Principles of Program Analysis:

Data Flow Analysis

Transparencies based on Chapter 2 of the book: Flemming Nielson, Hanne Riis Nielson and Chris Hankin: Principles of Program Analysis. Springer Verlag 2005.

Shape Analysis

Goal: to obtain a finite representation of the shape of the heap of a language with pointers.

The analysis result can be used for

- detection of pointer aliasing
- detection of sharing between structures
- software development tools
  - detection of errors like dereferences of `nil`-pointers
- program verification
  - `reverse` transforms a non-cyclic list to a non-cyclic list

Syntax of the pointer language

$$a ::= p \mid n \mid a_1 \; op_a \; a_2 \mid \texttt{nil}$$

$$p ::= x \mid x.sel$$

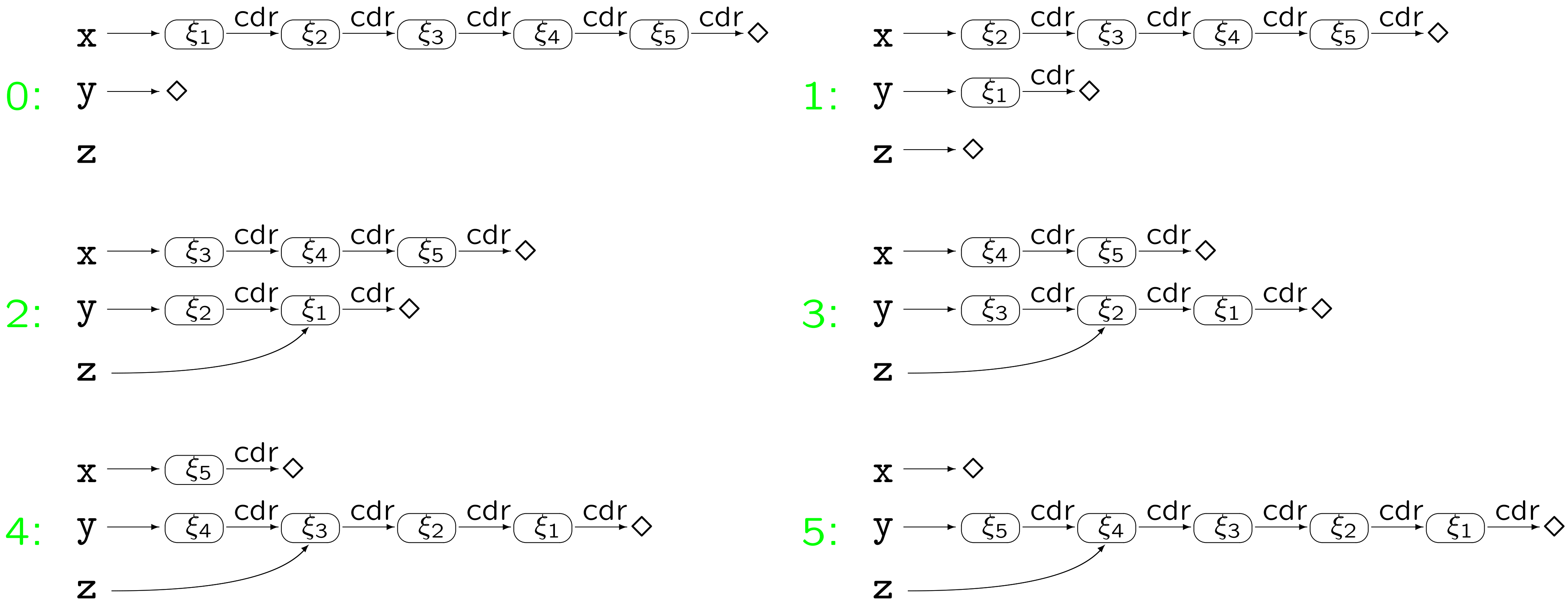
$$b ::= \texttt{true} \mid \texttt{false} \mid \texttt{not} \; b \mid b_1 \; op_b \; b_2 \mid a_1 \; op_r \; a_2 \mid op_p \; p$$

