16 May 2025 | MeFoSyLoMa Seminar | Paris, France

Verifying Timed Properties of Programs in IoT nodes using Parametric Time Petri Nets

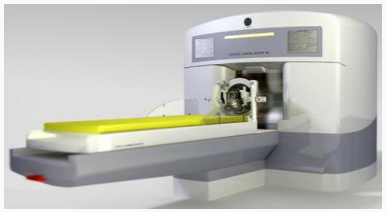
Paper presented at SAC-SVT 2025

Étienne André, Jean-Luc Béchennec, Sudipta Chattopadhyay, Sébastien Faucou, Didier Lime, ***Dylan Marinho***, Olivier H. Roux, Jun Sun

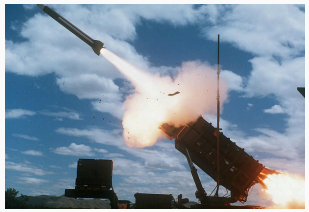
Sorbonne Université, CNRS UMR 7606, LIP6

Context: Verifying complex timed systems

- **Critical** systems: Failures may result in **dramatic** consequences
- Need for early bug detection
  - Bugs discovered when final testing: **expensive**
  - Need for a thorough **specification** and **verification** phase



Therac-25
(USA, 1980s)



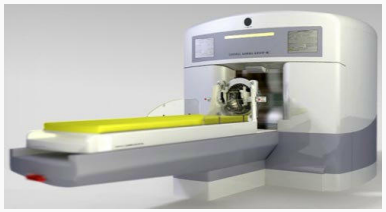
MIM-104 Pat. Mis. Fail.
(Iraq, 1991)

Sleipner A offshore platform
(Norway, 1991)

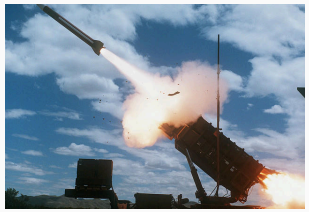
Ariane flight V88
(France, 1996)

Context: Verifying complex timed systems

- **Critical** systems: Failures may result in **dramatic** consequences
- Need for early bug detection
  - Bugs discovered when final testing: **expensive**
  - Need for a thorough **specification** and **verification** phase



Therac-25
(USA, 1980s)



MIM-104 Pat. Mis. Fail.
(Iraq, 1991)

Sleipner A offshore platform
(Norway, 1991)

Ariane flight V88
(France, 1996)

- Verification is needed to ensure the absence of bugs

Context: Timing attacks over programs

```c
// input pwd    : Real password
// input attempt: Tentative password
for (i = 0; i < min(len(pwd), len(attempt)); i++) {
    if(pwd[i] != attempt[i]){
        return false
    }
}
return true
```

pwd	M	e	F	o	S	y	L	o	M	a
attempt	M	e	S	y	M	a	F	o		

🕔 Execution time (ET):

Context: Timing attacks over programs

```c
// input pwd    : Real password
// input attempt: Tentative password
for (i = 0; i < min(len(pwd), len(attempt)); i++) {
    if(pwd[i] != attempt[i]){
        return false
    }
}
return true
```

pwd	M	e	F	o	S	y	L	o	M	a
attempt	M	e	S	y	M	a	F	o		

🕐 Execution time (ET): ε

Context: Timing attacks over programs

```c
// input pwd    : Real password
// input attempt: Tentative password
for (i = 0; i < min(len(pwd), len(attempt)); i++) {
    if(pwd[i] != attempt[i]){
        return false
    }
}
return true
```

pwd	M	e	F	o	S	y	L	o	M	a
attempt	M	e	S	y	M	a	F	o		

🕓 Execution time (ET): ε ε

Context: Timing attacks over programs

```c
// input pwd    : Real password
// input attempt: Tentative password
for (i = 0; i < min(len(pwd), len(attempt)); i++) {
    if(pwd[i] != attempt[i]){
        return false
    }
}
return true
```

pwd	M	e	F	o	S	y	L	o	M	a
attempt	M	e	S	y	M	a	F	o		

🕓 Execution time (ET): ε ε ε

Context: Timing attacks over programs

```c
// input pwd    : Real password
// input attempt: Tentative password
for (i = 0; i < min(len(pwd), len(attempt)); i++) {
    if(pwd[i] != attempt[i]){
        return false
    }
}
return true
```

pwd	M	e	F	o	S	y	L	o	M	a
attempt	M	e	S	y	M	a	F	o		

🕓 Execution time (ET): $\varepsilon \quad \varepsilon \quad \varepsilon \qquad = 3\varepsilon \qquad \Rightarrow$ 2 correct characters

