Abstract Interpretation and the Heap

Computer Science and Artificial Intelligence Laboratory
MIT

Nov 16, 2015

Recap

An abstract domain is a lattice

*Some analysis relax this restriction.

- Elements in the lattice are called *Abstract Values*

Need to relate elements in the lattice with states in the program

- **Abstraction Function**: $\alpha: \mathcal{P}(\mathcal{V}) \rightarrow Abs$
  - Maps a value in the program to the “best” abstract value
- **Concretization Function:** $\gamma: Abs \rightarrow \mathcal{P}(\mathcal{V})$
  - Maps an abstract value to a set of values in the program

Modeling the Heap

Giant Array vs. Collection of Objects (C vs Java view)

Giant array view

- $s \in S : Id \rightarrow Int$
- $h \in H : Nat \rightarrow Int$
- $[\![C]\!]: S \times H \rightarrow S \times H \cup \{\perp\}$
- $[\![E]\!] : S \rightarrow Int$
- $[\![x = [e]]\!]\ s\ h = s\{x \rightarrow h([\![e]\!]\ s)\}\ h$
- $[\![[e] = x]\!]\ s\ h = s\ h\{[\![e]\!]\ s \rightarrow \mathrm{s(x)}\}$
- $[\![x = cons(e_0 \dots e_k)]\!]\ s\ h = s\{x \rightarrow j\}\ h\{j \rightarrow [\![e_0]\!]\ s, \dots, j + k \rightarrow [\![e_k]\!]\ s\}$
  $where\ j = (\max dom\ h) + 1$

Modeling the Heap

Giant Array vs. Collection of Objects (C vs Java view)