$$\begin{aligned} S ::= \; & [p\texttt{:=}a]^\ell \mid [\texttt{skip}]^\ell \mid S_1;\; S_2 \mid \\ & \texttt{if} \; [b]^\ell \; \texttt{then} \; S_1 \; \texttt{else} \; S_2 \mid \texttt{while} \; [b]^\ell \; \texttt{do} \; S \mid \\ & [\texttt{malloc} \; p]^\ell \end{aligned}$$

Example

$[\texttt{y:=nil}]^1;$
$\texttt{while} \; [\texttt{not is-nil(x)}]^2 \; \texttt{do}$
$\quad ([\texttt{z:=y}]^3; [\texttt{y:=x}]^4; [\texttt{x:=x.cdr}]^5; [\texttt{y.cdr:=z}]^6);$
$[\texttt{z:=nil}]^7$

Reversal of a list



Structural Operational Semantics

A configurations consists of

- a state $\sigma \in \mathbf{State} = \mathbf{Var}_\star \to (\mathbf{Z} + \mathbf{Loc} + \{\diamond\})$

  mapping variables to values, locations (in the heap) or the nil-value

- a heap $\mathcal{H} \in \mathbf{Heap} = (\mathbf{Loc} \times \mathbf{Sel}) \to_{\mathsf{fin}} (\mathbf{Z} + \mathbf{Loc} + \{\diamond\})$

  mapping pairs of locations and selectors to values, locations in the heap or the nil-value

Pointer expressions

$$\wp \ : \ \mathbf{PExp} \to (\mathbf{State} \times \mathbf{Heap}) \to_{\mathsf{fin}} (\mathbf{Z} + \{\diamond\} + \mathbf{Loc})$$

is defined by

$$\wp[\![x]\!](\sigma, \mathcal{H}) \ = \ \sigma(x)$$

$$\wp[\![x.\mathit{sel}]\!](\sigma, \mathcal{H}) \ = \ \begin{cases} \mathcal{H}(\sigma(x), \mathit{sel}) \\ \qquad\qquad \text{if } \sigma(x) \in \mathbf{Loc} \text{ and } \mathcal{H} \text{ is defined on } (\sigma(x), \mathit{sel}) \\ \text{undefined} \qquad \text{otherwise} \end{cases}$$

Arithmetic and boolean expressions

$$\mathcal{A} \ : \ \mathbf{AExp} \to (\mathbf{State} \times \mathbf{Heap}) \to_{\mathsf{fin}} (\mathbf{Z} + \mathbf{Loc} + \{\diamond\})$$

$$\mathcal{B} \ : \ \mathbf{BExp} \to (\mathbf{State} \times \mathbf{Heap}) \to_{\mathsf{fin}} \mathbf{T}$$

Statements

Clauses for assignments:

$$\langle [x\texttt{:=}a]^\ell, \sigma, \mathcal{H}\rangle \rightarrow \langle \sigma[x \mapsto \mathcal{A}[\![a]\!](\sigma,\mathcal{H})], \mathcal{H}\rangle$$

if $\mathcal{A}[\![a]\!](\sigma,\mathcal{H})$ is defined

$$\langle [x.\mathit{sel}\texttt{:=}a]^\ell, \sigma, \mathcal{H}\rangle \rightarrow \langle \sigma, \mathcal{H}[(\sigma(x), \mathit{sel}) \mapsto \mathcal{A}[\![a]\!](\sigma,\mathcal{H})]\rangle$$

if $\sigma(x) \in \mathbf{Loc}$ and $\mathcal{A}[\![a]\!](\sigma,\mathcal{H})$ is defined

