SAC-SVT 2025

Catania, Sicilia

1st of April 2025

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

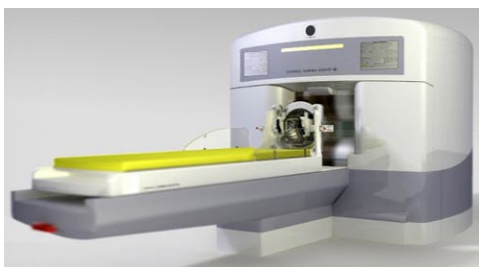
Étienne André, **Jean-Luc Béchennec**, Sudipta Chattopadhyay, Sébastien Faucou, Didier Lime, **Dylan Marinho**, Olivier H. Roux and Jun Sun

Université Sorbonne Paris Nord, LIPN, CNRS UMR 7030, F-93430 Villetaneuse, France

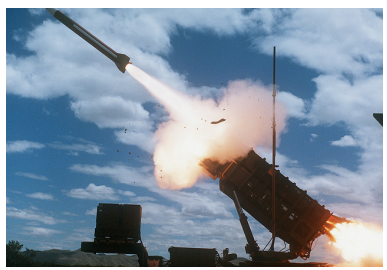
Context: Verifying complex timed systems

- 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 Patriot Missile Failure
(Iraq, 1991)

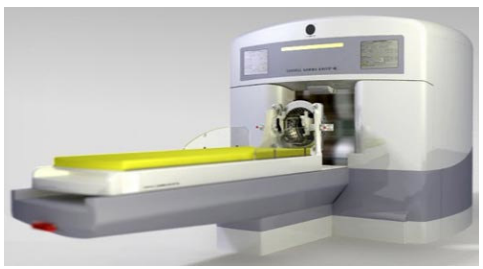
Sleipner A offshore platform
(Norway, 1991)

Ariane flight V88
(France, 1996)

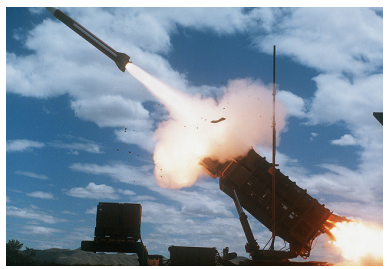
Context: Verifying complex timed systems

- 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 Patriot Missile Failure
(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

```
1 # input pwd    : Real password
2 # input attempt: Tentative password
3 for i = 0 to min(len(pwd), len(attempt)) - 1 do
4     if pwd[i] =/= attempt[i] then
5         return false
6 done
7 return true
```

Context: Timing attacks over programs

```
1 # input pwd    : Real password
2 # input attempt: Tentative password
3 for i = 0 to min(len(pwd), len(attempt)) - 1 do
4     if pwd[i] =/= attempt[i] then
5         return false
6 done
7 return true
```

pwd	p	a	t	e	H	e	n	a	f	f
attempt	p	a	s	t	a					

Execution time:

Context: Timing attacks over programs

```
1 # input pwd    : Real password
2 # input attempt: Tentative password
3 for i = 0 to min(len(pwd), len(attempt)) - 1 do
4     if pwd[i] =/= attempt[i] then
5         return false
6 done
7 return true
```

pwd	p	a	t	e	H	e	n	a	f	f
attempt	p	a	s	t	a					

Execution time: ϵ

Context: Timing attacks over programs

```
1 # input pwd    : Real password
2 # input attempt: Tentative password
3 for i = 0 to min(len(pwd), len(attempt)) - 1 do
4     if pwd[i] =/= attempt[i] then
5         return false
6 done
7 return true
```

pwd	p	a	t	e	H	e	n	a	f	f
attempt	p	a	s	t	a					

Execution time: $\epsilon + \epsilon$

Context: Timing attacks over programs

```
1 # input pwd    : Real password
2 # input attempt: Tentative password
3 for i = 0 to min(len(pwd), len(attempt)) - 1 do
4     if pwd[i] =/= attempt[i] then
5         return false
6 done
7 return true
```

pwd	p	a	t	e	H	e	n	a	f	f
attempt	p	a	s	t	a					

Execution time: $\epsilon + \epsilon + \epsilon$

Context: Timing attacks over programs

```
1 # input pwd    : Real password
2 # input attempt: Tentative password
3 for i = 0 to min(len(pwd), len(attempt)) - 1 do
4     if pwd[i] =/= attempt[i] then
5         return false
6 done
7 return true
```

pwd	p	a	t	e	H	e	n	a	f	f
attempt	p	a	s	t	a					

Execution time: $\epsilon + \epsilon + \epsilon$

- Problem: The execution time is proportional to the number of consecutive correct characters from the beginning of `attempt`

