A Local Shape Analysis based on Separation Logic

Authors: Dino Distefano, Peter W. O'Heran and Hongseok Yang
Presentor: Xie Li

June 1, 2021

Introduction

A **Local Shape Analysis** based on **Separation Logic**

- What is **Shape Analysis**?
- What is **Separation Logic**?
- How the analysis works?
- What does **Local** means?

Shape Analysis

- Questions in heap content: NULL-pointers, May-Alias, Must-Alias, Reachability, Disjointness, **Shape**.
- **Shapes** characterize data structures: singly linked list, linked list with cycle, doubly linked list, a binary tree...
- According to [1], shape analysis computes for each point in the program:
  A finite, conservative representation of the heap-allocated data structures that could arise when a path to this program point executed.

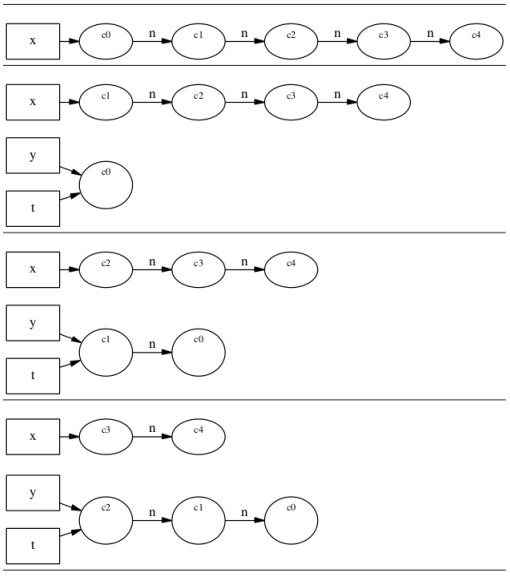
[1] Reinhard Wilheim et al. Shape Analysis. CC 2000.

Shape Analysis: Example

Example

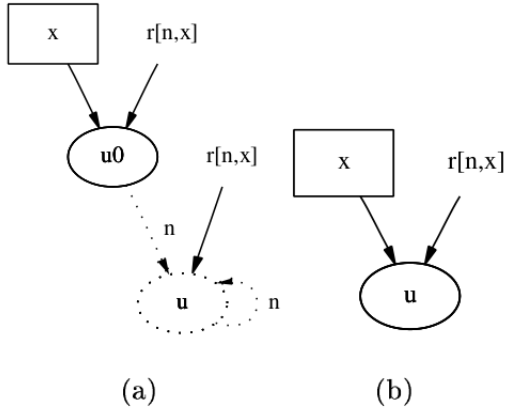
```
1   List reverse(List x){
2       List y, t;
3       y = NULL;
4       while(x != NULL){
5           t = y;
6           y = x;
7           x = x->n;
8           y->n = NULL;
9           y->n = t;
10      }
11      return y;
12  }
13
```

Execution state: cells, connectivity and values of pointer variables.



Shape Analysis: Example

Shape graph:



Compute for each program point a set of shape graphs as a description of the superset of execution states at the point.

Idea of the Paper

- Problem of old shape analysis: updating of a abstract location may affect properties for other cells. (Frame rule)
- Locality between different separation conjunctions.
- Existing symbolic execution method for separation logic[2].

[2] Josh Berdine, Cristiano Calcagno and Peter W. O'Hearn. Symbolic execution with separation logic.

