Pentagons

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

Sebastian Hack

Compiler Construction
W2017

Saarland University, Computer Science

Motivation

```java
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:

```
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 $\mathtt{i} \in [0, \infty]$
- Domain not powerful enough to provide relational information

  `i < array.length`

Strict Upper Bounds Domain (sub)

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

Strict Upper Bounds Domain (sub)

- Represent strict inequalities, like $\mathtt{x} < \mathtt{y}$
- Domain: $\mathit{Var} \to \mathcal{P}(\mathit{Var})$
  Map each $\mathtt{x}$ to all variables that are strictly greater than $\mathtt{x}$
- Concretization: $\gamma_{\mathsf{sub}} : s \mapsto \{\text{state } \sigma \mid \forall \mathtt{xy} : \mathtt{y} \in s(\mathtt{x}) \Rightarrow \sigma(\mathtt{x}) < \sigma(\mathtt{y})\}$
- Join: $s \sqcup_{\mathsf{sub}} t :\iff \lambda \mathtt{x}.(s(\mathtt{x}) \cap t(\mathtt{x}))$
  implies ordering via $a \sqsubseteq_{\mathsf{sub}} b \iff a \sqcup_{\mathsf{sub}} b = b$
- $\top = \lambda \mathtt{x}.\emptyset$ and $\bot = \lambda \mathtt{x}.\mathit{Var}$

Closures

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

$$s_1 = [\mathtt{x} \mapsto \{\mathtt{y}\}, \mathtt{y} \mapsto \{\mathtt{z}\}]$$
$$s_2 = [\mathtt{x} \mapsto \{\mathtt{y}, \mathtt{z}\}, \mathtt{y} \mapsto \{\mathtt{z}\}]$$

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

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

$$s_1 \sqcup [\mathtt{x} \mapsto \{\mathtt{z}\}] = \top$$
$$s_2 \sqcup [\mathtt{x} \mapsto \{\mathtt{z}\}] = [\mathtt{x} \mapsto \{\mathtt{z}\}]$$

- One can show that γ_{sub} preserves meets and therefore, for all s, s' with $\gamma(s) = \gamma(s')$ we have $\gamma(s) = \gamma(s) \cap \gamma(s') = \gamma(s \sqcap_{\mathsf{sub}} s')$
- Hence, there is a best abstraction $\alpha(c)$ for a given set of concrete states $c = \gamma(s)$

$$(\alpha \circ \gamma)(s) = \bigsqcap \{s' \mid \gamma(s') = \gamma(s)\}$$

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 $=$ sub $\times$ intervals

Reduced Product

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

$$\begin{array}{rl} & \gamma(\langle \{x \mapsto [0,100], y \mapsto [0,50]\}, \{x < y\}\rangle) \\ = & \gamma(\langle \{x \mapsto [0,49], y \mapsto [1,50]\}, \{x < y\}\rangle) \end{array}$$

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

$$\langle s, b\rangle \mapsto \langle s^*, b^*\rangle$$
$$b^* = \bigsqcap_{\{\mathtt{x}<\mathtt{y}\}\in s} [\![\mathtt{x} < \mathtt{y}]\!]^\sharp(b)$$
$$s^* = \lambda \mathtt{x}.s(\mathtt{x}) \cup \{\mathtt{y} \in \mathit{Var} \mid \mathtt{x}^u < \mathtt{y}^\ell\} \qquad \text{with } b(z) = [z^\ell, z^u]$$

Practice

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

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

$$b_r = \llbracket \mathtt{x} - \mathtt{y} \rrbracket^\sharp_{\mathsf{intv}}(b)(\mathtt{r}) \cap ((\mathtt{y} < \mathtt{x}) \in s \ ?\ [1, \infty] : \top_{\mathsf{intv}})$$

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