Constraint based alias analysis for Java

Thomas Jensen

SOS, Master Recherche Science Informatique, U. Rennes 1

2021-2022

Thomas Jensen

(slides by David Pichardie, Thomas Jensen)

Outline

Data flow equations of reachable definitions

Domain where the solution lives : $RD_{\text{in}}(l), RD_{\text{out}}(l) \in \wp(Var \times \mathbf{Lab}^?)$

$$
\begin{array}{lcl}
kill([\mathtt{x} := a]^l) & = & \{(\mathtt{x}, l') \mid l' \in \mathbf{Lab}^?\} \\
kill([\mathtt{skip}]^l) & = & \emptyset \\
kill([b]^l) & = & \emptyset
\end{array}
$$

$$
\begin{array}{lcl}
gen([\mathtt{x} := a]^l) & = & \{(x, l)\} \\
gen([\mathtt{skip}]^l) & = & \emptyset \\
gen([b]^l) & = & \emptyset
\end{array}
$$

For all $[b]^l \in P$,

$$
\begin{array}{lcl}
RD_{\text{in}}(l) & = & \begin{cases} \{(\mathtt{x}, ?) \mid \mathtt{x} \in Var\} & \text{if } l = init(P) \\ \bigcup_{(l',l) \in flow(P)} RD_{\text{out}}(l') \end{cases} \\
RD_{\text{out}}(l) & = & RD_{\text{in}}(l) \setminus kill([i]^l) \cup gen([i]^l)
\end{array}
$$

A constraint based specification

For all $[b]^l \in P$,

$$\begin{array}{rcl} RD_{\text{in}}(init(P)) & \supseteq & \{(\mathtt{x}, ?) \mid \mathtt{x} \in Var\} \\ RD_{\text{in}}(l) & \supseteq & \bigcup_{(l',l) \in flow(P)} RD_{\text{out}}(l') \text{ if } l \neq init(P) \\ RD_{\text{out}}(l) & \supseteq & RD_{\text{in}}(l) \setminus kill([i]^l) \cup gen([i]^l) \end{array}$$

or

$$\begin{array}{rcl} RD_{\text{in}}(init(P)) & \supseteq & \{(\mathtt{x}, ?) \mid \mathtt{x} \in Var\} \\ RD_{\text{in}}(l) & \supseteq & RD_{\text{out}}(l') \qquad \forall (l',l) \in flow(P),\ l \neq init(P) \\ RD_{\text{out}}(l) & \supseteq & RD_{\text{in}}(l) \setminus kill([i]^l) \cup gen([i]^l) \end{array}$$

This is still a *dataflow* style, based on a control flow graph.

Constraint systems

Analyses are sometimes specified as inequation (constraint) systems :

$$\begin{cases} x_1 & \sqsupseteq & f_1(x_1, \ldots, x_n) \\ & \vdots & \\ x_n & \sqsupseteq & f_n(x_1, \ldots, x_n) \end{cases}$$

Knaster-Tarski : For a complete lattice $(A, \sqsubseteq, \bigsqcup)$ and a monotone function f

$$\mathbf{lfp}(f) = \bigsqcap \{x \in A \mid f(x) \sqsubseteq x\}$$

Corollary : Under the same hypothesis, $\mathbf{lfp}(f) = \mathbf{lpfp}(f)$ where $\mathbf{lpfp}(f)$ is the least post-fixpoint of f.

Conclusion : the two specification styles are equivalent.

A constraint based specification

We will now present a *syntax-directed* specification.

We add a new label $end(P)$ to each program P.

We attach information $RD(l) \in \wp(Var \times \mathbf{Lab}^?)$ to each label of the program.

RD is specified as the least [1] solution of the predicate $RD \vdash P$ defined by

$$\{(\mathbf{x}, ?) \mid \mathbf{x} \in Var\} \subseteq RD(init(P))$$
$$\text{and} \quad RD \vdash (P, end(P))$$

The predicate $RD \vdash (S, l_{\text{next}})$ is defined for $S \in \mathbf{Stm}$ and $l_{\text{next}} \in \mathbf{Lab}$ by induction on the structure of **Stm**.

Intuitively, $RD \vdash (S, l_{\text{next}})$ generate constraints on RD for all labels in statement S, plus the label l_{next} that represents the next label after S.

1. Does it make sense?

A constraint based specification

$$\frac{RD(l) \setminus \{(\mathbf{x}, l') \mid l' \in \mathbf{Lab}^?\} \cup \{(x, l)\} \subseteq RD(l')}{RD \vdash ([x := a]^l, l')}$$

$$\frac{RD(l) \subseteq RD(l')}{RD \vdash ([\,\mathtt{skip}\,]^l, l')} \qquad \frac{RD \vdash (S_1, init(S_2)) \quad RD \vdash (S_2, l')}{RD \vdash (S_1 ; S_2, l')}$$

$$\frac{\begin{array}{ll} RD(l) \subseteq RD(init(S_1)) & RD \vdash (S_1, l') \\ RD(l) \subseteq RD(init(S_2)) & RD \vdash (S_2, l') \end{array}}{RD \vdash (\ \mathtt{if}\ [b]^l\ \mathtt{then}\ S_1\ \mathtt{else}\ S_2, l')}$$

$$\frac{RD(l) \subseteq RD(init(S)) \quad RD \vdash (S, l) \quad RD(l) \subseteq RD(l')}{RD \vdash (\ \mathtt{while}\ [b]^l\ \mathtt{do}\ S, l')}$$

Example

$$P = \texttt{if}\,[x = 0]^1\,\texttt{then}\,[\,\texttt{skip}\,]^2\,\texttt{else}\,\texttt{while}\,[y > 0]^3\,\texttt{do}\,([x := x + 1]^4\,;[y := y - 1]^5)$$

Dataflow specification

$$\begin{array}{ll}
RD_{\text{in}}(1) = \{(x,?),(y,?)\} & RD_{\text{out}}(1) = RD_{\text{in}}(1) \\
RD_{\text{in}}(2) = RD_{\text{out}}(1) & RD_{\text{out}}(2) = RD_{\text{in}}(2) \\
RD_{\text{in}}(3) = RD_{\text{out}}(1) \cup RD_{\text{out}}(5) & RD_{\text{out}}(3) = RD_{\text{in}}(3) \\
RD_{\text{in}}(4) = RD_{\text{out}}(3) & RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(\mathbf{x}, l') \mid l' \in \mathbf{Lab}^?\} \cup \{(x,4)\} \\
RD_{\text{in}}(5) = RD_{\text{out}}(4) & RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(\mathbf{y}, l') \mid l' \in \mathbf{Lab}^?\} \cup \{(y,5)\}
\end{array}$$

Constraint based, syntax-directed specification :
$RD \vdash P$

Example

$P = \texttt{if}\,[x = 0]^1\,\texttt{then}\,[\,\texttt{skip}\,]^2\,\texttt{else}\,\texttt{while}\,[y > 0]^3\,\texttt{do}\,([x := x + 1]^4\,;[y := y - 1]^5)$

Dataflow specification

$$RD_{\text{in}}(1) = \{(x,?),(y,?)\} \qquad RD_{\text{out}}(1) = RD_{\text{in}}(1)$$
$$RD_{\text{in}}(2) = RD_{\text{out}}(1) \qquad RD_{\text{out}}(2) = RD_{\text{in}}(2)$$
$$RD_{\text{in}}(3) = RD_{\text{out}}(1) \cup RD_{\text{out}}(5) \qquad RD_{\text{out}}(3) = RD_{\text{in}}(3)$$
$$RD_{\text{in}}(4) = RD_{\text{out}}(3) \qquad RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(\mathbf{x}, l') \mid l' \in \mathbf{Lab}^?\} \cup \{(x,4)\}$$
$$RD_{\text{in}}(5) = RD_{\text{out}}(4) \qquad RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(\mathbf{y}, l') \mid l' \in \mathbf{Lab}^?\} \cup \{(y,5)\}$$

Constraint based, syntax-directed specification :

