Abstract Interpretation: Gallois, Collecting, Widening

Meeting 15, CSCI 5535, Spring 2009

Survey from Homework 5

• Fun facts about your classmates:
  - From Middleton, Idaho
  - Sprinter in high school
  - Enjoys flying (pilot?)

Review of Abstract Interpretation

• We introduced abstract interpretation
• An abstraction mapping from concrete to abstract values
  - Has a concretization mapping which forms a Galois connection

Why Galois Connections?

• We have an abstract domain \( A \) (e.g., \( \{ -, 0, + \} \))
  - An abstraction function \( \beta: C \rightarrow A \)
  - Induces \( \alpha: \mathcal{P}(C) \rightarrow A \) and \( \gamma: A \rightarrow \mathcal{P}(C) \)
• We argued that for correctness
  \[ \gamma(\alpha(x_1 \text{ op } x_2)) \subseteq \gamma(\alpha(x_1)) \oplus \gamma(\alpha(x_2)) \]
  - Wish the set on the left to be as small as possible
  - To reduce the loss of info through abstraction
• For each set \( S \subseteq C \), define \( \alpha(S) \) as follows:
  - Pick smallest \( S' \) that includes \( S \) and \( x \) in the image of \( \gamma \)
  - Define \( \alpha(S) = \gamma(S') \)
  - Then we define: \( \alpha(x_1 \text{ op } x_2) = \alpha(x_1) \oplus \gamma(\alpha(x_2)) \)
• Then \( \alpha \) and \( \gamma \) form a Galois connection
Galois Connections

• A pair of functions between lattices $A$ and $P(C)$
  - $\gamma$ 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 $S \in P(C)$

More on Galois Connections

• All Galois connections are monotonic
• In a Galois connection one function uniquely and absolutely determines the other

Abstract Interpretation for Imperative Programs

• So far we abstracted the value of expressions
• Now we want to abstract the state at each point in the program
• First we define the concrete semantics that we are abstracting
  - We'll use a collecting semantics

Collecting Semantics

• Recall
  - A state $\sigma \in \Sigma$,
    Any state $\sigma$ has type $\text{Var} \rightarrow \mathbb{Z}$
  - States vary from program point to program point
• We introduce a set of program points: labels
• We want to answer questions like:
  - Is $x$ always positive at label $i$?
  - Is $x$ always greater or equal to $y$ at label $j$?

Collecting Semantics

• To answer these questions we'll construct
  $$C \in \text{Contexts}, \ C \text{ has type } \text{Labels} \rightarrow P(\Sigma)$$
  - For each label $i$,
    $$C(i) = \text{all possible states at label } i$$
  - This is called the collecting semantics of the program
  - This is basically what SLAM (and BLAST, ESP, ...) approximate (using BDDs to store $P(\Sigma)$ efficiently)
Defining the Collecting Semantics

- We first define relations between the collecting semantics at different labels
  - We do it for unstructured CFGs (flowchart programs)
  - Can do it for IMP with careful notion of program points
- Define a label on each edge in the CFG
  - For assignment
    \[
    x := e_i\quad C_j = \{\sigma[x := n] | \sigma \in C_i \land [e]\sigma = n\}
    \]
- For conditionals
  \[
  C_{\text{else}} = \{\sigma | \sigma \in C_{\text{in}} \land [b]\sigma = \text{false}\}
  \]
  \[
  C_{\text{then}} = \{\sigma | \sigma \in C_{\text{in}} \land [b]\sigma = \text{true}\}
  \]
- Assumes \(b\) has no side effects (as in IMP)
- For a join
  \[C_{\text{out}} = C_i \cup C_j\]
- Verify that these relations are monotonic
  - If we increase a \(C_i\), all other \(C_j\) can only increase
Collecting Semantics: Example

Consider the following program (assume $x \geq 0$ initially)

1. $y := 1$
2. $x == 0$
3. $x := x - 1$
4. $x := x * (x - 1)$
5. $C_4 = \{ y := \alpha(x)^* \alpha(x) \mid \sigma \in C_3 \}$

Why Does This Work?

- We just made a system of recursive equations that are defined largely in terms of themselves
- e.g., $C_2 = F(C_4)$, $C_4 = G(C_3)$, $C_3 = H(C_2)$
- Why do we have any reason to believe that this will get us what we want?

