Автоматический анализ слабых моделей памяти параллельного программирования

Юшковский А. В.[1,2]

магистрант

Руководители: **проф. Кеийо Хельянко**[1]

доц. Комаров И. И.[2]

[1]**Университет Аалто** (Эспоо, Финляндия)
Факультет Информатики, Школа Наук

[2]**Университет ИТМО** (Санкт-Петербург, Россия)
Факультет Безопасности Информационных Технологий

Эспоо, Санкт Петербург
2018

Цель работы

- Усовершенствовать прототип статического анализатора кода `Porthos` [2], учитывающий при анализе модель памяти вычислительной среды, путем расширения поддержки входного языка.

Задачи работы

- Изучить существующие подходы к автоматическому проведению анализа кода параллельных программ с учетом модели памяти вычислительной среды [1];
- Исследовать существующие инструменты анализа кода с учетом модели памяти;
- Разработать инфраструктуру C-компилятора как часть модуля абстрактной интерпретации нового анализатора `PorthosC`;
- Адаптировать схему кодирования произвольного графа потока управления в SMT-формулу;
- Сохранить и улучшить функциональные характеристики анализатора (производительность, расширяемость, надежность и поддерживаемость).

Верификация параллельных программ

Пример: Перестановка типа запись-запись (оптимизация на уровне компилятора)

{ 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 $(r_p = 0 \wedge r_q = 0)$	

Верификация параллельных программ

Пример: Перестановка типа запись-запись (оптимизация на уровне компилятора)

{ 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 ($r_p = 0 \wedge r_q = 0$)	

Один поток (отсутствие параллелизма)

$p_0, p_1, q_0, q_1 \quad (0; 1)$
$q_0, q_1, p_0, p_1 \quad (1; 0)$

Верификация параллельных программ

Пример: Перестановка типа запись-запись (оптимизация на уровне компилятора)

{ 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 $(r_p = 0 \wedge r_q = 0)$	

Последовательная Согласованность (классический параллелизм)

p_0, p_1, q_0, q_1 $(0; 1)$
q_0, q_1, p_0, p_1 $(1; 0)$
p_0, q_0, p_1, q_1 $(1; 1)$
p_0, q_0, q_1, p_1 $(1; 1)$
q_0, p_0, p_1, q_1 $(1; 1)$
q_0, p_0, q_1, p_1 $(1; 1)$

Верификация параллельных программ

Пример: Перестановка типа запись-запись (оптимизация на уровне компилятора)

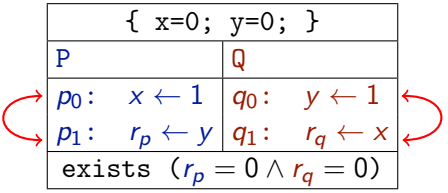
{ x=0; y=0; }	
P	Q
p_0: $x \leftarrow 1$	q_0: $y \leftarrow 1$
p_1: $r_p \leftarrow y$	q_1: $r_q \leftarrow x$
exists ($r_p = 0 \wedge r_q = 0$)	

Линейный порядок записи
(напр., x86)

p_0, p_1, q_0, q_1	$(0;1)$	$\underline{p_1}, \underline{p_0}, q_0, q_1$	$(0;1)$	$p_0, p_1, \underline{q_1}, \underline{q_0}$	$(0;1)$	$\underline{p_1}, \underline{p_0}, \underline{q_1}, \underline{q_0}$	$(0;1)$
q_0, q_1, p_0, p_1	$(1;0)$	$q_0, q_1, \underline{p_1}, \underline{p_0}$	$(1;0)$	$\underline{q_1}, \underline{q_0}, p_0, p_1$	$(1;0)$	$\underline{q_1}, \underline{q_0}, \underline{p_1}, \underline{p_0}$	$(1;0)$
p_0, q_0, p_1, q_1	$(1;1)$	$\underline{p_1}, q_0, \underline{p_0}, q_1$	$(0;1)$	$p_0, \underline{q_1}, p_1, \underline{q_0}$	$(0;1)$	$\underline{p_1}, \underline{q_1}, \underline{p_0}, \underline{q_0}$	$(0;0)$
p_0, q_0, q_1, p_1	$(1;1)$	$\underline{p_1}, q_0, q_1, \underline{p_0}$	$(0;1)$	$p_0, \underline{q_1}, \underline{q_0}, p_1$	$(1;1)$	$\underline{p_1}, \underline{q_1}, \underline{q_0}, \underline{p_0}$	$(0;0)$
q_0, p_0, p_1, q_1	$(1;1)$	$q_0, \underline{p_1}, \underline{p_0}, q_1$	$(1;1)$	$\underline{q_1}, p_0, p_1, \underline{q_0}$	$(0;0)$	$\underline{q_1}, \underline{p_1}, \underline{p_0}, \underline{q_0}$	$(0;0)$
q_0, p_0, q_1, p_1	$(1;1)$	$q_0, \underline{p_1}, q_1, \underline{p_0}$	$(1;0)$	$\underline{q_1}, p_0, \underline{q_0}, p_1$	$(1;0)$	$\underline{q_1}, \underline{p_1}, \underline{q_0}, \underline{p_0}$	$(0;0)$

