AMT: a Property-based Tool for Monitoring Analog Systems

Dejan Ničković
Verimag

Oded Maler
Verimag
Overview

- Introduction
- STL/PSL Specification Language
  - Analog Layer
  - Temporal Layer
  - Distance-based Operators
- Checking STL/PSL Properties
  - Offline Marking
  - Incremental Marking
- AMT Tool
- FLASH Memory Case Study
Introduction

- Verification of discrete systems
  - Model checking TL specs
  - Central role in algorithmic verification
  - Efficient algorithms for LTL, CTL, PSL etc.

- Verification of real-time systems
  - Emptiness checking of timed automata
    - KRONOS, UPPAAL, IF etc.
  - Many variants of real-time logics
    - MTL, MITL, TCTL etc.
  - Only TCTL used in a real-time verification tool

- Lightweight verification
  - Systems may be too complex to verify exhaustively
    - Software
    - Very large digital systems
    - Many real-time systems etc.

- Property monitors
  - Generated automatically from the specification
  - Observe individual simulation traces and check whether the property is violated
  - Incomplete but more reliable method than manual visual inspection of simulation traces
Introduction

- Verification of discrete systems
  - Model checking TL specs
  - Central role in algorithmic verification
  - Efficient algorithms for LTL, CTL, PSL etc.

- Verification of real-time systems
  - Emptiness checking of *timed automata*
    - KRONOS, UPPAAAL, IF etc.
  - Many variants of real-time logics
    - MTL, MITL, TCTL etc.
  - Only TCTL used in a real-time verification tool

- Lightweight verification
  - Systems may be too complex to verify exhaustively
    - Software
    - Very large digital systems
    - Many real-time systems etc.
  - Property monitors
    - Generated automatically from the specification
    - Observe individual simulation traces and check whether the property is violated
    - Incomplete but more reliable method than manual visual inspection of simulation traces
Introduction

- Verification of discrete systems
  - Model checking TL specs
  - Central role in algorithmic verification
  - Efficient algorithms for LTL, CTL, PSL etc.

- Verification of real-time systems
  - Emptiness checking of timed automata
    - KRONOS, UPPAAL, IF etc.
  - Many variants of real-time logics
    - MTL, MITL, TCTL etc.
  - Only TCTL used in a real-time verification tool

- Lightweight verification
  - Systems may be too complex to verify exhaustively
    - Software
    - Very large digital systems
    - Many real-time systems etc.
  - Property monitors
    - Generated automatically from the specification
    - Observe individual simulation traces and check whether the property is violated
    - Incomplete but more reliable method than manual visual inspection of simulation traces
**Introduction**

- Verification of continuous systems
  - Manual inspection of simulation traces
    - Dominant technique
    - Requires experienced specialists
    - Error prone
  - Exhaustive analog verification
    - Powerful formalisms such as hybrid automata
    - Limited scalability
- **Our approach:** Property-based lightweight verification of continuous signals
**Introduction**

- Verification of continuous systems
  - Manual inspection of simulation traces
    - Dominant technique
    - Requires experienced specialists
    - Error prone
  - Exhaustive analog verification
    - Powerful formalisms such as hybrid automata
    - Limited scalability

- Our approach: Property-based lightweight verification of continuous signals
Introduction

- Verification of continuous systems
  - Manual inspection of simulation traces
    - Dominant technique
    - Requires experienced specialists
    - Error prone
  - Exhaustive analog verification
    - Powerful formalisms such as hybrid automata
    - Limited scalability

- Our approach: Property-based lightweight verification of continuous signals
Signals

- Finite length signal $\xi$ defined over an abstract domain $\mathbb{D}$
  - Partial function $\xi : T \rightarrow \mathbb{D}$
  - Length of $\xi$ is $r$ ($|\xi| = r$)
  - $\xi[t] = \bot$ when $t \geq |\xi|$
  - **Boolean signals:** $(\xi_b) \mathbb{D} = \mathbb{B}$
  - **Continuous signals:** $(\xi_a) \mathbb{D} = \mathbb{R}$

- **Restriction** of a signal $\xi$ to length $d$
  $$\xi' = \langle \xi \rangle_d \text{ iff } \xi'[t] = \begin{cases} \xi[t] & \text{if } t < d \\ \bot & \text{otherwise} \end{cases}$$

- **Concatenation** $\xi = \xi_1 \cdot \xi_2$
  $$\xi[t] = \begin{cases} \xi_1[t] & \text{if } t < r_1 \\ \xi_2[t - r_1] & \text{otherwise} \end{cases}$$

- **$d$-suffix** of a signal $\xi$, $\xi' = d \setminus \xi$
  $$\xi'[t] = \xi[t + d] \text{ for every } t \in [0, |\xi| - d)$$
Signals

- **Minkowski sum** and **difference** of two sets $P_1$ and $P_2$ are defined as
  \[ P_1 \oplus P_2 = \{ x_1 + x_2 : x_1 \in P_1, x_2 \in P_2 \} \]
  \[ P_1 \ominus P_2 = \{ x_1 - x_2 : x_1 \in P_1, x_2 \in P_2 \} \].

