

# Статический поиск гонок в программах на языке Си

Студент: Фроловский Алексей  
Вадимович, ИУ7-47

Научный руководитель: Рудаков  
Игорь Владимирович

# Цель и задачи

**Цель:** разработать метод статического поиска гонок в программах на языке Си

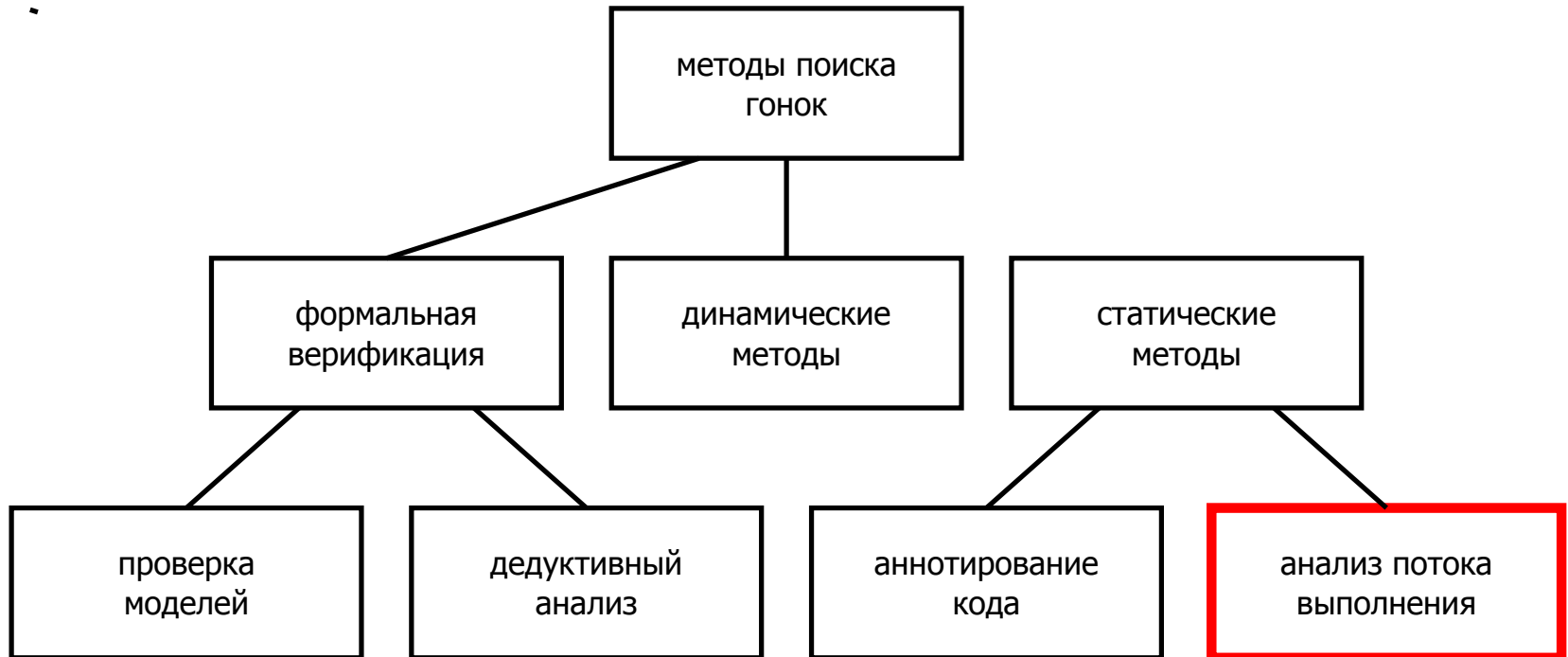
**Задачи:**

- Выполнить анализ методов поиска гонок в программах, выявить их достоинства и недостатки
- Разработать метод статического поиска гонок при доступе к разделяемой памяти
- Разработать алгоритмы, входящие в состав предложенного метода
- Разработать ПО, реализующее предлагаемый метод
- Провести исследование разработанного метода

