Static Program Analysis

Lecture 21: Extensions III (Pointer Analysis & Wrap-Up)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH Aachen University

noll@cs.rwth-aachen.de

http://www-i2.informatik.rwth-aachen.de/i2/spa11/

Summer Semester 2011

Outline

Pointer Analysis

- **So far:** only static data structures (variables)
- **Now:** pointer (variables) and dynamic memory allocation using heaps
- **Problem:**
  - Programs with pointers and dynamically allocated data structures are error prone
  - Identify subtle bugs at compile time
  - Automatically prove correctness
- **Interesting properties of heap-manipulating programs:**
  - No null pointer dereference
  - No memory leaks
  - Preservation of data structures
  - Partial/total correctness

The Shape Analysis Approach

- **Goal:** determine the possible shapes of a dynamically allocated data structure at given program point
- **Interesting information:**
  - data types (to avoid type errors, such as dereferencing `nil`)
  - sharing (different pointer variables referencing same address; aliasing)
  - reachability of nodes (garbage collection)
  - disjointness of heap regions (parallelizability)
  - shapes (lists, trees, absence of cycles, ...)
- **Concrete questions:**
  - Does `x.next` point to a shared element?
  - Does a variable `p` point to an allocated element every time `p` is dereferenced?
  - Does a variable point to an acyclic list?
  - Does a variable point to a doubly-linked list?
  - Can a loop or procedure cause a memory leak?
- **Here:** basic outline; details in [Nielson/Nielson/Hankin 2005, Sct. 2.6]

Outline

Extending the Syntax

Syntactic categories:

Category	Domain	Meta variable
Arithmetic expressions	*AExp*	a
Boolean expressions	*BExp*	b
Selector names	*Sel*	*sel*
Pointer expressions	*PExp*	p
Commands (statements)	*Cmd*	c

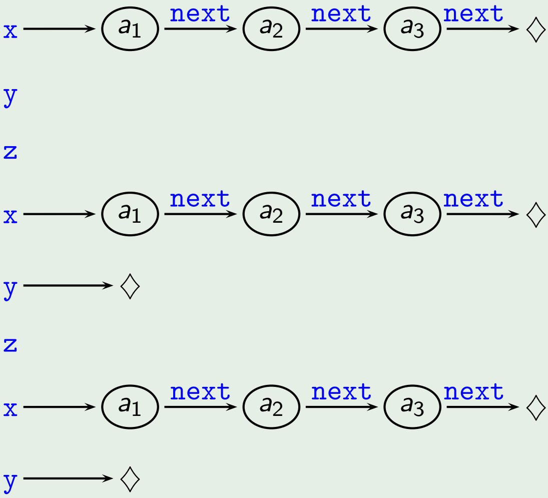
Context-free grammar:

$$a ::= z \mid x \mid a_1 + a_2 \mid \ldots \mid p \mid \texttt{nil} \in AExp$$
$$b ::= t \mid a_1 = a_2 \mid b_1 \wedge b_2 \mid \ldots \mid \texttt{is-nil}(p) \in BExp$$
$$p ::= x \mid x.sel$$
$$c ::= [\texttt{skip}]^l \mid [p := a]^l \mid c_1 ; c_2 \mid \texttt{if } [b]^l \texttt{ then } c_1 \texttt{ else } c_2 \mid \texttt{while } [b]^l \texttt{ do } c \mid [\texttt{malloc } p]^l \in Cmd$$

An Example

Example 21.1 (List reversal)

Program that reverses list pointed to by x and leaves result in y:



Outline

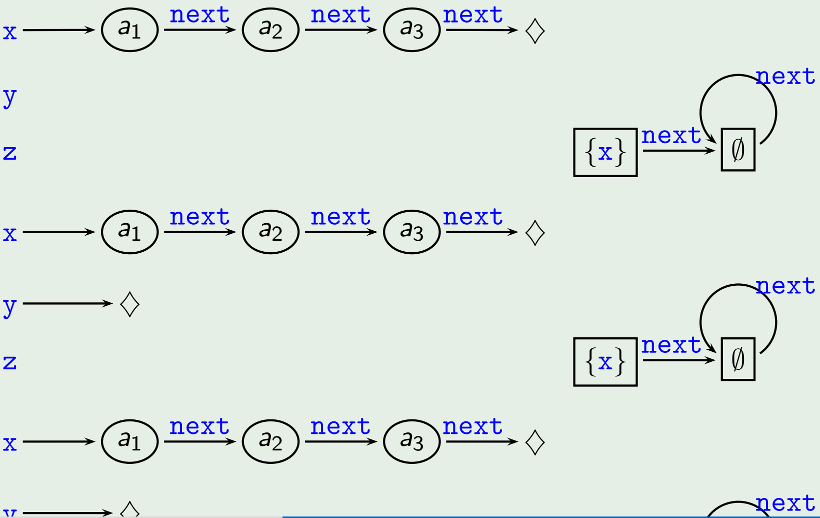
Shape Graphs I

Approach: representation of (infinitely many) concrete heap states by (finitely many) abstract shape graphs