Верификация параллельных программ

Пример: Перестановка типа запись-запись (оптимизация на уровне компилятора)



Линейный порядок записи
(напр., 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_0, p_1	(1; 0)	q_1, p_1, q_0, p_0	(0; 0)

Верификация параллельных программ

Пример: Буферизация записи (оптимизация на уровне процессора)

{ 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 ($r_p = 0 \wedge r_q = 0$)	

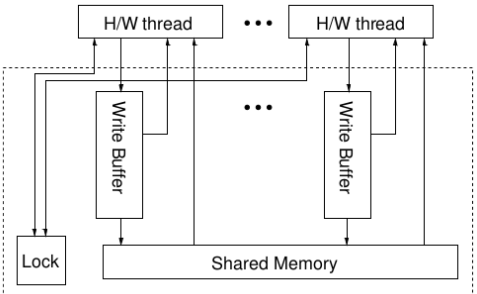


Figure: Модель абстрактной машины процессора x86 [4]

Верификация параллельных программ

Пример: Буферизация записи (оптимизация на уровне процессора)

{ 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 ($r_p = 0 \wedge r_q = 0$)	

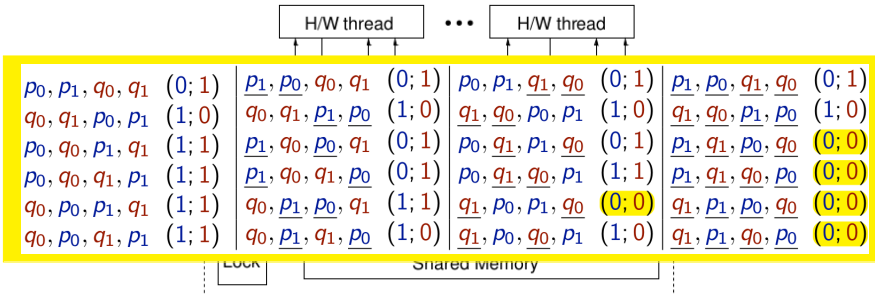


Figure: Модель абстрактной машины процессора x86 [4]

Статический анализ с учетом слабой модели памяти

Событийная модель кода

- **Событие** $\in \mathbb{E}$, низкоуровневая примитивная операция:
  - *событие памяти* $\in \mathbb{M} = \mathbb{R} \cup \mathbb{W}$: access to a local/shared memory,
  - *событие вычисления* $\in \mathbb{C}$: вычисление в локальной памяти
  - *событие синхронизации* $\in \mathbb{B}$: барьеры синхронизации;
- **Отношение** $\subseteq \mathbb{E} \times \mathbb{E}$:
  - *базовые отношения*:
    - *program-order* отношение po $\subseteq \mathbb{E} \times \mathbb{E}$: (поток управления),
    - *read-from* отношение `rf` $\subseteq \mathbb{W} \times \mathbb{R}$: (поток данных, и)
    - *coherence-order* отношение co $\subseteq \mathbb{W} \times \mathbb{W}$: (поток данных);
  - *производные отношения*:
    - *объединение* `r1 | r2`,
    - *последовательность* `r1 ; r2`,
    - *транзитивное замыкание* `r+`,
    - $\ldots$;
- **Ограничение** модели памяти, сформулированное в терминах отношений или множеств событий :
  - *ацикличность*, *антирефлексивность* or *отсутствие элементов во множестве*.

Статический анализ с учетом слабой модели памяти

Тестирование путей выполнения программы

{ 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 ($r_p = 0 \wedge r_q = 0$)	

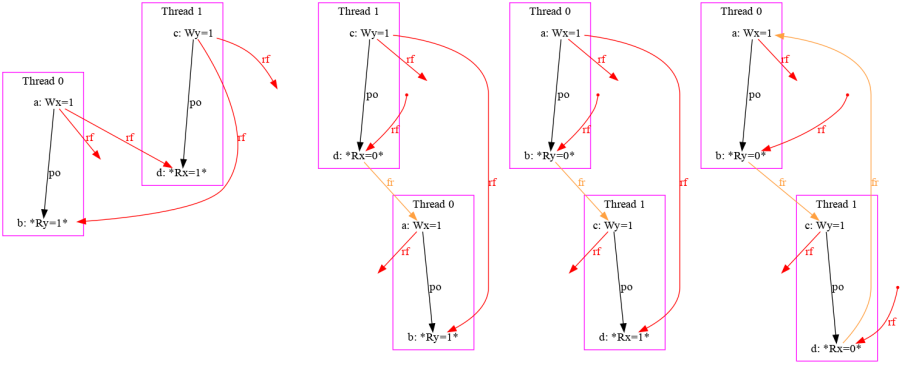


Figure: Четыре возможных выполнения программы, допустимых моделью x86-TSO

Статический анализ с учетом слабой модели памяти

Тестирование путей выполнения программы

{ 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 модель:

