Program Analyses for Memory

Oukseh Lee
Seoul National University
oukseh@ropas.snu.ac.kr

February 13, 2004
LiComR Winter School 2004

My Current State

- Developed an algorithm to replace allocations by memory reuse into ML-like programs.
  - Space and "runtime" improvement are satisfactory.
  - Not suitable for imperative languages.
  - Our improvement in ML is not expected in Java.
- Interested in automatic loop invariant inference for the separation logic.
  - Precise heap analyzer is necessary even with destructive update.
  - Shape analysis [SaReWi96,97,99,02] is precise but weak for finding alias relation and expensive.

Program Analyses for Memory

- May-alias analysis & points-to analysis.
- Shape analysis.
- Liveness analysis & escape analysis.

- Linear type system for heap values.
- Region-based type system.
- Alias type system.

Goals

- Program correctness such as
  - no null/dangling-pointer access, and
  - resource invariant preservation.
- Performance improvement by
  - early deallocation and lazy allocation,
  - less allocation, and
  - locality improvement.
- Help to other program analyses.

Why Difficult?

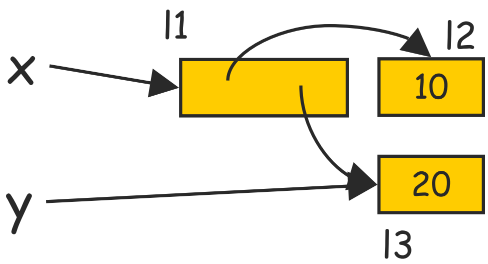
- The number of heap cells is possibly infinite.
- The length of recursive data structure is possibly infinite.
- Data structures can be shared irregularly.
- Destructive update usually requires high precision of analyses.
- Pointer arithmetic induces unexpected behaviors.

Contents

- Semantics-based program analyses for memory
  - Store-based model
    - shape analysis.
  - Storeless model
    - alias analysis, sharing analysis, escape analysis.

Store-Based Model

- The standard semantics for heap relies on environments, memory locations, and stores.



env $\left\{ \begin{array}{l} \text{x} \mapsto \text{l1} \\ \text{y} \mapsto \text{l3} \end{array} \right\}$

store $\left\{ \begin{array}{l} \text{l1} \mapsto \text{l2, l3} \\ \text{l2} \mapsto 10 \\ \text{l3} \mapsto 20 \end{array} \right\}$

Semantics

$$
\begin{array}{rcrcl}
\rho & \in & \mathit{Env} & = & \mathit{Var} \rightarrow \mathit{Loc} \\
\sigma & \in & \mathit{Store} & = & \mathit{Loc} \rightarrow \mathit{Value} \\
 & & \mathit{Value} & = & \mathit{Int} \cup (\mathit{Field} \rightarrow \mathit{Loc})
\end{array}
$$

$$
\begin{array}{rcl}
[\![i]\!]\,\rho\,\sigma & = & (l,\ \sigma\,\{i/l\})\ \text{new}\ l \\
[\![x]\!]\,\rho\,\sigma & = & (\rho(x),\ \sigma) \\
[\![(x_1, \cdots, x_n)]\!]\,\rho\,\sigma & = & (l,\ \sigma\,\{\{i \mapsto \rho(x_i)\}\,/l\})\ \text{new}\ l \\
[\![x.i]\!]\,\rho\,\sigma & = & (\sigma(\rho(x))(i),\ \sigma) \\
[\![\texttt{let}\ x\texttt{=}e_1\ \texttt{in}\ e_2]\!]\,\rho\,\sigma & = & \text{let}\ (l, \sigma') = [\![e_1]\!]\,\rho\,\sigma \\
 & & \text{in}\ \ [\![e_2]\!]\,(\rho\,\{l/x\})\,\sigma'
\end{array}
$$

Abstract Semantics

- Collecting semantics:

$$Lab \to \mathcal{P}(Env \times Store)$$
$$Lab \to \mathcal{P}(Loc \times Store)$$

- Component-wise abstraction [Deu90]:

$$\begin{array}{lcl} Env^{\#} & = & Var^{\#} \to \mathcal{P}(Loc^{\#}) \\ Store^{\#} & = & Loc^{\#} \to Value^{\#} \\ Value^{\#} & = & \mathcal{P}(Int)^{\#} \times (Field^{\#} \to \mathcal{P}(Loc^{\#})) \end{array}$$

  - How to abstract location?

