Clock Analysis of X10 Programs

Nalini Vasudevan

Mentor: Olivier Tardieu
Manager: Vijay Saraswat
Background

- X10 programming language
  - Parallel and Distributed
- Tasks/activities created using `async`
  ```
  async {
    /* body of async */
  }
  ```
- Other constructs: `finish`, `atomic`
- **Focus**: Synchronization between activities through clocks
Clocks in X10

- **Barriers**
  - Declare clocks
  - Share clocks
  - **Synchronize**
    - *next()* function
      - All tasks clocked on c have to synchronize

```java
final clock c = clock.factory.clock();
async clocked (c) {
  ..
  c.next();
  ..
  c.next();
}
async clocked (c) {
  ..
  c.next();
  ..
  c.next();
}
```
More about Clocks

- `resume()`
  - Indicates the end of current phase

```java
final clock c = clock.factory.clock();
async clocked (c) {
    ..
    c.next();
    ..
    c.next();
}
async clocked (c) {
    ..
    c.resume()
    ..
    c.next();
    ..
    c.next();
}
```
More about Clocks

- **resume()**
  - Indicates the end of current phase

```java
final clock c = clock.factory.clock();
async clocked (c) {
  ..
  c.next();
  ..
  c.next();
}
async clocked (c) {
  ..
  c.resume()
  ..
  c.next();
  ..
  c.next();
}
```
More about Clocks

● **resume()**
  - Indicates the end of current phase

```java
final clock c = clock.factory.clock();
async clocked (c) {
    .. c.next();  .. c.next();
}
```
More about Clocks

- **resume()**
  - Indicates the end of current phase

```java
final clock c = clock.factory.clock();
async clocked (c) {
    ..
    c.next();
    ..
    c.next();
}
```

```java
async clocked (c) {
    ..
    c.resume();
    ..
    c.next();
}
```
More about Clocks

- **resume()**
  - Indicates the end of current phase

```java
final clock c = clock.factory.clock();
async clocked (c) {
    ..
    c.next();
    ..
    c.next();
}
async clocked (c) {
    ..
    c.resume()
    ..
    c.next();
    ..
    c.next();
}
```
More about Clocks

- *drop()*
  - Explicitly drop the clock.
  - Task does not have to synchronize anymore
The Protocol

clock.factory.clock() / async clocked (c)

Active

Inactive

Resumed

c.next() / c.drop() / c.resume() / c.drop() / c.resume() / async clocked (c)

X
Motivation

- Default implementation handles all cases
  - Handles protocol violation dynamically by throwing exceptions
    - example: call next() after drop()

- We can generate more efficient code if we know that
  - Activity does not violate the protocol?
  - Activity never calls resume() on clock c?

- We can also provide feedback to the user
  - Activity may violate the protocol…
Implementation Overview

- **Static Analysis**
  - Use *wala* for intermediate representation and pointer analysis
  - Extract model from IR
    - One automaton per clock
  - Use NuSMV for model checking

- **Code Specialization**
  - X10 compiler plugin to choose clock implementation based on analysis result
c = clock.factory.clock()
c.next()
if (n > 1)
    c.resume()
else
    c.next();
c.next();
c.drop();
Building the Automaton

1. `c = clock.factory.clock()`
2. `c.next()`
   - if (n > 1)
     - `c.resume()`
   else
3. `c.next();`
4. `c.next();`
5. `c.drop();`
Building the Automaton

init (clock) = register;
next(clock) :=
case
  (clock = register) : next_2;
  (clock = next_2) : {resume_3, next_4};
  (clock = resume_3) : next_5;
  (clock = next_4) : next_5;
  (clock = next_5) : drop_6;
  1: clock;
esac;

DEFINE clock_next = clock in {next_2, next_4, next_5}
DEFINE clock_drop = clock in {drop_6}
init (status) = inactive;
next(status) :=
   case
      (status = inactive) & (clock_register) : active;
      (status = active) & (clock_drop) : inactive;
      (status = active) & (clock_resume): resumed;
      (status = resumed) & (clock_next): active;
   ..
   -- Exception cases
      (status = resumed) & (clock_resume): resume_exception;
      (status = inactive) & (clock_next) : drop_exception;
   ..
Checking for properties

