Shape Analysis

Alon Milchgrub

Overview

- Lisp review
- The concrete semantics
- The abstractions function
- The abstract semantics
- Discussion

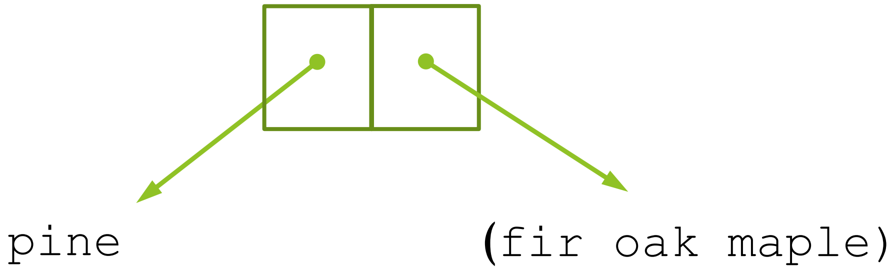
Lisp review

- In Lisp everything is a list
- The command `cons` concatenates two objects by creating a new object with pointers to both the original ones.
- The commands `car` and `cdr` are used to access the first and second elements respectively. e.g.

`(cons 'pine '(fir oak maple))` returns `(pine fir oak maple)`

`(car ‘(pine fir oak maple))` returns `pine`

`(cdr ‘(pine fir oak maple))` returns `(fir oak maple)`



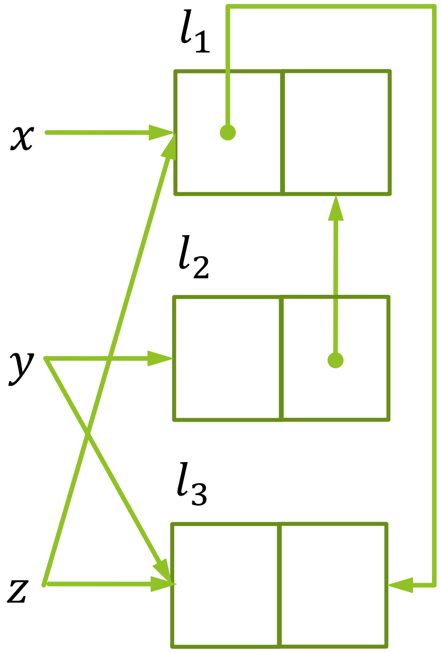
Preliminaries

- Let *PVar* be the set of pointers in a program.
- A *shape graph* if a directed graph with two type of edges: *variable-edges* $\mathrm{E_v}$ and *selector-edges* $\mathrm{E_s}$.
- $\mathrm{E_v}$ is a set of pairs of the form $[\mathrm{x}, \mathrm{n}]$ where $\mathrm{x} \in \mathrm{PVar}$ and n is a *shape-node*.
- E_s is a set of triplets of the form $\langle s, sel, t \rangle$ where $sel \in \{car, cdr\}$ and s and t are shape nodes.

- A shape graph is deterministic if from every PVar exit at most one edge and from every shape-node exit at most one edge of each of $\{car, cdr\}$.

The Concrete Semantics

- $x := \boldsymbol{new}$
- $y := \boldsymbol{new}$
- $y.cdr := x$
- $z := \mathbf{new}$
- $x.car := z$
- $y := \boldsymbol{nil}$
- $y := x.car$
- $z := \boldsymbol{nil}$
- $z := x$
- $gc(SG)$



The Concrete Semantics

- The transformations applied to the shape graph are defined by the **concrete semantics** $[\![st]\!]_{\mathcal{DSG}}: \mathcal{DSG} \rightarrow \mathcal{DSG}$.

