САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Методы анализа и обеспечения качества ПО

Михаил Моисеев

Обнаружения нефункциональных ошибок в многопоточных программах методами статического анализа

Санкт-Петербург
2013

Проверочная работа 2

1. Перечислите виды нефункциональных ошибок в программах на языке Java.
2. Для чего требуется строить модель программы перед проведением анализа?
3. В чем заключается основное преимущество анализа программы на основе состояний по сравнению с анализом на основе шаблонов?
4. Какие типы конструкций рассматривает интервальный анализ?
5. Для чего необходим объект $o_{invalid}$ в алгоритме анализа указателей?
6. Какая информация требуется для обнаружения выхода за границы объекта?
7. В чем заключается проблема обнаружения множественных ошибок в программе?
8. Имеется ли связь между точностью и полнотой обнаружения ошибок? Если да, то в чем она выражается?

Ошибки синхронизации

- Ошибки взаимной блокировки потоков/процессов (Deadlock)
- Ошибки конкурентной модификации данных из разных потоков/процессов программы (Data races)
- Недетерминированное выполнение конструкций синхронизации
- Некорректное использование функций синхронизации (Blocking call misuse)
- Отсутствие прогресса в выполнении программы (Livelock)
- Отсутствие достаточных ресурсов для выполнения (Starvation)

Пример ошибки типа Deadlock

```
pthread_mutex_t m1;
pthread_mutex_t m2;

void f() {
  pthread_mutex_lock(&m1);
  pthread_mutex_lock(&m2);
  …
}

int main() {
  …
  pthread_create(&t,NULL,f,NULL);
  pthread_mutex_lock(&m2);
  pthread_mutex_lock(&m1);
  …
}
```

Пример ошибки типа Deadlock

```
pthread_mutex_t m;

void f() {
  pthread_mutex_lock(&m);
  …
  pthread_mutex_unlock(&m);
}

int main() {
  …
  pthread_create(&t,NULL,f,NULL);
  pthread_mutex_lock(&m);
  …
  pthread_join(t, NULL);
  pthread_mutex_unlock(&m);
  …
}
```

Пример ошибки типа Data race

```
int a, b;

void f() {
  a = 1;
}

int main() {
  …
  pthread_create(&t,NULL,f,NULL);
  …
  a = 2;
  …
  pthread_join(t, NULL);
  b = a;        // b = 1 или 2
  …
}
```

Пример ошибки типа Data race

```
int a, b;

void f() {
  a = 1;
}

int main() {
  a = 0;
  pthread_create(&t,NULL,f,NULL);
  …
  b = a;        // b = 0 или 1
  …
}
```

Пример недетерминированной синхронизации

```
void f() {
  sem_wait(&s);
  …
}

void g() {
  sem_wait(&s);
  …
}

int main() {
  …
  pthread_create(&t1,NULL,f,NULL);
  pthread_create(&t2,NULL,g,NULL);
  …
  sem_post(&s);
  …
}
```

Примеры ошибок Blocking call misuse

- Отсутствие освобождения мьютекса при завершении потока программы
- Повторная блокировка мьютекса
- Попытка освобождения незахваченного мьютекса

Ошибки типа Livelock и Starvation

- **Livelock** – отсутствие прогресса в выполняющемся потоке. Может возникать, когда потоки запланированы, но не выполняют никаких операций, поскольку постоянно реагируют на изменение состояния другого потока
- **Starvation** – остановка работы одного или нескольких потоков многопоточного приложения на неопределенное или бесконечное время. Причина остановки – отсутствие процессорного времени, для выполнения потока

Проблемы анализа многопоточных программы

- Переключение контекста между потоками / очередность выполнения конструкций в параллельно выполняющихся потоках
  - при анализе необходимо учитывать возможные последовательности выполнения конструкций различных потоков
- Совместная работа с разделяемыми объектами программы
  - при анализе необходимо определять актуальные значения разделяемых объектов с учетом их модификации в разных потоках

Статический анализ многопоточных программ

- Применение алгоритмов статического анализа, не учитывающих специфику многопоточных программ, приводит к снижению полноты и точности результатов
- Методы статического анализа параллельных (многопоточных) программ
  - анализ частичных порядков
  - анализ количества конструкций синхронизации (Lockset-анализ)
  - подходы на основе извлечения инвариантов
  - подходы на основе преобразования к последовательной программе
  - комбинирование алгоритмов анализа

Подходы на основе анализа частичных порядков

- Проблема анализа многопоточных программ – определение последовательностей выполнения операторов в разных потоках
- Частичные порядки выполнения – последовательности выполнения операторов, которые не зависят от недетерминизма выполнения программы
- Информация о частичных порядках вносится в модель анализируемой программы – определяются точки переключения контекста, в такие точки добавляются специальные конструкции ψ-функции[1]
- В ψ-функциях выполняется распространяется информация, полученная при анализе отдельных потоков

1. Lee J., Midkift S., Padua D. Concurrent static single assignment form and constant propagation for explicitly parallel programs – Springer, 1998

Lockset-анализ

- Для определения частичных порядков необходимо анализировать конструкции управления потоками и конструкции синхронизации
- Алгоритмы Lockset-анализа определяют множество (число, порядок) выполненных конструкций синхронизации для разных объектов на всех путях выполнения в потоках программы[1]
- По результатам Lockset-анализа могут решаться и другие задачи: определение совместной достижимости точек в потоках программы, обнаружение ошибок синхронизации[2]

1. Bouajjani A., Esparza J., Touili T. A Generic Approach to the Static Analysis of Concurrent Programs with Procedures // SIGPLAN – ACM, 2003
2. Engler D., Ashcraft K. RacerX: Effective, Static Detection of Race Conditions and Deadlocks // OSP – ACM, 2003

Подходы на основе извлечения инвариантов

- Извлекаемые инварианты содержат информацию о состояниях отдельных объектов программы, зависимостях между объектами программы, а также связях между состояниями объектов и входными данными или результатами работы программы[1,2]
- Конкретные инварианты выполняются для некоторых частей или всей программы
- Для обнаружения Race Condition можно использовать следующий инвариант
  - для каждого разделяемого объекта определяется объект синхронизации, под защитой которого выполняются все операции

1. Pratikakis P., Foster J., Hicks M. LOCKSMITH: Practical Static Race Detection for C – ACM, 2011
2. Terauchi T. Checking Race Freedom via Linear Programming // PLDI – ACM,2008

Подходы на основе преобразования к последовательной программе

- Выполняется преобразование многопоточной программы к последовательному виду и дальнейший анализ полученной программы с помощью известных алгоритмов
- Для преобразования к последовательному виду предлагается заменять анализ параллельных потоков на анализ путей в последовательной программе – в последовательную программу вносятся искусственные конструкции, эмулирующие недетерминизм выполнения потоков[1]
- Для учета изменений разделяемой памяти для каждого разделяемого объекта создается несколько копий, значения которых представляют различные варианты переключения контекста[2]

1. Qadeer S., Wu D. KISS: Keep It Simple and Sequential // PLDI – ACM, 2004
2. Lahiri S., Qadeer S., Rakamaric Z. Static and Precise Detection of Concurrency Errors in Systems Code Using SMT Solvers – Springer, 2009

