Heap Abstraction

Lorenzo Gazzella, Roberto Bruni

Overview

	(Fwd) Program analysis	**Heap Manipulation**
Correctness	Hoare Logic	Separation Logic

Overview

	(Fwd) Program analysis	Heap Manipulation
Correctness	Hoare Logic	Separation Logic
Incorrectness	Incorectness Logic	Incorrectness Separation Logic

Overview

	(Fwd) Program analysis	**Heap Manipulation**
Correctness	Hoare Logic	Separation Logic
Incorrectness	Incorectness Logic	Incorrectness Separation Logic
Correctness **+** ***Incorrectness***	LCL_A	??

Language

$$
\begin{aligned}
r ::= \ & e \\
& \mid r_1 ; r_2 \\
& \mid r_1 + r_2 \\
& \mid r^*
\end{aligned}
$$

$$
\begin{aligned}
e ::= \ & skip \\
& \mid b? \\
& \mid x := a \\
& \mid x := [a] \\
& \mid [x] := a \\
& \mid x := alloc() \\
& \mid free(x)
\end{aligned}
$$

Separation Logic

$[\![r]\!]$ collecting semantics

- Hoare Logic's extension to Pointer Analysis
- Correctness: $\{P\}\ r\ \{Q\} \Rightarrow [\![r]\!]P \subseteq Q$
- Assertion Language:

$$Ast \ni p, q ::= false \mid \neg p \mid p \wedge q \mid a_1 \asymp a_2 \mid \exists x.p \qquad \text{(Pure part)}$$
$$\mid emp \mid x \mapsto a \mid p * q \qquad \text{(Spatial Part)}$$

$[\![r]\!]$ collecting semantics

Incorrectness Separation Logic

- Incorrectness Logic + Separation Logic
- Correctness and completeness: $[P]\ r\ [Q] \iff Q \subseteq [\![r]\!]P$
- Assertion language:

$$
\begin{aligned}
Ast \ni p, q ::= & false \mid \neg p \mid p \wedge q \mid a_1 \asymp a_2 \mid \exists x.p && \text{(Pure part)} \\
& \mid emp \mid x \mapsto a \mid \boldsymbol{x} \not\mapsto \mid p * q && \text{(Spatial Part)}
\end{aligned}
$$

Local Completeness Logic

$[\![r]\!]$ collecting semantics

- Correctness + Incorrectness + Abstract Interpretation
- Correctness:

$$\vdash_A [P]\ r\ [Q] \Rightarrow Q \subseteq [\![r]\!]P \subseteq A(Q)$$

- Parametric on the domain A

Heap abstraction

- Abstract domain for shape analysis
- Loosely based on the idea of [1]:
  - Divides the array into a sequence of possibly empty segments delimited by a set of segment bounds.
  - The content of each segment is uniformly abstracted
  - Fully automatic

[1] Patrick Cousot, Radhia Cousot, and Francesco Logozzo. “A parametric segmentation functor for fully automatic and scalable array content analysis”

Heap Manipulation Language for Lists

$$
\begin{aligned}
e ::= {} & skip \mid x := a \mid b? \\
& \mid x := new(e, y) && (allocation) \\
& \mid x := [y.data] \mid x := [y.next] && (lookup) \\
& \mid [x.data] := e \mid [x.next] := y && (mutation) \\
& \mid free(x) && (deallocation)
\end{aligned}
$$

Concrete Domain

- Stores: $\mathbb{S} \triangleq \mathbb{X} \to (\mathbb{Z} \cup \mathbb{L})$
- Heaps: $\mathbb{H} \triangleq \mathbb{L} \rightharpoonup (\mathbb{Z} \cup \mathbb{L} \cup \bot)$
- Memories: $\mathbb{M} \triangleq \mathbb{S} \times \mathbb{H}$
- Concrete domain: $\mathcal{P}(\mathbb{M})$

Concrete Semantics

- Regular commands: as usual
- Basic expressions:
  - Allocation:
    - $$(s, h) \xrightarrow{new(a,y)} (s[x \mapsto l], h[l \mapsto (\!|a, y|\!) s])$$
  - Mutation:
    - $$(s, h[s(x) \mapsto (v, \_)]) \xrightarrow{[x.next]:=y} (s, h[s(x) \mapsto (v, s(y))])$$

$$\vdash_A [P]\ r\ [Q] \Rightarrow Q \subseteq [\![r]\!]P \subseteq A(Q)$$

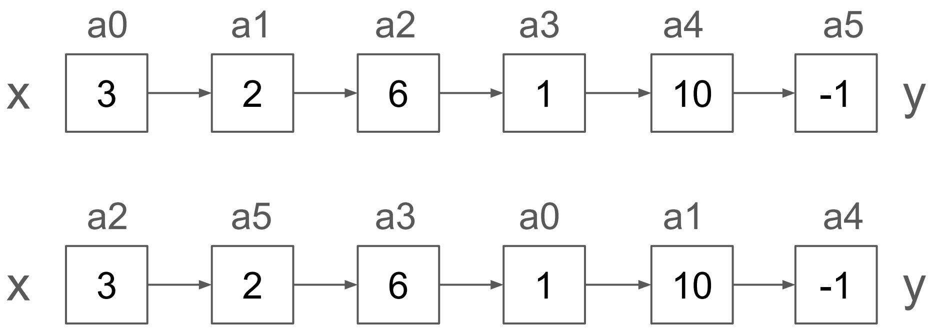
Assertion Language

- $$\begin{aligned} Ast \ni p, q ::= & \, false \mid true \mid p \wedge q \mid p \vee q \mid a_1 \asymp a_2 \mid \exists x.p && \text{(Pure part)} \\ & \mid emp \mid x \mapsto a \mid x \not\mapsto \ \mid p * q && \text{(Spatial Part)} \end{aligned}$$