- Check if a clock is protocol violation free?

  \[
  \text{DEFINE status\_exception} = \text{status in \{drop\_exception, async\_exception, ...\}}
  \]

  \[
  \text{SPEC AG(\neg status\_exception)}
  \]

- Check if a clock never calls \textit{resume()}?

  \[
  \text{SPEC AG(\neg clock\_resume)}
  \]
Dealing with *asyncs*

- Viewing *async* as a separate path

```javascript
c = clock.factory.clock() c.next();
async clocked (c) {
    c.resume();
}
c.next(); c.drop();
```
Combining with Aliasing Analysis

clock c = clock.factory.clock ();
clock d = clock.factory.clock ();
clock x = (n>1)? c: d;
  x.next();
  x.resume();
Combining with Aliasing Analysis

clock c = clock.factory.clock ();
clock d = clock.factory.clock ();
clock x = (n>1)? c: d;
x.next();
x.resume();

clock c = clock.factory.clock ();
clock d = clock.factory.clock ();
if(*) {
    c.next();
    c.resume();
}
else {
    d.next();
    d.resume();
}
Combining with Aliasing Analysis

clock c = clock.factory.clock ();
clock d = clock.factory.clock ();
clock x = (n>1)? c: d;
x.next();
x.resume();

clock c = clock.factory.clock ();
clock d = clock.factory.clock ();
if(*)
  c.next();
else
  d.next();
if(*)
  c.resume();
else
  d.resume();
## Results

<table>
<thead>
<tr>
<th>Example</th>
<th>Lines of Code</th>
<th>No. of Clocks</th>
<th>Analysis Time (sec)</th>
<th>Result</th>
<th>Speed Up</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>IR and Automata construction</td>
<td>NuSMV</td>
<td>Total</td>
</tr>
<tr>
<td>Kernel Algorithm</td>
<td>55</td>
<td>1</td>
<td>5.3</td>
<td>0.1</td>
<td>5.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(38.4/40.4) -&gt; 5.0%</td>
</tr>
<tr>
<td>All Reduction Barrier</td>
<td>65</td>
<td>1</td>
<td>21.7</td>
<td>0.1</td>
<td>21.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(13.1/13.3) -&gt; 1.5%</td>
</tr>
<tr>
<td>Sieve (Stream of Eratosthenes)</td>
<td>95</td>
<td>1</td>
<td>25.9</td>
<td>0.1</td>
<td>26.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(231.3/233.8) -&gt; 1.1%</td>
</tr>
<tr>
<td>N-Queens</td>
<td>155</td>
<td>1</td>
<td>20.0</td>
<td>0.3</td>
<td>20.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(58.7/58.8) -&gt; 0.2%</td>
</tr>
<tr>
<td>Sequence Alignment (Edmiston)</td>
<td>205</td>
<td>2</td>
<td>18.4</td>
<td>0.2</td>
<td>18.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(19.4/20.0) -&gt; 3%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Clock 2: NR</td>
</tr>
<tr>
<td>LU Decomposition</td>
<td>215</td>
<td>1</td>
<td>8.6</td>
<td>0.2</td>
<td>8.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(25.4/26) -&gt; 2.3%</td>
</tr>
<tr>
<td>Java Grande Forum Benchmark</td>
<td>930</td>
<td>1</td>
<td>24.7</td>
<td>0.3</td>
<td>25.0</td>
</tr>
<tr>
<td>Suite</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(30.4/30.8) -&gt; 1.3%</td>
</tr>
</tbody>
</table>

**EF**: Exception Free, **NR**: No Resume, **ON**: Only Clock Creator calls Next
Conclusion and Future Work

- Working tool!
- Sequential analysis for concurrency optimization
- Future Work
  - Inter-activity analysis
    - Deadlocks
  - Using clock information for refining aliasing analysis
Questions Guaranteed
Answers are not!