$\{(x,?),(y,?)\} \subseteq RD(1)$,

$RD \vdash (\,\texttt{if}\,[x = 0]^1\,\texttt{then}\,[\,\texttt{skip}\,]^2\,\texttt{else}\,\texttt{while}\,[y > 0]^3\,\texttt{do}\,([x := x + 1]^4\,;[y := y - 1]^5), 6)$

Example

$$P = \texttt{if}\,[x = 0]^1\,\texttt{then}\,[\,\texttt{skip}\,]^2\,\texttt{else}\,\texttt{while}\,[y > 0]^3\,\texttt{do}\,([x := x + 1]^4\,;[y := y - 1]^5)$$

Dataflow specification

$$\begin{array}{ll}
RD_{\text{in}}(1) = \{(x,?),(y,?)\} & RD_{\text{out}}(1) = RD_{\text{in}}(1) \\
RD_{\text{in}}(2) = RD_{\text{out}}(1) & RD_{\text{out}}(2) = RD_{\text{in}}(2) \\
RD_{\text{in}}(3) = RD_{\text{out}}(1) \cup RD_{\text{out}}(5) & RD_{\text{out}}(3) = RD_{\text{in}}(3) \\
RD_{\text{in}}(4) = RD_{\text{out}}(3) & RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(\mathbf{x}, l') \mid l' \in \mathbf{Lab}^?\} \cup \{(x,4)\} \\
RD_{\text{in}}(5) = RD_{\text{out}}(4) & RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(\mathbf{y}, l') \mid l' \in \mathbf{Lab}^?\} \cup \{(y,5)\}
\end{array}$$

Constraint based, syntax-directed specification :
$\{(x,?),(y,?)\} \subseteq RD(1)$,
$RD(1) \subseteq RD(2)$, $RD(1) \subseteq RD(3)$,
$RD \vdash ([\,\texttt{skip}\,]^2, 6)$,
$RD \vdash (\,\texttt{while}\,[y > 0]^3\,\texttt{do}\,([x := x + 1]^4\,;[y := y - 1]^5), 6)$

Example

$P = \texttt{if}\,[x = 0]^1\,\texttt{then}\,[\,\texttt{skip}\,]^2\,\texttt{else}\,\texttt{while}\,[y > 0]^3\,\texttt{do}\,([x := x + 1]^4\,;[y := y - 1]^5)$

Dataflow specification

$$RD_{\text{in}}(1) = \{(x,?),(y,?)\} \qquad RD_{\text{out}}(1) = RD_{\text{in}}(1)$$
$$RD_{\text{in}}(2) = RD_{\text{out}}(1) \qquad RD_{\text{out}}(2) = RD_{\text{in}}(2)$$
$$RD_{\text{in}}(3) = RD_{\text{out}}(1) \cup RD_{\text{out}}(5) \qquad RD_{\text{out}}(3) = RD_{\text{in}}(3)$$
$$RD_{\text{in}}(4) = RD_{\text{out}}(3) \qquad RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(\mathbf{x}, l') \mid l' \in \mathbf{Lab}^?\} \cup \{(x,4)\}$$
$$RD_{\text{in}}(5) = RD_{\text{out}}(4) \qquad RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(\mathbf{y}, l') \mid l' \in \mathbf{Lab}^?\} \cup \{(y,5)\}$$

Constraint based, syntax-directed specification :

$\{(x,?),(y,?)\} \subseteq RD(1)$,
$RD(1) \subseteq RD(2)$, $RD(1) \subseteq RD(3)$,
$RD(2) \subseteq RD(6)$,
$RD(3) \subseteq RD(4)$, $RD(3) \subseteq RD(6)$,
$RD \vdash ([x := x + 1]^4\,;[y := y - 1]^5, 3)$

Example

$P = \texttt{if}\,[x = 0]^1\,\texttt{then}\,[\,\texttt{skip}\,]^2\,\texttt{else}\,\texttt{while}\,[y > 0]^3\,\texttt{do}\,([x := x + 1]^4\,;[y := y - 1]^5)$

Dataflow specification

$$RD_{\text{in}}(1) = \{(x,?),(y,?)\} \qquad RD_{\text{out}}(1) = RD_{\text{in}}(1)$$
$$RD_{\text{in}}(2) = RD_{\text{out}}(1) \qquad RD_{\text{out}}(2) = RD_{\text{in}}(2)$$
$$RD_{\text{in}}(3) = RD_{\text{out}}(1) \cup RD_{\text{out}}(5) \qquad RD_{\text{out}}(3) = RD_{\text{in}}(3)$$
$$RD_{\text{in}}(4) = RD_{\text{out}}(3) \qquad RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(\mathbf{x}, l') \mid l' \in \mathbf{Lab}^?\} \cup \{(x,4)\}$$
$$RD_{\text{in}}(5) = RD_{\text{out}}(4) \qquad RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(\mathbf{y}, l') \mid l' \in \mathbf{Lab}^?\} \cup \{(y,5)\}$$

Constraint based, syntax-directed specification :

$\{(x,?),(y,?)\} \subseteq RD(1)$,
$RD(1) \subseteq RD(2)$, $RD(1) \subseteq RD(3)$,
$RD(2) \subseteq RD(6)$,
$RD(3) \subseteq RD(4)$, $RD(3) \subseteq RD(6)$,
$RD \vdash ([x := x + 1]^4, 5)$,
$RD \vdash ([y := y - 1]^5, 3)$

Example

$P = \texttt{if}\,[x = 0]^1\,\texttt{then}\,[\,\texttt{skip}\,]^2\,\texttt{else}\,\texttt{while}\,[y > 0]^3\,\texttt{do}\,([x := x+1]^4\,;[y := y-1]^5)$

Dataflow specification

$$RD_{\text{in}}(1) = \{(x,?),(y,?)\} \qquad RD_{\text{out}}(1) = RD_{\text{in}}(1)$$
$$RD_{\text{in}}(2) = RD_{\text{out}}(1) \qquad RD_{\text{out}}(2) = RD_{\text{in}}(2)$$
$$RD_{\text{in}}(3) = RD_{\text{out}}(1) \cup RD_{\text{out}}(5) \qquad RD_{\text{out}}(3) = RD_{\text{in}}(3)$$
$$RD_{\text{in}}(4) = RD_{\text{out}}(3) \qquad RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(\mathbf{x}, l') \mid l' \in \mathbf{Lab}^?\} \cup \{(x,4)\}$$
$$RD_{\text{in}}(5) = RD_{\text{out}}(4) \qquad RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(\mathbf{y}, l') \mid l' \in \mathbf{Lab}^?\} \cup \{(y,5)\}$$

Constraint based, syntax-directed specification :

$\{(x,?),(y,?)\} \subseteq RD(1)$,
$RD(1) \subseteq RD(2)$, $RD(1) \subseteq RD(3)$,
$RD(2) \subseteq RD(6)$,
$RD(3) \subseteq RD(4)$, $RD(3) \subseteq RD(6)$,
$RD(4) \setminus \{(\mathbf{x}, l') \mid l' \in \mathbf{Lab}^?\} \cup \{(x,4)\} \subseteq RD(5)$,
$RD(5) \setminus \{(\mathbf{y}, l') \mid l' \in \mathbf{Lab}^?\} \cup \{(y,5)\} \subseteq RD(3)$

Example : after simplification

$$P = \texttt{if}\,[x = 0]^1\,\texttt{then}\,[\,\texttt{skip}\,]^2\,\texttt{else}\,\texttt{while}\,[y > 0]^3\,\texttt{do}\,([x := x + 1]^4\,;[y := y - 1]^5)$$

Dataflow specification

$$\begin{array}{lcl}
RD_{\text{in}}(1) & = & \{(x,?),(y,?)\} \\
RD_{\text{in}}(2) & = & RD_{\text{in}}(1) \\
RD_{\text{in}}(3) & = & RD_{\text{in}}(1) \cup \Big(RD_{\text{in}}(5) \setminus \{(\mathbf{y}, l') \mid l' \in \mathbf{Lab}^?\} \cup \{(y,5)\}\Big) \\
RD_{\text{in}}(4) & = & RD_{\text{in}}(3) \\
RD_{\text{in}}(5) & = & RD_{\text{in}}(4) \setminus \{(\mathbf{x}, l') \mid l' \in \mathbf{Lab}^?\} \cup \{(x,4)\}
\end{array}$$

Constraint based, syntax-directed specification :

$$\begin{array}{lcl}
RD(1) & \supseteq & \{(x,?),(y,?)\} \\
RD(2) & \supseteq & RD(1) \\
RD(3) & \supseteq & RD(1) \cup \Big(RD(5) \setminus \{(\mathbf{y}, l') \mid l' \in \mathbf{Lab}^?\} \cup \{(y,5)\}\Big) \\
RD(4) & \supseteq & RD(3) \\
RD(5) & \supseteq & RD(4) \setminus \{(\mathbf{x}, l') \mid l' \in \mathbf{Lab}^?\} \cup \{(x,4)\} \\
RD(6) & \supseteq & RD(2) \cup RD(3)
\end{array}$$

Conclusion : the least solutions coincide.