Collection of Objects View

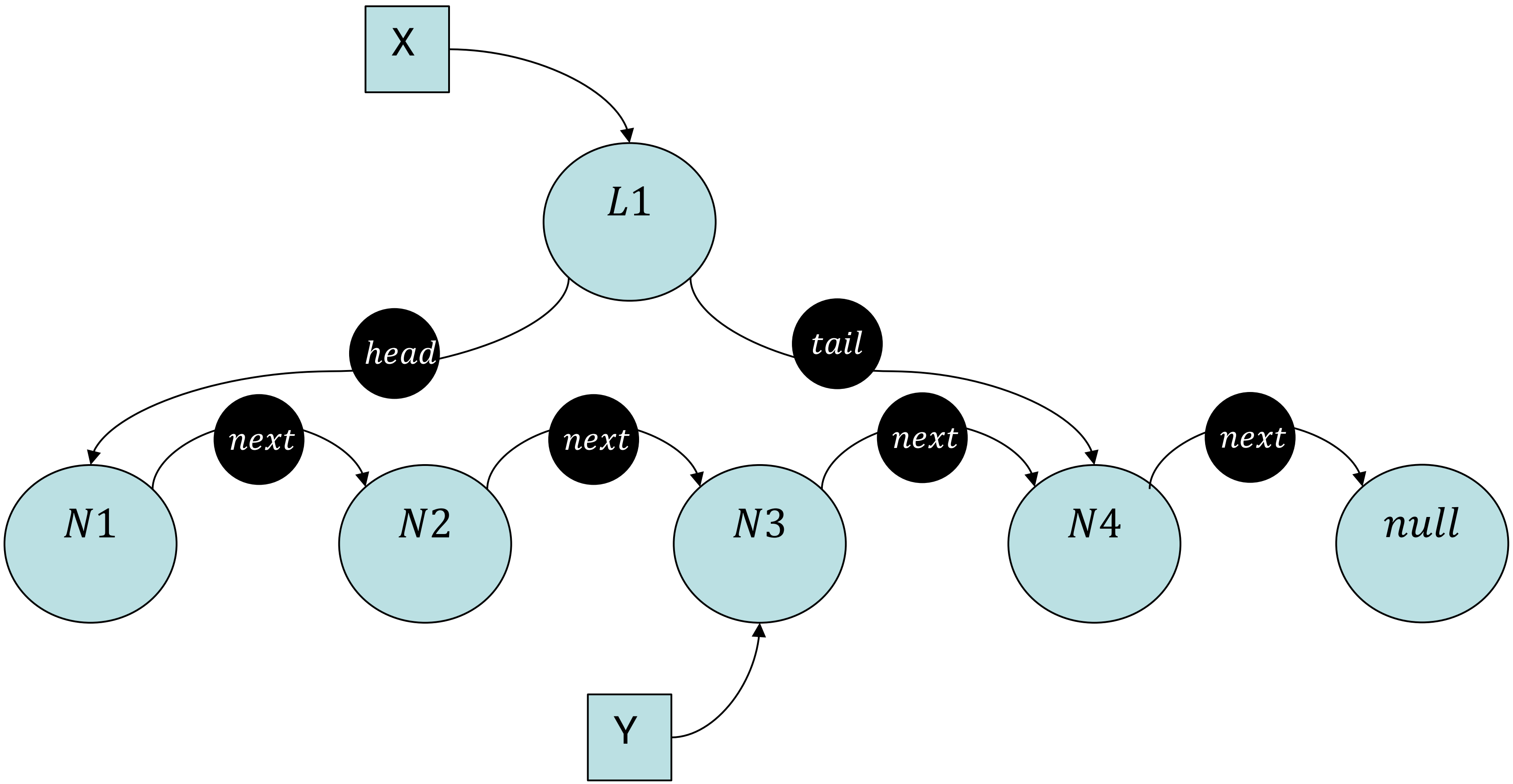
- $s \in S : Id \rightarrow Addr$
- $h \in H : Addr \times Id \rightarrow Addr$
- $[\![C]\!]: S \times H \rightarrow S \times H \cup \{\bot\}$
- $[\![E]\!] : S \rightarrow Addr$
- $[\![x = e.f]\!]\ s\ h = s\{x \rightarrow h([\![e]\!]\ s, f)\}\ h$
- $[\![e.f = x]\!]\ s\ h = s\ h\{([\![e]\!]\ s, f) \rightarrow \mathrm{s(x)}\}$
- $[\![x = cons(e_0 \ldots e_k)]\!]\ s\ h = s\{x \rightarrow j\}\ h\{(j, f_0) \rightarrow [\![e_0]\!]\ s, \ldots, (j, f_k) \rightarrow [\![e_k]\!]\ s\}$
  $where\ j = fresh\ address$