The Collecting Semantics

- We have an equation with the unknown $C$
  - The equation is defined by a monotonic and continuous function on the domain $\text{Labels} \rightarrow \mathcal{P}(\Sigma)$
  - We can use the least fixed-point theorem
    - Start with $C^0(L) = \emptyset$ (aka $C^0 = \lambda L. \emptyset$)
    - Apply the relations between $C_i$ and $C_j$ to get $C_i$ from $C_j$
    - Stop when all $C^k = C^{k-1}$
- Problem: we'll go on forever for most programs
- But we know the fixed point exists

Collecting Semantics: Example

Consider the following program (assume $x \geq 0$ initially)

- $C_1 = \{ \sigma \mid \sigma(x) \geq 0 \}$
- $C_2 = \{ \alpha[y:=1] \mid \sigma \in C_1 \}$
- $C_3 = C_2 \cap \{ \sigma \mid \sigma(x) = 0 \}$
- $C_4 = \{ \sigma[y:=\alpha(y)^* \alpha(x)] \mid \sigma \in C_3 \}$
Collecting Semantics: Example

- Consider the following program (assume \( x \geq 0 \) initially)

\[
\begin{array}{c}
\text{Line} \\
0 & 1 & 2 & 3 & 4 & 5 \\
\hline
x := x - 1 & y := y \times x & y := y \times x & x := x - 1 & \{x = 0, y = x + 1\} & \{x > 0, y = x\}
\end{array}
\]

\( C_1 = \{\sigma | \sigma(x) \geq 0\} \)
\( C_2 = \{\sigma | \sigma(y) = 1\} \)
\( C_3 = C_2 \cap \{\sigma | \sigma(x) = 0\} \)
\( C_4 = C_3 \cap \{\sigma | \sigma(x) = 0\} \)
\( C_5 = \{\sigma | \sigma(y) \times \sigma(x) | \sigma \in C_1\} \)

Back to Abstract Interpretation
Abstract Interpretation on CFG

- Pick a complete lattice $A$ (abstracts for $P(\Sigma)$)
- Along with a monotonic abstraction $\alpha : P(\Sigma) \to A$
- Alternatively, pick $\beta : \Sigma \to A$
- This uniquely defines its Galois connection $\gamma$
- Take the relations between $C_i$ and move them to the abstract domain:
  - Assignment
    - Concrete: $C_j = \{ \sigma | \sigma \in C_i \land [b] \sigma = n \}$
    - Abstract: $a_j = \alpha(\gamma(a_i) \land [b] \sigma = n)$

Least Fixed Points
In The Abstract Domain

- We have a recursive equation with unknown "a"
  - Defined by a monotonic and continuous function on the domain $Labels \to A$
- We can use the least fixed-point theorem:
  - Start with $a^0 = \lambda L. \perp$ (aka: $a^0(L) = \perp$)
  - Apply the monotonic func to compute $a^{n+1}$ from $a^n$
  - Stop when $a^{n+1} = a^n$
- Exactly the same computation as for the collecting semantics
  - What is new?

Least Fixed Points
In The Abstract Domain

- We have a hope of termination!
- Classic setup: $A$ has only uninteresting chains (finite number of elements in each chain)
  - $A$ has finite height $h$ (= "finite-height lattice")
- The computation takes $O(h \times |Labels|^2)$ steps
  - At each step "a" makes progress on at least one label
  - We can only make progress $h$ times
  - And each time we must compute $|Labels|$ elements
- This is a quadratic analysis: good news

That's It!
Program Analysis in a Nutshell

Define an Abstraction

Compute a Fixed Point in the Abstraction

What's new?

- Compute it!
Abstract Interpretation: Example
• Consider the following program

Let's Do It!

Weaknesses
• We abstracted the state of each variable independently
  \[ A = \{x, y\} \rightarrow \{\bot, -, 0, +, \top\} \]
  - Our abstraction is non-relational
• We lost relationships between variables
  - e.g., at a point \(x = 0\) and \(y\) may always have the same sign
  - In the previous abstraction, we get \(x := \top, y := \top\) at label 2 (when in fact \(y\) is always positive!)

Potential Solutions?
• Can also abstract the state as a whole
  \[ A = \mathcal{P}(\{\bot, -, 0, +, \top\} \times \{\bot, -, 0, +, \top\}) \]
  - For the previous example we now get the abstraction \(\{(-, -), (0, 0), (+, +)\}\) at 2
Other Abstract Domains

- Range analysis
  - Lattice of ranges: \( R = \{ \bot, \{n..m\}, (\rightarrow \infty, m], \{n, +\infty) \} \)
  - It is a complete lattice
    - \( \{n..m\} \sqcup \{n'..m'\} = \{\min(n, n')..\max(m, m')\} \)
    - \( \{n..m\} \sqcap \{n'..m'\} = \{\max(n, n')..\min(m, m')\} \)
  - With appropriate care in dealing with \( \infty \)
    - \( \beta : \mathbb{Z} \rightarrow \mathbb{R} \) such that \( \beta(n) = \{n..n\} \)
    - \( \alpha : P(\mathbb{Z}) \rightarrow \mathbb{R} \) such that \( \alpha(S) = \text{lub} \{ \beta(n) | n \in S \} = \{\min(S)..\max(S)\} \)
    - \( \gamma : \mathbb{R} \rightarrow P(\mathbb{Z}) \) such that \( \gamma(r) = \{ n | n \in r \} \)
  - This lattice has infinite-height chains
  - So the abstract interpretation might not terminate!

