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.
© Springer-Verlag Berlin Heidelberg 2001. All rights reserved.
This content is excluded from our Creative Commons license.
For more information, see http://ocw.mit.edu/help/faq-fair-use/.

November 2, 2015
Approach 1: Heap as Array

Consider a c-style language

- New expressions: \( e \equiv malloc(n) | *e \)
- Statements: \( c \equiv *e := e \)

In C, the heap is essentially one big array:

\[
\begin{align*}
x &= \text{malloc}(2); \\
*x &= 4; \\
*(x+1) &= z; \\
y &= *(x) + *(x+1); \\
\{y == 4 + z \}
\end{align*}
\]

\[
\begin{align*}
x &= \text{HEAP_PTR}; \\
\text{HEAP_PTR} &= \text{HEAP_PTR} + 2; \\
\text{HEAP}[x] &= 4; \\
\text{HEAP}[x+1] &= z; \\
y &= \text{HEAP}[x] + \text{HEAP}[x+1]; \\
\{y == 4 + z \}
\end{align*}
\]
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

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

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

```c
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}
```
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);
  }

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

- 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 disjoint pieces
- Focus on a few small pieces at a time
- Statements affect one piece at a time
The language

Imp + extensions

- Stmt : \( x = [e] \) | \([e] = x \) | \( x = \text{cons}(e_1, ..., e_k) \) | \( \text{dispose}(e) \)

Very similar in spirit to what we saw before with Op Sem

- \( s \in S : \text{Id} \to \text{Int} \)
- \( h \in H : \text{Nat} \to \text{Int} \)
- \( [C] : S \times H \to S \times H \cup \{ \perp \} \)
- \( [E] : S \to \text{Int} \)
- \( [x = e] s h = s\{x \to [e] s\} h \)
- \( [x = [e]] s h = s\{x \to h([e] s)\} h \)
- \( [[e] = x] s h = s h\{[e] s \to s(x)\} \)
- \( [x = \text{cons}(e_0, ..., e_k)] s h = s\{x \to j\} h\{j \to [e_0] s, ..., j + k \to [e_k] s\} \)
  where \( j = (\text{max dom } h) + 1 \)
- \( [\text{dispose}(e)] s h = s h\{[e] s \to \} \)
Separation Logic: Notation

Heaps are described by predicates in the following language:

emp := The heap is empty
   - There are no cells in this heap

\( x \mapsto y \) := 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 disjoint 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

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

\[ s, h \models E \rightarrow F \quad \text{iff} \quad \{[E]s\} = \text{dom}(h) \text{ and } h([E]s) = [F]s \]

\[ s, h \models \text{false} \quad \text{never} \]

\[ s, h \models P \rightarrow Q \quad \text{iff if } s, h \models P \text{ then } s, h \models Q \]

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

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

\[ s, h \models P \ast Q \quad \text{iff } \exists h_0, h_1. \ h_0 \# h_1, \ h_0 \ast h_1 = h, \ s, h_0 \models P \text{ and } s, h_1 \models Q \]

\( h_0 \# h_1 \equiv \text{Domains of } h_0 \text{ and } h_1 \text{ are disjoint} \)

\( h_0 \ast h_1 \equiv \text{Union of disjoint heaps} \)
Some additional shorthand

\[ E \leftrightarrow F_0, \ldots, F_n \triangleq (E \leftrightarrow F_0) \ast \cdots \ast (E + n \leftrightarrow F_n) \]
\[ E \triangleright F \triangleq (E = F) \land \text{emp} \]
\[ E \leftrightarrow - \triangleq \exists y. E \leftrightarrow y \]

An interesting property

\[(E \triangleright F) \ast P \iff (E = F) \land P.\]
Algebra of heap predicates

Which assertions are valid?