Why does a least solution exist?

Because the predicate $RD \vdash P$ is equivalent to satisfying a set of constraints

$$F_P(RD) \sqsubseteq RD$$

- in the canonic complete lattice on $\left(\mathbf{Lab} \to \wp(Var \times \mathbf{Lab}^?), \sqsubseteq, \bigsqcup\right)$,
- with F_P a monotone function.

Discussion

Constraint based specifications

- can be more readable than dataflow specifications, due to their *local* nature,
- are often easier to manipulate for soundness proofs,
- generate constraint systems that may solved efficiently by iterative and/or symbolic means.

Outline

Analysing real languages like Java...

... is more difficult that analysing the *While* language!

- The control flow graph is more difficult to build (virtual calls, exceptions, ...).
- Memory updates are not so explicit because of aliasing :
  - example :

    $$y.f := 1 \; ; \; x.f := 0$$

    may end with $y.f = 0$!
- and then there is multi-threading, class initialisers, reflection, ...

The $While_0$ language

We consider a toy object oriented language

Syntax

- Variables ($x, y \in$ **Var**)
- Fields ($f \in$ **Field**)
- Classes ($C \in$ **Class**)
- Commands ($S \in$ **Stm**)

$$S ::= x := y \mid x := y.f \mid x.f := y \mid x := \texttt{new}\ C \mid x := \texttt{null} \mid \texttt{skip} \mid S_1 ; S_2 \mid \texttt{if}\ (*)\ \texttt{then}\ S_1\ \texttt{else}\ S_2 \mid \texttt{while}\ (*)\ \texttt{do}\ S$$

Semantic domains

We assume a countable set **Ref** of references (heap address) and reserve the symbol $\diamond$ for the `null` value.

- Value : $\mathbf{Value} = \mathbf{Ref} + \diamond$
- Local variables : $\rho \in \mathbf{Loc} = \mathbf{Var} \to \mathbf{Value}$
- Heap : $\sigma \in \mathbf{Heap} = \mathbf{Ref} \rightharpoonup (\mathbf{Field} \to \mathbf{Value})$ (partial function)
- State : $(\rho, \sigma) \in \mathbf{Loc} \times \mathbf{Heap}$

We give this language a (non-deterministic) structural operational semantics.

Structural operational semantics (1/2)

$$\frac{}{(x := \texttt{null}, \rho, \sigma) \rightarrow \rho[\quad\quad], \sigma}$$

$$\frac{}{(x := y, \rho, \sigma) \rightarrow \rho[\quad\quad], \sigma}$$

$$\frac{\rho(y) \in \mathbf{dom}(\sigma)}{(x := y.f, \rho, \sigma) \rightarrow \rho[x \mapsto \quad\quad], \sigma}$$

$$\frac{}{(x.f := y, \rho, \sigma) \rightarrow \rho, \sigma[\quad\quad]}$$

$$\frac{}{(x := \texttt{new}\ C, \rho, \sigma) \rightarrow \rho[\quad\quad], \sigma[\quad\quad]}$$

Structural operational semantics (1/2)

$$\frac{}{(x := \mathbf{null}\,, \rho, \sigma) \rightarrow \rho[x \mapsto \diamond], \sigma}$$

$$\frac{}{(x := y, \rho, \sigma) \rightarrow \rho[x \mapsto \rho(y)], \sigma}$$

$$\frac{\rho(y) \in \mathbf{dom}(\sigma)}{(x := y.f, \rho, \sigma) \rightarrow \rho[x \mapsto \sigma(\rho(y))(f)], \sigma}$$

$$\frac{\rho(x) = r \in \mathbf{dom}(\sigma) \qquad o' = \sigma(r)[f \mapsto \rho(y)]}{(x.f := y, \rho, \sigma) \rightarrow \rho, \sigma[r \mapsto o']}$$

$$\frac{r \notin \mathbf{dom}(\sigma)}{(x := \mathbf{new}\ C, \rho, \sigma) \rightarrow \rho[x \mapsto r], \sigma[r \mapsto \lambda f.\, \diamond\,]}$$

Structural operational semantics (2/2)

$$(\ \mathbf{skip}\ , \rho, \sigma) \rightarrow \rho, \sigma$$

$$\frac{(S_1, \rho, \sigma) \rightarrow \rho', \sigma'}{(S_1\ ;\ S_2, \rho, \sigma) \rightarrow (S_2, \rho', \sigma')} \qquad \frac{(S_1, \rho, \sigma) \rightarrow (S_1', \rho', \sigma')}{(S_1\ ;\ S_2, \rho, \sigma) \rightarrow (S_1'\ ;\ S_2, \rho', \sigma')}$$

$$(\ \mathbf{if}\ (*)\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2, \rho, \sigma) \rightarrow (S_1, \rho, \sigma)$$

$$(\ \mathbf{if}\ (*)\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2, \rho, \sigma) \rightarrow (S_2, \rho, \sigma)$$

$$(\ \mathbf{while}\ (*)\ \mathbf{do}\ S, \rho, \sigma) \rightarrow (S\ ;\ \mathbf{while}\ (*)\ \mathbf{do}\ S, \rho, \sigma)$$

$$(\ \mathbf{while}\ (*)\ \mathbf{do}\ S, \rho, \sigma) \rightarrow \rho, \sigma$$

Vocabulary

Heap expressions :

$$e \;:=\; x \mid x.f$$

May-alias :

There exists an execution path to a given program point such that two given memory accesses have the same target.

Must-alias :

For all execution paths to a given program point, two given memory accesses have always the same target.

Dynamic allocation makes things harder

The set of possible heaps after a loop like

$$\texttt{while}\ (*)\ \texttt{do}\ (x := \texttt{new}\ C\ ;\ \ldots)$$

may be unbounded.

We need to have a finite representation of something which is infinite...

Points-to analysis

Points-to analysis :

determine the set of objects whose addresses may be stored in variables or fields of objects.

Basic idea : all reference that are created at the same allocation site are merged into the same equivalence class.

We attach an allocation label $h \in \mathcal{H}$ at each instruction $\texttt{new}\ C$.

$$\texttt{new}^{\,h} C$$

The set of abstract object names is then defined to be $\mathcal{O} = \mathcal{H}$.