General Semantic Setting

Working with complete lattice: D.
D is constructed from $\mathcal{P}(S')$, where $S' = S \cup \{\top\}$. $\top$ is a special element cprresponds to memory fault.
semantic of a command $[\![c]\!] : D \to D$.
Semantic for key commands:

$$[\![c\,;c']\!] = [\![c]\!]\,;[\![c']\!] \qquad [\![\textbf{if } b \textbf{ then } c \textbf{ else } c']\!] = (\mathsf{filter}(b)\,;[\![c]\!]) \sqcup (\mathsf{filter}(\neg b)\,;[\![c']\!])$$
$$[\![\textbf{while } b \textbf{ do } c]\!] = \lambda d.\ \mathsf{filter}(\neg b)\Big(\mathit{fix}\ \lambda d'.\ d \sqcup (\mathsf{filter}(b)\,;[\![c]\!])(d')\Big)$$

where $\texttt{filter}(b) : D \to D$.
Execution semantic:
$p \Longrightarrow \subseteq S \times (S \cup \{\top\})$
The execution semantic on the powerset of S' is a function $\bar{p} : \mathcal{P}(S') \to \mathcal{P}(S')$:

$$\bar{p}X = \{\sigma' \mid \exists \sigma \in X.(\sigma, p \Longrightarrow \sigma') \vee (\sigma = \sigma' = \top)\}$$

Separation Logic: Concrete and Symbolic Heaps

$$\mathsf{Values} = \mathsf{Locations} \cup \{\mathsf{nil}\} \qquad \mathsf{Heaps} = \mathsf{Locations} \rightharpoonup_f \mathsf{Values}$$
$$\mathsf{Stacks} = (\mathsf{Vars} \cup \mathsf{Vars}') \to \mathsf{Values} \qquad \mathsf{States} = \mathsf{Stacks} \times \mathsf{Heaps}$$

Definition (Symbolic Heap)

A *symbolic heap* $\Pi \mid \Sigma$ consists of a finite set Π of **equalities** between **expressions** and a finite set Σ of heap predicates, where

- Expressions E, F are variables x, primed variables x' or `nil`.
- Heap predicates includes $E \mapsto F$, $\mathtt{ls}(E, F)$ and `junk`.

The meaning of symbolic heap corresponds to a formula:

$$\exists x'_1 x'_2 \ldots x'_n. \left(\bigwedge\nolimits_{P \in \Pi} P\right) \wedge \left(\bigstar\nolimits_{Q \in \Sigma} Q\right)$$

We use $\mathcal{SH}$ to denote the set of consistent symbolic heaps.

Separation Logic: Concrete and Symbolic Heaps

Semantic of symbolic heap:

Example

- $s, h \models E \mapsto F$:
  $s_0(x) = 1, s_0(y) = 10$ and $h_0(1) = 10$. Then $s_0, h_0 \models x \mapsto y$.
- $s, h \models \texttt{ls}(E, F)$: intuitively, this means we have a path from E to F.
  - $s_0(x) = 1, s_0(y) = 10$ and $h_0(1) = 10$. Then $s_0, h_0 \models \texttt{ls}(x, y)$
  - $s_1(x) = 1, s_1(y) = 2, s_1(z) = 3, s_1(w) = 4$ and $h_1(1) = 2, h_1(2) = 3, h_1(3) = 4$. Then $s_1, h_1 \models \texttt{ls}(x, w)$. Or $s_1, h_1 \models x \mapsto y * \texttt{ls}(y, w)$.
- $s, h \models \texttt{junk}$ iff $h \neq \emptyset$

Or graphically:

Questions of the Analysis

The analysis requires us to answer some questions algorithmically:

- $\Pi \vdash E = F$
  Check by consider the equivalent classes.
- $\Pi \mid \Sigma \vdash \texttt{false}$
  - $\Pi \vdash E = \texttt{nil}$ and $\texttt{allocated}(\Sigma, E)$
  - $\Pi \vdash E = F$ and $\texttt{ls}(E, F) \in \Sigma$
  - $\Pi \vdash E = F$ and the two distinct predicates whose lhs are E, F.
- $\Pi \mid \Sigma \vdash E \neq F$ when $\texttt{Vars}'(E, F) = \emptyset$
  Add $E = F$ to the pure part and utilize analysis for `false`.
- $\Pi \mid \Sigma \vdash \texttt{Allocated}(E)$ when $\texttt{Vars}'(E) = \emptyset$

All of the above analysis can be done syntactically.