This is the view we will focus on today

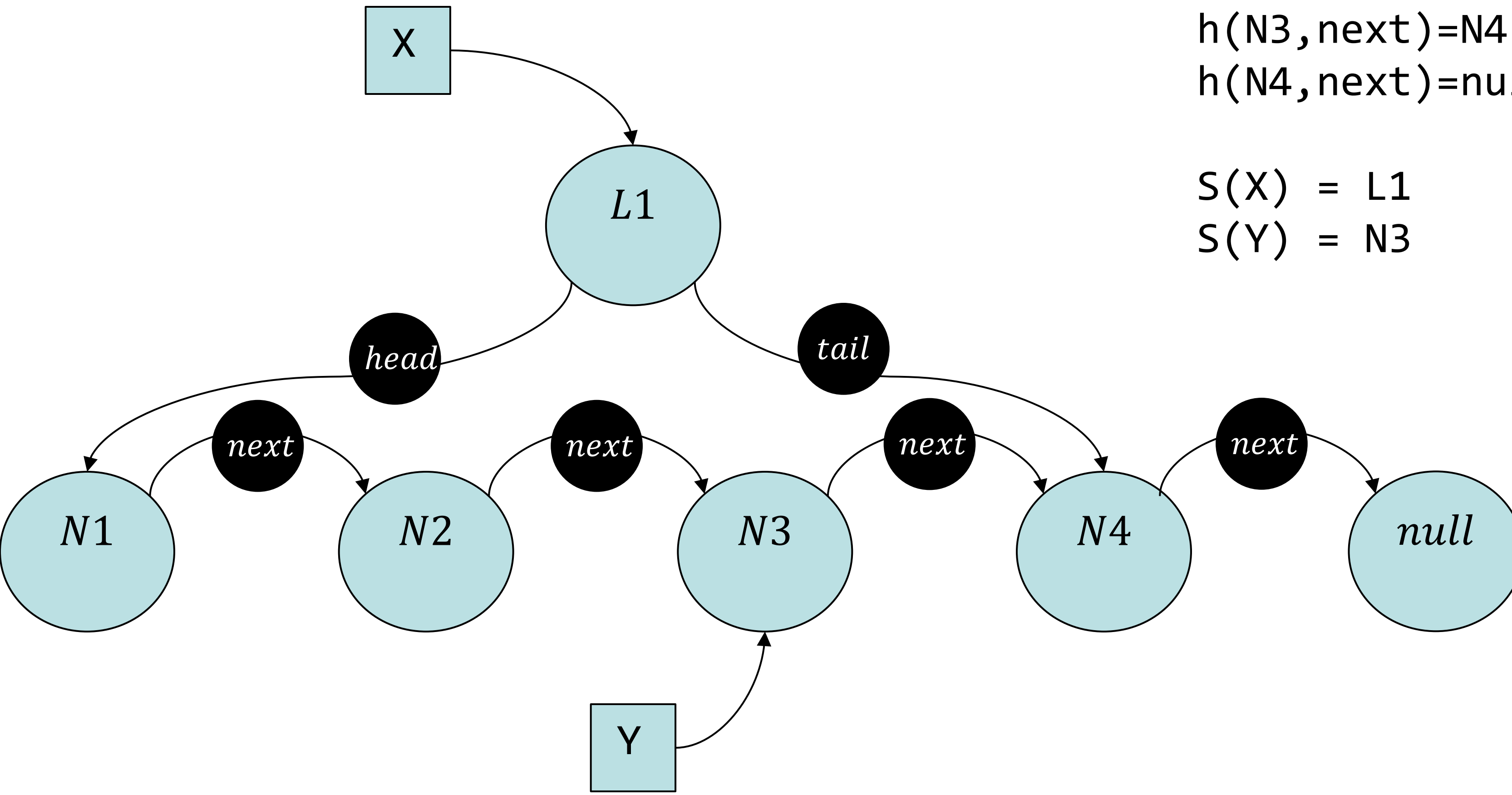
The pset provides a third alternative

- Each object is indexed by integer offsets rather than fields
- Not significantly different from this alternative

