Timing Is Everything

Joël Ouaknine

Department of Computer Science
Oxford University

BCS Meeting, Oxford

17 May 2012
“In theory, there is no difference between theory and practice. In practice, there is.”

Jan L.A. van de Snepscheut
Ariane 5 Explosion, French Guyana, 1996
NASA Mars Missions, 1997–2004

- 1997: Mars Rover loses contact
- 1999: Mars Climate Orbiter is lost
- 1999: Mars Polar Lander is lost
- 2004: Mars Rover freezes
Intel Pentium FDIV Bug, 1994
Northeast Blackout, 2003

Wednesday, August 13

Thursday, August 14
December 2006: DaimlerChrysler recalls 128,000 Pacifica sports utility vehicles because of a problem with the software governing the fuel pump and power train control. The defect could cause the engine to stall unexpectedly. [Washington Post]
Automated Verification

“A Grand Challenge for computing research.”

Sir Tony Hoare, 2003
Automated Verification

“A Grand Challenge for computing research.”

Sir Tony Hoare, 2003

Now one of a small handful of areas ‘targetted for growth’ by UK funding council EPSRC.
Automated Verification

“Nobody is going to run into a friend’s office with a program verification. Nobody is going to sketch a verification out on a paper napkin... One can feel one’s eyes glaze over at the very thought.”

Rich de Millo, Richard Lipton, Alan Perlis, 1979
“Nobody is going to run into a friend’s office with a program verification. Nobody is going to sketch a verification out on a paper napkin... One can feel one’s eyes glaze over at the very thought.”

Rich de Millo, Richard Lipton, Alan Perlis, 1979

“The success of program verification as a generally applicable and completely reliable method for guaranteeing program performance is not even a theoretical possibility.”

James H. Fetzer

Program Verification: The Very Idea, CACM 31(9), 1988
Automated Verification: A High-Level Overview

1. Modelling
2. Specification

Verification

- G(a ==> Fb)
- G(!c && d)
- Properties:
  - 1. bug found
  - 2. system ok
  - parameter values
  - performance indices
SLAM

TERMINATOR
proof tools for termination and liveness
```c
int Ack(int m, int n) {
    if (m == 0)
        return n + 1;
    else if (n == 0)
        return Ack(m - 1, 1);
    else
        return Ack(m - 1, Ack(m, n - 1));
}
```
```c
int Ack(int m, int n) {
    if (m == 0)
        return n + 1;
    else if (n == 0)
        return Ack(m - 1, 1);
    else
        return Ack(m - 1, Ack(m, n - 1));
}
```

$\text{Ack}(n, n) : 1, 3, 7, 61, 2^{2^{2^{2^2}}} - 3, \underbrace{2^{2^{2\ldots^2}}}_{\text{Ack}(5,4)+3} - 3$
Timing Is Everything
A Login Protocol
A Login Protocol

START

\( login\_name \quad x:=0 \quad \rightarrow \quad VALIDATE \)
A Login Protocol

START

login_name

\[ x := 0 \]

VALIDATE

restart

\[ x \geq 60? \]

log_pw_wrong

pw_correct

START VALIDATE

LOG_ERRORDELAY

connected

\[ x \leq 60? \]
A Login Protocol

START

log_in_name

x:=0

restart

x ≥ 60?

VALIDATE

pw_correct

x<60?

connected
A Login Protocol

START

- login_name
  - \(x := 0\)

- restart
  - \(x \geq 60?\)

VALIDATE

- pw_correct
  - \(x < 60?\)

LOG_ERROR

- \(x < 60?\)

- pw_wrong

connected
A Login Protocol

START

login_name
x:=0
restart
x≥60?

VALIDATE

pw_correct
x<60?

connected

DELAY

log_pw_wrong
x:=0

LOG_ERROR

x<60?
pw_wrong
A Login Protocol

START

login_name

x:=0

restart

x≥60?

x≥10? restart

DELAY

log_pw_wrong

x:=0

VALIDATE

pw_correct

x<60?

connected

LOG_ERROR_DELAY

<60?:=0

:=0

<60?
BMW Hydrogen 7
\(\square (\textit{PEDAL} \rightarrow \diamond \textit{BRAKE})\)
BMW Hydrogen 7

\[ \square (PEDAL \rightarrow \Diamond BRAKE) \]

\[ \square (PEDAL \rightarrow \Diamond [25,40] BRAKE) \]
Timed Automata

Introduced by Rajeev Alur at Stanford during his PhD under David Dill:

Timed Automata

START

login_name
\[ x := 0 \]

restart
\[ x \geq 60? \]

x \geq 10?
restart

DELAY

log_pw_wrong
\[ x := 0 \]

LOG_ERROR

VALIDATE

pw_correct
\[ x < 60? \]

connected

x < 60?
pw_wrong

x \leq 60?