…

`fr = (rf^-1; co)`

`acyclic(fr ∪ po)`

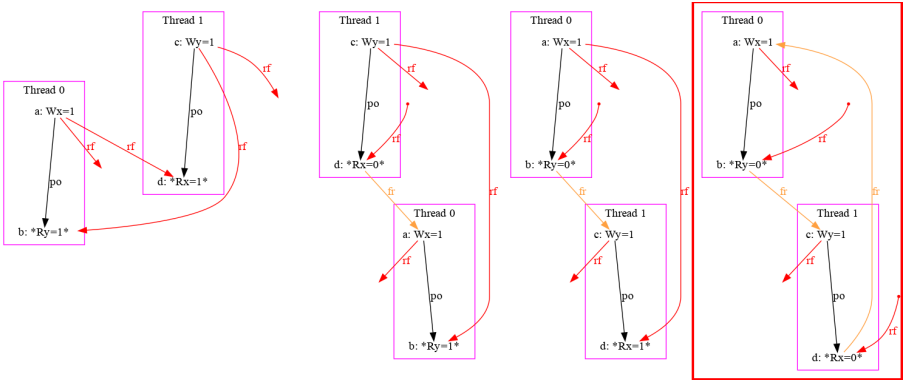


Figure: Четыре возможных выполнения программы, допустимых моделью x86-TSO

Статический анализ с учетом слабой модели памяти

Тестирование путей выполнения программы

{ 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 модель:

...

`fr = (rf^-1; co)`

`acyclic(fr ∪ po)`

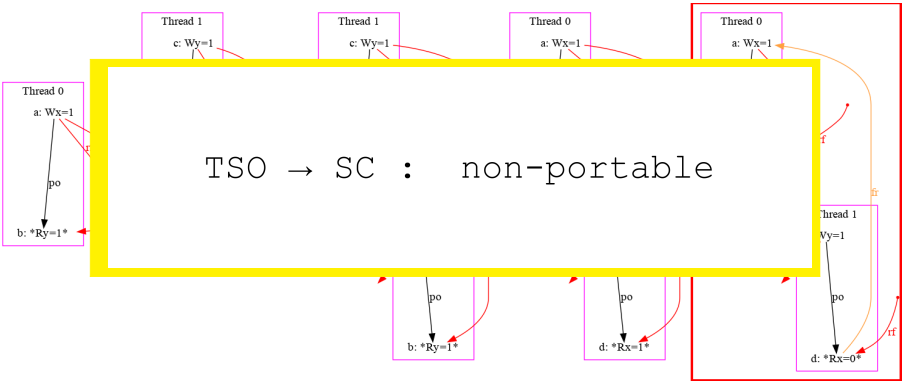


Figure: Четыре возможных выполнения программы, допустимых моделью x86-TSO

Анализ портируемости

Анализатор Porthos

```
/* Prints Hello World */
#include <stdio.h>
int main (void){
    printf ("Hello World!");
    return 0;
}
```

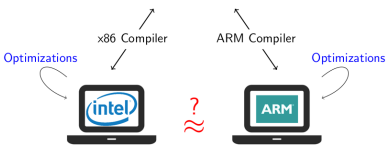


Figure: Иллюстрация проблемы определения портируемости программы [3]

Анализ портируемости

Анализатор Porthos

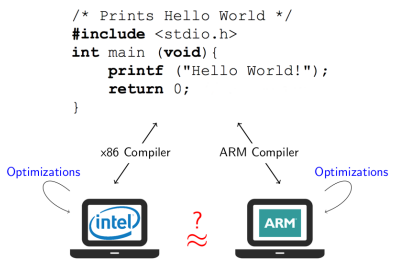


Figure: Иллюстрация проблемы определения портируемости программы [3]

Definition (Портируемость [2])

Программа P называется портируемой с платформы $\mathcal{M}_{\mathcal{S}}$ на платформу $\mathcal{M}_{\mathcal{T}}$, если $cons_{\mathcal{M}_{\mathcal{T}}}(P) \subseteq cons_{\mathcal{M}_{\mathcal{S}}}(P)$

- Портируемость как проблема ограниченной достижимости на основе SMT: $\phi = \phi_{CF} \wedge \phi_{DF} \wedge \phi_{\mathcal{M}_{\mathcal{T}}} \wedge \phi_{\neg\mathcal{M}_{\mathcal{S}}}$
- $\mathtt{SAT}(\phi) \Longrightarrow$ ошибка портирования