Problems

Timing analysis of programs is hard

- timed behavior strongly impacted by the hardware
  - most importantly by the micro-architecture (pipeline, memory hierarchy)

- techniques abstracting time away or focusing on gross grain timing properties (schedulability, WCET) are unsuited for finer grain properties
  - such as timed side channel detection or mitigation

Our contributions in a nutshell

1. A modular and automated approach to build formal models of binary code with the hardware to analyze timing behaviors

Our contributions in a nutshell

1. A modular and automated approach to build formal models of binary code with the hardware to analyze timing behaviors

2. An implementation targeting a realistic micro-architecture of a simple micro-controller capable of running a ARMv6-M instruction set and producing time Petri nets models

Our contributions in a nutshell

1. A modular and automated approach to build formal models of binary code with the hardware to analyze timing behaviors
2. An implementation targeting a realistic micro-architecture of a simple micro-controller capable of running a ARMv6-M instruction set and producing time Petri nets models
3. An application to timing attacks in C programs using the Roméo model checker

Outline

Outline

Overall methodology

Inputs

- binary code

Workflow

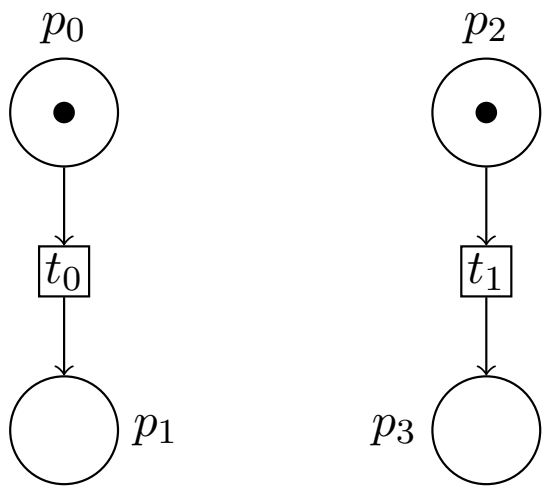
1. translate the binary code into a parametric time Petri net (PTPN)
2. add a (static) model of the hardware, modeled as a PTPN
3. perform parametric timed model checking on these models

Outputs

- set of timing valuations for which a property is satisfied
  - application: computation of possible execution times for a C program, and application to password leak detection

Parametric time Petri nets with variables

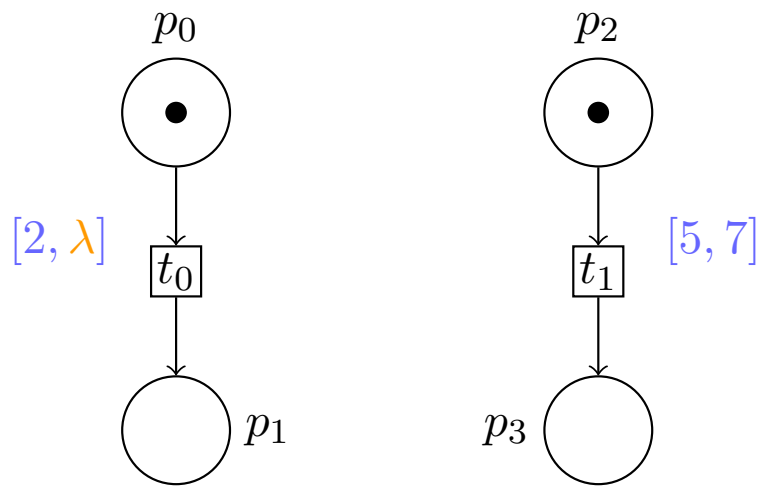
Extension of Petri nets



[TLR09] Louis-Marie Traonouez, Didier Lime, and Olivier H. Roux. "Parametric Model-Checking of Stopwatch Petri Nets". In: *Journal of Universal Computer Science* 15.17 (2009), pp. 3273–3304

Parametric time Petri nets with variables

Extension of Petri nets with firing times and timing parameters [TLR09]

