

# Automated Analysis of Weak Memory Models

## Task statement

Artem Yushkovskiy<sup>1,2</sup>

MSc Candidate

Supervisors: Assoc. Prof. Keijo Heljanko<sup>1</sup>  
Docent Igor I. Komarov<sup>2</sup>

<sup>1</sup>Aalto University (Espoo, Finland)  
Department of Computer Science,  
School of Science

<sup>2</sup>ITMO University (Saint Petersburg, Russia)  
Faculty of Information Security  
and Computer Technologies

Espoo, Saint Petersburg  
2018

- ▶ Improve the proof-of-concept memory model-aware analysis tool Porthos [2] by extending support for the input language.

## Task specification

- ▶ Study the general framework for memory model-aware analysis of concurrent programs [1];
- ▶ Review existing tools for memory model-aware analysis;
- ▶ Develop a *C compiler infrastructure* as a part of an abstract interpretation engine for the new tool PorthosC;
- ▶ Improve the SMT-encoding scheme for an *arbitrary control-flow*;
- ▶ Enhance performance, extensibility, reliability and maintainability of the tool.

## Verification of concurrent software

Motivating example: Write-write reordering (compiler relaxations)

{ x=0; y=0; }	
P	Q
$p_0: x \leftarrow 1$	$q_0: y \leftarrow 1$
$p_1: r_p \leftarrow y$	$q_1: r_q \leftarrow x$
exists $(r_p = 0 \wedge r_q = 0)$	

Total Store Order (e.g., x86)			
$p_0, p_1, q_0, q_1 (0; 1)$	$p_1, p_0, q_0, q_1 (0; 1)$	$p_0, p_1, q_1, q_0 (0; 1)$	$p_1, p_0, q_1, q_0 (0; 1)$
$q_0, q_1, p_0, p_1 (1; 0)$	$q_0, q_1, p_1, p_0 (1; 0)$	$q_1, q_0, p_0, p_1 (1; 0)$	$q_1, q_0, p_1, p_0 (1; 0)$
$p_0, q_0, p_1, q_1 (1; 1)$	$p_1, q_0, p_0, q_1 (0; 1)$	$p_0, q_1, p_1, q_0 (0; 1)$	$p_1, q_1, p_0, q_0 (0; 0)$
$p_0, q_0, q_1, p_1 (1; 1)$	$p_1, q_0, q_1, p_0 (0; 1)$	$p_0, q_1, q_0, p_1 (1; 1)$	$p_1, q_1, q_0, p_0 (0; 0)$
$q_0, p_0, p_1, q_1 (1; 1)$	$q_0, p_1, p_0, q_1 (1; 1)$	$q_1, p_0, p_1, q_0 (0; 0)$	$q_1, p_1, p_0, q_0 (0; 0)$
$q_0, p_0, q_1, p_1 (1; 1)$	$q_0, p_1, q_1, p_0 (1; 0)$	$q_1, p_0, q_1, p_0 (0; 0)$	$q_1, p_1, q_0, p_0 (0; 0)$

## Verification of concurrent software

Motivating example: Store buffering (hardware relaxations)

{ x=0; y=0; }	
P	Q
$p_0: x \leftarrow 1$	$q_0: y \leftarrow 1$
$p_1: r_p \leftarrow y$	$q_1: r_q \leftarrow x$
exists $(r_p = 0 \wedge r_q = 0)$	

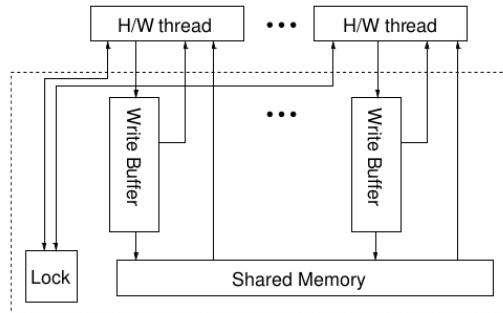


Figure: An x86-TSO abstract machine [4]