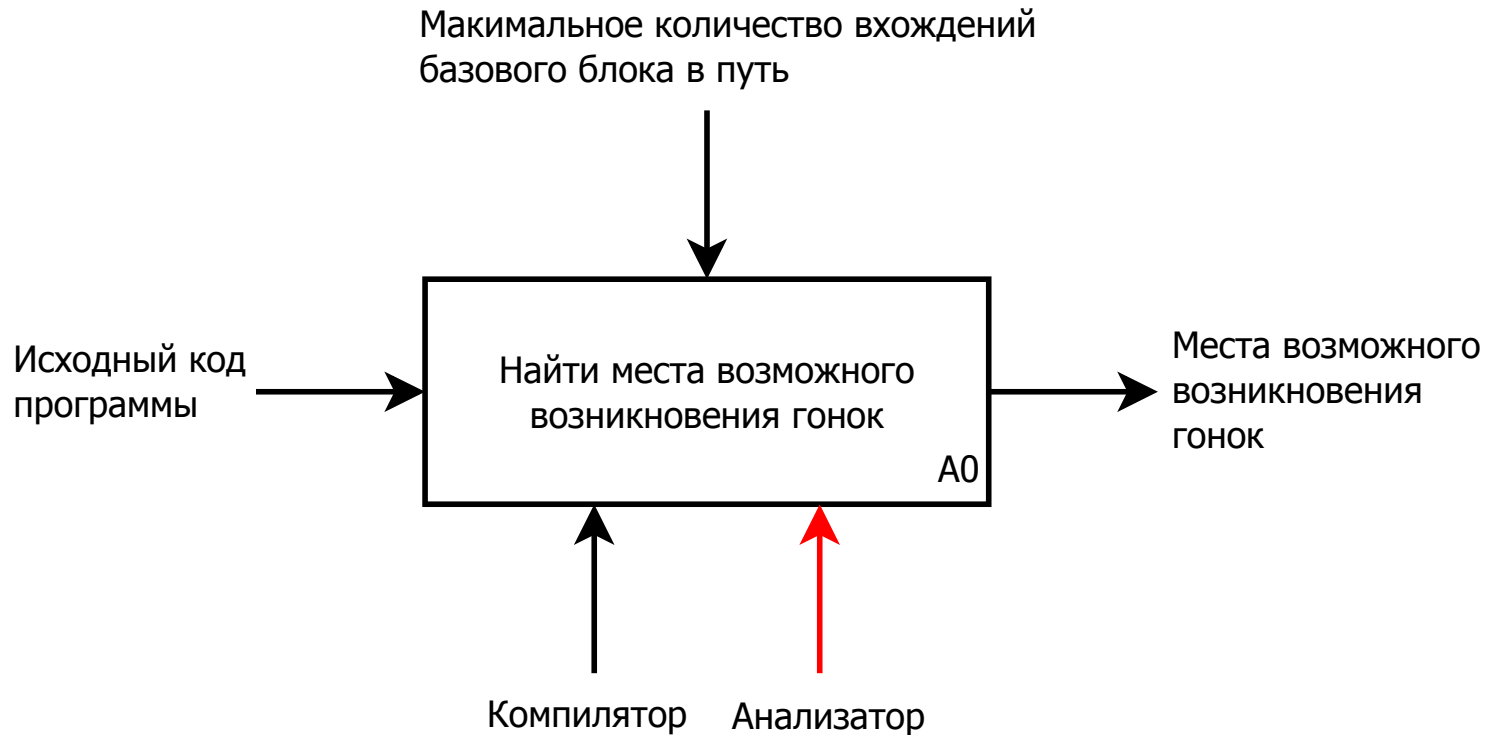
# Понятие гонки

```
static int count = 0;
void *foo(void *arg) {
    ...
    count++; // ВОЗМОЖНО ВОЗНИКНОВЕНИЕ ГОНКИ
    ...
}
int main(int argc, char *argv[]) {
    ...
    pthread_create(&thread1, NULL, &foo, NULL);
    pthread_create(&thread2, NULL, &foo, NULL);
    ...
}
```