- **Projection** of the signal $\xi$ on the dimension with domain $\mathbb{B}$ which corresponds to the proposition $p$, $\xi_p = \pi_p(\xi)$
  
  ✦ Likewise $\xi_s = \pi_s(\xi)$ is the projection of the signal $\xi$ on the dimension with domain $\mathbb{R}$ which corresponds to the continuous variable $s$

- **Signal representation**
  
  ✦ Boolean signals:
    - Non-Zeno finite length signals admit finite representation
    - Sequence of adjacent intervals with value constant in each interval
  
  ✦ Continuous signals:
    - Do not admit an exact finite representation
    - But, numerical simulators produce a **finite** collection of sampling points
    - The signal value at missing points in time is interpolated
Extension of real-time temporal logic MITL with analog constructs

- PSL-like layered approach
  - Analog layer: allows reasoning about continuous signals
  - Temporal layer: relates the temporal behavior of input traces

- “Communication” between two layers via static abstractions
  - Partitioning of the continuous state space according to the satisfaction of some inequality constraints on the continuous variables

- Targeted to be used in lightweight verification
  - PSL-like finitary interpretation of temporal operators
Extension of real-time temporal logic MITL with analog constructs

PSL-like layered approach

- **Analog layer**: allows reasoning about continuous signals
- **Temporal layer**: relates the temporal behavior of input traces

“Communication” between two layers via static abstractions

- Partitioning of the continuous state space according to the satisfaction of some inequality constraints on the continuous variables

Targeted to be used in *lightweight* verification

- PSL-like *finitary interpretation* of temporal operators
Extension of real-time temporal logic MITL with analog constructs

PSL-like layered approach

- **Analog layer**: allows reasoning about continuous signals
- **Temporal layer**: relates the temporal behavior of input traces

“Communication” between two layers via static abstractions

- Partitioning of the continuous state space according to the satisfaction of some inequality constraints on the continuous variables

Targeted to be used in lightweight verification

- PSL-like finitary interpretation of temporal operators
**STL/Psl Specification Language**

- Extension of real-time temporal logic MITL with analog constructs
- PSL-like layered approach
  - Analog layer: allows reasoning about continuous signals
  - Temporal layer: relates the temporal behavior of input traces
- “Communication” between two layers via static abstractions
  - Partitioning of the continuous state space according to the satisfaction of some inequality constraints on the continuous variables
- Targeted to be used in lightweight verification
  - PSL-like finitary interpretation of temporal operators
**STL/PSL: Analog Layer**

- **Syntax:**

  \[ \phi ::= s \mid \text{shift}(\phi, k) \mid \phi_1 \ast \phi_2 \mid \phi \ast c \mid \text{abs}(\phi) \]

  where \( s \) belongs to a set \( S = \{s_1, s_2, \ldots, s_n\} \) of continuous variables, \( \ast \in \{+, -, \ast\} \), \( c \in \mathbb{Q} \) and \( k \in \mathbb{Q}^+ \).

- **Semantics:**

  \[
  \begin{align*}
  &s[t] = \pi_s(\xi)[t] \\
  &\text{shift}(\phi, k)[t] = \phi[t + k] \\
  &\phi_1 \ast \phi_2[t] = \phi_1[t] \ast \phi_2[t] \\
  &\phi \ast c[t] = \phi[t] \ast c \\
  &\text{abs}(\varphi)[t] = \begin{cases} \\
  \phi[t] & \text{if } \phi[t] \geq 0 \\
  -\phi[t] & \text{otherwise} \\
  \end{cases}
  \end{align*}
  \]

- **Pragmatic choice of analog operators**
  - Based on the feedback of analog designers
  - Can be naturally extended
**STL/PSL: Analog Layer**

- **Syntax:**

  \[ \phi ::= s \mid \text{shift}(\phi, k) \mid \phi_1 \circ \phi_2 \mid \phi \circ c \mid \text{abs}(\phi) \]

  where \( s \) belongs to a set \( S = \{s_1, s_2, \ldots, s_n\} \) of continuous variables, \( \circ \in \{+, -, \ast\} \), \( c \in \mathbb{Q} \) and \( k \in \mathbb{Q}^+ \).

- **Semantics:**

  \[
  \begin{align*}
  s[t] & = \pi_s(\xi)[t] \\
  \text{shift}(\phi, k)[t] & = \phi[t + k] \\
  (\phi_1 \ast \phi_2)[t] & = \phi_1[t] \ast \phi_2[t] \\
  (\phi \ast c)[t] & = \phi[t] \ast c \\
  \text{abs}(\varphi)[t] & = \begin{cases} 
  \phi[t] & \text{if } \phi[t] \geq 0 \\
  -\phi[t] & \text{otherwise}
  \end{cases}
  \end{align*}
  \]

- **Pragmatic choice of analog operators**

  ✦ Based on the feedback of analog designers
  ✦ Can be naturally extended
STL/PSL: Analog Layer

- **Syntax:**
  \[
  \phi ::= s \mid \text{shift}(\phi, k) \mid \phi_1 \star \phi_2 \mid \phi \star c \mid \text{abs}(\phi)
  \]
  where \( s \) belongs to a set \( S = \{s_1, s_2, \ldots, s_n\} \) of continuous variables, \( \star \in \{+, -, \ast\} \), \( c \in \mathbb{Q} \) and \( k \in \mathbb{Q}^+ \).

