Automated Verification via Separation Logic

Cristina David

National University of Singapore

Advisor: Prof. Chin Wei-Ngan

Seoul National University, October 2009
Overview

• Verification of shape, size and bag props
  • Automated Verification of Shape, Size and Bag Properties, W.N.Chin, C.David, H.H.Nguyen, S.Qin, ICECCS'07
  • Automated Verification of Shape and Size Properties via Separation Logic, H.H.Nguyen, C.David, S.Qin, W.N.Chin, VMCAI'07
  • Multiple Pre/Post Specifications for Heap-Manipulating Methods, W.N.Chin, C.David, H.H.Nguyen, S.Qin, HASE'07

• Verification of object-oriented programs
  • Enhancing Modular OO Verification with Separation Logic, W.N.Chin, C.David, H.H.Nguyen, S.Qin, POPL'08

• Verification of exception-handling programs
  • Translation and Optimization for a Core Calculus with Exceptions, C.David, C.Gherghina, W.N.Chin, PEPM'09
Goal

- Verifying mutable data structures with invariants involving size properties
Challenges

• Strong updates in the presence of aliasing

• Entailment checking with inductive predicates
Separation logic

- **Foundations:**

- Extension to Hoare logic to reason about shared mutable data structures
Examples (Reynolds LICS’02)

\[ p_1 \ast p_2: \text{the heap can be split into two disjoint parts (} p_1 \text{ holds for one part and } p_2 \text{ holds for the other) } \]

\[ x \mapsto 3, y \]

\[ x \mapsto 3, y \ast y \mapsto 3, z \]
Local reasoning in separation logic

Reasoning about command $c$ involves only parts of the heap that are actually used by $c$. 

\[
\frac{\{p\} c \{q\}}{\{p \ast r\} c \{q \ast r\}}
\]
Our work

• Size properties
  • Data structure invariants involving arithmetic constraints.
  • Sortedness, length, height-balanced, etc.

• User-defined inductive predicates
  • Data structures are user custom design
Example predicates

Singly-linked list:
\[ \text{ll}(n) \equiv self=\text{null} \land n=0 \lor self::\text{node}(\_ ,q) \ast q::\text{ll}(n-1) \]

Non-empty sorted list:
\[ \text{sortl}(n,\text{min}) \equiv self::\text{node}(\text{min},\text{null}) \land n=1 \lor self::\text{node}(\text{min},q) \ast q::\text{sortl}(n-1,k) \land \text{min} \leq k \]

Singly-linked list with bag of values:
\[ \text{llB}(B) \equiv self=\text{null} \land B=\{\} \lor self::\text{node}(v,q) \ast q::\text{llB}(B_1) \land B = B_1 \cup \{v\} \]
What can be written?

- **Heap part**
  - Describes shapes using separation logic

- **Pure part**
  - Size properties (arithmetic constraints)
  - Pointer constraints
  - Bag constraints

- **self is a pointer from which every node is reachable**
Verification

- Methods are annotated with pre and postconditions
- Loop invariants are supplied
- Entailment checks
  - Precondition at call site
  - Postcondition at end of method
Insert Into a Sorted List

define insert(node x, node vn)
requires x points to a sorted list with length n and vn points to a disjoint node
ensures res points to a sorted list with length n+1
{
    if (vn.val <= x.val) {
        vn.next = x;
        return vn;
    } else if (x.next = null) then {
        x.next = vn; vn.next = null;
        return x;
    } else {
        x.next = insert(x.next, vn);
        return x;
    }
}
Insert Into a Sorted List

node insert(node x, node vn)
requires $x::\text{sortl}(n, \text{sm}) \land vn::\text{node}(v, \_)$
ensures $\text{res}::\text{sortl}(n + 1, \text{min}(v, \text{sm}))$