Abstraction of Location

- Location = State in its allocation [Deu90]

$$\alpha_{Loc} = \alpha_{\times}(\alpha_{Lab}, \alpha'_{Env}, \alpha'_{Store})$$

- For instance,
  - allocation site [JoMu82,RuMu88,Mo87,ChWeZa90]
  - context-sensitive analysis: additional continuation (or call sequence) in the state [Deu90]

Abstraction by Allocation Site

```
fun gen n =
  if n=0 then []
  else n::gen(n-1) L
val x = gen 5
val y = gen 6
```

x% # y% ?
x% acyclic?

NO

$$\left\{\begin{array}{l} \mathbf{x} \mapsto \{L\} \\ \mathbf{y} \mapsto \{L\} \end{array}\right\} \qquad \left\{ L \mapsto \{L\} \right\}$$

Abstraction by Allocation Site and k-Length Call String

```
fun gen n =
  if n=0 then []
  else n::gen(n-1) L M
val x = gen 5 N
val y = gen 6 O
```

x% # y% ?
x% acyclic?

NO

length 2; recent call-sites

$$\left\{\begin{array}{l} \mathbf{x} \mapsto \{NL\} \\ \mathbf{y} \mapsto \{OL\} \end{array}\right\} \qquad \left\{\begin{array}{l} NL \mapsto \{NML\} \\ NML \mapsto \{MML\} \\ MML \mapsto \{MML\} \\ OL \mapsto \{OML\} \\ OML \mapsto \{MML\} \end{array}\right\}$$

Abstraction by Allocation Site and k-Length Call String

```
fun gen n =
  if n=0 then []
  else n::gen(n-1)
val x = gen 5
val y = gen 6
```

L M N O

x% # y% ? YES

x% acyclic? NO

length 1, first call-site

$$\begin{Bmatrix} \mathbf{x} \mapsto \{\mathrm{NL}\} \\ \mathbf{y} \mapsto \{\mathrm{OL}\} \end{Bmatrix}$$

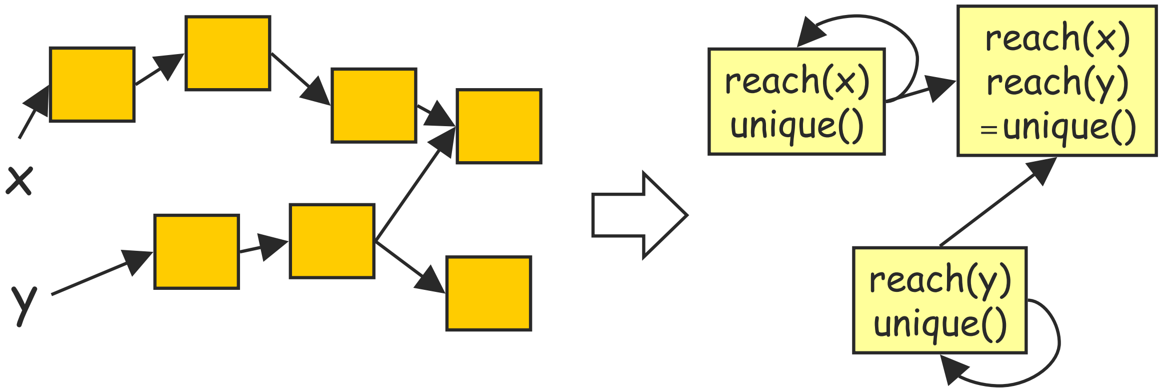
$$\begin{Bmatrix} \mathrm{NL} \mapsto \{\mathrm{NL}\} \\ \mathrm{OL} \mapsto \{\mathrm{OL}\} \end{Bmatrix}$$

Another Abstraction

- Interesting properties of heap cells are not always related to their locations (=allocation context).
  - pointed-to-by(x).
  - reached-from(x).
  - unique(): one or zero in-edge from the heap.
- Abstraction on (a set of) environment and store pairs:
  - graph-based or property-based abstraction.

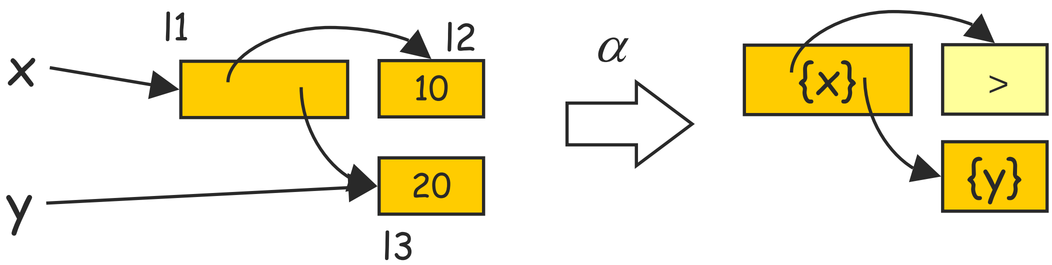
Shape Analysis

- Abstracts a set of locations by their (spatial) properties, instead of allocation context [SaReWi96,97,99,02].