Комбинирование алгоритмов анализа

- Совместное применение нескольких известных алгоритмов анализа[1]
  - идентификация потоков и объектов синхронизации в разных потоках программы (Points-to)
  - определение разделяемых объектов программы (Escape)
  - анализ параллельно выполняющихся конструкций из разных потоков программы (May-happen-in-parallel)
  - определение объектов синхронизации, защищающих доступ к разделяемым объектам программы
- Используемые алгоритмы применяются итеративно, с уточнением результатов на каждой итерации

1. Naik M., Park C., Sen K., Gay D. Effective Static Deadlock Detection // Conference on Software Engineering – IEEE Computer Society , 2009

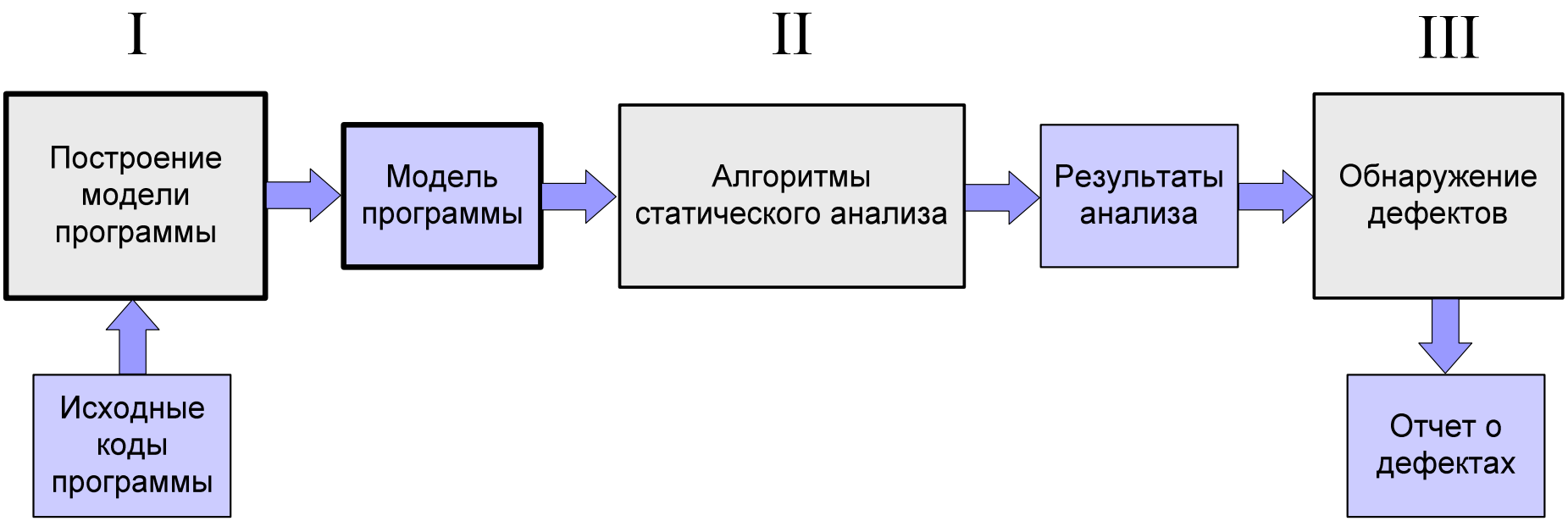
Средства статического анализа C/C++

- Промышленные средства для обнаружения ошибок в многопоточных программах на языках C/C++
  - Coverity Prevent SA
  - Klocwork Insight
  - Parasoft C++ test
  - Mathworks PolySpace
  - IBM RSA
  - FramaC
- Экспериментальные средства анализа многопоточных программ
  - RacerX (Stanford University)
  - RADAR (University of California)
  - KISS (Microsoft Research)
  - STORM (Microsoft Research)
  - LOCKSMITH (University of Maryland)
  - Loopfrog (Oxford University)
  - PREfix (Microsoft Research)
  - VeriSoft (Bell Laboratories)

Недостатки существующих методов и средств

- Основные недостатки существующих методов
  - ограничения на количество потоков и объектов синхронизации
  - отсутствие идентификации объектов синхронизации
  - ограничение на типы обнаруживаемых дефектов
  - отсутствие учета взаимных влияний между алгоритмами анализа
  - применение аппроксимаций существенно снижающих полноту и точность
- Основные недостатки существующих средств
  - недостаточная полнота и точность обнаружения
  - поддерживается ограниченное подмножество языка C и конструкций синхронизации

Обнаружение программных дефектов на основе статического анализа



Способы создания многопоточных программ

- Существуют различные стандарты и библиотеки для создания многопоточных программ на языках C/C++
  - OpenMP, POSIX, Intel TBB, Boost Threads, Win API, …
- **POSIX** (Portable Operating System Interface) – включает интерфейсы для создания, управления и синхронизации потоков
  - стандарт IEEE 1003, ISO/IEC 9945
  - ОС Linux, FreeBSD, Solaris, QNX, MS Windows
  - все основные типы объектов синхронизации

Потоки и объекты синхронизации Pthreads

- Потоки программы (pthread_t)
- Объекты синхронизации
  - мьютекс (pthread_mutex_t)
  - семафор (sem_t)
  - rwlock (pthread_rwlock_t)
  - барьер (pthread_barrier_t)
  - spin (pthread_spinlock_t)
  - ключ данных отдельных потоков (pthread_key_t)
  - переменная состояния (pthread_cond_t)
  - переменная однократного выполнения (pthread_once_t)
- Рассматриваются потоки программы и объекты синхронизации **мьютекс** и **семафор**

Базис конструкций управления параллельным выполнением программы

Конструкция	Краткое описание
`create(t, pf)`	Создание потока программы
`join(t)`	Ожидание завершения потока программы
`init(m, value)`	Инициализация объекта синхронизации
`destroy(m)`	Уничтожение объекта синхронизации
`lock(m, action)`	Блокировка мьютекса
`unlock(m, action)`	Освобождение мьютекса
`wait(m, action)`	Блокировка семафора
`post(m, action)`	Освобождение семафора
`state(m)`	Получение состояния объекта синхронизации

Представление функций для мьютекса

Исходная функция	Представление в базисе
`pthread_mutex_init(&m, &attr)`	`init(m, 1)`
`pthread_mutex_destroy(&m)`	`destroy(m)`
`pthread_mutex_unlock(&m)`	`unlock(m, 1)`
`pthread_mutex_lock(&m)`	`lock(m, 0)`

`pthread_mutex_trylock(&m)`	`if (state(m) == 1) {`
	`  lock(m);`
	`  return true;`
	`} else {`
	`  return false;`
	`}`

Представление функций для семафора

Исходная функция	Представление в базисе
`sem_init(&m, val, shared)`	`init(m, val)`
`sem_destroy(&m)`	`destroy(m)`
`sem_post(&m)`	`post(m, 1)`
`sem_wait(&m)`	`wait(m, -1)`
`sem_getvalue(&m, &result)`	`result = state(m)`