- Same as ISL
- **Abstracts from locations**
- $\mathcal{P}(\mathbb{M}) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} Ast \underset{\alpha_{\mathbb{M}}}{\overset{\gamma_{\mathbb{M}}}{\leftrightarrows}} \mathbb{M}^{\#}$

Assertion Language

- $listseg(x, y, [3, 2, 6, 1, 10, -1])$ acyclic



Abstract Domain

- Stores: $\mathbb{S}^{\#} \triangleq \mathbb{V}^{\#} \times Eq^{\#}$
  - Integer part: $\mathbb{V}^{\#} \in Abs([\mathbb{X} \to \mathbb{Z}])$
  - Address part: $Eq^{\#}$

Abstract Domain

- Stores: $\mathbb{S}^\# \triangleq \mathbb{V}^\# \times Eq^\#$
  - Integer part: $\mathbb{V}^\# \in Abs([\mathbb{X} \to \mathbb{Z}])$
  - Address part: $Eq^\#$
- Heaps: $\mathbb{H}^\# \triangleq \mathbb{P}^\# \times \mathbb{C}^\#$
  - Shape predicates: $\mathbb{P}^\# \ni p^\# ::= emp \mid LS(\alpha, \beta, v^\#) \mid p_1^\# * p_2^\#$
  - Length constraints: $\mathbb{C}^\#$

Abstract Domain

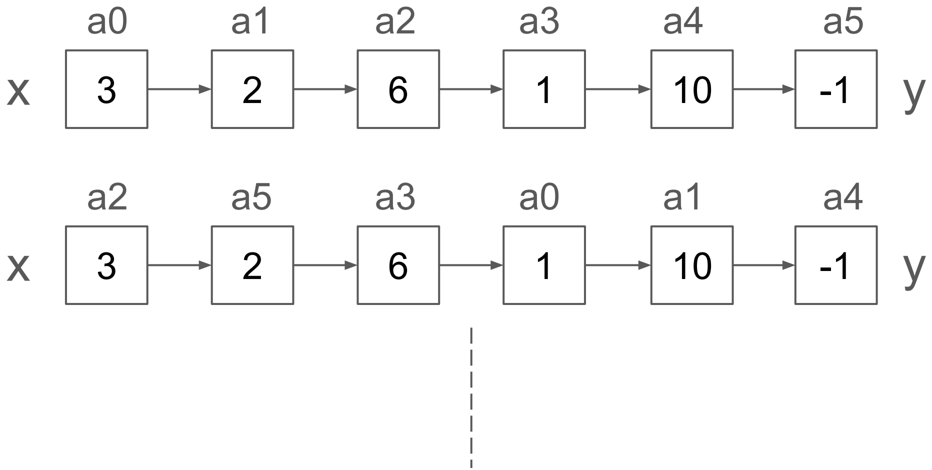
- Stores: $\mathbb{S}^{\#} \triangleq \mathbb{V}^{\#} \times Eq^{\#}$
  - Integer part: $\mathbb{V}^{\#} \in Abs([\mathbb{X} \to \mathbb{Z}])$
  - Address part: $Eq^{\#}$
- Heaps: $\mathbb{H}^{\#} \triangleq \mathbb{P}^{\#} \times \mathbb{C}^{\#}$
  - Shape predicates: $\mathbb{P}^{\#} \ni p^{\#} ::= emp \mid LS(\alpha, \beta, v^{\#}) \mid p_1^{\#} * p_2^{\#}$
  - Length constraints: $\mathbb{C}^{\#}$
    - **ls(x):** the length of the segment starting from x
    - **l(x):** the length of the list starting from x and ending in null

Abstract Domain

- Stores: $\mathbb{S}^\# \triangleq \mathbb{V}^\# \times Eq^\#$
  - Integer part: $\mathbb{V}^\# \in Abs([\mathbb{X} \to \mathbb{Z}])$
  - Address part: $Eq^\#$
- Heaps: $\mathbb{H}^\# \triangleq \mathbb{P}^\# \times \mathbb{C}^\#$
  - Shape predicates $\mathbb{P}^\# \ni p^\# ::= emp \mid LS(\alpha, \beta, v^\#) \mid p_1^\# * p_2^\#$
  - Length constraints: $\mathbb{C}^\#$
    - **ls(x):** the length of the segment starting from x
    - **l(x):** the length of the list starting from x and ending in null
- Memories: $\mathbb{M}^\# \triangleq \mathbb{S}^\# \times \mathbb{H}^\#$