\[ E \Rightarrow E \ast E \]
\[ E \ast F \Rightarrow E \]
\[ 10 \mapsto 3 \Rightarrow 10 \mapsto 3 \ast 10 \mapsto 3 \]
\[ 10 \mapsto 3 \Rightarrow 10 \mapsto 3 \ast 42 \mapsto 5 \]
\[ E \mapsto 3 \Rightarrow 0 \leq E \]
\[ E \mapsto - \ast E \mapsto - \]
\[ E \mapsto - \ast F \mapsto - \Rightarrow E \neq F \]
\[ E \mapsto - \land F \mapsto - \Rightarrow E = F \]
\[ E \mapsto 3 \ast F \mapsto 3 \Rightarrow E \neq F \]
Algebra of heap predicates

Which assertions are valid?

\[
\begin{align*}
E \Rightarrow E \ast E & \times \\
E \ast F \Rightarrow E & \times \\
10 \leftarrow 3 \Rightarrow 10 & \leftarrow 3 \ast 10 \leftarrow 3 \times \\
10 \leftarrow 3 \Rightarrow 10 & \leftarrow 3 \ast 42 \leftarrow 5 \times \\
E \leftarrow 3 \Rightarrow 0 & \leq E \\
\langle E \leftarrow \neg \rangle \ast \langle E \leftarrow \neg \rangle & \times \\
E \leftarrow \neg \ast F & \leftarrow \neg \Rightarrow E \neq F \\
E \leftarrow \neg \land F & \leftarrow \neg \Rightarrow E = F \\
E \leftarrow 3 \ast F & \leftarrow 3 \Rightarrow E \neq F
\end{align*}
\]
Describing data-structures

What does this heap describe?

- \((x \mapsto a, o) \ast (x + o \mapsto b, -o)\)
Proofs Rules for Separation Logic

Small Axioms

\[
\begin{align*}
\{ E \leftarrow \neg \} [E] & := F \{ E \leftarrow F \} \\
\{ E \leftarrow \neg \} \text{dispose}(E) & \{ \text{emp} \} \\
\{ x \leftarrow m \} x := \text{cons}(E_1, \ldots, E_k) & \{ x \leftarrow E_1[m/x], \ldots, E_k[m/x] \} \\
\{ x \leftarrow n \} x := E & \{ x \leftarrow (E[n/x]) \} \\
\{ E \leftarrow n \land x = m \} x := [E] & \{ x = n \land E[m/x] \leftarrow n \}
\end{align*}
\]

\((x=m) \land \text{emp}\)
Proof Rules for Separation Logic

\[
\frac{\{P\}C\{Q\}}{\{P \land R\}C\{Q \land R\}} \quad \text{where } \text{Mod}(C) \cap \text{Free}(R) = \emptyset
\]

\[
\frac{\{P\}C\{Q\}}{\{P \land R\}C\{Q \land R\}} \quad \text{where } \text{Mod}(C) \cap \text{Free}(R) = \emptyset
\]

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

\[
\frac{\{P\}C\{Q\}}{\{\exists x. P\}C\{\exists x. Q\}} \quad x \notin \text{Free}(C)
\]

Free(P) is the set of variables occurring freely in P

Copied from the paper by O’Hearn, Reynolds and Yang
More Cycle Free Data-structures

Linked lists.

\[
\text{lseg}(e, f) \iff \begin{cases} e = f & \text{then emp} \\ \exists y. e \mapsto -, y \ast \text{lseg}(y, f) & \text{else} \end{cases}
\]

\[
\text{list}(e) \iff \text{lseg}(e, \text{nil})
\]

\[
\text{lseg}(x, y) \ast \text{lseg}(y, x)
\]

\[
\text{lseg}(x, t) \ast t \mapsto -, y \ast \text{list}(y)
\]

Copied from the paper by O’Hearn, Reynolds and Yang
Examples

Proof of \( \{\text{list}(x)\}y = \text{cons}(b, x)\{\text{list}(y)\} \)
Examples

Proof of \( \{ \text{list}(x) \land x \neq \text{nil} \} t = [x] \{ x \mapsto t \ast \text{list}(t) \} \)
6.820 Fundamentals of Program Analysis
Fall 2015

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.