Announcements

- HW3 due? Extension?
- HW4 out today or tomorrow
- 3 larger benchmarks
- Run RTA and XTA including library classes
- Compare RTA vs. XTA call graphs
- Write a 1-2 page summary of your results
- Make HW4 a team assignment?

Abstract Interpretation

- Patrick Cousot and Radhia Cousot, POPL'77
  - A general framework
  - Building static analyses
  - Reasoning about correctness of static analysis
  - Comparing static analyses
  - Combines ideas from dataflow analysis (monotone frameworks and fixpoint iteration) and formal verification (semantics)

Lecture Notes Based On

- "Principles of Program Analysis" by Nielsen, Nielsen and Hankin, Chapter 3
  - Alex Salcianu’s friendlier account of Chapter 3: https://web.eecs.umich.edu/~bchandra/courses/papers/Salcianu_AbstractInterpretation.pdf
- Lecture notes by Xavier Rival, ENS

Overview

Program Execution:

Points-to Analysis (PTA):

x = y.n(z)  x = y.n(z)
pts(z) ⊆ pts(p)

Analysis 1 Space:  Analysis 2 Space:

Intuitively, Abstract Interpretation is about relating one Analysis to another Analysis
Overview

**PTA:**

\[ x \rightarrow \alpha : A \]

**XTA:**

\[ S_m \rightarrow A \]

\[ x = y.n(z) \]

\[ \text{pts}(z) \subseteq \text{pts}(p) \]

\[ x = y.n(z) \text{ Sm} \text{ SubTypes}(\text{StaticType}(p)) \subseteq S_m \]

---

**Overview**

**RTA:**

\[ I \rightarrow A \]

\[ x = y.n(z) \]

\[ \text{SubTypes(p)} \subseteq S_m \]

\[ \text{Sm} \cap S_m \]

---

Outline

- Abstract interpretation, last class
- Semantics, last class
- Notion of abstraction
- Concretization and abstraction functions
- Galois Connections
- Applications of abstract interpretation

---

Collecting Semantics

- Collects all states a program can have at a given label (i.e., program point)
  - E.g., \( x \) cannot ever be null at \( j \)
  - \( n \) is always greater than 100 at \( j \)
- Given a label, we are interested in a function
  - \( C : \text{Labels} \rightarrow 2^A \)
  - The set of all states a program can have at \( t \)

---

We’ve Seen This Before...

- Operational semantics
  - Defines “concrete transfer functions”
  - Works on values drawn from \( \mathbb{Z} \) (concrete state)
- A static analysis, e.g., Constant Propagation
  - Defines “transfer functions”
  - Work on values from the flat lattice (abstract state)

---

Collecting Semantics

\[ C[L2] = \{ \sigma[x \rightarrow n] | \sigma \in C[L1] \} \]

\[ C[L1] = \{ \sigma | \sigma \in C[L1], \llbracket e \rrbracket \sigma = \text{true} \} \]

\[ C[L3] = C[L1] \cup C[L2] \]

---

RTA is an abstraction of XTA.

Abstracts \( S_m \) into \( I \), the set of instantiated types across _all_ of the program.
Collecting Semantics

- "Ground truth"
  - We base reasoning about correctness (soundness) of our static analysis off of it
- Undecidable
- Relation to MOP solution?
- Define abstraction of state and semantics
- Goal: show that abstraction "properly represents" all values computed by collecting semantics

Abstraction Example 1: signs

Concrete values: sets of integers
Abstract values: signs
Lattice of signs:

- \( \bot \) represents the empty set
- \( + \) represents any set of positive integers
- \( 0 \) represents set \( \{ 0 \} \)
- \( - \) represents any set of negative integers
- \( T \) represents any set of integers

Concrete space: A lattice!
Abstract space: A lattice!

Analysis computes over abstract elements

Correctness conclusion, informally: if analysis (works on abstract elements \( a \)) determines that \( x \) at label \( \ell \) is \( a \), then \( a \) represents the set of concrete values \( c \) collected by the collecting semantics for \( x \) at \( \ell \)
Abstraction Example 1: signs

- We can also use U and ∩
  - if \( x_1 \) is \( \pm \) and \( x_2 \) is \( \pm \) then \( x_1 \cup x_2 \) is \( \pm \)

How about if \( x_1 \) is \( \pm \) and \( x_2 \) is \( 0 \)?
- then \( x_1 \cup x_2 \) is \( T \)
  - because only \( \{0,1,2,3,\ldots\} \) \( \cap \) \( T \) holds
  - No other relation holds
- In the abstract, we include negative integers in \( x_1 \cup x_2 \) (we lose precision!)