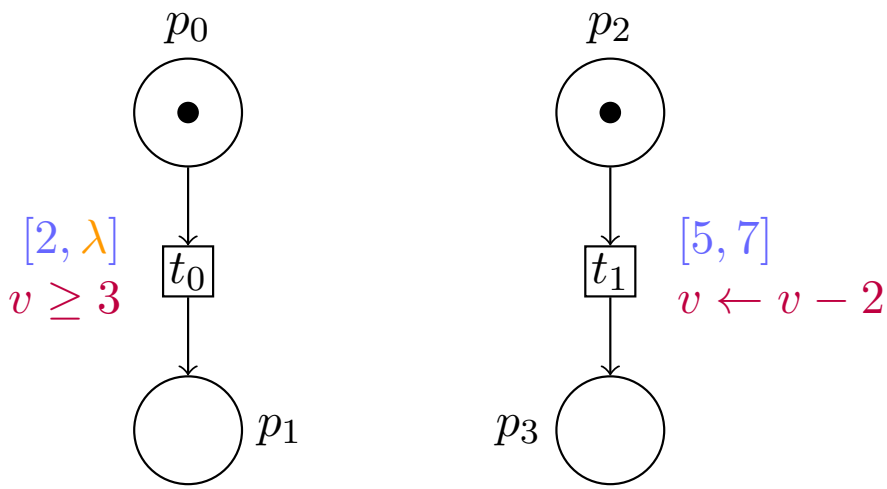


[TLR09] Louis-Marie Traonouez, Didier Lime, and Olivier H. Roux. "Parametric Model-Checking of Stopwatch Petri Nets". In: *Journal of Universal Computer Science* 15.17 (2009), pp. 3273–3304

Parametric time Petri nets with variables

Extension of Petri nets with firing times and timing parameters [TLR09]

- and integer-valued variables (with guards and updates)



- PTPNs come with a formal semantics

[TLR09] Louis-Marie Traonouez, Didier Lime, and Olivier H. Roux. "Parametric Model-Checking of Stopwatch Petri Nets". In: *Journal of Universal Computer Science* 15.17 (2009), pp. 3273–3304

Outline

Considered hardware

Our hardware consists of

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

Features:

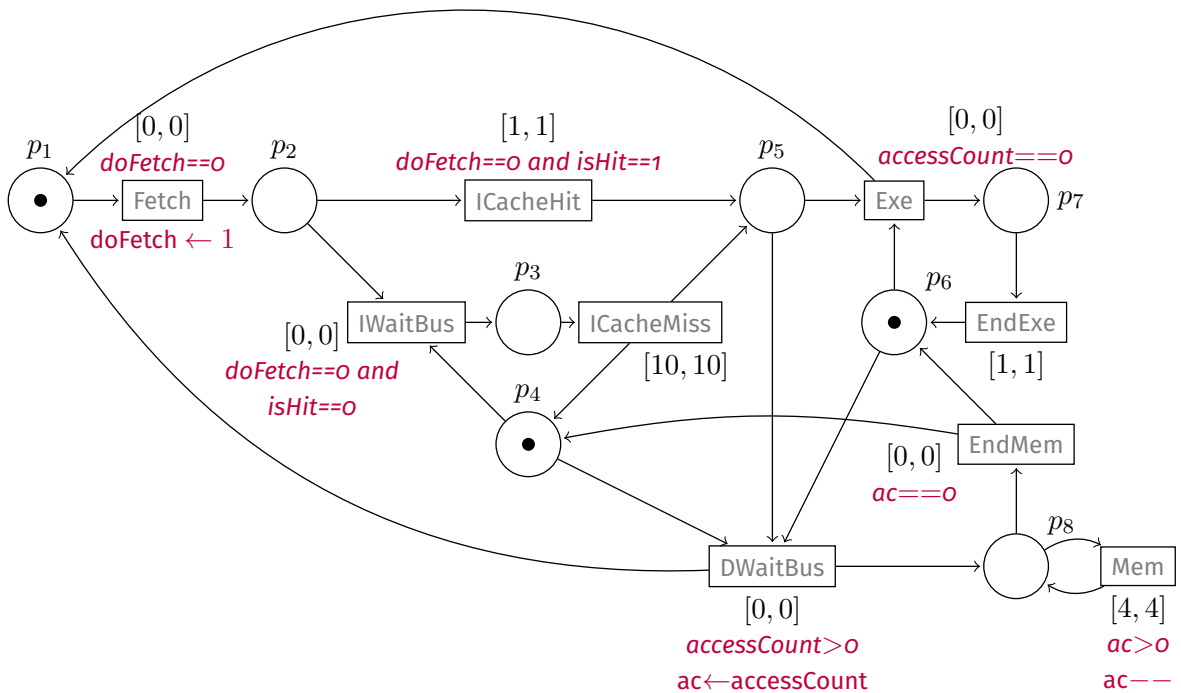
- a unique memory space for instructions and data
- a bus between the processor and memory
- a direct-mapped instruction cache with 16 lines of 32 bytes
  - the cache does not contain actual instructions, but only information about their presence
- no data cache
- the execution pipeline of the processor

