Pentagons

Based on Logozzo & Fähndrich. Pentagons: [...] Science of Computer Programming 75(9) 2010

Sebastian Hack

Compiler Construction
W2017

Saarland University, Computer Science
int binarySearch(int[] array, int value) {
    int l = 0, u = array.length - 1;
    while (l <= u) {
        int i = (l + u) / 2;
        int v = array[i];
        if (v == value) return i;
        if (v < value) l = i + 1;
        else u = i - 1;
    }
    return ~l;
}

Java requires to throw an exception if the array access is out of bounds.
Motivation

So the code that is really executed is:

```java
int binarySearch(int[] array, int value) {
    int l = 0, u = array.length - 1;
    while (l <= u) {
        int i = (l + u) / 2;
        int v;
        if (i < 0 || i >= array.length) throw new ... 
        else v = array[i];
        if (v == value) return i;
        if (v < value) l = i + 1;
        else u = i - 1;
    }
    return ~l;
}
```

- Apparently, the condition is always true and the compiler should eliminate the bounds check and remove the throw.
- With interval analysis we only get the bound $i \in [0, \infty]$
- Domain not powerful enough to provide relational information $i < \text{array.length}$
Strict Upper Bounds Domain (sub)

- Represent strict inequalities, like \( x < y \)
- Domain: \( Var \rightarrow \mathcal{P}(Var) \)
  Map each \( x \) to all variables that are strictly greater than \( x \)
- Concretization: \( \gamma_{\text{sub}} : s \mapsto \{\text{state } \sigma \mid \forall xy : y \in s(x) \Rightarrow \sigma(x) < \sigma(y)\} \)
Strict Upper Bounds Domain (sub)

- Represent strict inequalities, like $x < y$

- Domain: $\text{Var} \rightarrow \mathcal{P}(\text{Var})$
  Map each $x$ to all variables that are strictly greater than $x$

- Concretization: $\gamma_{\text{sub}} : s \mapsto \{\text{state } \sigma \mid \forall xy : y \in s(x) \Rightarrow \sigma(x) < \sigma(y)\}$

- Join: $s \sqcup_{\text{sub}} t : \iff \lambda x.(s(x) \cap t(x))$
  implies ordering via $a \sqsubseteq_{\text{sub}} b \iff a \sqcup_{\text{sub}} b = b$

- $\top = \lambda x.\emptyset$ and $\bot = \lambda x.\text{Var}$
Closures

- Because \(<\) is transitive, there are many elements in sub that concretize to the same set of states, e.g. consider

  \[
  s_1 = [x \mapsto \{y\}, y \mapsto \{z\}]
  \]

  \[
  s_2 = [x \mapsto \{y, z\}, y \mapsto \{z\}]
  \]

  for which we have \(\gamma(s_1) = \gamma(s_2)\)

- When joining, it actually makes a difference which one we have:

  \[
  s_1 \sqcup [x \mapsto \{z\}] = \top
  \]

  \[
  s_2 \sqcup [x \mapsto \{z\}] = [x \mapsto \{z\}]
  \]

- One can show that \(\gamma_{\text{sub}}\) preserves meets and therefore, for all \(s, s'\) with \(\gamma(s) = \gamma(s')\) we have \(\gamma(s) = \gamma(s) \cap \gamma(s') = \gamma(s \sqcap_{\text{sub}} s')\)

- Hence, there is a best abstraction \(\alpha(c)\) for a given set of concrete states \(c = \gamma(s)\)

  \[
  (\alpha \circ \gamma)(s) = \bigcap \{s' \mid \gamma(s') = \gamma(s)\}
  \]
Closures

- To make the join most precise one could compute the closure $\alpha \circ \gamma$ and join with the best abstractions.

- The closure operator can in practice be expensive: In sub one has to compute the transitive closure of the relation represented by an abstract element.

- In practice other operations that overapproximate the join are imaginable.
Reduced Product

- Using sub without intervals does not help in proving the array access in bounds in our example. Information about constants missing

- Hence: Use both analyses: pentagons $= \text{sub} \times \text{intervals}$
Reduced Product

- In the product, there are typically multiple abstract elements that are concretized to the same value:

\[
\gamma\left(\langle\{x \mapsto [0, 100], y \mapsto [0, 50]\}, \{x < y\}\rangle\right)
= \gamma\left(\langle\{x \mapsto [0, 49], y \mapsto [1, 50]\}, \{x < y\}\rangle\right)
\]

- Therefore, one also gets a closure operator that gives the best abstraction in sub \(\times\) intervals for a given abstraction:

\[
\langle s, b \rangle \mapsto \langle s^*, b^* \rangle
\]

\[
b^* = \bigcap_{\{x < y\} \in s} \lbrack x < y \rbrack^\#(b)
\]

\[
s^* = \lambda x.s(x) \cup \{y \in Var \mid x^u < y^\ell\} \quad \text{with} \quad b(z) = [z^\ell, z^u]
\]
Practice

- Applying this closure operator might be expensive. In pentagons, it is $O(\text{Var}^2)$

- To get the best precision, one has to do it before every operation: join, application of abstract transformer.

- Hence, in practice, one uses
  - A less precise but more efficient join, e.g. in Pentagons, ignore sub information for interval join.
  - Modified abstract transformers, that integrate information from both domains, intervals and sub. For example, consider subtraction with:

$$[[r \leftarrow x - y]]^\# \langle s, b \rangle = \langle s[r \mapsto s_r], b[r \mapsto b_r] \rangle \quad \text{with}$$

$$b_r = [[x - y]]_{\text{intv}}(b)(r) \cap ((y < x) \in s ? [1, \infty] : T_{\text{intv}})$$

$$s_r = y^\ell > 0 ? \{x\} \cup s(x) : \emptyset$$