Анализ портируемости

Анализатор Porthos

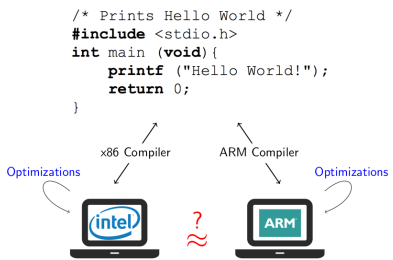


Figure: Иллюстрация проблемы определения портируемости программы [3]

Definition (Портируемость [2])

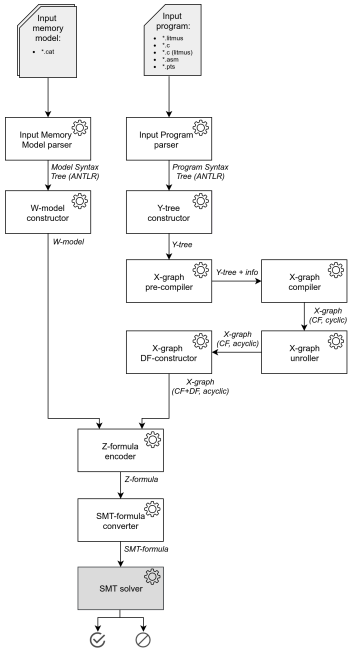
Программа P называется портируемой с платформы $\mathcal{M}_{\mathcal{S}}$ на платформу $\mathcal{M}_{\mathcal{T}}$, если $cons_{\mathcal{M}_{\mathcal{T}}}(P) \subseteq cons_{\mathcal{M}_{\mathcal{S}}}(P)$

- Портируемость как проблема ограниченной достижимости на основе SMT:
  $\phi = \phi_{CF} \wedge \phi_{DF} \wedge \phi_{\mathcal{M}_{\mathcal{T}}} \wedge \phi_{\neg \mathcal{M}_{\mathcal{S}}}$
- $\mathtt{SAT}(\phi) \Longrightarrow$ ошибка портирования

PorthosC: Архитектура

Основные компоненты

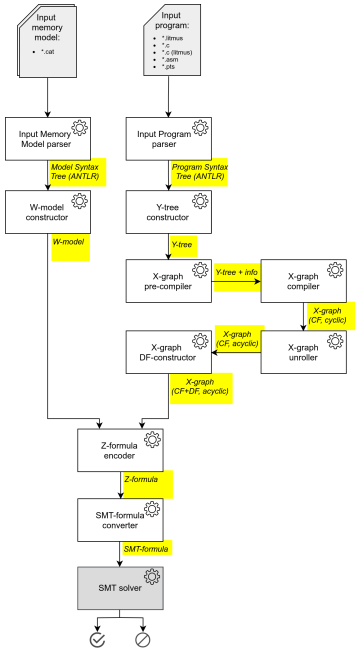
- Усовершенственная версия Porthos, способная анализировать программы на языке программирования C, была названа PorthosC.



PorthosC: Архитектура

Основные компоненты

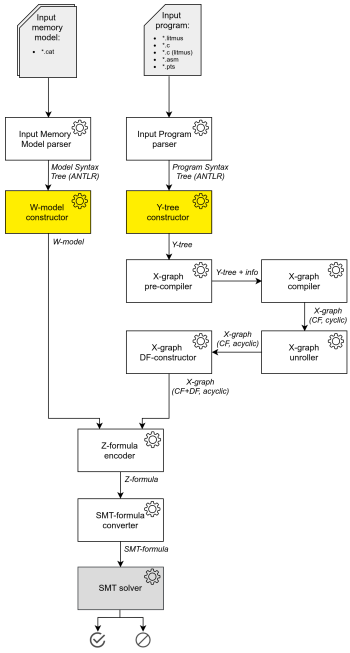
- Усовершенственная версия `Porthos`, способная анализировать программы на языке программирования C, была названа `PorthosC`.



PorthosC: Архитектура

Основные компоненты

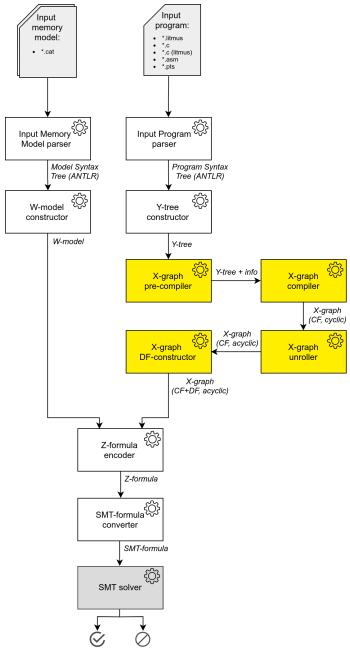
- Усовершенственная версия Porthos, способная анализировать программы на языке программирования C, была названа PorthosC.