## Weak memory model-aware analysis

Event-based program representation

- **EventСобытие**  $\in \mathbb{E}$ , a low-level primitive operation:

- $memory\ event \in \mathbb{M} = \mathbb{R} \cup \mathbb{W}$ : access to a local/shared memory,
- $computational\ event \in \mathbb{C}$ : computation over local memory, and
- $barrier\ event \in \mathbb{B}$ : synchronisation fences;

- **Relation**  $\subseteq \mathbb{E} \times \mathbb{E}$ :

- *basic relations*:

- *program-order relation*  $po \subseteq \mathbb{E} \times \mathbb{E}$ : (control-flow),
- *read-from relation*  $rf \subseteq \mathbb{W} \times \mathbb{R}$ : (data-flow, and)
- *coherence-order relation*  $co \subseteq \mathbb{W} \times \mathbb{W}$ : (data-flow);

- *derived relations*:

- *union*  $r_1 \sqcup r_2$ ,
- *sequence*  $r_1 ; r_2$ ,
- *transitive closure*  $r^+$ ,
- $\dots$ ;

- **Assertion** over relations or sets of events :

- *acyclicity, irreflexivity or emptiness*.

## Weak memory model-aware analysis

Testing candidate executions

{ x=0; y=0; }	
P	Q
$p_0: x \leftarrow 1$	$q_0: y \leftarrow 1$
$p_1: r_p \leftarrow y$	$q_1: r_q \leftarrow x$
exists $(r_p = 0 \wedge r_q = 0)$	

SC model:  
...  
 $fr = (rf^{\sim -1}; co)$   
 $acyclic(fr \cup po)$

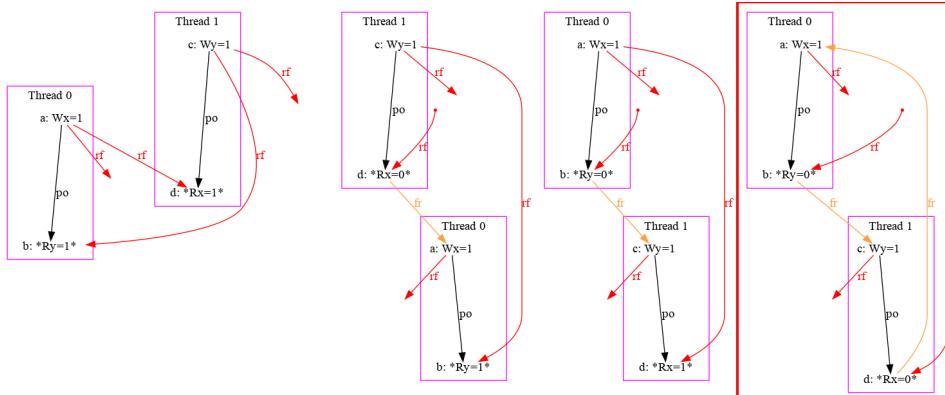


Figure: The four candidate executions allowed under x86-TSO

## Portability analysis

The Porthos tool

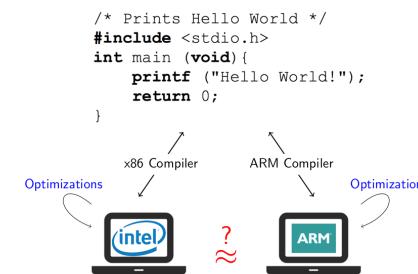


Figure: The illustration of the portability problem [3]

