МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
Факультет информационных технологий
Кафедра параллельных вычислений
09.03.01 Информатика и вычислительная техника. Программная
инженерия и компьютерные науки

Обнаружение семантических ошибок во фрагментированных программах для системы LuNA при помощи технологии Model Checking

Выполнил: Усенко Никита Сергеевич

Научный руководитель: Власенко Андрей Юрьевич, к.т.н., доц. каф. ПВ ФИТ НГУ

Соруководитель: Матвеев Алексей Сергеевич, ст. преп. каф. ПВ ФИТ НГУ

Новосибирск 2025

20.06.2025

Предметная область

LuNA (Language for Numerical Algorithms) — система, позволяющая упростить рабочий процесс программиста за счет сокращения трудозатрат на создание параллельных реализаций алгоритмов.

Фрагмент данных (ФД) – единица информации.

Фрагмент вычислений (ФВ) – порождаемая во время выполнения программы единица работы.

Фрагмент кода (ФК) – пользовательская процедура (LuNA, C/C++).

Актуальность

Системе LuNA свойственны специфические виды ошибок.

- Имеется недостаток инструментальных средств отладки.
- Встроенные средства зачастую не предоставляют пользователю понятных сообщений об ошибках.

Существующие методы и средства отладки параллельных программ

- диалоговая отладка (TotalView, Distributed Debugging Tool);
- сравнительная отладка (DVM);
- динамический анализ (Valgrind);
- статический анализ (CSA);
- анализ по трассе (Intel Message Checker);
- **верификация модели программы** (SPIN, TLC).
- …

Примеры существующих решений для автоматизированной отладки

1. Clang Static Analyzer (CSA)*

Используется для поиска ошибок в программах на C/C++.

Анализирует исходный код на этапе компиляции.

Позволяет находить утечки памяти, неопределенное поведение, ошибки при работе с памятью и т.д.

* https://clang-analyzer.llvm.org

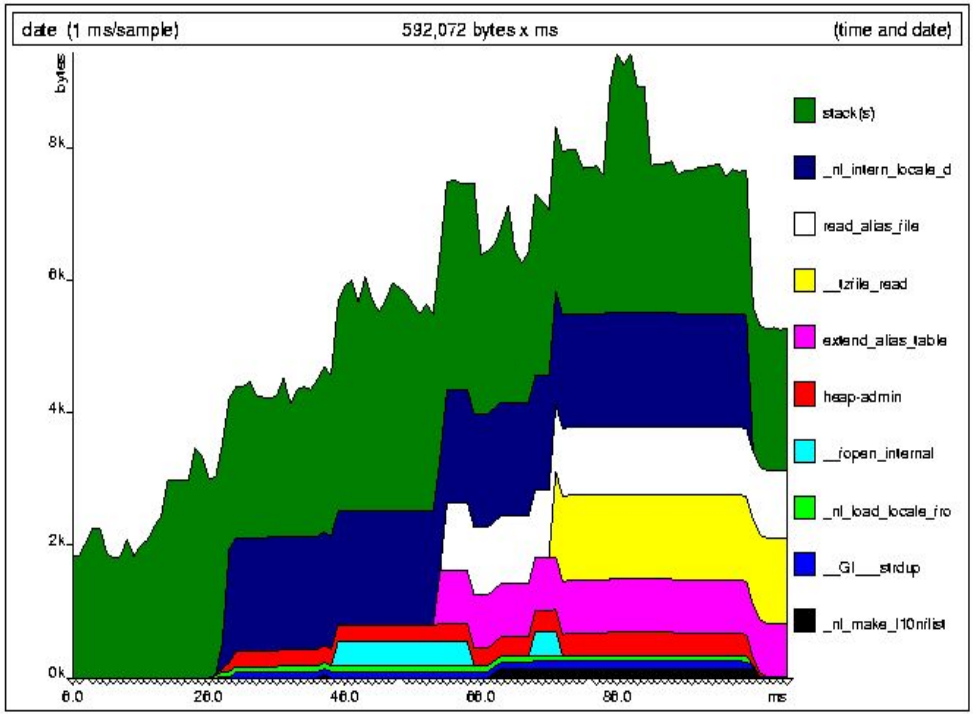
Примеры существующих решений для автоматизированной отладки

2. Valgrind*

Инструмент динамического анализа.

Анализирует программу во время выполнения.

Позволяет обнаруживать утечки памяти, проводить профилирование, анализировать использование памяти.