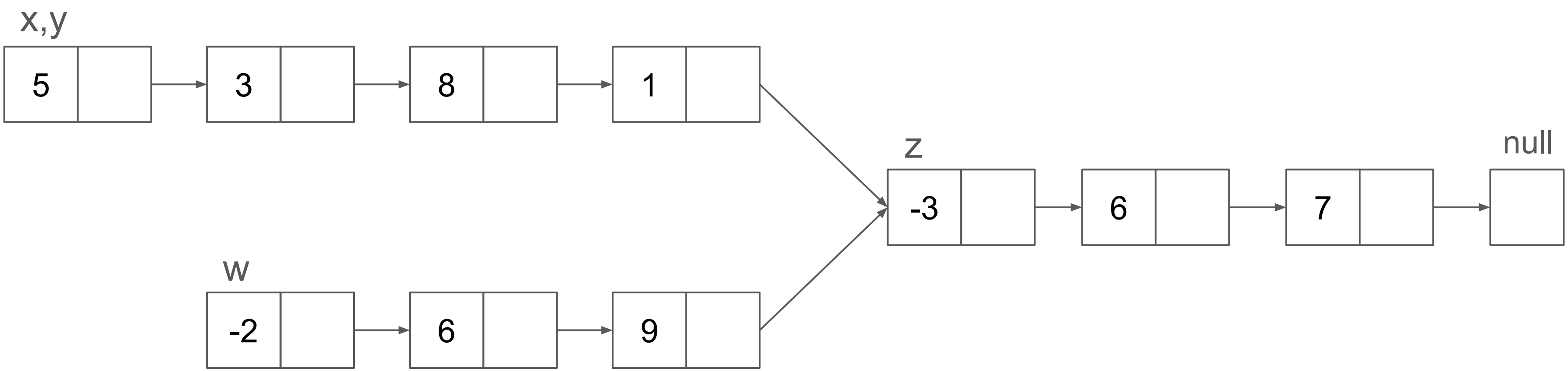
Abstraction function

$$listseg(x, y, [3, 2, 6, 1, 10, -1])$$

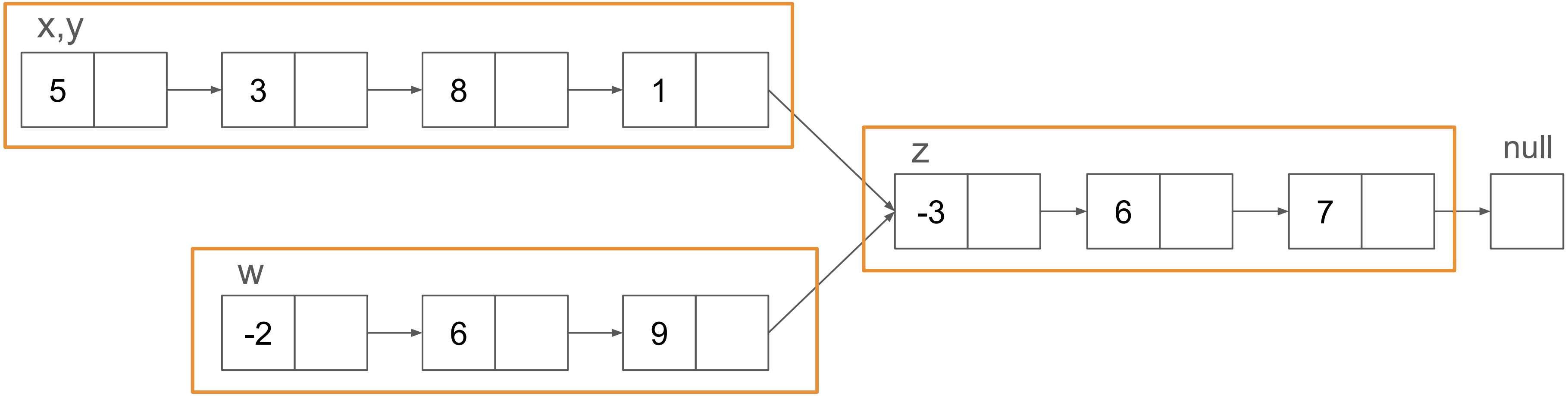


$$LS(x, y, [-1, 10])$$

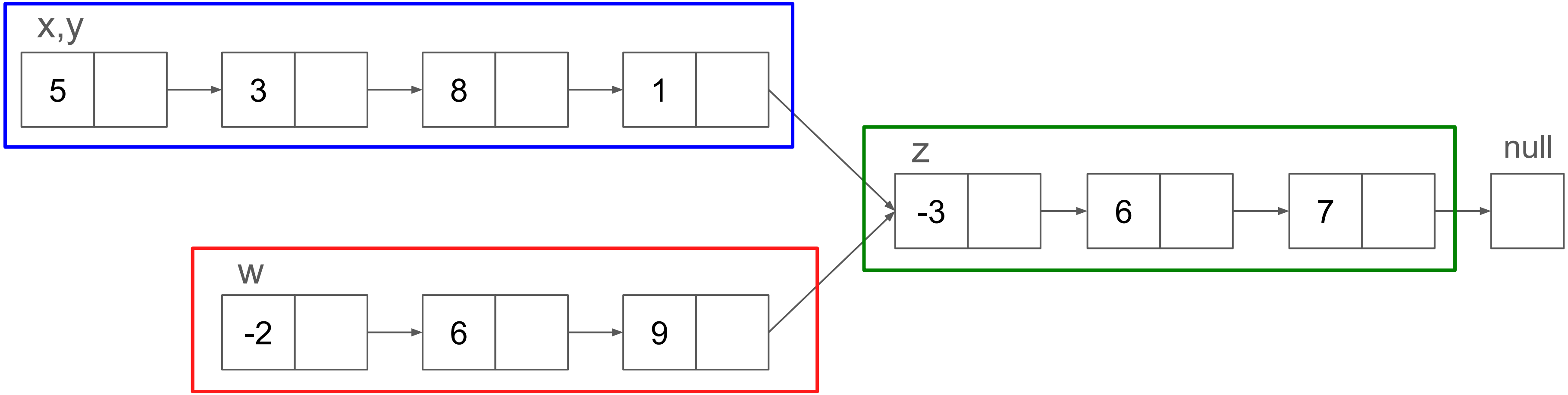
Example



Example

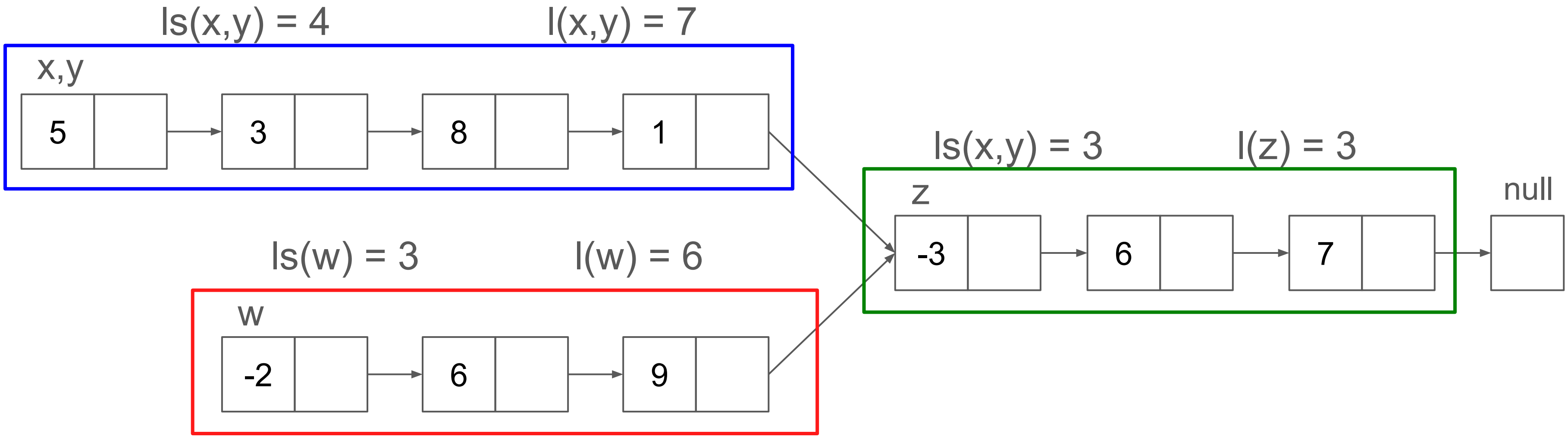


Example - shape

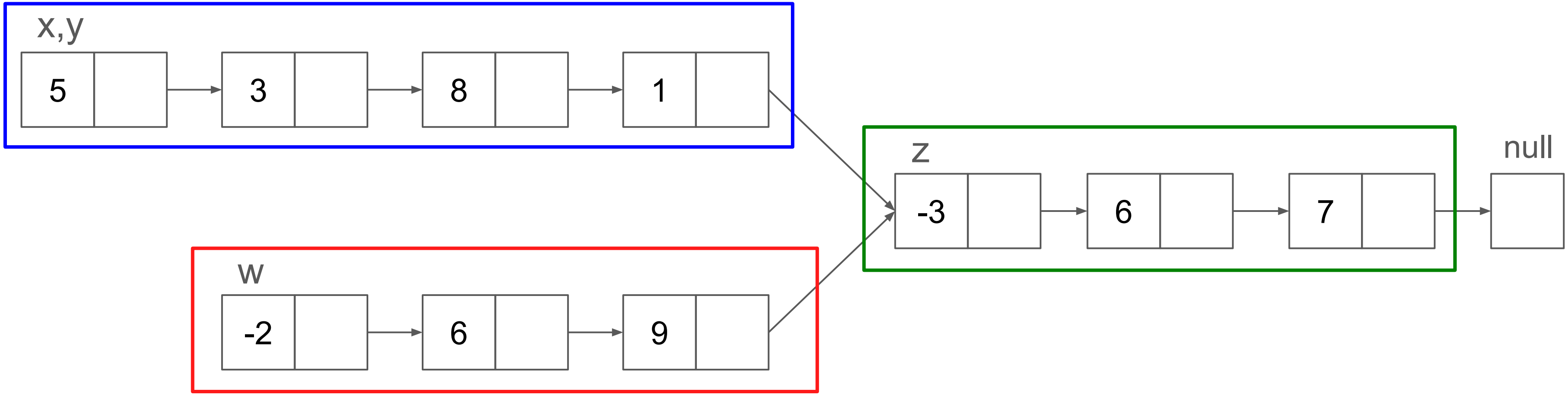