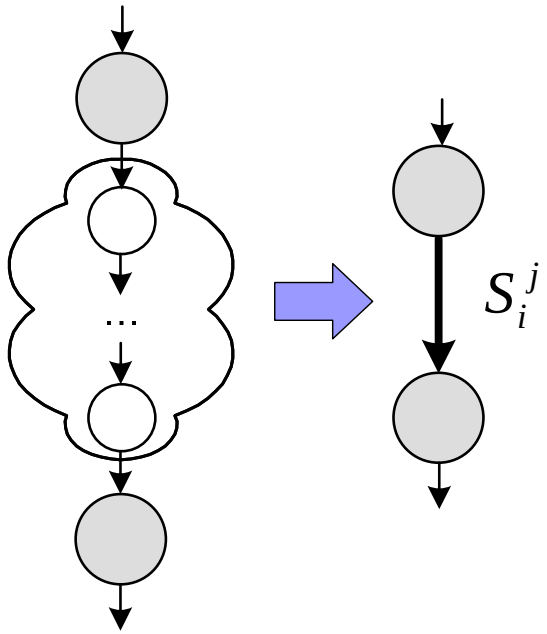
`sem_trywait(&m)`	`if (state(m) > 0) {`
	`  wait(m, -1);`
	`  return true;`
	`} else {`
	`  return false;`
	`}`

Модель программы

- Программа преобразуется к SSA-форме (Static Single Assignment form)
  - каждой переменной значение присваивается не более одного раза
  - вводятся конструкции слияния путей выполнения (***Φ*-функции**)
  - циклы заменяются конструкциями ветвления и безусловных переходов
  - сложные выражения разбиваются на несколько выражений в трёхоперандной форме
- Модель программы – CFG (Control Flow Graph), расширенный
  - конструкциями управления областью видимости переменных
  - информацией о разметке циклов
- Вызовы функций управления потоками и функции синхронизации заменяются на их представления в разработанном базисе

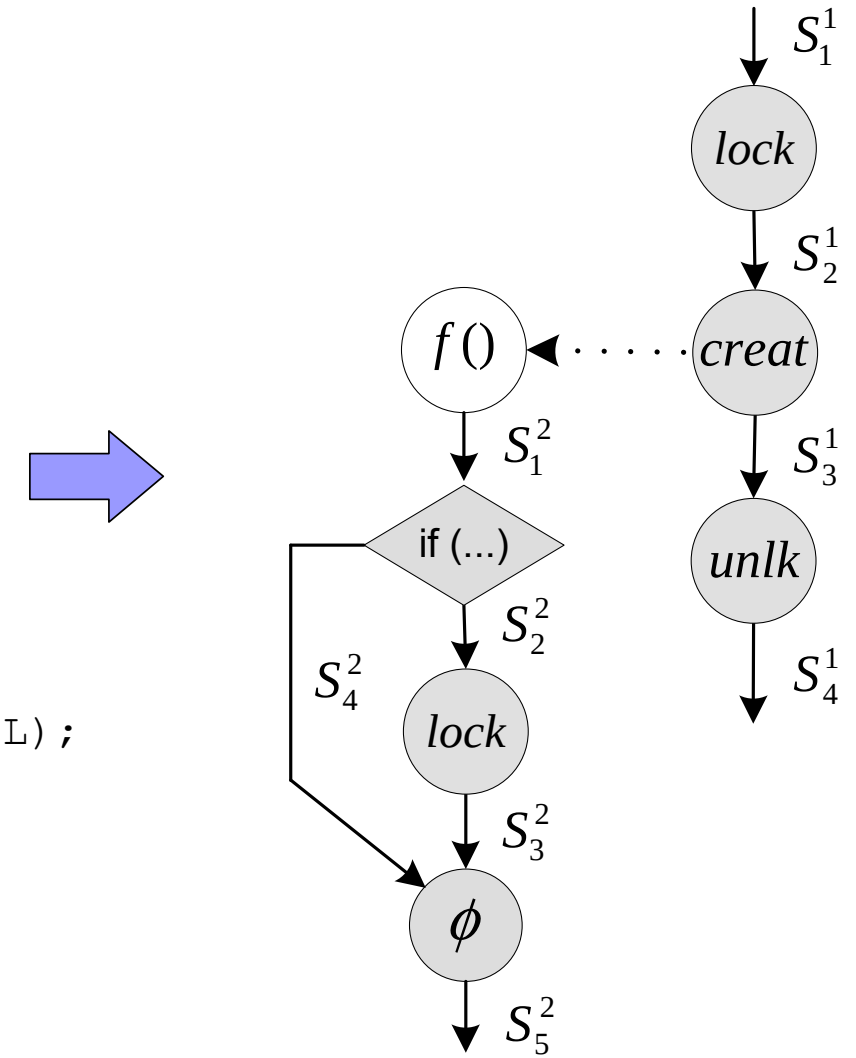
Блоки программы

- Конструкции программы объединяются в блоки
- **Блок программы** – набор последовательных конструкций, не содержащий конструкций ветвления и *Ф*-функций, а также конструкций управления параллельным выполнением программы
- Блок S_i^j, j – номер потока, i – номер блока в потоке

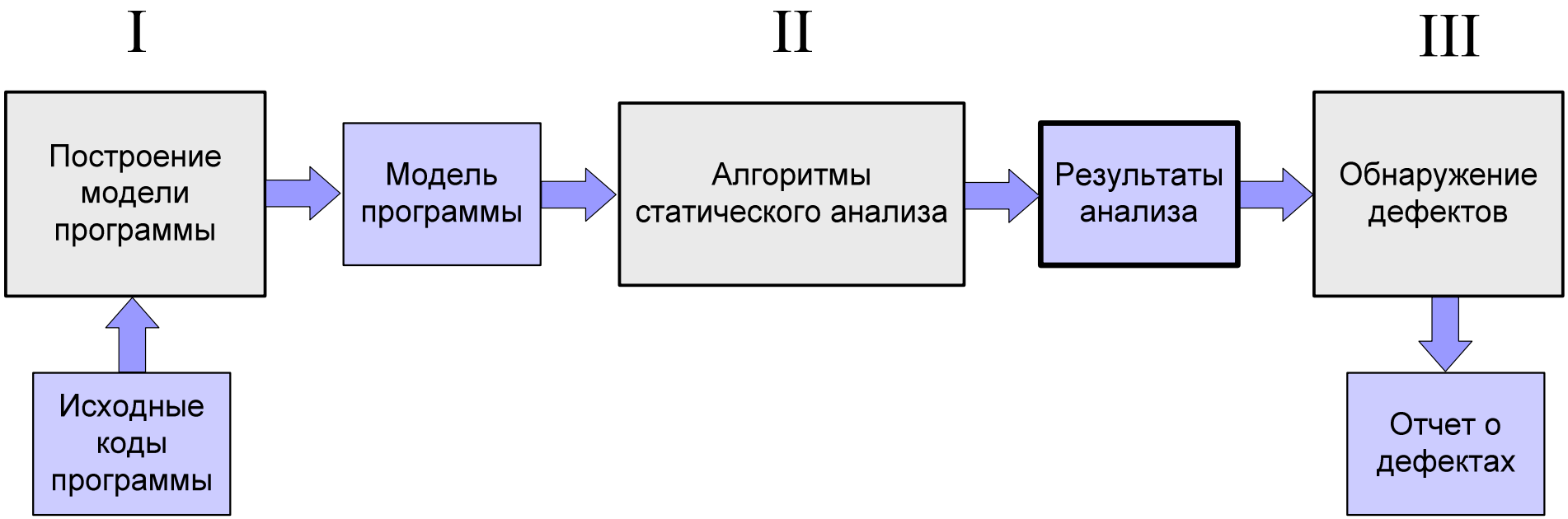


Пример построения модели программы

```
#include <pthread.h>

void f() {
  if (…) {
     pthread_mutex_lock(&m);
  }
  …
}

int main() {
  …
  pthread_mutex_lock(&m);
  pthread_create(&t,NULL,f,NULL);
  …
  pthread_mutex_unlock(&m);
  …
}
```

