

Abstract Interpretation

<http://d3s.mff.cuni.cz>

Department of
Distributed and
Dependable
Systems



Pavel Parízek



FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Abstract interpretation

- Theoretical framework unifying different program analyses
- Formal underpinning of **sound** and **correct** static analyses
- Practice: Code Contracts, Astree, Polyspace

Reusing concepts from static analysis

- Control-flow graphs
- Finite lattices
- Transfer functions
- Fixed points
 - Iterative computation
 - Work list algorithm

New concepts

- Explicit abstraction
- Concrete domain
- Abstract domain
- Galois connections
- **Purpose: constructing sound abstractions**

Example program

```
int compute(int x, int y) {  
    Requires (x >= 0 && y >= 0);  
  
    y = y + 1;  
    z = x + y;  
  
    Assert (z >= 1);  
  
    return z;  
}
```

Concrete domain

- Finite lattice $C = (E_C, \sqsubseteq^C)$
 - Set of concrete elements E_C
 - Partial order \sqsubseteq^C on E_C
- Notation abuse
 - Symbol C means both the finite lattice and the set of concrete elements
- Example
 - Possible values of an integer variable
 - $E_C = 2^{\mathbb{N}}$ (all possible subsets)
 - $\sqsubseteq^C = \subseteq$ (plain subset ordering)

Abstract domain

- Finite lattice $A = (E_A, \sqsubseteq^A, \perp, \top, \sqcup, \sqcap)$
 - Set of abstract elements E_A
 - Partial order \sqsubseteq^A on E_A
 - Least abstract element \perp
 - Greatest abstract element \top
 - Join operator \sqcup
 - Meet operator \sqcap

Abstract domain: Intervals

- Definition

- $E_A = \{ [x, y] \mid x, y \in \mathbb{Z} \cup \{-\infty, +\infty\} \}$
- Partial order \sqsubseteq^A : interval inclusion
- \perp is empty interval
- $\top = \{-\infty, +\infty\}$

- Examples

- $[0, 2] \sqsubseteq^A [0, 4]$
- $[0, 2] \not\sqsubseteq^A [1, 3]$

Relation between domains

- Abstraction function $\alpha : C \mapsto A$
 - Computes the most precise abstract representation
- Concretization function $\gamma : A \mapsto C$
- Example: interval domain
 - $\alpha(S) = [\min(S), \max(S)], S = \{s_1, \dots, s_N\}$
 - $\gamma([u, v]) = \{x \in \mathbb{Z} \mid u \leq x \leq v\}$

Galois connection

- Necessary conditions
 - Both functions α and γ are monotone
 - $\forall a \in A, c \in C : \alpha(c) \sqsubseteq^A a \Leftrightarrow c \sqsubseteq^C \gamma(a)$
- Relation between partially ordered sets A and C
- Characterizes **sound abstraction**
 - We can lose precision (over-approximating)

$$c \sqsubseteq^C \gamma \circ \alpha(c) \quad \alpha \circ \gamma(a) \sqsubseteq^A a$$

Transfer functions

- Goal: represent effects of program statements
- Concrete transfer function $\tau_C : C \mapsto C$
 - Expresses concrete semantics of program statements
- Abstract transfer function $\tau_A : A \mapsto A$
 - Expresses abstract semantics of program statements
- Relation: $\forall a \in A : \tau_C \circ \gamma(a) \subseteq \gamma \circ \tau_A(a)$
- Concrete program P_C
- Abstract program P_A

How to compute solution

- Input problem
 - Concrete program P_C
 - Abstract domain A
 - Functions α and γ
 - Transfer function τ_A
- Representing information
 - Separate analysis value for each program variable
 - One large set with values for all program variables

How to compute solution

- Input analysis problem
- Representing information
- Approach: find $lfp(P_A)$
 - Symbol $lfp \sim$ least fixed point
 - Using the **work-list** algorithm
- Result: $lfp(P_C) \sqsubseteq^C \gamma(lfp(P_A))$