$$\color{blue}{LS(\{\hat{x},\hat{y}\},\{\hat{z}\},[1,8])} * \color{red}{LS(\{\hat{w}\},\{\hat{z}\},[-2,9])} * \color{green}{LS(\{\hat{z}\},\{null\#\},[-3,7])}$$

Example - lengths constraint

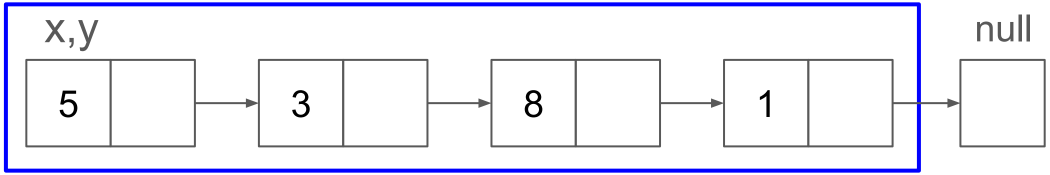


Heap-induced ordering



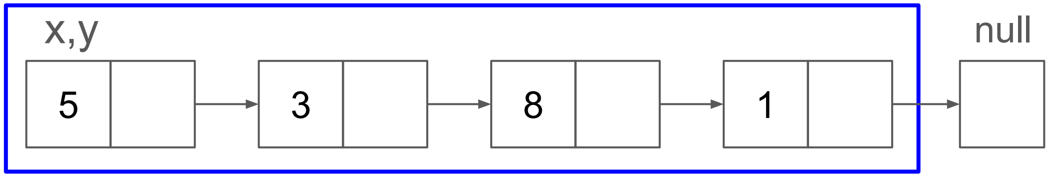
{x,y} ⩽ {z} and {w} ⩽ {z} and {z} ⩽ null