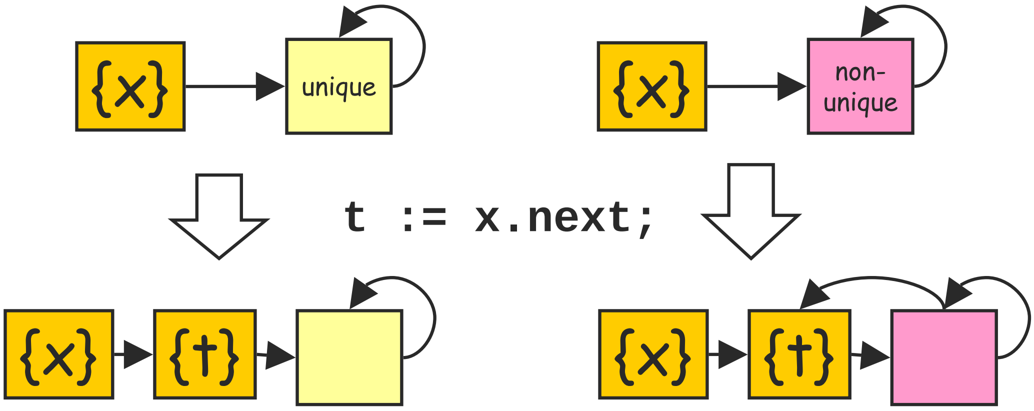
Shape Analysis [SaReWi96,97]

- Group heap cells by a set of variables that points to themselves & uniqueness.



Focus (Materialization)

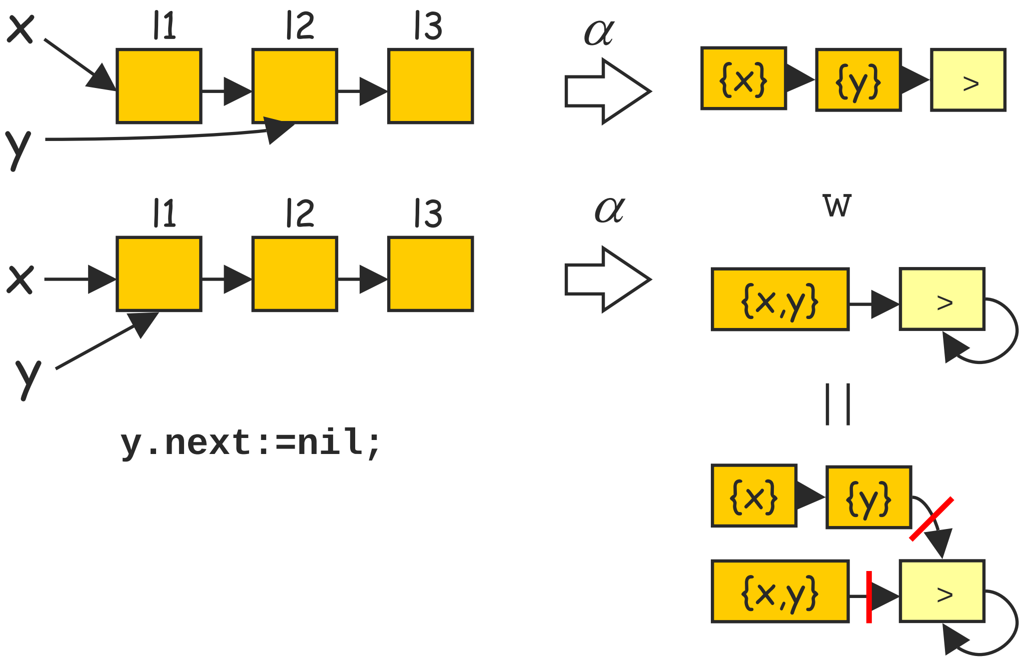
- Concretize shape-graph when necessary.



- Uniqueness information contributes on better materialization.