Example

```
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
      List temp = new List();
1:    temp.val = new T();
2:    temp.val.f = new A();
3:    temp.next = l;
      l = temp }
    while (*) {
      T t = new T();
4:    t.data = l;
      t.start();
5:    t.f = ...;}
    return;
   }
}

class T extends java.lang.Thread {
  A f;
  List data;
  void run(){
    while(*){
6:    List m = this.data;
7:    while (*) { m = m.next; }
8:    synchronized(m){ m.val.f = ...;}}
    return;}}
```

Example

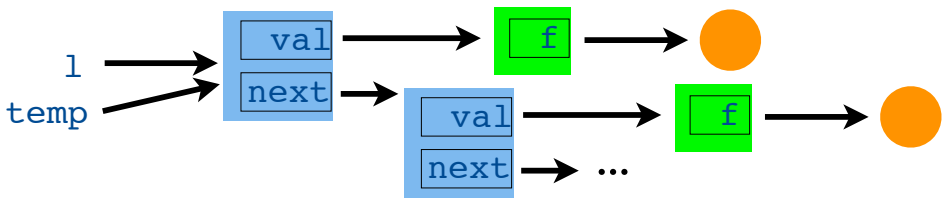
```
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
      List temp = new List();
1:    temp.val = new T();
2:    temp.val.f = new A();
3:    temp.next = l;
      l = temp }
    while (*) {
      T t = new T();
4:    t.data = l;
      t.start();
5:    t.f = ...;}
    return;
   }
}

class T extends java.lang.Thread {
  A f;
  List data;
  void run(){
    while(*){
6:    List m = this.data;
7:    while (*) { m = m.next; }
8:    synchronized(m){ m.val.f = ...;}}
    return;}}
```

1. We create a link list `l`

Threads: M



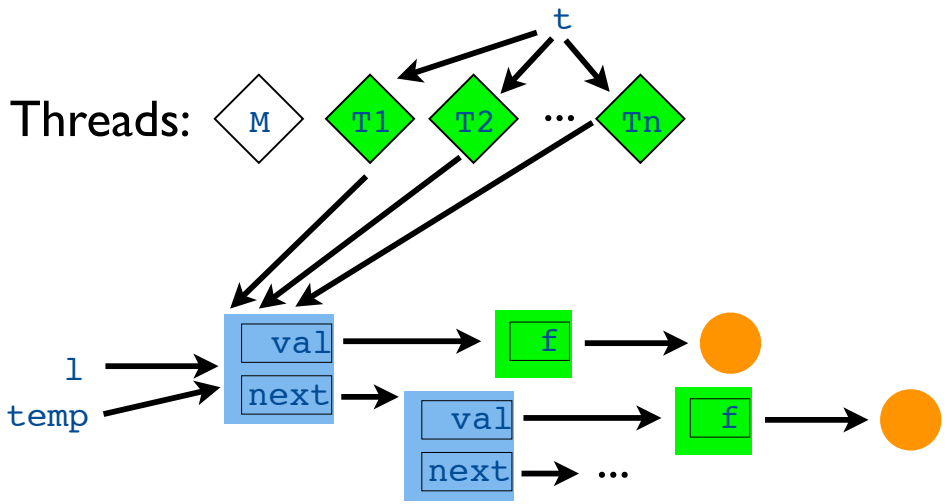
Example

```
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
      List temp = new List();
1:    temp.val = new T();
2:    temp.val.f = new A();
3:    temp.next = l;
      l = temp }
    while (*) {
      T t = new T();
4:    t.data = l;
      t.start();
5:    t.f = ...;}
    return;
   }
}

class T extends java.lang.Thread {
  A f;
  List data;
  void run(){
    while(*){
6:    List m = this.data;
7:    while (*) { m = m.next; }
8:    synchronized(m){ m.val.f = ...;}}
    return;}}
```

1. We create a link list `l`
2. We create a bunch of thread that all share the list `l`



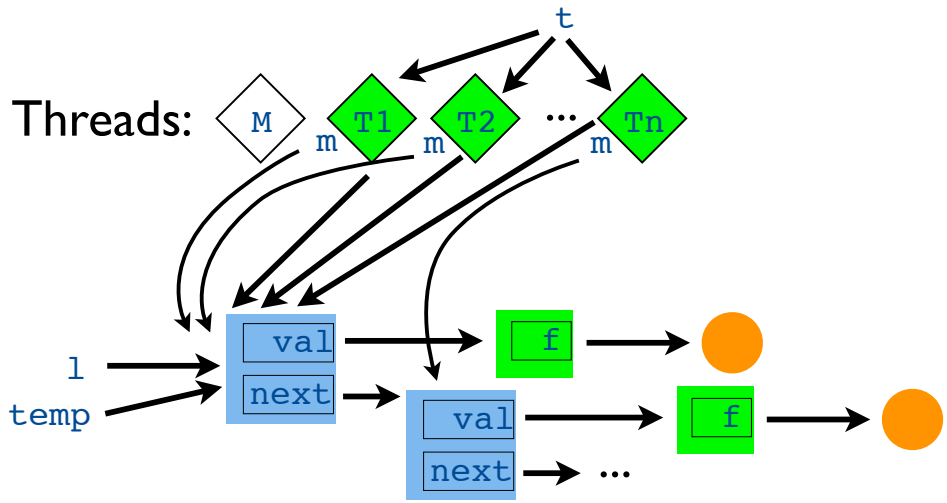
Example

```
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
      List temp = new List();
1:    temp.val = new T();
2:    temp.val.f = new A();
3:    temp.next = l;
      l = temp }
    while (*) {
      T t = new T();
4:    t.data = l;
      t.start();
5:    t.f = ...;}
    return;
   }
}

class T extends java.lang.Thread {
  A f;
  List data;
  void run(){
    while(*){
6:    List m = this.data;
7:    while (*) { m = m.next; }
8:    synchronized(m){ m.val.f = ...;}}
    return;}}
```

1. We create a link list `l`
2. We create a bunch of thread that all share the list `l`
3. Each thread chooses a cell, takes a lock on it and updates it.



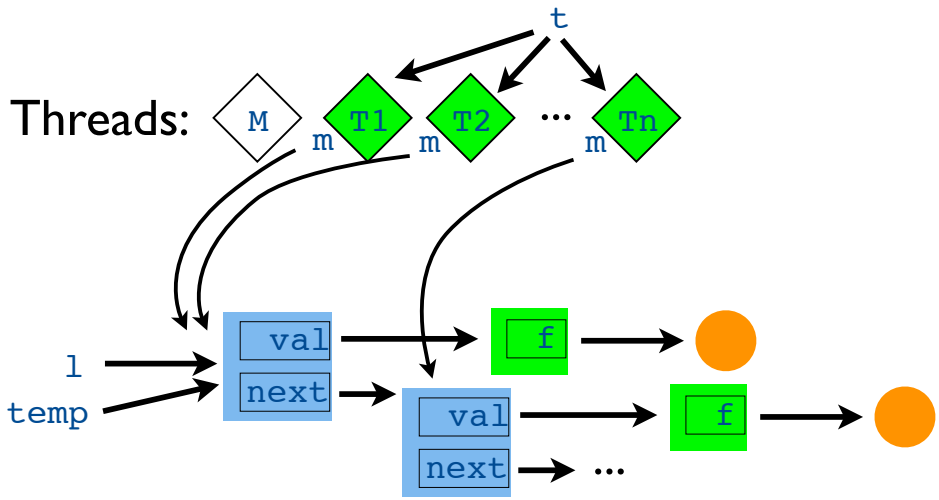
Example

```
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
      List temp = new List();
1:    temp.val = new T();
2:    temp.val.f = new A();
3:    temp.next = l;
      l = temp }
    while (*) {
      T t = new T();
4:    t.data = l;
      t.start();
5:    t.f = ...;}
    return;
   }
}

class T {
  A f;
  List data;
  void run(){
    while(*){
6:    List m = this.data;
7:    while (*) { m = m.next; }
8:    synchronized(m){ m.val.f = ...;}}
    return;}}
```

Points-to analysis computes a finite abstraction of the memory where locations are abstracted by their allocation site



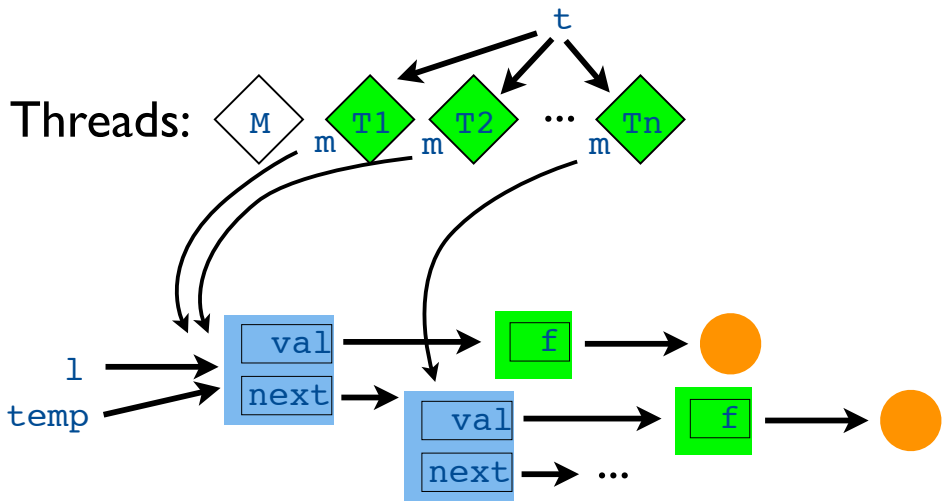
Example

```
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
    h1 List temp = new List();
1:  h2 temp.val = new T();
2:  h3 temp.val.f = new A();
3:    temp.next = l;
      l = temp }
    while (*) {
    h4 T t = new T();
4:    t.data = l;
      t.start();
5:    t.f = ...;}
    return;
   }
}

class T {
  A f;
  List data;
  void run(){
    while(*){
6:    List m = this.data;
7:    while (*) { m = m.next; }
8:    synchronized(m){ m.val.f = ...;}}
    return;}}
```