## Definition (Portability [2])

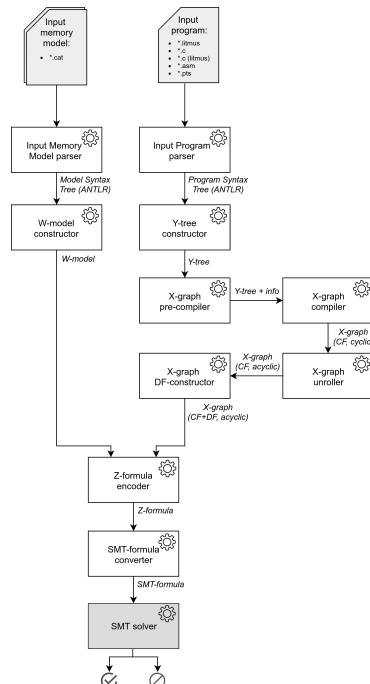
The program  $P$  is portable from the source memory model  $\mathcal{M}_S$  to the target model  $\mathcal{M}_T$  if  $cons_{\mathcal{M}_T}(P) \subseteq cons_{\mathcal{M}_S}(P)$

- Portability as an SMT-based bounded reachability problem:  
 $\phi = \phi_{CF} \wedge \phi_{DF} \wedge \phi_{\mathcal{M}_T} \wedge \phi_{\neg \mathcal{M}_S}$
- $SAT(\phi) \implies$  portability bug

PorthosC: Architecture

## Main components

- ▶ The enhanced version of Porthos, that is able to analyse C programs, has got a new name *PorthosC*.

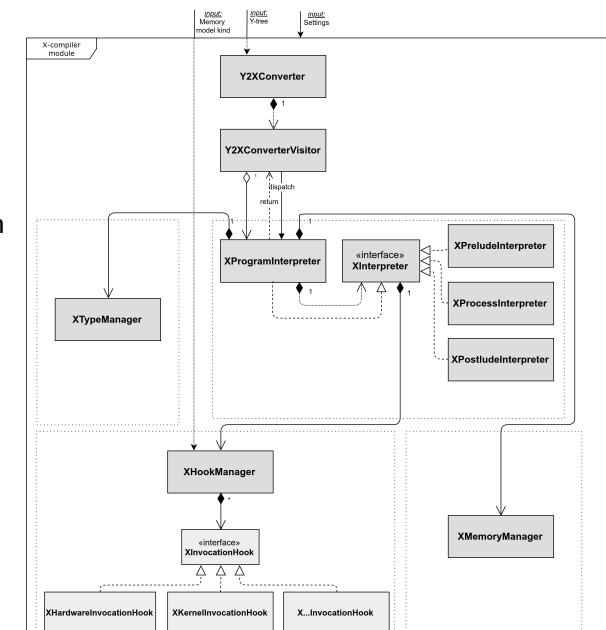


10

PorthosC: Architecture

## Abstract interpretation module

- ▶ The abstract interpretation module is an essential part of the compiler of C code into the event-flow graph representation (X-graph).



A set of small, light-blue navigation icons typically found in Beamer presentations, including symbols for back, forward, search, and table of contents.

Porthos v1: The input language

## Static syntactic determination of the variables kind

```

{ x, y }

thread t0 {
    r0 <= 1;
    r1 <= (2 + r0) * 3;
    y := r2;
    while (r0 > 4) {
        r0 <= x;
        r1 <= (r0 + 5);
        x.store(_rx, r1);
        y = x.load(_rx)
    }
}
exists x = 1, y = 2, 0:r1 = 3,

```

...

```

⟨instr⟩
: ⟨atom⟩
| ‘?’ ⟨instr⟩ ‘?’
| ⟨instr⟩ ‘;’ ⟨instr⟩
| ‘while’ ‘(’ ⟨bool-expr⟩ ‘)’ ⟨instr⟩
| ‘if’ ‘(’ ⟨bool-expr⟩ ‘{’ ⟨instr⟩ ‘}’ ⟨instr⟩
;

⟨atom⟩
: ⟨reg⟩ ‘<=’ ⟨expression⟩
| ⟨reg⟩ ‘<=?’ ⟨loc⟩
| ⟨loc⟩ ‘:=’ ⟨reg⟩
| ⟨reg⟩ ‘:=’ ⟨loc⟩ ‘.’ ‘store’ ‘(’ ⟨atomic⟩ ‘,’ ⟨reg⟩ ‘)’
| ⟨reg⟩ ‘:=’ ⟨loc⟩ ‘.’ ‘load’ ‘(’ ⟨atomic⟩ ‘)’
| ‘(’ ‘mifence’ ‘|’ ‘sync’ ‘|’ ‘lwsync’ ‘|’ ‘isync’ ‘)’
;
```