Divergence

- Problem: **fixpoint computation may diverge**
 - Why: infinite increasing chains (sequences)
- Ascending Chain Condition (ACC)
 - Strictly ascending sequence of elements reaches some fixed point (terminates)
 - Example: $a_1 \sqsubseteq a_2 \sqsubseteq a_3 \sqsubseteq \dots \sqsubseteq a_n = a_{n+1} = a_{n+2}$
- Solution: **Widening operator**

Widening

- Widening operator $\nabla : A \times A \mapsto A$
 - $\forall a_1, a_2 \in A : (a_1 \sqsubseteq a_1 \nabla a_2) \wedge (a_2 \sqsubseteq a_1 \nabla a_2)$
- Sequence $w_0 = a_0, \dots, w_{i+1} = w_i \nabla a_{i+1}$ not strictly increasing
 - ➔ fixed point computation will terminate
- Other benefit: faster convergence

Widening: Intervals

- “Intervals” does not satisfy ACC
- Option 1
 - Keep stable bounds (preserve the value)
 - Extrapolate unstable bounds ($\{-\infty, +\infty\}$)
- Option 2
 - Keep valid bounds from the first operand
 - Extrapolate bounds otherwise ($\{-\infty, +\infty\}$)

Widening

- Consequence: losing precision
- Practice
 - Use widening operator
 - on backward edges in CFG
 - for really too big intervals
- Remedy: Narrowing
 - Complementary operator
 - Goal: improving precision

Numerical abstract domains

- Non-relational domains
 - Program variables treated separately
 - **Examples: signs, intervals**
- Relational domains
 - Consider relations between variables
 - **Example: predicate abstraction**

Cartesian abstraction

- Very important special case
- Key idea
 - α : flattening the analysis information
 - γ : restores all possible combinations
- Pros: better scalability
- Cons: loses precision
- Example: predicate abstraction

Other abstract domains

- Octagon
 - Values represented as constraints $\pm x \pm y \leq c$
- Polyhedral
 - Values represented as constraints $\sum_i a_{ij} * x_i \leq c_j$
- Linear equations
- Strings: Prefix, Suffix, Character inclusion, ...

Using abstract interpretation



- 1) Design the abstract domains
- 2) Define abstraction functions
- 3) Design widening operators
- 4) Define all transfer functions

Multiple abstract domains

- Combination

- Two abstract domains A_1, A_2
- Cartesian product: $A_1 \times A_2$
- $\forall \langle a_1, a_2 \rangle \in A_1 \times A_2 : \gamma_{A_1 \times A_2}(\langle a_1, a_2 \rangle) \subseteq (\gamma_{A_1}(a_1) \cap \gamma_{A_2}(a_2))$

- Composition

- One concrete domain C
- Abstract domains A_1, A_2
- Galois connection
 - α_1, γ_1 between C and A_1
 - α_2, γ_2 between A_1 and A_2
- We get the connection between C and A_2
 - Functions: $\alpha_2 \circ \alpha_1, \gamma_1 \circ \gamma_2$

- Clousot
 - Program analyzer for Code Contracts (C#/.NET)
 - Verifies method contracts and low-level errors
 - <https://www.microsoft.com/en-us/research/project/code-contracts/>
- Astrée
 - Static analyzer for programs written in C
 - Programs without dynamic memory allocation and recursion
 - Industrial applications (Airbus A340 SW)
 - <http://www.astree.ens.fr/>
- Polyspace
 - Static analysis toolset for programs in C/C++/Ada
 - <http://www.mathworks.com/products/polyspace/>

Further reading

- F. Nielson, H. R. Nielson, and Chris Hankin. **Principles of Program Analysis**. Springer, 2005
- P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Mine, and X. Rival. **Why does Astrée Scale Up?** Formal Methods in System Design, 35(3), 2009
- P. Ferrara, F. Logozzo, and M. Fahndrich. **Safer Unsafe Code for .NET**. OOPSLA 2008