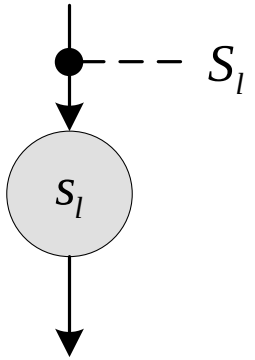


Обнаружение программных дефектов на основе статического анализа



Представление результатов анализа

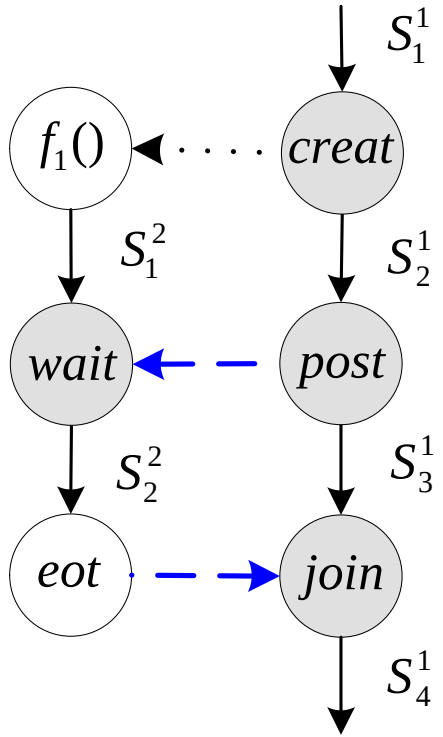
- Состояние программы – значения всех видимых объектов
  - значения объектов-указателей
  - значения интервальных объектов
  - состояния ресурсов
- Информация о параллельном выполнении программы
  - отношения синхронизации
  - отношения параллельности
  - состояния объектов синхронизации



$$S_l = \begin{Bmatrix} (p, (o_{j_1}, k_1)), \ldots \\ (a, (i_1^{\min}, i_1^{\max})), \ldots \\ (d, r_{j_1}), \ldots \end{Bmatrix}$$

Отношение синхронизации

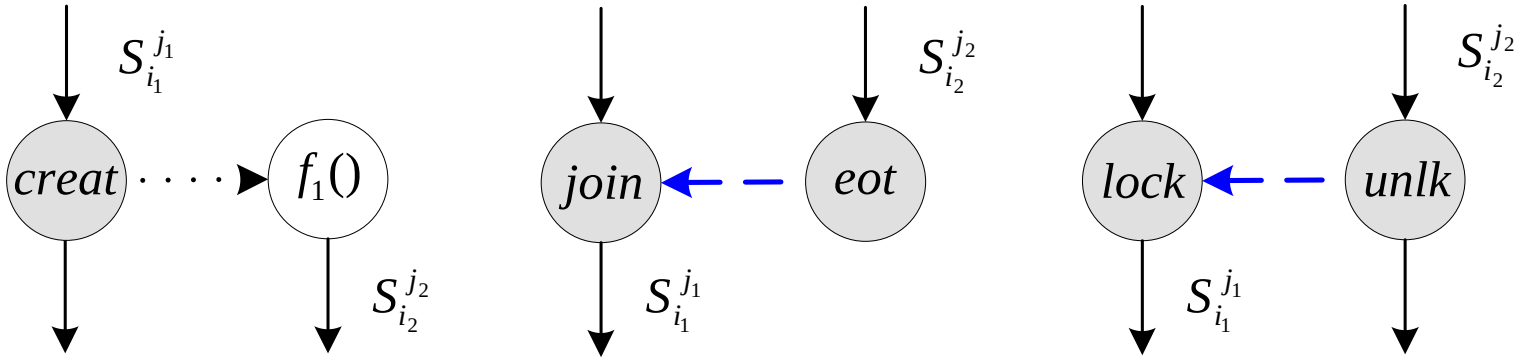
- Для блока $S_{i_1}^{j_1}$, непосредственно перед которым имеется конструкция синхронизации s_1, и блока $S_{i_2}^{j_2}$, непосредственно после которого имеется конструкция синхронизации s_2, имеет место **отношение синхронизации** если существует набор входных данных, на котором выполнение s_1 обеспечивает выполнение s_2 – $S_{i_2}^{j_2} \rightarrow S_{i_1}^{j_1}$



$S_2^1 \rightarrow S_2^2, S_2^2 \rightarrow S_4^1$

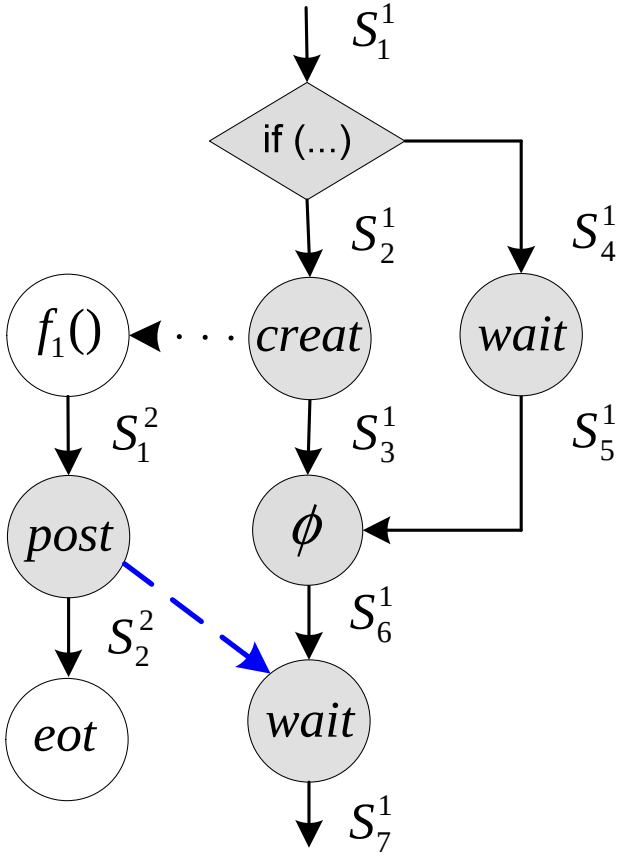
Строгая последовательность выполнения блоков программы

- Между блоками $S_{i_1}^{j_1}$ и $S_{i_2}^{j_2}$ существует строгая последовательность выполнения, если на всех наборах входных данных программы сначала выполняется блок $S_{i_1}^{j_1}$, а затем блок $S_{i_2}^{j_2}$ или наоборот



