Abstract Interpretation and the Heap

Computer Science and Artificial Intelligence Laboratory
MIT

Nov 16, 2015
Recap

An abstract domain is a lattice

*Some analysis relax this restriction.

- Elements in the lattice are called Abstract Values

Need to relate elements in the lattice with states in the program

- **Abstraction Function**: $\alpha: \mathcal{P}(\mathcal{V}) \rightarrow \text{Abs}$
  
  - Maps a value in the program to the “best” abstract value

- **Concretization Function**: $\gamma: \text{Abs} \rightarrow \mathcal{P}(\mathcal{V})$
  
  - Maps an abstract value to a set of values in the program
Modeling the Heap

Giant Array vs. Collection of Objects (C vs Java view)

Giant array view

- $s \in S : Id \rightarrow \text{Int}$
- $h \in H : Nat \rightarrow \text{Int}$
- $\llbracket C \rrbracket : S \times H \rightarrow S \times H \cup \{\bot\}$
- $\llbracket E \rrbracket : S \rightarrow \text{Int}$
- $\llbracket x = [e] \rrbracket s h = s \{x \rightarrow h(\llbracket e \rrbracket s)\} h$
- $\llbracket [e] = x \rrbracket s h = s h(\llbracket e \rrbracket s \rightarrow s(x))$
- $\llbracket x = \text{cons}(e_0 \ldots e_k) \rrbracket s h = s \{x \rightarrow j\} h\{j \rightarrow \llbracket e_0 \rrbracket s, \ldots, j + k \rightarrow \llbracket e_k \rrbracket s\}$
  where $j = (\max \text{dom } h) + 1$
Modeling the Heap

Giant Array vs. Collection of Objects (C vs Java view)

Collection of Objects View

- $s \in S : Id \rightarrow Addr$
- $h \in H : Addr \times Id \rightarrow Addr$
- $[C] : S \times H \rightarrow S \times H \cup \{\perp\}$
- $[E] : S \rightarrow Addr$
- $[x = e. f] s h = s \{x \rightarrow h([e] s, f)\} h$
- $[e. f = x] s h = s h\{([e] s, f) \rightarrow s(x)\}$
- $[x = \text{cons}(e_0 ... e_k)] s h = s \{x \rightarrow j\} h\{(j, f_0) \rightarrow [e_0] s, ..., (j, f_k) \rightarrow [e_k] s\}$ where $j = \text{fresh address}$

This is the view we will focus on today

The pset provides a third alternative

- Each object is indexed by integer offsets rather than fields
- Not significantly different from this alternative
The state as a graph
The state as a graph

\[
\begin{align*}
\text{h}(L1, \text{head}) &= N1 \\
\text{h}(L1, \text{tail}) &= N4 \\
\text{h}(N1, \text{next}) &= N2 \\
\text{h}(N2, \text{next}) &= N3 \\
\text{h}(N3, \text{next}) &= N4 \\
\text{h}(N4, \text{next}) &= \text{null}
\end{align*}
\]

\[
S(X) = L1 \\
S(Y) = N3
\]
Try 1: A simple abstraction

Have a single node for all objects of the same type

Concrete

Abstract

Nodes are abstract heap locations representing multiple concrete heap nodes. Known as *summary nodes*
Formal definition

Let $\tau(addr)$ be the summary node representing an address (we have one for each type)

- We can define a special node $null = \tau(null)$

Abstraction function

- $\alpha(h,S) := (\bar{h},\bar{S})$
- $\bar{h}(t,f) := \{ t'| \exists a \in Addr, \tau(a) = t \land h(a,f) = a' \land \tau(a') = t'\}$
- $\bar{S}(x) := \{\tau(S(x))\}$

Partial order

- $(\bar{h}_1,\bar{S}_1) \subseteq (\bar{h}_2,\bar{S}_2)$ iff $\forall t,f \ \bar{h}_1(t,f) \subseteq \bar{h}_2(t,f) \land \forall x \ \bar{S}_1(x) \subseteq \bar{S}_2(x)$

Concretization

- $(h,S) \in \gamma(\bar{h},\bar{S})$ iff
  $\left( h(a,f) = b \Rightarrow \tau(b) \in \bar{h}(\tau(a),f) \right) \land (S(x) = a \Rightarrow \tau(a) \in \bar{S}(x))$
Update

\[ [e.f = x](\bar{h}, \bar{S}) = (\bar{h}', \bar{S}) \]

Where \( \bar{h}'(t, f) = \begin{cases} 
\bar{h}(t, f) & \text{if } t \notin [e](\bar{h}, \bar{S}) \\
\bar{h}(t, f) \cup \bar{S}(x) & \text{if } t \in [e](\bar{h}, \bar{S}) 
\end{cases} \]
The problem of destructive updates

\[ x = new T(\_); \]
\[ x.f = null; \]
\[ x.f = new P(\_); \]
The problem of destructive updates

\[
\begin{align*}
x &= \text{new } T(\ ); \\
x.f &= \text{null}; \\
x.f &= \text{new } P(\ ); \\
\end{align*}
\]

The abstraction cannot tell that \(x.f\) is no longer null

Why not?
The problem of destructive updates

\[
x = \text{new } T();
\]
\[
x.f = \text{null};
\]
\[
x.f = \text{new } P();
\]

Why is this the best we can do?

