Automatic Trace Analysis for Logic of Constraints

Xi Chen, Harry Hsieh
University of California, Riverside
Felice Balarin, Yosinori Watanabe
Cadence Berkeley Laboratories

Outline

Introduction
- System-level design
- Logic of Constraints

Trace analysis methodology
- Methodology and algorithms
- Case studies

Proving LOC formulas

Summary
**System-Level Design Methodology**

- RTL level design is no longer efficient for systems containing millions of gates.
- System-level design becomes necessary
  - Reuse of design components
  - Reduce overall complexity
  - Ease debugging
- Verification methods must accompany every step in the design flow
- Constraints need to be specified at the highest level and verified ASAP

**Logic of Constraints (LOC)**

- A transaction-level quantitative constraint language
- Works on a sequence of events from a particular execution trace
- The basic components of an LOC formula:
  - Boolean operators: ¬ (not), ∨ (or), ∧ (and) and → (imply)
  - Event names, e.g. “in”, “out”, “Stimuli” or “Display”
  - Instances of events, e.g. “Stimuli[0]”, “Display[10]”
  - Annotations, e.g. “t(Display[5])”
  - Index variable i, the only variable in a formula, e.g. “Display[i-5]” and “Stimuli[i]”
**LOC Constraints**

Throughput: "at least 3 Display events will be produced in any period of 30 time units".
\[ t(\text{Display}[i+3]) - t(\text{Display}[i]) \leq 30 \]

Other LOC constraints
- Performance: rate, latency, jitter, burstiness
- Functional: data consistency

**Assertion Languages**

- IBM’s Sugar and Synopsis' OpenVera
- Good for both formal verification and simulation verification
- Implemented as libraries to support different HDLs
- Assertions are expressed with
  - Boolean expressions, e.g. \[ a[0:3] \& b[0:3] = "0000" \]
  - Temporal logics, e.g. always !(a \& b)
  - HDL code blocks, e.g. handshake protocol
- Mainly based on Linear Temporal Logic
**Characteristics of LOC Formulism**

- Constraints can be automatically synthesized into static checkers, runtime monitors and formal verification models.
- Performance constraints in addition to functional constraints
- A different domain of expressiveness than LTL.

\[ t(\text{Display}[i]) - t(\text{Stimuli}[i]) \leq 25 \]

\[ [\neg \langle Display_occurrence \rightarrow Data > 10 \rangle] \]

LOC: \text{data(\text{Display}[i])} > 10  
LTL: \text{[]}(\text{Display occurrence} \rightarrow \text{Data} > 10)  

**Outline**

- **Introduction**
- **Trace analysis methodology**
  - Methodology and algorithms
  - Case studies
- **Proving LOC formulas**
- **Summary**
Trace Analysis Methodology

- An efficient checking algorithm
- An automatic LOC checker generator
- Extended to runtime constraint monitoring

**Algorithm of the LOC Checker**

**Throughput**: “at least 3 Display events will be produced in any period of 30 time units”

\[ t(\text{Display}[i+3]) - t(\text{Display}[i]) \leq 30 \]

FIR Trace

- Display[0]
- i = 0
- Display[3]

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>t(Display[i])</td>
<td>13</td>
<td>23</td>
<td>33</td>
<td>43</td>
<td>53</td>
<td>63</td>
</tr>
</tbody>
</table>

Queue data structure
Input and Output

Input of the checker generator – formula and trace format:

- **LOC: rate**
  - formula: \( t(\text{Display}[i + 1]) - t(\text{Display}[i]) = 10 \)
  - annotation: event value \( t \)
  - trace: "%s : %d at time %d"

- **LOC: latency**
  - formula: \( t(\text{Display}[i]) - t(\text{Stimul}[i]) = 25 \)
  - annotation: event value \( t \)
  - trace: "%s : %d at time %d"