PorthosC: Архитектура

Основные компоненты

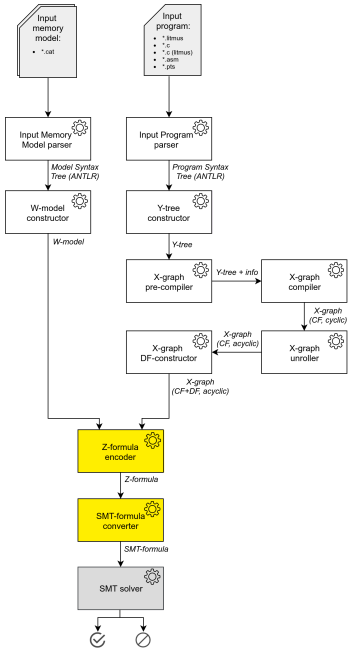
- Усовершенственная версия Porthos, способная анализировать программы на языке программирования C, была названа PorthosC.



PorthosC: Архитектура

Основные компоненты

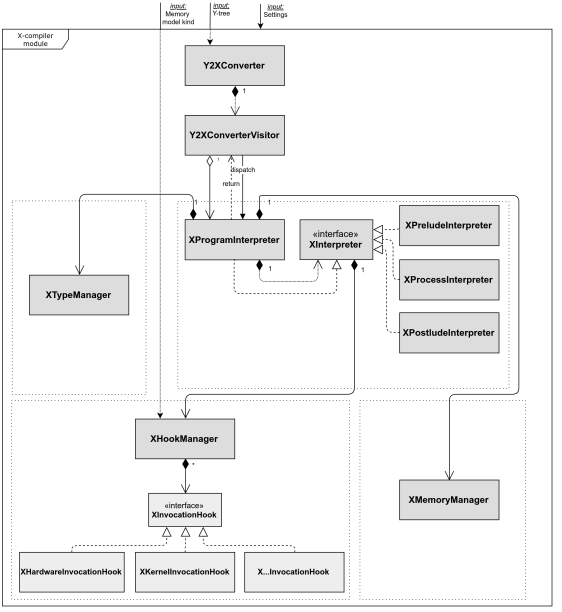
- Усовершенственная версия Porthos, способная анализировать программы на языке программирования C, была названа PorthosC.



PorthosC: Архитектура

Модуль абстрактной интерпретации

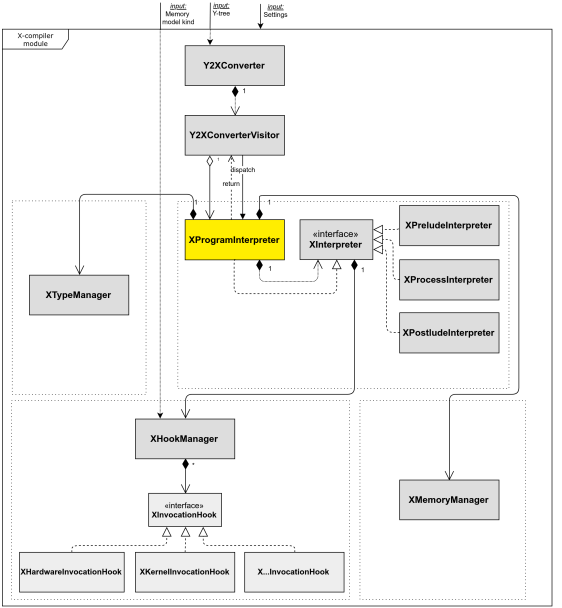
- Модуль абстрактной интерпретации является ключевым компонентом компилятора, преобразующего программы на C в низкоуровневое событийное представление (X-graph).



PorthosC: Архитектура

Модуль абстрактной интерпретации

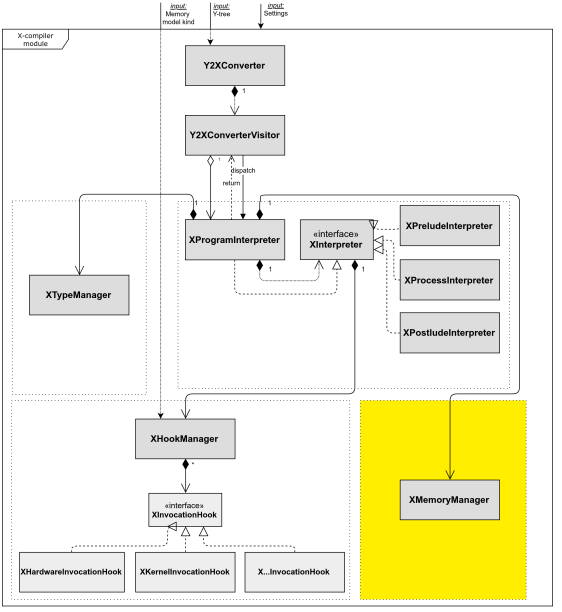
- Модуль абстрактной интерпретации является ключевым компонентом компилятора, преобразующего программы на C в низкоуровневое событийное представление (`X-graph`).