{
    if (vn.val \leq x.val) {
        vn.next = x;
        return vn;
    }
    else if (x.next = null) then {
        x.next = vn; vn.next = null;
        return x;
    }
    else {
        x.next = insert(x.next, vn);
        return x;
    }
}
Sorted List with Bag of Values

\[ \text{sortB}(B) \equiv \text{self} = \text{null} \land B = \{\} \]
\[ \lor \text{self}::\text{node}(v, q) \land q::\text{sortB}(B_1) \land B = B_1 \cup \{v\} \land \forall a. a \in B_1 \Rightarrow v \leq a \]
Insert Into a Sorted List

node insert(node x, node vn)

requires x::sortB\langle B \rangle \ast vn::node\langle v, _ \rangle

ensures res::sortl\langle B \cup \{v\} \rangle

{ 
   if (vn.val \leq x.val) {
      vn.next = x;
      return vn; }

   else if (x.next=null) then {
      x.next = vn; vn.next = null;
      return x; }

   else {
      x.next = insert(x.next, vn);
      return x; }

}
Entailment

\[ \Delta_1 \vdash \Delta_2 \ast \Delta_3 \]

- \( \Delta_1 \) “provides” all heap nodes “requested” by \( \Delta_2 \)
- Remaining nodes are kept in \( \Delta_3 \)
Entailment

- **Matching**
  - Aliased data nodes/predicates are matched and their components/arguments equated

- **Unfolding**
  - Replaces a predicate on LHS by its definition to match a node on the RHS

- **Folding**
  - Recursive entailment call to check predicate on RHS
Unfolding and Matching

\[
x::ll\langle n \rangle \land n > 1 \vdash \\
\exists r, m \cdot x::node\langle _, r \rangle * r::ll\langle m \rangle \land m > 0
\]

- **Unfolding**
  \[
  \exists q \cdot x::node\langle _, q \rangle * q::ll\langle n-1 \rangle \land n > 1 \vdash \\
  \exists r, m \cdot x::node\langle _, r \rangle * r::ll\langle m \rangle \land m > 0
  \]

- **Matching**
  \[
  q::ll\langle n-1 \rangle \land n > 1 \land q = r \vdash \exists m \cdot r::ll\langle m \rangle \land m > 0
  \]
Entailment $\Delta_1 \vdash \Delta_2 \ast \Delta_3$

Entailment between separation formulae

1. Successively remove heap nodes from $\Delta_2$

2. When $\Delta_2$ is pure, the heap formula in $\Delta_1$ is approximated by function $\text{XPure}$

Entailment between pure formulae
Approximation

• Each predicate is approximated by a pure constraint.

\[
X_{Pure}(x::node(\_,\_) * y::node(\_,\_))
\]

\[
= \exists i,j. (x=i \land i>0 \land y=j \land j>0 \land i \neq j)
\]

\[
= x \neq y
\]

\[
X_{Pure}(x::ll(\langle n\rangle))
\]

\[
= \exists i. (self=0 \land n=0 \lor self=i \land i>0 \land n>0)
\]