- Let v be a control flow graph vertex and $pathsTo(v)$ the set of paths in the control flow graph from *start* to predecessors of v
- Then the **collecting semantics** is defined as follows:

$$cs(v) = \left\{[\![st(v_k)]\!]_{\mathcal{DSG}}\left(\dots\left([\![st(v_1)]\!]_{\mathcal{DSG}}(\langle\emptyset,\emptyset\rangle)\right)\right) \middle| [v_1,\dots,v_k] \in pathsTo(v)\right\}$$

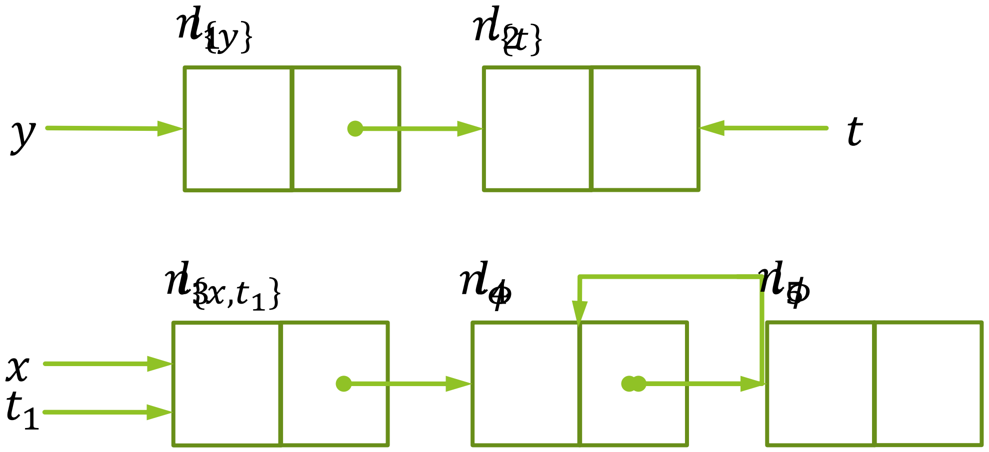
- This is the set of possible shape graphs at v.

The Abstract Semantics

- A static shape graph (SSG) is a pair $\langle SG, is\_shared \rangle$, where
- SG is a shape graph, whose shape nodes are a subset of $\{n_X | X \subseteq PVar\}$.
- is_shared is a function for the shape nodes of SG to $\{true, false\}$.
  - Semantically, $is\_shared(n) = true$ indicates that n is pointed to by more than 1 pointer **on the heap.**

The Abstract Semantics

- Given a DSG, the mapping $\hat{\alpha}$ generates a SSG by replacing the concrete locations by the set of pointers pointing to the same location (after gc).



- For the image of $\hat{\alpha}(DSG)$ $is\_shared(n_Z) = true \Leftrightarrow n_z$ represents a concrete location that is pointed by more than 1 pointer **on the heap.**

The Abstract Semantics

- For a set of shape graphs S the abstraction function α is defined as follows:

$$\alpha(S) = \bigsqcup_{DSG \in S} \hat{\alpha}(SDG)$$

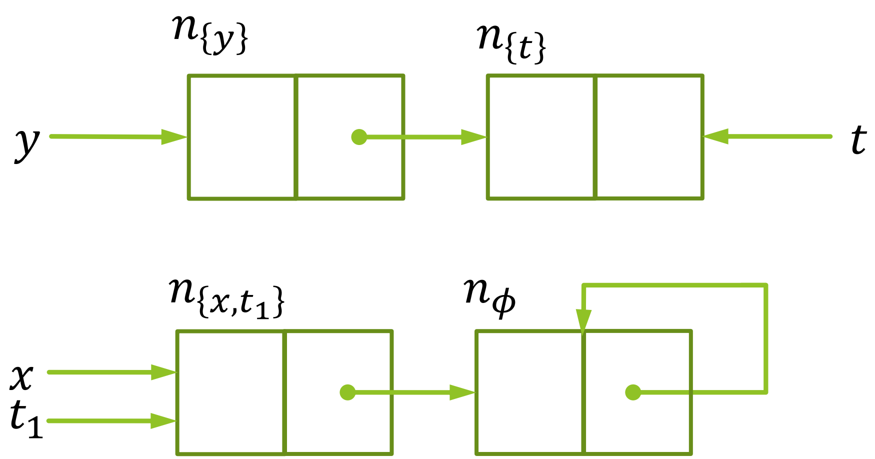
- Where for two SSGs SG and SG':

$$SG \sqcup SG' = \langle \langle E_v \cup E_v', E_s \cup E_s' \rangle, is\_shared \vee is\_shared' \rangle$$