Output of the trace checking – error report:

```bash
username@chimera $ checker latency.trace
Reading from trace file "latency.trace"
Formula \( t(\text{Display}[i]) - t(\text{Stimul}[i]) = 25 \) is violated at trace line# 278: 
  Display : -6 at time 87

where i = 23
  t (Display)[i] = 87
  t (Stimul)[i] = 60

```

Dealing with Memory Limitation

- scan trace and store the annotations only once.
- If the memory limit has been reached,
  - stop storing more annotations
  - search the rest of trace for current \( i \)
  - resume storing annotations after freeing memory
Static Trace Checking v.s. Runtime Monitoring

- Runtime constraint monitoring:
  - Integrate the trace checker into a compiled-code simulator, e.g. SystemC modules
  - At runtime, the events and annotations are passed to the monitor module directly
  - Static trace checking v.s. runtime monitoring

<table>
<thead>
<tr>
<th></th>
<th>Static Trace Checking</th>
<th>Runtime Monitoring</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steps</td>
<td>Simulation, then checking</td>
<td>Simulation</td>
</tr>
<tr>
<td>Time</td>
<td>More</td>
<td>Less</td>
</tr>
<tr>
<td>Space</td>
<td>More</td>
<td>Less</td>
</tr>
<tr>
<td>Debug</td>
<td>Easy</td>
<td>Hard</td>
</tr>
<tr>
<td>Simulator</td>
<td>Independent</td>
<td>Dependent</td>
</tr>
</tbody>
</table>

Case Study – FIR Filter

Stimuli : 0 at time 9
Display : 0 at time 13
Stimuli : 1 at time 19
Display : -8 at time 23
Stimuli : 2 at time 29
Display : -16 at time 33
Stimuli : 3 at time 39
Display : -13 at time 43
Stimuli : 4 at time 49
Display : 6 at time 53

Rate: \( t(\text{Display}[i+1]) - t(\text{Display}[i]) = 10 \)

Latency: \( t(\text{Display}[i]) - t(\text{Stimuli}[i]) \leq 25 \)

Jitter: \( | t(\text{Display}[i]) - i(1) \times 10 | \leq 4 \)

Throughput: \( t (\text{Display}[i+1000]) - t(\text{Display}[i]) \leq 1001 \)

Burstiness: \( t (\text{Display}[i+1000]) - t(\text{Display}[i]) > 9999 \)
Trace Checking Results (FIR)

<table>
<thead>
<tr>
<th>Lines of Trace</th>
<th>$10^5$</th>
<th>$10^6$</th>
<th>$10^7$</th>
<th>$10^8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rate</td>
<td>Time(s)</td>
<td>1</td>
<td>8</td>
<td>89</td>
</tr>
<tr>
<td></td>
<td>Memory</td>
<td>28B</td>
<td>28B</td>
<td>28B</td>
</tr>
<tr>
<td>Latency</td>
<td>Time(s)</td>
<td>1</td>
<td>12</td>
<td>120</td>
</tr>
<tr>
<td></td>
<td>Memory</td>
<td>28B</td>
<td>28B</td>
<td>28B</td>
</tr>
<tr>
<td>Jitter</td>
<td>Time(s)</td>
<td>1</td>
<td>7</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>Memory</td>
<td>28B</td>
<td>28B</td>
<td>28B</td>
</tr>
<tr>
<td>Throughput</td>
<td>Time(s)</td>
<td>1</td>
<td>7</td>
<td>77</td>
</tr>
<tr>
<td></td>
<td>Memory</td>
<td>0.4KB</td>
<td>0.4KB</td>
<td>0.4KB</td>
</tr>
<tr>
<td>Burstiness</td>
<td>Time(s)</td>
<td>1</td>
<td>7</td>
<td>79</td>
</tr>
<tr>
<td></td>
<td>Memory</td>
<td>4KB</td>
<td>4KB</td>
<td>4KB</td>
</tr>
</tbody>
</table>

Resource Usage for Checking Constraints (1) – (5)

Runtime Monitoring (FIR)

- The checker implemented as a SystemC module and applied on the latency constraint, i.e.
  \[ t(\text{Display}[i]) - t(\text{Stimuli}[i]) \leq 25 \]  
  (C2)
- Simulation trace is no longer written to a file but passed to the monitoring module directly.