Отношение параллельности

- Два блока являются совместными если существует набор входных данных, на котором могут выполняться оба этих блока
- Для блоков $S_{i_1}^{j_1}, S_{i_2}^{j_2}$, выполняющихся в разных потоках программы, имеет место **отношение параллельности**, если эти блоки являются совместными и между ними отсутствует строгая последовательность выполнения – $S_{i_1}^{j_1} \,||\, S_{i_2}^{j_2}$



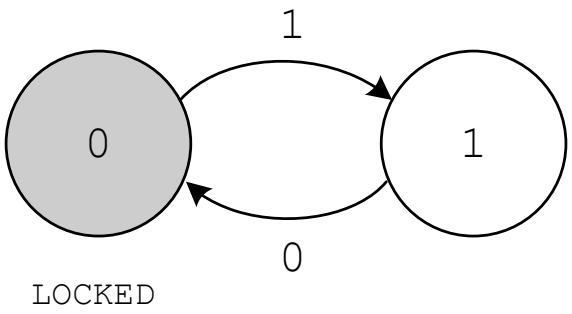
$S_1^2 \,||\, S_3^1, S_1^2 \,||\, S_6^1$

$S_1^2 \not|| S_4^1, S_1^2 \not|| S_7^1$

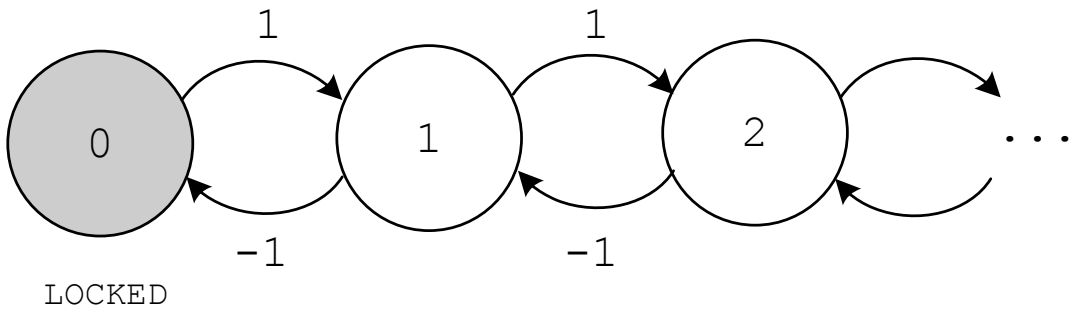
Состояние объекта синхронизации

- **Состояние объекта синхронизации** в блоке программы – множество возможных значений $m\left(S_i^j\right)=\{m_1,\ldots,m_n\}$
- **Множество действий над объектом синхронизации** в блоке программы – действия над этим объектом на всех путях от первой конструкции потока до этого блока $A\left(m,S_i^j\right)=\{a_1,\ldots,a_n\}$

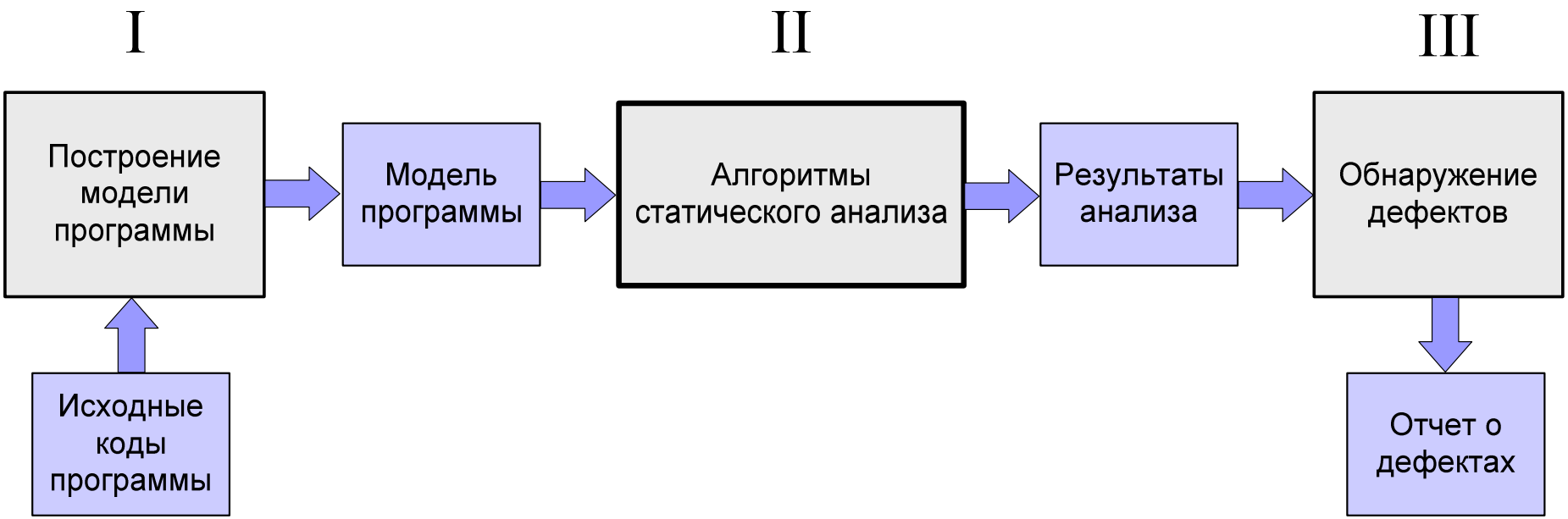
Мьютекс



Семафор

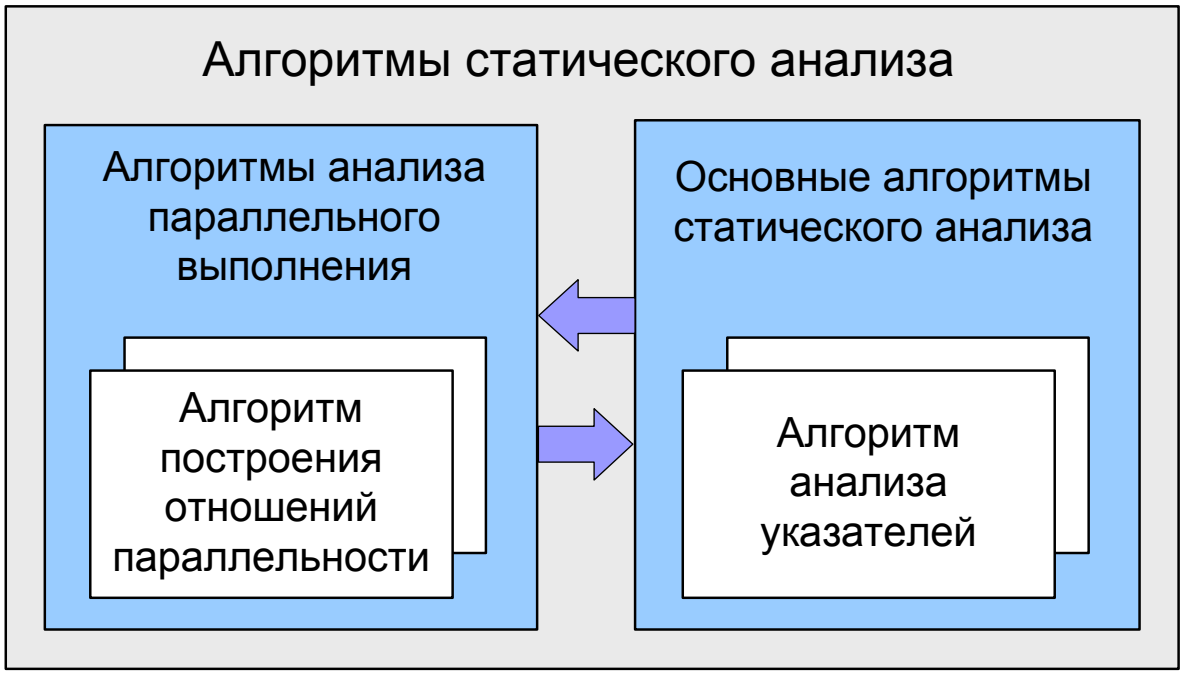


Обнаружение программных дефектов на основе статического анализа



Основная идея предлагаемого подхода

- Расширение алгоритмов статического анализа последовательных программ на многопоточные программы с помощью разработанных методов анализа параллельного выполнения