The Abstract Semantics

- For a single DSG the shape-nodes of $\hat{\alpha}(DSG)$ represent disjoint sets of points.
- Let S be a set of DSGs, and $\alpha(S) = \langle\langle E_v, E_s\rangle, is\_shared\rangle$, then it follow that:

  For all $\langle n_X, sel, n_Y\rangle \in E_s$ either $X = Y$ or $X \cap Y = \emptyset$



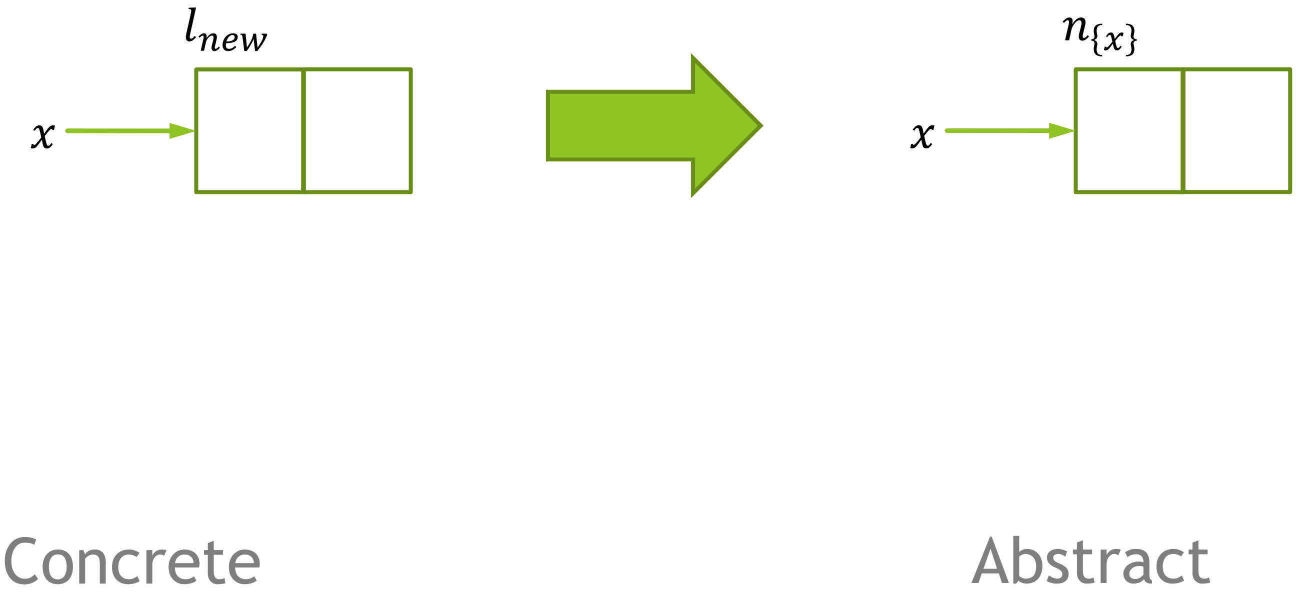
The Abstract Semantics

- In order for the abstraction to be useful, one should be able to compute it directly by transforming the static shape graph (in contrast to by abstracting the concrete shape graph).

- For this purpose the SSG meaning function $[\![st]\!]_{\mathcal{SSG}}: \mathcal{SSG} \rightarrow \mathcal{SSG}$ is defined.