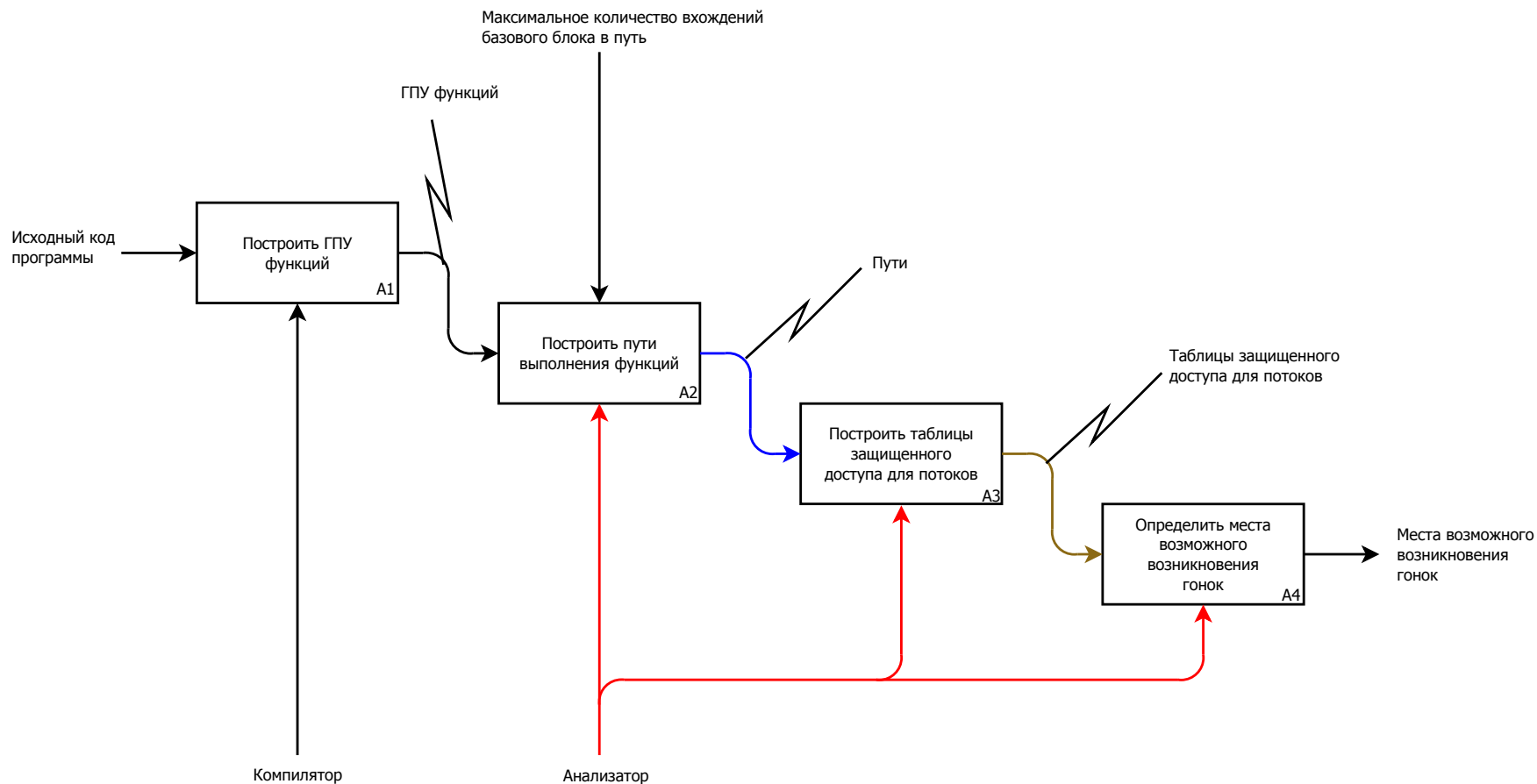
# Методы поиска гонок



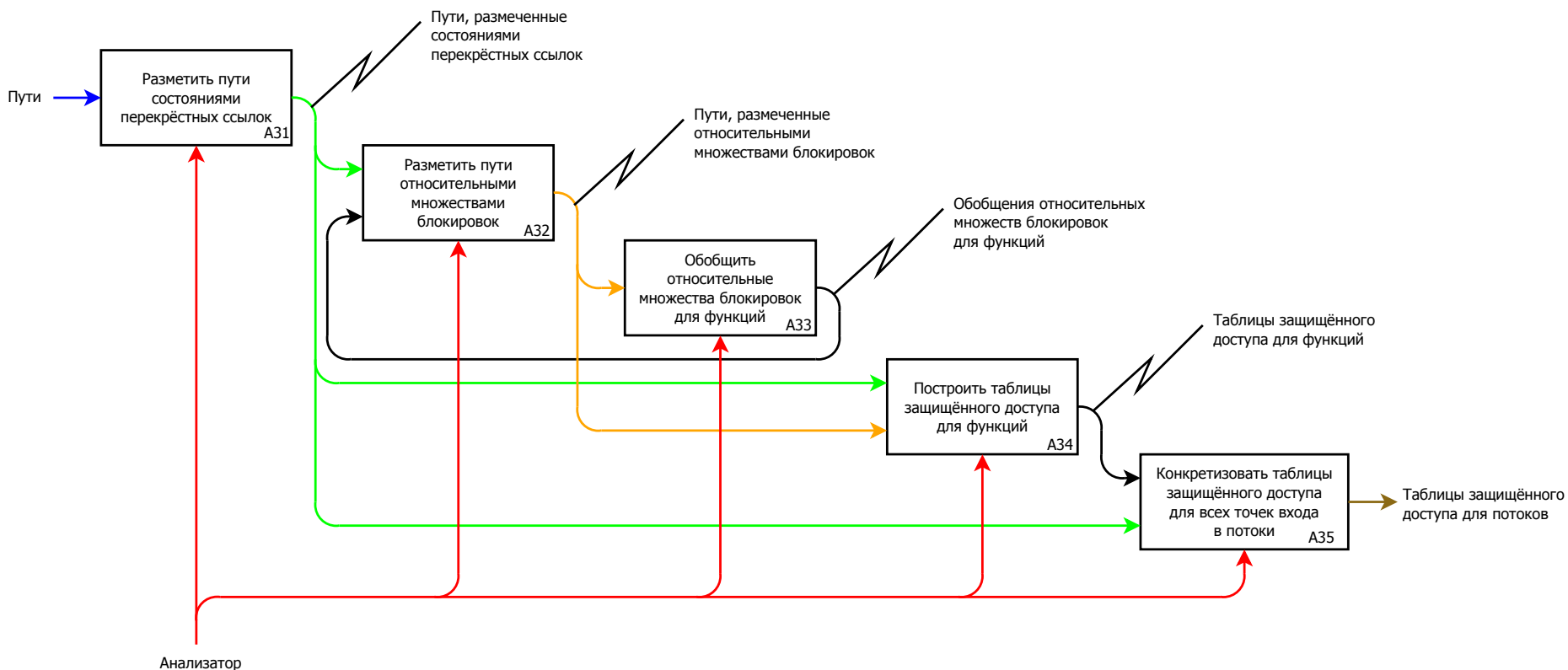
# Метод статического поиска гонок



# Метод статического поиска гонок



# Построение таблиц защищённого доступа для потоков

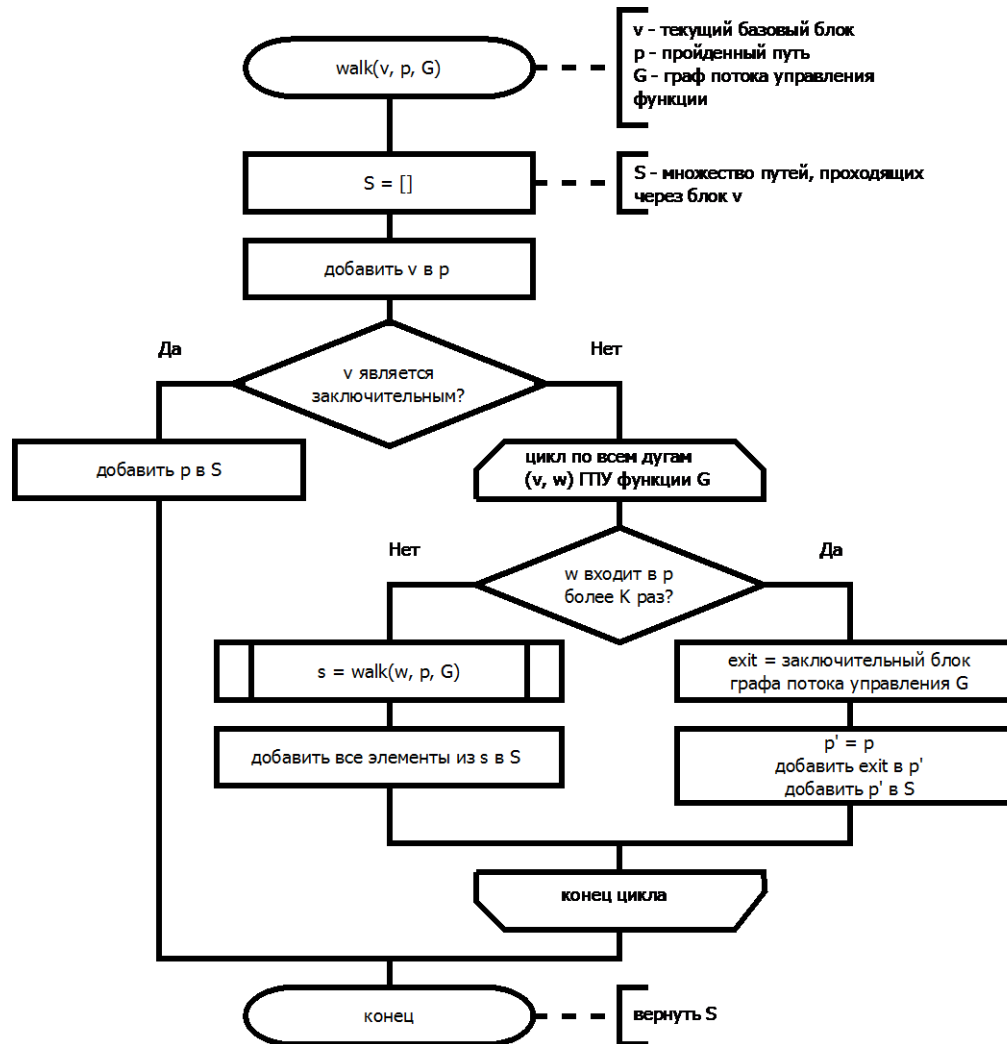


# Ограничения метода

- Отсутствие рекурсивных вызовов функций
- Отсутствие указателей на функции
- Отсутствие обращений к памяти по заранее заданным адресам
- Отсутствие динамического выделения памяти
- Отсутствие арифметики указателей
- Отсутствие обращений к элементам массива