Points-to analysis computes a finite abstraction of the memory where locations are abstracted by their allocation site



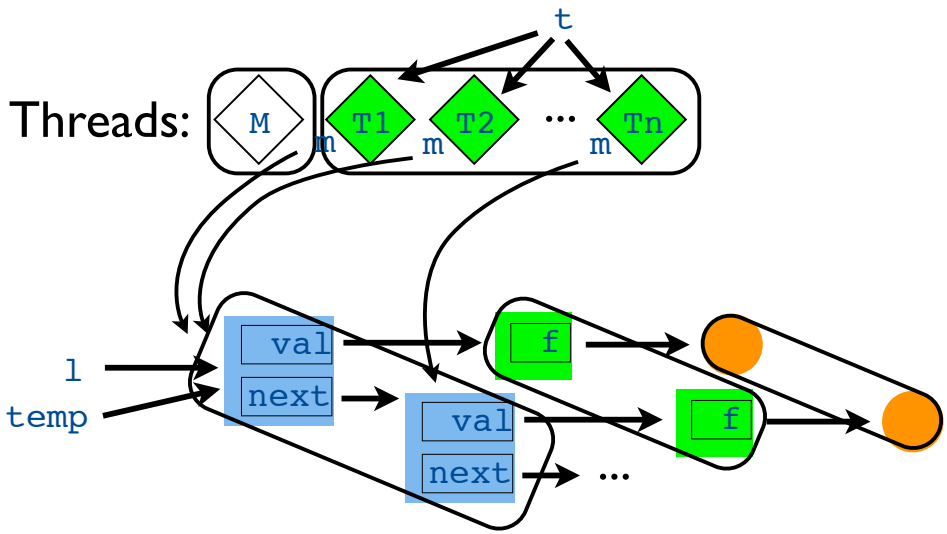
Example

```
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
    h1 List temp = new List();
1:  h2 temp.val = new T();
2:  h3 temp.val.f = new A();
3:    temp.next = l;
      l = temp }
    while (*) {
    h4 T t = new T();
4:    t.data = l;
      t.start();
5:    t.f = ...;}
    return;
   }
}

class T {
  A f;
  List data;
  void run(){
    while(*){
6:    List m = this.data;
7:    while (*) { m = m.next; }
8:    synchronized(m){ m.val.f = ...;}}
    return;}}
```

Points-to analysis computes a finite abstraction of the memory where locations are abstracted by their allocation site



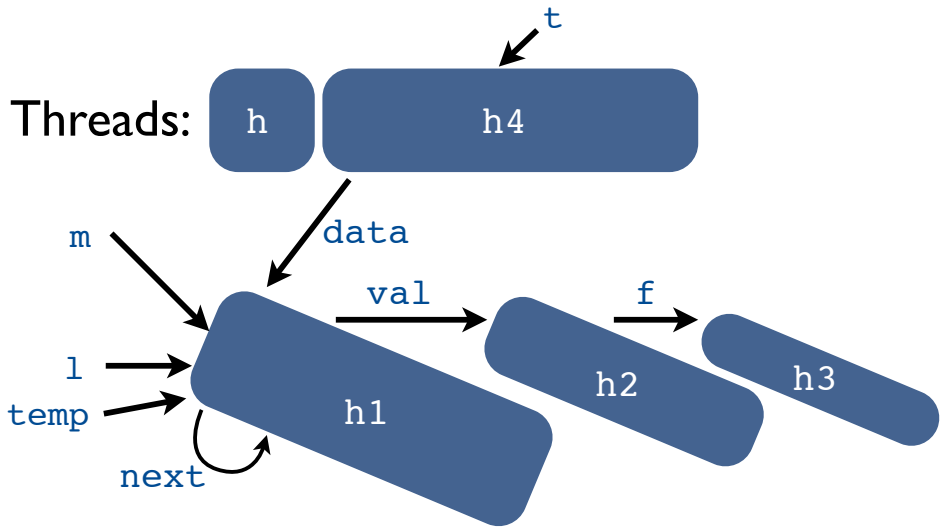
Example

```
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
    h1 List temp = new List();
1:  h2 temp.val = new T();
2:  h3 temp.val.f = new A();
3:    temp.next = l;
      l = temp }
    while (*) {
    h4 T t = new T();
4:    t.data = l;
      t.start();
5:    t.f = ...;}
    return;
   }
}

class T {
  A f;
  List data;
  void run(){
    while(*){
6:    List m = this.data;
7:    while (*) { m = m.next; }
8:    synchronized(m){ m.val.f = ...;}}
    return;}}
```

Points-to analysis computes a finite abstraction of the memory where locations are abstracted by their allocation site

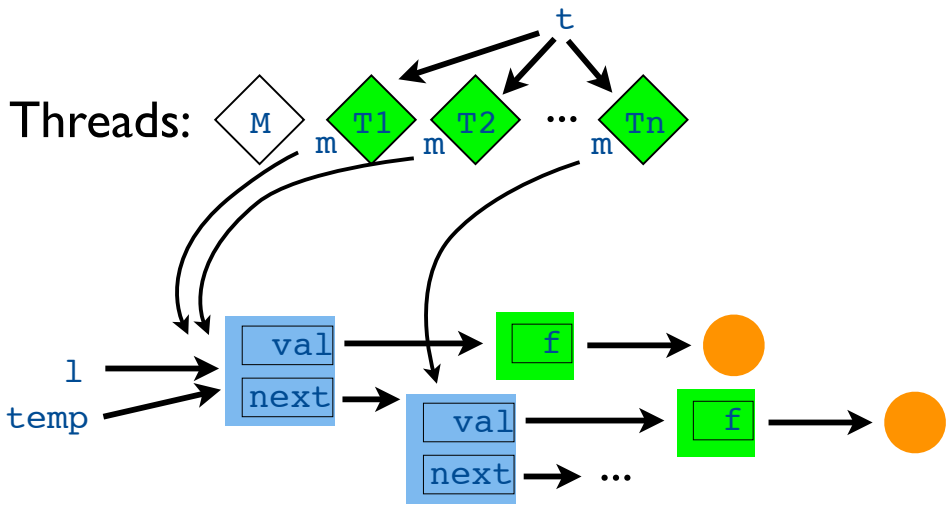


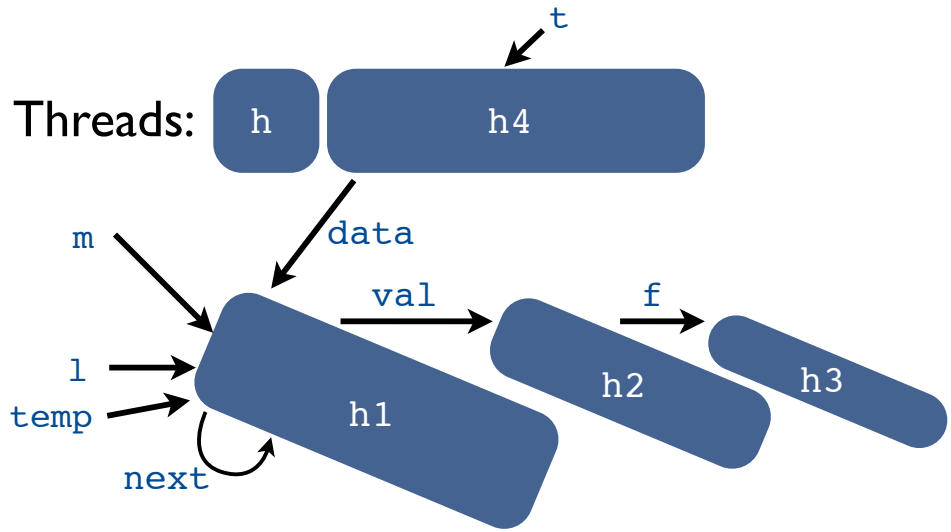
Example

```
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
    h1 List temp = new List();
1:  h2 temp.val = new T();
2:  h3 temp.val.f = new A();
3:    temp.next = l;
      l = temp }
    while (*) {
    h4 T t = new T();
4:    t.data = l;
      t.start();
5:    t.f = ...;}
    return;
   }
}

class T {
  A f;
  List data;
  void run(){
    while(*){
6:    List m = this.data;
7:    while (*) { m = m.next; }
8:    synchronized(m){ m.val.f = ...;}}
    return;}}
```





