Separation logic and its application to HLS

Felix Winterstein

13/12/2012

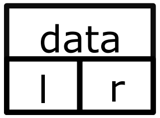
Outline

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

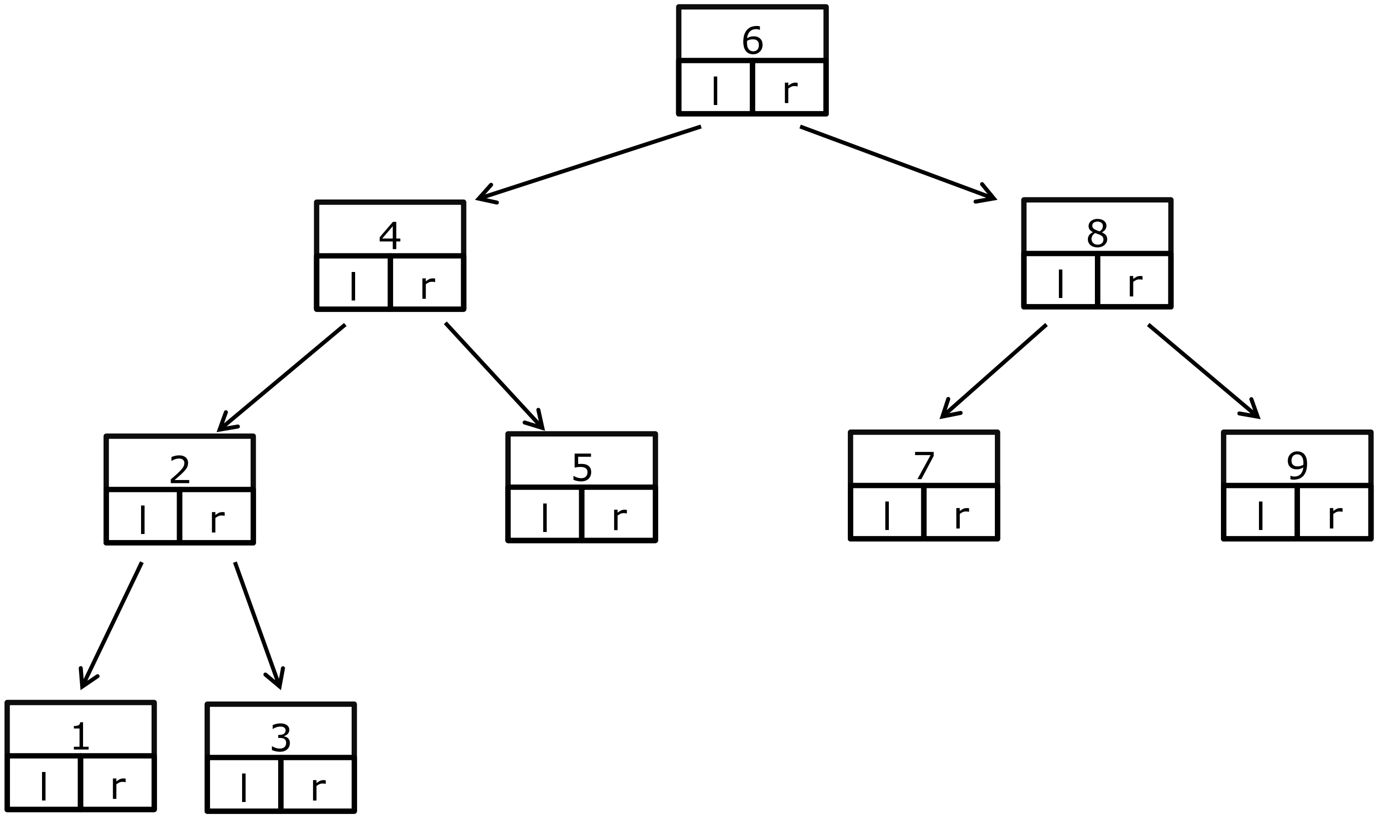
Motivating example

Pointer-based data structure: Binary tree

Located in heap

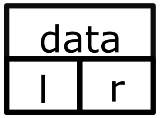


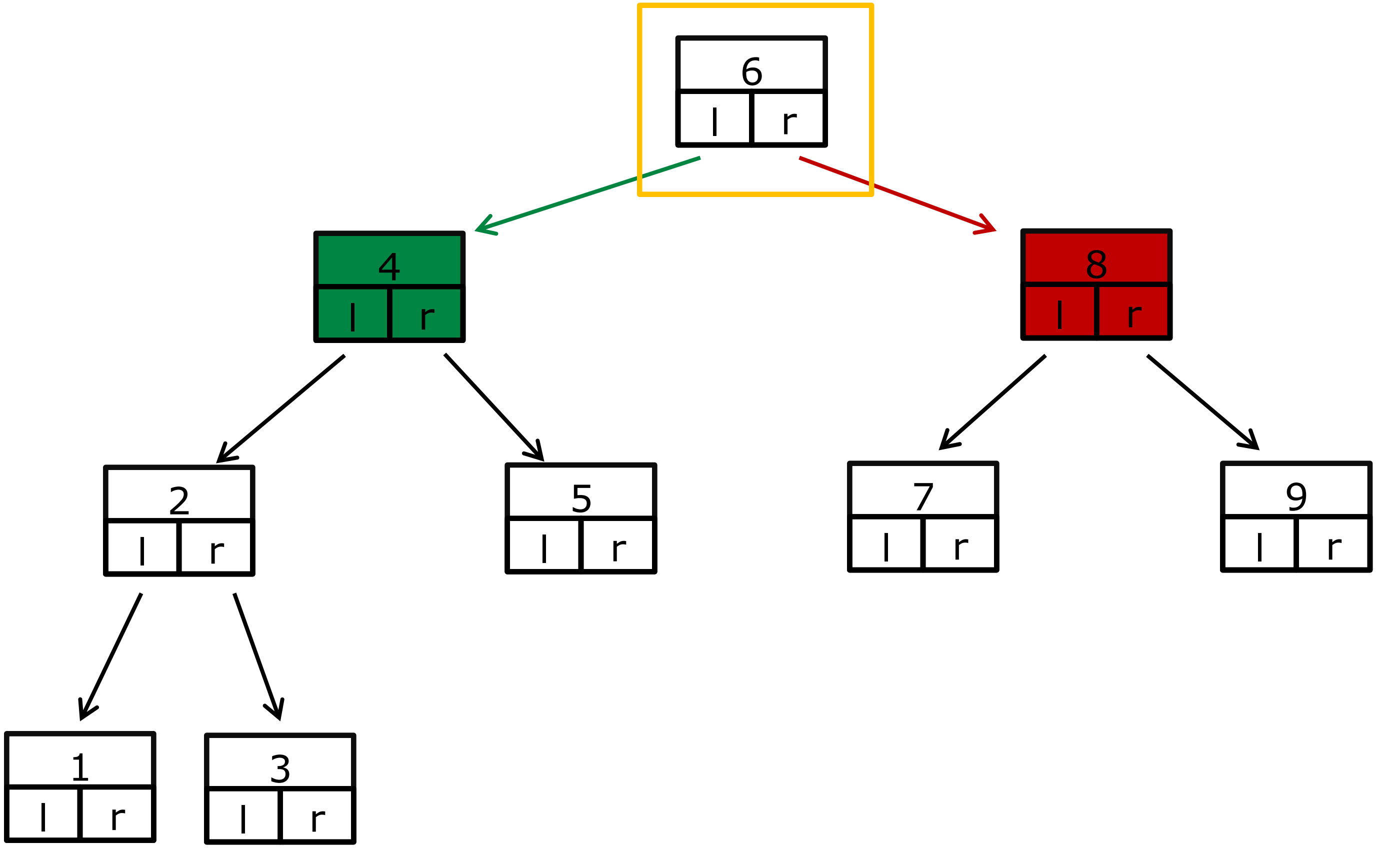