Clauses for malloc:

$$\langle [\texttt{malloc}\ x]^\ell, \sigma, \mathcal{H}\rangle \rightarrow \langle \sigma[x \mapsto \xi], \mathcal{H}\rangle$$

where ξ does not occur in σ or $\mathcal{H}$

$$\langle [\texttt{malloc}\ (x.\mathit{sel})]^\ell, \sigma, \mathcal{H}\rangle \rightarrow \langle \sigma, \mathcal{H}[(\sigma(x), \mathit{sel}) \mapsto \xi]\rangle$$

where ξ does not occur in σ or $\mathcal{H}$ and $\sigma(x) \in \mathbf{Loc}$

Shape graphs

The analysis will operate on *shape graphs* (S, H, is) consisting of

- an abstract state, S,
- an abstract heap, H, and
- sharing information, is, for the abstract locations.

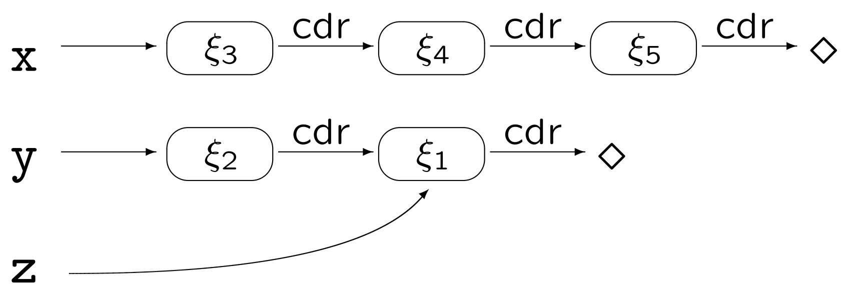
The nodes of the shape graphs are abstract locations:

$$\mathbf{ALoc} = \{n_X \mid X \subseteq \mathbf{Var}_\star\}$$

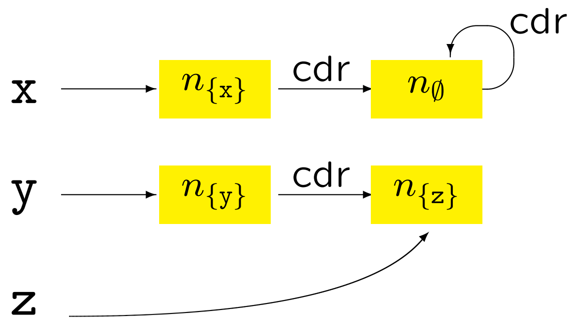
Note: there will only be *finitely* many abstract locations

Example

In the semantics:



In the analysis:



Abstract Locations

The abstract location n_X represents the location $\sigma(x)$ if $x \in X$

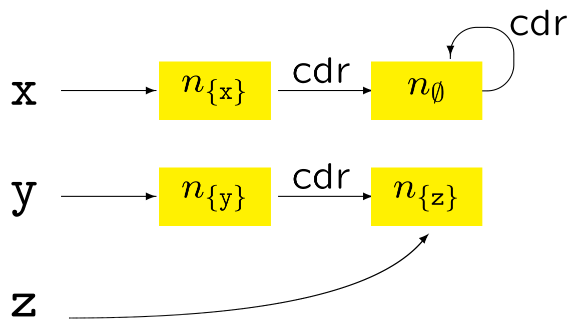
The abstract location $n_\emptyset$ is called the *abstract summary location*: $n_\emptyset$ represents all the locations that cannot be reached directly from the state without consulting the heap

Invariant 1 If two abstract locations n_X and n_Y occur in the same shape graph then either $X = Y$ or $X \cap Y = \emptyset$

Abstract states and heaps

$$\mathsf{S} \in \mathbf{AState} = \mathcal{P}(\mathbf{Var}_\star \times \mathbf{ALoc}) \quad \text{abstract states}$$