Problem: The ET is proportional to the **number of consecutive correct** characters from the beginning of attempt

Problems

Timing analysis of programs is **hard**: it depends not just on code, but also on **low-level details** of execution

Impact of hardware

- ET is heavily influenced by the **micro-architecture**
  - Especially: pipelines, caches, memory hierarchy

Limitations of existing techniques

- Most abstract time away or focus on **coarse properties**
  - e.g., schedulability analysis, worst-case execution time (WCET)
- **Insufficient** for fine-grained timing behaviors
  - e.g., detecting or mitigating *timed side-channels*

Our contributions in a nutshell

- A **modular** and **automated** approach to build formal models to analyze **timing behaviors**
  - *binary code with the hardware*

Our contributions in a nutshell

⚙️ A **modular** and **automated** approach to build formal models to analyze **timing behaviors**

- *binary code with the hardware*

💻 An **implementation**

- *targeting a realistic micro-architecture of a simple micro-controller*
- *producing **time Petri nets** models*

Our contributions in a nutshell

⚙️ A **modular** and **automated** approach to build formal models to analyze **timing behaviors**
- *binary code with the hardware*

💻 An **implementation**
- *targeting a realistic micro-architecture of a simple micro-controller*
- *producing **time Petri nets** models*

⚔️ An application to **timing attacks** in **C programs** using the **Roméo** model checker

Methodology

Overall methodology

Inputs

01100
10110
11110

Binary code

Overall methodology

Inputs

01100
10110
11110 Binary code

➔

Workflow

- Translate binary code → PTPN
- Add static hardware model (PTPN)
- ✓ Perform parametric timed model checking

Overall methodology

Inputs

01100
10110
11110 Binary code

Workflow

 Translate binary code → PTPN

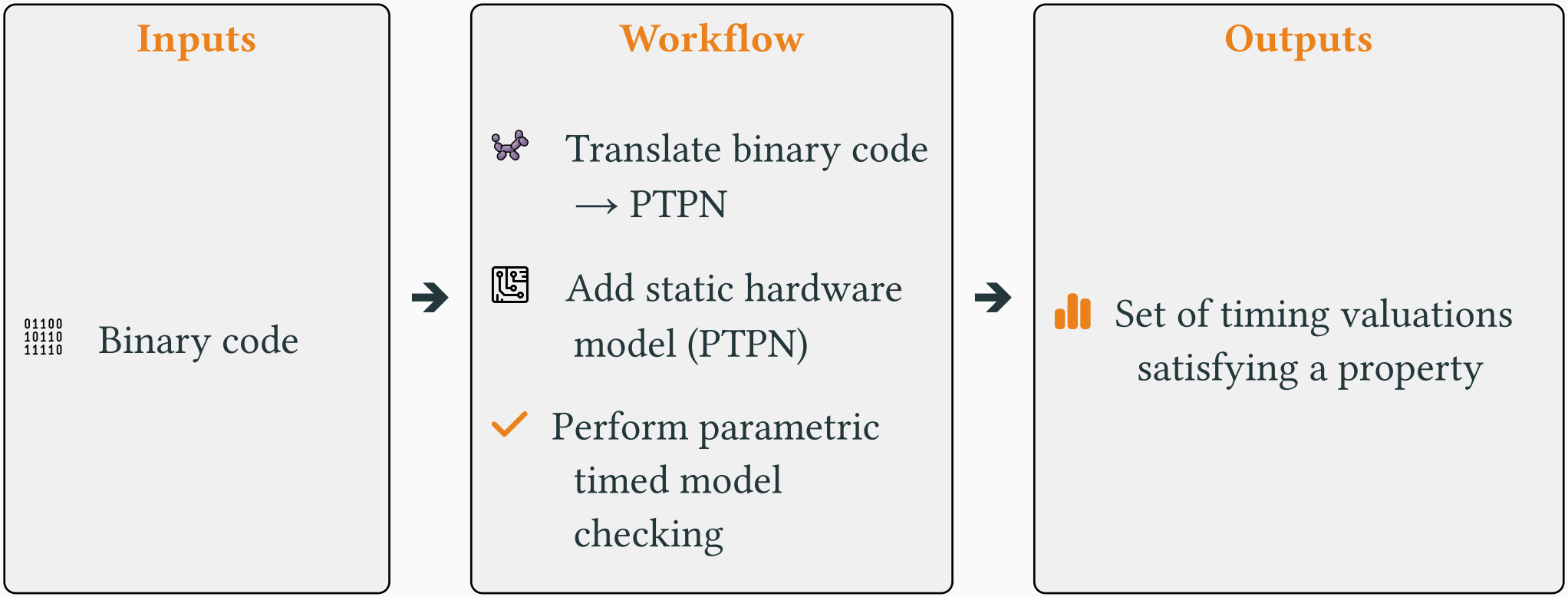
 Add static hardware model (PTPN)