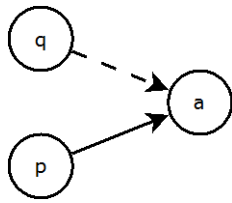
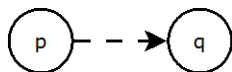
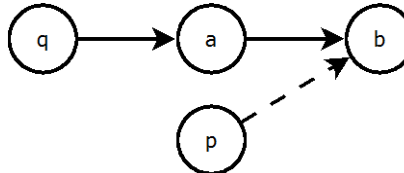


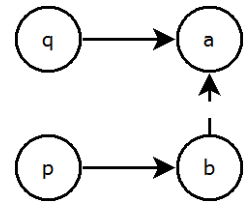
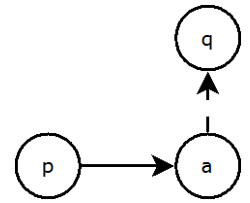
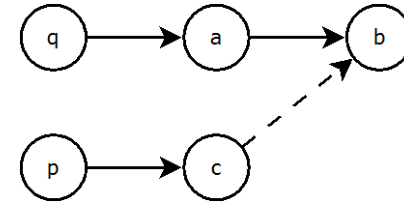
# Построение путей выполнения



# Обновление перекрестных ссылок

## Анализируемые ситуации

<u>Инструкция</u>	<u>Пример</u>
$p = q$	 <p>A diagram showing two pointer nodes, 'q' and 'p', and a target node 'a'. Node 'q' has a dashed arrow pointing to node 'a'. Node 'p' has a solid arrow pointing to node 'a'.</p>
$p = \&q$	 <p>A diagram showing a pointer node 'p' with a dashed arrow pointing to a pointer node 'q'.</p>
$p = *q$	 <p>A diagram showing a sequence of pointers. Node 'q' has a solid arrow pointing to node 'a'. Node 'a' has a solid arrow pointing to node 'b'. Node 'p' has a dashed arrow pointing to node 'b'.</p>

<u>Инструкция</u>	<u>Пример</u>
$*p = q$	 <p>A diagram showing two pointer nodes, 'q' and 'p', and two target nodes, 'a' and 'b'. Node 'q' has a solid arrow pointing to node 'a'. Node 'p' has a solid arrow pointing to node 'b'. Node 'b' has a solid arrow pointing to node 'a'.</p>
$*p = \&q$	 <p>A diagram showing a pointer node 'p' with a solid arrow pointing to a pointer node 'a'. Node 'a' has a solid arrow pointing to a pointer node 'q'.</p>
$*p = *q$	 <p>A diagram showing a sequence of pointers. Node 'q' has a solid arrow pointing to node 'a'. Node 'a' has a solid arrow pointing to node 'b'. Node 'p' has a solid arrow pointing to node 'c'. Node 'c' has a dashed arrow pointing to node 'b'.</p>

# Относительное множество блокировок

**Относительное множество блокировок  $L$**  – пара  $(L_+, L_-)$ , где:

- $L_+$  - множество захваченных блокировок,
- $L_-$  - множество освобожденных блокировок.

$\text{lock\_update}((L_+, L_-), (L_+', L_-')) = ((L_+ \cup L_+') - L_+', (L_- \cup L_-') - L_+')$

где:

- $(L_+, L_-)$  – текущее состояние относительного множества блокировок,
- $(L_+', L_-')$  – конкретизованное относительное множество блокировок для вызываемой функции

$\text{lock\_summary}(L_1, \dots, L_n) = (\bigcap_{i \in \overline{1, n}} L_i[0] - \bigcup_{i \in \overline{1, n}} L_i[1], \bigcup_{i \in \overline{1, n}} L_i[1])$

где:

- $L_i$  – относительное множество блокировок, полученное на  $i$ -м пути,
- $n$  – количество анализируемых путей выполнения функции

