Lecture 07 – Shape Analysis

PROGRAM ANALYSIS & SYNTHESIS

Eran Yahav
Previously

- LFP computation and join-over-all-paths
- Inter-procedural analysis
  - call-string approach
  - functional approach
Today

- Shape Analysis
- Typestate Verification
Shape Analysis

Automatically verify properties of programs manipulating dynamically allocated storage

Identify all possible shapes (layout) of the heap
Timeline: Shape Analysis

- **Flow analysis and optimization of Lisp-like structures**
- **Analysis of pointers and structures**
- **Parametric Shape Analysis via 3-valued Logic**
- **Verifying Concurrent Heap Manipulating Programs**
- **Verifying Linearizability with Heap Decomposition**
- **Numerical Abstractions**
- **Logical Characterization of Heap Abstractions**
- **Interprocedural Recursive Programs**
- **Procedure Local Heaps**
- **Verifying Linearizability**
- **Thread Modular Shape Analysis**

Years:
- 1981
- 1990
Timeline: Shape Analysis via 3-Valued Logic

- Parametric Shape Analysis via 3-valued Logic [Sagiv, Reps, Wilhelm POPL'99, TOPLAS'02]
- Verifying Concurrent Heap Manipulating Programs [Yahav, Sagiv, POPL'01]
- Verifying Linearizability with Heap Decomposition [Amit et al., CAV'07]
- Verifying Linearizability [Amit et al., CAV'07]
- Numerical Abstractions
- A Local Shape Analysis Based on Separation Logic
- Thread Modulo Shape Analysis

Sequential Stack

void push (int v) {
    Node *x = malloc(sizeof(Node));
    x->d = v;
    x->n = Top;
    Top = x;
}

int pop () {
    if (Top == NULL) return EMPTY;
    Node *s = Top->n;
    int r = Top->d;
    Top = s;
    return r;
}

Want to Verify
No Null Dereference
Underlying list remains acyclic after each operation
Shape Analysis via 3-valued Logic

1) Abstraction
   - 3-valued logical structure
   - canonical abstraction

2) Transformers
   - via logical formulae
   - soundness by construction
     - embedding theorem, [SRW02]
Concrete State

- represent a concrete state as a two-valued logical structure
  - Individuals = heap allocated objects
  - Unary predicates = object properties
  - Binary predicates = relations
- parametric vocabulary

(Storeless, no heap addresses)
Concrete State

- $S = \langle U, \iota \rangle$ over a vocabulary $P$
- $U$ – universe
- $\iota$ - interpretation, mapping each predicate from $p$ to its truth value in $S$

- $U = \{ u_1, u_2, u_3 \}$
- $P = \{ \text{Top}, n \}$
- $\iota(n)(u_1, u_2) = 1, \iota(n)(u_1, u_3) = 0, \iota(n)(u_2, u_1) = 0, \ldots$
- $\iota(\text{Top})(u_1) = 1, \iota(\text{Top})(u_2) = 0, \iota(\text{Top})(u_3) = 0$
void push (int v) {
    Node *x = malloc(sizeof(Node));
    x->d = v;
    x->n = Top;
    Top = x;
}

Top != null

No node precedes Top

No Cycles
Concrete Interpretation Rules

<table>
<thead>
<tr>
<th>Statement</th>
<th>Update formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = NULL</td>
<td>x'(v) = 0</td>
</tr>
<tr>
<td>x = malloc()</td>
<td>x'(v) = IsNew(v)</td>
</tr>
<tr>
<td>x = y</td>
<td>x'(v) = y(v)</td>
</tr>
<tr>
<td>x = y \rightarrow next</td>
<td>x'(v) = \exists w: y(w) \land n(w, v)</td>
</tr>
<tr>
<td>x \rightarrow next = y</td>
<td>n'(v, w) = (\neg x(v) \land n(v, w)) \lor (x(v) \land y(w))</td>
</tr>
</tbody>
</table>
Example: $s = \text{Top} \rightarrow n$

\[
s'(v) = \exists v_1: \text{Top}(v_1) \land n(v_1, v)
\]

### Tables

<table>
<thead>
<tr>
<th>Top</th>
<th>n</th>
<th>u1</th>
<th>u2</th>
<th>u3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>u1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>u2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>u3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>u1</td>
<td>0</td>
</tr>
<tr>
<td>u2</td>
<td>0</td>
</tr>
<tr>
<td>u3</td>
<td>0</td>
</tr>
</tbody>
</table>