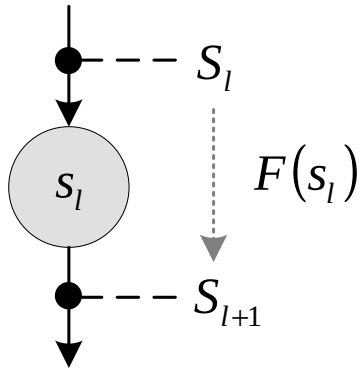


Задачи алгоритмов анализа

- Основные алгоритмы статического анализа
  - определение значений указателей на функции
  - идентификация потоков и объектов синхронизации
  - извлечение информации для обнаружения дефектов
- Алгоритмы анализа параллельного выполнения программы
  - анализ создаваемых потоков и запускаемых в них функций
  - определение отношений параллельности, отношений синхронизации и состояний объектов синхронизации
  - управление последовательностью анализа конструкций разных потоков
  - распространение результатов анализа между потоками программы
  - извлечение информации для обнаружения ошибок синхронизации

Совместное выполнения алгоритмов

- Все используемые алгоритмы анализа представляются с помощью правил для конструкций модели программы
  - правила основных алгоритмов анализа определяют состояния программы
  - правила методов анализа параллельного выполнения определяют отношения параллельности, отношения синхронизации и состояния объектов синхронизации



- Правила всех алгоритмов применяются одновременно
- Обеспечивается обмен промежуточными результатами и учет всех взаимодействий между алгоритмами анализа

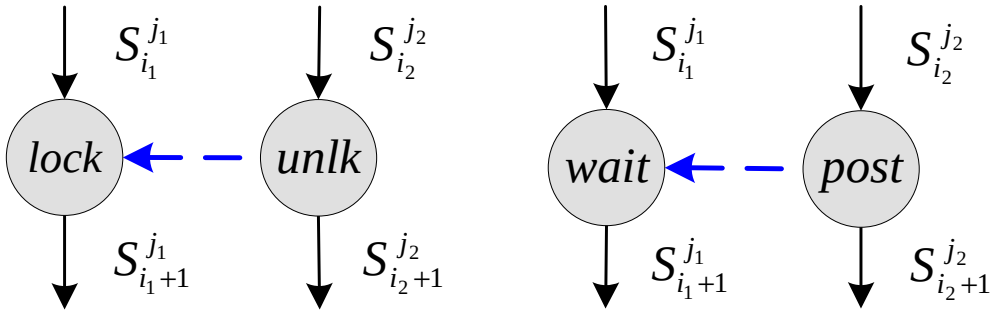
Алгоритм определения действий над объектами синхронизации

$$[lock(m, act)]: A\left(m, S_{i_1+1}^{j_1}\right) = \{0\}$$

$$[unlock(m, act)]: A\left(m, S_{i_2+1}^{j_2}\right) = \{1\}$$

$$[wait(m, act)]: A\left(m, S_{i_1+1}^{j_1}\right) = A\left(m, S_{i_1}^{j_1}\right) \oplus \{-1\}$$

$$[post(m, act)]: A\left(m, S_{i_2+1}^{j_2}\right) = A\left(m, S_{i_2}^{j_2}\right) \oplus \{1\}$$



Операции сложения и умножения множеств действий

$$A\left(m, S_{i_3}^{j_3}\right) = A\left(m, S_{i_1}^{j_1}\right) \oplus A\left(m, S_{i_2}^{j_2}\right) \Leftrightarrow$$
$$A\left(m, S_{i_3}^{j_3}\right) = \left\{a_{k_1} + a_{k_2}\right\}, \forall a_{k_1}, a_{k_2} : a_{k_1} \in A\left(m, S_{i_1}^{j_1}\right), a_{k_2} \in A\left(m, S_{i_2}^{j_2}\right).$$

$$A\left(m, S_{i_3}^{j_3}\right) = A\left(m, S_{i_1}^{j_1}\right) \otimes A\left(m, S_{i_2}^{j_2}\right) \Leftrightarrow$$
$$A\left(m, S_{i_3}^{j_3}\right) = \left\{a_{k_1} \cdot a_{k_2}\right\}, \forall a_{k_1}, a_{k_2} : a_{k_1} \in A\left(m, S_{i_1}^{j_1}\right), a_{k_2} \in A\left(m, S_{i_2}^{j_2}\right).$$

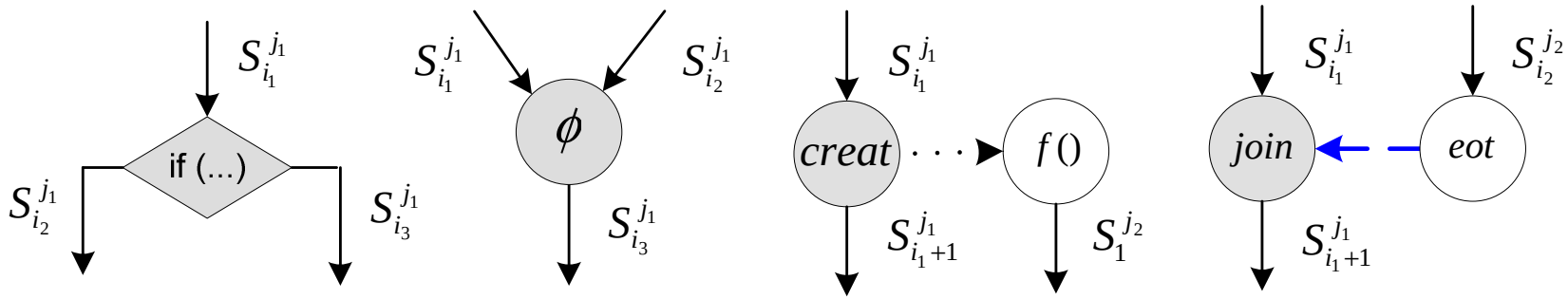
Алгоритм определения действий над объектами синхронизации

$$\left[S_{i_1}^{j_1}; if\,()\,S_{i_2}^{j_1} else\, S_{i_3}^{j_1}\right]: A\left(m, S_{i_2}^{j_1}\right) = A\left(m, S_{i_1}^{j_1}\right), A\left(m, S_{i_3}^{j_1}\right) = A\left(m, S_{i_1}^{j_1}\right)$$

$$\left[\phi\left(S_{i_1}^{j_1}, S_{i_2}^{j_1}\right); S_{i_3}^{j_1}\right]: A\left(m, S_{i_3}^{j_1}\right) = A\left(m, S_{i_1}^{j_1}\right) \cup A\left(m, S_{i_2}^{j_1}\right)$$