- **Semantics:**
  \[
  s[t] = \pi_s(\xi)[t] \\
  \text{shift}(\phi, k)[t] = \phi[t + k] \\
  (\phi_1 \star \phi_2)[t] = \phi_1[t] \star \phi_2[t] \\
  (\phi \star c)[t] = \phi[t] \star c \\
  \text{abs}(\varphi)[t] = \begin{cases} 
  \phi[t] & \text{if } \phi[t] \geq 0 \\
  -\phi[t] & \text{otherwise}
  \end{cases}
  \]

- **Pragmatic choice of analog operators**
  ✦ Based on the feedback of analog designers
  ✦ Can be naturally extended
Syntax:

\[ \varphi ::= p | \phi \circ c | \text{not } \varphi \mid \varphi_1 \text{ or } \varphi_2 \mid \text{eventually! } \varphi \mid \text{eventually![}a:b\text{]} \varphi \mid \varphi_1 \text{ until! } \varphi_2 \mid \varphi_1 \text{ until![}a:b\text{]} \varphi_2 \]

where \( p \) belongs to a set \( P = \{p_1, p_2, \ldots, p_n\} \) of propositional variables, \( a, b, c \in \mathbb{Q} \) and \( \circ \in \{>, \geq, <, \leq\} \).

Semantics: The satisfaction relation \( (\xi, t) \models \varphi \), indicating that signal \( \xi \) satisfies \( \varphi \) at time \( t \) is defined inductively as follows:

\[
(\xi, t) \models \text{eventually! } \varphi \quad \text{iff} \quad \exists t' \geq t \text{ st } t' < |\xi| \text{ and } (\xi, t') \models \varphi \\
(\xi, t) \models \text{eventually![}a:b\text{]} \varphi \quad \text{iff} \quad \exists t' \in t + [a, b] \text{ st } t' < |\xi| \text{ and } (\xi, t') \models \varphi \\
(\xi, t) \models \varphi_1 \text{ until! } \varphi_2 \quad \text{iff} \quad \exists t' \geq t \text{ st } t' < |\xi| \text{ and } (\xi, t') \models \varphi_2 \text{ and } \\
\forall t'' \in [t, t'] (\xi, t'') \models \varphi_1 \\
(\xi, t) \models \varphi_1 \text{ until![}a:b\text{]} \varphi_2 \quad \text{iff} \quad \exists t' \in t + [a, b] \text{ st } t' < |\xi| \text{ and } (\xi, t') \models \varphi_2 \text{ and } \\
\forall t'' \in [t, t'] (\xi, t'') \models \varphi_1
STL/PSL: Temporal Layer

- **Syntax:**

\[
\varphi ::= p | \phi \circ c | \text{not } \varphi | \varphi_1 \text{ or } \varphi_2 | \text{eventually! } \varphi |
\]

\[
\text{eventually!}[a:b] \varphi | \text{eventually}[a:b] \varphi |
\]

\[
\varphi_1 \text{ until! } \varphi_2 | \varphi_1 \text{ until!}[a:b] \varphi_2
\]

where \( p \) belongs to a set \( P = \{p_1, p_2, \ldots, p_n\} \) of propositional variables, \( a, b, c \in \mathbb{Q} \)
and \( \circ \in \{>,\geq,<,\leq\} \).

- **Semantics:** The satisfaction relation \( (\xi, t) \models \varphi \), indicating that signal \( \xi \) satisfies \( \varphi \) at time \( t \) is defined inductively as follows:

\[
(\xi, t) \models \text{eventually! } \varphi \quad \text{iff} \quad \exists t' \geq t \text{ st } t' < |\xi| \text{ and } (\xi, t') \models \varphi
\]

\[
(\xi, t) \models \text{eventually!}[a:b] \varphi \quad \text{iff} \quad \exists t' \in t \oplus [a,b] \text{ st } t' < |\xi| \text{ and } (\xi, t') \models \varphi
\]

\[
(\xi, t) \models \varphi_1 \text{ until! } \varphi_2 \quad \text{iff} \quad \exists t' \geq t \text{ st } t' < |\xi| \text{ and } (\xi, t') \models \varphi_2 \text{ and } \forall t'' \in [t,t'] (\xi, t'') \models \varphi_1
\]

\[
(\xi, t) \models \varphi_1 \text{ until!}[a:b] \varphi_2 \quad \text{iff} \quad \exists t' \in t \oplus [a,b] \text{ st } t' < |\xi| \text{ and } (\xi, t') \models \varphi_2 \text{ and } \forall t'' \in [t,t'] (\xi, t'') \models \varphi_1
\]
**Motivation:** Enrich STL/PSL with **metric properties**