\[
= (self=0 \land n=0) \lor (self \neq 0 \land n>0)
\]
## Experiments

<table>
<thead>
<tr>
<th>Programs</th>
<th>Without size/bag</th>
<th>Omega Calculator</th>
<th>Isabelle Prover</th>
<th>MONA Prover</th>
<th>Isabelle Prover</th>
<th>MONA Prover</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linked List</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>delete</td>
<td>0.02</td>
<td>0.09</td>
<td>8.35</td>
<td>0.33</td>
<td>5.00</td>
<td>0.34</td>
</tr>
<tr>
<td>reverse</td>
<td>0.02</td>
<td>0.07</td>
<td>3.28</td>
<td>0.21</td>
<td>3.01</td>
<td>0.20</td>
</tr>
<tr>
<td>Circular Linked List</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>delete (first)</td>
<td>0.01</td>
<td>0.09</td>
<td>5.46</td>
<td>0.26</td>
<td>7.17</td>
<td>0.40</td>
</tr>
<tr>
<td>count</td>
<td>0.04</td>
<td>0.16</td>
<td>14.99</td>
<td>0.71</td>
<td>21.01</td>
<td>2.29</td>
</tr>
<tr>
<td>Doubly Linked List</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>append</td>
<td>0.05</td>
<td>0.16</td>
<td>28.18</td>
<td>0.83</td>
<td>23.73</td>
<td>0.93</td>
</tr>
<tr>
<td>flatten (from tree)</td>
<td>0.08</td>
<td>0.30</td>
<td>158.3</td>
<td>6.65</td>
<td>55.78</td>
<td>2.03</td>
</tr>
<tr>
<td>Sorted List</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>delete</td>
<td>0.02</td>
<td>0.13</td>
<td>34.09</td>
<td>26.68</td>
<td>51.39</td>
<td>0.60</td>
</tr>
<tr>
<td>insertion_sort</td>
<td>0.07</td>
<td>0.27</td>
<td>41.17</td>
<td>18.22</td>
<td>27.34</td>
<td>0.73</td>
</tr>
<tr>
<td>selection_sort</td>
<td>0.10</td>
<td>0.41</td>
<td>79.08</td>
<td>20.62</td>
<td>221.7</td>
<td>1.10</td>
</tr>
<tr>
<td>bubble_sort</td>
<td>0.16</td>
<td>0.64</td>
<td>358.7</td>
<td>9.36</td>
<td>221.2</td>
<td>2.84</td>
</tr>
<tr>
<td>merge_sort</td>
<td>0.11</td>
<td>0.61</td>
<td>342.9</td>
<td>105.1</td>
<td>150.1</td>
<td>21.75</td>
</tr>
<tr>
<td>quick_sort</td>
<td>0.19</td>
<td>0.59</td>
<td>642.0 (out of memory)</td>
<td>failed</td>
<td>3.40</td>
<td></td>
</tr>
<tr>
<td>Binary Search Tree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>insert</td>
<td>0.03</td>
<td>0.20</td>
<td>failed</td>
<td>11.92</td>
<td>99.57</td>
<td>0.95</td>
</tr>
<tr>
<td>delete</td>
<td>0.06</td>
<td>0.38</td>
<td>97.5</td>
<td>6.86</td>
<td>943.5</td>
<td>3.03</td>
</tr>
<tr>
<td>Priority Queue Heap</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>insert</td>
<td>0.15</td>
<td>0.45</td>
<td>520.8</td>
<td>41.55</td>
<td>416.2</td>
<td>6.45</td>
</tr>
<tr>
<td>delete_max</td>
<td>0.55</td>
<td>7.17</td>
<td>failed</td>
<td>290.7</td>
<td>failed</td>
<td>626.1</td>
</tr>
<tr>
<td>AVL Tree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>insert</td>
<td>1.04</td>
<td>5.06</td>
<td>failed</td>
<td>36.02</td>
<td>1973</td>
<td>7.38</td>
</tr>
<tr>
<td>Red-Black Tree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>insert</td>
<td>0.44</td>
<td>1.53</td>
<td>2992</td>
<td>352.4</td>
<td>failed</td>
<td>392.8</td>
</tr>
<tr>
<td>delete</td>
<td>8.29</td>
<td>5.98</td>
<td>24335</td>
<td>out of memory</td>
<td>42416</td>
<td>1691</td>
</tr>
</tbody>
</table>
## Experiments