Questions of the Analysis: Example

Example

- $\Pi \vdash E = F$
  Let Π be $\{x = x', x' = y', y' = y, z = \texttt{nil}\}$. There are two equivalent classes of the variables: $\{x, x', y, y'\}$ and $\{z, \texttt{nil}\}$. From this information we have $\Pi \vdash x = y$ and $\Pi \vdash z = nil$.
- $\Pi \mid \Sigma \vdash \texttt{false}$
  - $x = \texttt{nil} \in \Pi$ and $x \mapsto y \in \Sigma$. Inconsistent.
  - $x = y \in \Pi$ and $\texttt{ls}(x, y) \in \Sigma$. Inconsistent.
  - $x = y', z \neq y', x = z \in \Pi$. Inconsistent.
- $\Pi \mid \Sigma \vdash E \neq F$ when $\texttt{Vars}'(E, F) = \emptyset$
  Add $E = F$ to the pure part and utilize analysis for `false`.
- $\Pi \mid \Sigma \vdash \texttt{Allocated}(E)$ when $\texttt{Vars}'(E) = \emptyset$

Concrete and Symbolic Execution Semantic: Program Considered

$$
\begin{aligned}
b &::= E{=}E \mid E{\neq}E \\
p &::= x{:=}E \mid x{:=}[E] \mid [E]{:=}F \mid \mathbf{new}(x) \mid \mathbf{dispose}(E) \\
c &::= p \mid c\,;c \mid \mathbf{while}\ b\ \mathbf{do}\ c \mid \mathbf{if}\ b\ \mathbf{then}\ c\ \mathbf{else}\ c
\end{aligned}
$$

We use $A(E)$ for any element from the set of primitive commands $\{x := [E], [E] := F, \mathbf{dispose}(E)\}$.
With the existing result of [3], the author provide the following execution semantics.

[3] Josh Berdine et al. Symbolic Execution with Separation Logic

Concrete Execution Semantic

Concrete Execution Rules

$$\frac{\mathcal{C}[\![E]\!]s = n}{s, h,\ x{:=}\,E \implies (s \mid x \mapsto n), h} \qquad \frac{\mathcal{C}[\![E]\!]s = \ell \quad h(\ell) = n}{s, h,\ x{:=}\,[E] \implies (s \mid x \mapsto n), h}$$

$$\frac{\mathcal{C}[\![E]\!]s = \ell \quad \mathcal{C}[\![F]\!]s = n \quad \ell \in \mathsf{dom}(h)}{s, h,\ [E]{:=}\,F \implies s, (h \mid \ell \mapsto n)} \qquad \frac{\ell \notin \mathsf{dom}(h)}{s, h,\ \mathbf{new}(x) \implies (s \mid x \mapsto \ell), (h \mid \ell \mapsto n)}$$

$$\frac{\mathcal{C}[\![E]\!]s = \ell}{s, h * [\ell \mapsto n],\ \mathbf{dispose}(E) \implies s, h} \qquad \frac{\mathcal{C}[\![E]\!]s \notin \mathsf{dom}(h)}{s, h,\ A(E) \implies \top}$$

The σ and σ' here in the concrete execution rules are the elements of `States` defined before.
We extend it to $\mathcal{C}[\![c]\!] : \mathcal{P}(\texttt{States}) \to \mathcal{P}(\texttt{States})$

Example

An example for the mutation rule:
$s(x) = 10, s(y) = 1$ and h has definition on 10.
$s, h, [x] := y \implies s, (h \mid 10 \mapsto 1)$

Symbolic Execution Semantic

The symbolic execution semantic $\sigma, p \Longrightarrow \sigma'$, σ here is a symbolic heap.