✓ Perform parametric timed model checking

Outputs

 Set of timing valuations satisfying a property

Overall methodology



- *e.g., possible execution times*
- *application: password leak detection*

Parametric time Petri nets with variables

Extension of **Petri nets** with

-
-
-

[TLR09]

[TLR09] Louis-Marie Traonouez, Didier Lime, and Olivier H. Roux, "Parametric Model-Checking of Stopwatch Petri Nets," *Journal of Universal Computer Science*, 2009.

Parametric time Petri nets with variables

Extension of **Petri nets** with

- firing times
- timing parameters
-

[TLR09]

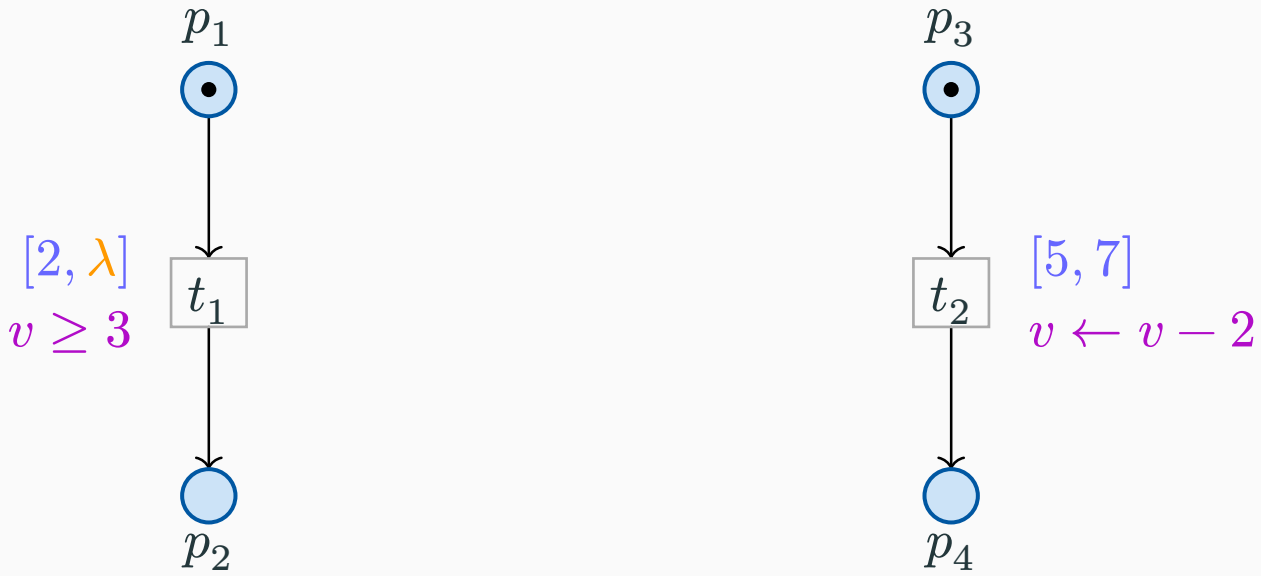
[TLR09] Louis-Marie Traonouez, Didier Lime, and Olivier H. Roux, "Parametric Model-Checking of Stopwatch Petri Nets," *Journal of Universal Computer Science*, 2009.