### Diagrams

![Diagram](image-url)
Collecting Semantics

\[
\{ <\emptyset,\emptyset> \} \]

if \( v = \text{entry} \)

\[
\bigcup \{ \llbracket st(w) \rrbracket(S) \mid S \in CSS[w] \} \bigcup \]
\[(w,v) \in E(G),\]
\[w \in \text{Assignments}(G)\]

\[
\bigcup \{ S \mid S \in CSS[w] \} \bigcup \]
\[(w,v) \in E(G),\]
\[w \in \text{Skip}(G)\]

\[
\bigcup \{ S \mid S \in CSS[w] \text{ and } S \models \text{cond}(w) \} \bigcup \]
\[(w,v) \in \text{True-Branches}(G)\]

\[
\bigcup \{ S \mid S \in CSS[w] \text{ and } S \models \neg \text{cond}(w) \} \bigcup \]
\[(w,v) \in \text{False-Branches}(G)\]

CSS \([v] = \)
Collecting Semantics

- At every program point – a potentially infinite set of two-valued logical structures
- Representing (at least) all possible heaps that can arise at the program point

- Next step: find a bounded abstract representation
3-Valued Logic

- $1 = \text{true}$
- $0 = \text{false}$
- $1/2 = \text{unknown}$

- A join semi-lattice, $0 \sqcup 1 = 1/2$
3-Valued Logical Structures

- A set of individuals (nodes) $U$
- Relation meaning
  - Interpretation of relation symbols in $P$
    $p^0() \rightarrow \{0, 1, 1/2\}$
    $p^1(v) \rightarrow \{0, 1, 1/2\}$
    $p^2(u,v) \rightarrow \{0, 1, 1/2\}$
- A join semi-lattice: $0 \sqcup 1 = 1/2$
Boolean Connectives [Kleene]

<table>
<thead>
<tr>
<th>∧</th>
<th>0</th>
<th>1/2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1/2</td>
<td>0</td>
<td>1/2</td>
<td>1/2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1/2</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>∨</th>
<th>0</th>
<th>1/2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1/2</td>
<td>1</td>
</tr>
<tr>
<td>1/2</td>
<td>1/2</td>
<td>1/2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Property Space

- $3\text{-struct}[P] = \text{the set of 3-valued logical structures over a vocabulary (set of predicates) } P$

- Abstract domain
  - $\emptyset (3\text{-Struct}[P])$
  - $\subseteq \text{ is } \subseteq$
    - We will see alternatives later (maybe)
Embedding Order

- Given two structures $S = <U, \mathcal{I}>$, $S' = <U', \mathcal{I}'>$ and an onto function $f : U \to U'$ mapping individuals in $U$ to individuals in $U'$
- We say that $f$ embeds $S$ in $S'$ (denoted by $S \sqsubseteq^f S'$) if
  - for every predicate symbol $p \in P$ of arity $k$: $u_1, \ldots, u_k \in U$, $(p)(u_1, \ldots, u_k) \sqsubseteq^f (p)(f(u_1), \ldots, f(u_k))$
  - and for all $u' \in U'$
    - $(| \{ u | f(u) = u' \} | > 1) \sqsubseteq^f (s)(u')$
- We say that $S$ can be embedded in $S'$ (denoted by $S \sqsubseteq S'$) if there exists a function $f$ such that $S \sqsubseteq^f S'$
Tight Embedding

- $S' = <U',\ i'>$ is a tight embedding of $S=<U,\ i>$ with respect to a function $f$ if:
  - $S'$ does not lose unnecessary information

\[
i'(u'_1,\ldots, u'_k) = \bigsqcup\{i(u_1,\ldots, u_k) \mid f(u_1)=u'_1,\ldots, f(u_k)=u'_k\}
\]

- One way to get tight embedding is canonical abstraction
Canonical Abstraction

[Sagiv, Reps, Wilhelm, TOPLAS02]
Canonical Abstraction

[Top] $u_1 \xrightarrow{n} u_2 \xrightarrow{n} u_3$

[Top] $u_1 \xrightarrow{n} u_3$

[Sagiv, Reps, Wilhelm, TOPLAS 2002]
Canonical Abstraction

\[ \eta(u_1, u_{2,3}) = ? \]

\[ U(\eta(u_1, u_2) = 1) \]

\[ \eta(u_1, u_3) \]

\[ = \frac{1}{2} \]
Canonical Abstraction

\[ \text{Top} \rightarrow u_1 \rightarrow u_2 \rightarrow u_3 \]

\[ \text{Top} \rightarrow \text{Ring} \]
Canonical Abstraction

Top

u1 \rightarrow u2 \rightarrow u3

Top
Canonical Abstraction

\[ \text{Top} \rightarrow u_1 \rightarrow u_2 \rightarrow u_3 \]

\[ \text{Top} \rightarrow \varnothing \rightarrow \varnothing \]
Canonical Abstraction ($\beta$)

- Merge all nodes with the same unary predicate values into a single summary node
- Join predicate values

$$i'(u'_1,\ldots,u'_k) = \sqcup \{ i(u_1,\ldots,u_k) \mid f(u_1)=u'_1,\ldots,f(u_k)=u'_k \}$$