**Figure:** A program example in the Porthos v1 input language

```

<instr>
: <atom>
| '{' <instr> '}',
| <instr> ';' <instr>
| 'while' '(' <bool-expr> ')', <instr>
| 'if' <bool-expr> '{' <instr> '}', <instr>
;

<atom>
: <reg> '<' <expression>
| <reg> '<=' <loc>
| <loc> '<=' <reg>
| <reg> '>=' <loc> ',' 'store' '(' <atomic> ',' <reg> ')'
| <reg> '>=' <loc> ',' 'load' '(' <atomic> ')'
| ('mfence' | 'sync' | 'lwsync' | 'isync')
;
```

**Figure:** A sketch of the Porthos v1 input language grammar

Porthos v1: The input language

## Lack of support for functions invocations

```

{ x, y }

thread t0 {
    r0 <- 1;
    r1 <- (2 + r0) * 3;
    y := r2;
    while (r0 > 4) {
        r0 <= -x;
        r1 <- (r0 + 5);
        x.store(_rx, r1);
        y = x.load(_rx)
    };
}

```

**Figure:** A program example in the Porthos v1 input language

```

<instr>
: <atom>
| '{' <instr> '}'
| <instr> ';' <instr>
| 'while' '(' <bool-expr> ')', <instr>
| 'if' <bool-expr> '{', <instr> '}', <instr>
;

<atom>
: <reg> '<-' <expression>
| <reg> '<:->' <loc>
| <loc> ':=' <reg>
| <reg> ':=' <loc> ',' <store> '(' <atomic> ',', <reg> ')',
| <reg> ':=' <loc> ',' <load> '(' <atomic> ')',
| ('mifence' | 'sync' | 'lwsync' | 'isync')
;
```

**Figure:** A sketch of the Porthos v1 input language grammar

## PorthosC: The input language

Pre-compilation phase for the syntactic determination of the variables kind

```
{ int *x = 1; }

extern int z;

void thread_0(int &x, int &y) {
    LO: x = 0;
    int r;
    while (x * (5 + 4 / 2) % 3 == 1) {
        if (x != 0)
            goto LO;
        if (y > 6)
            continue;
        else if (++y > 7) {
            r = r + 10;
            break;
        }
        else
            goto L1;
        r = 11;
    }
    y = x.load(_rx) + 1;
    L1: x.store(_rx, r);
}

exists (y == x + 1 && thread_0:r > 21)
```

Figure: A program example in C

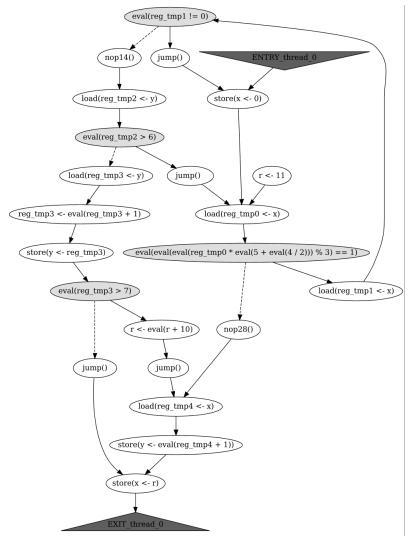


Figure: The compiled event-flow graph

## PorthosC: The input language

Support for functions invocations: The invocation hooking mechanism

```
{ int *x = 1; }

extern int z;

void thread_0(int &x, int &y) {
    LO: x = 0;
    int r;
    while (x * (5 + 4 / 2) % 3 == 1) {
        if (x != 0)
            goto LO;
        if (y > 6)
            continue;
        else if (++y > 7) {
            r = r + 10;
            break;
        }
        else
            goto L1;
        r = 11;
    }
    y = x.load(_rx) + 1;
    L1: x.store(_rx, r);
}

exists (y == x + 1 && thread_0:r > 21)
```

