Verification of programs with arrays using Horn Clauses

Julien Braine
ENS de Lyon, France

Ph.D advisors: Laure Gonnord, David Monniaux
julien.braine@ens-lyon.fr

Our Team: CASH

Static Analyses in the team
- Design new low-cost analyses to allow compiler optimizations
- Design safe domain specific languages to avoid programmer bugs
- Design precise, non domain specific, static analysis to ensure correctness of code

Static analysis to ensure functional correctness of code

Goal: Ensure code correctness ≠ finding as many bugs as possible
Setting: Programs have assertions describing the desired behavior
My focus: The case of programs with data-structures, especially arrays

Setting: Horn clauses as semantics of a program

Example: array copy program.

```
int j = rand()%N;
for(i=0; i<N; i++) {
    a[i] = b[i];
} assert (b[j] == a[j]);
```

Horn clauses: A logical formula expressing the assertion and the program’s semantics.

Shape of Horn clauses:
- Existentially quantified predicates, represent possible values at each program point
- Universally quantified variables to define the transition relation

Result of a Horn clauses solver:
- SAT ⇒ Found instantiation for predicate ⇒ Program correct
- UNSAT ⇒ No possible predicate instantiation ⇒ Program is buggy
- Unknown or timeout ⇒ Unable to find instantiation or disprove its existence

Translation from programs: Abstracts memory, and specifics of the language

Example:

```
True ∧ j < N → Start(a, b, N, i, j)
Start(a, b, N, i, j) → Loop(a, b, N, 0, j)
Loop(a, b, N, i, j) ∧ ∀ j' = a[i] + b[i] ∧ i < N → Loop(a', b, N, i + j, j)
Loop(a, b, N, i, j) ∧ i ≥ N → Assert(a, b, N, i, j)
Assert(a, b, N, i, j) ∧ a[i] ≠ b[j] → False
```

Horn Clauses
- can express the semantics of programs with no information loss
- have clear and easily defined semantics (its a logical formula!)
- have a very simple unified syntax ⇒ very good intermediate representation
- tools (such as SealHorn) can generate Horn clauses from programs (LLVM bytecode)
- have efficient solvers such as Z3

Related work: Interpolants and Abstract Interpretation to solve Horn Clauses

Interpolants

<table>
<thead>
<tr>
<th>k = 0</th>
<th>Yes</th>
<th>The program is buggy!</th>
</tr>
</thead>
</table>
| k = k + 1 | No | From proof, generalize "why no counterexample",
| | | Does the generalization prove correctness? |

Abstract Interpretation

Abstract Interpretation consists in over-approximating the set of possible values at each program (the predicates) using an abstract domain.

Comparison of these techniques

<table>
<thead>
<tr>
<th>Requires</th>
<th>Abstract domain and abstraction of program’s operations</th>
<th>Interpolants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Soundness</td>
<td>Fixed by abstract domain</td>
<td>Fixed by underlying logic</td>
</tr>
<tr>
<td>Precision</td>
<td>Uses assertion</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Termination</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Predictable failures</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Horn Solver</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Handles well arrays</td>
<td></td>
</tr>
</tbody>
</table>

PhD intro: Handling arrays in Horn clauses

Problem: Arrays ⇒ quantified invariants ⇒ no good enough interpolation technique.
Solution: Create a new Horn problem without arrays by using abstract interpretation and solve it with a state of the art solver.

Example:
- Program: array copy.
- Technique: SAS15-16, Monniaux & Alberti & Gonnord
- Abstract domain: Cell abstraction. An array \( a \) is abstracted by its cells, that is \( \{ \langle k, a[k] \rangle, k \in N \} \).
- Using the abstract domain in Horn clauses (simple version) Replace \( P(a, v) \) by \( P^a(k, a[k], v) \) in the Horn clauses
- Fully removing arrays: no array type in predicates ⇒ Apply array axioms ⇒ no arrays
- Solving the abstracted problem: Launch Z3. Answer: SAT in <1s

Tool: Vaphor by Braine & Monniaux & Gonnord

My PhD

In the context of Horn clauses, my goal is:
- Improve existing array abstractions
- Function summaries for scaling
- Implement a verified equivalent of STL (but in C)
- Extend to other data-structures
- Implement and test these techniques in a tool (FrameC?)
- Use this “STL” in verified algorithms

http://perso.ens-lyon.fr/julien.braine/

Related Work: Interpolants and Abstract Interpretation to solve Horn Clauses

Interpolants

<table>
<thead>
<tr>
<th>k = 0</th>
<th>Yes</th>
<th>The program is buggy!</th>
</tr>
</thead>
</table>
| k = k + 1 | No | From proof, generalize "why no counterexample",
| | | Does the generalization prove correctness? |

Abstract Interpretation

Abstract Interpretation consists in over-approximating the set of possible values at each program (the predicates) using an abstract domain.

Comparison of these techniques

<table>
<thead>
<tr>
<th>Requires</th>
<th>Abstract domain and abstraction of program’s operations</th>
<th>Interpolants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Soundness</td>
<td>Fixed by abstract domain</td>
<td>Fixed by underlying logic</td>
</tr>
<tr>
<td>Precision</td>
<td>Uses assertion</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Termination</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Predictable failures</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Horn Solver</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Handles well arrays</td>
<td></td>
</tr>
</tbody>
</table>

PhD intro: Handling arrays in Horn clauses

Problem: Arrays ⇒ quantified invariants ⇒ no good enough interpolation technique.
Solution: Create a new Horn problem without arrays by using abstract interpretation and solve it with a state of the art solver.

Example:
- Program: array copy.
- Technique: SAS15-16, Monniaux & Alberti & Gonnord
- Abstract domain: Cell abstraction. An array \( a \) is abstracted by its cells, that is \( \{ \langle k, a[k] \rangle, k \in N \} \).
- Using the abstract domain in Horn clauses (simple version) Replace \( P(a, v) \) by \( P^a(k, a[k], v) \) in the Horn clauses
- Fully removing arrays: no array type in predicates ⇒ Apply array axioms ⇒ no arrays
- Solving the abstracted problem: Launch Z3. Answer: SAT in <1s

Tool: Vaphor by Braine & Monniaux & Gonnord

My PhD

In the context of Horn clauses, my goal is:
- Improve existing array abstractions
- Function summaries for scaling
- Implement a verified equivalent of STL (but in C)
- Extend to other data-structures
- Implement and test these techniques in a tool (FrameC?)
- Use this “STL” in verified algorithms

http://perso.ens-lyon.fr/julien.braine/