Separation logic and its application to HLS

Felix Winterstein
13/12/2012
Outline

- Motivating example
- Separation logic
- Extensions of the framework
- Application to HLS
- Conclusion
Motivating example

Pointer-based data structure: Binary tree
Located in heap
Motivating example

Program `rotateTree`

Tree node

data

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>r</td>
<td></td>
</tr>
</tbody>
</table>

Tree node

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>r</td>
<td></td>
</tr>
</tbody>
</table>

Tree node

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>r</td>
<td></td>
</tr>
</tbody>
</table>

Tree node

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>r</td>
<td></td>
</tr>
</tbody>
</table>
Motivating example

Program rotateTree

Tree node

data
l r

Tree node

8
l r

7
l r

9
l r

5
l r

4
l r

2
l r

1
l r

Tree node

6
l r

Program rotateTree

Motivating example
Motivating example

Program rotateTree

Tree node
Motivating example

Program rotateTree

- Left and right sub-trees are disjoint data structures
- Operate on local sub-trees in parallel?
- Program analysis to discover possible parallelisation?
Outline

- Motivating example
- **Separation logic**
- Extensions of the framework
- Application to HLS
- Conclusion
Hoare logic

- Hoare logic: System for proving correctness of computer programs
- (Imperative) program is a sequence of program commands (neglect loops etc. for the moment)
- Hoare triple:
  \[ \{ P \} C \{ Q \} \]
  - Command \( C \)
  - Pre-condition \( P \) for program state
  - Post-condition \( Q \) for program state

- Assignment command:
  \( \{ y = 4 \} x := 3 \{ y = 4 \land x = 3 \} \)
What about pointers?

\[ y \rightarrow 4 \]

\[
\{ y \rightarrow 4 \land x \rightarrow \_ \} [x] := 3 \{ y \rightarrow 4 \land x \rightarrow 3 \}
\]

\[
\{ x \neq y \land y \rightarrow 4 \land x \rightarrow \_ \} [x] := 3 \{ y \rightarrow 4 \land x \rightarrow 3 \}
\]

Not true if \( y = x \)!

OK

- \( x, y \) syntactically unrelated but interdependent -> analysis complicated
- Can we express disjointness explicitly?
Separation logic*

- Memory model with two components: store (stack) and heap
  - Store $s$ maps variables to values
  - Heap $h$ maps locations to values
  - State: $s,h$-pair
- “Assertion $P$ holds for a pair $s,h$”
  $s, h \models P$
- An assertion consists of stack and heap assertions:
  
  $P, Q ::= \text{true} \mid E = F \mid x = \text{nil} \mid \exists y. Q \mid P \land Q \mid P \Rightarrow Q \mid E \rightarrow F \mid emp \mid P \ast Q \mid P \rightarrow \ast Q$

  Atomic formulae & classical logic
  Spatial formulae

  “Separating Conjunction”

  $s, h_0 \models P \quad s, h_1 \models Q \quad h_0 \ast h_1 = h$

  *P. O’Hearn, J. Reynolds, H. Yang, “Local reasoning about programs that alter data structures,” CSL 2001
A simple imperative programming language:
Define Hoare triples for atomic commands (augmented with spatial formulae)

\[
\begin{align*}
\{ \text{true} \} & \quad x := E \quad \{ x = E \} & \text{Stack assignment} \\
\{ E \rightarrow \_ \} & \quad [E] := F \quad \{ E \rightarrow F \} & \text{Heap assignment} \\
\{ E \rightarrow n \} & \quad x := [E] \quad \{ x = n \} & \text{Dereferencing} \\
\{ \text{true} \} & \quad \text{new}(x) \quad \{ x \rightarrow \_ \} & \text{Allocation} \\
\{ x \rightarrow \_ \} & \quad \text{dispose}(x) \quad \{ \text{emp} \} & \text{Deallocation}
\end{align*}
\]
Back to Hoare style (II)

- How to include the “separating conjunction” in our assertions?
- Inference rules

\[
\text{if this holds} \quad \ldots \\
\ldots \text{then I can infer this}
\]

- The “Frame Rule”

\[
\begin{align*}
\{P\} C \{Q\} \\
\{P \ast F\} C \{Q \ast F\}
\end{align*}
\]

s.t. C doesn’t modify any free variables in F

- $F$ is the (unmodified) frame
- Foundation of local reasoning about heap-manipulating commands
- Example:

\[
\{x \rightarrow \_ \ast y \rightarrow 4\} [x] := 3 \{x \rightarrow 3 \ast y \rightarrow 4\} \quad F \equiv y \rightarrow 4
\]
Outline