* https://valgrind.org

Примеры существующих решений для автоматизированной отладки

3. Simple Promela INterpreter (SPIN)*

Инструмент для обнаружения ошибок в параллельных программах методом верификации на моделях.

Проводит поиск всех возможных состояний системы и проверяет, удовлетворяют ли они заданным свойствам.

* https://spinroot.com

Цель и задачи

Цель

Проектирование и разработка системы отладки LuNA-программ методом верификации на моделях.

Задачи

- Проанализировать ошибки, присущие фрагментированным программам.
- Разработать и протестировать программное средство, реализующее обнаружение ошибок в LuNA-программах методом верификации на моделях.
- Интегрировать разработанное программное средство в существующую систему автоматизированной отладки ADAPT*.

* Власенко А.Ю., Мичуров М.А., Царёв В.Д., Курбатов М.А. Построение комплекса автоматизированной отладки фрагментированных программ // Вестник НГУ. Серия: Информационные технологии. 2024. Т.22, №1. С. 5–20. https://doi.org/10.25205/1818-7900-2024-22-1-5-20

Научная новизна работы состоит в том, что метод Model Checking впервые применен для автоматизированного обнаружения семантических ошибок во фрагментированных программах.

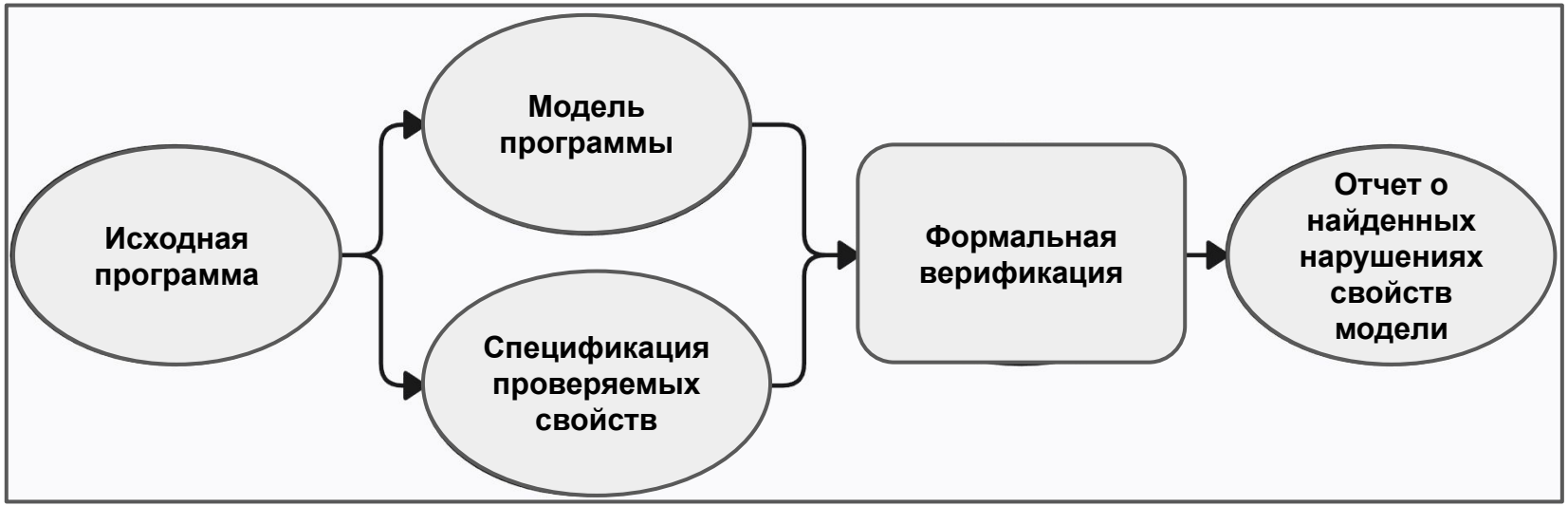
Практическая ценность работы состоит в автоматическом обнаружении ошибок в LuNA-программах во время разработки и повышении скорости выхода конечных программных продуктов.

Свойства* моделей в методе Model Checking

- абстракция данных;
- конечность;
- корректность (модель соответствует исходной программе);
- адекватность (модель соответствует проверяемым свойствам).

* Кларк Э.М., Грамберг О., Пелед Д. Верификация моделей программ : Model Checking. – 2002. – №1 (416). – С. 37-54.