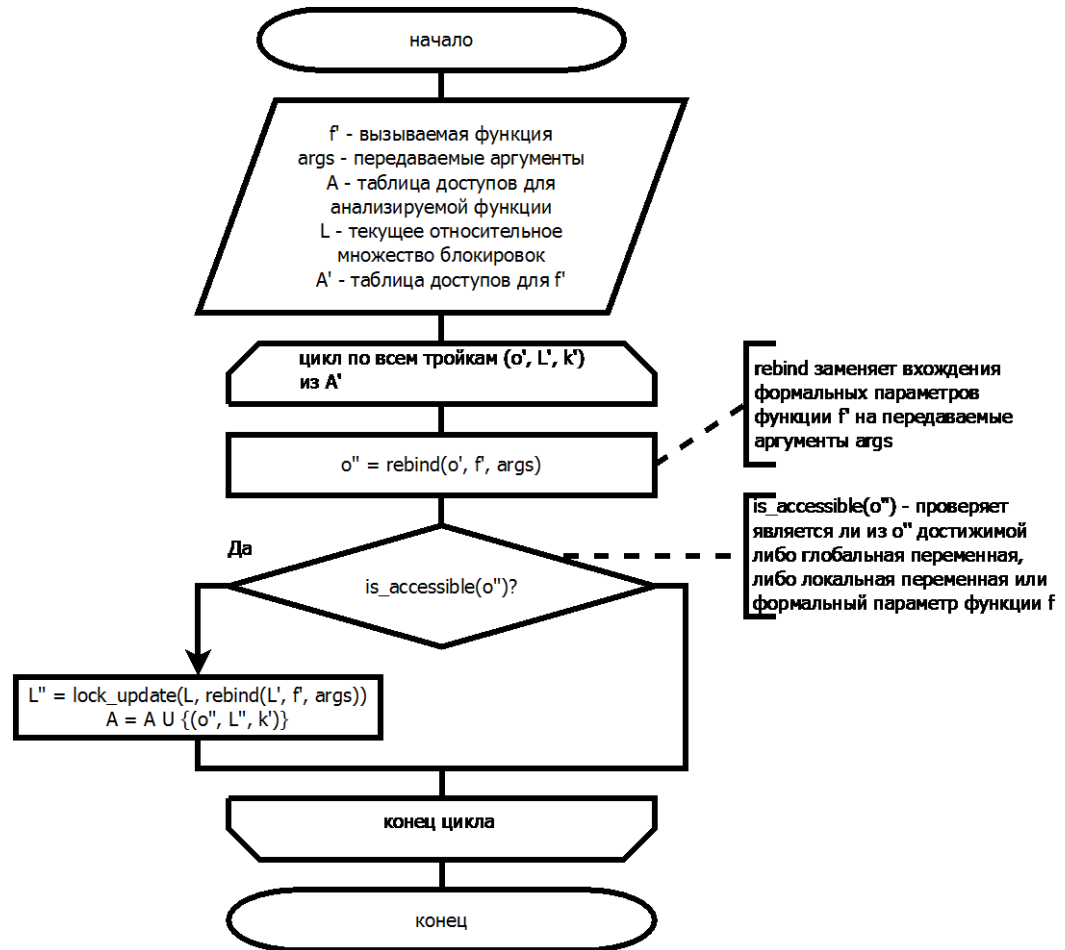
$\text{lock\_summary}(\text{lock}(l)) = (\{l\}, \{\})$     $\text{lock\_summary}(\text{unlock}(l)) = (\{\}, \{l\})$

