Modeling the Heap: Arrays and Separation Logic

Computer Science and Artificial Intelligence Laboratory
MIT

With content from the paper "Local Reasoning about Programs that Alter Data Structures" by O'Hearn, Reynolds and Yang.

November 2, 2015

Approach 1: Heap as Array

Consider a c-style language

- New expressions: $e \coloneqq malloc(n) \mid *e$
- Statements: $c \coloneqq \ *e \coloneqq e$

In C, the heap is essentially one big array:

```
x = malloc(2);
*x = 4;
*(x+1) = z;
y = *(x) + *(x+1);
{y == 4 + z }
```

```
x = HEAP_PTR;
HEAP_PTR = HEAP_PTR + 2;
HEAP[x] = 4;
HEAP[x+1] = z;
y = HEAP[x] + HEAP[x+1];
{y == 4 + z}
```

Heap as Array

- Treat the heap as a giant array
- Use special values/ghost arrays to distinguish un-allocated memory from un-initialized memory
- Use simple counters to model the allocator
- Model using the theory of arrays

Advantages

- No new machinery required
- Very general
- Many opportunities for refinement and optimization

Heap as Array

Can even model deallocation

```
x = malloc(2);
*x = 4;
*(x+1) = z;
y = *(x) + *(x+1);
free(x);
{y == 4 + z }
```

```
x = HEAP_PTR;
LIVE[x] = true;
LIVE[x+1] = true;
SIZE[x] = 2;
HEAP_PTR = HEAP_PTR + 2;
HEAP[x] = 4;
Assert LIVE[x];
HEAP[x+1] = z;
Assert LIVE[x+1];
y = HEAP[x] + HEAP[x+1];
Assert LIVE[x] && LIVE[x+1];
Assert LIVE[x];
for(i=0; i<size[x]; ++i){
  Assert LIVE[x+i];
  LIVE[x+i] = false;
}
{y == 4 + z}
```

```
Works really well as long as
you don’t need to interact
with the user.
```

Heap as Array

What about loops?

- x points to a list of the form List{ val:int, List:next}
- At the end of the loop, t = sum(x) the sum of all elements in the list.
- How do we even express this?

```
t = 0;
while( x != null){
    t = t + *x;
    x = *(x+1);
}
```

$$\exists g.\ g[x] \wedge \forall i, j.\ \ g[i]\ \ \wedge\ \ H[i+1] = j \Leftrightarrow g[j]\ \ \wedge\ \ t = \sum_{k \in \{k | g[k]\}} H[k]$$

  - Maybe?

- What about the invariant?

Heap as Array

The approach is not entirely useless

```
t = 0;
while( x != null){
   t = t + *x;
   x = *(x+1);
}
```

- Unrolling loops can eliminates the need for invariants
- But it sacrifices soundness

```
t = 0;
if( x != null){
   t = t + *x;
   x = *(x+1);
   if( x != null){
     t = t + *x;
     x = *(x+1);
    if( x != null){
      t = t + *x;
      x = *(x+1);
      if( x != null){
       Assume false;
}  } }}
```

Heap as an Array approach

Widely used in practical tools

- One of the main drivers for scalable TOA in SMT solvers

Writing invariants can be a challenge

- Most tools that use this approach don't bother with invariants
- Problem is that the structure of any data-structure in the program gets lost in the low-level representation

Approach 2: Separation logic

- See: "Local Reasoning about Programs that Alter Data Structures"
  - By O'Hearn, Reynolds and Yang

Key idea:

- Break the heap into <u>disjoint</u> pieces
- Focus on a few small pieces at a time
- Statements affect one piece at a time

The language

- Imp + extensions
  - Stmt : $x = [e] \mid [e] = x \mid x = cons(e1, \dots, ek) \mid dispose(e)$
- Very similar in spirit to what we saw before with Op Sem
  - $s \in S : Id \to Int$
  - $h \in H : Nat \to Int$
  - $[\![C]\!] : S \times H \to S \times H \cup \{\bot\}$
  - $[\![E]\!] : S \to Int$
  - $[\![x = e]\!]\ s\ h = s\{x \to [\![e]\!]\ s\ \}\ h$
  - $[\![x = [e]]\!]\ s\ h = s\{x \to h([\![e]\!]\ s)\}\ h$
  - $[\![[e] = x]\!]\ s\ h = s\ h\{[\![e]\!]\ s \to \mathrm{s(x)}\}$
  - $[\![x = cons(e_0 \dots e_k)]\!]\ s\ h = s\{x \to j\}\ h\{j \to [\![e_0]\!]\ s, \dots, j + k \to [\![e_k]\!]\ s\}$
    $where\ j = (\max dom\ h) + 1$
  - $[\![dispose(e)]\!]\ sh = s\ h\{[\![e]\!]s \ \to \}$

