Lecture 08(a) – Shape Analysis – continued
Lecture 08(b) – Typestate Verification
Lecture 08(c) – Predicate Abstraction

Eran Yahav
Previously

- Shape Analysis
Today

- Shape Analysis – continued
- Concurrent Shape Analysis
- Typestate Verification
- Predicate Abstraction (optimistically!)
Shape Analysis

Automatically verify properties of programs manipulating dynamically allocated storage

Identify all possible shapes (layout) of the heap
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]
Collecting State Semantics

\[ CSS[v] = \begin{cases} \emptyset, \emptyset & \text{if } v = \text{entry} \\ \bigcup \{ \text{st}(w)(S) \mid S \in CSS[w] \} \bigcup \bigcup \{ S \mid S \in CSS[w] \} & \text{otherwise} \\ \bigcup \{ S \mid S \in CSS[w] \text{ and } S \models \text{cond}(w) \} \bigcup \bigcup \{ S \mid S \in CSS[w] \text{ and } S \models \neg \text{cond}(w) \} \end{cases} \]
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 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: $\emptyset \sqcup 1 = 1/2$
Property Space

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

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

\[
\text{Top} \rightarrow u_1 \rightarrow u_2 \rightarrow u_3
\]
Canonical Abstraction ($\beta$)

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

$$\iota'(u'_1, ..., u'_k) = \sqcap \{ \iota(u_1, ..., u_k) \mid f(u_1) = u'_1, ..., f(u_k) = u'_k \}$$

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

$$\alpha(C) = \sqcap \{ \beta(c) \mid c \in C \}$$
Abstract Semantics

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

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

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

- **Focus**: Formula→(φ(3-Struct) ↩ φ(3-Struct))
- Generalizes materialization
- For every formula φ
  - Focus(φ)(X) yields structure in which φ evaluates to a definite values in all assignments
  - Only maximal in terms of embedding
  - Focus(φ) is a semantic reduction
  - But Focus(φ)(X) may be undefined for some X
Partial Concretization Based on Transformer (s=Top→n)

Abstract Semantics

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

Focus (Top→n)

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

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 structural properties possessed by all stores (Global invariants)
- Structural properties captured by constraints
- Apply a constraint solver
Apply Constraint Solver

Top $\rightarrow r_{Top}$

$\Rightarrow$ $x$

Top $\rightarrow r_{Top}$

$\Rightarrow$ Top $\rightarrow r_{Top}$

Top $x, r_y$

$\Rightarrow$ Top $x, r_y$

Top $x, r_y$

$\Rightarrow$ Top $x, r_y$

Top $x, r_y$

$\Rightarrow$ Top $x, r_y$
Sources of Constraints

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

\( x(v_1) \land x(v_2) \rightarrow \text{eq}(v_1, v_2) \)

\( n(v, v_1) \land n(v, v_2) \rightarrow \text{eq}(v_1, v_2) \)

\( n(v_1, v) \land n(v_2, v) \land \neg \text{eq}(v_1, v_2) \leftrightarrow \text{is}(v) \)

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

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

$$SS[v] = \begin{cases} \{ <\emptyset, \emptyset> \} & \text{if } v = \text{entry} \\ \bigcup \{ t_{\text{embed}}(\text{coerce}(\llbracket \text{st}(w) \rrbracket_3(\text{focus}_{F(w)}(SS[w]))) \}) \bigcup (w,v) \in E(G), \\ w \in \text{Assignments}(G) \\ \bigcup \{ S \mid S \in SS[w] \} \bigcup (w,v) \in E(G), \\ w \in \text{Skip}(G) \\ \bigcup \{ t_{\text{embed}}(S) \mid S \in \text{coerce}(\llbracket \text{st}(w) \rrbracket_3(\text{focus}_{F(w)}(SS[w]))) \text{ and } S \models_3 \text{cond}(w) \} \bigcup (w,v) \in \text{True-Branches}(G) \\ \bigcup \{ t_{\text{embed}}(S) \mid S \in \text{coerce}(\llbracket \text{st}(w) \rrbracket_3(\text{focus}_{F(w)}(SS[w]))) \text{ and } S \models_3 \neg \text{cond}(w) \} \bigcup (w,v) \in \text{False-Branches}(G) \end{cases}$$
Recap

- Abstraction
  - canonical abstraction
  - recording derived information

- Transformers
  - partial concretization (focus)
  - constraint solver (coerce)
  - sound information extraction
void push (int v) {
    Node *x = alloc(sizeof(Node));
    x->d = v;
    x->n = Top;
    Top = x;
}

\[ \exists v: x(v) \]
\[ x \rightarrow d = v; \]
\[ \exists v: x(v) \]
\[ x \rightarrow n = \text{Top}; \]
\[ \forall v: \neg c(v) \]
\[ \neg \exists v_1, v_2: n(v_1, v_2) \land \text{Top}(v_2) \]
Non-blocking Stack [Treiber 1986]

```c
#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] }
```
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

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

\[
isThread = \{ u_1, u_2 \}
\]

\[
\text{at}[pc=1] = \{}
\]

\[
\ldots
\]

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

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

\[
\text{Top} = \{ u_5 \}
\]

\[
\ldots
\]

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

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

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

\[
t_1 = \{ u_1 \}
\]

\[
t_2 = \{ u_2 \}
\]
Exploration

[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] }

∀v: ¬C(v)
Representing an Unbounded Number of Threads

Listing 1

[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] }

Diagram:

- pc=5
- Node x
- n
- Top

pc=5

pc=5

pc=5

Node x
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] }

Abstract Semantics

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));
}

[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] }
Example - Mutual Exclusion

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

∀t_1,t_2: (t_1 ≠ t_2) → ¬(at[pc=c](t_1) ∧ at[pc=c](t_2))

Initial configuration

A thread enters the critical section

Other threads may be blocked or just beginning execution
Recap

```c
#define EMPTY -1
typedef int data_type;
typedef struct node t {
data_type d;
struct node *n
} Node;
typedef struct stack t {
struct node *Top;
} Stack;
```

- No null dereferences
- Structural shape invariants
- Linearizability
- Dynamic Allocation
- Destructive Updates
- Concurrency