Among the limitations

- no switch/case, function pointers…

PTPN hardware model

- doFetch, isHit and accessCount: variables used to synchronize the pipeline model with the program model
- token in p_1: the pipeline can fetch a new instruction
- from p_2: instruction fetch can lead to an instruction cache hit (ICacheHit transition that takes 1 cycle) or a miss according to isHit variable
- and so on

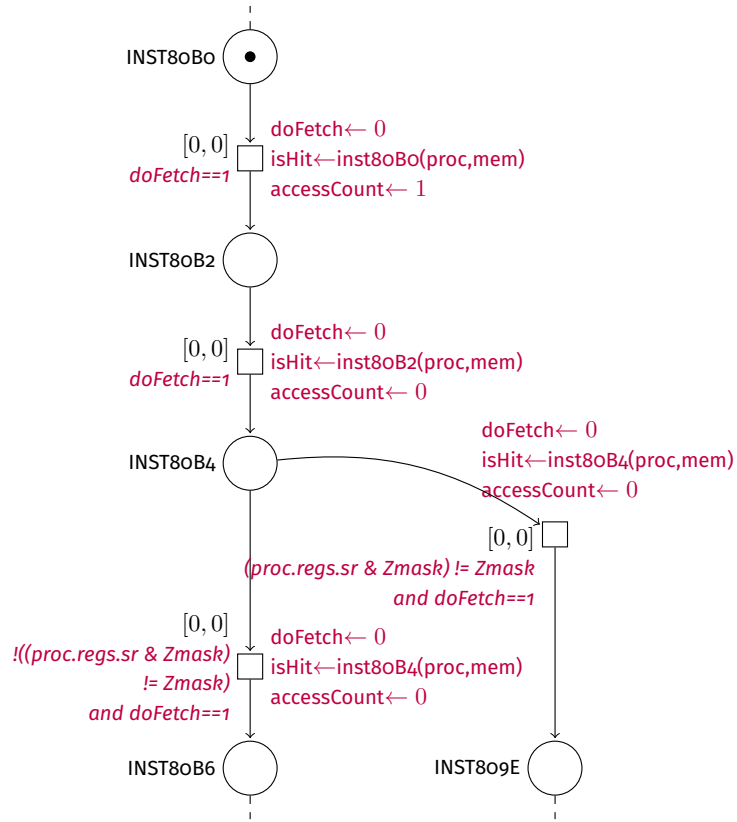


Outline

PTPN software model

Software part

- captures the ARMv6-M binary-level representation of the program as a (P)TPN which synchronizes with the pipeline model
- structurally identical to the control flow graph of the program
- Firing a transition corresponds to executing an instruction
  - A transition includes a guard which synchronizes with the pipeline Fetch stage via the doFetch variable, and an update which includes functional execution of the instruction and assignment of the accessCount and isHit variables

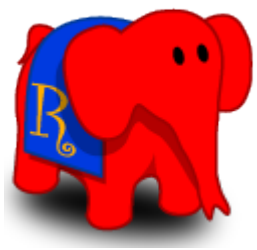


A fully automated translation

- 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: `https://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*. vol. 5505. Lecture Notes in Computer Science. Springer, Mar. 2009, pp. 54–57