Figure: A program example in C

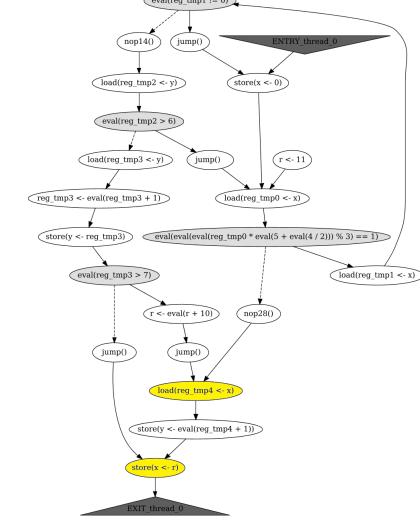


Figure: The compiled event-flow graph

## PorthosC: The input language

Support for an arbitrary control-flow

```
{ int *x = 1; }

extern int z;

void thread_0(int &x, int &y) {
    LO: x = 0;
    int r;
    while (x * (5 + 4 / 2) % 3 == 1) {
        if (x != 0)
            goto LO;
        if (y > 6)
            continue;
        else if (++y > 7) {
            r = r + 10;
            break;
        }
        else
            goto L1;
        r = 11;
    }
    y = x.load(_rx) + 1;;
    L1: x.store(_rx, r);
}

exists (y == x + 1 && thread_0:r > 21)
```

Figure: A program example in C

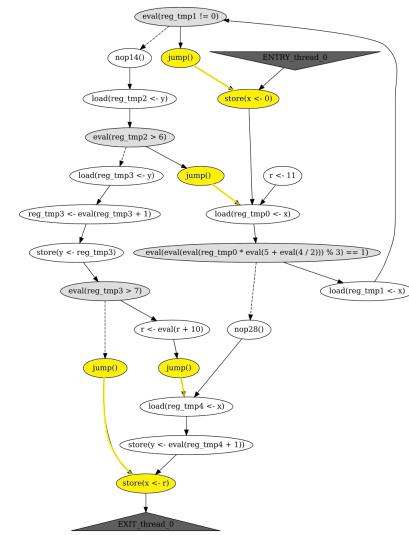
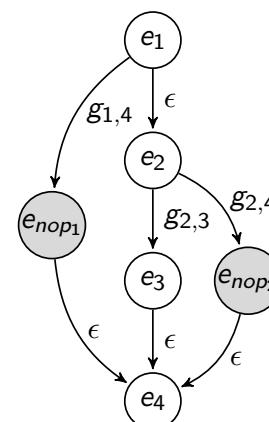


Figure: The compiled event-flow graph

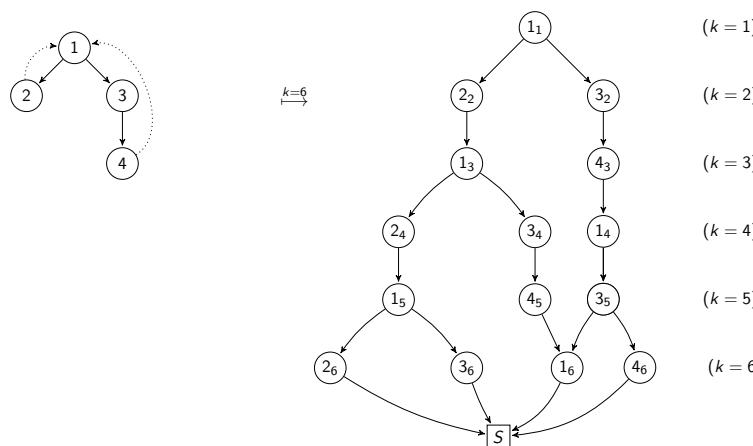
## PorthosC: The new control-flow encoding scheme