PorthosC: Архитектура

Модуль абстрактной интерпретации

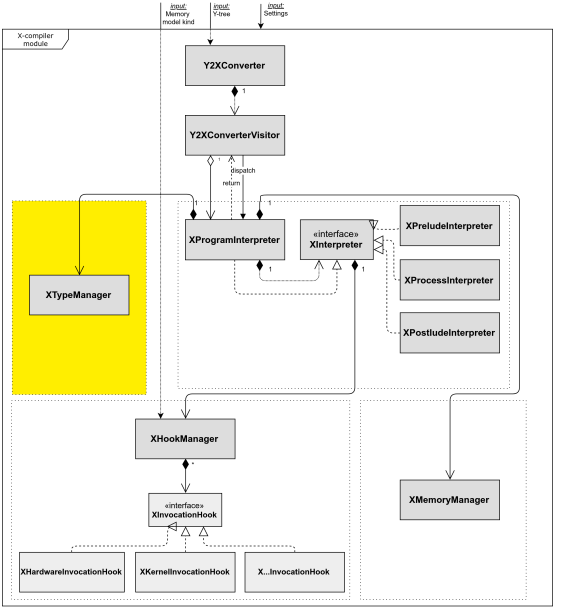
- Модуль абстрактной интерпретации является ключевым компонентом компилятора, преобразующего программы на C в низкоуровневое событийное представление (`X-graph`).



PorthosC: Архитектура

Модуль абстрактной интерпретации

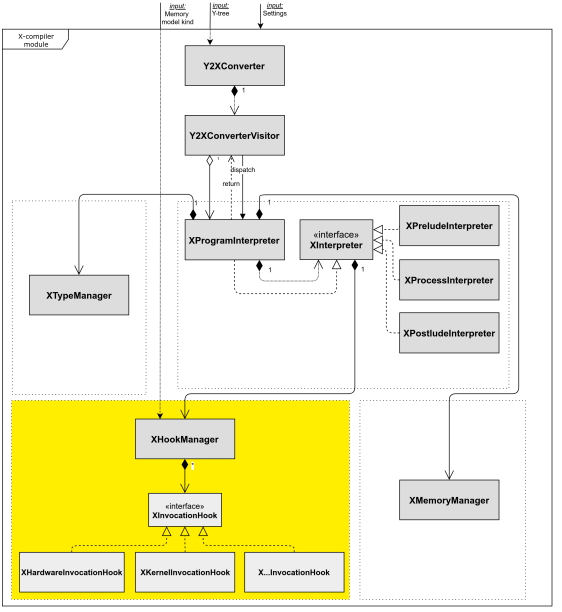
- Модуль абстрактной интерпретации является ключевым компонентом компилятора, преобразующего программы на C в низкоуровневое событийное представление (X-graph).



PorthosC: Архитектура

Модуль абстрактной интерпретации

- Модуль абстрактной интерпретации является ключевым компонентом компилятора, преобразующего программы на C в низкоуровневое событийное представление (X-graph).



Porthos v1: Входной язык

```
{ 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,
```

Figure: Пример программы на входном языке анализатора Porthos v1

Porthos v1: Входной язык

```
{ 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,
```

Figure: Пример программы на входном языке анализатора Porthos v1

```
...

⟨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: Набросок грамматики входного языка анализатора Porthos v1

Porthos v1: Входной язык

Статическое определение вида переменных на синтаксическом уровне

```
{ 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,
```

Figure: Пример программы на входном языке анализатора Porthos v1

```
...

⟨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: Набросок грамматики входного языка анализатора Porthos v1

Porthos v1: Входной язык

Отсутствие поддержки вызовов функций

```
{ 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,
```

Figure: Пример программы на входном языке анализатора Porthos v1

```
...

⟨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: Набросок грамматики входного языка анализатора Porthos v1

Porthos v1: Входной язык

Отсутствие поддержки безусловных переходов

```
{ 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,
```

Figure: Пример программы на входном языке анализатора Porthos v1

```
...

⟨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: Набросок грамматики входного языка анализатора Porthos v1

PorthosC: Входной язык

```
{ int *x = 1; }

extern int z;

void thread_0(int &x, int &y) {
  L0: x = 0;
  int r;
  while (x * (5 + 4 / 2) % 3 == 1) {
    if (x != 0)
      goto L0;
    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: Пример программы на языке C

PorthosC: Входной язык

```
{ int *x = 1; }

extern int z;

void thread_0(int &x, int &y) {
  L0: x = 0;
  int r;
  while (x * (5 + 4 / 2) % 3 == 1) {
    if (x != 0)
      goto L0;
    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: Пример программы на языке C