Points-to analysis

We compute a points-to relation for each variable and each field :

$$(PtV, PtF) \in (Var \to \wp(\mathfrak{O})) \times (\mathbf{Field} \to \wp(\mathfrak{O} \times \mathfrak{O}))$$

Informal meaning :

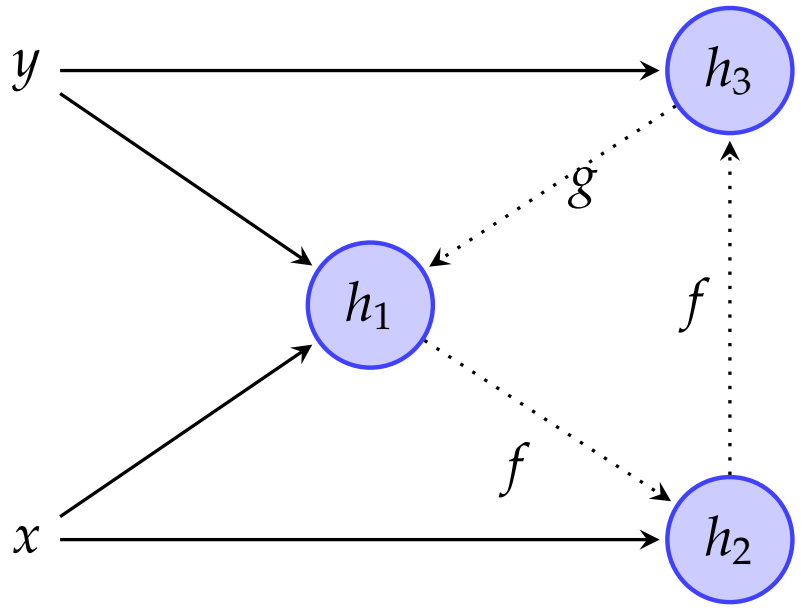
- $h \in PtV(x)$: the variable x may contain a reference to an object allocated at site h.
- $(h, h') \in PtF(f)$: the heap may contain an object allocated at site h whose fields f points to an object allocated at site h'.

Points-to graph

Each analysis result (PtV, PtF) can be represented by a graph.

Example :

$$\begin{aligned} PtV &= [x \mapsto \{h_1, h_2\}, y \mapsto \{h_1, h_3\}] \\ PtF &= [f \mapsto \{(h_1, h_2), (h_2, h_3)\}, g \mapsto \{(h_3, h_1)\}] \end{aligned}$$



Constraint based specification

Complete lattice structure on $(\mathit{Var} \to \wp(\mathcal{O})) \times (\mathbf{Field} \to \wp(\mathcal{O} \times \mathcal{O}))$:

Order :

$$(\mathit{PtV}_1, \mathit{PtF}_1) \sqsubseteq (\mathit{PtV}_2, \mathit{PtF}_2) \text{ iff } \begin{cases} \forall x \in \mathit{Var},\ \mathit{PtV}_1(x) \subseteq \mathit{PtV}_2(x) \\ \text{and} \\ \forall f \in \mathbf{Field},\ \mathit{PtF}_1(f) \subseteq \mathit{PtF}_2(f) \end{cases}$$

Least upper bound of an $S \subseteq (\mathit{Var} \to \wp(\mathcal{O})) \times (\mathbf{Field} \to \wp(\mathcal{O} \times \mathcal{O}))$:

$$\bigsqcup S = (\lambda x. \bigcup\{\mathit{PtV}(x) \mid (\mathit{PtV}, \_) \in S\}, \lambda f. \bigcup\{\mathit{PtF}(f) \mid (\_, \mathit{PtF}) \in S\})$$

Analysis of program P : the least solution $(\mathit{PtV}, \mathit{PtF})$ of the constraint system

$$\mathit{PtV}, \mathit{PtF} \vdash P$$

Constraint based specification

$$\frac{}{PtV, PtF \vdash \texttt{skip}} \qquad \frac{}{PtV, PtF \vdash x := \texttt{null}}$$

$$\frac{\{h\} \subseteq PtV(x)}{PtV, PtF \vdash x := \texttt{new}^{\,h} C} \qquad \frac{PtV(y) \subseteq PtV(x)}{PtV, PtF \vdash x := y}$$

$$\frac{\{h' \mid \exists h \in PtV(y), (h, h') \in PtF(f)\} \subseteq PtV(x)}{PtV, PtF \vdash x := y.f}$$

$$\frac{\{(h, h') \mid h \in PtV(x), h' \in PtV(y)\} \subseteq PtF(f)}{PtV, PtF \vdash x.f := y}$$

$$\frac{PtV, PtF \vdash S_1 \qquad PtV, PtF \vdash S_2}{PtV, PtF \vdash S_1 \,;\, S_2} \qquad \frac{PtV, PtF \vdash S_1 \qquad PtV, PtF \vdash S_2}{PtV, PtF \vdash \texttt{if}\ (*)\ \texttt{then}\ S_1\ \texttt{else}\ S_2}$$

$$\frac{PtV, PtF \vdash S}{PtV, PtF \vdash \texttt{while}\ (*)\ \texttt{do}\ S}$$

Exercise

Build and solve the constraint system for the points-to analysis of the following program :

$$x := \texttt{new}^{\,h_1} C \,;\, y := \texttt{new}^{\,h_2} C \,;\, \texttt{if}\ (*)\ \texttt{then}\ (x.f := y \,;\, y := x)\ \texttt{else}\ (y := x.f)$$

Exercise

Build and solve the constraint system for the points-to analysis of the following program :

$$x := \texttt{new}^{\,h_1} C \,;\, y := \texttt{new}^{\,h_2} C \,;\, \texttt{if}\ (*)\ \texttt{then}\ (x.f := y \,;\, y := x)\ \texttt{else}\ (y := x.f)$$

$$\begin{array}{rcl} \{h_1\} & \subseteq & PtV(x) \\ \{h_2\} & \subseteq & PtV(y) \\ \{(h,h') \mid h \in PtV(x), h' \in PtV(y)\} & \subseteq & PtF(f) \\ PtV(x) & \subseteq & PtV(y) \\ \{h' \mid \exists h \in PtV(x), (h,h') \in PtF(f)\} & \subseteq & PtV(y) \end{array}$$

The set of constraints is then transformed into a set of equations.

Exercise

Build and solve the constraint system for the points-to analysis of the following program :

$$x := \texttt{new}^{h_1} C \,;\, y := \texttt{new}^{h_2} C \,;\, \texttt{if}\ (*)\ \texttt{then}\ (x.f := y \,;\, y := x)\ \texttt{else}\ (y := x.f)$$

$$\begin{aligned} PtV(x) &= \{h_1\} \\ PtV(y) &= \{h_2\} \cup PtV(x) \cup \{h' \mid \exists h \in PtV(x), (h,h') \in PtF(f)\} \\ PtF(f) &= \{(h,h') \mid h \in PtV(x), h' \in PtV(y)\} \end{aligned}$$

This set of equations can be simplified . . .

