Scalable (yet Precise) Timing Analysis: Of Course Model-Based!

Wang Yi
Uppsala University
(ETAPS 2015, London)

Can P finish its execution within D sec’s?
Joint work with my students:

Nan Guan
Martin Stigge
Pontus Ekberg
Jakaria Abdullah
OUTLINE

• Modeling with graph-based models

• **Scalable** Analysis (pseudo-polynomial time)
  – for the tractable cases

• **Efficient** Analysis (combinatorial refinement)
  – for the intractable cases
Embedded Systems

Timing Analysis

• What is the maximal delay at each component?
• What is the maximal end-to-end delay?

Example size of applications:
- 80-100 tasks (Ind. Robots)
- 30-50 tasks (FMS, THALES)
TACAS, Aarhus, April 1995

UPPAAL

Johan Bengtsson
Kim Larsen
Fredrik Larsson
Paul Pettersson
Wang Yi

Photo: Kim Larsen, Aalborg Univ.
Timed Systems

- **UPPAAL, UPPAAL-Tiga (FUSC):** Timed automata (Uppsala Univ., Aalborg Univ.)
  - manipulate **UPPAAL** XML (GPL-3, Python)
  - Yggdrasil (?, ?): UML (subset) -> **Uppaal**, intended for test generation (Aalborg Univ.)
- **METAMOC** (GPL-3, Python): WCET Analysis of ARM Processors using Real-Time model checking (Aalborg Univ.)
- **SARTS** (?, Java): Model Based Schedulability Analysis of Real-Time Systems (Rouen, UPPAAL, Aalborg Univ.)
- **OPAAL** (GPL-3, Python): distributed/parallel (discrete time) model checker for networks of timed automata using MPI
- **ECDAR** (FUSC): timed interface theory (Aalborg, INRIA, ITU)
- **PyECDAR** (GPL-2, Python): solve timed games based on timed automata models (ITU)
- **IOA** (MIT, Java): I/O automata formal language (MIT)
- **TEMPO** (closed, Java): Formal language for modeling distributed systems w/ w/o timing constraints as collections of interacting state machines, i.e., timed input/output automata (TIOA) (UIUC)
  - Tempo2HSal (?, Python): Tempo (.tioa) -> HSal (.hsla) translator (SRI)
- **ATAS** (GPL-3, Python): Alternating 1-clock (fully decidable) Timed Automata Solver
- **PPL binding** (GPL-2, Python): for Parma Polyhedral Lib features some specific methods for Timed Automata analysis
- **MCPTA** (FUSC, ?): Probabilistic Timed Automata model checker for MoDeST | **UPPAAL** | PRISM - maps on PRISM (Saarland Univ.)
- **SSAT** (?): Abstraction refinement model checker for Timed Automata based on extended SAT-solving, **UPPAAL**-like input format (Univ. Stuttgart, CWI)
- **Fortuna** (GPL-3, C++/Eclipse): MC priced probabilistic timed automata (PPTAs) (Univ. Twente)
- **COSPAN** (?, ?): Automata-theoretic verification of coordinating processes with timing constraints (UPenn)
- **Romeo:** timed Petri nets (IRCCyN)
- **ExSched:** develop operating system schedulers for VxWorks and Linux w/o modifying the underlying kernel ([Malardalen Univ. http://www.es.mdh.se/staff/197-Mikael__berg])
- **RTComposer** (Java): classes and utilities for predictable real-time scheduling (BenGurion, UPenn)
- **ASTRAL:** MC of real-time systems (UCSB)
- **PAT** (?, C#): simulator, MC, refinement checker for concurrent and RT systems (Nanyang Tech. Univ.)
- **HCMC** (?, C++): Compositional model checking for real-time systems (ENS-Cachan)

Could these tools solve this problem?
State of the art

I can’t solve the problem, neither can all these famous Model-Checkers
Timing Analysis

**Sequential Case** (WCET Analysis)

**Concurrent Case** (Response Time Analysis)

- **Non-deterministic releases**
- WCRT = WCET
- D1, D2, D3
Timing Analysis

**Sequential Case (WCET Analysis)**

- Assume the WCET of each task is given (resource budget)
- How to estimate the Worst-Case Response Time of a task?