# Таблица защищенного доступа

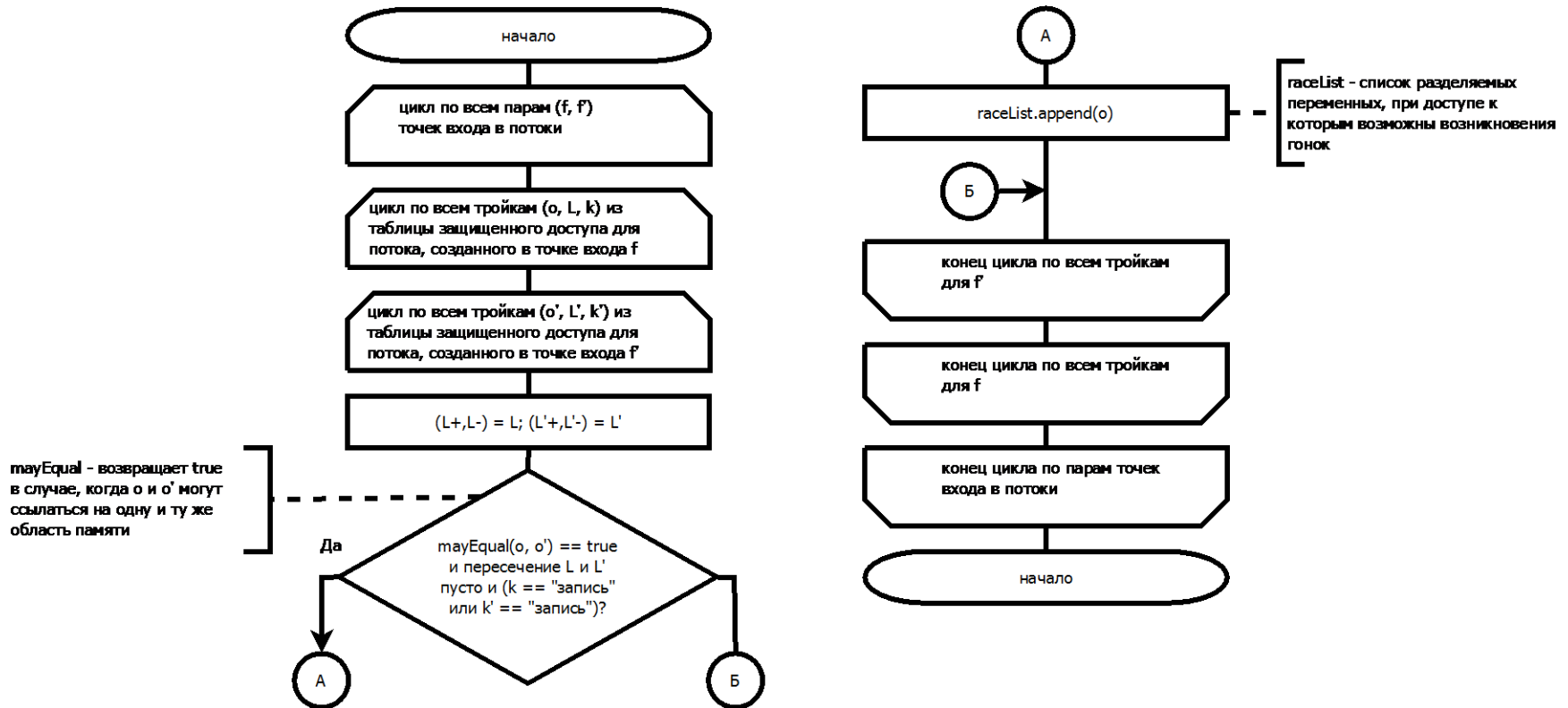
## Защищенный доступ $A$

– тройка  $(o, L, k)$ , где:

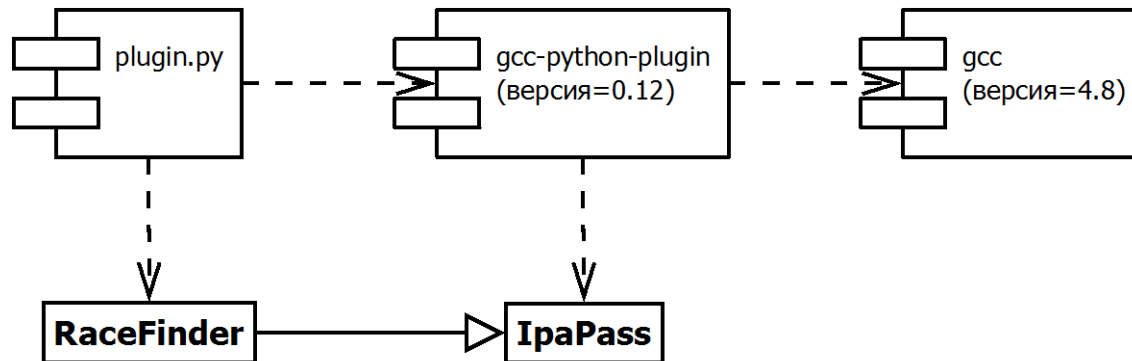
- $o$  – область, к которой производится доступ,
- $L$  – относительное множество блокировок на момент доступа,
- $k$  – вид доступа: “чтение”, “запись”.



# Определение мест возможного возникновения гонок



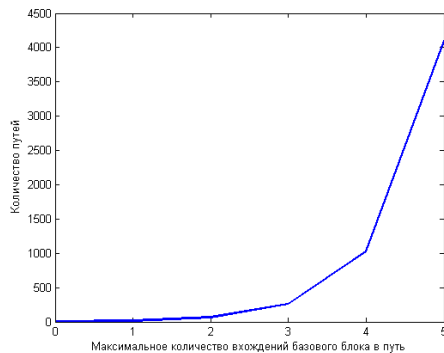
# Структура ПО. Ограничения реализации



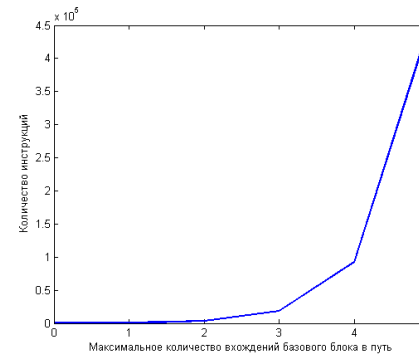
- Использование POSIX API для работы с потоками и объектами взаимного исключения
- Отсутствие обращений к полям структур
- Уникальность имён переменных в пределах функции

# Результаты исследований

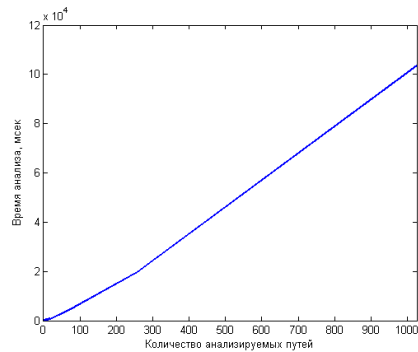
**Зависимость количества анализируемых путей от максимального количества вхождений базового блока в путь**



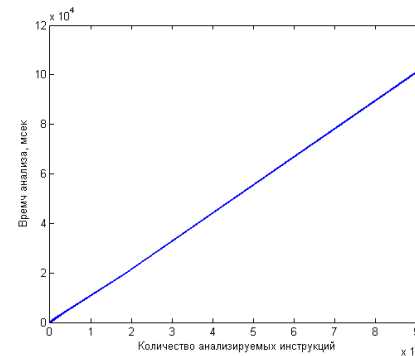
**Зависимость количества анализируемых инструкций от максимального количества вхождений базового блока в путь**