Separation Logic: Notation

Heaps are described by predicates in the following language:

emp := The heap is empty

- There are no cells in this heap

$x \longmapsto y$:= The heap has exactly one cell.

- This cell is at location x
- This cell stores the value y

A * B := Heap can be partitioned into two <u>disjoint</u> regions,

- one region where A is true,
- one region where B is true

Formalizing the notation

Copied from the paper by O'Hearn, Reynolds and Yang

$$s, h \models B \quad \text{iff } [\![B]\!]s = \mathit{true}$$

$$s, h \models E \mapsto F \quad \text{iff } \{[\![E]\!]s\} = \mathit{dom}(h) \text{ and } h([\![E]\!]s) = [\![F]\!]s$$

$$s, h \models \mathtt{false} \quad \text{never}$$

$$s, h \models P \Rightarrow Q \quad \text{iff if } s, h \models P \text{ then } s, h \models Q$$

$$s, h \models \forall x.P \quad \text{iff } \forall v \in \mathtt{Ints}.\, [s \mid x \mapsto v], h \models P$$

$$s, h \models \mathtt{emp} \quad \text{iff } h = [\,] \text{ is the empty heap}$$

$$s, h \models P * Q \quad \text{iff } \exists h_0, h_1.\, h_0 \# h_1,\ h_0 * h_1 = h,\ s, h_0 \models P \text{ and } s, h_1 \models Q$$

$h_0 \# h_1 \equiv$ Domains of h_0 and h_1 are disjoint
$h_0 * h_1 \equiv$ Union of disjoint heaps

Some additional shorthand

Copied from the paper by O'Hearn, Reynolds and Yang

$$
\begin{array}{lll}
E \mapsto F_0, \ldots, F_n & \stackrel{\Delta}{=} & (E \mapsto F_0) * \cdots * (E + n \mapsto F_n) \\
E \doteq F & \stackrel{\Delta}{=} & (E = F) \wedge \mathtt{emp} \\
E \mapsto - & \stackrel{\Delta}{=} & \exists y. E \mapsto y
\end{array}
$$

An interesting property

$$(E \doteq F) * P \quad \Leftrightarrow \quad (E = F) \wedge P.$$

Algebra of heap predicates

Which assertions are valid?

$$E \Rightarrow E * E$$

$$E * F \Rightarrow E$$

$$10 \mapsto 3 \Rightarrow 10 \mapsto 3 * 10 \mapsto 3$$

$$10 \mapsto 3 \Rightarrow 10 \mapsto 3 * 42 \mapsto 5$$

$$E \mapsto 3 \Rightarrow 0 \leq E$$

$$E \mapsto - * E \mapsto -$$

$$E \mapsto - * F \mapsto - \Rightarrow E \neq F$$

$$E \mapsto - \wedge F \mapsto - \Rightarrow E = F$$

$$E \mapsto 3 * F \mapsto 3 \Rightarrow E \neq F$$

Copied from the paper by O'Hearn, Reynolds and Yang

Algebra of heap predicates

Which assertions are valid?