- Compare waveforms with some reference signal that specifies a desired behavior
- Distance function (metric)
  - Quantifies numerically the resemblance of two signals

\[
\begin{align*}
\text{distance}(\phi_1, \phi_2, c) & = \text{abs}(\phi_1 - \phi_2) \leq c \\
\text{distance}(\phi_1, \phi_2, c, t, T) & = \text{abs}(\phi_1 - \phi_2) > c \rightarrow \text{eventually!}[\leq t] \\
& \quad \text{always}[\leq T-t]\text{abs}(\phi_1 - \phi_2) \leq c \\
\text{distance}(\varphi_1, \varphi_2, t, T) & = (\varphi_1 \oplus \varphi_2) \rightarrow \text{eventually!}[\leq t] \\
& \quad \text{always}[\leq T-t](\varphi_1 \iff \varphi_2)
\end{align*}
\]
**STL/PSL: Distance-based Operators Example**
Marking: a procedure that determines the truth values of each subformula of an STL/PSL specification at every time instant $t$

- Doubly-recursive procedure, on time and the structure of the formula

Two algorithms for checking STL/PSL properties:

- Offline marking: input is fully available
- Incremental marking: input is dynamically observed

Based on [MalerN04]
**Marking**: a procedure that determines the truth values of each subformula of an STL/PSL specification at every time instant $t$

- Doubly-recursive procedure, on time and the structure of the formula

**Two algorithms for checking STL/PSL properties:**

- **Offline marking**: input is fully available
- **Incremental marking**: input is dynamically observed

Based on [MalerN04]
Marking: a procedure that determines the truth values of each subformula of an STL/PSL specification at every time instant $t$

- Doubly-recursive procedure, on time and the structure of the formula

Two algorithms for checking STL/PSL properties:

- Offline marking: input is fully available
- Incremental marking: input is dynamically observed

Based on [MalerN04]
Offline Marking

<table>
<thead>
<tr>
<th>input</th>
<th>STL/PSL Temporal Formula $\varphi$ and signal $\xi$</th>
</tr>
</thead>
</table>

\[
\text{switch } \varphi \text{ do} \\
\begin{align*}
\text{case } p & : \\
\chi_\varphi & := \pi_p(\xi); \\
\text{end case} \\
\text{OP}_2(\varphi_1, \varphi_2) & : \\
\text{OFFLINE}(\varphi_1, \varphi_2); \\
\chi_\varphi & := \text{COMBINE}(\text{OP}_2, \chi_{\varphi_1}, \chi_{\varphi_2}); \\
\text{end case} \\
\end{align*}
\]

- **Inputs:**
  - Multidimensional signal $\xi$
  - STL/PSL specification $\varphi$
  - Compute, from **bottom-up**, a signal $\chi_\psi(\xi)$ for each subformula $\psi$ of $\phi$
  - **COMBINE** computes from input signals a new signal based on the specific operation
### Offline Marking

**Input:** STL/PSL Temporal Formula $\varphi$ and signal $\xi$

```plaintext
switch $\varphi$ do
    case $p$
        $\chi_\varphi := \pi_p(\xi)$;
    end
    case $\text{OP}_2(\varphi_1, \varphi_2)$
        $\text{OFFLINE}(\varphi_1, \varphi_2)$;
        $\chi_\varphi := \text{COMBINE}(\text{OP}_2, \chi_{\varphi_1}, \chi_{\varphi_2})$;
    end
end
```

- **Inputs:**
  - Multidimensional signal $\xi$
  - STL/PSL specification $\varphi$

- **Compute, from **bottom-up**, a signal $\chi_\psi(\xi)$ for each subformila $\psi$ of $\varphi$**

- **COMBINE** computes from input signals a new signal based on the specific operation.
Offline Marking

---

**input**: STL/PSL Temporal Formula $\varphi$ and signal $\xi$

```plaintext
switch $\varphi$ do
  case $p$
    $\chi_{\varphi} := \pi_p(\xi)$;
  end
  case $\text{OP}_2(\varphi_1, \varphi_2)$
    $\text{OFFLINE}(\varphi_1, \varphi_2)$;
    $\chi_{\varphi} := \text{COMBINE}(\text{OP}_2, \chi_{\varphi_1}, \chi_{\varphi_2})$;
  end
end
```

- **Inputs:**
  - Multidimensional signal $\xi$
  - STL/PSL specification $\varphi$
  - Compute, from **bottom-up**, a signal $\chi_\psi(\xi)$ for each subformula $\psi$ of $\varphi$
  - **COMBINE** computes from input signals a new signal based on the specific operation
Disjunction

- Refine the intervals of $\chi_{\phi_1}$ and $\chi_{\phi_2}$ so that the mutual values of both signals become uniform in every interval
- Compute the disjunction interval-wise
- Merge the adjacent intervals having the same value

Bounded Eventually

- For every positive interval $I \in \chi_{\phi_1}$
- Compute its **back shifting** $I - [a, b] \cap T$
- Merge the overlapping intervals in $\chi_{\varphi}$
**Disjunction**

- Refine the intervals of $\chi_{\phi_1}$ and $\chi_{\phi_2}$ so that the mutual values of both signals become uniform in every interval
- Compute the disjunction interval-wise
- Merge the adjacent intervals having the same value

**Bounded Eventually**

- For every positive interval $I \in \chi_{\varphi_1}$
- Compute its **back shifting** $I - [a, b] \cap \mathbb{T}$
- Merge the overlapping intervals in $\chi_{\varphi}$
**Disjunction**