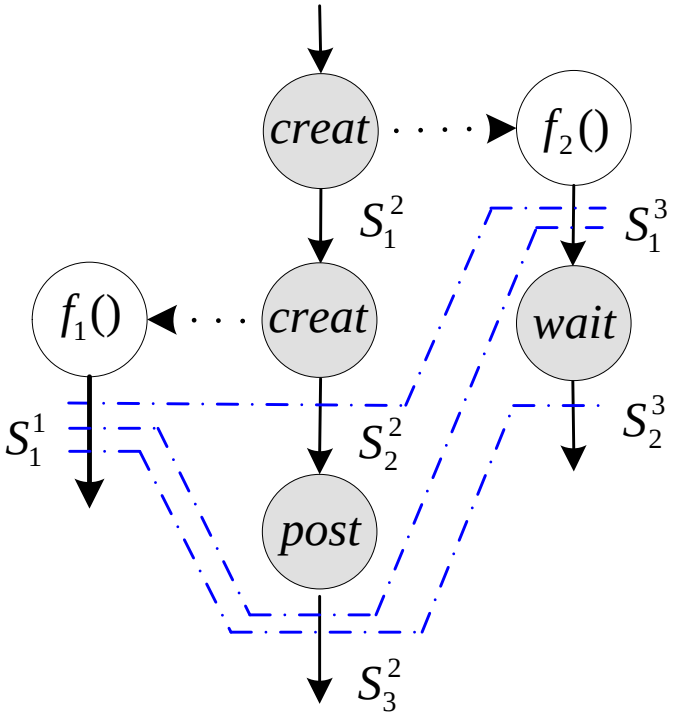
$$\left[create(t,\, pf)\right]: A\left(m, S_{i_1+1}^{j_1}\right) = A\left(m, S_{i_1}^{j_1}\right),\ A\left(m, S_1^{j_2}\right) = \{1 \mid 0\}$$ (мьютекс|семафор)

$$\left[join(t)\right]: A\left(m, S_{i_1+1}^{j_1}\right) = A\left(m, S_{i_1}^{j_1}\right) \oplus A\left(m, S_{i_2}^{j_2}\right)$$ (семафор)



Множество допустимых комбинаций

- Состояние объекта синхронизации в блоке программы определяется с учетом действий над этим объектом в параллельных блоках
- Необходимо построить все комбинации блоков, в которых может выполняться данный блок – **множество допустимых комбинаций**
- Множество допустимых комбинаций для блока $S_{i_1}^{j_1}$ – $C\left(S_{i_1}^{j_1}\right)$



$$\langle S_2^2, S_1^3 \rangle, \langle S_3^2, S_1^3 \rangle, \langle S_3^2, S_2^3 \rangle \in C\left(S_1^1\right)$$

Правило построения допустимых комбинаций

- Правило построения допустимых комбинаций

$$\forall C : C \in C\left(S_{i_1}^{j_1}\right) \Rightarrow \left(\forall S_i^j : S_i^j \in C \Rightarrow \begin{pmatrix} S_i^j \,||\, S_{i_1}^{j_1}, \forall i_2 : i_2 \neq i \Rightarrow S_{i_2}^j \notin C, \\ \forall S_{i_3}^{j_3} : S_{i_3}^{j_3} \in C \Rightarrow S_i^j \,||\, S_{i_3}^{j_3} \end{pmatrix} \right)$$

- При построении комбинаций учитывается, что блок $S_{i_1}^{j_1}$ может выполняться без блоков некоторых потоков – строятся неполные комбинации
- Для построения неполных комбинаций используется механизм виртуальных блоков

Правило определения состояний объектов синхронизации

- Правила определения состояний объектов синхронизации

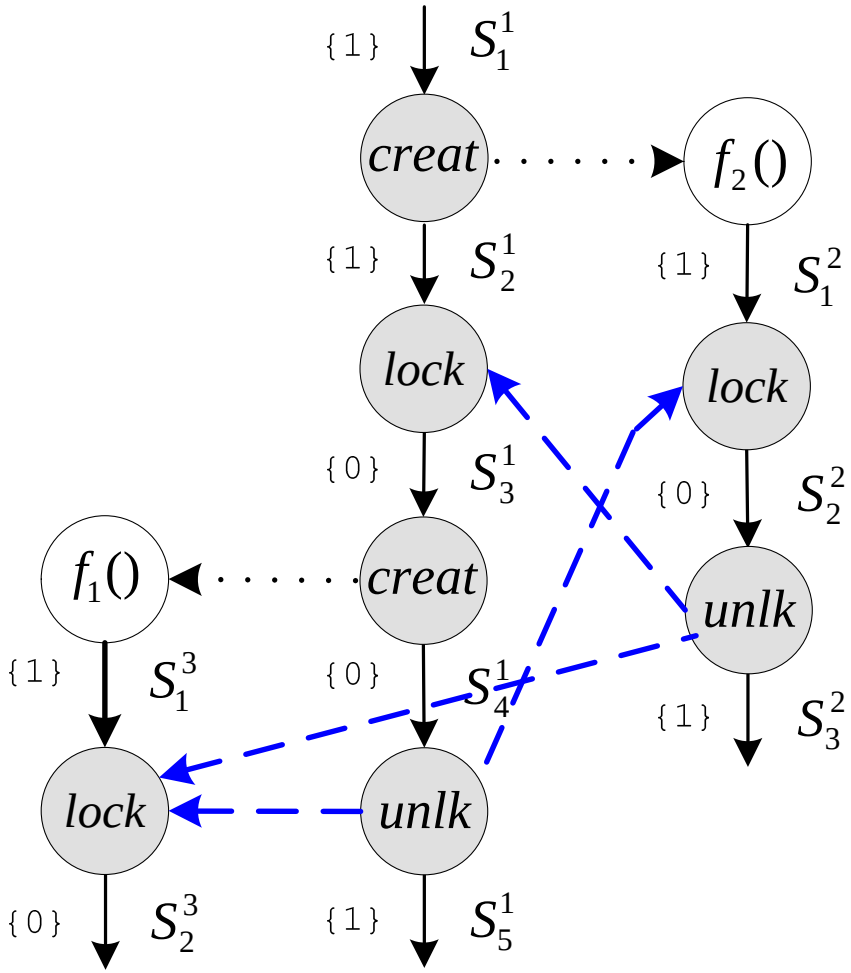
$$m_C\left(S_{i_1}^{j_1}\right) = A\left(m, S_{i_1}^{j_1}\right) \otimes \bigotimes_{\forall S_i^j \in C} A\left(m, S_i^j\right) \left(\text{мьютекс}\right),$$

$$m_C\left(S_{i_1}^{j_1}\right) = A\left(m, S_{i_1}^{j_1}\right) \oplus \bigoplus_{\forall S_i^j \in C} A\left(m, S_i^j\right) \left(\text{семафор}\right),$$

$$m\left(S_{i_1}^{j_1}\right) = \bigcup_{\forall C \in C\left(S_{i_1}^{j_1}\right)} m_C\left(S_{i_1}^{j_1}\right),$$