Tree node



Motivating example

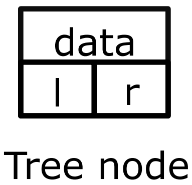
Program rotateTree

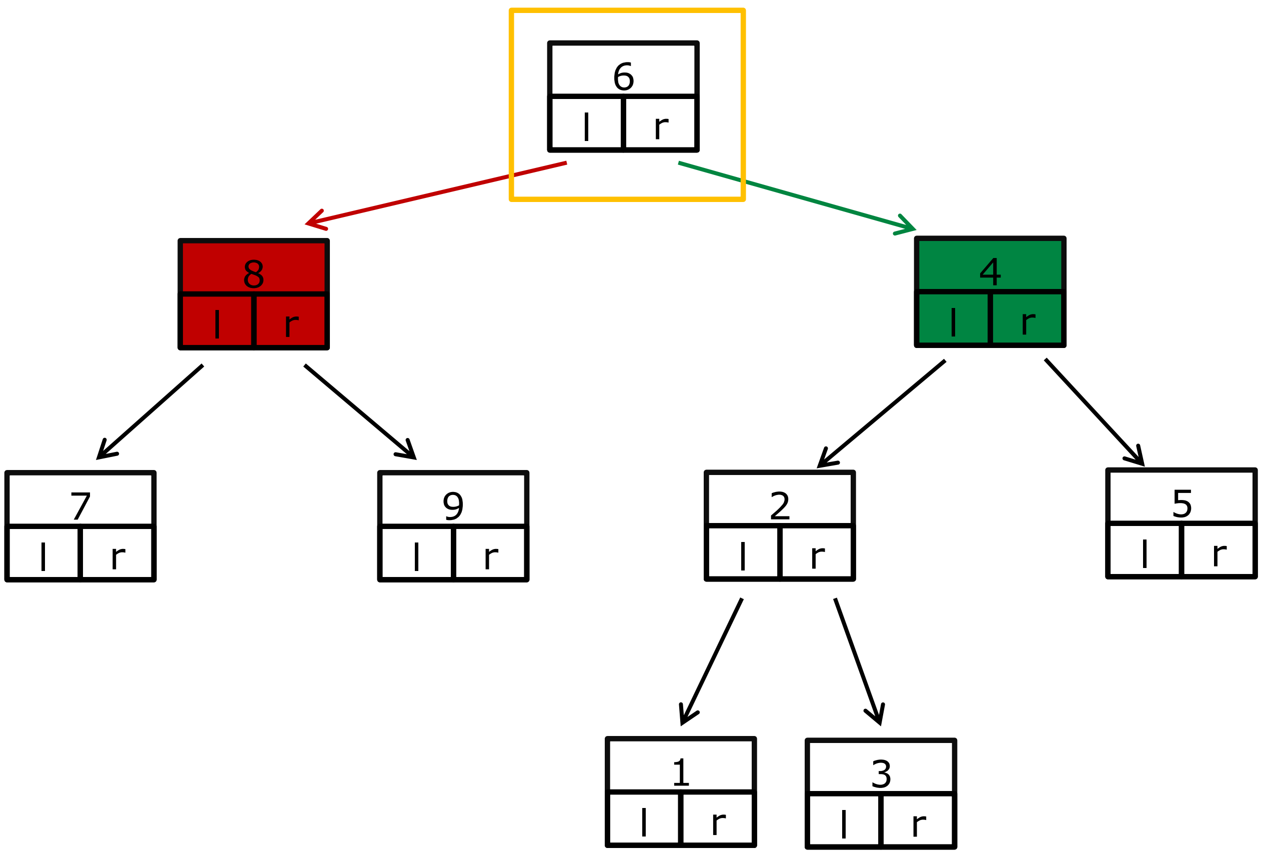




Motivating example

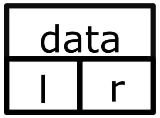
Program rotateTree



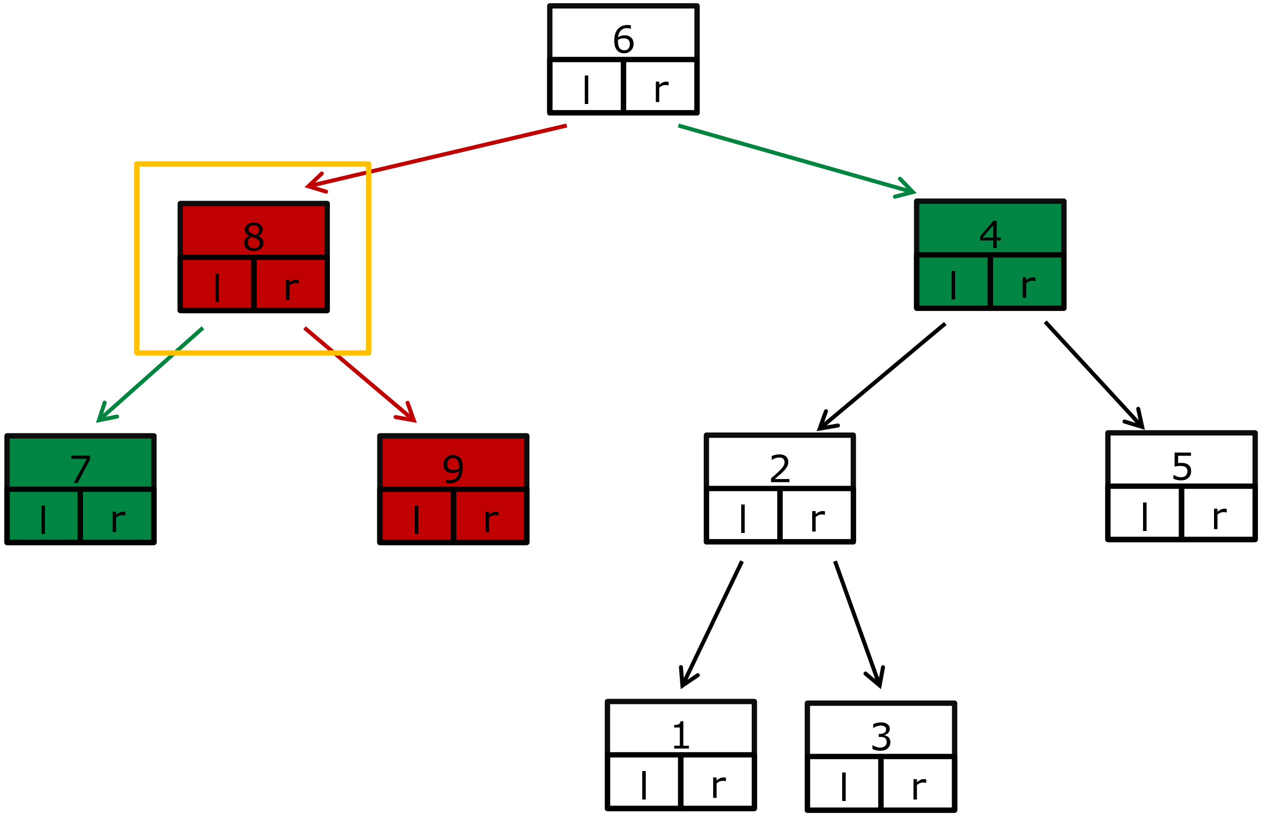


Motivating example

Program rotateTree



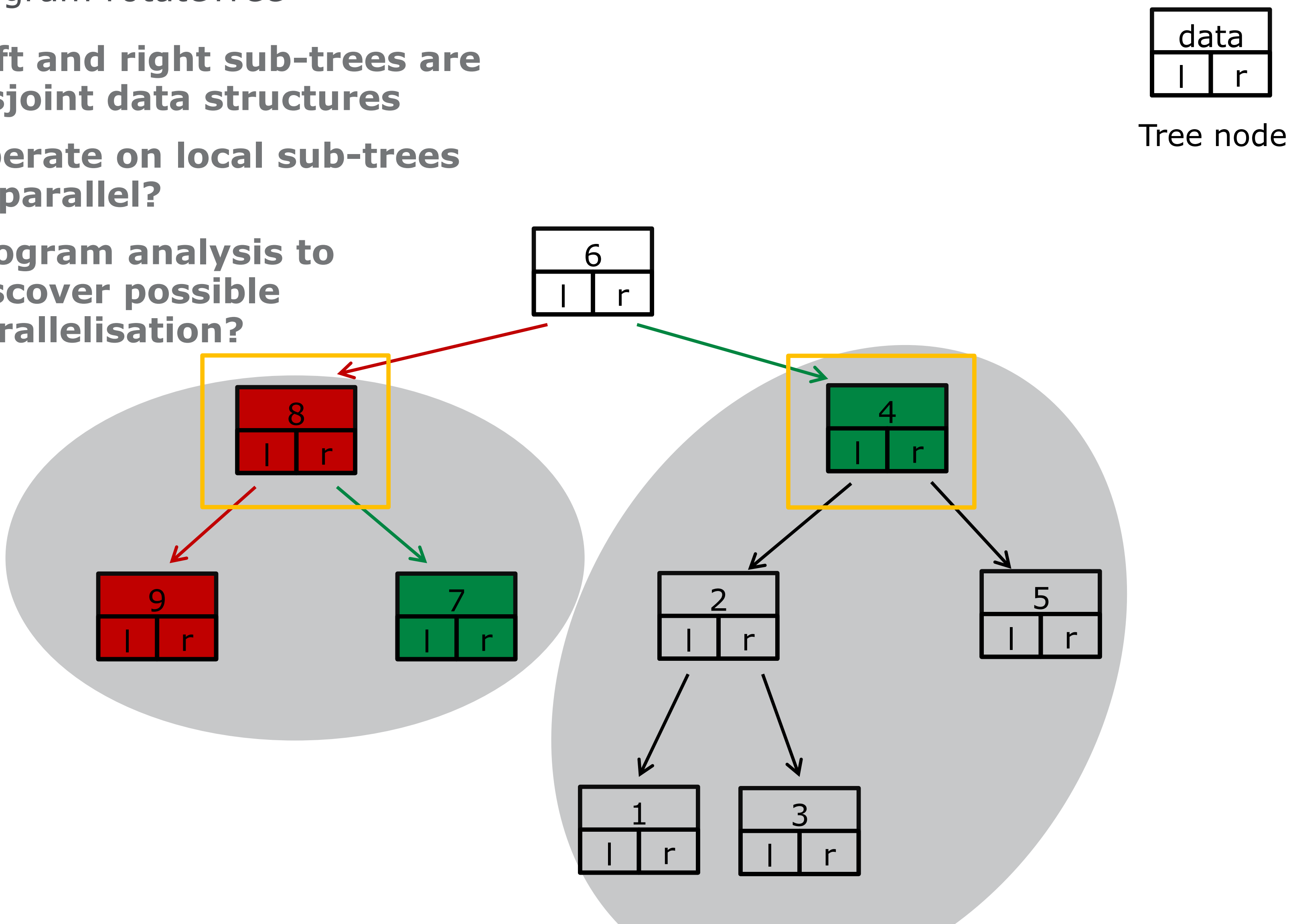
Tree node