- Converts a state of arbitrary size into a 3-valued abstract state of bounded size

$$\alpha(C) = \sqcup \{ \beta(c) \mid c \in C \}$$
Information Loss

![Diagram]

Top $\rightarrow$ $\rightarrow$ $\rightarrow$

Canonical abstraction

Top $\rightarrow$ $\rightarrow$ $\rightarrow$

Top $\rightarrow$ $\rightarrow$ $\rightarrow$

Top $\rightarrow$ $\rightarrow$ $\rightarrow$

Top $\rightarrow$ $\rightarrow$ $\rightarrow$
Instrumentation Predicates

- Record additional derived information via predicates

\[ r_x(v) = \exists v_1: x(v_1) \land n^*(v_1, v) \]
\[ c(v) = \exists v_1: n(v_1, v) \land n^*(v, v_1) \]
Embedding Theorem:
Conservatively Observing Properties

No Cycles
\[ \neg \exists v_1, v_2 : n(v_1, v_2) \land n^*(v_2, v_1) \]

No cycles (derived)
\[ \forall v : \neg c(v) \]

1/2

1
Operational Semantics

```c
void push (int v) {
    Node *x = malloc(sizeof(Node));
    x->d = v;
    x->n = Top;
    Top = x;
}

int pop() {
    if (Top == NULL) return EMPTY;
    Node *s = Top->n;
    int r = Top->d;
    Top = s;
    return r;
}
```

\[
[s = \text{Top} \rightarrow n] \\
s'(v) = \exists v_1: \text{Top}(v_1) \land n(v_1, v)
\]
Abstract Semantics

\[ s = \text{Top} \rightarrow n \]

\[
[s = \text{Top} \rightarrow n]
\]

\[ s'(v) = \exists v_1: \text{Top}(v_1) \land n(v_1, v) \]
Best Transformer ($s = \text{Top} \rightarrow n$)

Concrete Semantics

Abstract Semantics

Canonical Abstraction

$s'(v) = \exists v_1: \text{Top}(v_1) \land n(v_1, v)$
Then a Miracle Occurs

"I think you should be more explicit here in step two."
Semantic Reduction

- Improve the precision of the analysis by recovering properties of the program semantics
- A Galois connection \((C, \alpha, \gamma, A)\)
- An operation \(op:A \rightarrow A\) is a semantic reduction when
  - \(\forall l \in L_2 \; op(l) \sqsubseteq l\) and
  - \(\gamma(op(l)) = \gamma(l)\)
The Focus Operation

- Focus: Formula $\mapsto (\varphi (3\text{-Struct}) \leftrightarrow \varphi (3\text{-Struct}))$
- Generalizes materialization
- For every formula $\varphi$
  - Focus($\varphi$)(X) yields structure in which $\varphi$ evaluates to a definite values in all assignments
  - Only maximal in terms of embedding
  - Focus($\varphi$) is a semantic reduction
  - But Focus($\varphi$)(X) may be undefined for some X
Partial Concretization Based on Transformer ($s=\text{Top} \rightarrow n$)

Abstract Semantics

$s'(v) = \exists v_1: \text{Top}(v_1) \land n(v_1, v)$

Partial Concretization

$\exists u: \text{top}(u) \land n(u, v)$

Canonical Abstraction
Partial Concretization

- Locally refine the abstract domain per statement
- Soundness is immediate
- Employed in other shape analysis algorithms
  [Distefano et.al., TACAS’06, Evan et.al., SAS’07, POPL’08]
- Employed in other analysis algorithms
  [Typestate verification, ISSTA’06]
The Coercion Principle

- Another Semantic Reduction
- Can be applied after Focus or after Update or both
- Increase precision by exploiting some structural properties possessed by all stores (Global invariants)
- Structural properties captured by constraints
- Apply a constraint solver
Apply Constraint Solver
Sources of Constraints

- Properties of the operational semantics
- Domain specific knowledge
  - Instrumentation predicates
- User supplied
Example Constraints

\[ x(v_1) \land x(v_2) \rightarrow eq(v_1, v_2) \]

\[ n(v, v_1) \land n(v, v_2) \rightarrow eq(v_1, v_2) \]

\[ n(v_1, v) \land n(v_2, v) \land \neg eq(v_1, v_2) \leftarrow is(v) \]

\[ n^*(v_3, v_4) \leftarrow t[n](v_1, v_2) \]
Abstract Transformers: Summary

- Kleene evaluation yields sound solution
- Focus is a statement-specific partial concretization
- Coerce applies global constraints
Abstract Semantics