Exercise

Build and solve the constraint system for the points-to analysis of the following program :

$$x := \texttt{new}^{h_1} C \,;\, y := \texttt{new}^{h_2} C \,;\, \texttt{if}\ (*)\ \texttt{then}\ (x.f := y \,;\, y := x)\ \texttt{else}\ (y := x.f)$$

$$\begin{aligned} PtV(x) &= \{h_1\} && (1)\\ PtV(y) &= \{h_1, h_2\} \cup \{h' \mid (h_1, h') \in PtF(f)\} && (2)\\ PtF(f) &= \{(h_1, h') \mid h' \in PtV(y)\} && (3) \end{aligned}$$

and the least solution can be calculated by iteration :

Exercise

Build and solve the constraint system for the points-to analysis of the following program :

$$x := \texttt{new}^{h_1} C \,;\, y := \texttt{new}^{h_2} C \,;\, \texttt{if}\ (*)\ \texttt{then}\ (x.f := y \,;\, y := x)\ \texttt{else}\ (y := x.f)$$

$$\begin{aligned} PtV(x) &= \{h_1\} && (1)\\ PtV(y) &= \{h_1, h_2\} \cup \{h' \mid (h_1, h') \in PtF(f)\} && (2)\\ PtF(f) &= \{(h_1, h') \mid h' \in PtV(y)\} && (3) \end{aligned}$$

and the least solution can be calculated by iteration :

		(2)	(3)	
$PtV(y)$	$\emptyset$	$\{h_1, h_2\}$	$\{h_1, h_2\}$	stable
$PtF(f)$	$\emptyset$	$\emptyset$	$\{(h_1, h_1), (h_1, h_2)\}$	stable

Exercise

Build and solve the constraint system for the points-to analysis of the following program :

$$x := \texttt{new}^{\,h_1} C \,;\, y := \texttt{new}^{\,h_2} C \,;\, \texttt{if}\ (*)\ \texttt{then}\ (x.f := y \,;\, y := x)\ \texttt{else}\ (y := x.f)$$

$$\begin{aligned} PtV(x) &= \{h_1\} && (1)\\ PtV(y) &= \{h_1, h_2\} \cup \{h' \mid (h_1, h') \in PtF(f)\} && (2)\\ PtF(f) &= \{(h_1, h') \mid h' \in PtV(y)\} && (3) \end{aligned}$$

and the least solution can be calculated by iteration :

		(2)	(3)	
$PtV(y)$	$\emptyset$	$\{h_1, h_2\}$	$\{h_1, h_2\}$	stable
$PtF(f)$	$\emptyset$	$\emptyset$	$\{(h_1, h_1), (h_1, h_2)\}$	stable

$$PtV(x) = \{h_1\} \qquad PtV(y) = \{h_1, h_2\} \qquad PtF(f) = \{(h_1, h_1), (h_1, h_2)\}$$

Vocabulary

An analysis is said *flow-insensitive* if the order of statements in a program does not affect the result of the analysis.

Example :

- the points-to analysis seen before is flow-insensitive,
- the reachable definition analysis is flow-sensitive

Flow-sensitive points-to analysis

We can define a flow-sensitive version (only for local variables) of the previous points-to analysis :

$$(PtV, PtF) \in (\mathbf{Lab} \to Var \to \wp(\mathcal{O})) \times (\mathbf{Field} \to \wp(\mathcal{O} \times \mathcal{O}))$$

using the following convenient notation : for $V, V' \in Var \to \wp(\mathcal{O})$, we define $V \sqsubseteq_S V'$ by

$$V \sqsubseteq_S V' \text{ iff } \forall y \in S,\ V(y) \subseteq V'(y)$$

Flow-sensitive points-to analysis

We can define a flow-sensitive version (only for local variables) of the previous points-to analysis :

$$(PtV, PtF) \in (\mathbf{Lab} \to Var \to \wp(\mathcal{O})) \times (\mathbf{Field} \to \wp(\mathcal{O} \times \mathcal{O}))$$

using the following convenient notation : for $V, V' \in Var \to \wp(\mathcal{O})$, we define $V \sqsubseteq_S V'$ by

$$V \sqsubseteq_S V' \text{ iff } \forall y \in S,\ V(y) \subseteq V'(y)$$

$$\frac{PtV(l) \sqsubseteq_{Var} PtV(l')}{PtV, PtF \vdash [\ \mathtt{skip}\ ]^l, l'} \qquad \frac{PtV(l) \sqsubseteq_{Var\setminus\{x\}} PtV(l')}{PtV, PtF \vdash [x := \mathtt{null}\ ]^l, l'}$$

$$\frac{\{h\} \subseteq PtV(l')(x) \qquad PtV(l) \sqsubseteq_{Var\setminus\{x\}} PtV(l')}{PtV, PtF \vdash [x := \mathtt{new}\ ^h C]^l, l'}$$

$$\frac{PtV(l)(y) \subseteq PtV(l')(x) \qquad PtV(l) \sqsubseteq_{Var\setminus\{x\}} PtV(l')}{PtV, PtF \vdash [x := y]^l, l'}$$

Flow-sensitive points-to analysis

$$\frac{\{h' \mid \exists h \in PtV(l)(y), (h,h') \in PtF(f)\} \subseteq PtV(l')(x) \qquad PtV(l) \sqsubseteq_{Var\setminus\{x\}} PtV(l')}{PtV, PtF \vdash [x := y.f]^l, l'}$$

$$\frac{\{(h,h') \mid h \in PtV(l)(x), h' \in PtV(l)(y)\} \subseteq PtF(f) \qquad PtV(l) \sqsubseteq_{Var} PtV(l')}{PtV, PtF \vdash [x.f := y]^l, l'}$$

$$\frac{PtV, PtF \vdash S_1, init(S_2) \qquad PtV, PtF \vdash S_2, l'}{PtV, PtF \vdash S_1 \,;\, S_2, l'}$$

$$\frac{\begin{array}{ll} PtV, PtF \vdash S_1, l' & PtV(l) \sqsubseteq_{Var} PtV(init(S_1)) \\ PtV, PtF \vdash S_2, l' & PtV(l) \sqsubseteq_{Var} PtV(init(S_2)) \end{array}}{PtV, PtF \vdash \texttt{if}\,[(*)]^l\,\texttt{then}\,S_1\,\texttt{else}\,S_2, l'}$$

$$\frac{PtV, PtF \vdash S, l \qquad PtV(l) \sqsubseteq_{Var} PtV(init(S)) \qquad PtV(l) \sqsubseteq_{Var} PtV(l')}{PtV, PtF \vdash \texttt{while}\,[(*)]^l\,\texttt{do}\,S, l'}$$

Method call

The previous analysis is *intraprocedural* because it deals with a simple language without functions or procedures.

Taking into account procedures (called methods in the OO vocabulary) requires an *interprocedural* analysis.

Language extension :

- A method $m \in \mathbf{Meth}$ is given by a sequence of parameters $m.params \in Var^\star$ and a set of instruction $m.instrs \in \mathbf{Stm}$.
- A program is now a sequence of methods.
- We add a new statement $x.m(x_1, \ldots, x_n)$ with $n = |m.params|$
- Informal semantics of a method call : $x.m(x_1, \ldots, x_n)$ calls the method m with formal parameters $m.params = \{p_1, \ldots, p_n\}$ bound to the values of $x_1, \ldots, x_n$ and a special parameter $this$ bound to the value of x.