The state as a graph



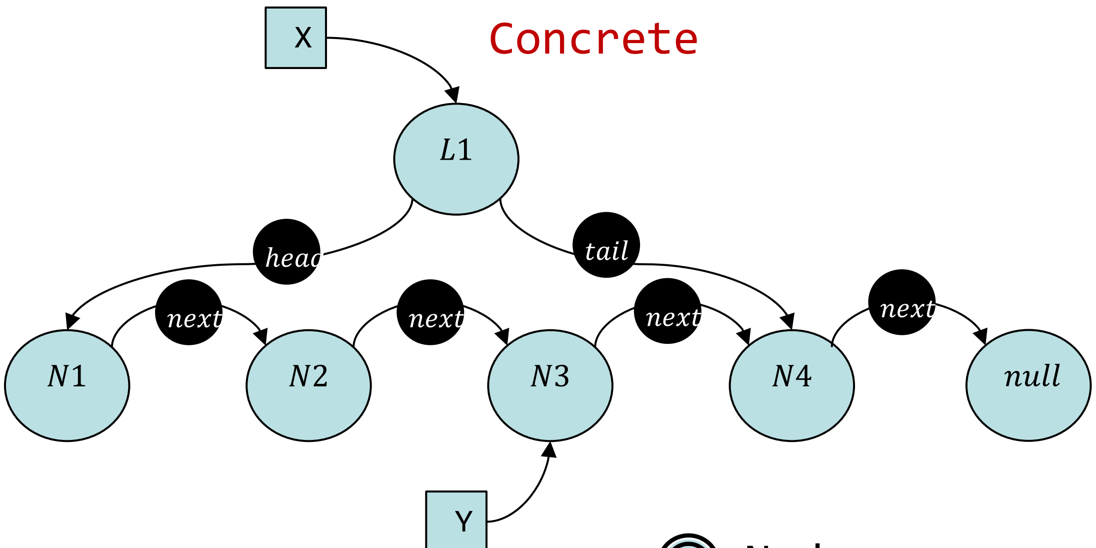
The state as a graph



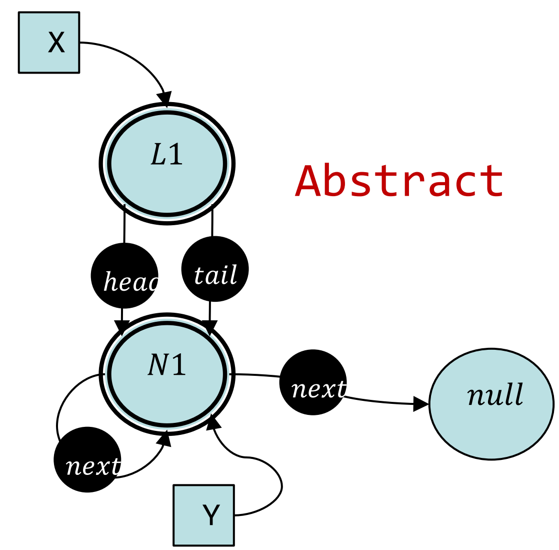

```
h(L1,head)=N1
h(L1,tail)=N4
h(N1,next)=N2
h(N2,next)=N3
h(N3,next)=N4
h(N4,next)=null

S(X) = L1
S(Y) = N3
```

Try 1: A simple abstraction

Have a single node for all objects of the same type



Nodes are abstract heap locations representing multiple concrete heap nodes. Known as *summary nodes*



Formal definition

Let $\tau(addr)$ be the *summary* node representing an address (we have one for each type)

- We can define a special node $null = \tau(null)$

Abstraction function

- $\alpha(h, S) \coloneqq (\bar{h}, \bar{S})$
- $\bar{h}(t, f) \coloneqq \{\, t' \mid \exists\, a \in Addr\ ,\ \tau(a) = t \wedge h(a, f) = a' \wedge \tau(a') = t'\}$
- $\bar{S}(x) \coloneqq \{\tau\big(S(x)\big)\}$

Partial order

- $\left(\overline{h_1}, \overline{S_1}\right) \sqsubseteq \left(\overline{h_2}, \overline{S_2}\right)\ iff\ \forall t, f\ \ \bar{h}_1(t, f) \subseteq \bar{h}_2(t, f) \wedge \forall x\ \bar{S}_1(x) \subseteq \bar{S}_2(x)$

Concretization

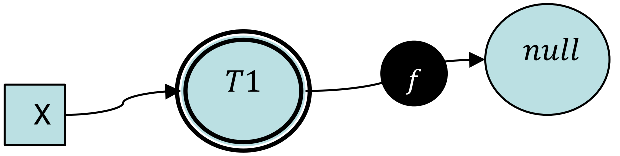
- $(h, S) \in \gamma(\bar{h}, \bar{S})$ iff
  $\Big(h(a, f) = b \Rightarrow \tau(b) \in \bar{h}(\tau(a), f)\Big) \wedge \big(S(x) = a \Rightarrow \tau(a) \in\ \bar{S}(x)\big)$