Reshaped Heap

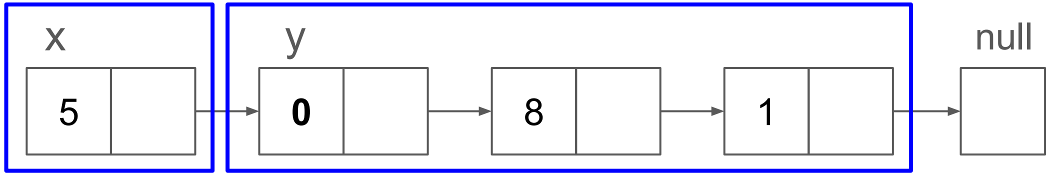


$l(x,y) = ls(x,y) = 4$

Reshaped Heap

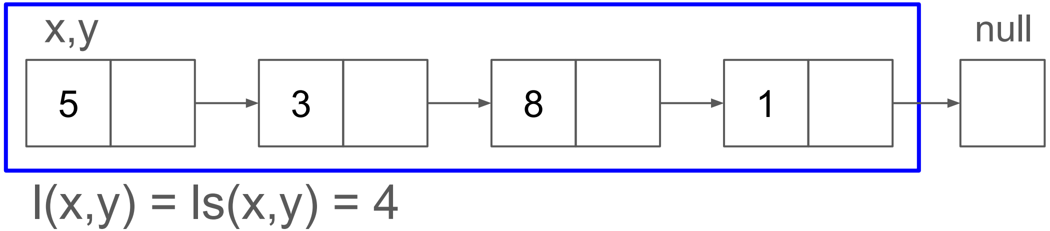


l(x,y) = ls(x,y) = 4

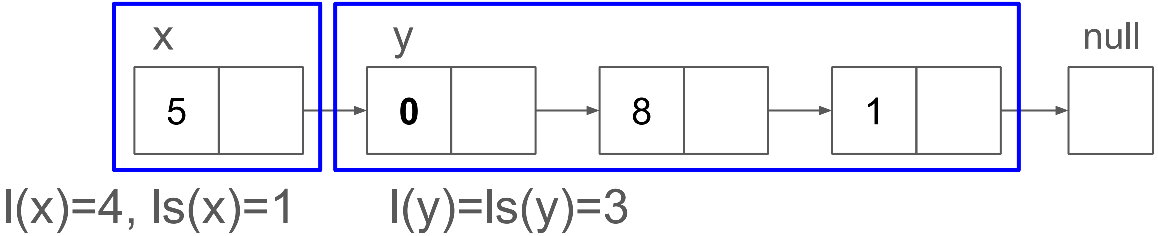


l(x)=4, ls(x)=1   l(y)=ls(y)=3

Reshaped Heap

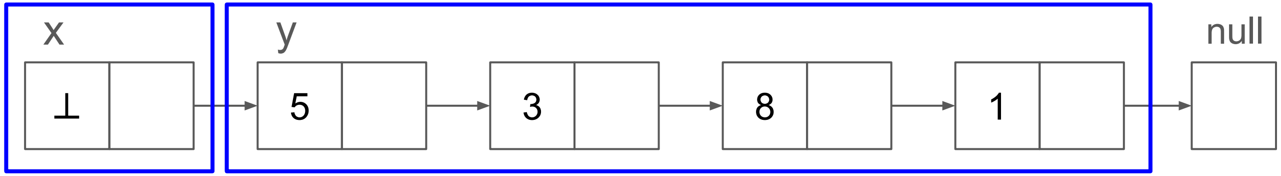


$\{x,y\} \preccurlyeq \{null\}$



$\{x\} \preccurlyeq \{y\} \preccurlyeq \{null\}$

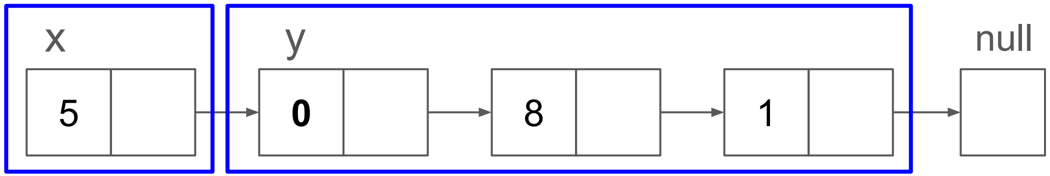
Reshaped Heap



$\{x\} \leqslant \{y\} \leqslant \{null\}$

ls(x)=0
l(x)=4

l(y)=ls(y)=4



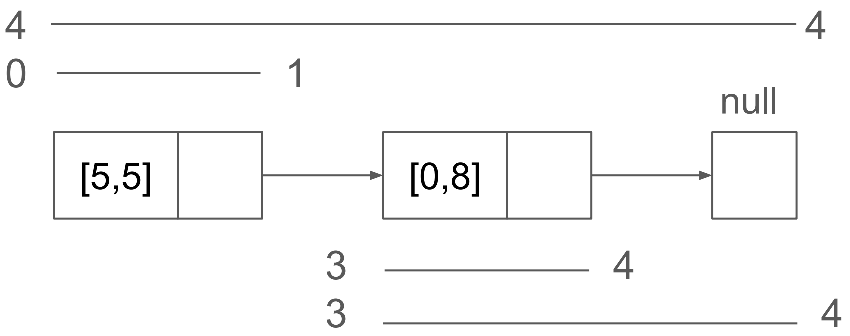
$\{x\} \leqslant \{y\} \leqslant \{null\}$

l(x)=4, ls(x)=1

l(y)=ls(y)=3

Heap Join

- $p_1^{\#} = LS(x, y, \bot) * LS(y, null, [1, 8])$
- $p_2^{\#} = LS(x, y, [5, 5]) * LS(y, null, [0, 8])$
- $p_1^{\#} \sqcup p_2^{\#} = LS(x, y, [5, 5]) * LS(y, null, [0, 8])$



