Runtime Verification Based on Executable Models: On-the-Fly Matching of Timed Traces

Mikhail Chupilko,
Alexander Kamkin
Outline

• Hardware models
• Runtime verification
• Elements of formalization
• Conformance relation
• Conclusion
Hardware models

• They are developed in Hardware Description Languages, like Verilog or VHDL
• The result of development is the program being executed in HDL simulator
• The common approach for verification of hardware models is testing of HDL programs
• To automatize testing is possible by means of executable models (e.g. in C++)
HDL programs

```c
input S;
output R1, R2;
void design() {
    while (true) {
        wait(S);
        delay(6);
        R1 = 1;
        delay(1);
        R1 = 0;
        R2 = 1;
        delay(1);
        R2 = 0;
    }
}
```
Hardware model behavior
Reference model-based test oracle

- HDL
- Test oracle
- Reference model
- Reaction arbiters
- Reaction comparators
- Input interface adapters
- Stimuli
- Output interface adapters
- HDL-model reactions
Behavior correctness checking

Functional properties

• Set of reactions is correct
• Each reaction is correct

Time restrictions

• Reaction order is correct
• Delays between reactions are correct
Cycle-accurate checking

Reference model reactions

\texttt{send}(R1)

\texttt{delay}(3)

\texttt{send}(R2)

Reactions of HDL-model

Comparison

- \checkmark
- \times
- \checkmark
- \times
Ambiguity in reaction order

Execution of reference model

```c
recv(in_iface, S);
...
send(out_iface, R1);
...
send(out_iface, R2);
```

Execution of HDL-model

Error: \( R_2 \neq R_1 \)

Allowed: \( R_2 \in \text{Order} \)

Reverse order
Arbitration of reactions

• Reaction arbiter finds a reaction corresponding to the reference model one
• Behavior checking depends on both reference model and on arbitration
• Reaction arbiters encapsulate parts of test oracle functionality aimed at reaction order checking
Types of reaction arbiters

• Deterministic model-based arbiter
  \[arbiter: 2^{\text{Reaction}} \rightarrow \text{Reaction} \cup \{\text{fail}\}\]

• Adaptive arbiter
  \[arbiter: 2^{\text{Reaction}} \times \text{Reaction} \rightarrow \text{Reaction} \cup \{\text{fail}\}\]

• Two-level arbiter
  \[arbiter(\text{reactions}) \equiv arbiter_2(arbiter_1(\text{reactions}), \text{reaction})\]
  
  – Non-deterministic arbiter
  
  – Adaptive arbiter
Deterministic arbiter

Reference model reactions

\texttt{send(R1);}

\ldots

\texttt{send(R2);}

HDL-model reactions

Known order

Reaction arbiter

FIFO

Comparison
Adaptive arbiter

Reference model reactions

send(R1);
...
send(R2);

HDL-model reactions

Unknown order

Get(R1)

Reaction arbiter

R1
R2

Hint

Comparison

nx

nx

nx
Two-level arbiter

Reference model reactions

\[ \text{send}(R_1); \]
\[ \ldots \]
\[ \text{send}(R_2); \]

HDL-model reactions

Comparison

Hint

Candidates

Arbiter #1

Arbiter #2

Partially known order

Get(R1)
Timed word (Alur & Dill, 1994)

\[ \Sigma - \text{alphabet of events} \]
\[ T - \text{time domain (} \mathbb{R}^{\geq 0} \text{ or } \mathbb{N} \text{)} \]

\[ w = (a_0, t_0)(a_1, t_1), \ldots \in (\Sigma \times T)^\omega(*) \]

- \( \forall i \cdot t_i < t_{i+1} \) \( (t_i \leq t_{i+1}) \) – monotonicity
- \( \forall T \exists i \cdot t_i > T \) – progress (if \(|w| = \infty\))
Mazurkiewicz trace (1977)

\(\Sigma\) – alphabet of events
\(I \subset \Sigma \times \Sigma\) – relation of independence

*Equivalent*: \(u \equiv v \iff u\) is derived from \(v\) by means of reordering of closest independence events

*Trace* is a class of equivalence of event chains in respect to equivalent relation \(\equiv\)
Mazurkiewicz trace (1977) - Example

\[ \Sigma = \{ a, b, c, d \} \]
\[ I = \{ (a, b), (c, d) \text{ + symmetry} \} \]

\[ [ab]_{\equiv} = \{ ab, ba \} \]
\[ [bc]_{\equiv} = \{ bc \} \]
\[ [abcd]_{\equiv} = \{ abcd, bacd, abdc, badc \} \]
Partially ordered set – Pratt (1982)

Σ – alphabet of events
Pomset is tuple $\langle V, \leq, \lambda \rangle$

- $V$ – set of vertexes
- $\leq \subset V \times V$ – partial set
- $\lambda: V \to \Sigma$ – labeling function
Partially ordered set – Pratt (1982)
Examples
Timed trace – Chieu & Hung (2012)

\( \Sigma \) – alphabet of events, \( T \) – time domain

Timed trace – \( \langle V, \leq, \lambda, \theta [, \delta] \rangle \)

- \( V \) – set of vertexes
- \( \leq \subset V \times V \) – partial order
- \( \lambda: V \rightarrow \Sigma \) – labeling function
- \( \theta: V \rightarrow T \) – time of event
- \( \delta: V \rightarrow \Delta T \) – allowed interval
Timed trace – Chieu & Hung (2012)

Examples

• \{ abcd, bacd, abdc, badc \}
• \{ abcd, bacd \} – time restrictions
Behavior of specification and implementation

Implementation behavior
\[ \langle V_I, \emptyset, \lambda_I, \theta_I \rangle \]

Specification behavior
\[ \langle V_S, \leq, \lambda_S, \theta_S, \delta_S \rangle \]

Allowed time interval
\[ \delta_S(x) = [\theta_S(x) - \Delta t(x), \theta_S(x) + \Delta t(x)] \]

Correspondence of events
\[ \text{match}(x, y) = \left( \lambda_I(y) = \lambda_S(x) \right) \land \left( \theta_I(y) \in \delta_S(x) \right) \]
Conformance relation

\[ I \sim S \iff \forall t \in T . \]

\[ \exists M \subseteq \{ (x, y) \in \text{past}_S(t) \times \text{past}_I(t) \mid \text{match}(x, y) \} \]

- \( M \) – one-to-one relation
- \( \forall x \in \text{past}_S(t-\Delta t) \ \exists y \in \text{past}_I(t) . (x, y) \in M \)
- \( \forall y \in \text{past}_I(t-\Delta t) \ \exists x \in \text{past}_S(t) . (x, y) \in M \)
- \( \forall (x, y), (x', y') \in M . x \leq x' \Rightarrow \theta(y) \leq \theta(y') \)
Reaction arbiters

\[
\text{arbiter}_1 = \min_{\leq}(X)
\]

\[X \subseteq V_S\]

\[
\text{arbiter}_2(y, X) = \begin{cases} 
  x, \exists x \in X \cdot \mathsf{match}(x, y) \\
  \epsilon, \forall x \in X \cdot \mathsf{match}(x, y)
\end{cases}
\]

\[y \in V_I, X \subseteq V_S\]
Conformance relation checking
Conclusion

• Based on the theory of traces and partially ordered multisets method of on-the-fly analysis of hardware systems has been developed

• The method has been implemented in C++TESK Testing ToolKit and has been successfully used in a number of projects

• Future research is connected with failure diagnostics: giving hints to localization of bugs
THANK YOU

- Any questions?