SYMBOLIC EXECUTION RULES

$$
\begin{array}{lll}
\Pi \mid \Sigma, & x{:=}E & \Longrightarrow x{=}E[x'/x] \wedge (\Pi \mid \Sigma)[x'/x] \\
\Pi \mid \Sigma * E{\mapsto}F, & x{:=}[E] & \Longrightarrow x{=}F[x'/x] \wedge (\Pi \mid \Sigma * E{\mapsto}F)[x'/x] \\
\Pi \mid \Sigma * E{\mapsto}F, & [E]{:=}G & \Longrightarrow \Pi \mid \Sigma * E{\mapsto}G \\
\Pi \mid \Sigma, & \mathbf{new}(x) & \Longrightarrow (\Pi \mid \Sigma)[x'/x] * x{\mapsto}y' \\
\Pi \mid \Sigma * E{\mapsto}F, & \mathbf{dispose}(E) & \Longrightarrow \Pi \mid \Sigma
\end{array}
$$

$$
\frac{\Pi \mid \Sigma \not\vdash \mathtt{Allocated}(E)}{\Pi \mid \Sigma, A(E) \Longrightarrow \top}
$$

These rules also results in an symbolic execution semantic function $\mathcal{I}[\![c]\!] : \mathcal{P}(\mathcal{SH}) \to \mathcal{P}(\mathcal{SH})$.

Symbolic Execution Semantic: Example

Symbolic Execution Rules

$$
\begin{array}{lll}
\Pi \mid \Sigma, & x{:=}E & \Longrightarrow x{=}E[x'/x] \wedge (\Pi \mid \Sigma)[x'/x] \\
\Pi \mid \Sigma * E{\mapsto}F, & x{:=}[E] & \Longrightarrow x{=}F[x'/x] \wedge (\Pi \mid \Sigma * E{\mapsto}F)[x'/x] \\
\Pi \mid \Sigma * E{\mapsto}F, & [E]{:=}G & \Longrightarrow \Pi \mid \Sigma * E{\mapsto}G \\
\Pi \mid \Sigma, & \mathbf{new}(x) & \Longrightarrow (\Pi \mid \Sigma)[x'/x] * x{\mapsto}y' \\
\Pi \mid \Sigma * E{\mapsto}F, & \mathbf{dispose}(E) & \Longrightarrow \Pi \mid \Sigma
\end{array}
$$

$$
\frac{\Pi \mid \Sigma \not\vdash \mathtt{Allocated}(E)}{\Pi \mid \Sigma, A(E) \Longrightarrow \top}
$$

Example

```
1  int main(){
2      // true|emp
3      x = 1;
4      // x=x'|emp
5      new(y);
6      // x=x'|emp * y|->y'
7      [y]=x;
8      // x=x'|emp * y|->x
9      dispose(y);
10     // x=x'|emp
11     dispose(y);
12     // T
13 }
```

Rearrangement Rules for `ls` Predicate

Rearrangement Rules

$$P(E,F) \;::=\; E{\mapsto}F \mid \mathsf{ls}(E,F)$$

$$\frac{\Pi_0 \mid \Sigma_0 * P(E,G),\, A(E) \implies \Pi_1 \mid \Sigma_1}{\Pi_0 \mid \Sigma_0 * P(F,G),\, A(E) \implies \Pi_1 \mid \Sigma_1}\; \Pi_0 \vdash E{=}F$$

$$\frac{\Pi_0 \mid \Sigma_0 * E{\mapsto}x' * \mathsf{ls}(x',G),\, A(E) \implies \Pi_1 \mid \Sigma_1}{\Pi_0 \mid \Sigma_0 * \mathsf{ls}(E,G),\, A(E) \implies \Pi_1 \mid \Sigma_1} \qquad \frac{\Pi \mid \Sigma * E{\mapsto}F,\, A(E) \implies \Pi' \mid \Sigma'}{\Pi \mid \Sigma * \mathsf{ls}(E,F),\, A(E) \implies \Pi' \mid \Sigma'}$$

