Metropolis Verification
Backends

Xi Chen, Guang Yang, Harry Hsieh,
Felice Balarin and Yoshi Watanabe

Outline

• Promela Backend in Metropolis
  • Formal Verification Methodology
  • Model Checker SPIN
  • Implementation
  • Case Study
• Constraint Checking for Simulation in Metropolis
  • Constraint elaboration
  • Annotation trace generation
  • LOC trace checker generation
Formal Verification Methodology

Model Checker SPIN and Promela

- Promela language elements
  - Concurrent process
  - Communication channels
  - Atomic statement
  - Flow control statements
  - Case selection statements

Promela is a C-style procedural format with simple constructs suitable for protocol verification
Translation from MMM to Promela

- Processes/media
  \(\rightarrow\) Proctypes with functional inlining
- Dynamic objects, e.g. dynamic arrays
  \(\rightarrow\) Restricted and static objects, e.g. static arrays
- Function-architecture mapping
  \(\rightarrow\) Using rendezvous channels to synchronize functional processes and mapping processes

Translation of "await"

```plaintext
await {  
  (guard_1; testlist_1 setlists_1) {stmts_1}  
  (guard_k; testlist_k setlists_k) {stmts_k}  
}
```

"await" statement
- Synchronize concurrent processes
- Multiple critical sections
- Guard and semaphores
- Non-deterministic selection of critical sections
Case Study – Producer/Consumer

**Property**: “Whenever the producer starts to write an item into the medium, there must be some space in the medium”

\[ \text{LTL: } \Box ((P_1\text{-write } \lor \ldots \lor P_m\text{-write}) \rightarrow M_1\text{-not-full}) \]

*When m = 2, n = 1 and M has single space, source code: 120 lines, Promela code: 650 lines. Verification is passed within 1s CPU time and 13MB memory.*

Case Study – YAPI and TTL

To check if there is a deadlock situation within the YAPI and TTL channel, use:

“once the writer starts writing data into the channel, it will finish it eventually”.

- \((\text{datagen\_start } \rightarrow (\leftrightarrow \text{ datagen\_finish}))\)
  - This property is verified on the YAPI level with exhaustive verification within 9 hours.
  - A deadlock situation is found in the TTL channel.
  - After the bug in TTL channel fixed, the property is passed within 4 hours.
Future Work

- Safe abstraction

- Automatic Abstraction propagation in Metropolis

Outline

- Promela Backend in Metropolis
  - Formal Verification Methodology
  - Model Checker SPIN
  - Implementation
  - Case Study

- Constraint Checking for Simulation in Metropolis
  - Constraint elaboration
  - Annotation trace generation
  - LOC trace checker generation
LOC Constraints in MMM

- LOC is a transaction-level quantitative constraint language
- Directly supported by MMM syntax
- Using MMM keywords constraint and loc
- For example (a latency constraint):

```java
constraint {
    event P0_start = beg(p0, p0.start);
    event P0_finish = beg(p0, p0.finish);
    loc(forall (int i) t@(P0_finish, i) - t@(P0_start, i) <= 20);
}
```

- Constraints can be extracted and instantiated during network elaboration

Constraint Elaboration - An Example

- If m = 2, there are actually 2 different constraint instances
Constraint Elaboration – An Example (cont’d)

```
netlist test.sumnet {
  o Instance name: top_level_netlist
  o component name: null
  o Components:
  ...
  o Not refined by a netlist
  o Does not refine any node
  o Constraints:
    - LOC Constraint (# 0)
      o Container: top_level_netlist
      o Event references:
        - beg(datagen1, y2bf1.tokenLabel)
        - beg(sum1, bf2y1.tokenLabel)
    - LOC Constraint (# 1)
      o Container: top_level_netlist
      o Event references:
        - beg(datagen1, y2bf1.tokenLabel)
        - beg(sum1, bf2y1.tokenLabel)

}
```

The print-out of the elaborated constraints

- Constraints are indexed in an node
- Event references are saved
- A list of annotations are saved in the network

Annotation Trace Generation

- An application of elaborated constraints
- Utilize elaborated constraints and annotations
- Trace generation – insert “print” statements into SystemC code
An Example of Annotation Trace Generation

```plaintext
constraint {
    event P1_start = beg(p1, p1.start);
    event C_start = beg(c, c.start);
    loc(forall (int i) w@(P1_start, i) == w@(C_start, i));
}
```

A Constraint in MMM

Elaborated annotations

```
<table>
<thead>
<tr>
<th>Event</th>
<th>Annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>beg(p1, p1.start)</td>
<td>w</td>
</tr>
<tr>
<td>beg(c, c.start)</td>
<td>w</td>
</tr>
</tbody>
</table>
```

SystemC Backend

SystemC Simulation w/ trace generation

Elaboration

• Another utilization of elaborated constraints
• Generate checkers from elaborated constraints and annotations

LOC Backend

Executable LOC Checkers
A Complete Example of LOC Checking

Elaborated Network

SystemC Backend

BEG_Consumer_Consumer_start_0
BEG_Producer1_Producer1_start_0
BEG_Producer1_Producer1_start_1
BEG_Producer1_Producer1_start_2
BEG_Consumer_Consumer_start_1
BEG_Producer1_Producer1_start_3
BEG_Consumer_Consumer_start_2
BEG_Producer1_Producer1_start_4
BEG_Consumer_Consumer_start_3

Trace from SystemC Simulation

LOC Backend

Executable Checker

Trace Checker

Error Report
Future Work

- Integrate LOC monitors into SystemC simulation
- Check or monitor LTL constraints

Thank you!