**Concurrent Case (Response Time Analysis)**

Non-deterministic releases

- WCRT = WCET
- D1
- D2
- D3
Modeling for (System-Level) Timing Analysis

- The event arrival patterns e.g. using timed automata
- Synchronization between components,
- Resource arbitration, protocols and scheduling algorithms
- The resource demands or budget e.g. the WCET
- The timing constraints e.g. deadlines
Timed Models

- **Timed Petri Nets**, early 80s
  - Time Intervals over transition firing
- **Process Algebras**, 80s – 90s
  - Delays + untimed models e.g. Milner’s CCS
- **Timed Automata**, early 90s
  - finite automata + clock constraints
- **Real-Time Task Models since 70s**
  - Layland and Liu’s periodic tasks, 1973
  - The variants of L&L model [RTSS community]
- **Real-Time Programming e.g. Ada 83**
  - Delay, Tasking, Run-Time System
- **Hybrid Systems/Automata, Modelica ... UML RT ... (yesterday)**
Liu and Layland’s Model, 1973

A system is a set of periodic tasks each described by two numbers:
- \( e \): the worst case execution time (WCET)
- \( P \): the minimum inter-release delay (implicit deadline)

- The workload of each task: \( e/p \)
- The system workload or utilization: \( U = \sum e_i/p_i \)

**Feasibility (i.e. EDF-schedulability):** no deadline miss if \( U \leq 1 \)

**Fixed-priority Schedulability:** no deadline miss if \( U \leq n(2^{1/n} - 1) \)

The well-known Rate-Monotonic Scheduling
Hierarchy of Models

Feasibility test

difficult

efficient

Expressiveness

high

low

Strongly (co)NP-hard
Pseudo-Polynomial

two integers
L&L
implicit deadline
ALL these models are “tractable” but have limited expressiveness

[Survey, RTS journal, Martin and Wang, 2015]
Example: Tree/DAG-task model

Restrictions of Non-Cyclic RRT

- Tasks are still *recurrent*
  - Always revisit source $J_1$
  - *No cycles allowed!*

- Consequences:
  - No *local loops*
  - Not compositional (for modes etc.)
Restrictions of Tree/DAG model

- Tasks are still *recurrent*
  - Always revisit source $J_1$
  - *No cycles allowed!*

- Consequences:
  - No *local loops*

- Not *compositionality* (for modes etc.)
Restrictions of Tree/DAG model

- Tasks are still *recurrent*
  - Always revisit source $J_1$
  - *No cycles allowed!*

- Consequences:
  - No *local loops*
  - Not *compositional* (for modes etc.)
Further extension without crossing the “tractable” borderline?
The Digraph Real-Time Model (DRT)

- Pairs on nodes are the WCET and deadline on the task code
  e.g. A has WCET 2 and relative deadline 4
- Numbers on edges are the minimum inter-release delays

The WCET, deadlines and release delays should be ensured by the Ada run-time system

In Ada Tasking:

```ada
Procedure PA
  "release A"
  Delay(2);
  PA
Procedure PB
  "release B"
  Delay(25);
Procedure PC
  "release C"
  If "condition"
     then Delay(10); PA
  else Delay (11); PB
```

[Stigge et al, RTAS 2011]
DRT: Semantics  (any path of the graph is a possible behavior)

Path $\pi = (v_4)$

Demand bound: (10, 5)
DRT: Semantics (any path of the graph is a possible behavior)

Path $\pi = (v_4, v_2)$

Demand bound: (10, 5)  Demand bound: (28, 6)
DRT: Semantics  (any path of the graph is a possible behavior)

Path $\pi = (v_4, v_2, v_3)$

Workload: (10, 5)

Workload: (28, 6)

Demand bound: (43, 9)
Workload of a DRT

Demand Bounds Function (dbf)
A system model = a set of DRT’s modeling the components.

The system workload:
Hierarchy of Models

Feasibility test

- difficult
- efficient

Expressiveness

- high
- low

Models:
- arbitrary graph
- tree
- cycle graph
- three integers
- two integers
- sporadic
- DRT
- RB
- GMF
- L&L

Branching, loops, ...
branching
different job types
explicit deadline
implicit deadline
Hierarchy of Models

[Stigge et al, RTAS 2011]

Feasibility test: efficient vs. difficult

Expressiveness: low vs. high

- **arbitrary graph**: DRT (branching, loops, ...)
- **tree**: RB (branching)
- **cycle graph**: GMF (different job types)
- **three integers**: sporadic (explicit deadline)
- **two integers**: L&L (implicit deadline)

Strongly (co)NP-hard
Pseudo-Polynomial
Complexity Result  [RTAS 2011]

Theorem (S. et al., 2011)

*For DRT task systems $\tau$ with a utilization bounded by any $c < 1$, feasibility can be decided in pseudo-polynomial time.*

Pseudo-polynomial time = Tractable/efficient

Ideas for feasibility analysis

• Characterize the system workload ...
• If the worst-case workload is over 100%, it is over-loaded, implying deadline miss

Units of work a CPU can compute over time (100%)

Workload
Of course, if the **BLUE line** is always below the **RED**, the system should work well without deadline miss!

