. I evaluate your commands, derive your judgments, prove your soundness. I learn plenty!
- Miyagi: Not everything is as seems.
- Daniel: You’re not even relatively complete! I’m going home, man.
- Miyagi: Daniel-san!
- Daniel: What?
- Miyagi: Come here. Show me “compute the VC”.

**Homework**
- Exciting, practical HW 5 out today
- If you’ve been skiving, now is a great time to try one out
- Easily applicable to other research

**Abstract Interpretation (Non-Standard Semantics)**
- “Picking The Right Abstraction”

**The Problem**
- It is extremely useful to predict program behavior **statically** (= without running the program)
  - For optimizing compilers, program analyses, software engineering tools, finding security flaws, etc.
- The semantics we studied so far give us the precise behavior of a program
- However, precise static predictions are impossible
  - The exact semantics is not computable
- We must settle for approximate, but correct, static analyses (e.g. VC vs. WP)

**The Plan**
- We will introduce **abstract interpretation** by example
- Starting with a miniscule language we will build up to a fairly realistic application
- Along the way we will see most of the ideas and difficulties that arise in a big class of applications
A Tiny Language

- Consider the following language of arithmetic (“shrIMP?”)
  \[ e ::= n \mid e_1 \ast e_2 \]
- The denotational semantics of this language
  \[ [n] = n \]
  \[ [e_1 \ast e_2] = [e_1] \times [e_2] \]
- We’ll take deno-sem as the “ground truth”
- For this language the precise semantics is computable (but in general it’s not)

An Abstraction

- Assume that we are interested not in the value of the expression, but only in its sign:
  - positive (+), negative (-), or zero (0)
- We can define an abstract semantics that computes only the sign of the result
  \[ \sigma: \text{Exp} \rightarrow \{-, 0, +\} \]
  \[ \sigma(n) = \text{sign}(n) \]
  \[ \sigma(e_1 \ast e_2) = \sigma(e_1) \otimes \sigma(e_2) \]

I Saw the Sign

- Why did we want to compute the sign of an expression?
  - One reason: no one will believe you know abstract interp if you haven’t seen the sign thing
- What could we be computing instead?
  - Alex Aiken was here ...

Correctness of Sign Abstraction

- We can show that the abstraction is correct in the sense that it predicts the sign
  \[ [e] > 0 \iff \sigma(e) = + \]
  \[ [e] = 0 \iff \sigma(e) = 0 \]
  \[ [e] < 0 \iff \sigma(e) = - \]
- Our semantics is abstract but precise
- Proof is by structural induction on the expression \( e \)
  - Each case repeats similar reasoning

Another View of Soundness

- Link each concrete value to an abstract one:
  \[ \beta: \mathbb{Z} \rightarrow \{-, 0, +\} \]
- This is called the abstraction function (\( \beta \))
  - This three-element set is the abstract domain
- Also define the concretization function (\( \gamma \)):
  \[ \gamma: \{-, 0, +\} \rightarrow \mathcal{P}(\mathbb{Z}) \]
  \[ \gamma(+) = \{ n \in \mathbb{Z} \mid n > 0 \} \]
  \[ \gamma(0) = \{ 0 \} \]
  \[ \gamma(-) = \{ n \in \mathbb{Z} \mid n < 0 \} \]

Another View of Soundness 2

- Soundness can be stated succinctly
  \[ \forall e \in \text{Exp. } [e] \in \gamma(\sigma(e)) \]
  (the real value of the expression is among the concrete values represented by the abstract value of the expression)
- Let \( C \) be the concrete domain (e.g. \( \mathbb{Z} \)) and \( A \) be the abstract domain (e.g. \( \{-, 0, +\} \))
- Commutative diagram:
  \[ \text{Exp} \xrightarrow{\sigma} A \]
  \[ \gamma \]
  \[ \mathcal{P}(C) \]
  \[ \epsilon \]
Another View of Soundness 3

- Consider the generic abstraction of an operator
  \[ \sigma(e_1 \text{ op } e_2) = \sigma(e_1) \text{ op } \sigma(e_2) \]

- This is sound iff
  \[ \forall a_1 \forall a_2. \gamma(a_1, \text{ op } a_2) \supseteq \{ n_1 \text{ op } n_2 | n_1 \in \gamma(a_1), n_2 \in \gamma(a_2) \} \]

- e.g.
  \[ \gamma(a_1 \otimes a_2) \supseteq \{ n_1 \ast n_2 | n_1 \in \gamma(a_1), n_2 \in \gamma(a_2) \} \]

Abstract Interpretation

- This is our first example of an abstract interpretation
- We carry out computation in an abstract domain
- The abstract semantics is a sound approximation of the standard semantics
- The concretization and abstraction functions establish the connection between the two domains

Adding Unary Minus and Addition

- We extend the language to
  \[ e ::= n \mid e_1 \ast e_2 \mid - e \]
- We define \( \sigma(-e) = \ominus \sigma(e) \)