**Зависимость времени анализа от количества анализируемых путей**

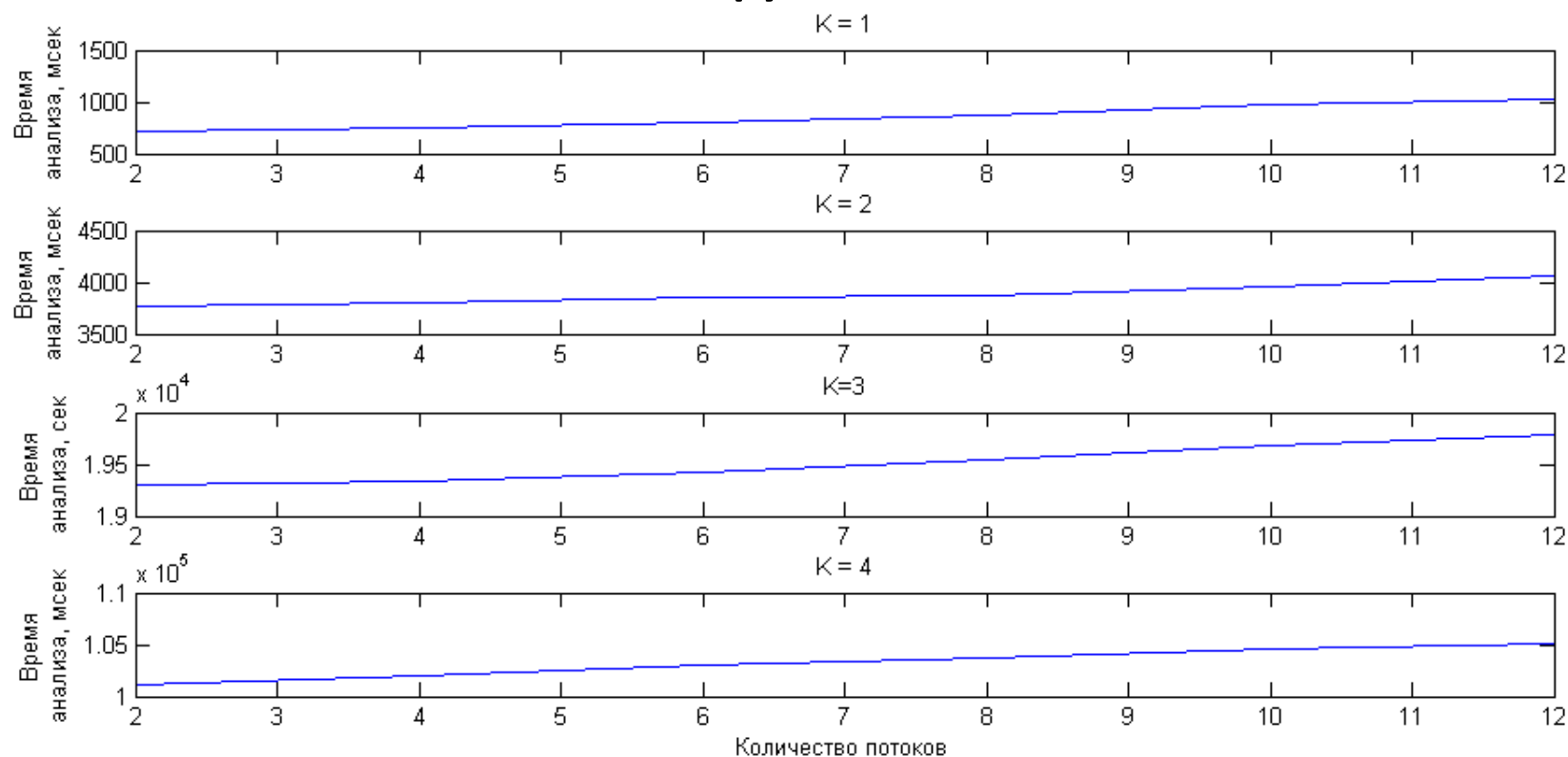


**Зависимость времени анализа от количества анализируемых инструкций**



# Результаты исследований

## Зависимость времени анализа от количества анализируемых потоков



K – максимальное количество вхождений базового блока в путь



# Заключение

- Проведен анализ существующих методов поиска гонок, выявлены их достоинства и недостатки.
- Разработан метод статического поиска гонок на основе относительных множеств блокировок.
- Предложены алгоритмы, входящие в состав разработанного метода.
- Предложенные алгоритмы реализованы в виде загружаемого модуля к компилятору gcc.
- Проведено исследование с использованием разработанного ПО.