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/`

Software Modeling and Verification Chair
RWTH AACHEN UNIVERSITY

Outline of Lecture 21

Dedicated Algorithms for Pointer Analysis

- `null` Pointer Analysis: checks whether dereferencing operations possibly involve `null` pointers
  - with shape analysis: $x = \texttt{nil}$ possible for $x \in \mathit{Var}$ at $l \in \mathit{Lab}$ if there exists heap configuration $H^\# \in HC^\#$ at l which does not contain x
  - simple algorithms based on types (usually flow-insensitive)
  - more advanced approaches employ constraint-based abstract interpretation[1]

[1] https://doi.org/10.1007/978-3-540-68863-1_9
[2] https://doi.org/10.1145/237721.237727

Dedicated Algorithms for Pointer Analysis

- `null` Pointer Analysis: checks whether dereferencing operations possibly involve `null` pointers
  - with shape analysis: $x = \texttt{nil}$ possible for $x \in \mathit{Var}$ at $l \in \mathit{Lab}$ if there exists heap configuration $H^\# \in HC^\#$ at l which does not contain x
  - simple algorithms based on types (usually flow-insensitive)
  - more advanced approaches employ constraint-based abstract interpretation[1]
- Points-To Analysis: yields function pt that for each $x \in \mathit{Var}$ 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 $H^\# \in HC^\#$ with a node that is referred to by both x and y
  - algorithms with linear complexity based on type inference and constraint solving[2]

[1] https://doi.org/10.1007/978-3-540-68863-1_9
[2] https://doi.org/10.1145/237721.237727

Dedicated Algorithms for Pointer Analysis

- `null` Pointer Analysis: checks whether dereferencing operations possibly involve `null` pointers
  - with shape analysis: $x = \texttt{nil}$ possible for $x \in \mathit{Var}$ at $l \in \mathit{Lab}$ if there exists heap configuration $H^\# \in HC^\#$ at l which does not contain x
  - simple algorithms based on types (usually flow-insensitive)
  - more advanced approaches employ constraint-based abstract interpretation[1]
- Points-To Analysis: yields function pt that for each $x \in \mathit{Var}$ 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 $H^\# \in HC^\#$ with a node that is referred to by both x and y
  - algorithms with linear complexity based on type inference and constraint solving[2]
- 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)

[1] https://doi.org/10.1007/978-3-540-68863-1_9
[2] https://doi.org/10.1145/237721.237727

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)

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!

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

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

Outline of Lecture 21

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

Outline of Lecture 21

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)

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

Forthcoming Teaching Activities

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, ...)

Forthcoming Teaching Activities

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

Forthcoming Teaching Activities

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

Outline of Lecture 21

Q & A Session

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 I

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

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:

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

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

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

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

A Worklist Algorithm I

Observation: fixpoint iteration re-computes every AI_l in every step

$\Longrightarrow$ redundant if $\mathsf{AI}_{l'}$ at no F-predecessor l' changed

$\Longrightarrow$ optimisation by worklist over control-flow edges

Algorithm 4.9 (Worklist algorithm)

Input: *dataflow system* $S = (Lab, E, F, (D, \sqsubseteq), \iota, \varphi)$

Variables: $W \in (Lab \times Lab)^*$, $\{\mathsf{AI}_l \in D \mid l \in Lab\}$

Procedure:

$W := \varepsilon$; **for** $(l, l') \in F$ **do** $W := W \cdot (l, l')$; *% Initialise W*

  for $l \in Lab$ **do**

    if $l \in E$ **then** $\mathsf{AI}_l := \iota$ **else** $\mathsf{AI}_l := \bot_D$; *% Initialise* AI

  while $W \neq \varepsilon$ **do**

    $(l, l') := \mathbf{head}(W)$; $W := \mathbf{tail}(W)$; *% Next control-flow edge*

    if $\varphi_l(\mathsf{AI}_l) \not\sqsubseteq \mathsf{AI}_{l'}$ **then** *% Fixpoint not yet reached*

      $\mathsf{AI}_{l'} := \mathsf{AI}_{l'} \sqcup \varphi_l(\mathsf{AI}_l)$; *% Update analysis information*

      for $(l', l'') \in F$ **do**

        if (l', l'') *not in* W **then** $W := (l', l'') \cdot W$; *% Propagate modification*

Output: $\{\mathsf{AI}_l \mid l \in Lab\}$

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 $\text{fix}^{\nabla}(\Phi_S)$)

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

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

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

(since Φ_S and thus Φ_S^k monotonic)