START VALIDATE

LOG_ERROR_DELAY

connected
Timed Automata

Time is modelled as the non-negative reals, $\mathbb{R}_{\geq 0}$. 
Timed Automata

Time is modelled as the non-negative reals, $\mathbb{R}_{\geq 0}$.

Theorem (Alur, Courcourbetis, Dill 1990)

Reachability is decidable (in fact PSPACE-complete).
Timed Automata

Time is modelled as the non-negative reals, $\mathbb{R}_{\geq 0}$.

**Theorem (Alur, Courcourbetis, Dill 1990)**

*Reachability is decidable (in fact PSPACE-complete).*

Unfortunately:

**Theorem (Alur & Dill 1990)**

*Language inclusion is undecidable for timed automata.*
“The paradigmatic idea of the automata-theoretic approach to verification is that we can compile high-level logical specifications into an equivalent low-level finite-state formalism.”

Moshe Vardi
“The paradigmatic idea of the automata-theoretic approach to verification is that we can compile high-level logical specifications into an equivalent low-level finite-state formalism.”

Moshe Vardi

**Theorem**

Automata are closed under all Boolean operations. Moreover, the language inclusion problem \([ L(A) \subseteq L(B) \text{ ?} ]\) is decidable.
An Uncomplementable Timed Automaton

$L(A)$: $A$ cannot be complemented: There is no timed automaton $B$ with $L(B) = L(A)$. 
An Uncomplementable Timed Automaton

$L(A)$:

$A$ cannot be complemented.
There is no timed automaton $B$ with $L(B) = L(A)$. 
An Uncomplementable Timed Automaton

$L(A)$:

\[ A: \begin{array}{c}
\text{a} \\
\text{a} \\
\text{a} \\
\text{a}
\end{array}
\begin{array}{c}
\xrightarrow{x:=0} \\
\xrightarrow{x=1?}
\end{array}
\]

$L(A)$: 

\[ \text{L(A):} \]

\[ |<-- \quad 1 \quad -->| \]
An Uncomplementable Timed Automaton

\[ A : \]

\[ L(A): \]

\[ L(A): \]

\[ A \text{ cannot be complemented: There is no timed automaton } B \text{ with } L(B) = L(A). \]
An Uncomplementable Timed Automaton

\[ A : \]

\[ A \text{ cannot be complemented:} \]
There is no timed automaton \( B \) with \( L(B) = \overline{L(A)} \).
Metric Temporal Logic

$\square(a \rightarrow \diamond_{[0,1]} b)$
Metric Temporal Logic

\[ \Box(a \rightarrow \Diamond_{[0,1]} b) \]
Metric Temporal Logic

\[
\square (a \rightarrow \diamond_{[0,1]} b)
\]

Does the timed word satisfy the specification?
Metric Temporal Logic

\( \square(a \rightarrow \Diamond_{[0,1]} b) \)

Does the timed word satisfy the specification?
Metric Temporal Logic

\[ \square(a \rightarrow \lozenge_{[0,1]} b) \]

Does the timed word satisfy the specification?
Metric Temporal Logic

$\square(a \rightarrow \diamond_{[0,1]} b)$

Does the timed word satisfy the specification?
Metric Temporal Logic

\( \square (a \rightarrow \Diamond_{[0,1]} b) \)

Does the timed word satisfy the specification?
Metric Temporal Logic

\[ \Box (a \rightarrow \Diamond_{[0,1]} b) \]

Does the timed word satisfy the specification?
Metric Temporal Logic

\[ \square (a \rightarrow \diamond [0,1] \ b) \]

Does the timed word satisfy the specification?
Metric Temporal Logic

\[ \Box (a \rightarrow \Diamond_{[0,1]} b) \]

Does the timed word satisfy the specification?
Metric Temporal Logic

$\square (a \to \Diamond_{[0,1]} b)$

Does the timed word satisfy the specification?
Metric Temporal Logic

\[ \square(a \rightarrow \diamond_{[0,1]} b) \]

Does the timed word satisfy the specification? Yes.
Given a timed automaton $A$ and a Metric Temporal Logic specification $\varphi$, does every timed word of $A$ satisfy $\varphi$?
Real-Time Model Checking

Given a timed automaton $A$ and a Metric Temporal Logic specification $\varphi$, does every timed word of $A$ satisfy $\varphi$?

- For about 15 years ($\sim 1990–2005$), the real-time model-checking problem was widely claimed in the literature to be undecidable.
Real-Time Model Checking

Given a timed automaton $A$ and a Metric Temporal Logic specification $\varphi$, does every timed word of $A$ satisfy $\varphi$?

