KreYol

Maximilian Dylla

Chalmers University of Technology

8th KeY Symposium 2009, Speyer
• executable OO modelling language
• verifyable by design
• developed at University of Oslo
1st Level of Parallelism: System

- distributed system of objects
- message passing
- communication via (co)interfaces
- asynchronous communication:
  
  ```
  label ! obj.meth(x, y);
  ...
  label ? (y)
  ```

- only assumption on network: Messages are delivered eventually
2nd Level of Parallelism: Inside an Object

- thread creation on method invocation
- at most one active thread at the time
- communication via shared variables
- cooperative scheduling
  ⇒ release points:

  ```
  release
  ```

  ```
  await exp
  ```

- no assumptions on scheduling strategy
Verifying Release Points

Class Invariant

- ensures properties of class attributes
- must hold on a thread switch
  \[ \Rightarrow \text{must hold at release points} \]

\[ \equiv \text{Inv}_{\text{class}} \quad \Rightarrow \quad U_A(v_{\text{attr}})(\text{Inv}_{\text{class}} \rightarrow \langle \omega \rangle \phi) \]

\[ \Rightarrow \langle \text{release}; \omega \rangle \phi \]

- no other threads considered
Verifying a Method

- interface contains contract
- class invariant must hold before and after

\[
\text{op meth}(\text{in } a: \text{Int}; \text{ out } b: \text{Bool}) \equiv \text{body}
\]

\[\Rightarrow \text{Pre}_{\text{meth}}(a) \land \text{Inv}_{\text{class}} \rightarrow \langle \text{body} \rangle \text{Post}_{\text{meth}}(b) \land \text{Inv}_{\text{class}}\]
Verifying Method Calls

History $\mathcal{H}$

- system wide communication log containing:
  - invocation messages
  - completion messages
  - new object messages
- ordered by sending time (the only known order)

Verifying a Class

- local history $\mathcal{H}/this$ as a ghost class attribute
  $\Rightarrow$ class invariant talks about local history
- ensure well formedness of history (similar to reachable state)

Verifying the System

- given $\mathcal{H}/o$ for all classes, show $\exists \mathcal{H}$
Local History

Problem: Arriving messages

- sending time unknown
- sending order unknown
- number unknown

Solution

- model the history with uncertainty
- assert existence of seen messages
Local History: Example

\[ h_0 \leq h_1 \leq h_2 \leq h_3 \]

\[ \neg \text{Invoc}(h_0, l) \]
\[ \neg \text{Comp}(h_0, l) \]
\[ \pi; \]

\[ \neg \text{Invoc}(h_1, l) \]
\[ \neg \text{Comp}(h_1, l) \]
\[ l!o.m(x); \]

\[ \neg \text{Invoc}(h_2, l) \]
\[ \text{Comp}(h_3, l) \]
\[ \neg \text{Comp}(h_2, l) \]

\[ \pi; \]

\[ \Rightarrow o \neq \text{null} \land Wf(h_{pre}) \]

\[ \Rightarrow \{ \mathcal{H} := \text{some } h. \ Wf(h) \land h_{pre} \leq h \land \neg \text{Invoc}(h, l) \land \neg \text{Comp}(h, l) \} \]

\[ \Rightarrow (\text{Prem}(x) \land \langle \omega \rangle \phi) \]

\[ \Rightarrow \langle l!o.m(x); \ \omega \rangle \phi \]

\[ \Rightarrow l \neq \text{null} \land Wf(h_{pre}) \land \text{Invoc}(h_{pre}, l) \]

\[ \Rightarrow \{ \mathcal{H} := \text{some } h. Wf(h) \land h_{pre} \leq h \land \text{Comp}(h, l) \} U_A(y)(\text{Post}(y) \rightarrow \langle \omega \rangle \phi) \]

\[ \Rightarrow \langle l?(y); \ \omega \rangle \phi \]
Hybrid History

Problem

- only assertions about existence of messages possible

Solution

- order of sending is known
  \[ \Rightarrow \text{divide into: } \mathcal{H}_{obj} \text{ and } \mathcal{H}_{send} := \mathcal{H}_{obj} / this \]
- keep \( \mathcal{H}_{send} \) as a list of messages (between release points)
- drawback: consistency checks
Hybrid History: Example

\[ \begin{array}{c}
\frac{\neg \text{Invoc}(h_0, l) \quad \neg \text{Comp}(h_0, l)}{hs} \quad \frac{\text{Invoc}(h_1, l) \quad \neg \text{Comp}(h_0, l)}{hs := hs \vdash [\text{this} \rightarrow o.m(x)]} \\
\frac{\text{Comp}(h_2, l)}{\pi} \quad \frac{l! o.m(x)}{l?(y)}
\end{array} \]
Case study

```plaintext
class Buffer
begin
  var cell : Any;
  with Any
    op put(in a : Any) == await cell = null;
       cell := a
    op get(out b : Any) == await cell != null;
       b := cell; cell := null
end
```

- \( Pre_{\text{put}}(a) := a \neq \text{null} \)
- \( Post_{\text{get}}(b) := b \neq \text{null} \)
- \( Inv_C := \left( \neg\text{cell} \neq \text{null} \leftrightarrow (\mathcal{H} \vdash [\text{caller} \leftarrow \text{this.put()}]) \right) \land \neg\text{cell} \neq \text{null} \leftrightarrow (\mathcal{H} \vdash [\text{caller} \leftarrow \text{this.get()}]) \right) \land \text{Prefix}(\mathcal{H}) \)
Prototype Version

- standard rules working
- rules involving histories need adaptations to specific example
- no support for program loading
Implementation

<table>
<thead>
<tr>
<th>package</th>
<th>lines of code (without strategy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>key.lang clang</td>
<td>25k</td>
</tr>
<tr>
<td>key.lang creol</td>
<td>3k</td>
</tr>
<tr>
<td>key.java</td>
<td>50k</td>
</tr>
</tbody>
</table>

- data structures created on startup
  ⇒ configurable, but slower
- one class for AST
  ⇒ e.g. ifThenElse.getCondition() impossible
- pushdown automaton for AST creation
Good luck with HATS!