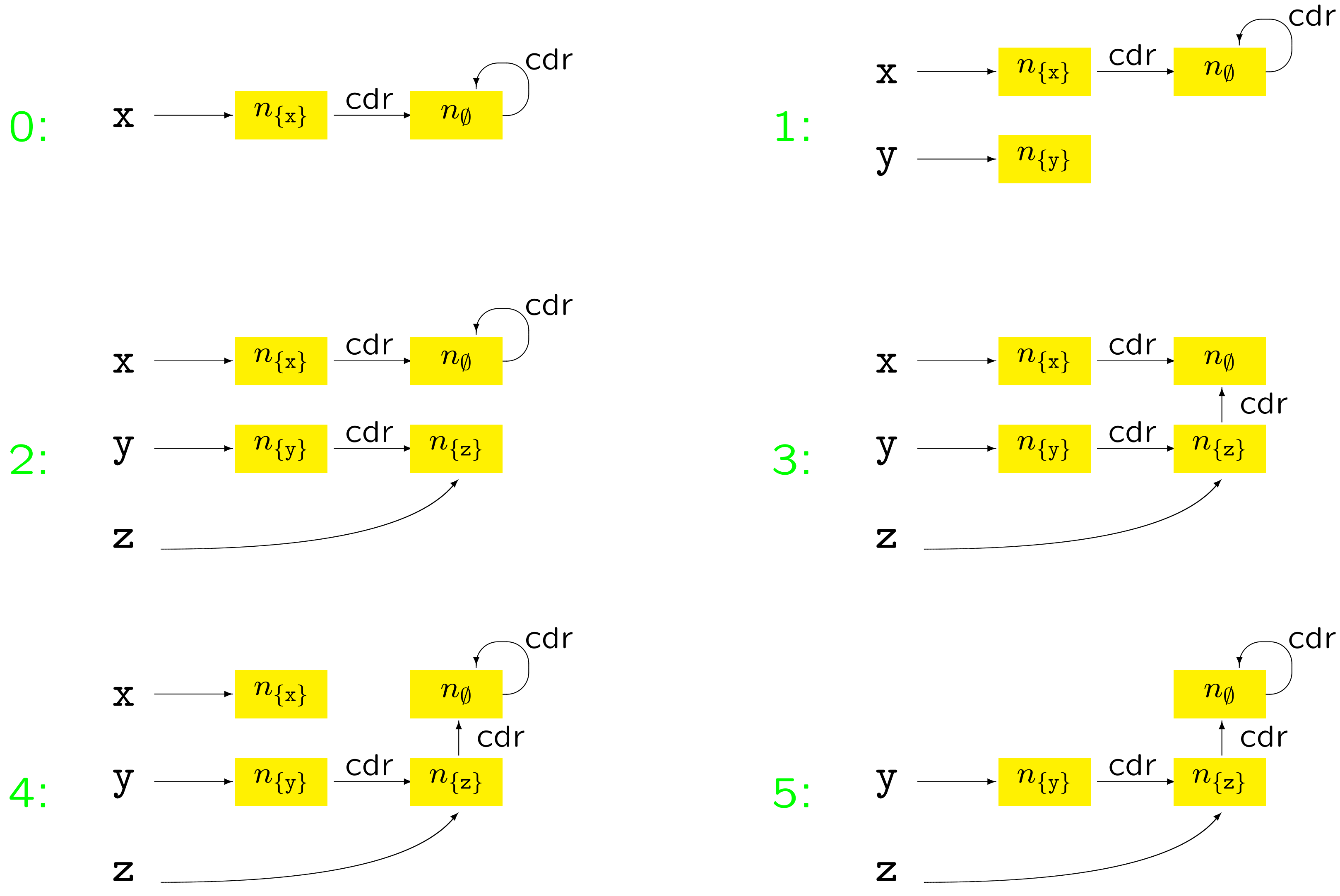
$$\mathsf{H} \in \mathbf{AHeap} = \mathcal{P}(\mathbf{ALoc} \times \mathbf{Sel} \times \mathbf{ALoc}) \quad \text{abstract heap}$$



