**Overview of Transformations in ConVeY**

<table>
<thead>
<tr>
<th>Input</th>
<th>Transformation</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verification Information</td>
<td>Validation as Program</td>
<td>Model</td>
</tr>
<tr>
<td>Validation as Program</td>
<td>Output</td>
<td>Witness</td>
</tr>
<tr>
<td>Model</td>
<td>Verdict</td>
<td></td>
</tr>
</tbody>
</table>

**Own Contributions**

Leveraging ACSL [7]: ACSL is a specification language for C programs, used by interactive verifiers, while automatic verifiers store invariant information in so-called witnesses. By translating between these formats, we can create novel interactions between interactive and automatic verifiers.

MetaVal [6]: Verification Witnesses increase the trust in the verification result, since they allow to validate the proof. The number of available validators however is limited. MetaVal Encodes the validation as a verification problem into the original program, which allows to use off-the-shelf verifiers for validation.

Unifying Loop Abstractions [5]: Loop abstractions are easy yet often very powerful program transformations which correspond to transition invariants of the program. Determining which of the available loop abstractions need to be applied for a successful proof is not straightforward. Our approach allows to apply loop abstractions in a CEGAR-style approach, determining the right level of abstraction automatically.

**MetaVal**

Validation Witnesses increase the trust in the verification result, since they allow to validate the proof. The number of available validators however is limited. MetaVal Encodes the validation as a verification problem into the original program, which allows to use off-the-shelf verifiers for validation.

Unifying Loop Abstractions [5]: Loop abstractions are easy yet often very powerful program transformations which correspond to transition invariants of the program. Determining which of the available loop abstractions need to be applied for a successful proof is not straightforward. Our approach allows to apply loop abstractions in a CEGAR-style approach, determining the right level of abstraction automatically.

**Interfacing Tools with ACSL**

Loop abstractions are woven into the CFA of the program. State space exploration is then guided into the abstractions. On spurious counterexamples, CEGAR refinement will lead to other paths being taken. Interfacing with the CFA allows to reuse existing analysis in CPAchecker straight away.

**References**