- abstract nodes X = sets of variables (interpretation: $x \in X$ iff x points to concrete node represented by X)
- $\emptyset$ represents all concrete nodes that are not directly reachable
- if $x.sel$ and y refer to the same heap address and if X, Y are abstract nodes with $x \in X$ and $y \in Y$, then this yields abstract edge $X \xrightarrow{sel} Y$
- transfer functions transform (sets of) shape graphs

Example 21.2 (List reversal (cf. Example 21.1))

Concrete heap **Shape graph**



Shape Graphs III

Definition 21.3 (Shape graph)

A shape graph $G = (S, H)$ consists of

- a set $S \subseteq 2^{Var}$ of abstract locations and
- an abstract heap $H \subseteq S \times Sel \times S$
  (notation: $X \xrightarrow{sel} Y$ for $(X, sel, Y) \in H$).

with the following properties:

Disjointness: $X, Y \in S \implies X = Y$ or $X \cap Y = \emptyset$
(a variable can refer to at most one heap location)

Determinacy: $X \xrightarrow{sel} Y$ and $X \xrightarrow{sel} Z \implies X = \emptyset$ or $Y = Z$
(target location is unique if source location non-empty)

SG denotes the set of all shape graphs.

Remark: the following example shows that determinacy requires $X \neq \emptyset$:

Concrete: $y \longrightarrow \bullet \xleftarrow{sel} \bullet$

$z \longrightarrow \bullet \xleftarrow{sel} \bullet$

Abstract: $\boxed{Y = \{y\}} \xleftarrow{sel} \boxed{X = \emptyset} \xrightarrow{sel} \boxed{Z = \{z\}}$

Shape Graphs and Concrete Heap Properties

Let $G = (S, H)$ be a shape graph.

- $\texttt{x} \neq \texttt{nil}$
  $\iff \exists X \in S : \texttt{x} \in X$
- $\texttt{x} = \texttt{y} \neq \texttt{nil}$ (aliasing)
  $\iff \exists X \in S : \texttt{x}, \texttt{y} \in X$
- $\texttt{x.sel} = \texttt{y} \neq \texttt{nil}$ (sharing)
  $\iff \exists X, Y \in S : \texttt{x} \in X, \texttt{y} \in Y, X \xrightarrow{\texttt{sel}} Y$

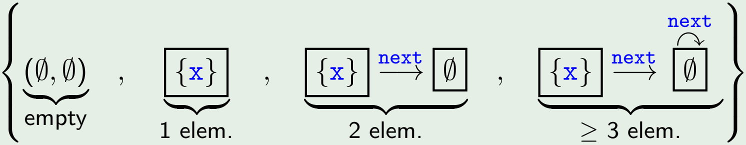
Outline

The Domain

- **Approach:** forward analysis to determine all shape graphs that represent all possible heap structures at the respective label
- Domain: $(D, \sqsubseteq) := (2^{SG}, \subseteq)$
  - Var, Sel finite $\implies SG$ finite $\implies 2^{SG}$ finite $\implies$ ACC
- Extremal value: $\iota :=$ {shape graphs for possible initial values of Var}

Example 21.4 (List reversal (cf. Example 21.2))

- Variables: $Var = \{\mathtt{x}, \mathtt{y}, \mathtt{z}\}$
- Assumption: $\mathtt{x}$ points to any (finite, non-cyclic) list, $\mathtt{y} = \mathtt{z} = \mathtt{nil}$

$\implies \iota =$ 

The Transfer Functions

Transfer functions: $\varphi_l : 2^{SG} \to 2^{SG}$ (monotonic)

- Transform each single shape graph into a set of shape graphs: $\varphi_l(\{G_1, \ldots, G_n\}) = \bigcup_{i=1}^{n} \varphi_l(G_i)$
- $\varphi_l(G)$ determined by B^l:
  - $[\texttt{skip}]^l$: $\varphi_l(G) := \{G\}$
  - $[b]^l$: $\varphi_l(G) := \{G\}$
  - $[p \texttt{ := } a]^l$: case-by-case analysis w.r.t. p and a
    - [Nielson/Nielson/Hankin 2005]: 12 cases
    - may involve (high degree of) non-determinism
  - $[\texttt{malloc } x]^l$: $\varphi_l(G) := \{(S' \cup \{\{x\}\}, H')\}$ where
    - $G = (S, H)$
    - $S' := \{X \setminus \{x\} \mid X \in S\}$
    - $H' := H \cap S' \times Sel \times S'$
  - $[\texttt{malloc } x.sel]^l$: equivalent to