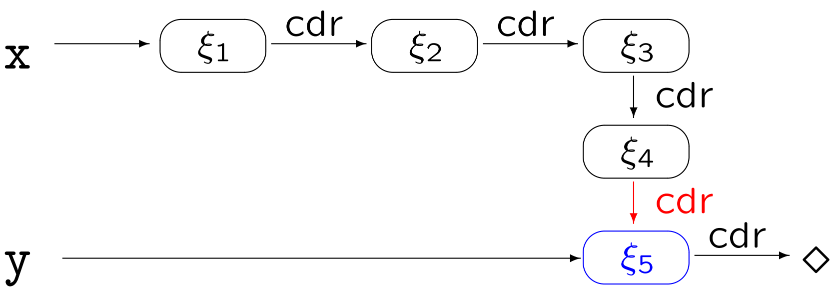
Invariant 2 If x is mapped to n_X by the abstract state S then $x \in X$

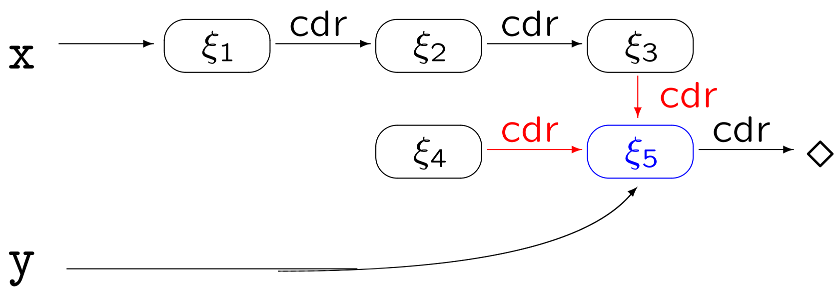
Invariant 3 Whenever (n_V, sel, n_W) and $(n_V, sel, n_{W'})$ are in the abstract heap H then either $V = \emptyset$ or $W = W'$