Outline

Model checking in a nutshell

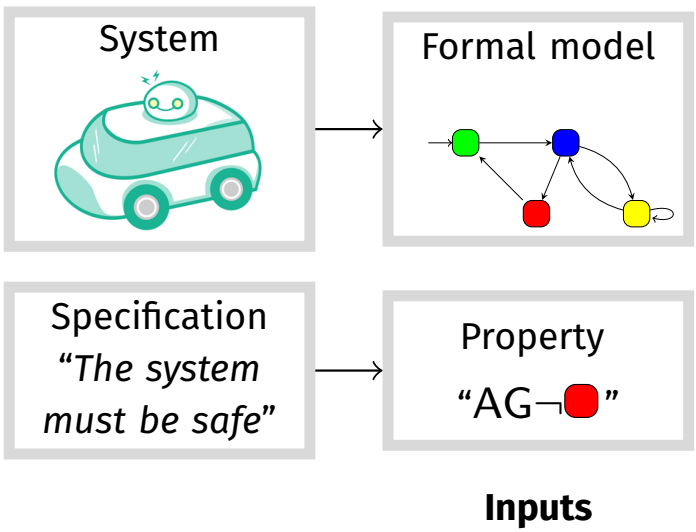
- Principle of model checking

System

Specification
"The system must be safe"