$$\begin{aligned} \phi_{CF} = & [x(e_2) \Rightarrow x(e_1)] \\ \wedge & [x(e_3) \Rightarrow x(e_2)] \\ \wedge & [x(e_{nop_1}) \Rightarrow x(e_1)] \\ \wedge & [x(e_{nop_2}) \Rightarrow x(e_2)] \\ \wedge & [x(e_4) \Rightarrow (x(e_{nop_1}) \vee x(e_3) \vee x(e_{nop_2}))] \\ \wedge & [x(e_{nop_1}) \wedge x(e_1) \Rightarrow g_{1,4}] \\ \wedge & [(x(e_3) \wedge x(e_2)) \Rightarrow g_{2,3}] \\ \wedge & [(x(e_{nop_2}) \wedge x(e_2)) \Rightarrow g_{2,4}] \\ \wedge & \neg[x(e_2) \wedge x(e_{nop_1})] \\ \wedge & \neg[x(e_3) \wedge x(e_{nop_2})] \end{aligned}$$

Figure: Example of encoding for the control-flow of the event-flow graph

PorthosC: The new DFS-based X-graph unrolling scheme

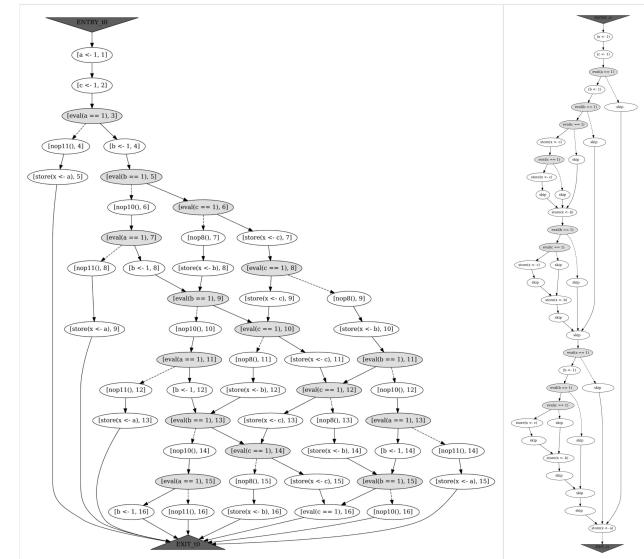


**Figure:** Example of the flow graph unrolling up to bound  $k = 6$



## Evaluation

## The new unrolling scheme

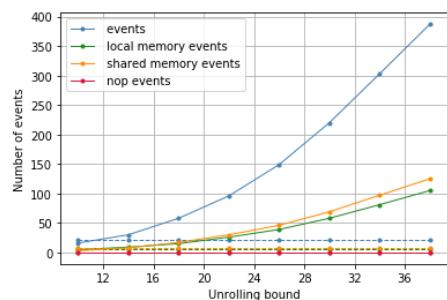


**Figure:** Illustration of differences in unrolling schemes of PorthosC (left) and Porthos v1 (right)

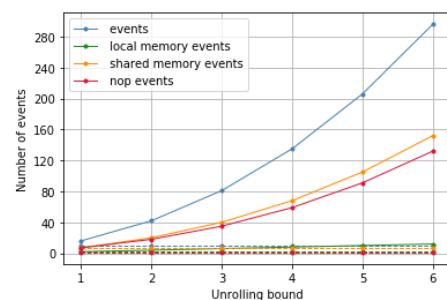


## Evaluation

Overhead of the new unrolling scheme: Number of events



(a) The graph unrolled by PorthosC

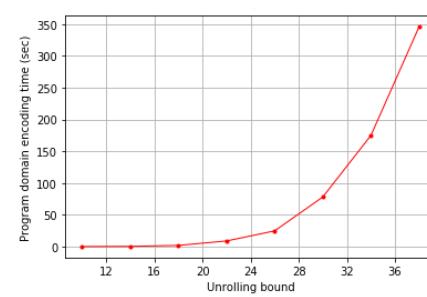


(b) The graph unrolled by Porthos v1

**Figure:** Dependency of unrolled events number on the unrolling bound  $k$

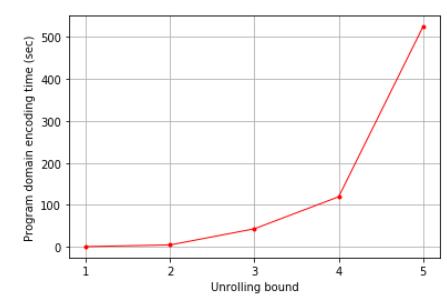


### Overhead of the new unrolling scheme: Execution time



(a) Program domain encoding time  
of the graph unrolled by PorthosC

Figure: Dependency of program domain encoding time (in seconds) on the unrolling bound



(b) Program domain encoding time  
of the graph unrolled by PorthosC

A set of small, light-blue navigation icons typically found in Beamer presentations, including symbols for back, forward, search, and table of contents.



## Summary

- ▶ The result of the work is the generalised framework for memory model-aware analysis PorthosC;
  - ▶ The *input language* has been extended to a higher subset of C language (including unconditional control-flow jumps);
  - ▶ The old *architecture* of Porthos has been revised and redesigned for PorthosC;