- Refine the intervals of $\chi_{\phi_1}$ and $\chi_{\phi_2}$ so that the mutual values of both signals become uniform in every interval
- Compute the disjunction interval-wise
- Merge the adjacent intervals having the same value

**Bounded Eventually**

- For every positive interval $I \in \chi_{\varphi_1}$
- Compute its **back shifting** $I - [a, b] \cap T$
- Merge the overlapping intervals in $\chi_{\varphi}$
**Disjunction**

- Refine the intervals of $\chi_{\phi_1}$ and $\chi_{\phi_2}$ so that the mutual values of both signals become uniform in every interval
- Compute the disjunction interval-wise
- Merge the adjacent intervals having the same value

**Bounded Eventually**

- For every positive interval $I \in \chi_{\varphi_1}$
- Compute its **back shifting** $I - [a, b] \cap T$
- Merge the overlapping intervals in $\chi_\varphi$
Disjunction

- Refine the intervals of $\chi_{\phi_1}$ and $\chi_{\phi_2}$ so that the mutual values of both signals become uniform in every interval
- Compute the disjunction interval-wise
- Merge the adjacent intervals having the same value

Bounded Eventually

- For every positive interval $I \in \chi_{\varphi_1}$
- Compute its back shifting $I - [a, b] \cap T$
- Merge the overlapping intervals in $\chi_{\varphi}$

1

2

1 - $\chi_{\varphi_1}$
2 - eventually!$[a,b] \chi_{\varphi_1}$
**Disjunction**
- Refine the intervals of $\chi_{\phi_1}$ and $\chi_{\phi_2}$ so that the mutual values of both signals become uniform in every interval
- Compute the disjunction interval-wise
- Merge the adjacent intervals having the same value

**Bounded Eventually**
- For every positive interval $I \in \chi_{\phi_1}$
- Compute its **back shifting** $I - [a, b] \cap \mathbb{T}$
- Merge the overlapping intervals in $\chi_\varphi$

![Diagram showing disjunction and bounded eventually]
**Disjunction**
- Refine the intervals of $\chi_{\phi_1}$ and $\chi_{\phi_2}$ so that the mutual values of both signals become uniform in every interval
- Compute the disjunction interval-wise
- Merge the adjacent intervals having the same value

**Bounded Eventually**
- For every positive interval $I \in \chi_{\phi_1}$
- Compute its **back shifting** $I - [a, b] \cap T$
- Merge the overlapping intervals in $\chi_{\phi}$

---

1 - $\chi_{\phi_1}$
2 - eventually $[a, b] \chi_{\phi_1}$
**Disjunction**
- Refine the intervals of $\chi_{\phi_1}$ and $\chi_{\phi_2}$ so that the mutual values of both signals become uniform in every interval
- Compute the disjunction interval-wise
- Merge the adjacent intervals having the same value

**Bounded Eventually**
- For every positive interval $I \in \chi_{\phi_1}$
- Compute its back shifting $I - [a, b] \cap T$
- Merge the overlapping intervals in $\chi_{\phi}$
- **Pointwise arithmetic operation on two signals**
- Take the union of their sampling points
- Extend each signal to the new points by interpolation
- Apply the operation on each pair of sampling points

\[
\chi_{\varphi_1} - \chi_{\varphi_2}
\]
**COMBINE: Arithmetic Operations**

- Pointwise arithmetic operation on two signals
- Take the union of their sampling points
- Extend each signal to the new points by interpolation
- Apply the operation on each pair of sampling points

\[ \chi_{\varphi_1} \]

\[ \chi_{\varphi_2} \]

\[ \chi_{\varphi_1 - \varphi_2} \]
Pointwise arithmetic operation on two signals
- Take the union of their sampling points
- Extend each signal to the new points by interpolation
- Apply the operation on each pair of sampling points
Incremental Marking

\[
\text{input} : \text{STL/PSL Temporal Formula } \varphi \text{ and increment } \Delta \xi
\]

\[
\text{switch } \varphi \text{ do}
\]

\[
\text{case } p
\]

\[
\Delta \varphi := \Delta \varphi \cdot \pi_p(\Delta \xi);
\]

\[
\text{endcase}
\]

\[
\text{end}
\]

\[
\text{case } \text{OP}_2(\varphi_1, \varphi_2)
\]

\[
\text{INCREMENTAL } (\varphi_1, \varphi_2);
\]

\[
\alpha_\varphi := \text{COMBINE}(\text{OP}_2, \chi \varphi_1, \chi \varphi_2);
\]

\[
d := \lvert \alpha_\varphi \rvert;
\]

\[
\Delta \varphi := \Delta \varphi \cdot \alpha_\varphi;
\]

\[
\chi \varphi_1 := \chi \varphi_1 \cdot \langle \Delta \varphi_1 \rangle d;
\]

\[
\Delta \varphi_1 := d \setminus \Delta \varphi_1;
\]

\[
\chi \varphi_2 := \chi \varphi_2 \cdot \langle \Delta \varphi_2 \rangle d;
\]