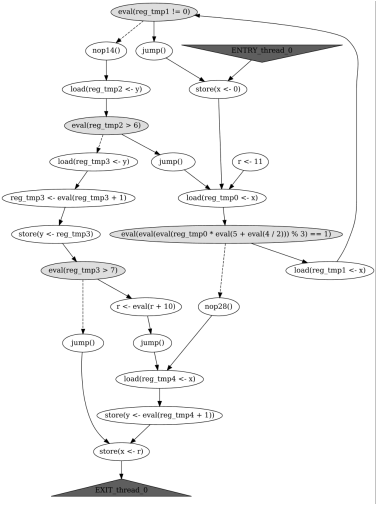


Figure: Скомпилированный граф потока событий

PorthosC: Входной язык

Стадия предкомпиляции для синтаксического определения вида переменных

```
{ int *x = 1; }

extern int z;

void thread_0(int &x, int &y) {
  L0: x = 0;
  int r;
  while (x * (5 + 4 / 2) % 3 == 1) {
    if (x != 0)
      goto L0;
    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: Пример программы на языке C

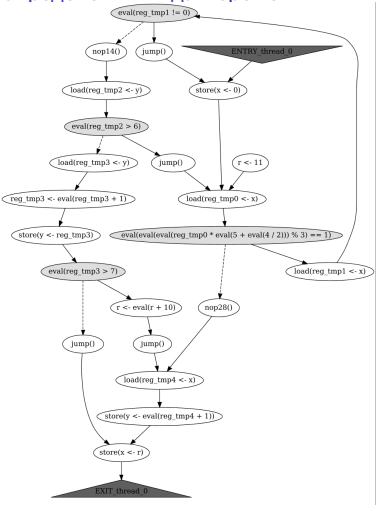


Figure: Скомпилированный граф потока событий

PorthosC: Входной язык

Поддержка вызовов функций: Механизм перехвата вызовов

```
{ int *x = 1; }

extern int z;

void thread_0(int &x, int &y) {
  L0: x = 0;
  int r;
  while (x * (5 + 4 / 2) % 3 == 1) {
    if (x != 0)
      goto L0;
    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: Пример программы на языке C