- For about 15 years (~1990–2005), the real-time model-checking problem was widely claimed in the literature to be undecidable.
- In 2005, James Worrell and I showed decidability through the development of the theory of timed alternating automata.
\(\Box (a \rightarrow \Diamond_{=1} b)\)
\[\Box (a \rightarrow \Diamond =_1 b)\]
\(\square (a \rightarrow \Diamond_{=1} b)\)
\(\square(a \rightarrow \Diamond_1 b)\)

\[
\begin{align*}
\text{State } s_0 & : a, b \rightarrow x := 0, & s_1 & : a \rightarrow x < 1?, & s_2 & : \{a, b\} \rightarrow x = 1? b
\end{align*}
\]
$\square (a \rightarrow \diamond_{=1} b)$
$$\square \left( a \rightarrow \Diamond_{=1} b \right)$$
\(\square (a \rightarrow \Diamond =_1 b)\)
\[ \square(a \rightarrow \diamond =_1 b) \]
\( \Box (a \rightarrow \diamond =_1 b) \)
\( \square (a \rightarrow \Diamond =_1 b) \)
\[ \square (a \rightarrow \Diamond =_1 b) \]

Diagram:

- State transitions:
  - From \( s_0 \):
    - with label 'a' to \( s_1 \) with \( x := 0 \)
    - with label 'b' to \( s_0 \)
  - From \( s_1 \):
    - with label 'a' to \( s_1 \) with \( x < 1? \)
    - with label 'b' to \( s_2 \) with \( x = 1? \)
  - From \( s_2 \):
    - with label 'a' to \( s_1 \) with \( x < 1? \)
    - with label 'b' to \( s_2 \)

Line diagram:

- Transition values:
  - \( s_0 \):
    - \( 0 \rightarrow a \rightarrow 0.3 \rightarrow b \rightarrow 0.5 \rightarrow a \rightarrow 0.8 \rightarrow 1 \rightarrow b \rightarrow 1.3 \rightarrow a \rightarrow 1.4 \rightarrow b \rightarrow 1.8 \rightarrow 2 \)
\[ \square (a \rightarrow \Diamond =_{1} b) \]
Real-Time Model Checking: A High-Level Algorithm

Real-time model checking problem
Real-Time Model Checking: A High-Level Algorithm

Real-time model checking problem

⇓

Alternating timed automaton emptiness problem
Real-Time Model Checking: A High-Level Algorithm

Real-time model checking problem

⇓

Alternating timed automaton emptiness problem

⇓

Halting problem for Turing machine with insertion errors
\[ x^2 - y^5 - (xy)^2 - t^2 - (xt)^2 = (yt)^2 + (xyt)^5 \]

\[ \text{over } \langle x, yt \rangle = D_{10} \]

\[ \text{Genus 2} \]
Theorem

The subword order over a finite alphabet is a well-quasi order.

Higman’s Lemma

Theorem

The subword order over a finite alphabet is a well-quasi order.


“HIGMAN” is a subword of “HIGHMOUNTAIN”.
**Theorem**

The subword order over a finite alphabet is a well-quasi order.


“HIGMAN” is a subword of “HIGHMOUNTAIN”.

Theorem

The subword order over a finite alphabet is a well-quasi order.


“HIGMAN” is a subword of “HIGHMOUNTAIN”.

Any infinite sequence of words $W_1, W_2, W_3, \ldots$ must eventually have two words, $W_i$ and $W_{i+k}$, such that the first is a subword of the second.
Higman’s Lemma

Theorem
The subword order over a finite alphabet is a well-quasi order.


“HIGMAN” is a subword of “HIGHMOUNTAIN”.

Any infinite sequence of words \( W_1, W_2, W_3, \ldots \) must eventually have two words, \( W_i \) and \( W_{i+k} \), such that the first is a subword of the second.

- aba
Higman’s Lemma

Theorem
The subword order over a finite alphabet is a well-quasi order.


“HIGMAN” is a subword of “HIGHMOUNTAIN”.

Any infinite sequence of words $W_1, W_2, W_3, \ldots$ must eventually have two words, $W_i$ and $W_{i+k}$, such that the first is a subword of the second.

- aba, abbb
Higman’s Lemma

Theorem
The subword order over a finite alphabet is a well-quasi order.


“HIGMAN” is a subword of “HIGHMOUNTAIN”.

Any infinite sequence of words $W_1, W_2, W_3, \ldots$ must eventually have two words, $W_i$ and $W_{i+k}$, such that the first is a subword of the second.

- aba, abbb, baab
Higman’s Lemma

Theorem
The subword order over a finite alphabet is a well-quasi order.


“HIGMAN” is a subword of “HIGHMOUNTAIN”.

Any infinite sequence of words \( W_1, W_2, W_3, \ldots \) must eventually have two words, \( W_i \) and \( W_{i+k} \), such that the first is a subword of the second.

- aba, abbb, baab, aa
Higman’s Lemma

Theorem
The subword order over a finite alphabet is a well-quasi order.