Parametric time Petri nets with variables

Extension of **Petri nets** with

- firing times
- timing parameters
- integer-valued variables (with guards and updates)

[TLR09]



[TLR09] Louis-Marie Traonouez, Didier Lime, and Olivier H. Roux, "Parametric Model-Checking of Stopwatch Petri Nets," *Journal of Universal Computer Science*, 2009.

Considered hardware

Our hardware

- Model of the **processor architecture**
  - *relatively simple micro-architecture similar to ARM Cortex M0+ core, with a 2-stage pipeline (Fetch and Execute)*
- Model of the **instruction set** architecture (ISA)
  - *ARMv6-M ISA*

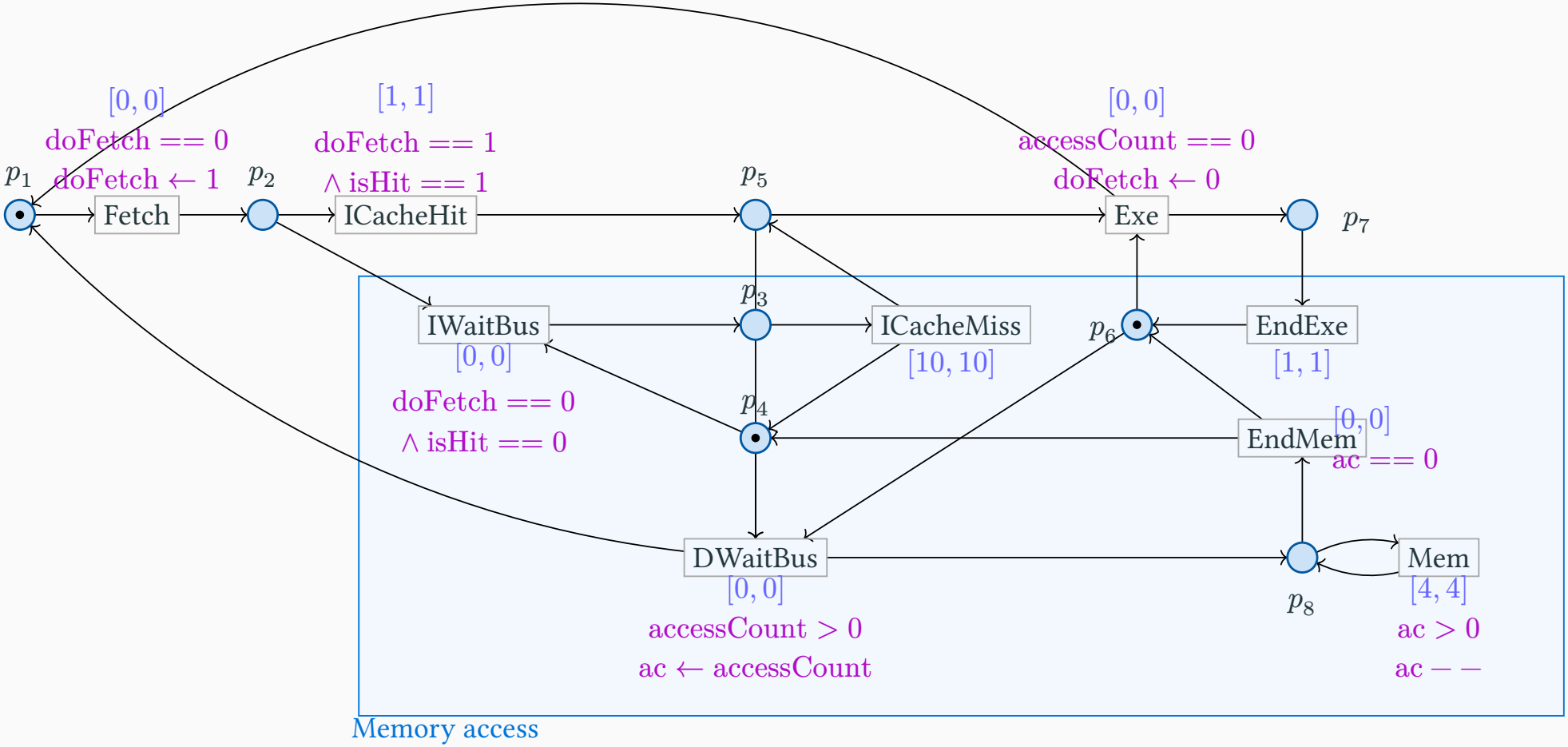
Features

- Execution pipeline of the processor
- Unique memory space
  - *(instructions and data)*
- Bus between the processor and memory
- Direct-mapped **instruction cache**
  - *with 16 lines of 32 bytes*
  - *no actual instructions, but only information about their presence*
- **No data** cache

- Among the limitations: no switch/case, function pointers. . .

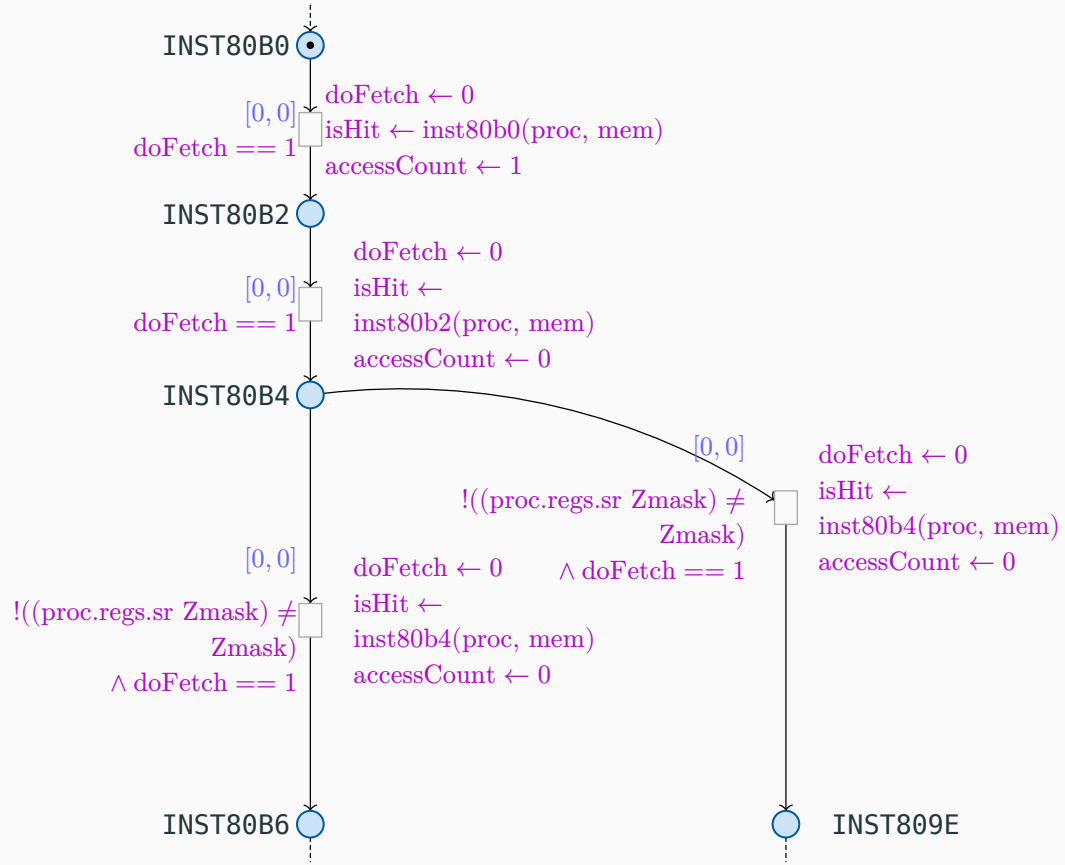
PTPN hardware model

- doFetch, isHit and accessCount: variables used to synchronize with the software