\[
\Delta \varphi_2 := d \setminus \Delta \varphi_2;
\]

\[
\text{endcase}
\]

\[
\text{end}
\]
**Incremental Marking**

**input**: STL/PSL Temporal Formula $\varphi$ and increment $\Delta_\xi$

**switch** $\varphi$ **do**

  **case** $p$
        | $\Delta_\varphi := \Delta_\varphi \cdot \pi_p(\Delta_\xi)$;
  **end**
  **case** $\text{OP}_2(\varphi_1, \varphi_2)$
       | $\text{INCREMENTAL}(\varphi_1, \varphi_2)$;
       | $\alpha_\varphi := \text{COMBINE}(\text{OP}_2, \chi_{\varphi_1}, \chi_{\varphi_2})$;
       | $d := |\alpha_\varphi|$;
       | $\Delta_\varphi := \Delta_\varphi \cdot \alpha_\varphi$;
       | $\chi_{\varphi_1} := \chi_{\varphi_1} \cdot \langle \Delta_\varphi_1 \rangle d$;
       | $\Delta_\varphi_1 := d \backslash \Delta_\varphi_1$;
       | $\chi_{\varphi_2} := \chi_{\varphi_2} \cdot \langle \Delta_\varphi_2 \rangle d$;
       | $\Delta_\varphi_2 := d \backslash \Delta_\varphi_2$
  **end**

**end**
**Incremental Marking**

**input**: STL/PSL Temporal Formula $\varphi$ and increment $\Delta_\xi$

switch $\varphi$ do
  case $p$
    $\Delta_\varphi := \Delta_\varphi \cdot \pi_p(\Delta_\xi)$;
  end
  case $\text{OP}_2(\varphi_1, \varphi_2)$
    $\text{INCREMENTAL} (\varphi_1, \varphi_2)$;
    $\alpha_\varphi := \text{COMBINE}(\text{OP}_2, \chi\varphi_1, \chi\varphi_2)$;
    $d := |\alpha_\varphi|$;
    $\Delta_\varphi := \Delta_\varphi \cdot \alpha_\varphi$;
    $\chi\varphi_1 := \chi\varphi_1 \cdot \langle\Delta\varphi_1\rangle d$;
    $\Delta\varphi_1 := d \setminus \Delta\varphi_1$;
    $\chi\varphi_2 := \chi\varphi_2 \cdot \langle\Delta\varphi_2\rangle d$;
    $\Delta\varphi_2 := d \setminus \Delta\varphi_2$
  end
end
### Incremental Marking

**input** : STL/PSL Temporal Formula $\varphi$ and increment $\Delta_\xi$

```plaintext
switch $\varphi$ do
  | case $p$
  |     $\Delta \varphi := \Delta \varphi \cdot \pi_p(\Delta_\xi)$;
  | end
  | case $\text{OP}_2(\varphi_1, \varphi_2)$
  |     $\text{INCREMENTAL } (\varphi_1, \varphi_2)$;
  |     $\alpha_{\varphi} := \text{COMBINE}(\text{OP}_2, \chi_{\varphi_1}, \chi_{\varphi_2})$;
  |     $d := |\alpha_{\varphi}|$;
  |     $\Delta \varphi := \Delta \varphi \cdot \alpha_{\varphi}$;
  |     $\chi_{\varphi_1} := \chi_{\varphi_1} \cdot \langle \Delta \varphi_1 \rangle d$;
  |     $\Delta \varphi_1 := d \setminus \Delta \varphi_1$;
  |     $\chi_{\varphi_2} := \chi_{\varphi_2} \cdot \langle \Delta \varphi_2 \rangle d$;
  |     $\Delta \varphi_2 := d \setminus \Delta \varphi_2$
end
```
**Incremental Marking**

input : STL/PSL Temporal Formula $\varphi$ and increment $\Delta \xi$

switch $\varphi$ do
  case $p$
    $\Delta \varphi := \Delta \varphi \cdot \pi_p(\Delta \xi)$;
  end
  case $\text{OP}_2(\varphi_1, \varphi_2)$
    $\text{INCREMENTAL}(\varphi_1, \varphi_2)$;
    $\alpha \varphi := \text{COMBINE}(\text{OP}_2, \chi \varphi_1, \chi \varphi_2)$;
    $d := |\alpha \varphi|$;
    $\Delta \varphi := \Delta \varphi \cdot \alpha \varphi$;
    $\chi \varphi_1 := \chi \varphi_1 \cdot \langle \Delta \varphi_1 \rangle d$
    $\Delta \varphi_1 := d \setminus \Delta \varphi_1$;
    $\chi \varphi_2 := \chi \varphi_2 \cdot \langle \Delta \varphi_2 \rangle d$
    $\Delta \varphi_2 := d \setminus \Delta \varphi_2$
  end
end
**Incremental Marking**