Strong Update (Nullification)

```
if (…) { y:=x; } else { y:=x.next; }
```

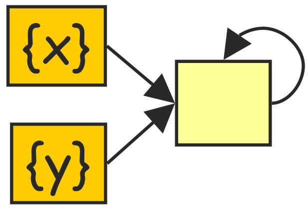


Example: Shape Analysis

```
fun gen n =
 let t = ref [] in
  for i=1 to n do
    t:=i::!t
  end;
  !t
 end
val x = gen 5
val y = gen 6
```

x% # y% ? Yes

x% acyclic? Yes



Summary: Store-Based Model

- A standard model to intuitively reason about the heap.
- Location-based abstraction:
  - it can judge that two heap cells are separated.
  - allocation site and call string do not seem to be sufficient.
- Graph-based abstraction:
  - good to judge the shape of the heap.
  - precise for destructive update.
  - not scalable.

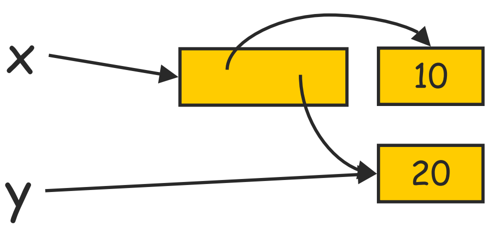
Alias Analyses

- Problem: *x and *y have the same location?
  - It is not necessary to know what locations x and y may have.
  - It is not necessary to know how the heap is structured.

- How about semantics to directly expose the alias relation?

Storeless Model [Jo81,De92,94]

- Access path
  - a sequence of variable names, record field selectors, and so on.
- Alias relations
  - a set of pairs of **aliased** access paths.



$$\text{tree} \left\{ \begin{array}{l} x.1 \mapsto 10 \\ x.2 \mapsto 20 \\ y \mapsto 20 \\ x \mapsto \Omega \end{array} \right\}$$

$$\text{aliases} \; \{ (x.2, y) \}$$

Semantics

$$\begin{array}{rcrcl} t & \in & \mathit{Tree} & = & \Sigma^* \to \mathit{Int} \cup \{\Omega\} \\ \alpha, \equiv & \in & \mathit{Aliases} & = & \mathcal{P}(\Sigma^* \times \Sigma^*) \\ & & \Sigma & = & \mathit{Var} \cup \mathit{Field} \end{array}$$

$$\begin{array}{rcl} \mathsf{copy}(t, s_1, s_2) & = & \{v/s_2.s \,|\, v/s_1.s \in t\} \\ \mathsf{rem}(\alpha, s) & = & \{(s_1, s_2) \in \alpha \,|\, s_1, s_2 \notin \{s'.s'' \,|\, (s, s') \in \alpha, |s''| > 0\}\} \end{array}$$

$$\begin{array}{rcl} [\![i]\!]\; t\, \alpha & = & (\{i/\iota\}\,,\ \alpha) \\ [\![x]\!]\; t\, \alpha & = & (\mathsf{copy}(t, x, \iota),\ \alpha \cup \{(\iota, x)\}) \\ [\![(x_1, \cdots, x_n)]\!]\; t\, \alpha & = & (\{\Omega/\iota\} \cup \bigcup_i \mathsf{copy}(t, x_i, \iota.i),\ \alpha \cup \{(\iota.i, x_i)\}) \\ [\![x.i]\!]\; t\, \alpha & = & (\mathsf{copy}(t, x.i, \iota),\ \alpha \cup \{(x.i, \iota)\}) \\ [\![\mathtt{let}\ x\mathtt{=}e_1\ \mathtt{in}\ e_2]\!]\; t\, \alpha & = & \mathsf{let}\ \ (t_1, \alpha_1) = [\![e_1]\!]\; t\, \alpha \\ & & \qquad (t_2, \alpha_2) = [\![e_2]\!]\; (t \cup t_1[x/\iota])\; (\alpha_1[x/\iota]) \\ & & \mathsf{in}\ \ (t_2, \mathsf{rem}(\alpha_2, x)) \end{array}$$

Abstract Semantics

- Collecting semantics:

  $\mathit{Lab} \to \mathcal{P}(\mathit{Tree} \times \mathit{Aliases})$

- Instances:
  - may-alias analyses: k-limited approach.

