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

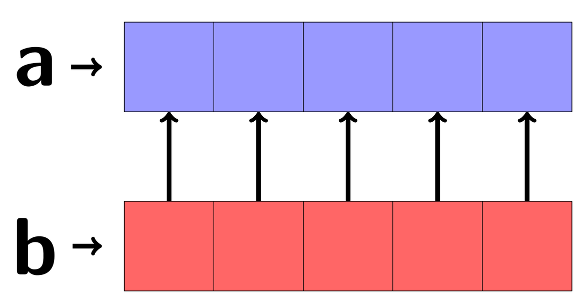
Static analysis to ensure functional correctness of code

Goal: Ensure code correctness $\neq$ finding as many bugs as possible
Setting: Programs have assertions describing the desired behavior
General problem: How to check all possible runs?
My focus: The case of programs with data-structures, especially arrays

Setting: Horn clauses as semantics of a program

Example: array copy program.

```
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 $\Rightarrow$ Found instanciation for predicate $\Rightarrow$ Program correct
- UNSAT $\Rightarrow$ No possible predicate instantiation $\Rightarrow$ Program is buggy
- Unknown or timeout $\Rightarrow$ Unable to find instanciation or disprove its existence

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

Example:

$$\begin{array}{lrl} True & \wedge\, j < N \longrightarrow & Start(a,b,N,i,j) \\ Start(a,b,N,i,j) & \longrightarrow & Loop(a,b,N,0,j) \\ Loop(a,b,N,i,j) & \wedge\, a' = a[i \leftarrow b[i]] \wedge i < N \longrightarrow & Loop(a',b,N,i+1,j) \\ Loop(a,b,N,i,j) & \wedge\, i \geq N \longrightarrow & Assert(a,b,N,i,j) \\ Assert(a,b,N,i,j) & \wedge\, a[j] \neq b[j] \longrightarrow & False \end{array}$$

Horn Clauses

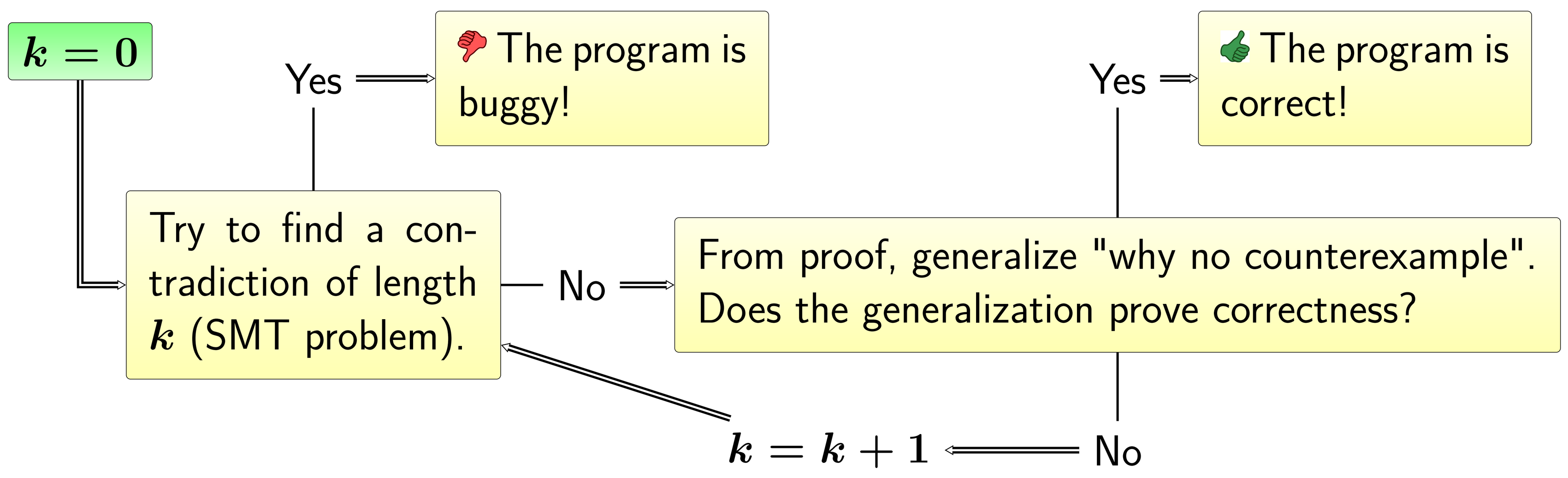
- can express the semantics of programs with no information loss
- have clear and easily defined semantics (its a logical formula!)
- have a very simple unified syntax $\Rightarrow$ very good intermediate representation
- tools (such as SeaHorn[1]) can generate Horn clauses from programs (LLVM bytecode)
- have efficient solvers such as Z3[2]

[1] https://seahorn.github.io/
[2] https://github.com/Z3Prover/z3/
[3] https://hal.archives-ouvertes.fr/hal-01162795/document