<table>
<thead>
<tr>
<th>Programs</th>
<th>Without size/bag</th>
<th>Omega Calculator</th>
<th>Isabelle Prover</th>
<th>MONA Prover</th>
<th>Isabelle Prover</th>
<th>MONA Prover</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Linked List</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>delete</td>
<td>0.02</td>
<td>0.09</td>
<td>8.35</td>
<td>0.33</td>
<td>5.00</td>
<td>0.34</td>
</tr>
<tr>
<td>reverse</td>
<td>0.02</td>
<td>0.07</td>
<td>3.28</td>
<td>0.21</td>
<td>3.01</td>
<td>0.20</td>
</tr>
<tr>
<td><strong>Circular Linked List</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>delete (first)</td>
<td>0.01</td>
<td>0.09</td>
<td>5.46</td>
<td>0.26</td>
<td>7.17</td>
<td>0.40</td>
</tr>
<tr>
<td>count</td>
<td>0.04</td>
<td>0.16</td>
<td>14.99</td>
<td>0.71</td>
<td>21.01</td>
<td>2.29</td>
</tr>
<tr>
<td><strong>Doubly Linked List</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>append</td>
<td>0.05</td>
<td>0.16</td>
<td>28.18</td>
<td>0.83</td>
<td>23.73</td>
<td>0.93</td>
</tr>
<tr>
<td>flatten (from tree)</td>
<td>0.08</td>
<td>0.30</td>
<td>158.3</td>
<td>6.65</td>
<td>55.78</td>
<td>2.03</td>
</tr>
<tr>
<td><strong>Sorted List</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>delete</td>
<td>0.02</td>
<td>0.13</td>
<td>34.09</td>
<td>26.68</td>
<td>51.39</td>
<td>0.60</td>
</tr>
<tr>
<td>insertion_sort</td>
<td>0.07</td>
<td>0.27</td>
<td>41.17</td>
<td>18.22</td>
<td>27.34</td>
<td>0.73</td>
</tr>
<tr>
<td>selection_sort</td>
<td>0.10</td>
<td>0.41</td>
<td>79.08</td>
<td>20.62</td>
<td>221.7</td>
<td>1.10</td>
</tr>
<tr>
<td>bubble_sort</td>
<td>0.16</td>
<td>0.64</td>
<td>358.7</td>
<td>9.36</td>
<td>221.2</td>
<td>2.84</td>
</tr>
<tr>
<td>merge_sort</td>
<td>0.11</td>
<td>0.61</td>
<td>342.9</td>
<td>105.1</td>
<td>150.1</td>
<td>21.75</td>
</tr>
<tr>
<td>quick_sort</td>
<td>0.19</td>
<td>0.59</td>
<td>642.0</td>
<td>out of memory</td>
<td>failed</td>
<td>3.40</td>
</tr>
<tr>
<td><strong>Binary Search Tree</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>insert</td>
<td>0.03</td>
<td>0.20</td>
<td>failed</td>
<td>11.92</td>
<td>99.57</td>
<td>0.95</td>
</tr>
<tr>
<td>delete</td>
<td>0.06</td>
<td>0.38</td>
<td>97.5</td>
<td>6.86</td>
<td>943.5</td>
<td>3.03</td>
</tr>
<tr>
<td><strong>Priority Queue Heap</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>insert</td>
<td>0.15</td>
<td>0.45</td>
<td>520.8</td>
<td>41.55</td>
<td>416.2</td>
<td>6.45</td>
</tr>
<tr>
<td>delete_max</td>
<td>0.55</td>
<td>7.17</td>
<td>failed</td>
<td>290.7</td>
<td>failed</td>
<td>626.1</td>
</tr>
<tr>
<td><strong>AVL Tree</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>insert</td>
<td>1.04</td>
<td>5.06</td>
<td>failed</td>
<td>36.02</td>
<td>1973</td>
<td>7.38</td>
</tr>
<tr>
<td><strong>Red-Black Tree</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>insert</td>
<td>0.44</td>
<td>1.53</td>
<td>2992</td>
<td>352.4</td>
<td>failed</td>
<td>392.8</td>
</tr>
<tr>
<td>delete</td>
<td>8.29</td>
<td>5.98</td>
<td>24335</td>
<td>out of memory</td>
<td>42416</td>
<td>1691</td>
</tr>
</tbody>
</table>
Verification of OO programs

- Must support behavioral subtyping.
- Must support class inheritance.
- Must support casting.
- Good to support class invariants.
- Good to support super/direct calls.

- precision
- efficiency (minimize code re-verification)
Overview

• Verification of shape, size and bag props

• Verification of OO programs
  - Enhanced Spec Subsumption
  - Static & Dynamic Specs
  - Key Principles
Behavioral subtyping

- Liskov's Substitutivity Principle (1988):
  - an object of a subclass can always be passed to a location where an object of its superclass is expected
Spec subsumption

class A {
    t mn(..) where preA $\rightarrow\rightarrow\rightarrow$ postA {...}
}
class B extends A {
    t mn(..) where preB $\rightarrow\rightarrow\rightarrow$ postB {...}
}

Spec (preB $\rightarrow\rightarrow\rightarrow$ postB) is a subtype of (preA $\rightarrow\rightarrow\rightarrow$ postA) if:

\[
\text{type}(\text{this}) \preceq B \quad \text{old}(\text{preA}) \wedge \text{postB} \Rightarrow \text{postA}
\]

(preB $\rightarrow\rightarrow\rightarrow$ postB) $\preceq_B$ (preA $\rightarrow\rightarrow\rightarrow$ postA)

Contravariance

Covariance

Castagna('95)

Leavens&Naumann('06)
Enhanced spec subsumption

- With the help of frame rule

\[ \vdash \{P\} \subseteq \{Q\} \]
\[ \vdash \{P \Delta\} \subseteq \{Q \Delta\} \]