$$E \Rightarrow E * E$$ ✗

$$E * F \Rightarrow E$$ ✗

$$10 \mapsto 3 \Rightarrow 10 \mapsto 3 * 10 \mapsto 3$$ ✗

$$10 \mapsto 3 \Rightarrow 10 \mapsto 3 * 42 \mapsto 5$$ ✗

$$E \mapsto 3 \Rightarrow 0 \leq E$$

$$(E \mapsto -) * (E \mapsto -)$$ ✗

$$E \mapsto - * F \mapsto - \Rightarrow E \neq F$$

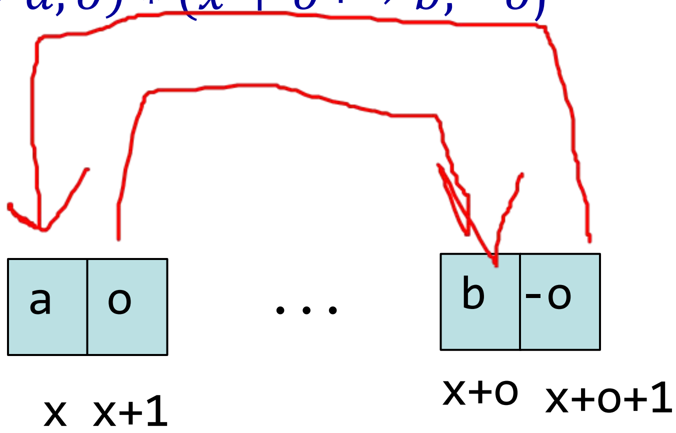
$$E \mapsto - \wedge F \mapsto - \Rightarrow E = F$$

$$E \mapsto 3 * F \mapsto 3 \Rightarrow E \neq F$$

Describing data-structures

What does this heap describe?

- $(x \longmapsto a, o) * (x + o \longmapsto b, -o)$



Proofs Rules for Separation Logic

Small Axioms

Copied from the paper by O'Hearn, Reynolds and Yang

$$\{E \mapsto -\}\, [E] := F\, \{E \mapsto F\}$$

$$\{E \mapsto -\}\, \mathtt{dispose}(E)\, \{\mathtt{emp}\}$$

$$\{x \doteq m\} x := \mathtt{cons}(E_1, ..., E_k) \{x \mapsto E_1[m/x], ..., E_k[m/x]\,\}$$

$$\{x \doteq n\}\, x := E\, \{x \doteq (E[n/x])\}$$

$$\{E \mapsto n \wedge x = m\}\, x := [E]\, \{x = n \wedge E[m/x] \mapsto n\}$$

$(x = m) \wedge emp$

Proof Rules for Separation Logic

Copied from the paper by O'Hearn, Reynolds and Yang

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}} \quad \text{where } \mathsf{Mod}(C) \cap \mathsf{Free}(R) = \emptyset$$

$$\frac{\{P\}C\{Q\}}{\{P \wedge R\}C\{Q \wedge R\}} \quad \text{where } \mathsf{Mod}(C) \cap \mathsf{Free}(R) = \emptyset$$

note: $\mathsf{Mod}(x = e) = \{x\}, \mathsf{Mod}([e] = x) = \emptyset$

$$\frac{\{P\}C\{Q\}}{\{\exists x. P\}C\{\exists x. Q\}} \quad x \notin Free(C)$$

$\mathrm{Free}(P)$ is the set of variables occurring freely in P

More Cycle Free Data-structures

Copied from the paper by O'Hearn, Reynolds and Yang

Linked lists.

$$\begin{aligned}
\mathsf{lseg}(e, f) &\Leftrightarrow \text{if } e = f \text{ then emp else} \\
&\quad \exists y. e \mapsto -, y * \mathsf{lseg}(y, f) \\
\mathsf{list}(e) &\Leftrightarrow \mathsf{lseg}(e, \mathsf{nil})
\end{aligned}$$

$$\mathsf{lseg}(x, y) * \mathsf{lseg}(y, x)$$

$$\mathsf{lseg}(x, t) * t \mapsto -, y * \mathsf{list}(y)$$

Examples

Proof of $\{\mathsf{list}(x)\}y = \mathsf{cons}(b, x)\{\mathsf{list}(y)\}$

$$\frac{}{\{\mathsf{list}(x)\}y = \mathsf{cons}(b, x)\{\mathsf{list}(y)\}}$$

Examples

Proof of $\{\mathsf{list}(x) \wedge x \neq \mathsf{nil}\} t = [x] \{x \mapsto t * \mathsf{list}(t)\}$

$$\frac{}{\{\mathsf{list}(x) \wedge x \neq \mathsf{nil}\} t = [x] \{x \mapsto t * \mathsf{list}(t)\}}$$

6.820 Fundamentals of Program Analysis
Fall 2015