$$SS[v] = \begin{cases} \{ <\emptyset, \emptyset> \} & \text{if } v = \text{entry} \\ \bigcup \{ t\_\text{embed}(\text{coerce}(\llbracket \text{st}(w) \rrbracket_3(\text{focus}\_F(w)(SS[w]))) \}) \bigcup \\ (w,v) \in E(G), \\ w \in \text{Assignments}(G) \\ \bigcup \{ S \mid S \in SS[w] \} \bigcup \\ (w,v) \in E(G), \\ w \in \text{Skip}(G) \\ \bigcup \{ t\_\text{embed}(S) \mid S \in \text{coerce}(\llbracket \text{st}(w) \rrbracket_3(\text{focus}\_F(w)(SS[w]))) \} \bigcup \\ (w,v) \in \text{True-Branches}(G) \\ \bigcup \{ t\_\text{embed}(S) \mid S \in \text{coerce}(\llbracket \text{st}(w) \rrbracket_3(\text{focus}\_F(w)(SS[w]))) \} \bigcup \\ (w,v) \in \text{False-Branches}(G) \end{cases}$$
Recap

- Abstraction
  - canonical abstraction
  - recording derived information

- Transformers
  - partial concretization (focus)
  - constraint solver (coerce)
  - sound information extraction
Stack Push

```c
void push (int v) {
    Node *x = alloc(sizeof(Node));
    x->d = v;
    x->n = Top;
    Top = x;
}
```
#define EMPTY -1

typedef int data_type;

typedef struct node t {
    data_type d;
    struct node t *n
} Node;

typedef struct stack t {
    struct node t *Top;
} Stack;

[1] void push(Stack *S, data_type v) {
[2]     Node *x = alloc(sizeof(Node));
[3]     x->d = v;
[4]     do {
[5]         Node *t = S->Top;
[6]         x->n = t;
[7]     } while (!CAS(&S->Top,t,x));
[8] }

[9] data_type pop(Stack *S){
[10]    do {
[12]        if (t == NULL)
[13]            return EMPTY;
[14]        Node *s = t->n;
[15]        data_type r = t->d;
[16]    } while (!CAS(&S->Top,t,s));
[17]    return r;
[18] }

Non-blocking Stack [Treiber 1986]
Concurrent Shape Analysis

- a thread is represented as a thread object
- add predicates to vocabulary

Recipe
1) abstraction: canonical abstraction
2) transformers: interleaving + as before

- Bounded threads
  - Static thread names
- Unbounded threads
  - thread objects abstracted via canonical abstraction
Concrete State

U = \{ u_1, u_2, u_3, \ldots, u_7 \}

isThread = \{ u_1, u_2 \}

at[pc=1] = \{ \}

\ldots

at[pc=6] = \{ u_3 \}

at[pc=7] = \{ u_1 \}

Top = \{ u_5 \}

\ldots

x = \{ (u_1,u_4), (u_2,u_3) \}

\( t = \{ (u_1,u_5), (u_2,u_6) \} \)

n = \{ (u_5,u_6), (u_6,u_7) \}

\( t_1 = \{ u_1 \} \quad t_2 = \{ u_2 \} \)
void push(Stack *S, data_type v) {
    Node *x = alloc(sizeof(Node));
    x->d = v;
    do {
        Node *t = S->Top;
        x->n = t;
    } while (!CAS(&S->Top, t, x));
}

∀x: ¬c(v)

Exploration
Representing an Unbounded Number of Threads

[1] void push(Stack *S, data_type v) {
[2]     Node *x = alloc(sizeof(Node));
[3]     x->d = v;
[4]     do {
[5]         Node *t = S->Top;
[6]         x->n = t;
[7]     } while (!CAS(&S->Top, t, x));
[8] }

\[\text{pc}=5\] \quad \text{pc}=5 \quad \text{pc}=5 \quad \text{Top}
Representing an Unbounded Number of Threads

[1] void push(Stack *S, data_type v) {
[2]     Node *x = alloc(sizeof(Node));
[3]     x->d = v;
[4]     do {
[5]         Node *t = S->Top;
[6]         x->n = t;
[7]     } while (!CAS(&S->Top, t, x));
[8] }

54
Abstract Semantics

[1] void push(Stack *S, data_type v) {
[2]     Node *x = alloc(sizeof(Node));
[3]     x->d = v;
[4]     do {
[5]         Node *t = S->Top;
[6]         x->n = t;
[7]     } while (!CAS(&S->Top, t, x));
[8] }

55
Example - Mutual Exclusion

[1] while (true) {
[2]   lock(shared)
[C]   // critical actions
[3]   unlock(shared)
[4] }

\[ \forall t_1, t_2: (t_1 \neq t_2) \rightarrow \neg(at[pc=c](t_1) \wedge at[pc=c](t_2)) \]

Initial configuration

A thread enters the critical section

Other threads may be blocked or just beginning execution
Recap

- No null dereferences
- Structural shape invariants
- Linearizability

- Dynamic Allocation
- Destructive Updates
- Concurrency