Общая схема верификации на моделях



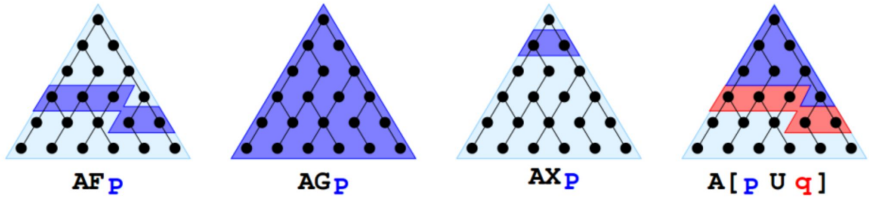
Верификатор SPIN, описание моделей на языке Promela

LTL (Linear temporal logic) – логика линейного времени.

Promela (Process Meta Language) – недетерминированный язык, задача которого описывать такие модели, которые могут быть верифицированы.

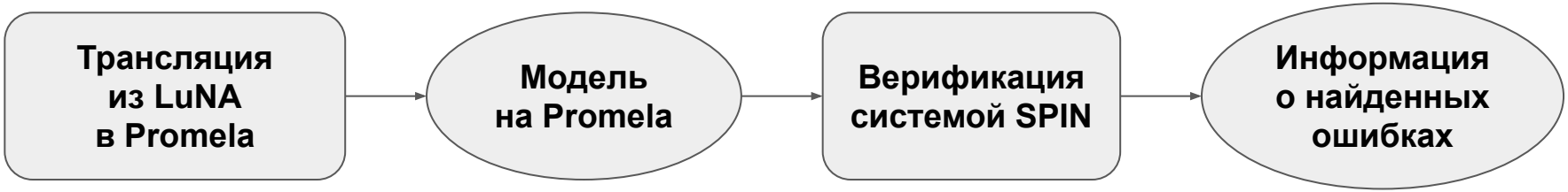
SPIN используется для верификации Promela-программ.

Примеры свойств LTL



Найдется состояние <>p	Верно всегда []p	В следующем состоянии	Верно *p* до наступления условия *q*

Разработка анализатора MC-analyzer



Вспомогательные конструкции для описания моделей

init – соответствует инициализации ФД. Увеличивает счетчик инициализаций ФД.

use – соответствует использованию ФД. Увеличивает счетчик использований ФД.

depends_on – соответствует зависимости по данным (для инициализации ФД необходим другой ФД).

enddef – соответствует концу блока видимости ФД.

destroy – соответствует рекомендации удаления. Увеличивает счетчик удалений.

inc_true – соответствует истинному условию. Увеличивает счетчик истинных условий в if/while.

inc_false – соответствует ложному условию. Увеличивает счетчик ложных условий в if/while.

LTL-свойства для обнаружения ошибок в LuNA-программах

В настоящий момент база типов ошибок* насчитывает 20 синтаксических и 20 семантических типов ошибок.

SEM2.1 – повторная инициализация одиночного ФД	**[] (init_count_varN < 2)**
SEM3.1 – неинициализированный ФД используется вне цикла	**[] (enddef_label_varN -> (is_used_varN -> is_init_varN))**
SEM3.2 – циклическая зависимость по данным	**[] !(depends_on_varM_varK &&…&& depends_on_varN_varM)**
SEM3.6 – использование ФД после его удаления	**[] (destroy_count_varN > 0 -> use_count_varN < 2)**
SEM4 – неиспользуемый ФД	**[] (enddef_label_varN -> (is_init_varN -> is_used_varN))**
SEM5 – формула в if/while тождественно истинна/ложна	**[] ((true_count_condN == 0) && (false_count_condN == 0))**
SEM6 – формула в if/while истинна/ложна во всех путях выполнения)	**[] (((true_count_condN > 0)->(<>(false_count_condN > 0))) && ((false_count_condN > 0)->(<> (true_count_condN > 0)))**

* Полный перечень типов ошибок и их описания приведен на сайте:
https://github.com/LuNA-Static-Analysis/LuNA-Static-Analysis-Repository/wiki/База-ошибок

LTL-свойство для SEM6

SEM6 – формула в if/while истинна/ложна во всех путях выполнения.

```
ltl SEM6_condN { [] (
        ( (true_count_condN > 0) -> (<> (false_count_condN > 0)) ) &&
        ( (false_count_condN > 0) -> (<> (true_count_condN > 0)) )
)}
```

Для любого состояния модели:

- Если хотя бы раз условие было истинным,

  то найдется такое состояние, когда условие будет ложным.
- Если хотя бы раз условие было ложным,

  то найдется такое состояние, когда условие будет истинным.