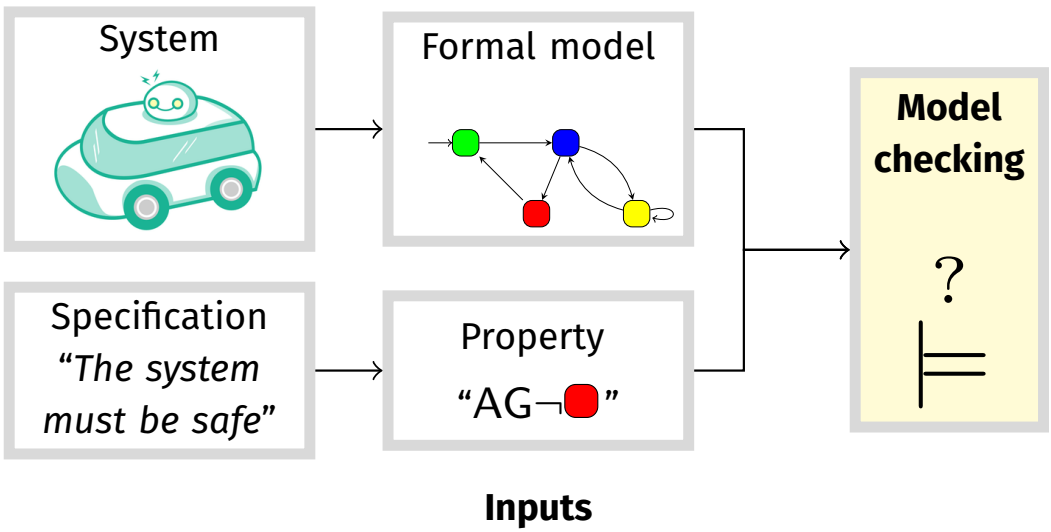
Model checking in a nutshell

- Principle of model checking



Model checking in a nutshell

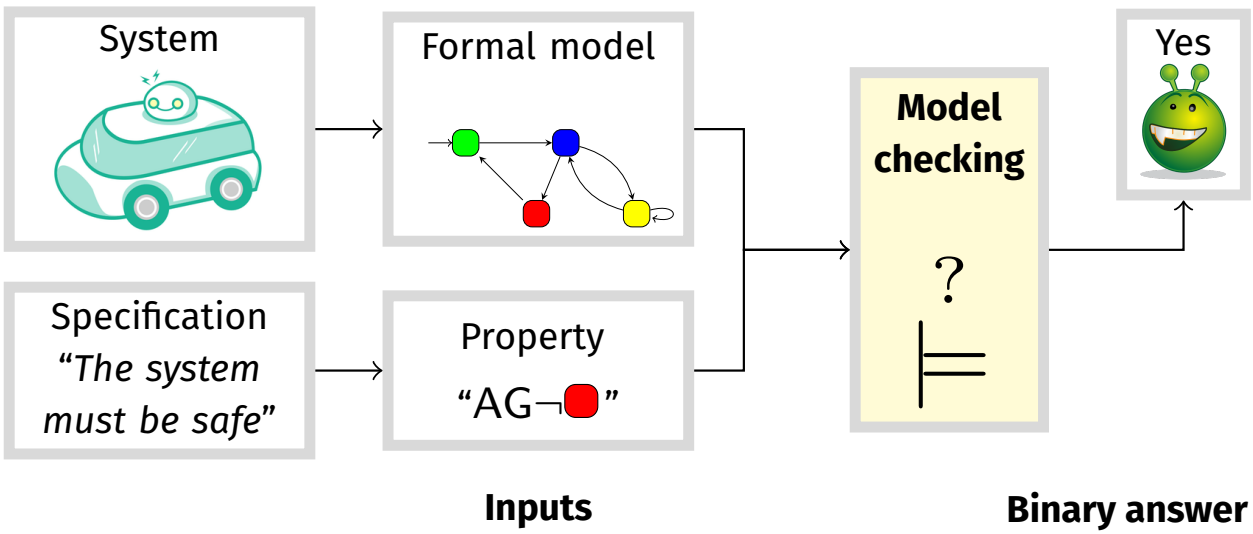
- Principle of model checking



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

Model checking in a nutshell

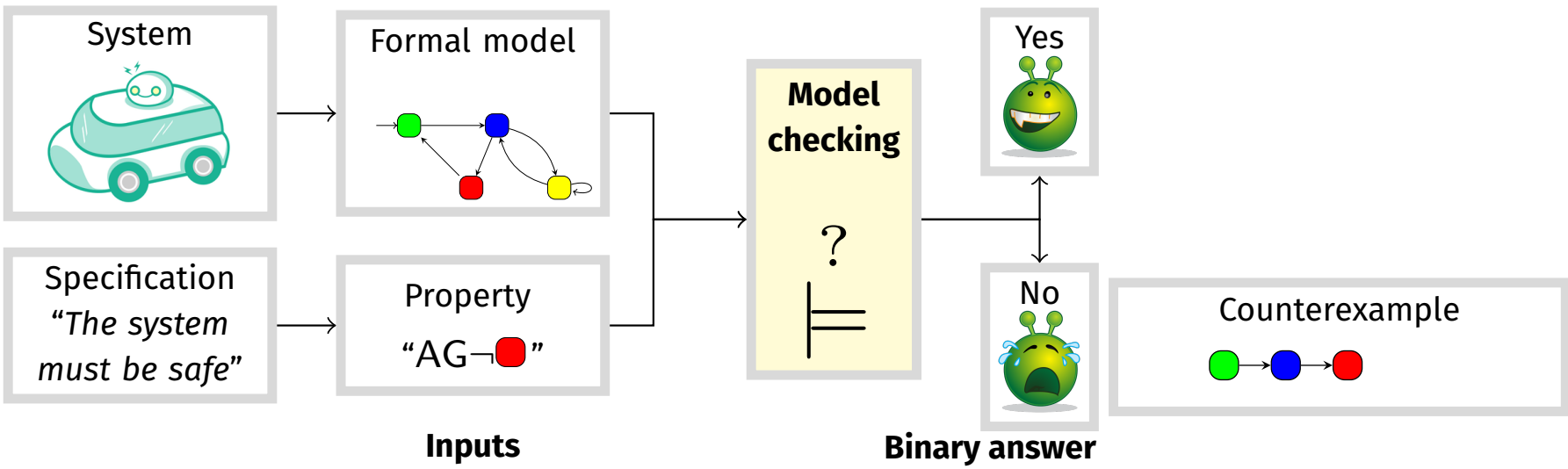
- Principle of model checking



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

Model checking in a nutshell

- Principle of model checking



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

Outline

Application to timing leaks

Application to execution-time opacity [And+23]

Definition (Execution-time opacity)

"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*. vol. 392. Electronic Proceedings in Theoretical Computer Science. Invited paper. 2023, pp. 1–26

Application to timing leaks

Application to execution-time opacity [And+23]

Definition (Execution-time opacity)

"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*. vol. 392. Electronic Proceedings in Theoretical Computer Science. Invited paper. 2023, pp. 1–26

A secure and an insecure program

Which of the following two programs is insecure?

```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
}
```

A secure and an insecure program

Which of the following two programs is insecure?

```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
}
```

Insecure

A secure and an insecure program

Which of the following two programs is insecure?

```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 execution time

```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
}
```

Insecure

Execution time sensitive to the password length

A secure and an insecure program

Which of the following two programs is insecure?

```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 execution time

```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
}
```

Insecure

Execution time sensitive to the password length

Applying our method:

- secure program: the execution time is always 876
- insecure program: the execution time can be
  - 758 for the secret password
  - $\{362, 404, 446, 488, 530, 572, 614, 656, 698, 740\}$ for any other password
  - $\Rightarrow$ highly insecure!