  - ▶ The *new unrolling scheme* produces complete state space of the analysing program within the user-defined bound  $k$ , though the encoding time growth rapidly (exponentially) as the unrolling bound grows;
  - ▶ The *invocation hooking mechanism* is an important part of the abstract interpretation engine, that serves as a knowledge base for the program domain and increases the extensibility of the tool.

## Directions for future work

- ▶ Extending the *knowledge base* of domain-specific functions to model synchronisation primitives;
- ▶ Supporting *new input languages* (e.g., different assembly languages);
- ▶ Supporting *complex data types* (s.a. arrays, pointers, structures, etc.);
- ▶ Adding the *inter-procedural analysis mode*;
- ▶ Handling the *state explosion problem* (with standard model-checking techniques adjusted by the information about the weak memory model of the execution environment).

Thank you for attention



## Bibliography I

- Jade Alglave. "A shared memory poetics". In: *La Thèse de doctorat, L'université Paris Denis Diderot* (2010).
- Hernán Ponce de León et al. "Portability Analysis for Weak Memory Models. PORTHOS: One Tool for all Models". In: *Static Analysis - 24th International Symposium, SAS 2017, New York, NY, USA, August 30 - September 1, 2017, Proceedings*. 2017, pp. 299–320. DOI: 10.1007/978-3-319-66706-5\_15. URL: [https://doi.org/10.1007/978-3-319-66706-5\\_15](https://doi.org/10.1007/978-3-319-66706-5_15).
- Hernán Ponce de León et al. *Slides: Portability Analysis for Weak Memory Models. PORTHOS: One Tool for all Models*. Aug. 2017. URL: [https://www.researchgate.net/publication/319403158\\_Slides\\_Portability\\_Analysis\\_for\\_Weak\\_Memory\\_Models\\_PORTHOS\\_One\\_Tool\\_for\\_all\\_Models](https://www.researchgate.net/publication/319403158_Slides_Portability_Analysis_for_Weak_Memory_Models_PORTHOS_One_Tool_for_all_Models).
- Peter Sewell et al. "x86-TSO: a rigorous and usable programmer's model for x86 multiprocessors". In: *Communications of the ACM* 53.7 (2010), pp. 89–97.