



# Static Program Analysis

Lecture 21: Wrap-Up

Summer Semester 2018

Thomas Noll

Software Modeling and Verification Group

RWTH Aachen University

<https://moves.rwth-aachen.de/teaching/ss-18/spa/>

### Dedicated Algorithms for Pointer Analysis

- **null Pointer Analysis:** checks whether dereferencing operations possibly involve `null` pointers
  - with shape analysis:  $x = \text{nil}$  possible for  $x \in \text{Var}$  at  $l \in \text{Lab}$  if there exists heap configuration  $H^\# \in \text{HC}^\#$  at  $l$  which does not contain  $x$
  - simple algorithms based on types (usually flow-insensitive)
  - more advanced approaches employ constraint-based abstract interpretation<sup>1</sup>
- **Points-To Analysis:** yields function  $pt$  that for each  $x \in \text{Var}$  returns set  $pt(x)$  of possible pointer target nodes
  - $x$  and  $y$  may be aliases if  $pt(x) \cap pt(y) \neq \emptyset$
  - with shape analysis: there exists heap configuration  $H^\# \in \text{HC}^\#$  with a node that is referred to by both  $x$  and  $y$
  - algorithms with linear complexity based on type inference and constraint solving<sup>2</sup>
- Tailored algorithms usually **faster** and sometimes **more precise** than shape analysis, but **less general** (only “shallow” properties)
- Fastest algorithms are **flow-insensitive** (points-to relations only added but never removed)

---

<sup>1</sup>[https://doi.org/10.1007/978-3-540-68863-1\\_9](https://doi.org/10.1007/978-3-540-68863-1_9)

<sup>2</sup><https://doi.org/10.1145/237721.237727>

# Further Topics in Static Program Analysis

---

## Correctness of Dataflow Analyses

- **So far:** **semantics** and **dataflow analysis** of programs considered independently (formal soundness proofs only for abstract interpretation; cf. Lecture 10–12)
- Of course both are (and should be) related!
- To this aim: compare results of **concrete semantics** (Definition 10.9) with **outcome of analysis**
- See [Nielson/Nielson/Hankin 2005, Sct. 2.2] for details

### Example 21.1 (Correctness of Constant Propagation)

Let  $c \in \text{Cmd}$ ,  $l \in \text{Lab}_c$ ,  $x \in \text{Var}$ , and  $z \in \mathbb{Z}$  such that  $\text{CP}_l(x) = z$ .  
Then for all  $\sigma_0, \sigma \in \Sigma$  such that  $\langle \text{init}(c), \sigma_0 \rangle \rightarrow^* \langle l, \sigma \rangle$ ,  $\sigma(x) = z$ .

# The Exam

---

## Written Exam

- **Dates:**
  - Thu 26 July, 10:00–12:00, AH 1/2
  - Tue 18 Sep, 10:00–12:00, AH 2
- **Questions** via mail or Q & A session offered by Christoph

# Forthcoming Teaching Activities

---

## Courses in Winter 2018/19

### *Modelling and Verification of Probabilistic Systems [Katoen; V3 Ü2]*

1. Formal models for complex systems that involve random aspects (Markov chains and decision processes)
2. Specification of (quantitative) system properties (probabilistic variants of temporal logics)
3. Algorithmic verification of (reachability) properties (probabilistic model checking)
4. Optimisation by model reduction (bisimulation, compositional modelling and minimisation)

### *Compiler Construction [Noll; V3 Ü2]*

1. Lexical analysis of programs (Scanner)
2. Syntactic analysis of programs (Parser)
3. Semantic analysis of programs
4. Code generation
5. Tools for compiler construction

## Seminar in Winter 2018/19

### *Static Methods for Quantitative Program Analysis [tentative]*

- Worst-case execution time analysis
  - goal: determine bounds on execution times of a task when executed on a particular hardware
  - mostly based on abstract interpretation
  - problems: data dependence of control flow and context dependence of execution times (speculative execution, cache behaviour, ...)
- Heap resource analysis
  - proving termination of data-structure operations
  - ensuring balancedness of data structures
  - permission accounting for concurrency
- Probabilistic programs
  - programs operating on uncertain data and/or using probabilistic programming constructs
  - compute (interval bounds on) probabilities of assertions over program variables or termination
  - adapt abstract interpretation and slicing techniques to probabilistic setting