    $\forall s'_1, s'_2. s_1 s'_1 \equiv s_2 s'_2$ where $|s_i| \leq k$
  - Deutsch's may-alias analyses [92,94]:

    $x.tl^i.hd \equiv x.tl^j.hd.tl^k$ where $j = i + k$ and $k \geq 1$

Example: May-Alias Analysis

```
fun gen n =
  if n=0 then []
  else n::gen(n-1)
val x = gen 5
val y = gen 6
```

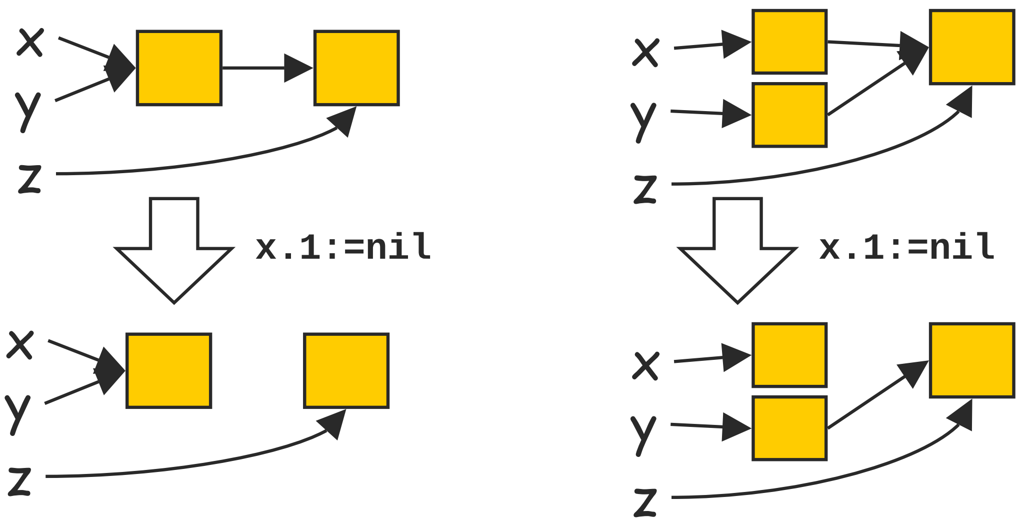
x% # y% ?
x% acyclic?

YES

>

Destructive Update

$\{(x,y), (x.1,z), (y.1,z)\} \sqsupseteq \{(x.1,y.1), (x.1,z), (y.1,z)\}$



$\{(x,y)\} \not\sqsupseteq \{(y.1,z)\}$

Instance: Escape Analysis

- Compute the alias relation between (free) variables and the result value.

```
let x = (1, 2, (4, 5)) in
let y = (7, x) in
y.2.3
```

$$\left\{ \begin{array}{c} (y.2, x) \\ (y.2.3, \iota) \\ (x.3, \iota) \end{array} \right\}$$

$$\left\{ \begin{array}{l} x \mapsto \{3\} \\ y \mapsto \{2.3\} \end{array} \right\}$$

Blanchet's Escape Analysis

- More abstraction on access paths by using type information.

```
let x = (1, 2, (4, 5)) in
let y = (7, x) in
y.2.3
```

$$\left\{ \begin{array}{l} x \mapsto 1 \\ y \mapsto 2 \end{array} \right\}$$

- Experiments
  - analysis time: 5~37% of compile time.
  - heap size decrease: 0~99%
  - runtime: -9%~23%

Summary: Storeless Model

- An alternative to reason about the heap.
- A model to directly expose the alias relation.
  - Precise may-alias analysis.
  - Cost-effective Blanchet's escape analysis.
- Compositional.

- Difficult to precisely handle destructive update.

Summary & Discussion

- Two semantics for memory.
  - Store-based (graph-based) model
    - good for the heap shape.
    - possible to precisely handle destructive update.
    - shape analysis.
  - Storeless model
    - good for precise alias relation.
    - difficult to precisely handle destructive update.
    - may-alias analysis, escape analysis.

References (1/2)

- Alain Deutsch
  - On determining lifetime and aliasing of dynamically allocated data in higher-order functional specification, POPL 1990
  - A storeless model of aliasing and its abstractions using finite representation of right-regular equivalence relations, IEEE ICCL 1992
  - Interprocedural may-alias analysis for pointer: beyond k-limiting, PLDI 1994
  - Semantic models and abstract interpretation techniques for inductive data structures and pointers, PEPM 1995

References (2/2)

- Mooly Sagiv, Thomas Reps, Reinhard Wilhelm
  - Parametric shape analysis via 3-valued logic, POPL 1999, TOPLAS 2002
  - Solving shape-analysis problems in languages with destructive updating, POPL 1996, TOPLAS 1997
- Bruno Blanchet
  - Escape analysis for Java(TM): theory and practice. TOPLAS 2003.
  - Escape analysis for object oriented languages: application to Java(TM), OOPSLA 1999
  - Escape analysis: correctness proof, implementation and experimental results, POPL 1998