Permissions to Specify the Composite Design Pattern

Kevin Bierhoff & Jonathan Aldrich
Carnegie Mellon University
SAVCBS 9 November 2008
Composites: Whole-part relationships as object trees

Invariant: even iff subtree has even number of nodes

Two conflicting goals:
Nodes depending on subnodes and adding children to any node
We will use permissions to force parent updates

- Two conflicting goals
  - Nodes depend on their subnodes in invariants
  - Allow adding new children to any node

- Solution: Do not add children without parents’ knowledge
  - Update parent after adding new child

- Permissions to enforce parent update
  - Distribute permission to update node between node and its immediate parent
Sample Composite tracks even vs. odd number of nodes

```java
public final class Composite {
    boolean odd;
    Composite parent, left, right;

    public Composite() { odd = true; parent = null; left = null; right = null; }

    public void setLeft(Composite c) {
        c.parent = this; left = c;
        if(c.odd) {
            odd = ! odd;
            Composite p = parent;
            while(p != null) {
                p.odd = ! p.odd;
                p = p.parent;
            }
        }
    }

    public boolean odd() {
        return odd;
    }
}
```
Access Permissions: typestate tracking under aliasing [OOPSLA ’07]

<table>
<thead>
<tr>
<th>Other references</th>
<th>Current reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/write</td>
<td>Read-only</td>
</tr>
<tr>
<td>Read/write</td>
<td>share</td>
</tr>
<tr>
<td>Read-only</td>
<td>full</td>
</tr>
<tr>
<td></td>
<td>immutable</td>
</tr>
</tbody>
</table>

- Consistency
  - 1 full and many pure
  - Many immutables and many pures

- Example: **full**(this, PARENT) in orphan
Fractions enable permission splitting and merging

- Split permissions to introduce aliases
  - $\text{full}(x, P) \Rightarrow \text{full}(x, P) \otimes \text{pure}(x, P)$

- Fractions allow permission merging [Boyland ’03]
  - $\text{full}(x, P) \Leftrightarrow \text{immutable}(x, P, \frac{1}{2}) \otimes \text{immutable}(x, P, \frac{1}{2})$
  - Can recover write permission from read-only permissions!

<table>
<thead>
<tr>
<th>Other references</th>
<th>Current reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/write</td>
<td>Read-only</td>
</tr>
<tr>
<td>Read/write</td>
<td>share</td>
</tr>
<tr>
<td>Read-only</td>
<td>pure</td>
</tr>
<tr>
<td>Read-only</td>
<td>full</td>
</tr>
<tr>
<td>Read-only</td>
<td>immutable</td>
</tr>
</tbody>
</table>
Dimensions, states, and invariants

- Permissions for each dimension
- Invariants separate for each dimension
- State invariants only hold when in state

states WEIGHT = { even, odd }
states PARENT = { orphan, hasParent }
states LEFT = { lefteven, leftodd }
states RIGHT = { righteven, rightodd }

boolean odd; in WEIGHT;
Composite parent; in PARENT;
Composite left; in LEFT;
Composite right; in RIGHT;
Linear logic for composing atomic predicates [Girard ’87]

- Permissions are linear resources
- Multiplicative conjunction
  - $A \otimes B$ (similar to separation logic’s $A \ast B$)
  - $A$ and $B$ at the same time
- Additive conjunction (internal choice)
  - $A \& B$ (similar to separation logic’s $A \land B$)
  - $A$ and $B$ available, but must choose one or the other
- Disjunction (external choice)
  - $A \oplus B$ (similar to separation logic’s $A \lor B$)
  - Either $A$ or $B$ non-deterministically—be ready for either
- Linear implication
  - $A \multimap B$ (similar to separation logic’s $A \rightarrow B$)
  - $B$ holds if $A$ can be consumed
Invariants we care about

\[
\begin{align*}
\text{invariant lefeven} & : \text{left} = \text{null} \oplus (\text{left} \neq \text{null} \otimes \text{left in even}); \\
\text{invariant leftodd} & : \text{left} \neq \text{null} \otimes \text{left in odd}; \\
\text{invariant righteven} & : \text{right} = \text{null} \oplus (\text{right} \neq \text{null} \otimes \text{right in even}); \\
\text{invariant rightodd} & : \text{right} \neq \text{null} \otimes \text{right in odd}; \\
\text{invariant WEIGHT:} & \\
& \quad (\text{odd} = \text{false} \rightarrow ((\text{this in leftodd} \otimes \text{this in righteven}) \oplus \\
& \quad \quad (\text{this in lefteven} \otimes \text{this in rightodd}))) \land \\
& \quad (\text{odd} = \text{true} \rightarrow ((\text{this in lefteven} \otimes \text{this in righteven}) \oplus \\
& \quad \quad (\text{this in leftodd} \otimes \text{this in rightodd}))); \\
\text{invariant even} & : \text{odd} = \text{false}; \\
\text{invariant odd} & : \text{odd} = \text{true};
\end{align*}
\]
Insight: Use immutable permissions for dependencies