Пример модели LuNA программы

Программа на LuNA

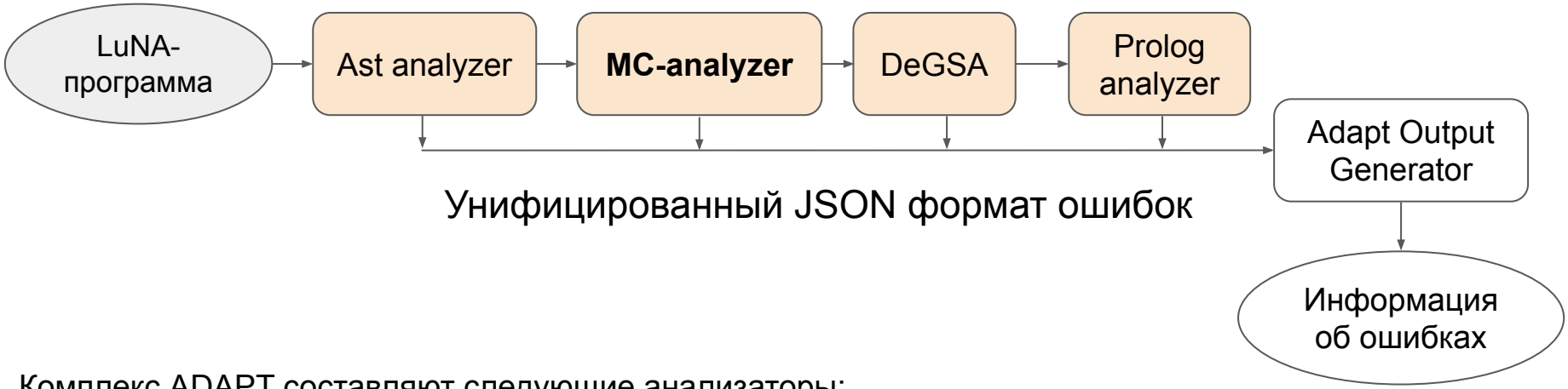
```
import c_init_int(name, int) as init;
import c_print_int(int) as print;


sub main(){
    df x, y;
    init(x, y);
    if (x == 1) {
        while ( x == 1 ), i = 0 .. out N {
            if (x == x + 1) { // всегда ложь
                init(y, x);
            }
        }
    }
}
```

Модель на Promela

```
active proctype main() {
  if
  :: cond0=0;cond1=0
  :: cond0=0;cond1=1
  :: cond0=1;cond1=0
  :: cond0=1;cond1=1
  fi
  enddef(constant);
  def(var0);
  def(var1);
  init(var0);
  depends_on(var0, var1);
  use(var1);
  use(var0);
```

```
if
:: cond0 -> use(var0);
   if
   :: cond0 -> inc_true(cond0);
         use(var0);
         use(var0);
      if
      :: false -> inc_true(cond1)
            init(var1);
            depends_on(var1, var0);
            use(var0);
      :: else -> inc_false(cond1);
      fi
   :: else -> inc_false(cond0);
   fi
:: else -> inc_false(cond0);
fi
enddef(var0); enddef(var1);}
```

Интеграция в ADAPT



Комплекс ADAPT составляют следующие анализаторы:

- **Ast analyzer** – поиск ошибок по абстрактному синтаксическому дереву (AST).
- **MC-analyzer** – поиск ошибок методом верификации на модели.
- **DeGSA** – поиск ошибок по графу информационных зависимостей.
- **Prolog analyzer** – поиск ошибок на основе логического представления программы на языке Prolog.

Пример ошибки в формате ADAPT

```
{
  "error_code":"SEM6"          // код ошибки
  "details":{                  // описание ошибки
    "condition":"a > 0",       // условное выражение в if
    "type":"true",             // всегда истина
    "where":{                  // конкретная строка в файле, где найдена ошибка
      "file":"/tmp/main.fa",   // путь к исполняемому файлу
      "line":"5",              // номер строки в этом файле
      "name":"if"              // ошибка в конструкции if
    }
  }
}
```

Вывод анализатора

(1) warning[SEM6]: Condition a > 0 is always true.