**input**: STL/PSL Temporal Formula $\varphi$ and increment $\Delta_\xi$

switch $\varphi$ do

  case $p$
  | $\Delta_\varphi := \Delta_\varphi \cdot \pi_p(\Delta_\xi)$;
  end

  case $\text{OP}_2(\varphi_1, \varphi_2)$
  | $\text{INCREMENTAL}((\varphi_1, \varphi_2))$;
  | $\alpha_\varphi := \text{COMBINE}($,$\varphi_1, \varphi_2)$);
  | $d := |\alpha_\varphi|$;
  | $\Delta_\varphi := \Delta_\varphi \cdot \alpha_\varphi$;
  | $\chi_\varphi_1 := \chi_\varphi_1 \cdot \langle \Delta_\varphi_1 \rangle d$;
  | $\Delta_\varphi_1 := d \setminus \Delta_\varphi_1$;
  | $\chi_\varphi_2 := \chi_\varphi_2 \cdot \langle \Delta_\varphi_2 \rangle d$;
  | $\Delta_\varphi_2 := d \setminus \Delta_\varphi_2$
  end

end
Advantages of the incremental algorithm

- Often more memory efficient
- Determined parts of the signal may be discarded
- Early detection of errors
Stand alone tool for lightweight verification of properties on continuous signals

Inputs:
- STL/PSL specification
- Input signals (Boolean or continuous)
  - From a file (raw, vcd and out format)
  - Dynamic inputs through TCP/IP packets

Property evaluation:
- Offline
- Incremental

Visual evaluation of results
AMT Tool Overview

- Stand alone tool for lightweight verification of properties on continuous signals

**Inputs:**
- STL/PSL specification
- Input signals (Boolean or continuous)
  - From a file (**raw**, **vcd** and **out** format)
  - Dynamic inputs through TCP/IP packets

**Property evaluation:**
- Offline
- Incremental

- Visual evaluation of results
Stand alone tool for lightweight verification of properties on continuous signals

**Inputs:**
- **STL/PSL specification**
- **Input signals (Boolean or continuous)**
  - From a file (*raw, vcd* and *out* format)
  - Dynamic inputs through TCP/IP packets

**Property evaluation:**
- Offline
- Incremental

Visual evaluation of results
Stand alone tool for lightweight verification of properties on continuous signals

**Inputs:**
- STL/PSL specification
- Input signals (Boolean or continuous)
  - From a file (**raw**, **vcd** and **out** format)
  - Dynamic inputs through TCP/IP packets

**Property evaluation:**
- Offline
- Incremental

**Visual evaluation of results**
**Flash Memory Case Study**

- **Provided by STM Italy**
- **Why Flash memory?**
  - **Analog** circuit that implements **digital** behavior
  - Good connection between **analog** and **digital** worlds

- **Different modes**
  - Programming, reading, erasing, etc.

- **Characteristic signals**
  - **bl**: bit line terminal
  - **pw**: p-well terminal
  - **wl**: word line
  - **s**: source terminal
  - **vt**: threshold voltage of cell
  - **id**: drain current of cell

- **Correct functioning in a given mode determined by the behavior of the characteristic signals**
- **5 properties specifying the correct behavior**
**Flash Memory Case Study**

- Provided by STM Italy
- Why Flash memory?
  - **Analog** circuit that implements **digital** behavior
  - Good connection between **analog** and **digital** worlds

- Different modes
  - Programming, reading, erasing, etc.

- Characteristic signals
  - **bl**: bit line terminal
  - **pw**: p-well terminal
  - **wl**: word line
  - **s**: source terminal
  - **vt**: threshold voltage of cell
  - **id**: drain current of cell

- Correct functioning in a given mode determined by the behavior of the characteristic signals
- 5 properties specifying the correct behavior
Why Flash memory?
- Analog circuit that implements digital behavior
- Good connection between analog and digital worlds

Different modes
- Programming, reading, erasing, etc.

Characteristic signals
- bl: bit line terminal
- pw: p-well terminal
- wl: word line
- s: source terminal
- vt: threshold voltage of cell
- id: drain current of cell

Correct functioning in a given mode determined by the behavior of the characteristic signals

5 properties specifying the correct behavior
Why Flash memory?

- Analog circuit that implements digital behavior
- Good connection between analog and digital worlds

Different modes

- Programming, reading, erasing, etc.

Characteristic signals

- \( \text{bl} \): bit line terminal
- \( \text{pw} \): p-well terminal
- \( \text{wl} \): word line
- \( \text{s} \): source terminal
- \( \text{vt} \): threshold voltage of cell
- \( \text{id} \): drain current of cell

Correct functioning in a given mode determined by the behavior of the characteristic signals

- 5 properties specifying the correct behavior
Flash Memory Case Study

- Provided by STM Italy
- Why Flash memory?
  - Analog circuit that implements digital behavior
  - Good connection between analog and digital worlds

- Different modes
  - Programming, reading, erasing, etc.