PTPN software model

- Captures the **binary code** of the program *(ARMv6-M)*
  - Firing a transition corresponds to executing the instruction
  - Pipeline fetch: doFetch
  - Memory access: accessCount and isHit
- Structurally identical to the **control flow graph**



Parametric timed model checking

System

Specification

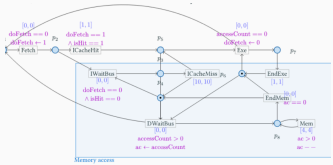
"The system must be safe"

Parametric timed model checking

System

→

Formal model



Specification

"The system must be safe"

→

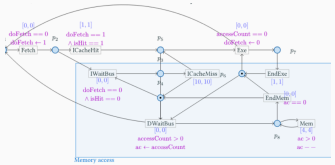
Property

$AG \neg$ ○

Parametric timed model checking

System

Formal model



Specification

"The system must be safe"

Property

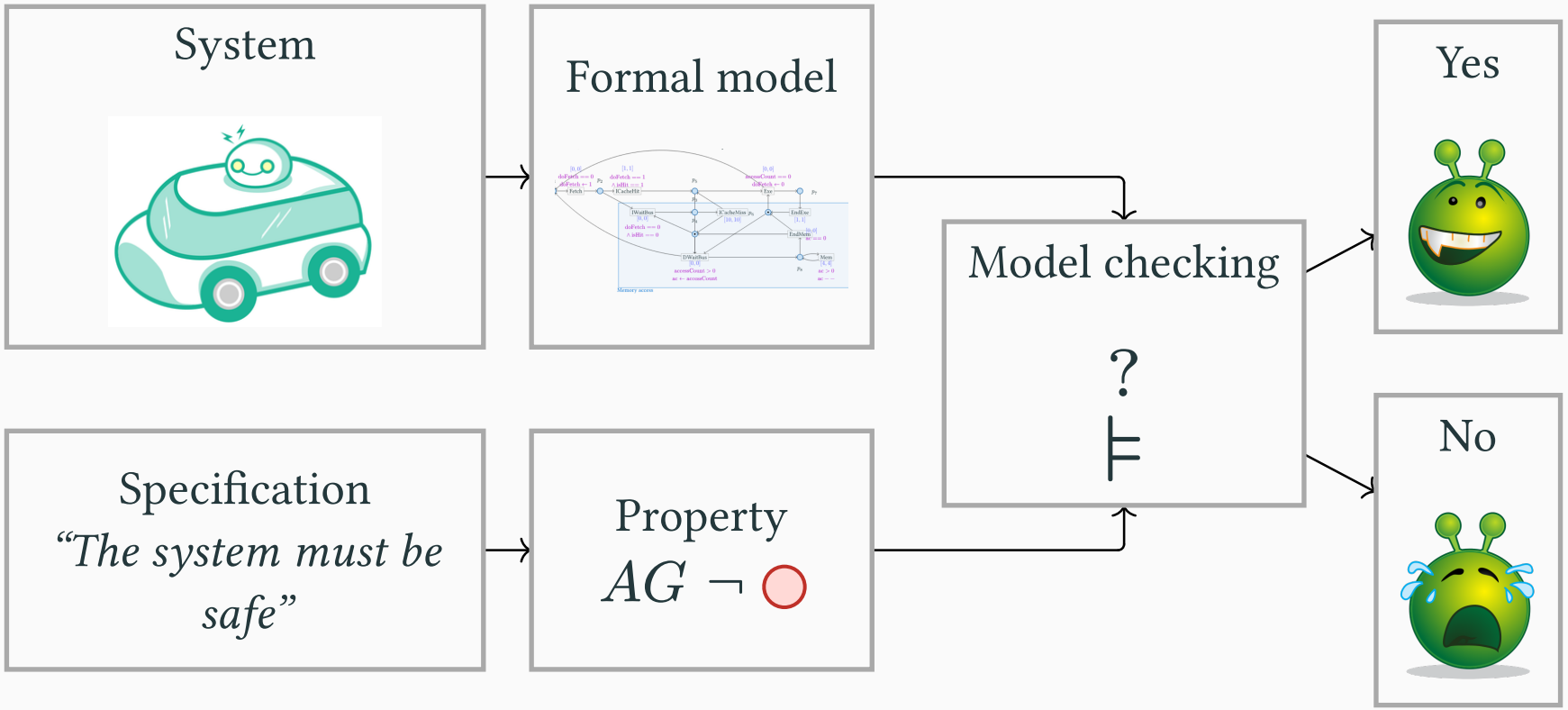
$AG \neg$ 

Model checking

?