In:

File "/tmp/main.fa", line 5, in if

Нагрузочное тестирование №1

Программа на LuNA

```
import c_init_int(int, name) as init;
import c_print_int(int) as print;

sub main() {
    df a;
    print(a[0]);
    print(a[1]);
    ...
    print(a[N-1]);
}
```

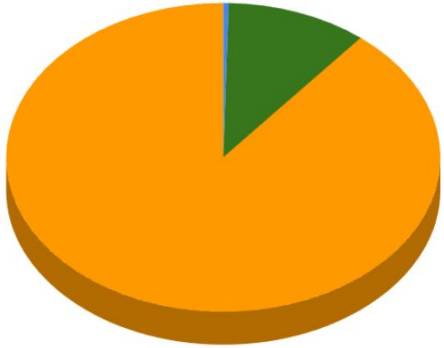
Модель на Promela

```
ltl SEM3_1_var0 {[] (check_init(var0))}
ltl SEM3_1_var1 {[] (check_init(var1))}
ltl SEM3_1_var2 {[] (check_init(var2))}
…
```

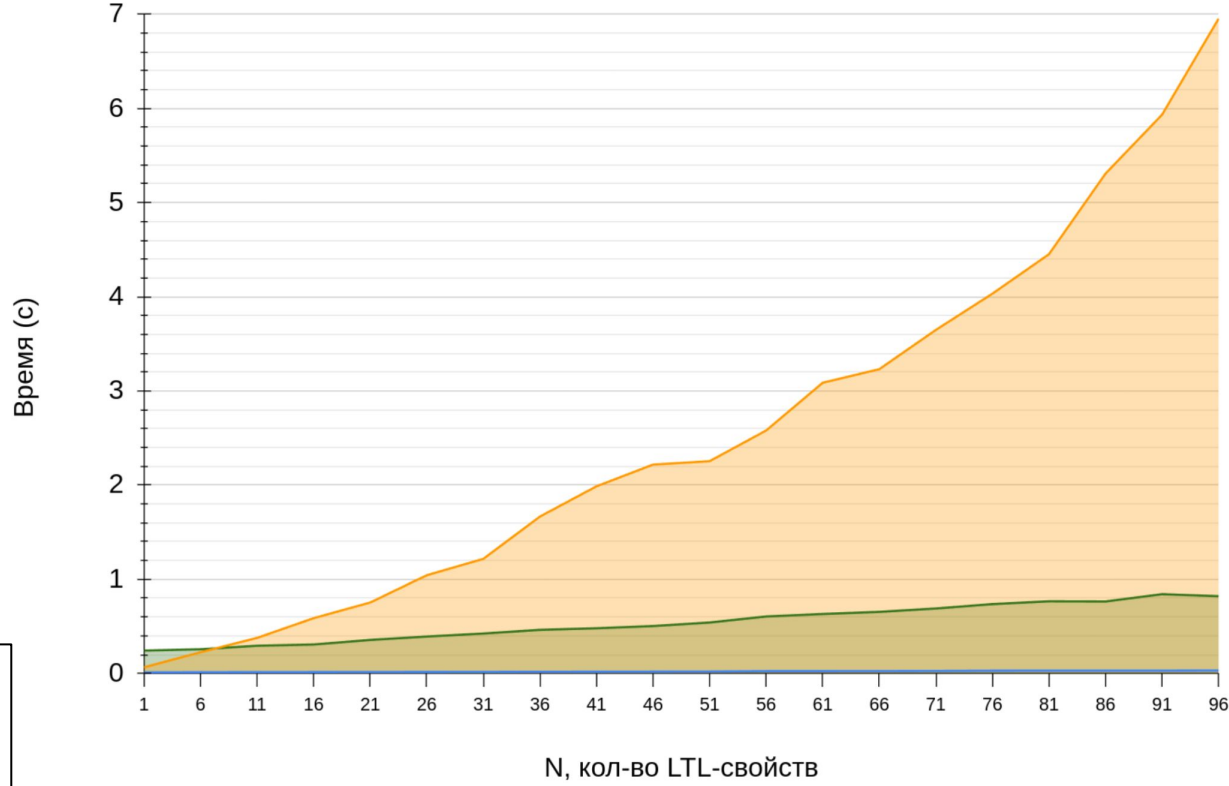
N *свойств*

```
active proctype main() {
 def(var0); def(var1); def(var2);
 use(var0); use(var1); use(var2);
 enddef(var0); enddef(var1); enddef(var2);
}
```

Нагрузочное тестирование №1



- Генерация Promela-программы
- Компиляция Promela-программы
- Проверка всех свойств