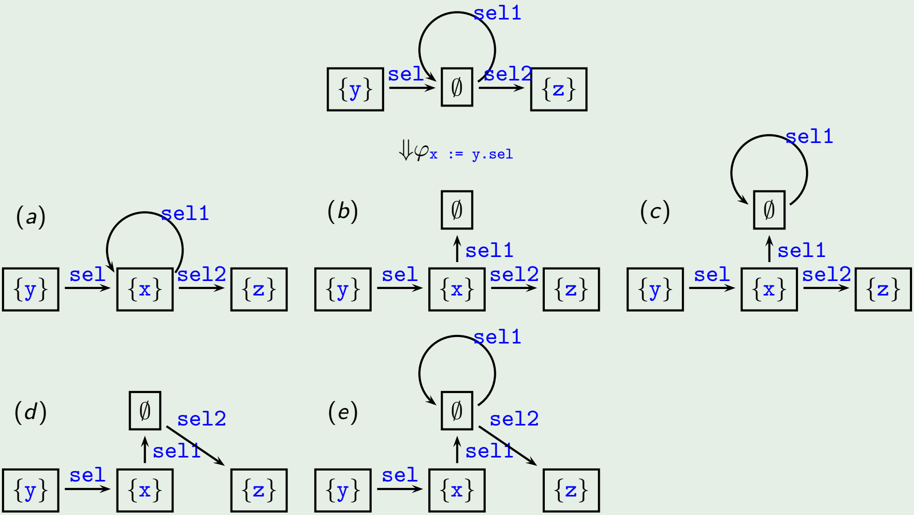
    $[\texttt{malloc } t]^{l_1}\texttt{;}[x\texttt{.}sel \texttt{ := } t]^{l_2}\texttt{;}[t \texttt{ := nil}]^{l_3}\texttt{;}$

    with fresh $t \in Var$ and $l_1, l_2, t_3 \in L$
- Crucial for soundness: safety of approximation

  if shape graph G approximates heap h and $h \stackrel{[p \texttt{ := } a]^l}{\longrightarrow} h'$,
  then there exists $G' \in \varphi_l(G)$ such that G' approximates h'

An Example

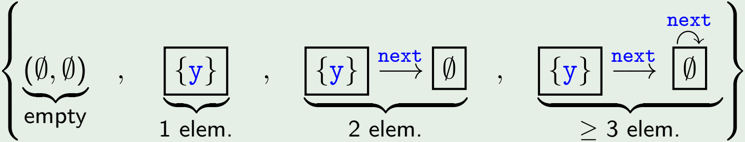
Example 21.5



Application to List Reversal

Example 21.6 (List reversal (cf. Example 21.2))

Shape analysis of list reversal program yields final result



Interpretation:

+ Result again finite list
- but potentially cyclic (a “lasso”, not a ring)
- also “reversal” property not guaranteed

Outline

Alternative Approaches to Pointer Analysis

- **Dedicated algorithms:**
  - `nil` Pointer Analysis: checks whether dereferencing operations possibly involve `nil` pointers
  - Points-To Analysis: yields function pt that for each $x \in Var$ returns set $pt(x)$ of possible pointer targets
    - x and y may be aliases if $pt(x) \cap pt(y) \neq \emptyset$

  Usually faster and sometimes more precise, but less general (only "shallow" properties)
- **Graph grammar approaches:**
  - e.g., J. Heinen, T. Noll, S. Rieger: *Juggrnaut: Graph Grammar Abstraction for Unbounded Heap Structures* TTSS 2009, ENTCS 266, Elsevier, 2010, 93–107
  - idea: specify data structures by graph production rules
  - concretization by forward application
  - abstraction by backward application
  - all pointer operations remain concrete

  $\Longrightarrow$ avoids complicated definition of transfer functions

Correctness of Analyses

- **So far:** semantics and dataflow analysis of programs independent
- Of course both are (and should be) related!
- To this aim: compare results of concrete semantics (Definition 12.9) with outcome of analysis
- **Example:** correctness of Constant Propagation

  Let $c \in Cmd$ with $l_0 = \text{init}(c)$, and let $l \in L_c$, $x \in Var$, and $z \in \mathbb{Z}$ such that $\text{CP}_l(x) = z$. Then for every $\sigma_0, \sigma \in \Sigma$ such that $\langle l_0, \sigma_0 \rangle \to^* \langle l, \sigma \rangle$, $\sigma(x) = z$.
- see [Nielson/Nielson/Hankin 2005, Sct. 2.2]

Outline

Final Remarks

Fortcoming courses in Wintersemester 2011/12

- *Introduction to Model Checking* [Katoen; V3/4 Ü2]
  1. Transition systems
  2. Property classes: safety, liveness, invariants, fairness
  3. Linear Temporal Logic (LTL)
  4. Computational Tree Logic (CTL)
  5. Model Checking algorithms for LTL and CTL
- *Semantics and Verification of Software* [Noll; V3 Ü2]
  1. The imperative model language WHILE
  2. Operational, denotational and axiomatic semantics of WHILE
  3. Equivalence of operational and denotational semantics
  4. Applications: compiler correctness, optimizing transformations
  5. Extensions: procedures and dynamic data structures