Logic in program analysis and verification

Patrick Cousot

NYU, New York
pcousot@cs.nyu.edu  cs.nyu.edu/~pcousot

Sunday, Nov 15th, 2020
Subject of discussion

- For **program specification and verification**, logic is a natural choice.
- However, for **static analysis**, logic is rarely used, even as a user interface.
- We briefly discuss the weaknesses of logic from this **static analysis perspective**.
Which logic for specification?
Specification

- **decidable logics** (such as Presburger arithmetic [12]):
  - validity can be mechanically checked
  - incomplete (the invariant of a program that computes the multiplication $*$ using iteration and addition $+$ is not expressible)

- **first-order logic**:
  - undecidable (user-interaction is needed for proofs)
  - incomplete (no recursion mechanism, transitive closure is not expressible [11])

- **higher-order logic**:
  - necessary to discuss the relative completeness go Hoare logic
  - necessary to discuss the soundness of static analyzers (e.g. hyperproperties in $\mathcal{P}(\mathcal{P}(S))$ where $S$ is the semantic domain)
Which logic for property representation in a static analyzer?
Internal representation of abstract properties

- **great advantage: uniform representation** by (the abstract syntax) of a formula in the logic
  - many operations have simple implementations (e.g. connectives)
  - exploited in the static analysis of Prolog [10]
- **great disadvantage: uniformity**
  - no (useful) normal form
  - efficient algorithms require specific representations (e.g. matrices + systems of generators for linear equalities or inequalities [8])
  - algorithmically, syntax-based representation uniformity is not tenable
Abstract domains
Abstract domains

- order-theoretic/algebraic concept of properties (representation + operations)
- hard to translate in logic (e.g. how to express “to be a number between $a$ and $b$”)
- the semantics is formally defined by concretization to sets
- operations (e.g. logical connectives, transformers) are (predictable and efficient) algorithms

- in logic, the failure of theorem provers or SMT solvers may be very hard to explain [9]
Combinations of abstract domains

- the uniformity of representation of properties is lost with abstract domains
- combinations of abstract domains handle non-uniform representations
- communication of shared information between abstract domains

- example: the reduced product $[3]$ for conjunction
- the combination of theories in SMT solvers is a reduced product $[5]$ (the shared information is equalities and disqualifies for Nelson-Oppen $[13]$)
Induction
Proofs by induction

- infering inductive arguments in proofs is the basis for verification and analysis of programs
- asking the users for induction hypotheses makes verification simpler than program analysis [6]
- hardly scale up (invariants are much larger than programs [4])

- induction in logic is predefined
- no mechanism in logic to specify how to automate approximate induction or co-induction
- the complexity of an object and its logical denotation may be completely unrelated.
Extrapolation and interpolation

- induction tailored to a level of abstraction [1]
- often based on geometric considerations (e.g. widenings extrapolate in the direction of growth)
- finitary abstract domains are not expressive [2] (e.g. liquid types [14])
- the evolution of the iterates is monitored for induction [7]
Conclusion

- logic reduces the representations of properties and formal reasonings to purely syntactic manipulations (copy/paste :)
- this is great for logicians to reason about proofs (≠ making proofs)
- mathematicians do not use logics to make proofs
Conclusion

- logic reduces the representations of properties and formal reasonings to purely syntactic manipulations (copy/paste :)
- this is great for logicians to reason about proofs (≠ making proofs)
- mathematicians do not use logics to make proofs

- computer scientists do, maybe that’s the problem
The End, Thank you
Bibliography I

Abstracting induction by extrapolation and interpolation.

Comparing the Galois connection and widening/narrowing approaches to abstract interpretation.

Systematic design of program analysis frameworks.

Why does Astrée scale up?

Theories, solvers and static analysis by abstract interpretation.

Program analysis is harder than verification: A computability perspective.
In CAV (2), volume 10982 of Lecture Notes in Computer Science, pages 75–95. Springer, 2018.

A^2i: abstract^2 interpretation.

Automatic discovery of linear restraints among variables of a program.
The strategy challenge in SMT solving.  

[10] Isabel García-Contreras, José F. Morales, and Manuel V. Hermenegildo.  
Incremental analysis of logic programs with assertions and open predicates.  

On transitive closure logic.  

[12] Christoph Haase.  
A survival guide to Presburger arithmetic.  

Simplification by cooperating decision procedures.  

Liquid types.  