Meaning Function and Safe Semantic

To state that the symbolic semantic is sound, we define a *meaning function*:

$$\gamma : \mathcal{P}(\mathcal{SH}) \to \mathcal{P}(\texttt{States})$$

to maps the set of symbolic heap to the set of states that whose elements at least make one of the symbolic heap satisfied.
Then by checking the rules we have

Theorem

The symbolic semantics is a sound overapproximation of the concrete semantics:

$$\forall X \in \mathcal{P}(\mathcal{SH}).\mathcal{C}[\![c]\!](\gamma(X)) \subseteq \gamma(\mathcal{I}[\![c]\!]X)$$

Problem Encountered

If there is no loop in the program, the analysis can be done by only using the symbolic execution on separation logic.

$$[\![c\,;c']\!] = [\![c]\!]\,;[\![c']\!] \qquad [\![\textbf{if } b \textbf{ then } c \textbf{ else } c']\!] = (\mathsf{filter}(b)\,;[\![c]\!]) \sqcup (\mathsf{filter}(\neg b)\,;[\![c']\!])$$
$$[\![\textbf{while } b \textbf{ do } c]\!] = \lambda d.\ \mathsf{filter}(\neg b)\Big(\mathit{fix}\ \lambda d'.\ d \sqcup (\mathsf{filter}(b)\,;[\![c]\!])(d')\Big)$$

The domain $\mathcal{SH}$ is infinite, plus there is no limit on the primed variables:
Define an abstract domain for the fix-point convergence and abstraction rules for the conversion.

- Expression replacement for primed variables.
- Garbage collection rules.
- List abstraction rules.

The conversion is given by $\rightsquigarrow$. The abstract domain after the abstraction is $\mathcal{CSH}$ and corresponding abstract semantic function $\mathcal{A}[\![c]\!] : \mathcal{P}(\mathcal{CSH}) \to \mathcal{P}(\mathcal{CSH})$.

The Analysis

Theorem
$\mathcal{CSH}$ is finite.

Theorem
The abstract semantic is a sound overapproximation of the concrete semantic.

$$\forall X \in \mathcal{P}(\mathcal{SH}).\mathcal{C}[\![c]\!](\gamma(X)) \subseteq \gamma(\mathcal{A}[\![c]\!]X)$$

Example

A program to dispose a list.
`Program`: **while** $(c \neq 0)$ **do** $(t := c; c := c \to tl; \mathbf{dispose(t)})$
`Pre`: $\{\} \mid \{\mathtt{ls}(c, 0)\}$
`Post`: $\{c = 0\} \mid \{\}$
`Inv`: $\{c = 0\} \mid \{\} \vee \{\} \mid \{\mathtt{ls}(c, 0)\}$

Locality

Example

Program: $x := c; c := c \to tl$

Pre: $\{\} \mid \{\texttt{ls}(c, d) * d \mapsto d'\}$

Post: $\{c = d\} \mid \{x \mapsto d * d \mapsto d'\} \vee \{\} \mid \{x \mapsto c * \texttt{ls}(c, d) * d \mapsto d'\}$

A smaller input provide information we need:

Pre: $\{\} \mid \{\texttt{ls}(c, d)\}$

Post: $\{c = d\} \mid \{x \mapsto d\} \vee \{\} \mid \{x \mapsto c * \texttt{ls}(c, d)\}$

The notion of $*$ on symbolic heaps:

$$(\Pi_1 \mid \Sigma_1) * (\Pi_2 \mid \Sigma_2) = ((\Pi_1 \cup \Pi_2 \mid \Sigma_1 * \Sigma_2))$$

Theorem (Frame Rule)

For all $X, Y \in \mathcal{P}(\mathcal{CSH})$, *if* $\texttt{Vars}(Y) \cap Mod = \emptyset$ *then* $\gamma(\mathcal{A}[\![c]\!](X * Y)) \subseteq \gamma(\mathcal{A}[\![c]\!]X) * Y$