Example

$$
\begin{array}{l}
main = \{\ params = \varepsilon, \\
\qquad instrs = \ \ x_1 := \mathtt{new}^{\,h_1} X\ ; \\
\qquad\qquad\quad\ x_2 := \mathtt{new}^{\,h_2} X\ ; \\
\qquad\qquad\quad\ y_1 := \mathtt{new}^{\,h_3} Y\ ; \\
\qquad\qquad\quad\ y_2 := \mathtt{new}^{\,h_4} Y\ ; \\
\qquad\qquad\quad\ y_1.set(x_1)\ ; \\
\qquad\qquad\quad\ y_2.set(x_2)\ ; \\
\} \\
set = \{params = \{v\},\ instrs = this.f := v\}
\end{array}
$$

Context-insensitive extension

The points-to analysis of a program $P = \{m_1, \ldots, m_k\}$ is defined as the least solution PtV, PtF of

$$PtV, PtF \vdash m_1, \quad \ldots, \quad PtV, PtF \vdash m_k$$

with the extra rule

$$\frac{m.params = \{p_1, \ldots, p_n\} \qquad PtV(x) \subseteq PtV(this) \quad PtV(x_1) \subseteq PtV(p_1) \quad \cdots \quad PtV(x_n) \subseteq PtV(p_n)}{PtV, PtF \vdash x.m(x_1, \ldots, x_n)}$$

Exercise

Generate and solve the constraints for the following $While_{\mathcal{O}}$ program $P = \{main, set\}$ with

$$
\begin{array}{l}
main = \{\ params = \emptyset, \\
\qquad\quad instrs = \ \ x_1 := \texttt{new}^{\,h_1} C\ ; \\
\qquad\qquad\qquad\ \ x_2 := \texttt{new}^{\,h_2} C\ ; \\
\qquad\qquad\qquad\ \ y_1 := \texttt{new}^{\,h_3} D\ ; \\
\qquad\qquad\qquad\ \ y_2 := \texttt{new}^{\,h_4} D\ ; \\
\qquad\qquad\qquad\ \ y_1.set(x_1)\ ; \\
\qquad\qquad\qquad\ \ y_2.set(x_2)\ ; \\
\}
\end{array}
$$

$$
set = \{params = \{v\},\ instrs = this.f := v\}
$$

Vocabulary

This analysis is said to be *context-insensitive* since it analyses the method ***set*** only once, combining the two calling states of

1. $y_1.set(x_1)$ and
2. $y_2.set(x_2)$.

A *context-sensitive* version will distinguish these two call sites.

Outline

Soundness proof of the points-to analysis

The soundness of the analysis is proved w.r.t. an instrumented version of the operational semantics.

We tag references with their allocation sites :

$$\frac{r_{h'} \notin \mathbf{dom}(\sigma) \quad \forall h' \in \mathcal{H}}{(x := \mathbf{new}^{\,h} C, \rho, \sigma) \rightarrow \rho[x \mapsto r_h], \sigma[r_h \mapsto \lambda f.\diamond]}$$

The condition $r_{h'} \notin \mathrm{dom}(\sigma) \quad \forall h' \in \mathcal{H}$ guarantees that the reference r is **fresh**.

This is a safe instrumentation since we obtain exactly the previous semantics by discarding the reference tags.

Approximation relation

Approximation relation $\sim$ between a state $s \in \mathbf{State}$ and an analysis result (PtV, PtF)

$$(\rho, \sigma) \sim (PtV, PtF)$$

iff

1. $\forall x \in Var, \rho(x) = r_h \in \mathbf{dom}(\sigma) \Rightarrow h \in PtV(x)$

2. $\forall r_h \in \mathbf{dom}(\sigma), \forall f \in \mathbf{Field},$
   $\sigma(r_h)(f) = r'_{h'} \Rightarrow (h, h') \in PtF(f)$

Soundness proof

Lemma (Subject Reduction)

$(S, s) \rightarrow (S', s')$ *implies*

- *if* $PtV, PtF \vdash S$ *and* $s \sim (PtV, PtF)$,
- *then* $s' \sim (PtV, PtF)$ *and* $PtV, PtF \vdash S'$.

$(S, s) \rightarrow s'$ *implies*

- *if* $PtV, PtF \vdash S$ *and* $s \sim (PtV, PtF)$,
- *then* $s' \sim (PtV, PtF)$.

Proof : by induction on $\rightarrow$.

Proof of lemma : a few cases

Assignment :

$$\frac{}{(x := y, \rho, \sigma) \rightarrow \rho[x \mapsto \rho(y)], \sigma}$$

Assume $\boldsymbol{PtV, PtF \vdash x := y}$ which, according to the rules defining the analysis, means that $PtV(y) \subseteq PtV(x)$.

Assume $(\rho, \sigma) \sim (PtV, PtF)$.
Have to show $(\rho[x \mapsto \rho(y)], \sigma) \sim (PtV, PtF)$.

This amounts to showing

$$\forall z \in Var, \rho[x \mapsto \rho(y)](z) = r_h \in \mathrm{dom}(\sigma) \Rightarrow h \in PtV(z).$$

Case $z = x$: the assumption is that $\rho(y) = r_h \in \mathrm{dom}(\sigma)$, implying that $h \in PtV(y)$. Use $PtV(y) \subseteq PtV(x)$ to deduce that $h \in PtV(x) = PtV(z)$.

Proof of lemma : a few cases

Sequence (1/2) :

$$\frac{(S_1, \rho, \sigma) \rightarrow \rho', \sigma'}{(S_1 \; ; S_2, \rho, \sigma) \rightarrow (S_2, \rho', \sigma')}$$

Assume $PtV, PtF \vdash S_1 \; ; S_2$ which from the rule for $\;;\;$ means that $PtV, PtF \vdash S_1$ and $PtV, PtF \vdash S_2$.
Assume furthermore $(\rho, \sigma) \sim (PtV, PtF)$.

The induction hypothesis (IH) now becomes

- if $PtV, PtF \vdash S_1$ and $(\rho, \sigma) \sim (PtV, PtF)$,
- then $(\rho', \sigma') \sim (PtV, PtF)$.

From the IH and the assumptions we can then deduce that $\rho', \sigma' \sim (PtV, PtF)$.
Another assumption was that $PtV, PtF \vdash S_2$.

These two facts conclude the case for this rule.

Proof of lemma : a few cases

Sequence (2/2) :

$$\frac{(S_1, \rho, \sigma) \to (S_1', \rho', \sigma')}{(S_1 \,;\, S_2, \rho, \sigma) \to (S_1' \,;\, S_2, \rho', \sigma')}$$

Assume $PtV, PtF \vdash S_1$ and $PtV, PtF \vdash S_2$.
Assume $(\rho, \sigma) \sim (PtV, PtF)$.
The induction hypothesis (IH) is that

- if $PtV, PtF \vdash S_1$ and $(\rho, \sigma) \sim (PtV, PtF)$,
- then $(\rho', \sigma') \sim (PtV, PtF)$ and $PtV, PtF \vdash S_1'$.

Show $\rho', \sigma' \sim (PtV, PtF)$ and $PtV, PtF \vdash S_1' \,;\, S_2$.

We get $\rho', \sigma' \sim (PtV, PtF)$ from $PtV, PtF \vdash S_1$ and $(\rho, \sigma) \sim (PtV, PtF)$ and the IH.
We get $PtV, PtF \vdash S_1'$ from $PtV, PtF \vdash S_1$ and $(\rho, \sigma) \sim (PtV, PtF)$ and the IH.
We get $PtV, PtF \vdash S_1' \,;\, S_2$ from $PtV, PtF \vdash S_1'$ and $PtV, PtF \vdash S_2$.

Soundness proof

Initial state : $s_0 = (\lambda x.\diamond, \lambda r.\text{undefined})$

Theorem (Points-to soundness)

If $PtV, PtF \vdash P$ *then*

- $(P, s_0) \rightarrow^\star (S', s')$ *implies* $s' \sim (PtV, PtF)$
- $(P, s_0) \rightarrow^\star s'$ *implies* $s' \sim (PtV, PtF)$

Proof : by induction on $\rightarrow^\star$.

Theorem (May-alias soundness)

If $(\rho, \sigma) \sim (PtV, PtF)$ *then for all variable* $x, y \in Var$, $PtV(x) \cap PtV(y) = \emptyset$ *implies* $\rho(x) \neq \rho(y)$.

Summary

Constraint-based analysis :

- Compositional, directed by the syntax of the language.
- Equivalent to data flow techniques (for monotone constraints) via Knaster-Tarski Fixpoint Theorem.

Points-to analysis for finding may- and must aliases :

- Flow-sensitive analysis
- Context-sensitivity
- Correctness proof based on invariance (“subject-reduction”).

Reading

Sections 1 and 2 of

Ana Milanova, Atanas Rountev, and Barbara G. Ryder,
Parameterized Object Sensitivity for Points-to Analysis for Java,
ACM Transactions on Software Engineering and Methodology (TOSEM),
vol. 14, no. 1, pp. 1-41, January 2005.