Interactive Abstract Interpretation

Julian Erhard
julian.erhard@tum.de

Supervisors: Helmut Seidl & Dirk Beyer
Collaborators: Michael Schwarz (CONVeY) & Sarah Tilscher (CONVeY) & Simmo Saan (UTartu) & Vesal Vojdani (UTartu)

Goal
Putting static analysis at the fingertips of the developer

Yard-Sticks

1. Response time:
   Time it takes for the analysis to finish after a program change
2. Consistency:
   Level of precision that is retained compared to a from-scratch analysis
3. Usability:
   Level of integration into the developer workflow

Response-Time

Incremental analysis [2]:
• Exploit dependencis tracked by solver
• Reuse analysis results where possible
• Detect changed functions \( F_{\text{changed}} \)
• Mark results influenced by \( f \in F_{\text{changed}} \) as unstable
• Restart analysis from return-node \( r_{\text{main}} \) of \( \text{main} \)

Reluctant destabilization:
• First reanalyze \( f \in F_{\text{changed}} \)
• Only destabilize call-sites of \( f \) if results for node \( r_f \) changed

Fine-grained change detection:
• Match control-flow-graphs of \( f \in F_{\text{changed}} \) with previous version
• Reuse results for nodes within \( f \) that –can be matched, and –do not have any new (indirect) predecessor

Incremental postsolver:
• Track unknowns not touched by reanalsis
• Reuse warnings for such unknowns that are still live

```c
int f(int x) {
    int y = x + 42;
    if (x < 0)
        return y;
    return y;
}

int m() {
    int a = 25;
    printf(\"hello\", y);
    return b;
}
```

Listing 1: Code example with change. A `printf` is added to `f`.

Consistency

Low precision loss through incremental analysis out-of-the-box.

Issue: Values of flow-insensitive unknowns accumulate over reanalyses.

Solution: Restart subset \( G \) of globals, as follows:
• Reset \( g \in G \) to \( \perp \)
• Set unknowns that side-effect \( g \) and all that (transitively) depend on them to unstable
• Reanalysis from \( r_{\text{main}} \) triggers side-effects to \( g \) in new equation system

\( \implies \) New values for \( g \in G \) without contributions from previous runs.

Usability

IDE integration via MagpieBridge [1], using server mode for Goblint:
• Communication IDE \( \iff \) GOBLINT via sockets
• Configuration is maintained
• Works without restart of analyzer and reparsing of unchanged code

Results

Thread-modular, partially context-sensitive analysis with intervals and race-detection performed on commits in `std`, `chrony`, `figlet` repositories.

```
Figure 1: Cumulative distribution of commits analyzed within the given run time for setups (1)-(4) on `std`.
```

```
Table 1: Features active in confs. (1)-(4).

<table>
<thead>
<tr>
<th>Feature</th>
<th>(1)</th>
<th>(2)</th>
<th>(3)</th>
<th>(4)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>std</code></td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td><code>chrony</code></td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td><code>figlet</code></td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table 2: Median speedups of solving (incl. post solving) and overall run times achieved by configurations (2)-(4) compared to (1) on the benchmark repositories.

```

Conclusion

• Considerable speedups by interactive analysis
• Smaller overall speedups on smaller projects, due to other bottlenecks
• Restarting mitigates precision loss on flow-insensitive information

References