Abstraction cannot distinguish these two concrete cases

\[
x.f = \text{new } P();
\]
The problem

All abstract heap nodes represented multiple concrete heap nodes
- This makes it impossible to do destructive updates

The abstract domain in the pset is more refined but it suffers from the same problem
Try 2: Abstract based on Variables

“Solving Shape-Analysis Problems in Languages with Destructive Updating” Sagiv, Reps & Wilhelm
- We’ll simplify a little relative to this paper

Idea
- Objects pointed to by variables should be concretized
Example

\[
x = \text{new } T();
\]
\[
x.f = \text{null};
\]
\[
x.f = \text{new } T();
\]

X always points to a concrete location
This allows a destructive update to x.f
Example

Note that t1 is “the location pointed to by x” and not a specific concrete node.

```java
x = new T();
x.f = null;
x.f = new T();
x = x.f
x.f = null
```
Formalization

Let $PVar$ be the set of variables. Then the locations in the abstract state will be $\{n_Z | Z \subseteq PVar\}$

Not all $n_Z$ will be present in a given abstract state
- In particular, different $n_Z$ cannot share variables.

Abstraction
- $\alpha_s(a) = n_Z$ where $Z = \{x | S(x) = a\}$
- $\alpha(h, S) := (\overline{h}, \overline{S})$
- $\overline{h}(n_Z, f) := \{n_Z' | \exists a \in Addr, \alpha_s(a) = n_Z \land h(a, f) = a' \land \alpha_s(a') = n_Z'\}$
- $\overline{S}(x) := \{\alpha_s(S(x))\}$

Partial order
- $(\overline{h}_1, \overline{S}_1) \subseteq (\overline{h}_2, \overline{S}_2)$ iff $\forall t, f \ \overline{h}_1(t, f) \subseteq \overline{h}_2(t, f) \land \forall x \ \overline{S}_1(x) \subseteq \overline{S}_2(x)$
Update

\[ [e. f = x](\bar{h}, \bar{S}) = (\bar{h}', \bar{S}) \]

Where \( \bar{h}'(n_z, f) = \begin{cases} 
\bar{h}(n_z, f) & \text{if } n_z \notin [e](\bar{h}, \bar{S}) \\
\bar{S}(x) & \text{if } z \neq \emptyset \land n_z \in [e](\bar{h}, \bar{S}) \\
\bar{h}(n_z, f) \cup \bar{S}(x) & \text{if } z = \emptyset \land n_z \in [e](\bar{h}, \bar{S}) 
\end{cases} \]

\[ [x = e](\bar{h}, \bar{S}) = (\bar{h}', \bar{S}') \] (Note var update also affects heap)

- Let \( [e](\bar{h}, \bar{S}) = \{n_{z0}, ..., n_{zk}\} \)
- \( \bar{S}'(x) = \{n_{z0 \cup \{x\}}, ..., n_{z0 \cup \{x\}}\} \)
- For \( y \neq x \), \( \bar{S}'(y) = replace(n_{zi}, n_{zi \cup \{x\}}, \bar{S}(x)) \)
- How do we update \( \bar{h} \)?
Nodes $n_x$ and $n_y$ disappear (become unreachable)
New node $n_{x,y}$ now pointed by both $x$ and $y$
The old $n_x$ is now represented by $n_\emptyset$ which acquires a self loop
Updating the heap

Let $E_s(n_W, f, n_Y) \iff n_Y \in \overline{h}(n_W, f)$ (resp. for $E'_s$)
Then after $x = e$ with $\llbracket e \rrbracket(\overline{h}, S) = \{n_{z_0}, \ldots, n_{z_k}\}$

- $E_s(n_W, f, n_{z_i}) \Rightarrow E'_s(n_W, f, n_{z_i} \cup \{x\})$
  - And if $W \neq \emptyset$ $E'_s(n_W, f, n_{z_i})$ should now be false. Why?
- $E_s(n_{z_i}, f, n_W) \Rightarrow E'_s(n_{z_i} \cup \{x\}, f, n_W)$
  - And if $Z_i \neq \emptyset$ $E'_s(n_{z_i}, f, n_W)$ should now be false. Why?
- The old $n_{z_i}$ turned into $n_{z_i} \cup \{x\}$ so things that used to point to $n_{z_i}$ now point to $n_{z_i} \cup \{x\}$.
- Do we need to do something special when $x \in Z_i$?