Update

$$[\![e.f = x]\!](\bar{h}, \bar{S}) = (\bar{h}', \bar{S})$$

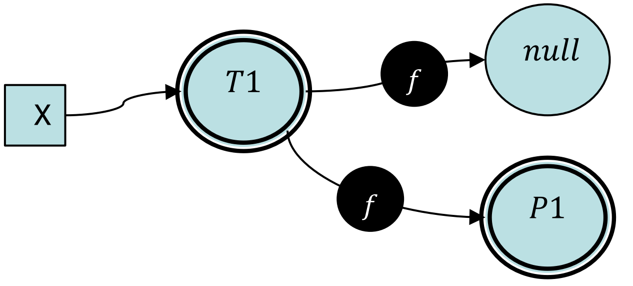
Where $\bar{h}'(t,f) = \begin{cases} \bar{h}(t,f) & if\ t \notin [\![e]\!](\bar{h}, \bar{S}) \\ \bar{h}(t,f) \cup \bar{S}(x) & if\ t \in [\![e]\!](\bar{h}, \bar{S}) \end{cases}$

The problem of destructive updates



$x = new\ T(\quad);$
$x.f = null;$
$x.f = new\ P(\quad);$

The problem of destructive updates



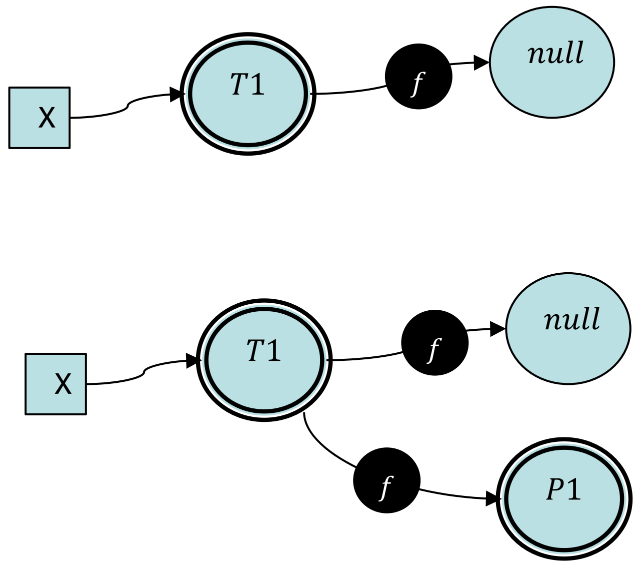
$x = new\ T(\ \ );$
$x.f = null;$
$x.f = new\ P(\ \ );$

The abstraction cannot tell that $x.f$ is no longer null

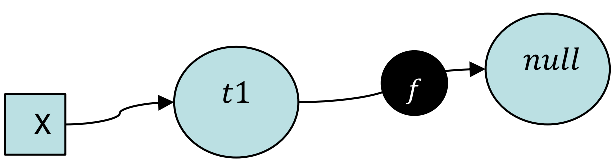
Why not?

The problem of destructive updates

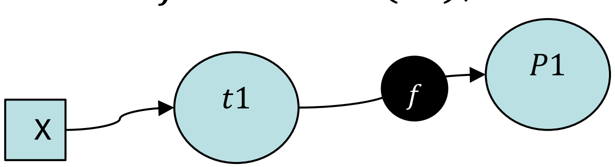
$x = new\, T(\ \ );$
$x.f = null;$
$x.f = new\, P(\ \ );$



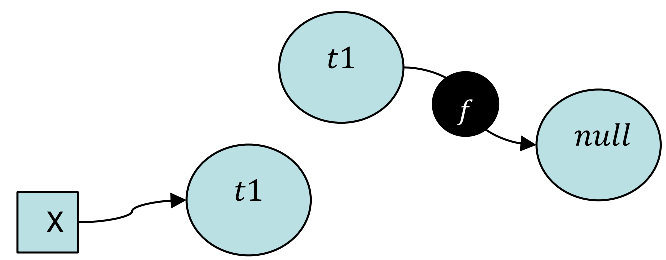
Why is this the best we can do?