**How to check this?**

Units of work a CPU can compute over time (100 %)

*dbf*

**Workload**

*Time*
Here is the intuition why “Pseudo-P”

If the utilization (long-term rates of DRT’s) of a system is bounded by a constant $c < 1$, any deadline miss, if exists, must appear before a pseudo-polynomial upper bound:
Calculating the Bound

\[ \sum \text{dbf}_T(t) \]

- Linear bound for \( \text{dbf}(t) \)
  - Slope: Less than 1
- Intersection with \( t \) gives bound \( D \)
- Check only up to \( D \)

\[ \text{dbf}(t) \leq t \cdot U(\tau) + e^{sum} \]

\[ D = \frac{e^{sum}}{1 - U(\tau)} \]
A system model = a set of DRT’s modeling the components

The system workload:
Evaluation: Runtime vs. Utilization

Setting:
- Randomly generated task sets
- 1-30 tasks, 5-10 vertices per task, branching degree 1-3, ...
• How about synchronization?
  – the analysis without considering synchronization is SAFE!
  – Precise analysis possible with “Combinatorial Refinement”
• How about “static priority scheduling”?
Hierarchy of Models

[Stigge/Wang, ECRTS 2012]
## Summary

<table>
<thead>
<tr>
<th>Models</th>
<th>Analysis Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Feasibility i.e. EDF-Schedulability</td>
</tr>
<tr>
<td>General graphs (Di-graph)</td>
<td>Pseudo-P</td>
</tr>
<tr>
<td>Trees/DAGs</td>
<td>Pseudo-P</td>
</tr>
<tr>
<td>Cyclic graphs (GMF)</td>
<td>Pseudo-P</td>
</tr>
<tr>
<td>Sporadic (L&amp;L, deadline≠period)</td>
<td>Pseudo-P</td>
</tr>
<tr>
<td>L&amp;L (periodic)</td>
<td>Linear</td>
</tr>
</tbody>
</table>

For systems with utilization bounded by a constant less than 1 (or below 100%)

Otherwise Strongly coNP-complete

!! The problem open for 25 years, theoretically interesting !!

[ECRTS 2015, Pontus Ekberg and Wang Yi]
Combinatorial Refinement
solving “Combinatorial Problems”
(for timing analysis, it works very well!)
A system model = a set of DRT’s modeling the components

This works perfectly for feasibility checking: the global worst case can be constructed from the local worst cases
A system model = a set of DRT’s modeling the components

In general, each component may have a set of behaviors e.g. Paths or traces
A system model = a set of DRT's modeling the components

Often, we have to check some property guaranteed by all the combinations of individual local behaviors and thus may have to enumerate ... (combinatorial explosion)
Construct an **Abstract Tree** for each individual component
Construct an **Abstract Tree**
for each individual component

Any non-leaf node **father** should be an over-approximation of his **sons** in the sense that

\[(\ldots \ldots \text{father} \ldots \ldots) \text{ sat } F \Rightarrow (\ldots \ldots \text{any son} \ldots \ldots) \text{ sat } F\]
Construct an **Abstract Tree** for each individual component.

For instance, the Combination of all roots satisfies the desired property implies that all combinations of the leaves satisfy the same property.

$$(\text{roots}) \text{ sat } F \Rightarrow (\text{any leave, any leave, ... any leave}) \text{ sat } F$$
Abstract Request Functions

![Graph with nodes and edges labeled with pairs and numbers.](image)

Graph with nodes labeled as $v_1$, $v_2$, $v_3$, $v_4$, $v_5$. Edges are labeled with pairs $(a,b)$ and numbers. The graph shows relationships between nodes.

Graph showing $rf(t)$ and $rf_{(v_4,v_2,v_3)}$ over time $t$. The graph displays the function values at different time points.
Abstract Request Functions

The diagram above illustrates the relationships between different nodes, labeled as $v_1, v_2, v_3, v_4, v_5$. The edges connecting these nodes are labeled with values such as 11, 10, 20, and 15, indicating some form of service request or function value. The nodes are also connected with labels such as $(1, 8), (2, 5), (3, 8), (1, 5), (5, 10)$, which might represent specific requests or conditions.

