Shape Analysis

Alon Milchgrub
Overview

- Lisp review
- The concrete semantics
- The abstractions function
- The abstract semantics
- Discussion
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 $E_v$ and selector-edges $E_s$.
- $E_v$ is a set of pairs of the form $[x, n]$ where $x \in PVar$ and $n$ is a shape-node.
- $E_s$ is a set of triplets of the form $\langle s, sel, t \rangle$ where $sel \in \{\text{car}, \text{cdr}\}$ 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 $\{\text{car}, \text{cdr}\}$.
The Concrete Semantics

- $x := \text{new}$
- $y := \text{new}$
- $y.cdr := x$
- $z := \text{new}$
- $x.car := z$
- $y := \text{nil}$
- $y := x.car$
- $z := \text{nil}$
- $z := x$
- $gc(SG)$
The Concrete Semantics

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

- 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:

\[
    cs(v) = \left\{ [st(v_k)]_{DSG} \left( \ldots \left( [st(v_1)]_{DSG}(\emptyset, \emptyset) \right) \right) \left| v_1, \ldots, v_k \in pathsTo(v) \right\}
\]

- This is the set of possible shape graphs at \(v\).
The Abstract Semantics

- A static shape graph (SSG) is a pair \(SG, is\_shared\), 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.
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 $\iff n_z$ 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 $\alpha$ is defined as follows:

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

- Where for two SSGs $SG$ and $SG'$:

$$SG \sqcup SG' = \langle \{E_v \cup E'_v, E_s \cup E'_s\}, is\_shared \lor is\_shared' \rangle$$
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 $\alpha(S) = \langle \{E_v, E_s\}, is\_shared \rangle$, 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 \([st]_{SSG} : SS\rightarrow SS\) is defined.
The Abstract Semantics

- $x := new$

Concrete  

Abstract
The Abstract Semantics

- $x := y$

Concrete

Abstract
The Abstract Semantics

- $x.cdr := y$

Concrete

Abstract
The Abstract Semantics

- $x := y \cdot cdr$

Abstract

Diagram showing the relationship between $y$, $n_{(y)}$, $n_{(t_1)}$, $n_{(t_2)}$, $t_1$, and $t_2$. The diagram illustrates the flow and interaction of these elements, possibly representing a computational or logical process.
The Abstract Semantics

- $x := y \cdot cdr$

![Diagram of the abstract semantics for $x := y \cdot cdr$.]
The Abstract Semantics

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

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

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

$$\alpha(cs(v)) \subseteq SG_v$$
“Strong Nullification” - When processing a statement of the type $x \cdot sel_0 = y$ the $sel_0$ edges currently emanating from $x$ are always removed.

Materialization - When processing a statement of the type $x = y \cdot sel_0$ the algorithm creates a copy of $y \cdot sel_0$ 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!