“HIGMAN” is a subword of “HIGHMOUNTAIN”.

Any infinite sequence of words $W_1, W_2, W_3, \ldots$ must eventually have two words, $W_i$ and $W_{i+k}$, such that the first is a subword of the second.

- aba, abbb, baab, aa, ba
Higman’s Lemma

Theorem

The subword order over a finite alphabet is a well-quasi order.


“HIGMAN” is a subword of “HIGHMOUNTAIN”.

Any infinite sequence of words $W_1$, $W_2$, $W_3$, ... must eventually have two words, $W_i$ and $W_{i+k}$, such that the first is a subword of the second.

- aba, abbb, baab, aa, ba, bbb
Higman’s Lemma

Theorem

The subword order over a finite alphabet is a well-quasi order.


“HIGMAN” is a subword of “HIGHMOUNTAIN”.

Any infinite sequence of words $W_1, W_2, W_3, \ldots$ must eventually have two words, $W_i$ and $W_{i+k}$, such that the first is a subword of the second.

- aba, abbb, baab, aa, ba, bbb, abb
Higman’s Lemma

Theorem

The subword order over a finite alphabet is a well-quasi order.


“HIGMAN” is a subword of “HIGHMOUNTAIN”.

Any infinite sequence of words $W_1, W_2, W_3, \ldots$ must eventually have two words, $W_i$ and $W_{i+k}$, such that the first is a subword of the second.

- aba, abbb, baab, aa, ba, bbb, abb, ab
Higman’s Lemma

Theorem
The subword order over a finite alphabet is a well-quasi order.


“HIGMAN” is a subword of “HIGHMOUNTAIN”.

Any infinite sequence of words \( W_1, W_2, W_3, \ldots \) must eventually have two words, \( W_i \) and \( W_{i+k} \), such that the first is a subword of the second.

- aba, abbb, baab, aa, ba, bbb, abb, ab, a
Higman’s Lemma

Theorem
The subword order over a finite alphabet is a well-quasi order.


“HIGMAN” is a subword of “HIGHMOUNTAIN”.

Any infinite sequence of words $W_1$, $W_2$, $W_3$, … must eventually have two words, $W_i$ and $W_{i+k}$, such that the first is a subword of the second.

- aba, abbb, baab, aa, ba, bbb, abb, ab, a, bb
Higman’s Lemma

Theorem

*The subword order over a finite alphabet is a well-quasi order.*


“HIGMAN” is a subword of “HIGHMOUNTAIN”.

Any infinite sequence of words $W_1, W_2, W_3, \ldots$ must eventually have two words, $W_i$ and $W_{i+k}$, such that the first is a subword of the second.

- aba, abbb, baab, aa, ba, bbb, abb, ab, a, bb, b
The Halting Problem for Faulty Turing Machines

\[(q_0, <>)\]
The Halting Problem for Faulty Turing Machines

$(q_0, <>)$
The Halting Problem for Faulty Turing Machines

$(q_0, <>)$
The Halting Problem for Faulty Turing Machines

(q₀, <>)

(q₀, <>)
The Halting Problem for Faulty Turing Machines

\[(q_0, <>) \rightarrow (q, W)\]

...
The Halting Problem for Faulty Turing Machines

\[(q_0, <>) \rightarrow (q, W) \rightarrow (q, W')\]
The Halting Problem for Faulty Turing Machines

\[ (q_0, <>), <>) \rightarrow (q, W) \]

is a subword of

\[ (q, W') \]
The Halting Problem for Faulty Turing Machines

\((q_0, <>)\) \quad \Rightarrow \quad \ldots \quad \Rightarrow \quad (q, W) \quad \Rightarrow \quad (q, W') \quad \Rightarrow \quad \text{halt}

is a subword of

\((q, W)\)
The Halting Problem for Faulty Turing Machines

(q₀, <>)

... (q, W)

is a subword of

(q, W')

halt

halt
The Halting Problem for Faulty Turing Machines

\[(q_0, \langle\rangle) \rightarrow (q, W)\]

is a subword of

\[(q, W')\]

...
Theorem

The real-time model-checking problem for Metric Temporal Logic is \textit{decidable} (under the pointwise semantics).
Theorem

The real-time model-checking problem for Metric Temporal Logic is **decidable** (under the pointwise semantics).

The complexity is provably **non-primitive recursive**. In particular, it grows faster than Ackermann’s function in the worst case.
From Timed Alternating Automata to Efficient Runtime Monitoring Algorithms

□(PEDAL $\rightarrow$ ♦[25,40] BRAKE)
Quantitative Verification: From Model Checking to Model Measuring
Quantitative Verification: From Model Checking to Model Measuring
Quantitative Verification:
From Model Checking to Model Measuring

quantitative systems

qualitative systems

runtime monitoring model checking module checking parametric synthesis model measuring