The graph shows a timeline $rf(t)$ on the x-axis, indicating the time at which different requests or functions are made or completed. The graph for $rf(v_4, v_2, v_3)$ and $rf(v_5, v_4, v_2)$ are plotted, with each curve showing the progression of these functions over time.
Abstract Request Functions

[Diagram showing a graph with nodes labeled v1, v2, v3, v4, v5 and edges with labels such as 11, 10, 15, 20, 20, 10.]

The graph illustrates the relationships between different requests with labeled edges. The diagram includes a graph with labeled nodes and edges, along with a graph showing the function $rf(t)$ for different request functions $arf$, $rf(v_4,v_2,v_3)$, and $rf(v_5,v_4,v_2)$. The graph is oriented to show the temporal progression, with time $t$ on the x-axis and the request function $rf(t)$ on the y-axis.
Abstraction Tree for each DRT

Define an abstraction tree per task:

- Leaves are concrete $rf$
- Each node: maximum function of child nodes
- Root is maximum of all $rf$
Abstraction Tree for each DRT

Define an abstraction tree per task:
- Leaves are concrete rf
- Each node: maximum function of child nodes
- Root is maximum of all rf
Abstraction Tree for each DRT

Define an abstraction tree per task:

- Leaves are concrete rf
- Each node: maximum function of child nodes
- Root is mrf, maximum of all rf
Combinatorial Abstraction Refinement

New Algorithm:
- Test one combination of all $mrf$.
- If fp-feasible: done
- Otherwise: Replace one $mrf$ with all child nodes, get 2 new combinations to test
- Repeat until:
  - All combinations show fp-feasibility, or
  - A combination of leaves shows non-fp-feasibility
Evaluation: Runtime vs. Utilization

Comparing runtimes of
- EDF-test using dbf (pseudo-polynomial)
- SP-test based on *Combinatorial Abstraction Refinement*
Evaluation: Tested vs. Total Combinations

$10^5$ samples of single-job tests.
- Executed tests: in 99.9% of all cases, less than 100
- Total combinations possible: up to $10^{12}$
Conclusions

“Code is Art” – Daniel Licata

• Model is “Abstract Art”, the key for scalable and precise analysis
  – it should be as simple as possible but not simpler
  – it should be as expressive as possible but not more

• Digraph Model instead of Timed Automata?
  – Expressive enough to capture Ada tasking
  – Efficient analysis possible: Pseudo-polynomial

• Combinatorial Refinement works well for timing problems
  – In particular when local search space can be abstracted & ordered
  – other verification problems?

• Current work
  – Synchronization and resource sharing
  – Multiprocessor mapping and scheduling
  – TIMES++, a new tool based on Digraph, aiming at industrial applications
The WCET Analysis Problem

• A fundamental problem for embedded systems design
  – Worst-Case Execution Time (WCET) analysis

• Challenges (“termination” doesn’t make the problem easy)
  – “too many input”  too many execution paths (difficult to find the worst-case)
  – hardware features e.g. caches (“the HW state” results in different execution times)
WCET Analysis

- **Path Analysis**
  - which path leads to the WCET?
  - well-known technique by ILP
  - need to know the timing delay of each instruction

- **Architecture Analysis**
  - **Cache Analysis:**
    - Is a memory access hit or miss?
  - other factors like pipeline...
WCET Analysis

- Path Analysis
  - which path leads to the WCET?
  - well-known technique by ILP
  - need to know the timing delay of each instruction

- Architecture Analysis [Survey 2015 wang et al]
  - Cache Analysis:
    Is a memory access hit or miss?
    - AH: always hit
    - FM: first miss, then always hit
    - AM: always miss
    - NC: not classified

  - other factors like pipeline...
WCET Analysis

• Path Analysis
  – which path leads to the WCET?
  – well-known technique by ILP
  – need to know the timing delay of each instruction

• Architecture Analysis  [Survey 2015 wang et al]
  – Cache Analysis:
    Is a memory access hit or miss?
    • AH: always hit
    • FM: first miss, then always hit
    • AM: always miss
    • NC: not classified → always miss

  – other factors like pipeline ...
Timing Analysis

Sequential Case (WCET Analysis)

- Assume the WCET of each task is given (resource budget)
- How to estimate the Worst-Case Response Time of a task?

Concurrent Case (Response Time Analysis)