A third program

It 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;
}
```

A third program

It 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, because very close to the former secure program…

A third program

It 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, because 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

A third program

It 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, because 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

In addition: we can reconfigure the program, by making it opaque

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

Outline

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 C 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

References I

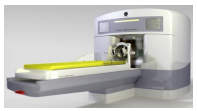
[And+23] Étienne André, Engel Lefaucheux, Didier Lime, Dylan Marinho, and Jun Sun. “Configuring Timing Parameters to Ensure Execution-Time Opacity in Timed Automata”. In: *TiCSA* (Apr. 23, 2023). Ed. by Maurice H. ter Beek and Clemens Dubslaff. Vol. 392. Electronic Proceedings in Theoretical Computer Science. Invited paper. Paris, France, 2023, pp. 1–26. DOI: 10.4204/EPTCS.392.1.

[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* (Mar. 22–29, 2009). Ed. by Stefan Kowalewski and Anna Philippou. Vol. 5505. Lecture Notes in Computer Science. York, United Kingdom: Springer, Mar. 2009, pp. 54–57. DOI: 10.1007/978-3-642-00768-2_6.

[TLR09] Louis-Marie Traonouez, Didier Lime, and Olivier H. Roux. “Parametric Model-Checking of Stopwatch Petri Nets”. In: *Journal of Universal Computer Science* 15.17 (2009), pp. 3273–3304. DOI: 10.3217/jucs-015-17-3273.

Additional information

Explanation for the 4 pictures in the beginning



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)

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

Instruction model

```
int inst80b0(core_t &core, mem_t &mem) { // 80b0: ldr.w r3, [r7, #4]
  core.regs.r[3] = memRead(mem, core.regs.r[7] + 4);
  return cacheAccess(core.ICache, 32944);
}

int inst80b2(core_t &core, mem_t &mem) { // 80b2: cmp r3, #0
  uint64_t op1 = core.regs.r[3];
  uint64_t op2 = 0;
  uint64_t val = op1 - op2;
  updateSR(core.regs, val, op1, -op2);
  return cacheAccess(core.ICache, 32946);
}

int inst80b4(core_t &core, mem_t &mem) { // 80b4: bne.n 809e
  return cacheAccess(core.ICache, 32948);
}
```

- For each instruction of a program, a function is generated in the C-like language supported by Roméo [Lim+09]
- In the first part, the content of the registers and of the memory is updated according to the semantics of the instruction
- In the second part, the state of the instruction cache is updated
- The return value indicates whether the instruction was present in the cache or not

[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*. vol. 5505. Lecture Notes in Computer Science. Springer, Mar. 2009, pp. 54–57

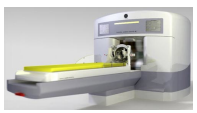
Licensing

Source of the graphics used I

Title : Explosion of first Ariane 5 flight, June 4, 1996
Author : ESA
Source : https://www.esa.int/ESA_Multimedia/Images/2009/09/Explosion_of_first_Ariane_5_flight
License : ESA Standard Licence

Title : Deepwater Horizon Offshore Drilling Platform on Fire
Author : ideum
Source : https://secure.flickr.com/photos/ideum/4711481781/
License : Creative Commons cc-by-sa

Title : DA-SC-88-01663
Author : imcomkorea
Source : https://secure.flickr.com/photos/imcomkorea/3017886760
License : Creative Commons cc-by-nc-nd



Title : Therac-25
Author : ?
Source : https://arquivonuclear.blogspot.com/2011/03/therac-25.html
License : unknown

Title : Autonomous robot vehicle or ADV typically used for food or grocery delivery
Author : Rlistmedia

Source of the graphics used II

Source : https://commons.wikimedia.org/wiki/File:Autonomous_delivery_robot_vehicles_ADV.png
License : Creative Commons cc-by

Title : Smiley green alien big eyes (aaah)
Author : LadyofHats
Source : https://commons.wikimedia.org/wiki/File:Smiley_green_alien_big_eyes.svg
License : Public domain

Title : Smiley green alien big eyes (cry)
Author : LadyofHats
Source : https://commons.wikimedia.org/wiki/File:Smiley_green_alien_big_eyes.svg
License : Public domain

Title : angler fish
Author : clipart-library.com (?)
Source : https://clipart-library.com/clipart/angler-fish-clipart_25.html
License : Personal use

Title : electric eel (?)
Author : Jihane37
Source : https://xsj.699pic.com/tupian/1ywir7.html
License : ?

License of this document

Authors: **Étienne André**