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:

$$\{\boldsymbol{P}\}\ \boldsymbol{C}\ \{\boldsymbol{Q}\}$$

  - Command *C*
  - Pre-condition *P* for program state
  - Post-condition *Q* for program state

- Assignment command:

$$\{\boldsymbol{y} = \boldsymbol{4}\}\ \boldsymbol{x} \coloneqq \boldsymbol{3}\ \{\boldsymbol{y} = \boldsymbol{4} \wedge \boldsymbol{x} = \boldsymbol{3}\}$$

Hoare logic (II)

- What about pointers?

$y \rightarrow \boxed{4}$

$$\{y \rightarrow 4 \wedge x \rightarrow \_\}\ [x] := 3\ \{y \rightarrow 4 \wedge x \rightarrow 3\}$$ Not true if $y=x$!

$$\{x \neq y \wedge y \rightarrow 4 \wedge x \rightarrow \_\}\ [x] := 3\ \{y \rightarrow 4 \wedge x \rightarrow 3\}$$ 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*"

$$s, h \vDash P$$

- An assertion consists of stack and heap assertions:

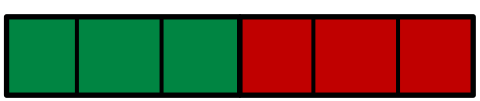
$$P, Q ::= true \mid E = F \mid x = nil \mid \exists y.\, Q \mid$$ Atomic formulae & classical logic

$$P \wedge Q \mid P \Rightarrow Q \mid E \rightarrow F \mid \ldots$$

$$emp \mid P * Q \mid P \mathrel{-\!\!*} Q$$ Spatial formulae

Just an example, list not complete!