$\models$

Parametric timed model checking

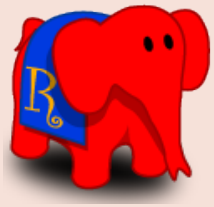


Question: does the model of the system **satisfy** the property?

A fully automated translation

- Including the hardware and software models 🧰
- Written in C++ and Python
- All the way from the C source code to the PTPN model
- Entirely open source *(github.com/DylanMarinho/codeToPN/)*

Target model checker: Roméo [Lim+09] ✨



- **Parametric timed model checker** supporting (extensions) of PTPNs
- Including **C-like code** to be executed during transitions

---

[Lim+09] Didier Lime, Olivier H. Roux, Charlotte Seidner, and Louis-Marie Traonouez, "Romeo: A Parametric Model-Checker for Petri Nets with Stopwatches," in *TACAS 2009*, 2009.

Application to security properties

Application to security properties

Timing attacks

- Attacker can **infer information** about the secret key by measuring the **execution time** of the program
  - *e.g., password checking program*

Execution-time opacity [And+23]

"Can the attacker deduce internal behavior by only observing the execution time?"

[And+23] Étienne André, Engel Lefaucheux, Didier Lime, Dylan Marinho, and Jun Sun, "Configuring Timing Parameters to Ensure Execution-Time Opacity in Timed Automata," in *TiCSA@ETAPS 2023*, 2023.

Application to security properties

Timing attacks

- Attacker can **infer information** about the secret key by measuring the **execution time** of the program
  - *e.g., password checking program*

Execution-time opacity [And+23]

"Can the attacker deduce internal behavior by only observing the execution time?"

- Use of timing parameters: to measure execution times

[And+23] Étienne André, Engel Lefaucheux, Didier Lime, Dylan Marinho, and Jun Sun, "Configuring Timing Parameters to Ensure Execution-Time Opacity in Timed Automata," in *TiCSA@ETAPS 2023*, 2023.

Which of the following two programs is not secure?

```c
int main() {
    int i;
    int length = 10; // length of the strings

    char ca[11] = "patehenaff";
    char cb[11] = "pasta";

    int result = 1; // true

    for (i = 0; i < length; i++){
        result &= (ca[i] == cb[i]);
    }
    return result;
}
```

```c
int main() {
    int i;
    int length = 10; // length of the strings

    char ca[11] = "patehenaff";
    char cb[11] = "pasta";

    for (i = 0; i < length; i++){
        if (ca[i] != cb[i]) {
            return 0; // false
        }
    }
    return 1; // true
}
```

Which of the following two programs is not secure?

```c
int main() {
    int i;
    int length = 10; // length of the strings

    char ca[11] = "patehenaff";
    char cb[11] = "pasta";

    int result = 1; // true

    for (i = 0; i < length; i++){
        result &= (ca[i] == cb[i]);
    }
    return result;
}
```

Secure

```c
int main() {
    int i;
    int length = 10; // length of the strings

    char ca[11] = "patehenaff";
    char cb[11] = "pasta";

    for (i = 0; i < length; i++){
        if (ca[i] != cb[i]) {
            return 0; // false
        }
    }
    return 1; // true
}
```

Unsecure

Which of the following two programs is not secure?

```c
int main() {
    int i;
    int length = 10; // length of the strings

    char ca[11] = "patehenaff";
    char cb[11] = "pasta";

    int result = 1; // true

    for (i = 0; i < length; i++){
        result &= (ca[i] == cb[i]);
    }
    return result;
}
```

Secure - Constant ET: 876

```c
int main() {
    int i;
    int length = 10; // length of the strings

    char ca[11] = "patehenaff";
    char cb[11] = "pasta";

    for (i = 0; i < length; i++){
        if (ca[i] != cb[i]) {
            return 0; // false
        }
    }
    return 1; // true
}
```