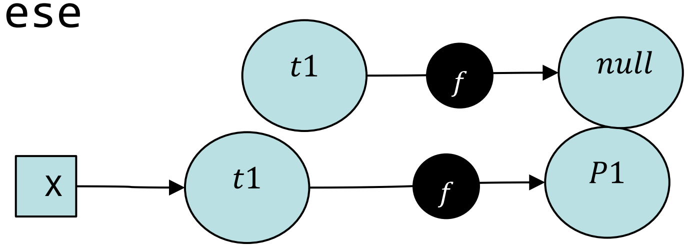
$x.f = new\, P(\ \ );$



Abstraction cannot distinguish these two concrete cases



$x.f = new\, P(\ \ );$



The problem

All abstract heap nodes represented multiple concrete heap nodes

- This makes it impossible to do destructive updates

The abstract domain in the pset is more refined but it suffers from the same problem

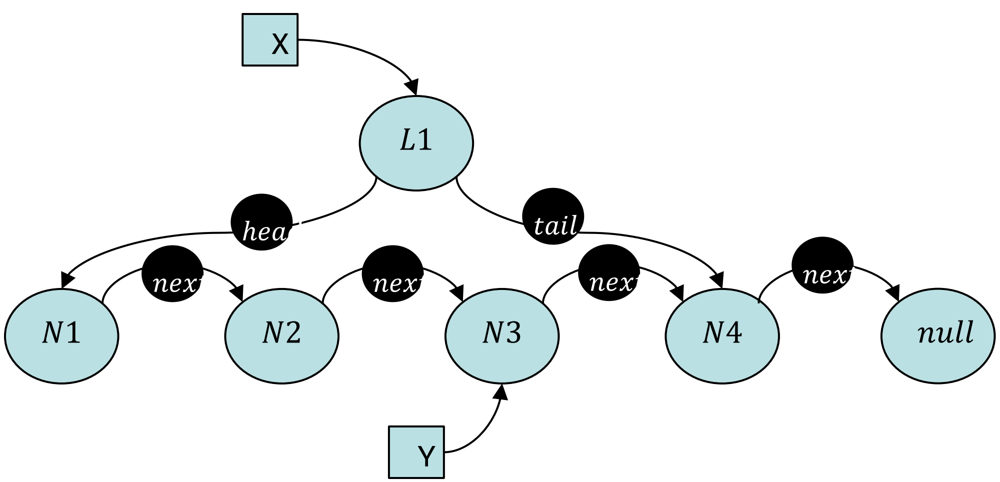
Try 2: Abstract based on Variables

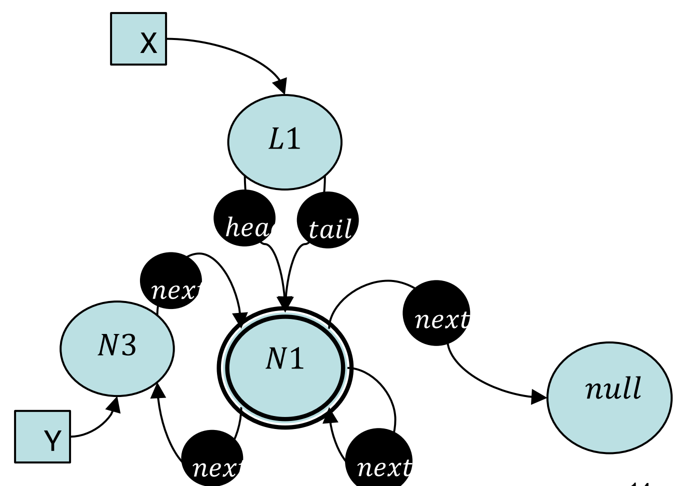
"Solving Shape-Analysis Problems in Languages with Destructive Updating" Sagiv, Reps & Wilhelm