- Invariants indicate dependencies
  - on other dimensions of the same object
  - on dimensions in other objects
- Need immutable permissions in those invariants
  - Guarantees that values do not change without knowledge of dependent object
  - Full permission would work, too
Immutable permissions to guarantee dependencies

\[\text{invariant WEIGHT} : \text{immutable}(\text{this, LEFT}) \otimes \text{immutable}(\text{this, RIGHT});\]
\[\text{invariant LEFT} : \text{left} \neq \text{null} \rightarrow \text{immutable(left, WEIGHT)};\]
\[\text{invariant RIGHT} : \text{right} \neq \text{null} \rightarrow \text{immutable(right, WEIGHT)};\]

\[\text{invariant PARENT} : \text{parent} \neq \text{this};\]
\[\text{invariant PARENT :}
\quad \text{parent} \neq \text{null} \rightarrow ((\text{immutable(parent, LEFT}) \otimes \text{parent.left} = \text{this}) \oplus
\quad (\text{immutable(parent, RIGHT}) \otimes \text{parent.right} = \text{this}));\]

\[\text{invariant lefteven} : \text{left} = \text{null} \oplus (\text{left} \neq \text{null} \otimes \text{left in even});\]
\[\text{invariant leftodd} : \text{left} \neq \text{null} \otimes \text{left in odd};\]
\[\text{invariant righteven} : \text{right} = \text{null} \oplus (\text{right} \neq \text{null} \otimes \text{right in even});\]
\[\text{invariant rightodd} : \text{right} \neq \text{null} \otimes \text{right in odd};\]

\[\text{invariant WEIGHT :}
\quad (\text{odd} = \text{false} \rightarrow ((\text{this in leftodd} \otimes \text{this in righteven}) \oplus
\quad (\text{this in lefteven} \otimes \text{this in rightodd}))) \&
\quad \ldots\]
Trick: Use half fractions to allow modification

- We can only modify a field if we have a full permission for its dimension
- Solution: Use ½ fractions
  - \( \text{full}(x, P) \Leftrightarrow \text{immutable}(x, P, ½) \otimes \text{immutable}(x, P, ½) \)

\begin{verbatim}
\text{invariant \textsc{weight} : \text{immutable}(this, \text{left}, ½) \otimes \text{immutable}(this, \text{right}, ½);} \\
\text{invariant \textsc{left} : \text{left} \neq \text{null} \rightarrow \text{immutable}(\text{left}, \text{weight}, ½);} \\
\text{invariant \textsc{right} : \text{right} \neq \text{null} \rightarrow \text{immutable}(\text{right}, \text{weight}, ½);} \\
\text{invariant \textsc{parent} : \text{immutable}(this, \text{weight}, ½) \otimes \text{parent} \neq this;} \\
\text{invariant \textsc{parent} :} \\
\text{(parent = null \rightarrow \text{immutable}(this, \text{weight}, ½)) \&} \\
\text{(parent \neq null \rightarrow (share(parent, \text{parent}) \otimes} \\
\text{((\text{immutable}(\text{parent, \text{left}, ½}) \otimes \text{parent.left} = this) \oplus} \\
\text{(\text{immutable}(\text{parent, \text{right}, ½}) \otimes \text{parent.right} = this)))};
\end{verbatim}
Permissions between a composite node and its neighbors

Legend

<table>
<thead>
<tr>
<th>Dimension</th>
<th>obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>permission</td>
<td>permission</td>
</tr>
</tbody>
</table>

PARENT

LEGAL

SHARE

½ imm

½ imm

parent

LEFT

RIGHT

Share

½ imm

½ imm

dimension

left
Specification for adding a child driven by invariants

```java
public void setLeft(Composite c)
    requires share(this, PARENT) \times immutable(this, LEFT, ½) \times
    share(c, PARENT) in orphan \times c \neq null \times c \neq this;
    ensures share(c, PARENT) in hasParent \times c.parent = this;
{ ... }
```

```java
invariant PARENT :
    parent \neq null \rightarrow (share(parent, PARENT) \times
    ((immutable(parent, LEFT, ½) \times parent.left = this) \oplus
    (immutable(parent, RIGHT, ½) \times parent.right = this)));

invariant orphan : parent = null;

invariant hasParent : parent \neq null;
```

"Consumed"
A taste of verifying implementation for new child

```java
public void setLeft(Composite c)
    requires share(this, PARENT) ⊗ immutable(this, LEFT, ½) ⊗
    share(c, PARENT) in orphan ⊗ c ≠ null ⊗ c ≠ this;
{
    unpack(c, PARENT);
    c.parent = this;
    unpack(this, PARENT);
    if(parent != null) {
        unpack(parent, PARENT); unpack(parent, WEIGHT);
        unpack(parent, LEFT); unpack(parent, RIGHT); }
    unpack(this, WEIGHT); unpack(this, LEFT);
    left = c;
    pack(this, LEFT); unpack(c, WEIGHT);
    if(c.odd) {
        pack(c, WEIGHT); pack(c, PARENT);
        odd = ! odd;
        pack(this, WEIGHT);
        if(parent != null) { pack(parent, LEFT); pack(parent, RIGHT); }
    }
    pack(this, PARENT); ...
}
```

immutable(this, WEIGHT, ½) ⊗
(parent ≠ null → (share(parent, PARENT)
⊗ immutable(parent, LEFT, ½)
⊗ parent.left = this))

immutable(parent.left, WEIGHT, ½)
merged: full(this, WEIGHT)

immutable(this, LEFT, ½)
merged: full(this, LEFT)
Future work:
Non-typestate invariants

- Permissions form frame of dependencies
- We think that any property could be tracked on top of them
  - invariant WEIGHT : weight = lw() + rw() + 1;
  - int lw() { return left == null ? 0 : left.weight; }
  - int rw() { return right == null ? 0 : right.weight; }
- May require theorem prover to reason about desired property
  - Or extend Boogie or ESC/Java to permissions
Permissions allow nodes to depend on their children and adding new nodes
  - Immutable permissions allow nodes to depend on other nodes
  - \( \frac{1}{2} \) fractions allow modifications when needed

Paper discusses how current limitations could be overcome
  - More complex invariants
  - Support for list of children
  - Specification overhead reduction