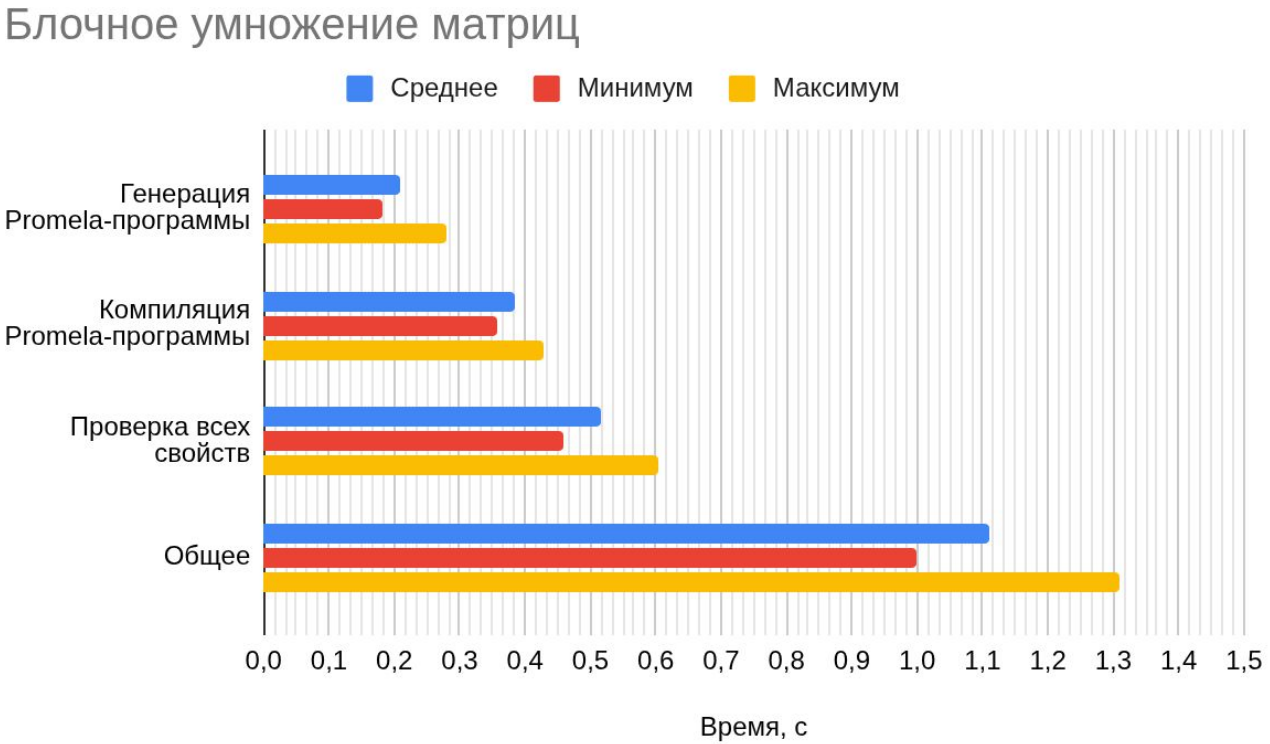


Компьютер для тестирования

Процессор: AMD Ryzen 7 7730U

ОЗУ: 64 Гб

Оценка накладных расходов на существующих программах



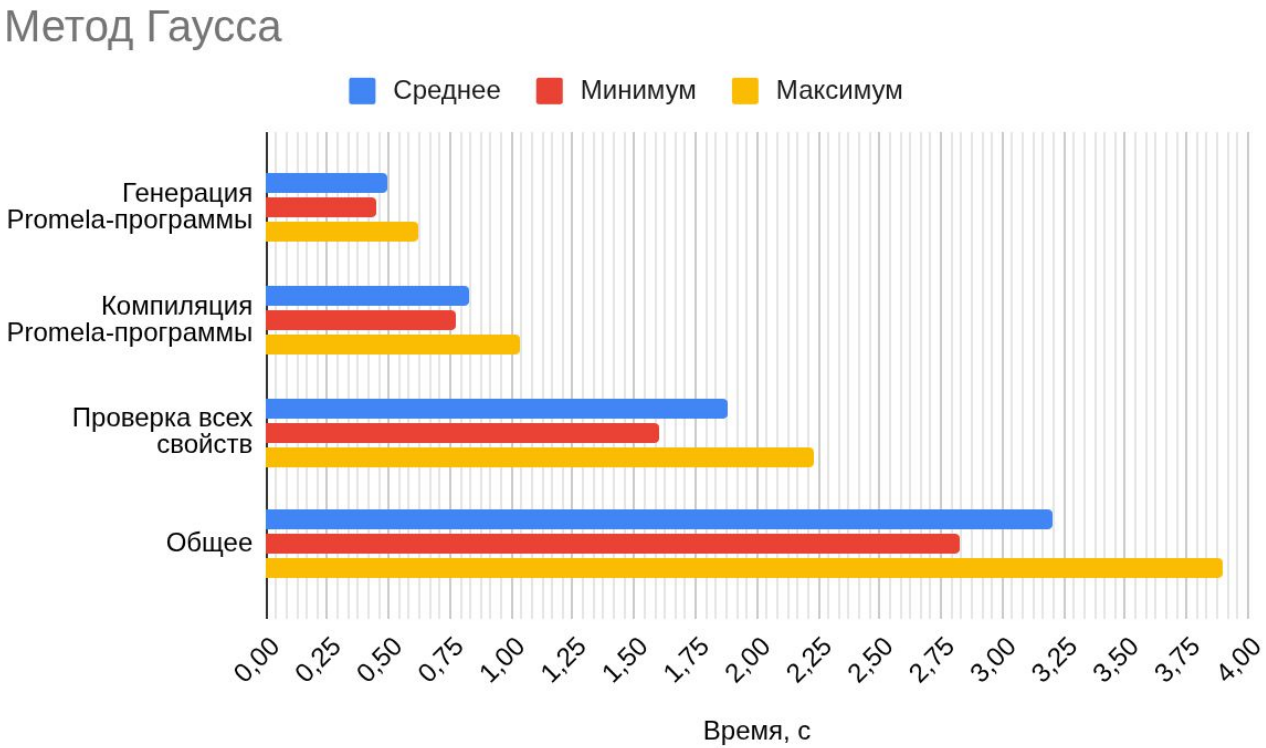
Количество запусков: 100

Строк кода в LuNA-файле: 68