Example of Non-Termination

- Consider this (common) program fragment
  - \( i := 0 \)
  - \( i \leq n \)
  - \( i := i + 1 \)

We want to do range analysis for it

Example of Non-Termination

- Consider the sequence of abstract states at point 2
  - \( \{1..1\}, \{1..2\}, \{1..3\}, \ldots \)
  - The analysis never terminates
  - Or terminates very late if the loop bound known statically
- It is time to approximate even more: widening
- We redefine the join (lub) operator of the lattice to ensure that from \( \{1..1\} \) upon union with \( \{2..2\} \) the result is \( \{1, +\infty\} \) and not \( \{1..2\} \)
- Now the sequence of states is
  - \( \{1..1\}, \{1, +\infty\}, \{1, +\infty\} \) Done (no more infinite chains)

Formal Definition of Widening

- A widening \( \triangledown : (P \times P) \rightarrow P \) on a poset \( (P, \sqsubseteq) \) satisfies:
  - For all increasing chains \( x_0 \sqsubseteq x_1 \sqsubseteq \ldots \)
    the increasing chain
    \( y_0 = \text{def} x_0, y_{n+1} = \text{def} y_n \triangledown x_{n+1}, \ldots \)
    is not strictly increasing.

Formal Widening Example

\[ [1,1] \triangledown [1,2] = [1, +\infty] \]

Uses of Widening

- Two different main uses:
  - Approximate missing lubs. (Not for us.)
  - Convergence acceleration. (This is the real use.)
- Used to compute an upper approximation of the least fixpoint of \( F \in L \triangledown L \) starting from below when \( L \) does not satisfy the ascending chain condition.

The magic of program analysis

Original \( x \)

L0: \( z := 1 \)
  - \( x_{0z}^0 = \bot \)
  - \( y_{0z}^0 = \bot \)
L1: \( z := 1 \)
  - \( x_{1z}^1 = [1..1] \)
  - \( y_{1z}^1 = [1..1] \)
L2: \( z := z + 1 \)
  - \( x_{2z}^2 = [1..1] \)
  - \( y_{2z}^2 = [1..1] \)
L3: done /* \( z \geq 99 \) */
  - \( x_{3z}^3 = [2..2] \)
  - \( y_{3z}^3 = [2..2] \)
L4:
  - \( x_{4z}^4 = [1\ldots1] \)
  - \( y_{4z}^4 = [1\ldots1] \)

\( x_i^j = \text{def} \) the \( j \)-th iterative attempt to compute an abstract value for \( z \) at label \( L_i \)

Recall lub \( S = [\min(S) \ldots \max(S)] \)
and \( ([1..\infty]([1..\infty])) = ([1..\infty]) \).
Other Abstract Domains

- Linear relationships between variables
  - A convex polyhedron is a subset of \( \mathbb{R}^k \) whose elements satisfy a number of inequalities:
    \[ a_1 x_1 + a_2 x_2 + \ldots + a_k x_k \geq c \]
  - This is a complete lattice; linear programming methods compute lubs

- Linear relationships with at most two variables
  - Convex polyhedra but with \( \leq 2 \) variables per constraint
  - Octagons \((x + y \geq c)\) have efficient algorithms

- Modulus constraints (e.g. even and odd)

Abstract Chatter

- AI, Dataflow, and Software Model Checking
  - The big three (aside from flow-insensitive type systems) for program analyses
  - Are in fact quite related:
    - David Schmidt. Data flow analysis is model checking of abstract interpretation. POPL '98.
  - AI is usually flow-sensitive (per-label answer)
  - AI can be path-sensitive (if your abstract domain includes \( \wedge \), for example), which is just where model checking uses BDD's
  - Metal, SLAM, ESP, ... can all be viewed as AI

Abstract Interpretation Summary

- AI is a very powerful technique that underlies a large number of program analyses
- AI can also be applied to functional and logic programming languages
- When the lattices have infinite height and widening heuristics are used, the result become harder to predictable
- AI is behind Astrée, which is used by Airbus