- Characteristic signals
  - bl: bit line terminal
  - pw: p-well terminal
  - wl: word line
  - s: source terminal
  - vt: threshold voltage of cell
  - id: drain current of cell

- Correct functioning in a given mode determined by the behavior of the characteristic signals
- 5 properties specifying the correct behavior
vprop erasing {
    define b:erasing_cond :=
        a:wl <= -6 and a:pw > 5;

erasing assert:
    always (b:erasing_cond ->
        (distance (a:s,a:pw,0.1)
         and (a:bl-a:pw)>-0.83));
}
## Tool Evaluation

<table>
<thead>
<tr>
<th>name</th>
<th>pgm sim # intervals</th>
<th>erase sim # intervals</th>
</tr>
</thead>
<tbody>
<tr>
<td>wl</td>
<td>34829</td>
<td>283624</td>
</tr>
<tr>
<td>pw</td>
<td>25478</td>
<td>283037</td>
</tr>
<tr>
<td>s</td>
<td>33433</td>
<td>282507</td>
</tr>
<tr>
<td>bl</td>
<td>32471</td>
<td>139511</td>
</tr>
<tr>
<td>id</td>
<td>375</td>
<td>n/a</td>
</tr>
</tbody>
</table>

Table 1: Input Size
### Tool Evaluation

#### Table 1: Input Size

<table>
<thead>
<tr>
<th>name</th>
<th>pgm sim # intervals</th>
<th>erase sim # intervals</th>
</tr>
</thead>
<tbody>
<tr>
<td>wl</td>
<td>34829</td>
<td>283624</td>
</tr>
<tr>
<td>pw</td>
<td>25478</td>
<td>283037</td>
</tr>
<tr>
<td>s</td>
<td>33433</td>
<td>282507</td>
</tr>
<tr>
<td>bl</td>
<td>32471</td>
<td>139511</td>
</tr>
<tr>
<td>id</td>
<td>375</td>
<td>n/a</td>
</tr>
</tbody>
</table>

#### Table 2: Offline Algorithm Evaluation

<table>
<thead>
<tr>
<th>property</th>
<th>time (s)</th>
<th># intervals</th>
</tr>
</thead>
<tbody>
<tr>
<td>programming1</td>
<td>0.14</td>
<td>99715</td>
</tr>
<tr>
<td>programming2</td>
<td>0.42</td>
<td>405907</td>
</tr>
<tr>
<td>p-well</td>
<td>0.12</td>
<td>89071</td>
</tr>
<tr>
<td>decay</td>
<td>0.50</td>
<td>594709</td>
</tr>
<tr>
<td>erasing</td>
<td>2.35</td>
<td>2968578</td>
</tr>
</tbody>
</table>

---

**AMT: a Property-based Tool for Monitoring Analog Systems**
### Tool Evaluation

#### Table 1: Input Size

<table>
<thead>
<tr>
<th>name</th>
<th>pgm sim # intervals</th>
<th>erase sim # intervals</th>
</tr>
</thead>
<tbody>
<tr>
<td>wl</td>
<td>34829</td>
<td>283624</td>
</tr>
<tr>
<td>pw</td>
<td>25478</td>
<td>283037</td>
</tr>
<tr>
<td>s</td>
<td>33433</td>
<td>282507</td>
</tr>
<tr>
<td>bl</td>
<td>32471</td>
<td>139511</td>
</tr>
<tr>
<td>id</td>
<td>375</td>
<td>n/a</td>
</tr>
</tbody>
</table>

#### Table 2: Offline Algorithm Evaluation

<table>
<thead>
<tr>
<th>property</th>
<th>time (s)</th>
<th># intervals</th>
</tr>
</thead>
<tbody>
<tr>
<td>programming1</td>
<td>0.14</td>
<td>99715</td>
</tr>
<tr>
<td>programming2</td>
<td>0.42</td>
<td>405907</td>
</tr>
<tr>
<td>p-well</td>
<td>0.12</td>
<td>89071</td>
</tr>
<tr>
<td>decay</td>
<td>0.50</td>
<td>594709</td>
</tr>
<tr>
<td>erasing</td>
<td>2.35</td>
<td>2968578</td>
</tr>
</tbody>
</table>

#### Table 3: Offline/Incremental Space Requirement Comparison

<table>
<thead>
<tr>
<th>Property</th>
<th>Offline t = total # intervals</th>
<th>Incremental m = max # active intervals</th>
<th>m/t * 100</th>
</tr>
</thead>
<tbody>
<tr>
<td>programming1</td>
<td>99715</td>
<td>65700</td>
<td>65.9</td>
</tr>
<tr>
<td>programming2</td>
<td>594709</td>
<td>242528</td>
<td>40.8</td>
</tr>
<tr>
<td>p-well</td>
<td>89071</td>
<td>8</td>
<td>0.01</td>
</tr>
<tr>
<td>decay</td>
<td>594709</td>
<td>279782</td>
<td>47.1</td>
</tr>
</tbody>
</table>
Conclusion

Main contributions:

- **AMT** tool that monitors temporal properties of continuous signals
  - Description of properties in **STL/PSL** specification language
  - Offline and incremental algorithms
  - Integration with numerical simulations via simulation dump files or **TCP/IP** link

- **FLASH** memory case study
  - Validates the tool and the approach
  - Shows the automation and efficiency of monitoring continuous signals
Main contributions:

- **AMT** tool that monitors temporal properties of continuous signals
  - Description of properties in **STL/PSL** specification language
  - Offline and incremental algorithms
  - Integration with numerical simulations via simulation dump files or **TCP/IP** link
- **FLASH** memory case study
  - Validates the tool and the approach
  - Shows the automation and efficiency of monitoring continuous signals