Reversal of a list

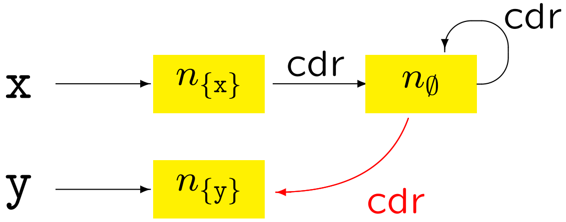


Sharing in the heap





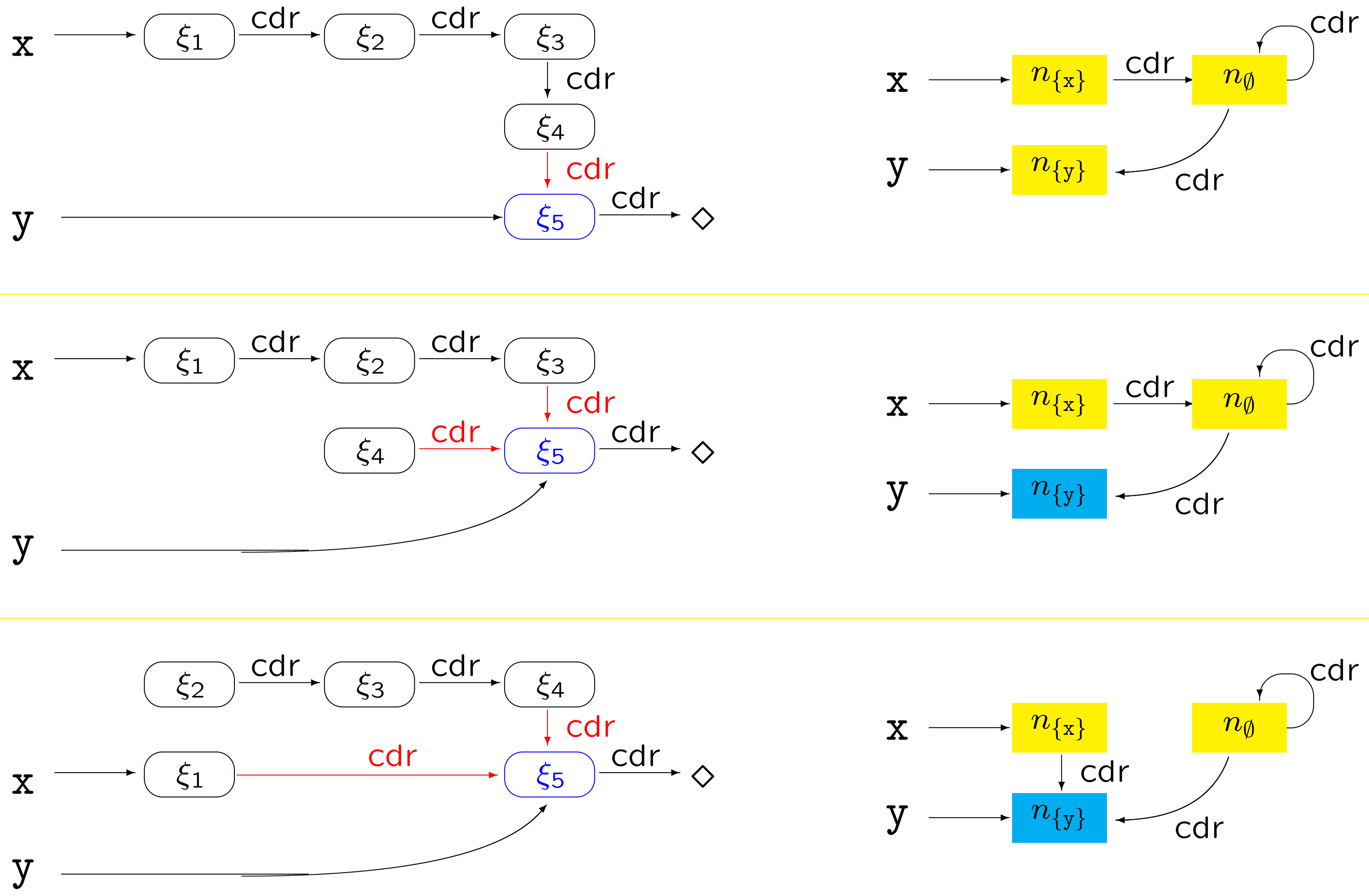
Give rise to the same shape graph:



is: the abstract locations that *might* be shared due to pointers in the heap:

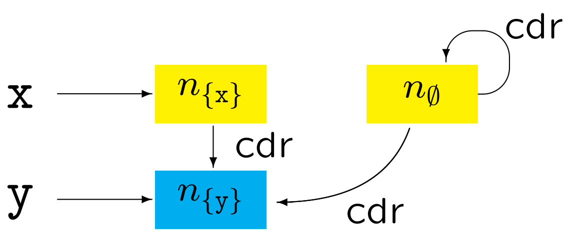
n_X is included in is if it might represents a location that is the target of more than one pointer in the heap

Examples: sharing in the heap



Sharing information

The implicit sharing information of the abstract heap must be consistent with the explicit sharing information:



Invariant 4 If $n_X \in$ is then either

- $(n_{\emptyset}, sel, n_X)$ is in the abstract heap for some *sel*, or
- there are two distinct triples (n_V, sel_1, n_X) and (n_W, sel_2, n_X) in the abstract heap

Invariant 5 Whenever there are two distinct triples (n_V, sel_1, n_X) and (n_W, sel_2, n_X) in the abstract heap and $X \neq \emptyset$ then $n_X \in$ is