- We'll simplify a little relative to this paper

Idea

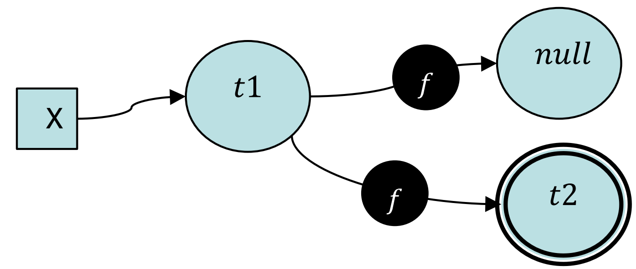
- Objects pointed to by variables should be concretized





Example

$x = new\, T(\quad);$
$x.f = null;$
$x.f = new\, T(\quad);$

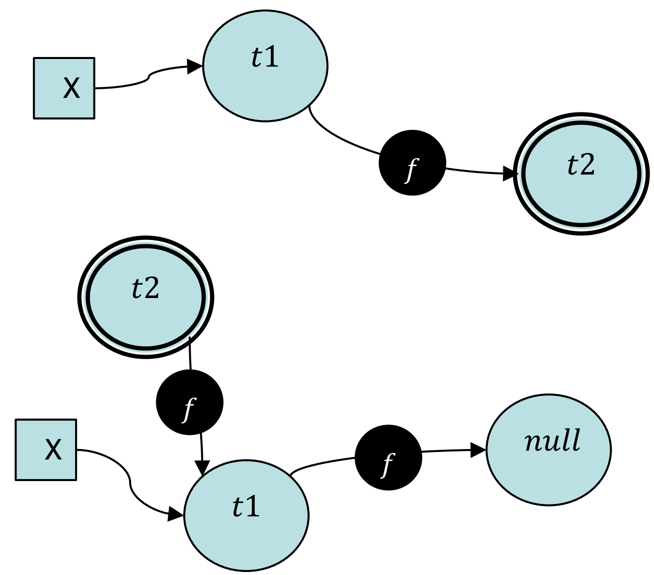


X always points to a concrete location

This allows a destructive update to x.f

Example

$x = new\ T(\ \ );$
$x.f = null;$
$x.f = new\ T(\ \ );$
$x = x.f$
$x.f = null$



Note that t1 is "the location pointed to by x" and not a specific concrete node

Formalization

Let $PVar$ be the set of variables. Then the locations in the abstract state will be $\{n_Z \mid Z \subseteq PVar\}$

Not all n_Z will be present in a given abstract state

- In particular, different n_Z cannot share variables.

Abstraction

- $\alpha_s(a) = n_Z \ where \ Z = \{x \mid S(x) = a\}$
- $\alpha(h, S) \coloneqq (\bar{h}, \bar{S})$
- $\bar{h}(n_Z, f) \coloneqq \{ n_{Z'} \mid \exists \ a \in Addr \ , \ \alpha_s(a) = n_z \wedge h(a, f) = a' \wedge \alpha_s(a') = n_{Z'}\}$
- $\bar{S}(x) \coloneqq \{\alpha_s\big(S(x)\big)\}$

Partial order

- $\left(\overline{h_1}, \overline{S_1}\right) \sqsubseteq \left(\overline{h_2}, \overline{S_2}\right) \ iff \ \forall t, f \ \ \bar{h}_1(t, f) \subseteq \bar{h}_2(t, f) \wedge \forall x \ \bar{S}_1(x) \subseteq \bar{S}_2(x)$