Количество узлов в AST: 460

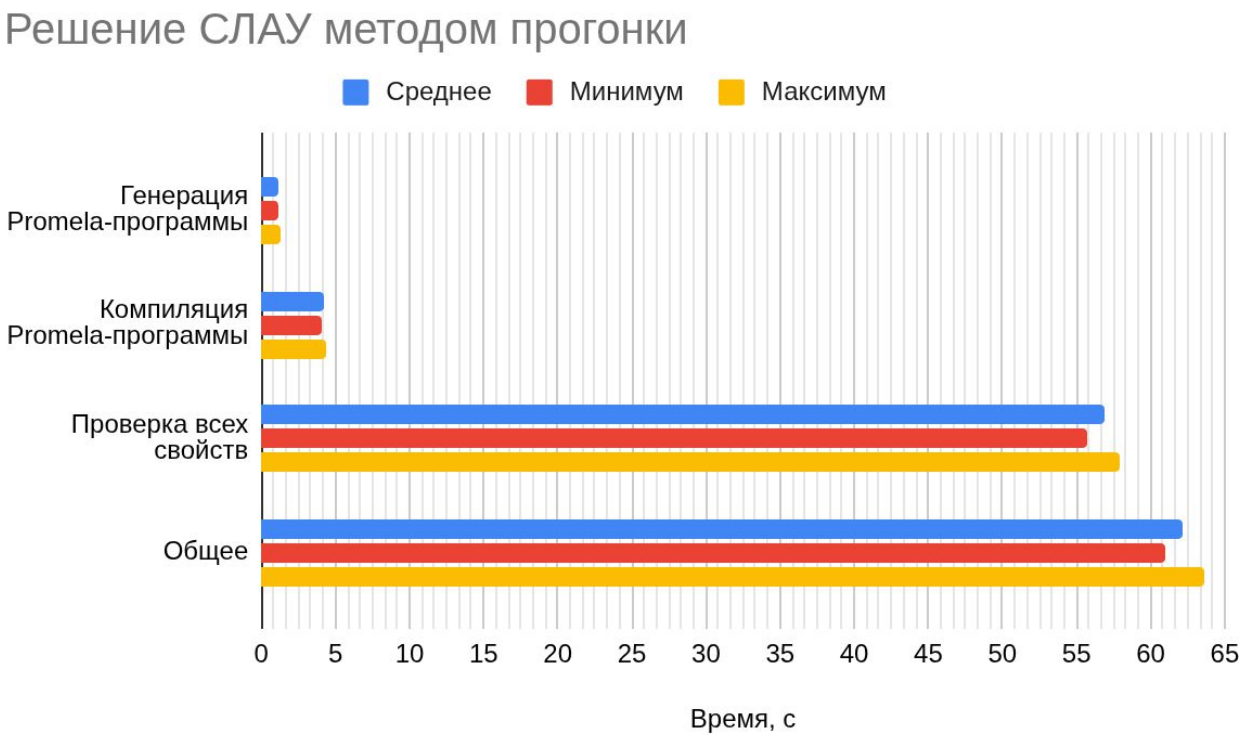
Количество строк в Promela-файле: 219

Оценка накладных расходов на существующих программах



Количество запусков: 100
Строк кода в LuNA-файле: 201
Количество узлов в AST: 1783
Количество строк в Promela-файле: 519

