Abstract Interpretation and the Heap

Computer Science and Artificial Intelligence Laboratory
MIT

With slides and examples by Mooly Sagiv.
Used with permission.

Nov 18, 2015
Recap: Collecting Semantics

Compute for each program the true set of states that can occur at each point
Example: Collecting Interpretation

\(x = \text{NULL}\)

\(F\) \(T\)

\(t = \text{malloc}(...)\);

\(t \rightarrow \text{next}=x;\)

\(x = t\)

return \(x\)
Example: Abstract Interpretation

```
x = NULL

F  T

t = malloc(..);

F  T

t → next = x;

F  T

x = t

return x
```

Empty

```
empty
```

```
t → x
```

```
t → n
```

```
x
```

```
t → n
```

```
x
```

```
t → n
```

```
x
```

```
x
```
Concrete Interpretation

A slightly different view of the state

- Env: $Var \rightarrow Values$
- One map per field
- Field: $Loc \rightarrow Values$
- $Values = Loc \cup Atoms$

Example

- Env = $[x \rightarrow 30, p \rightarrow 79]$
- Fields:
  - next = $[30 \rightarrow 40, 40 \rightarrow 50, 50 \rightarrow 79, 79 \rightarrow 90]$
  - val = $[30 \rightarrow 1, 40 \rightarrow 2, 50 \rightarrow 3, 79 \rightarrow 4, 90 \rightarrow 5]$
The TVLA Approach

Represent the store with logical predicates
- Then do abstraction on these predicates
- An approach to building abstractions instead of a single one

Locations $\approx$ Individuals
Program variables $\approx$ Unary predicates
Fields $\approx$ Binary predicates

Example
- $U = \{u1, u2, u3, u4, u5\}$
- $x = \{u1\}, p = \{u3\}$
- $n = \{\langle u1, u2 \rangle, \langle u2, u3 \rangle, \langle u3, u4 \rangle, \langle u4, u5 \rangle\}$
Important notation

Transitive closure of a binary predicate $n(u, v)$

- $n^*(u, v) := u = v \lor (\exists w. n(u, w) \land n^*(w, v))$
- $n^+(u, v) := (\exists w. n(u, w) \land n^*(w, v))$
Concrete Interpretation

State:
- \( x \): predicate for variable \( x \).
- \( n \): predicate for next field

Rules \([s](x, n) = (x', n')\)
- \([x = null]\) \( n' = n \quad \forall v. \ x'(v) = 0 \)
- \([x = malloc()]\) \( n' = n \quad \forall v. \ x'(v) = IsNew(v) \)
- \([x = y]\) \( n' = n \quad \forall v. \ x'(v) = y(v) \)
- \([x = y.\ next]\) \( n' = n \quad \forall v. \ x'(v) = \exists w. y(w) \land n(w, v) \)
- \([x.\ next = y]\) \( x' = x \quad \forall v \ w. \ n'(v, w) = (\neg x(v) \land n(v, w)) \lor (x(v) \land y(w)) \)
Stating program properties

x points to an acyclic list
- \( \forall v \ w. \ x(v) \land n^*(v, w) \rightarrow \neg n^+(w, v) \)

The heap \( n' \) reverses the list pointed at by \( x \) in \( n \)
- \( \forall v \ w \ r. \ x(v) \land n^*(v, w) \rightarrow (n(w, r) \leftrightarrow n'(r, w)) \)
Canonical Abstraction

Convert logical structures of unbounded size into bounded size

Guarantees that number of logical structures in every program is finite

Every first-order formula can be conservatively interpreted

Same idea we explored last time, but revisited in Three Valued Logic
Kleene Three-Valued Logic

1: True
0: False
1/2: Unknown

A join semi-lattice: $0 \sqcup 1 = 1/2$
### Boolean Connectives [Kleene]

<table>
<thead>
<tr>
<th>$\wedge$</th>
<th>0</th>
<th>1/2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1/2</td>
<td>0</td>
<td>1/2</td>
<td>1/2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1/2</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\lor$</th>
<th>0</th>
<th>1/2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1/2</td>
<td>1</td>
</tr>
<tr>
<td>1/2</td>
<td>1/2</td>
<td>1/2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Key idea

Predicates describing program state are now predicates in 3-Valued Logic

- Let \( U \) be the set of individuals in the concrete domain (potentially infinite)
- Let \( U' \) be the set of individuals in the abstract domain (finite)
- Let \( f : U \to U' \)
- Then a predicate \( p^B \) over \( U \) can be abstracted to \( p^S \) over \( U' \) as follows
  \[ p^S(u'_1, \ldots, u'_k) = \bigcup \{ p^B(u_1, \ldots, u_k) \mid f(u_1) = u'_1, \ldots, f(u_k) = u'_k \} \]
- Since \( U' \) is bounded, \( p^S \) can be represented with a table
Canonical Abstraction

```c
x = NULL;
while (...) do {
    t = malloc();
    t → next = x;
    x = t
}
```

\[
\begin{align*}
n(u1,u2) &= 1 \\
n(u1,u3) &= 0 \\
n(u2,u3) &= 1 \\
n(u3,u3) &= 0 \\
ns(u1,u23) &= 1/2 \\
n(u23,u23) &= 1/2
\end{align*}
\]
Big Idea

You can increase precision by tracking additional predicates
Cyclicity predicate

\[ c[x]() = \exists v_1, v_2: x(v_1) \land n^*(v_1, v_2) \land n^+(v_2, v_2) \]

From the abstract graph alone we cannot tell there are no cycles, but the predicate tells us this is the case.
Cyclicity predicate

\[ c[x]() = \exists v_1, v_2: x(v_1) \land n^*(v_1, v_2) \land n^+(v_2, v_2) \]

\[ c[x]() = 1 \]

\[ c[x]() = 1 \]
Heap Sharing predicate

\[ is(v) = \exists v_1, v_2: n(v_1, v) \land n(v_2, v) \land v_1 \neq v_2 \]
Heap Sharing predicate

$$is(v) = \exists v_1, v_2: n(v_1, v) \land n(v_2, v) \land v_1 \neq v_2$$

$$is(v) = 0 \quad is(v) = 1 \quad is(v) = 0$$
Reachability predicate

\[ t[n](v1, v2) = n^*(v1,v2) \]
Additional Instrumentation predicates

reachable-from-variable-x(\nu)

c_{fb}(\nu) = \forall v_1: f(v, v_1) \supseteq b(v_1, v)

tree(\nu)

dag(\nu)

inOrder(\nu) =

\forall v_1: n(v, v_1) \rightarrow dle(v, v_1)
Instrumentation (Summary)

- Refines the abstraction
- Adds global invariants

But requires update-formulas
(generated automatically in TVLA2)
Partial Concretization (focus)

Helpful in making transfer functions more precise
Expand an abstract heap into a collection of more concrete
Partial Concretization Based on Transformer ($s = \text{Top} \rightarrow n$)

$s'(v) = \exists v_1: \text{Top}(v_1) \land n(v_1, v)$