<table>
<thead>
<tr>
<th>Lines of Trace</th>
<th>$10^5$</th>
<th>$10^6$</th>
<th>$10^7$</th>
<th>$10^8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation (s)</td>
<td>1</td>
<td>14</td>
<td>148</td>
<td>1404</td>
</tr>
<tr>
<td>Static trace checking (s)</td>
<td>1</td>
<td>12</td>
<td>120</td>
<td>1229</td>
</tr>
<tr>
<td>Total: simulation+checking (s)</td>
<td>2</td>
<td>26</td>
<td>268</td>
<td>2633</td>
</tr>
<tr>
<td>Simulation w/ monitoring (s)</td>
<td>2</td>
<td>14</td>
<td>145</td>
<td>1420</td>
</tr>
</tbody>
</table>

Results of Runtime Monitoring on FIR for the Latency Constraint (2)

DAC June 2003
Case Study – Picture In Picture

**Picture-In-Picture (PIP)**
- a system level design for set-top video processing
- 19,000 lines of source code
- 120,000 lines of simulation output (control trace)

PIPs

PIPs

1. Data consistency: “The numbers of the fields read in and produced should be equal.”

\[ \text{field\_count}(\text{in}[i]) = \text{field\_count}(\text{out}[i]) \]

- **field\_count** is an annotation: number of fields processed
- **in, out** are events: reading a field and producing a field
2. “The field sizes of paired even and odd fields should be the same.”

\[
\text{size}(\text{field}_\text{start}[2i+2]) - \text{size}(\text{field}_\text{start}[2i]) = \\
\text{size}(\text{field}_\text{start}[2i+1]) - \text{size}(\text{field}_\text{start}[2i])
\]

3. “Latency between user control and actual size change <= 5.”

\[
\text{field}_{\text{count}}(\text{change}_\text{size}[i]) - \text{field}_{\text{count}}(\text{read}_\text{size}[j]) \leq 5
\]

**Trace Checking Results:** With the trace of about 120,000 lines, all these three constraints are checked within 1 minute.

---

**Outline**

- Introduction
- Trace analysis methodology
- Proving LOC formulas
- Summary
Formal Verification Tools and Methods

- Model checkers, e.g. SPIN, SMV
- Check if a finite state system (model) satisfy some property
- Properties are expressed with temporal logics, e.g. LTL
- Limitation
  - state explosion
  - finite state

Formal Verification for LOC

- We define a subset of LOC that has finite-state equivalents
  - represent the LOC formula with LTL
  - use LTL model checking directly
  - Example:

\[
t(\text{Display}[i+1]) - t(\text{Display}[i]) = 10
\]

\[
\text{Display_occur} \rightarrow \text{Display}_t - \text{Display}_t\_last = 10
\]
Formal Verification for LOC (cont’d)

- Other LOC formulas are beyond the finite-state domain, e.g. the latency constraint
  - make assumption to limit the system to finite-state domain
  - verify assumption and assumption → constraint separately

  **Verification Outcomes**

  - Assumption satisfied? → YES
  - Assumption satisfied? → Unknown
  - Assumption satisfied? → NO

Case Study – A FIFO Channel

- Data consistency constraint on the channel:
  \[ data(DataGen_write[i]) = data(Sum_read[i]) \]
- Assumption – “Sum_read always follows DataGen_write, between a write and its corresponding read, only 30 more writes can be produced”
Case Study – A FIFO Channel (cont’d)

Using the model checker SPIN, the assumption is verified in 1.5 hours and assumption → constraint is verified in 3 hours.

The FIFO channel is a library module:
- Repeated use
- Small 600 lines of source code v.s. PIP (19,000 lines)

Summary

- LOC is useful and is different from LTL
- Automatic trace analysis
- Case studies with large designs and traces
- Formal verification approach

DAC June 2003