Abstraction Example 2: constants

- Concrete elements: elements of concrete lattice, \( c \in 2^Z \)
- Abstract elements: \( \bot, T, n \), where \( n \in Z \)
  - Flat lattice:

- Abstraction relation:
  - \( c \vdash n \) if and only if \( c = \{ n \} \)
  - \( \bot \) represents the empty set
  - \( T \) represents any set of integers

Abstract semantics, works on abstract elements (the elements of the flat lattice)

- If \( x_1 \) is \( n_1 \) and \( x_2 \) is \( n_2 \) then \( x_1 + x_2 \) is \( n_1 + n_2 \)
  - \( n_1 \) represents exactly integer \( n_1 \)
  - \( n_2 \) represents \( n_2 \)
  - then \( x_1 + x_2 \) is \( n_1 + n_2 \), represented by \( n_1 + n_2 \)
- If \( x_1 \) is \( n_1 \) and \( x_2 \) is \( T \), then \( x_1 + x_2 \) is \( T \)
  - Since \( T \) represents any set, we cannot do better in the abstract but \( T \)

Abstraction Example 3: intervals

- Concrete elements: elements of \( 2^Z \)
- Abstract elements: \( \bot, T, n, \) intervals \([a,b]\) where \( a \in Z \cup \{ -\infty \} \) and \( b \in Z \cup \{ \infty \} \) and \( a \leq b \)
  - Is it a lattice?
  - Yes!

- Abstraction relation:
  - \( \emptyset \vdash \bot \)
  - \( S \vdash T \)
  - \( S \vdash [a,b] \) iff for every \( n \in S, a \leq n \leq b \)

Abstract semantics

- If \( x_1 \) is \([a_1,b_1]\) and \( x_2 \) is \([a_2,b_2]\) then \( x_1 + x_2 \) is?
- If \( x_1 \) is \([a_1,b_1]\) and \( x_2 \) is \([a_2,b_2]\) then \( x_1 \cup x_2 \) is?
- If \( x_1 \) is \([a_1,b_1]\) and \( x_2 \) is \([a_2,b_2]\) then \( x_1 \cap x_2 \) is?
Abstraction Example 4: heap

- We need to expand our syntax and operational semantics to model the heap
  - Realistic analyses of imperative languages do need to model heap

- “Collection of objects” view (Java):
  
  \[ s \in S : Id \rightarrow Addr \] // Stack
  
  \[ h \in H : Addr \times Id \rightarrow Addr \] // Heap

Abstraction Example 4: heap

- Concrete elements:
  - Sets of heap objects \( h_j \)

- Abstract elements:
  - Sets of abstract heap objects \( o_i \)
  - Abstraction relation (gist of it):
    - \( o_i \) represents any set of objects \( h_j \) allocated at static allocation site \( i \)
  - Abstract semantics:
    - If \( x \) points to \( o_i \), after \( y = x \), \( y \) points to \( o_i \)

Abstraction Example 4: heap

- | \( x = y.f \) | \( s \cdot h = s \cdot h[ (|y|)(s), f ]; h \)
  - | \( x.f = y \) | \( s \cdot h = s \cdot h[ (|x|)(s), f ]; (|y|)(s) \)

(Note: \(|x|\)(s) returns the address of \( x \) in stack \( s \).)

- | \( i: x = \text{cons}(x_1, \ldots, x_k) \) | \( s \cdot h = s[x \leftarrow j \cdot h(j,f_1); (|x_1|)(s), \ldots, h(j,f_k); (|x_k|)(s)] \)

(i is the unique static allocation site identifier; \( j \) is the next available address in \( h \))

(read: allocate new object at address \( j \) on heap \( h \), then initialize values of fields \( f_1 \ldots f_k \) of \( j \) with respective addresses resulting from evaluation of \( x_1 \ldots x_k \) in \( s \))

Outline

- Abstract interpretation
- Semantics
  - Notion of abstraction
  - Concretization and abstraction functions
  - Galois Connections
- Applications of abstract interpretation