Abstract Semantics

- Allocation:

$$(\!|x := new(e, y)|\!)^{\#}(p^{\#} * LS(\alpha, \beta, v_1^{\#})) =$$

$$= p^{\#} * LS(\{x\}, \alpha, v^{\#}) * LS(\alpha, \beta, v_1^{\#})$$

- $y \in \alpha$

α	$v_1^{\#}$	β

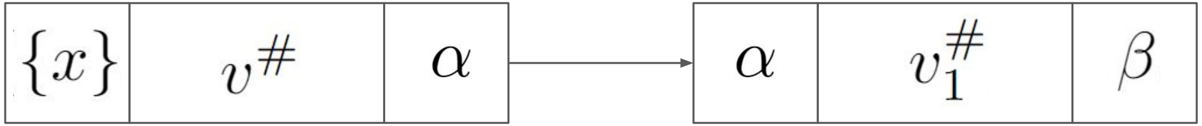
Abstract Semantics

- Allocation:

$$(\!|x := new(e, y)|\!)^{\#}(p^{\#} * LS(\alpha, \beta, v_1^{\#})) =$$

$$= p^{\#} * LS(\{x\}, \alpha, v^{\#}) * LS(\alpha, \beta, v_1^{\#})$$

- $y \in \alpha$



Abstract Semantics

- Mutation:

$$(\![x.next] := y]\!)^{\#}(p^{\#} * LS(\alpha, null^{\#}, v_1^{\#}) * LS(\delta, \beta, v_2^{\#})) =$$
$$= p^{\#} * LS(\alpha, \delta, v_1^{\#}) * LS(\delta, \beta, v_2^{\#})$$

- $x \in \alpha\,, \quad y \in \delta$

α	$v_1^{\#}$	

δ	$v_2^{\#}$	β

Abstract Semantics

- Mutation:

$$(\!|[x.next] := y|\!)^{\#}(p^{\#} * LS(\alpha, null^{\#}, v_1^{\#}) * LS(\delta, \beta, v_2^{\#})) =$$
$$= p^{\#} * LS(\alpha, \delta, v_1^{\#}) * LS(\delta, \beta, v_2^{\#})$$

- $x \in \alpha\,, \quad y \in \delta$

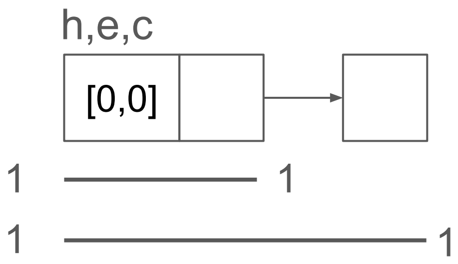
Example: Creation of a list

```
         i = 0, len = 5;
         elem := new(0, null#);
         head := elem;
         curr := elem;
(1:)     while (2:) (i<len) {
(3:)        elem := new(0, null#);
(4:)        [curr.next] := elem;
(5:)        curr := elem;
(6:)        i := i+1;
(7:)     }
```

Example: Creation of a list

h = head
e = elem
c = curr

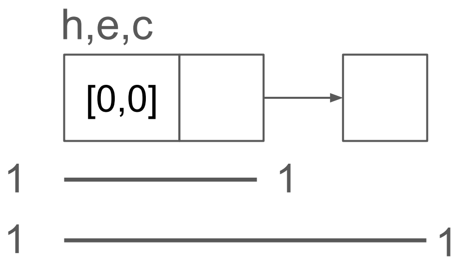
```
        i = 0, len = 5;
        elem := new(0, null#);
        head := elem;
        curr := elem;
(1:)    while (2:) (i<len) {
(3:)       elem := new(0, null#);
(4:)       [curr.next] := elem;
(5:)       curr := elem;
(6:)       i := i+1;
(7:) }
```



Example: Creation of a list

h = head
e = elem
c = curr

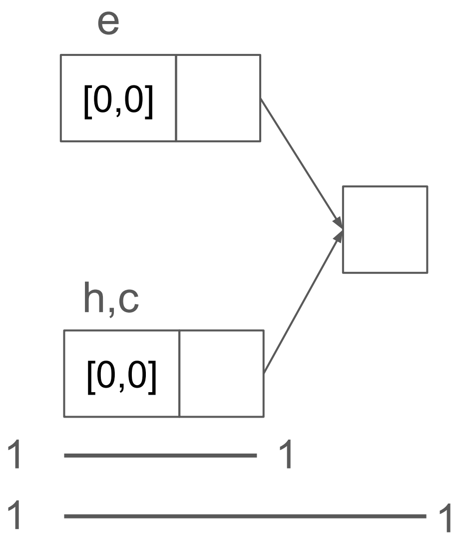
```
        i = 0, len = 5;
        elem := new(0, null#);
        head := elem;
        curr := elem;
(1:)    while (2:) (i<len) {
(3:)       elem := new(0, null#);
(4:)       [curr.next] := elem;
(5:)       curr := elem;
(6:)       i := i+1;
(7:) }
```