- Now we add addition:
  \[ e ::= n \mid e_1 \ast e_2 \mid - e \mid e_1 + e_2 \]
- We define \( \sigma(e_1 + e_2) = \sigma(e_1) \oplus \sigma(e_2) \)

Adding Addition

- The sign values are not closed under addition
- What should be the value of “+ ⊕ -”?  
- Start from the soundness condition:
  \[ \gamma(+ \oplus -) \supseteq \{ n_1 + n_2 | n_1 > 0, n_2 < 0 \} = \mathbb{Z} \]
- We don’t have an abstract value whose concretization includes \( \mathbb{Z} \), so we add one:
  \[ \top \]
  (“top” = “don’t know”)

Loss of Precision

- Abstract computation might lose information
  \[ [(1 + 2) + -3] = 0 \]
  but \( \sigma((1+2) + -3) = (\sigma(1) \oplus \sigma(2)) \oplus \sigma(-3) = (\oplus \oplus +) \ominus - = \top \)
- We lose some precision
- But this will simplify the computation of the abstract answer in cases when the precise answer is not computable

Adding Division

- Straightforward except for division by 0
  - We say that there is no answer in that case
    \[ \gamma(+ \oslash 0) = \{ n | n = n_1 / 0 \}, n_1 > 0 \} = \emptyset \]
  - Introduce \( \bot \) to be the abstraction of the 0
  - We also use the same abstraction for non-termination!
    \( \bot = “nothing” \)
    \( \top = “something unknown” \)
The Abstract Domain

- Our abstract domain forms a lattice
- A partial order is induced by $\gamma$
  $a_1 \leq a_2$ iff $\gamma(a_1) \subseteq \gamma(a_2)$
  - We say that $a_1$ is more precise than $a_2$!
- Every finite subset has a least-upper bound (lub) and a greatest-lower bound (glb)

Lattice Facts

- A lattice is complete when every subset has a lub and a gub
  - Even infinite subsets!
- Every finite lattice is (trivially) complete
- Every complete lattice is a complete partial order (recall: denotational semantics!)
  - Since a chain is a subset
- Not every CPO is a complete lattice
  - Might not even be a lattice

Lattice History

- Early work in denotational semantics used lattices
  - But it was later seen that only chains need to have lubs
  - And there was no need for $\top$ and $\bot$
- In abstract interpretation we’ll use $\top$ to denote “I don’t know”
  - Corresponds to all values in the concrete domain

From One, Many

- We can start with the abstraction function $\beta$
  $\beta : C \rightarrow A$
  (maps a concrete value to the best abstract value)
  - $A$ must be a lattice
- We can derive the concretization function $\gamma$
  $\gamma : A \rightarrow P(C)$
  $\gamma(a) = \{ x \in C \mid \beta(x) \subseteq a \}$
- And the abstraction for sets $\alpha$
  $\alpha : P(C) \rightarrow A$
  $\alpha(S) = \text{lub} \{ \beta(x) \mid x \in S \}$

Example

- Consider our sign lattice
  $\beta(n) = \begin{cases} + & \text{if } n > 0 \\ 0 & \text{if } n = 0 \\ - & \text{if } n < 0 \end{cases}$
- $\alpha(S) = \text{lub} \{ \beta(x) \mid x \in S \}$
  - Example: $\alpha(\{1, 2\}) = \text{lub} \{ + \} = +$
  $\alpha(\{1, 0\}) = \text{lub} \{ +, 0 \} = \top$
  $\alpha(\{\}) = \text{lub} \{ \} = \bot$
- $\gamma(a) = \{ n \mid \beta(n) \subseteq a \}$
  - Example: $\gamma(+) = \{ n \mid \beta(n) \subseteq + \} = \{ n \mid n > 0 \}$
  $\gamma(\top) = \{ n \mid \beta(n) \subseteq \top \} = \mathbb{Z}$
  $\gamma(\bot) = \{ n \mid \beta(n) \subseteq \bot \} = \emptyset$

Galois Connections

- We can show that
  - $\gamma$ and $\alpha$ are monotonic (with $\subseteq$ ordering on $P(C)$)
  - $\alpha(\gamma(a)) = a$ for all $a \in A$
  - $\gamma(\alpha(S)) \supseteq S$ for all $S \in P(C)$
- Such a pair of functions is called a Galois connection
  - Between the lattices $A$ and $P(C)$
Correctness Condition

- In general, abstract interpretation satisfies the following (amazingly common) diagram

![Diagram](image)

Correctness Conditions

- Three conditions define a correct abstract interpretation
  1. \( \alpha \) and \( \gamma \) are monotonic
  2. \( \alpha \) and \( \gamma \) form a Galois connection
     = “\( \alpha \) and \( \gamma \) are almost inverses”
  3. Abstraction of operations is correct
     \( a_1 \text{ op } a_2 = \alpha(\gamma(a_1) \text{ op } \gamma(a_2)) \)

Homework

- Homework 4 Due Today
- Homework 5 Out Today
- Read Ken Thompson Turing Award
- Project Proposal Due On Tuesday