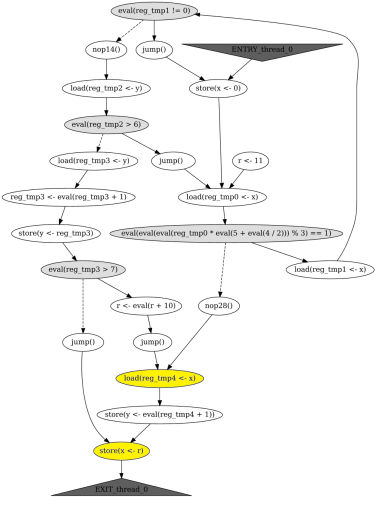


Figure: Скомпилированный граф потока событий

PorthosC: Входной язык

Поддержка произвольного ветвления графа потока управления

```
{ int *x = 1; }

extern int z;

void thread_0(int &x, int &y) {
  L0: x = 0;
  int r;
  while (x * (5 + 4 / 2) % 3 == 1) {
    if (x != 0)
      goto L0;
    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: Пример программы на языке C

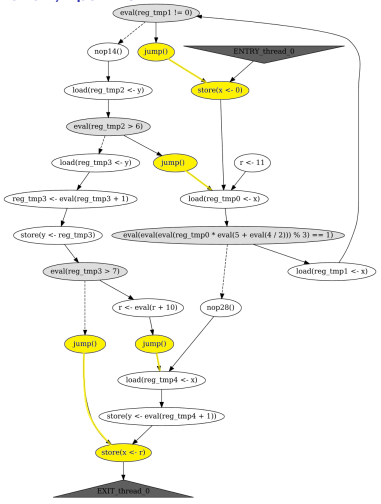
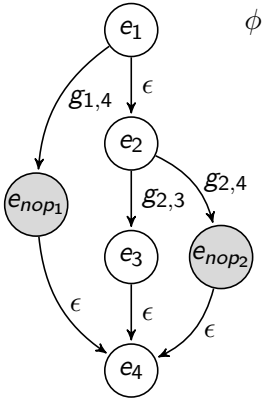


Figure: Скомпилированный граф потока событий

PorthosC: Новая схема кодирования потока управления



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

Figure: Пример кодирования потока управления с произвольным ветвлением

PorthosC: Новая схема развертки циклов, основанная на обходе графа в глубину

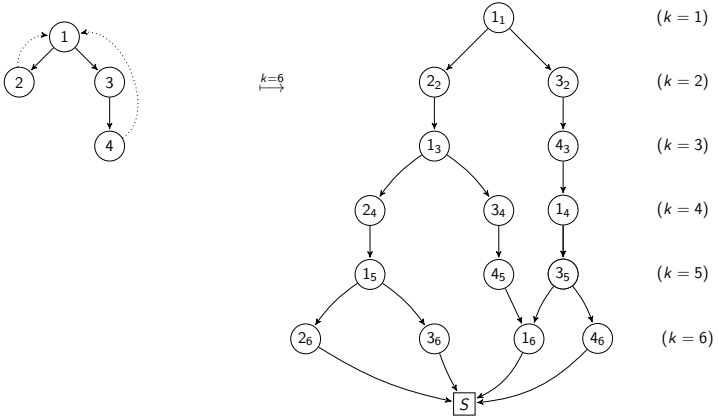


Figure: Пример развертки цикла до границы $k = 6$

Анализ программы

Новая схема развертки циклов

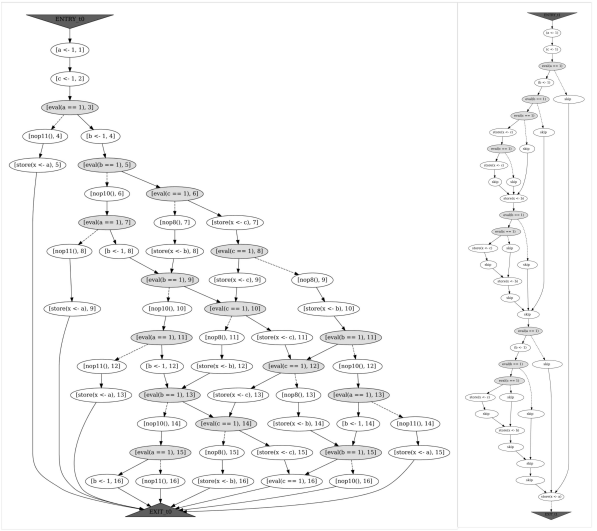
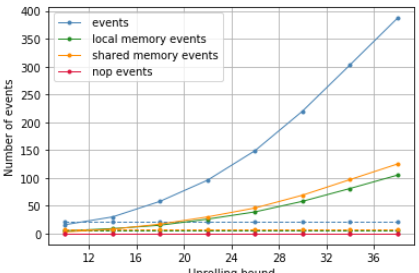


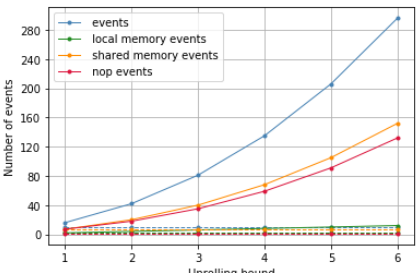
Figure: Иллюстрация различий в алгоритмах развертки циклов используемых `PorthosC` (слева) и `Porthos v1` (справа)

Анализ программы

Сравнение схем развертки циклов: Количество событий



(a) Граф, развернутый анализатором `PorthosC`

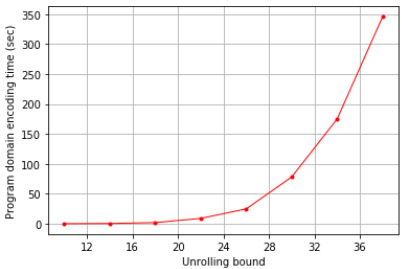


(b) Граф, развернутый анализатором `Porthos v1`

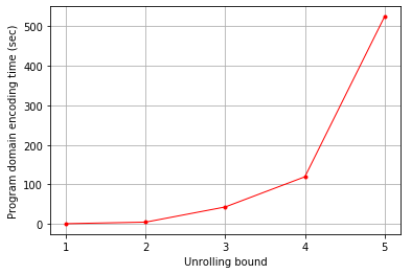
Figure: Зависимость количества событий от границы развертки циклов k

Анализ программы

Сравнение схем развертки циклов: Время кодирования программы



(a) Время кодирования программы в SMT-формулы, произведенное анализатором `PorthosC`



(b) Время кодирования программы в SMT-формулы, произведенное анализатором `Porthos v1`

Figure: Зависимость времени кодирования программы от границы развертки циклов k

Заключение

- Результатом выпускной квалификационной работы является платформа для проведения статического анализа кода с учетом слабых моделей памяти сред выполнения `PorthosC`;
  - В рамках работы над анализатором была расширена поддержка *входной языка C* (включая поддержку безусловных переходов);
  - Была пересмотрена и модифицирована общая *архитектура* анализатора;
  - Измененная *схема развертки циклов*, работающая на нерекурсивной структуре графа событий, производит *полное множество* выполнений программы, что улучшает качество производимого анализа;
  - *Механизм перехвата вызовов*, являющийся важным элементом модуля абстрактной интерпретации, служит в качестве базы знаний семантики функций и таким образом повышает расширяемость анализатора.

Направления работы в будущем

- Расширение *базы знаний* функций для моделирования различных примитивов синхронизации;
- Поддержка новых *входных языков* (напр., языков ассемблера различных архитектур);
- Поддержка *комплексных структур данных* (таких как массивы, структуры, указатели, и т.д.);
- Добавление режима *кросс-процедурного анализа*;
- Решение проблемы *комбинаторного взрыва пространства состояний* (с помощью стандартных техник анализа кода, модифицированных с целью учитывания модели памяти среды выполнения).

Спасибо за внимание

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.

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.

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.