"Separating Conjunction"



$s, h_0 \vDash P$ $s, h_1 \vDash Q$ $h_0 * h_1 = h$

* P. O'Hearn, J. Reynolds, H. Yang, "Local reasoning about programs that alter data structures," CSL 2001

Back to Hoare style (I)

A simple imperative programming language:

Define Hoare triples for atomic commands (augmented with spatial formulae)

$\{\boldsymbol{true}\}\ \boldsymbol{x} \coloneqq \boldsymbol{E}\ \{\boldsymbol{x} = \boldsymbol{E}\}$	Stack assignment
$\{\boldsymbol{E} \to \_\}\ [\boldsymbol{E}] \coloneqq \boldsymbol{F}\ \{\boldsymbol{E} \to \boldsymbol{F}\}$	Heap assignment
$\{\boldsymbol{E} \to \boldsymbol{n}\}\ \boldsymbol{x} \coloneqq [\boldsymbol{E}]\ \{\boldsymbol{x} = \boldsymbol{n}\}$	Dereferencing
$\{\boldsymbol{true}\}\ \boldsymbol{new}(\boldsymbol{x})\ \{\boldsymbol{x} \to \_\}$	Allocation
$\{\boldsymbol{x} \to \_\}\ \boldsymbol{dispose}(\boldsymbol{x})\ \{\boldsymbol{emp}\}$	Deallocation

Back to Hoare style (II)

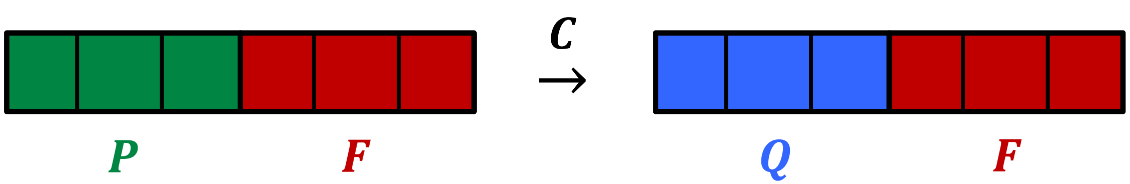
- How to include the “separating conjunction” in our assertions?
- Inference rules

$$\frac{\textit{if this holds } \ldots}{\ldots \textit{ then I can infer this}}$$

- The “Frame Rule“

$$\frac{\{P\}\; C\; \{Q\}}{\{P * F\}\; C\; \{Q * F\}}$$

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:

$$\{x \to \_ * y \to 4\}\; [x] \coloneqq 3\; \{x \to 3 * y \to 4\} \qquad F \equiv y \to 4$$

Outline

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

Extensions of the framework (I)

- Define assertions for abstract data structures

$$\boldsymbol{ls}(E,F) \Leftrightarrow \big(E \neq F \wedge \exists y'.\, E \rightarrow [n{:}\, y'] * \boldsymbol{ls}(y',F)\big) \vee (E = F \wedge \boldsymbol{emp})$$

$$\boldsymbol{tree}(E) \;\Leftrightarrow \big(\exists x', y'.\, E \rightarrow [l{:}\, x', r{:}\, y'] * \boldsymbol{tree}(x') * \boldsymbol{tree}(y')\big) \vee (E = \boldsymbol{nil} \wedge \boldsymbol{emp})$$

- Symbolic execution*

$\{\boldsymbol{ls}(x,\boldsymbol{nil})\}$

y := x

$\{x = y \wedge \boldsymbol{ls}(x,\boldsymbol{nil})\}$

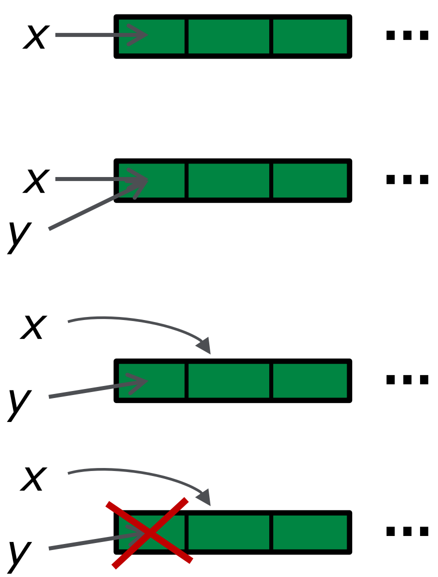
$\{x = y \wedge x, y \rightarrow [n{:}\, z'] * \boldsymbol{ls}(z',\boldsymbol{nil})\}$

x := [x.n]

$\{x = z' \wedge y \rightarrow [n{:}\, z'] * \boldsymbol{ls}(z',\boldsymbol{nil})\}$

dispose(y)

$\{x = z' \wedge \boldsymbol{emp} * \boldsymbol{ls}(z',\boldsymbol{nil})\}$



* J. Berdine, C. Calcagno, P. O'Hearn, "Symbolic execution with separation logic," APLAS 2005