Example: Creation of a list

h = head
e = elem
c = curr

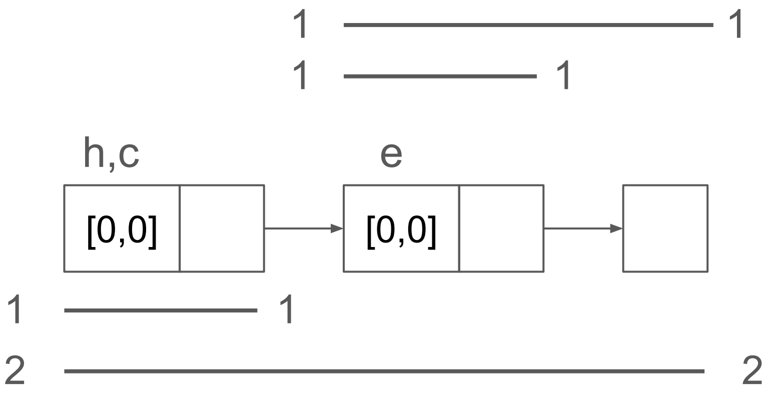
```
        i = 0, len = 5;
        elem := new(0, null#);
        head := elem;
        curr := elem;
(1:)    while (2:) (i<len) {
(3:)       elem := new(0, null#);
(4:)       [curr.next] := elem;
(5:)       curr := elem;
(6:)       i := i+1;
(7:) }
```



Example: Creation of a list

h = head
e = elem
c = curr

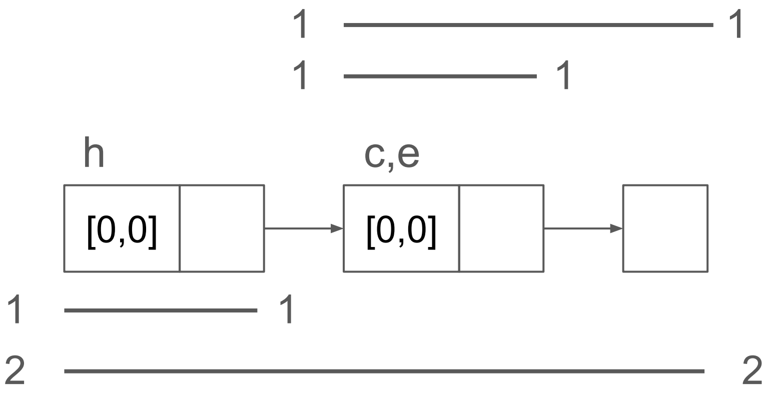
```
         i = 0, len = 5;
         elem := new(0, null#);
         head := elem;
         curr := elem;
(1:)   while (2:) (i<len) {
(3:)       elem := new(0, null#);
(4:)       [curr.next] := elem;
(5:)       curr := elem;
(6:)       i := i+1;
(7:) }
```



Example: Creation of a list

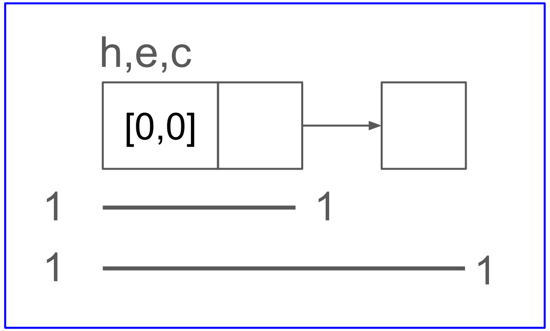
h = head
e = elem
c = curr

```
         i = 0, len = 5;
         elem := new(0, null#);
         head := elem;
         curr := elem;
(1:)   while (2:) (i<len) {
(3:)      elem := new(0, null#);
(4:)      [curr.next] := elem;
(5:)      curr := elem;
(6:)      i := i+1;
(7:) }
```

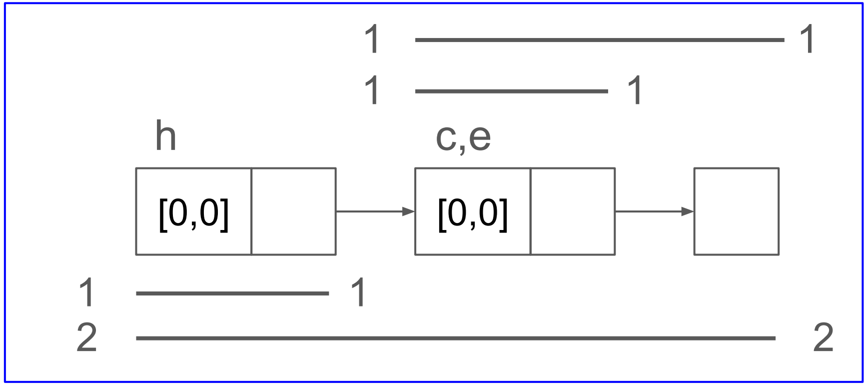


Example: Creation of a list

```
      i = 0, len = 5;
      elem := new(0, null#);
      head := elem;
      curr := elem;
(1:)  while (2:) (i<len) {
(3:)     elem := new(0, null#);
(4:)     [curr.next] := elem;
(5:)     curr := elem;
(6:)     i := i+1;
(7:) }
```