Update

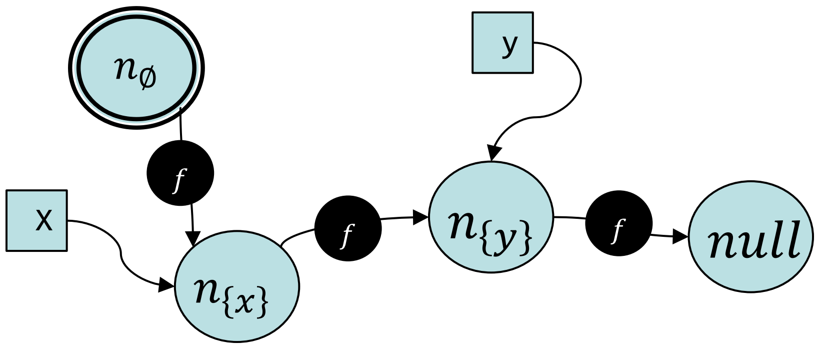
$[\![e.f = x]\!](\bar{h}, \bar{S}) = (\bar{h}', \bar{S})$

Where $\bar{h}'(n_Z, f) = \begin{cases} \bar{h}(n_Z, f) & if\ n_z \notin [\![e]\!](\bar{h}, \bar{S}) \\ \bar{S}(x) & if z \neq \emptyset \wedge\ n_z \in [\![e]\!](\bar{h}, \bar{S}) \\ \bar{h}(n_z, f) \cup \bar{S}(x) & if z = \emptyset \wedge\ n_z \in [\![e]\!](\bar{h}, \bar{S}) \end{cases}$

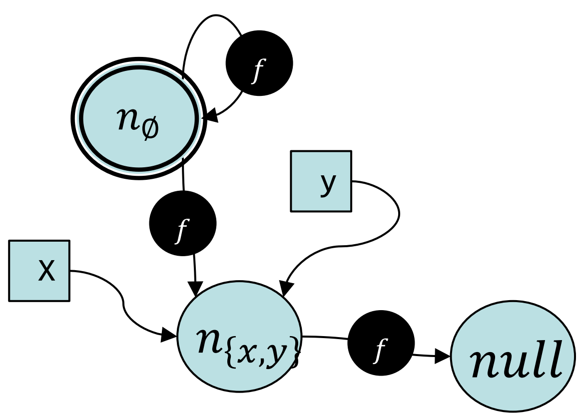
$[\![x = e]\!](\bar{h}, \bar{S}) = (\bar{h}', \bar{S}')$ (Note var update also affects heap)

- Let $[\![e]\!]\left(\bar{h}, \bar{S}\right) = \{n_{z0}, \dots, n_{zk}\}$
- $\bar{S}'(x) = \left\{n_{z0 \cup \{x\}}, \dots, n_{z0 \cup \{x\}}\right\}$
- For $y \neq x$, $\bar{S}'(y) = replace(n_{zi}, n_{zi \cup \{x\}}, \bar{S}(x))$
- How do we update $\bar{h}$?

Updating the heap



$$x = y$$



Nodes $n_{\{x\}}$ and $n_{\{y\}}$ disappear (become unreachable)

New node $n_{\{x,y\}}$ now pointed by both x and y.

The old $n_{\{x\}}$ is now represented by $n_{\emptyset}$ which acquires a self loop

Updating the heap

Let $E_s(n_W, f, n_Y) \Leftrightarrow n_Y \in \bar{h}(n_W, f)$ (resp. for $E_s{}'$)

Then after $x = e$ with $[\![e]\!]\left(\bar{h}, \bar{S}\right) = \{n_{z0}, \ldots, n_{zk}\}$

- $E_s\left(n_W, f, n_{z_i}\right) \Rightarrow E_s'\left(n_W, f, n_{z_i \cup \{x\}}\right)$
  - And if $W \neq \emptyset$ $E_s'\left(n_W, f, n_{z_i}\right)$ should now be false. Why?
- $E_s\left(n_{z_i}, f, n_W\right) \Rightarrow E_s'\left(n_{z_i \cup \{x\}}, f, n_W\right)$
  - And if $Z_i \neq \emptyset$ $E_s'\left(n_{z_i}, f, n_W\right)$ should now be false. Why?
- The old n_{zi} turned into $n_{z_i \cup \{x\}}$ so things that used to point to n_{zi} now point to $n_{z_i \cup \{x\}}$.
- Do we need to do something special when $x \in Z_i$?