- Motivating example
- Separation logic
- Extensions of the framework
- Application to HLS
- Conclusion
 Extensions of the framework (I)

- Define assertions for abstract data structures

\[ ls(E, F) \iff (E \neq F \land \exists y'. E \to [n: y'] * ls(y', F)) \lor (E = F \land emp) \]

\[ tree(E) \iff (\exists x', y'. E \to [l: x', r: y'] * tree(x') * tree(y')) \lor (E = nil \land emp) \]

- Symbolic execution*

\[
\begin{align*}
\{ls(x, nil)\} \\
y := x
\end{align*}
\]

\[
\begin{align*}
\{x = y \land ls(x, nil)\} \\
x := [x.n]
\end{align*}
\]

\[
\begin{align*}
\{x = z' \land y \to [n: z'] * ls(z', nil)\} \\
dispose(y)
\end{align*}
\]

\[
\begin{align*}
\{x = z' \land \text{emp} * ls(z', nil)\}
\end{align*}
\]

* J. Berdine, C. Calcagno, P. O’Hearn, “Symbolic execution with separation logic,” APLAS 2005
Extensions of the framework (II)

- **Labelled symbolic execution**
  \[
  \{\langle ls(x,\text{nil})\rangle_{\emptyset}\}
  \]

  l1: \(y := x\)
  \[
  \{x = y \land \langle ls(x,\text{nil})\rangle_{\emptyset}\}
  \]
  \[
  \{x = y \land \langle x, y \rightarrow [n:z']\rangle_{\{l2\}} \ast \langle ls(z',\text{nil})\rangle_{\emptyset}\}
  \]

  l2: \(x := [x.n]\)
  \[
  \{x = z' \land \langle y \rightarrow [n:z']\rangle_{\{l2\}} \ast \langle ls(z',\text{nil})\rangle_{\emptyset}\}
  \]

  l3: \(\text{dispose}(y)\)
  \[
  \{x = z' \land \langle \text{emp}_{\{l2,l3\}} \ast \langle ls(z',\text{nil})\rangle_{\emptyset}\}
  \]

- Keep track of commands' heap footprint
- l2, l3 have a heap-carried dependency
- They cannot be reordered or executed in parallel

---

Outline

- Motivating example
- Separation logic
- Extensions of the framework
- **Application to HLS**
- Conclusion
Application to HLS?

Program rotateTree

Back to our running example
Application to HLS?

Code snippet for rotateTree*

```plaintext
function rotateTree(x) {
    \{ \langle \text{tree}(x) \rangle_{\emptyset} \}\n    if ( x != \text{nil} ) {
        \{ x \neq \text{nil} \land \langle \text{tree}(x) \rangle_{\emptyset} \}\n        l1: x1 := [x.l]
        \{ ... \} 
        l2: x2 := [x.r]
        \{ ... \} 
        l3: [x.l] := x2
        \{ ... \}
        l4: [x.r] := x1
        \{ x2 = y' \land x1 = x' \land x \neq \text{nil} \land \langle x \rightarrow [l: x2, r: x1] \rangle_{\{l1,l2,l3,l4\}} \ast \langle \text{tree}(x') \rangle_{\emptyset} \ast \langle \text{tree}(y') \rangle_{\emptyset} \}\n        l5: rotateTree(x1)
        \{ x2 = y' \land x1 = x' \land x \neq \text{nil} \land \langle x \rightarrow [l: x2, r: x1] \rangle_{\{l1,l2,l3,l4\}} \ast \langle \text{tree}(x') \rangle_{\{l5\}} \ast \langle \text{tree}(y') \rangle_{\emptyset} \}\n        l6: rotateTree(x2)
        \{ x2 = y' \land x1 = x' \land x \neq \text{nil} \land \langle x \rightarrow [l: x2, r: x1] \rangle_{\{l1,l2,l3,l4\}} \ast \langle \text{tree}(x') \rangle_{\{l5\}} \ast \langle \text{tree}(y') \rangle_{\{l6\}} \}\n    }
}
```

No heap-carried dependency between the two recursive calls
-> they can execute in parallel!

*M. Raza, C. Calcagno, P. Gardner, “Automatic parallelization with separation logic,” POPL 2009*
Conclusion

- Separation logic is an extension of Hoare logic
- Disjointness of heap cells is expressed explicitly
- Local reasoning about in-place updates in heap data structures
- Symbolic execution framework for automatic program verification and parallelisation
- Interesting for HLS?
  - Automatic analysis and parallelisation of pointer-based programs
  - Memory localisation: logical disjointness <-> physical disjointness
- Interesting in other contexts?
  - Concurrent separation logic, ownership, reasoning about shared resources
  - Program analysis for multi-core architectures?