1

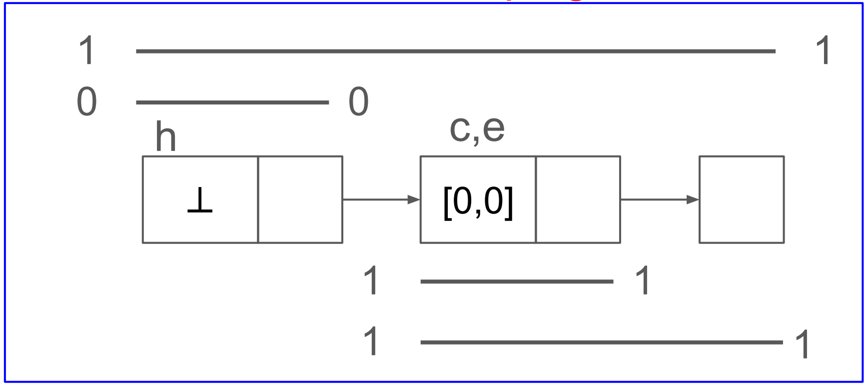


7

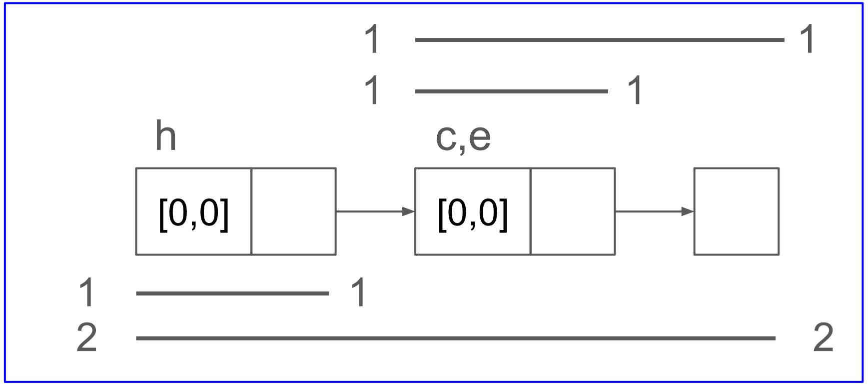
Example: Creation of a list

```
      i = 0, len = 5;
      elem := new(0, null#);
      head := elem;
      curr := elem;
(1:)  while (2:) (i<len) {
(3:)     elem := new(0, null#);
(4:)     [curr.next] := elem;
(5:)     curr := elem;
(6:)     i := i+1;
(7:) }
```

Reshaping



1

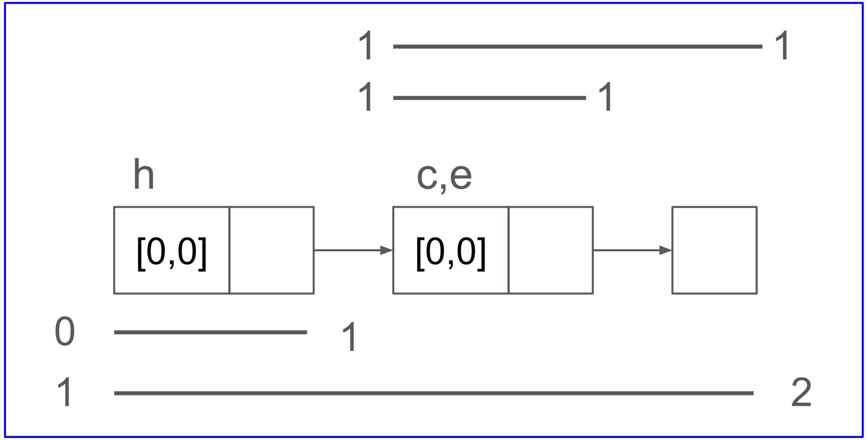


7

Example: Creation of a list

```
      i = 0, len = 5;
      elem := new(0, null#);
      head := elem;
      curr := elem;
(1:)  while (2:) (i<len) {
(3:)     elem := new(0, null#);
(4:)     [curr.next] := elem;
(5:)     curr := elem;
(6:)     i := i+1;
(7:) }
```

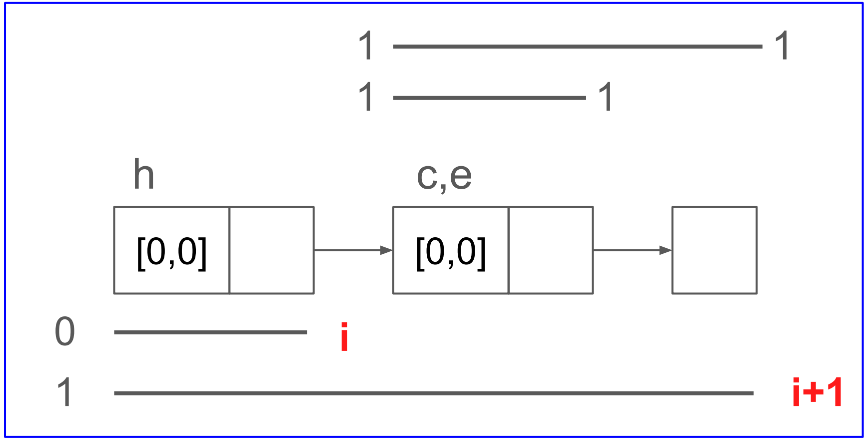
Join



Example: Creation of a list

```
        i = 0, len = 5;
        elem := new(0, null#);
        head := elem;
        curr := elem;
(1:)    while (2:) (i<len) {
(3:)       elem := new(0, null#);
(4:)       [curr.next] := elem;
(5:)       curr := elem;
(6:)       i := i+1;
(7:) }
```

Invariant



Conclusions

- Developed an abstract domain for shape analysis
- Able to automatically infer invariants on data structures like precedence graph without split
- $\alpha(p * q) = \alpha(p) * \alpha(q)$

Future work

- Local completeness of Ast w.r.t. the concrete domain
  - $\mathcal{P}(\mathbb{M}) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} Ast \underset{\alpha_{\mathbb{M}}}{\overset{\gamma_{\mathbb{M}}}{\leftrightarrows}} \mathbb{M}^{\#}$
  - $x \not\mapsto$
  - $x := new(e, y) \mid x := [y.field] \mid [x.data] := e \mid [x.next] := y \mid free(x)$
- Extending the assertion language to negation and universal quantification
- Extending the proof system of LCL with rules for heap manipulation
  - Frame rule soundness

Thank you for your attention!