Spec \((\text{preB} \rightarrow \text{postB})\) is a subtype of \((\text{preA} \rightarrow \text{postA})\) if:

\[ \text{preA} \land \text{type(this)} \ll B \Rightarrow \text{preB} \Delta \]
\[ \Delta \star \text{postB} \Rightarrow \text{postA} \]

\[ (\text{preB} \rightarrow \text{postB}) \ll_B (\text{preA} \rightarrow \text{postA}) \]
Static and dynamic specs: example

```java
class Cnt

int val;

Cnt(int v) {this.val = v}

void tick() {this.val = this.val + 1}

int get() {this.val}

void set(int x) {this.val = x}

}

class FastCnt extends Cnt {

FastCnt(int v) {this.val = v}

void tick() {this.val = this.val + 2}

}

class PosCnt extends Cnt inv this.val ≥ 0 {

PosCnt(int v) {this.val = v}

void set(int x) {if x > 0 then this.val = x else error()}

}
```

\[\text{static this::Cnt<n>}$ \rightarrow \text{this::Cnt<n+1>}$\]
class Cnt { int val;
    Cnt(int v) {this.val:=v}
    void tick() {this.val:=this.val+1}
    int get() {this.val}
    void set(int x) {this.val:=x} }

class FastCnt extends Cnt {
    FastCnt(int v) {this.val:=v}
    void tick() {this.val:=this.val+2} }

class PosCnt extends Cnt { inv this.val≥0 {
    PosCnt(int v) {this.val:=v}
    void set(int x) {if x≥0 then this.val:=x else error()} } }
Static and dynamic spec

A static spec:
- describes just a single method
- used for statically-dispatched calls (e.g. super/direct)
- can be very precise

A dynamic spec:
- describes a method and its overriding methods
- used for dynamically-dispatched calls
- less precise
Key principles

- Static spec must be given for each new method.
- Code verification is done only for static spec.
- Dynamic spec is either given or derived.
- Subsumption relations:

```java
class A {
    // defines mn
}
```

\[ \text{Static-Spec}(A.mn) \prec \text{Dyn-Spec}(A.mn) \]

\text{Min. re-verification}
Key principles

- Static spec must be given for each new method.
- Code verification is done only for static spec.
- Dynamic spec is either given or derived.
- Subsumption relations:

```java
class A {
    // defines mn
}
class B extends A {
    // overrides mn
}
```

Static-Spec(A.mn) :- Dynamic-Spec(A.mn)

- min. re-verification

Dynamic-Spec(B.mn)

- ensures behavioral subtyping
Key principles

- Static spec must be given for each new method.
- Code verification is done only for static spec.
- Dynamic spec is either given or derived.
- Subsumption relations:

```
class A {
    // defines mn
}
class B extends A {
    // overrides mn
}
class C extends A {
    // inherits mn
}
```
Initial experiment

- **Code Verification > Spec Subsumption Checking**

<table>
<thead>
<tr>
<th>Class</th>
<th>Our system (secs)</th>
<th>With code re-verif. (secs)</th>
<th>Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example 1</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>0.04</td>
<td>0.05</td>
<td>20%</td>
</tr>
<tr>
<td>B</td>
<td>0.11</td>
<td>0.12</td>
<td>8.3%</td>
</tr>
<tr>
<td><strong>Example 2</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>0.08</td>
<td>0.12</td>
<td>33.3%</td>
</tr>
<tr>
<td>B</td>
<td>0.02</td>
<td>0.02</td>
<td>0%</td>
</tr>
<tr>
<td><strong>The Cnt Example</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cnt</td>
<td>0.05</td>
<td>0.09</td>
<td>44.4%</td>
</tr>
<tr>
<td>FastCnt</td>
<td>0.05</td>
<td>0.08</td>
<td>37.5%</td>
</tr>
<tr>
<td>PosCnt</td>
<td>0.11</td>
<td>0.13</td>
<td>15.3%</td>
</tr>
<tr>
<td>TwoCnt</td>
<td>0.09</td>
<td>0.14</td>
<td>35.7%</td>
</tr>
</tbody>
</table>
Conclusion

- Verification system based on separation logic
- User-definable shape predicates with size and bag properties
- Static and dynamic specs for OO verification
Thank you!

Questions?