https://hal.archives-ouvertes.fr/hal-01206882v3/document
[4] https://github.com/vaphor

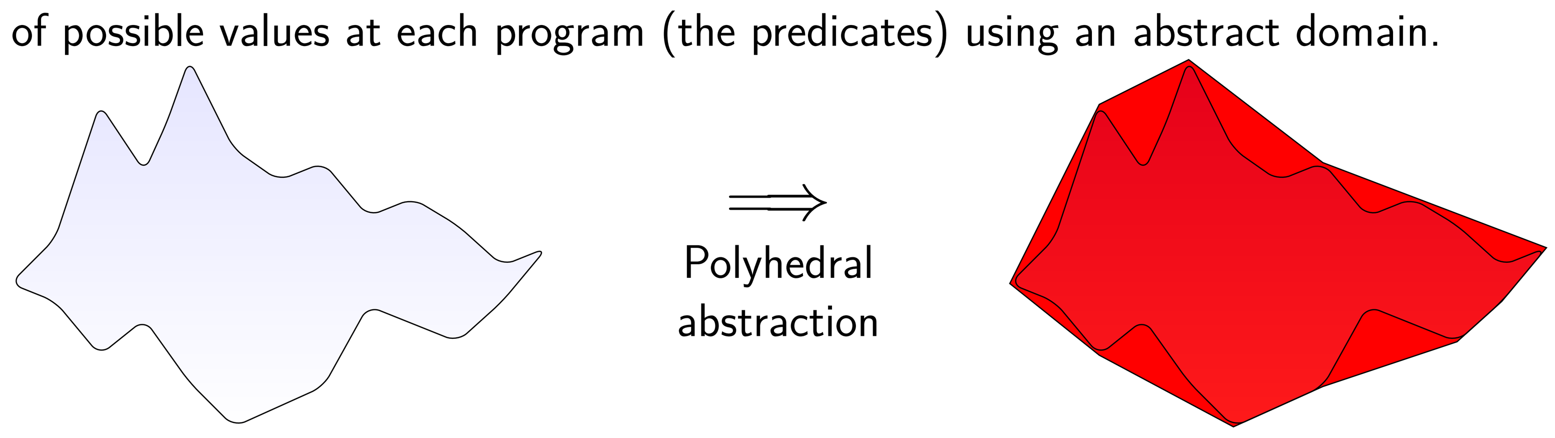
1 2 3 3 4

Related work: Interpolants and Abstract Interpretation to solve Horn Clauses

Interpolants



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

	Abstract Interpretation	Interpolants
Requires	Abstract domain and abstraction of program's operations	Interpolation technique for the given theory
Soundness	👍	👍
Precision	Fixed by abstract domain	Fixed by underlying logic
Uses assertion	👎	👍
Termination	👍	👎
Predictable failures	👍	👎
Horn Solver	?	Z3
Handles well arrays	👎	👎

PhD intro: Handling arrays in Horn clauses

Problem: Arrays $\Rightarrow$ quantified invariants $\Rightarrow$ 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[3]
- **Abstract domain:** Cell abstraction.
  An array a is abstracted by its cells, that is $\{(k, a[k]), k \in \mathbb{N}\}$.
- **Using the abstract domain in Horn clauses (simple version)**
  Replace $P(a, v)$ by $P^{\#}(k, a[k], v)$ in the Horn clauses
- **Fully removing arrays**: no array type in predicates $\Rightarrow$ Apply array axioms $\Rightarrow$ no arrays
- **Solving the abstracted problem**: Launch Z3. *Answer:* SAT in <1s. 👍

Tool: *Vaphor* by Braine & Monniaux & Gonnord[4]

My PhD

In the context of Horn clauses, my goal is:

Improve existing array abstractions	$\longrightarrow$	Extend to other data-structures
Function summaries for scaling	$\longrightarrow$	Implement and test these techniques in a tool (FramaC?)
Implement a verified equivalent of STL (but in C)	$\longrightarrow$	Use this "STL" in verified algorithms

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

http://www.inria.fr/ http://www.ens-lyon.eu/ http://www.ens-lyon.fr/LIP/ https://www.universite-lyon.fr/ http://www.cnrs.fr/