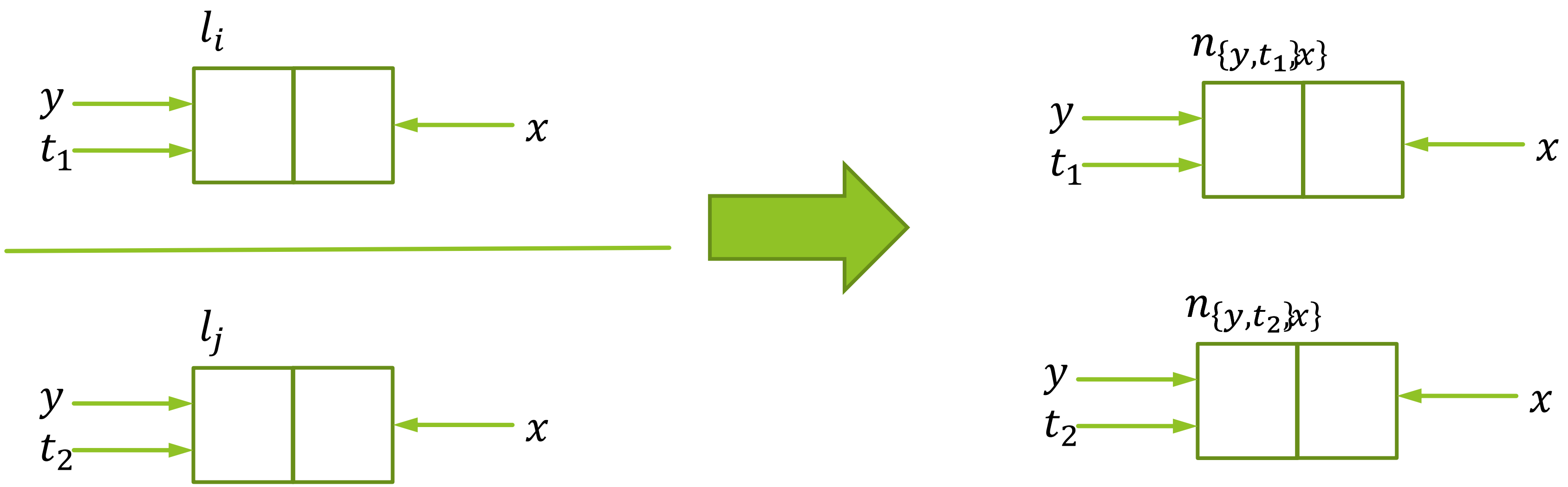
The Abstract Semantics

- $x := \boldsymbol{new}$



The Abstract Semantics

- $x \coloneqq y$

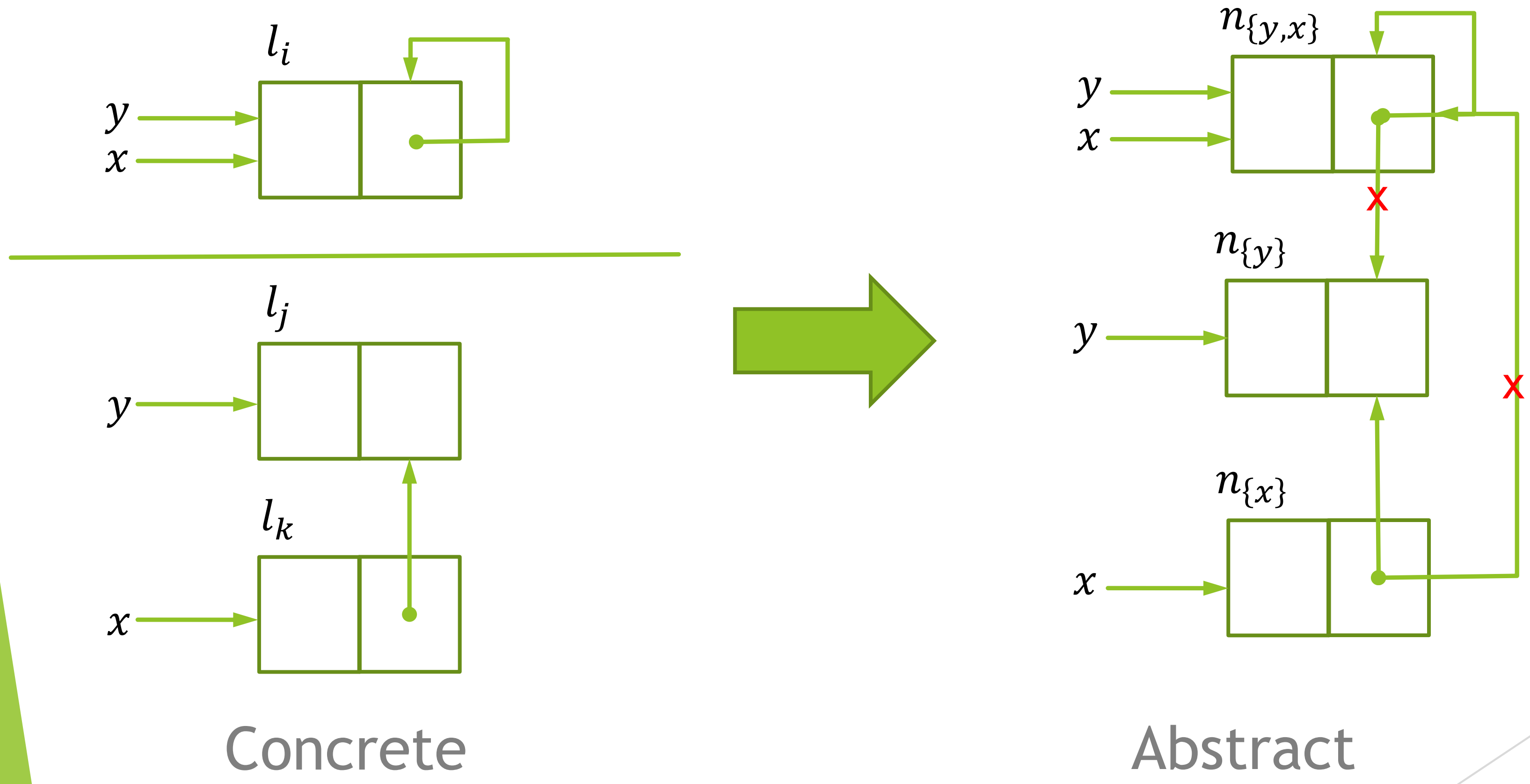


Concrete

Abstract

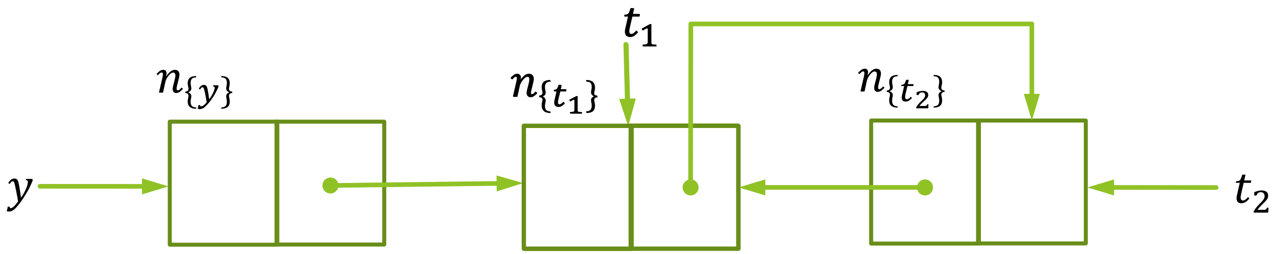
The Abstract Semantics

- $x.cdr := y$



The Abstract Semantics

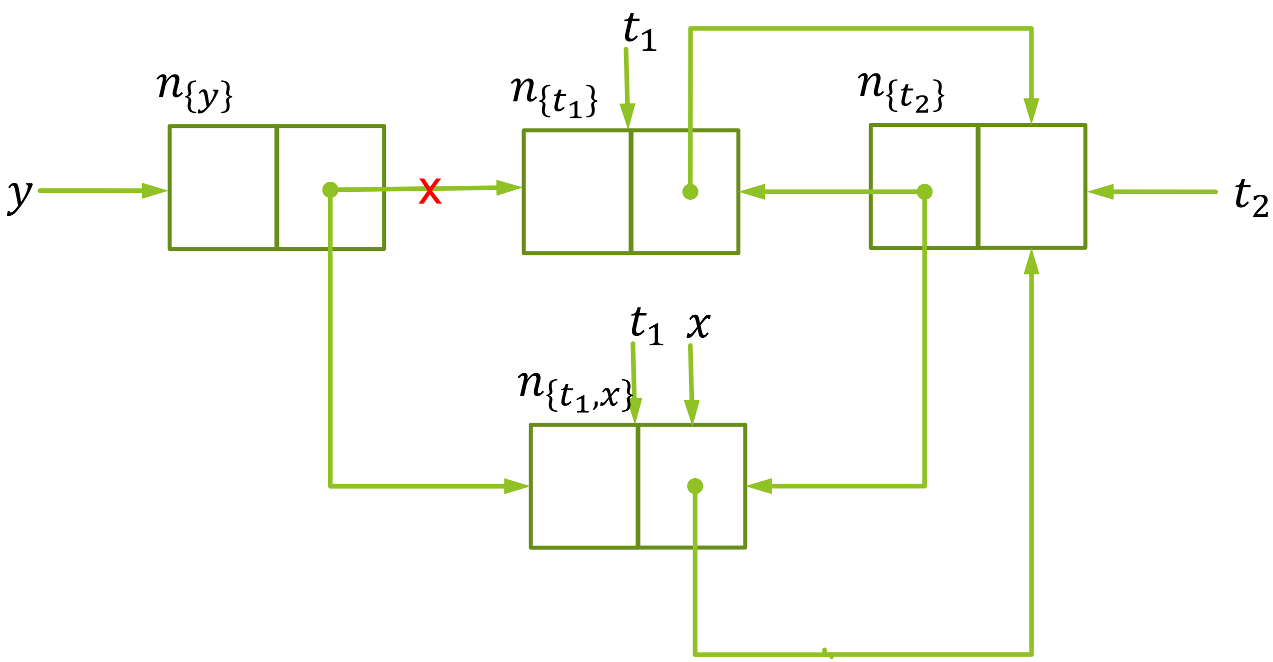
- $x := y.cdr$



Abstract

The Abstract Semantics

- $x := y.cdr$



Abstract

The Abstract Semantics

- The abstract semantics associate a SSG, SG_v, with every control-flow vertex v, defined by:

$$SG_v = \begin{cases} \langle\langle\emptyset,\emptyset\rangle, \lambda n. false\rangle & if\ v = start \\ \bigsqcup_{u \in pred(v)} [\![st(u)]\!]_{SSG}(SG_u) & otherwise \end{cases}$$

- **Theorem (Correctness):**
- For every control-flow graph vertex v:

$$\alpha\big(cs(v)\big) \subseteq SG_v$$

Properties and Achievements

- *"Strong Nullification"* – When processing a statement of the type $x.sel_0 = y$ the sel_0 edges currently emanating from x are always removed.

- *Materialization* – When processing a statement of the type $x = y.sel_0$ the algorithm creates a copy of $y.sel_0$ and thus is able to un-summarize shape-nodes.

- The shape analysis algorithm presented is able to verify shape preservation properties of data structures like lists, lists containing a cycle and trees.

Discussion

- What are possible uses of this kind of analysis?
- What are possible extensions of this method?
- What are possible flaws of this method?
  - Is it scalable?

WHO'S AWESOME?

You're awesome!