Оценка накладных расходов на существующих программах



Количество запусков: 70
Строк кода в LuNA-файле: 484
Количество узлов в AST: 15069
Количество строк в Promela-файле: 3840

Список публикаций

1. Усенко Н.С., Власенко А.Ю. Верификация на моделях фрагментированных программ в системе LuNA // Современное программирование: сб. ст. по материалам Междунар. науч.-практ. конф. (Нижневартовск, 2025). – Нижневартовск : НВГУ, 2025. – С. 398-406.

2. Усенко Н.С. Применение подхода Model Checking с целью обнаружения ошибок во фрагментированных программах для системы LuNA // Информационные технологии. Научный инжиниринг : материалы 63-й Международной научной студенческой конференции. — Новосибирск, 2025. — Принято к печати.

Результаты работы

1. Выявлен ряд характерных типов ошибок, свойственных LuNA-программам и расширена база типов ошибок.
2. Проведен обзор средств автоматизированной отладки программ.
3. Был спроектирован, реализован и протестирован анализатор LuNA-программ на основе метода верификации на моделях.
4. Произведена интеграция анализатора в комплекс ADAPT.

Спасибо за внимание!

Темпоральные операторы

Синтаксис	Описание	Пример
[]	в каждом состоянии	[](x >= 0)
<>	найдется состояние	<>(x == 1)
!	отрицание формулы	!error
&&	логическое И	(x != 0 && y != 0)
||	логическое ИЛИ	(x != 0 || y != 0)
->	логическая импликация	(x > 0 -> y > 0)

Пример. Повторная инициализация ФД (SEM2*)

Программа на LuNA

```
// Объявления ФК init и print
import c_init_int(int, name) as init;
import c_print_int(int) as print;

sub main() {      // Объявление ФК main
   df x;          // Объявление ФД x
   init(1, x);    // Вызов ФК. Инициализация x
   init(1, x);    // Повторная инициализация x
   print(x);      // Вызов ФК print. Использование x
}
```

Модель на Promela

```
ltl SEM2_var0 {[] (init_count_var0 < 2)}

active proctype main() { // sub main() {
   def(var0);            //   df x;
   init(var0);           //   init(1, x);
   init(var0);           //   init(1, x);
   use(var0);            //   print(x);
   enddef(var0);         // }
}
```

* Полный перечень классов ошибок и их описания приведен на сайте:
https://github.com/LuNA-Static-Analysis/LuNA-Static-Analysis-Repository/wiki/База-ошибок

Пример. Использование ФД после его удаления (SEM3.6)

Программа на LuNA

```
// Объявления ФК init и print
import c_init_int(int, name) as init;
import c_print_int(int) as print;

sub main() {      // Объявление ФК main
    df x;         // Объявление ФД x
    init(1, x);   // Вызов ФК. Инициализация x
    print(x) @ { // Вызов ФК print. Использование x
        delete x; // Очистка x
    };
    print(x);     // Ошибка
}
```

Модель на Promela

```
ltl SEM3_6_var0 {[] (
    destroy_count_var0>0 ->
    use_count_var0<2)
}
active proctype main() { // sub main() {
def(var0);               //   df x;
init(var0);              //    init(1, x);
use(var0);               //    print(x) @ {
destroy(var0);           //      delete x; };
use(var0);               //    print(x);
enddef(var0);            //  }
```

Пример. Поиск циклических зависимостей (SEM3.2)

Программа на LuNA

```
C++ sub int_set(name x, int v) ${{ x = v; $}}

sub main() {
    df x, y;
    int_set(x, y);
    int_set(y, z);
    int_set(z, x); //программа зависает.
}
```

LTL

```
ltl SEM3_2_varN_..._varK {
    [] !(
        depends_on_varN_varM &&
        ...
        depends_on_varK_varN
    )
}
```

Для любого состояния модели верно, что все указанные зависимости не активны одновременно

%var1% ... %varN% – перечислений по порядку переменных, формирующих цикл.

Перспективы

1. Расширение возможностей анализатора для поддержки новых семантических ошибок.
2. Параллельная проверка набора LTL-свойств для ускорения анализа.
3. Генерация отдельных моделей под каждое проверяемое свойство.