## Questions

1. **Relation** between Dataflow Analysis and Abstract Interpretation
  - abstract semantics (Definition 12.10) vs. fixpoint/MOP solution
2. Handling of **worklist** in fixpoint computation
  - Algorithm 4.9: as stack
3. **Narrowing** (Slide 7.18) for interval analysis
  - termination generally not guaranteed due to existence of infinite descending chains:

$$[0, +\infty] \supsetneq [1, +\infty] \supsetneq [2, +\infty] \supsetneq \dots$$

# Abstract Semantics of WHILE using Extraction Functions

## Abstract Semantics of WHILE I

**Reminder:** abstract domain is  $Abs := 2^{Var \rightarrow A}$

### Definition 12.10 (Abstract execution relation for statements)

If  $c \in Cmd$  and  $abs \in Abs$ , then  $\langle c, abs \rangle$  is called an **abstract configuration**. The **abstract execution relation** is defined by the following rules:

$$\frac{}{\text{(skip)} \quad \langle \text{skip}, abs \rangle \Rightarrow \langle \downarrow, abs \rangle}$$

$$\frac{}{\text{(asgn)} \quad \langle x := a, abs \rangle \Rightarrow \langle \downarrow, \{ \rho[x \mapsto a'] \mid \rho \in abs, a' \in val_{\rho}^{\#}(a) \} \rangle}$$

$$\frac{\langle c_1, abs \rangle \Rightarrow \langle c'_1, abs' \rangle \quad c'_1 \neq \downarrow}{\text{(seq1)} \quad \langle c_1 ; c_2, abs \rangle \Rightarrow \langle c'_1 ; c_2, abs' \rangle}$$

$$\frac{\langle c_1, abs \rangle \Rightarrow \langle \downarrow, abs' \rangle}{\text{(seq2)} \quad \langle c_1 ; c_2, abs \rangle \Rightarrow \langle c_2, abs' \rangle}$$



# Efficient Fixpoint Computation

## A Worklist Algorithm I

**Observation:** fixpoint iteration re-computes every  $AI_I$  in every step

⇒ **redundant** if  $AI_{I'}$  at no  $F$ -predecessor  $I'$  changed

⇒ optimisation by **worklist** over control-flow edges

### Algorithm 4.9 (Worklist algorithm)

```
Input: dataflow system  $S = (Lab, E, F, (D, \sqsubseteq), \iota, \varphi)$   
Variables:  $W \in (Lab \times Lab)^*$ ,  $\{AI_I \in D \mid I \in Lab\}$   
Procedure:  $W := \varepsilon$ ; for  $(I, I') \in F$  do  $W := W \cdot (I, I')$ ;           % Initialise  $W$   
  for  $I \in Lab$  do  
    if  $I \in E$  then  $AI_I := \iota$  else  $AI_I := \perp_D$ ;           % Initialise  $AI$   
  while  $W \neq \varepsilon$  do  
     $(I, I') := \text{head}(W)$ ;  $W := \text{tail}(W)$ ;           % Next control-flow edge  
    if  $\varphi_I(AI_I) \not\sqsubseteq AI_{I'}$  then           % Fixpoint not yet reached  
       $AI_{I'} := AI_{I'} \sqcup \varphi_I(AI_I)$ ;           % Update analysis information  
      for  $(I', I'') \in F$  do  
        if  $(I', I'')$  not in  $W$  then  $W := (I', I'') \cdot W$ ; % Propagate modification  
Output:  $\{AI_I \mid I \in Lab\}$ 
```

## Idea of Narrowing

- **Observation:** widening can “shoot above the target”, i.e., lead to **unnecessarily imprecise results**
- **Solution:** improvement by **iterating again** from the result obtained by widening (i.e., from  $\text{fix}^\nabla(\Phi_S)$ )

$\implies$  compute  $\Phi_S^k(\text{fix}^\nabla(\Phi_S))$  for  $k = 1, 2, \dots$

- **Soundness:**  $\text{fix}^\nabla(\Phi_S) \sqsupseteq \text{fix}(\Phi_S)$  (cf. Algorithm 7.3)

$\implies \Phi_S^k(\text{fix}^\nabla(\Phi_S)) \sqsupseteq \Phi_S^k(\text{fix}(\Phi_S)) = \text{fix}(\Phi_S)$

(since  $\Phi_S$  and thus  $\Phi_S^k$  monotonic)