Extensions of the framework (II)

- Labelled symbolic execution*

$\{\langle \boldsymbol{ls}(\boldsymbol{x},\boldsymbol{nil})\rangle_{\{\}}\}$

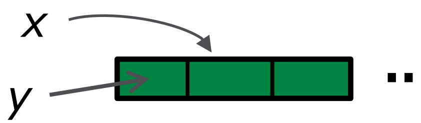
l1: y := x

$\{\boldsymbol{x} = \boldsymbol{y} \wedge \langle \boldsymbol{ls}(\boldsymbol{x},\boldsymbol{nil})\rangle_{\{\}}\}$

$\{\boldsymbol{x} = \boldsymbol{y} \wedge \langle \boldsymbol{x},\boldsymbol{y} \to [\boldsymbol{n}\colon \boldsymbol{z}']\rangle_{\{\}} * \langle \boldsymbol{ls}(\boldsymbol{z}',\boldsymbol{nil})\rangle_{\{\}}\}$

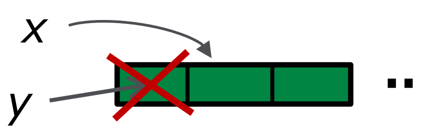
l2: x := [x.n]

$\{\boldsymbol{x} = \boldsymbol{z}' \wedge \langle \boldsymbol{y} \to [\boldsymbol{n}\colon \boldsymbol{z}']\rangle_{\{l2\}} * \langle \boldsymbol{ls}(\boldsymbol{z}',\boldsymbol{nil})\rangle_{\{\}}\}$



l3: dispose(y)

$\{\boldsymbol{x} = \boldsymbol{z}' \wedge \langle \boldsymbol{emp}\rangle_{\{l2,l3\}} * \langle \boldsymbol{ls}(\boldsymbol{z}',\boldsymbol{nil})\rangle_{\{\}}\}$



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

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

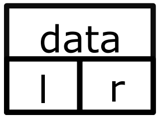
Outline

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

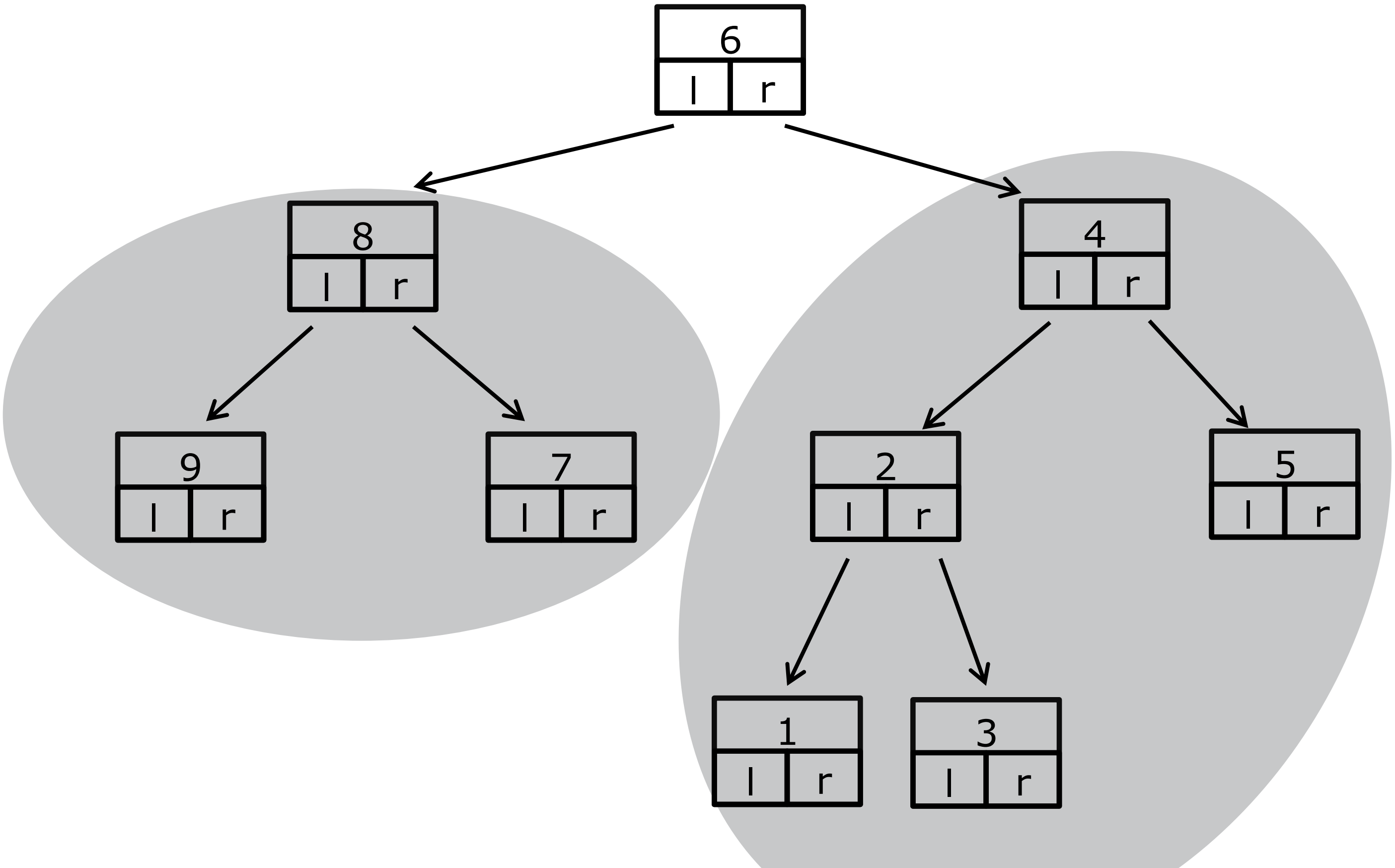
Application to HLS ?