Unsecure - ET sensitive

- 758 for the secret password
- {362, 404, 446, 488, 530, 572, 614, 656, 698, 740} for any other password

Is this third program secure?

```c
int main () {
  int i ;
  int length = 10; // length of the strings
  char ca[11] = " patehenaff " ;
  char cb[11] = " pasta " ;

  int result = 1; // true
  for (i=0; i<length ; i++) {
    if (ca[i] == cb[i]) {
      result &= 1;
    } else {
      result &= 0;
    }
  }
  return result ;
}
```

Is this third program secure?

```c
int main () {
  int i ;
  int length = 10; // length of the strings
  char ca[11] = " patehenaff " ;
  char cb[11] = " pasta " ;

  int result = 1; // true
  for (i=0; i<length ; i++) {
    if (ca[i] == cb[i]) {
      result &= 1;
    } else {
      result &= 0;
    }
  }
  return result ;
}
```

- It seems so: very close to the former secure program
- But it is not due to the **instruction cache**
  - 876 for the secret password
  - {816, 822, 828, 834, 840, 846, 852, 858, 864, 870} for any other password

Is this third program secure?

```c
int main () {
  int i ;
  int length = 10; // length of the strings
  char ca[11] = " patehenaff " ;
  char cb[11] = " pasta " ;

  int result = 1; // true
  for (i=0; i<length ; i++) {
    if (ca[i] == cb[i]) {
      result &= 1;
    } else {
      result &= 0;
    }
  }
  return result ;
}
```

- It seems so: very close to the former secure program
- But it is not due to the **instruction cache**
  - 876 for the secret password
  - {816, 822, 828, 834, 840, 846, 852, 858, 864, 870} for any other password

We can **reconfigure** the program, by **making it opaque**

- *adding 6 nop instructions at the end of one branch*
- *(see paper)*

Conclusion and perspectives

Conclusion

End-to-end approach on **binary code timing analysis**, subject to micro-architectural constraints

- automated production of **timed formal models** of both the program and the hardware architecture
- using (parametric) time Petri nets

Illustrative case-study: **detection of timing leaks** in programs

- via parameter synthesis techniques using **Roméo**
- (manual) **reconfiguration** of the program to make it opaque

Perspectives

- Modeling and analysis of programs on **multicore** architectures
- Automatic modification of a program to make it **opaque**

- Handling **more complex attacks**
  - Fault-injection
  - Cache side-channels
    - *flush and reload, prime and probe*
  - Energy-based attacks

- Formal proof of our translation?

Bibliography

Bibliography

[TLR09] Louis-Marie Traonouez, Didier Lime, and Olivier H. Roux, “Parametric Model-Checking of Stopwatch Petri Nets,” *Journal of Universal Computer Science*, 2009.

[Lim+09] Didier Lime, Olivier H. Roux, Charlotte Seidner, and Louis-Marie Traonouez, “Romeo: A Parametric Model-Checker for Petri Nets with Stopwatches,” in *TACAS 2009*, 2009.

[And+23] Étienne André, Engel Lefaucheux, Didier Lime, Dylan Marinho, and Jun Sun, “Configuring Timing Parameters to Ensure Execution-Time Opacity in Timed Automata,” in *TiCSA@ETAPS 2023*, 2023.

Additional information

Explanation of the pictures

- Therac-25 bug
- Computer bug, race condition
- Consequences: multiple fatalities

- Allusion to the MIM-104 Patriot Missile Failure (Iraq, 1991)
- 28 fatalities, hundreds of injured
- Computer bug: software error (clock drift)
- (Picture of an actual MIM-104 Patriot Missile, though not the one of 1991)

- Allusion to the sinking of the Sleipner A offshore platform (Norway, 1991)
- No fatalities
- Computer bug: inaccurate finite element analysis modeling
- (Picture actually from the Deepwater Horizon Offshore Drilling Platform)

Explanation of the pictures

- Ariane flight V88 (France, 1996)
- Computer bug (notably integer overflow)
- Consequences: US$370 million

Licensing

License of this document

Authors: Étienne André, **Dylan Marinho**