Program rotateTree

Back to our running example



Tree node



Application to HLS?

Code snippet for rotateTree*

function rotateTree(x)

$\{\langle \boldsymbol{tree}(\boldsymbol{x})\rangle_{\{\}}\}$

if (x!=nil) {

$\{\boldsymbol{x} \neq \boldsymbol{nil} \wedge \langle \boldsymbol{tree}(\boldsymbol{x})\rangle_{\{\}}\}$

l1: x1 := [x.l]

$\{\dots\}$

l2: x2 := [x.r]

$\{\dots\}$

l3: [x.l] := x2

$\{\dots\}$

l4: [x.r] := x1

$\{\boldsymbol{x2} = \boldsymbol{y}' \wedge \boldsymbol{x1} = \boldsymbol{x}' \wedge \boldsymbol{x} \neq \boldsymbol{nil} \wedge \langle \boldsymbol{x} \rightarrow [\boldsymbol{l}: \boldsymbol{x2}, \boldsymbol{r}: \boldsymbol{x1}]\rangle_{\{l1,l2,l3,l4\}} * \langle \boldsymbol{tree}(\boldsymbol{x}')\rangle_{\{\}} * \langle \boldsymbol{tree}(\boldsymbol{y}')\rangle_{\{\}}\}$

l5: rotateTree(x1)

$\{\boldsymbol{x2} = \boldsymbol{y}' \wedge \boldsymbol{x1} = \boldsymbol{x}' \wedge \boldsymbol{x} \neq \boldsymbol{nil} \wedge \langle \boldsymbol{x} \rightarrow [\boldsymbol{l}: \boldsymbol{x2}, \boldsymbol{r}: \boldsymbol{x1}]\rangle_{\{l1,l2,l3,l4\}} * \langle \boldsymbol{tree}(\boldsymbol{x}')\rangle_{\{l5\}} * \langle \boldsymbol{tree}(\boldsymbol{y}')\rangle_{\{\}}\}$

l6: rotateTree(x2)

$\{\boldsymbol{x2} = \boldsymbol{y}' \wedge \boldsymbol{x1} = \boldsymbol{x}' \wedge \boldsymbol{x} \neq \boldsymbol{nil} \wedge \langle \boldsymbol{x} \rightarrow [\boldsymbol{l}: \boldsymbol{x2}, \boldsymbol{r}: \boldsymbol{x1}]\rangle_{\{l1,l2,l3,l4\}} * \langle \boldsymbol{tree}(\boldsymbol{x}')\rangle_{\{l5\}} * \langle \boldsymbol{tree}(\boldsymbol{y}')\rangle_{\{l6\}}\}$

}

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?