где $\otimes$ – операция умножения множеств действий, результатом являются все произведения из действий каждого множества

Пример определения состояний мьютекса



Для блока S_1^3

$S_1^3 \,||\, S_4^1, S_5^1, S_1^2, S_2^2, S_3^2$

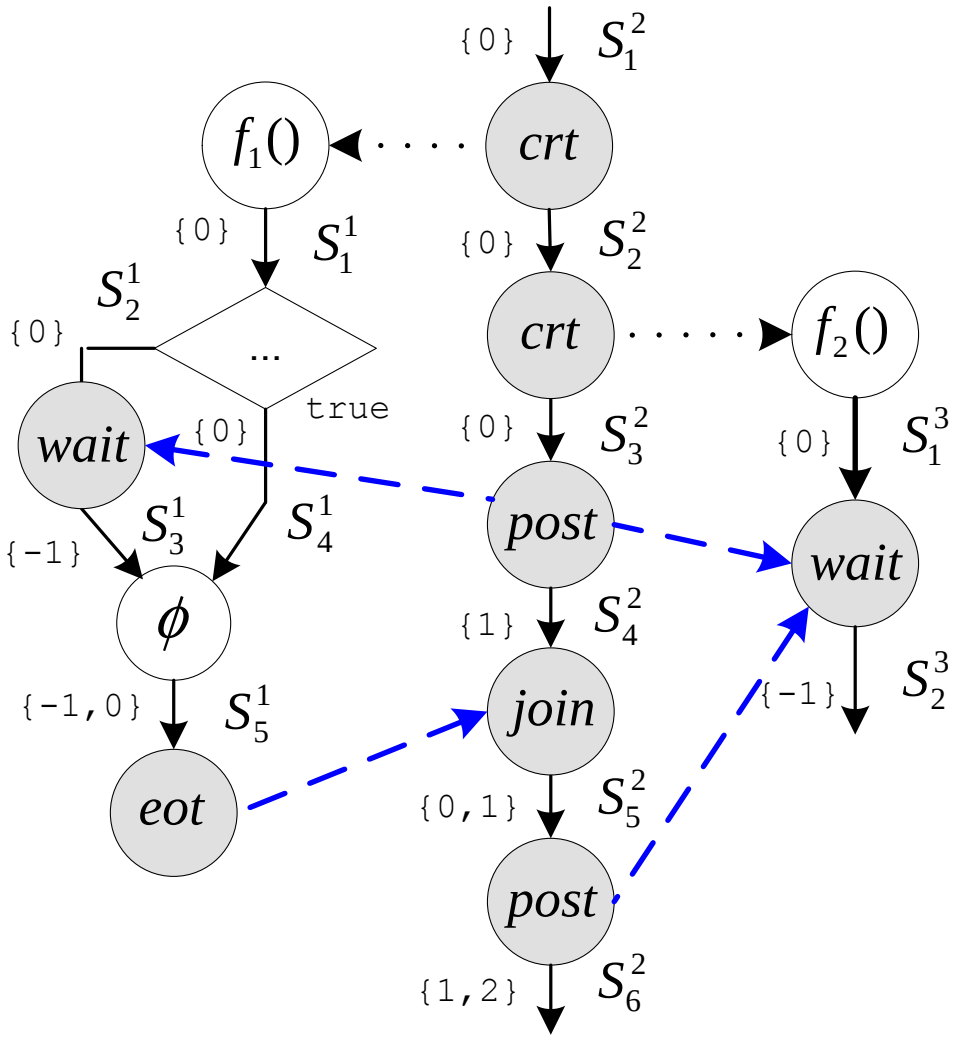
$S_4^1 \,||\, S_1^2, S_3^2 \;\; S_5^1 \,||\, S_1^2, S_2^2, S_3^2$

$\langle S_4^1, S_1^2 \rangle, \langle S_4^1, S_3^2 \rangle,$

$\langle S_5^1, S_1^2 \rangle, \langle S_5^1, S_2^2 \rangle, \langle S_5^1, S_3^2 \rangle \in C(S_1^3)$

$m(S_1^3) = \{0, 1\}$

Пример определения состояний семафора



Для блока S_1^3

$$S_1^3 \,||\, S_1^1 \div S_5^1, S_3^2 \div S_6^2$$

$$\langle S_3^2, S_1^1\rangle, \langle S_3^2, S_2^1\rangle, \langle S_3^2, S_4^1\rangle, \langle S_3^2, S_5^1\rangle,$$

$$\langle S_4^2, S_1^1\rangle, \langle S_4^2, S_2^1\rangle, \langle S_4^2, S_3^1\rangle, \langle S_4^2, S_4^1\rangle,$$

$$\langle S_4^2, S_5^1\rangle, \langle S_5^2, S_v^1\rangle, \langle S_6^2, S_v^1\rangle \in C(S_1^1)$$

$$m(S_1^1) = \{0,1,2\}$$

Алгоритм построения отношений параллельности

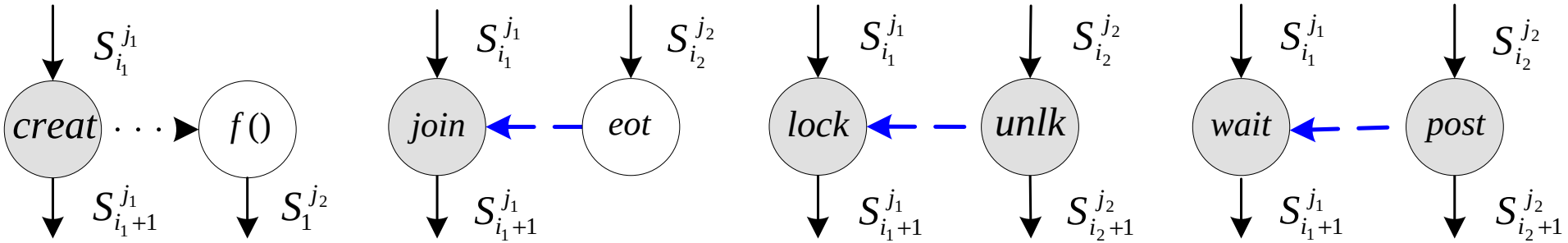
$$\frac{\forall S_i^j : S_{i_1}^{j_1} \,||\, S_i^j}{[create(t, pf)] : S_{i_1+1}^{j_1} \,||\, S_1^{j_2}, S_{i_1+1}^{j_1} \,||\, S_i^j, S_1^{j_2} \,||\, S_i^j},$$

$$\frac{\forall S_i^j : S_{i_1}^{j_1} \,||\, S_i^j}{[S_{i_1}^{j_1}; if\,() S_{i_2}^{j_1} else\, S_{i_3}^{j_1}] : S_{i_2}^{j_1} \,||\, S_i^j, S_{i_3}^{j_1} \,||\, S_i^j},$$

$$\frac{\forall S_i^j \in C : C \in C(S_{i_1}^{j_1}), S_{i_2}^{j_2} \in C}{[join(t)] : S_{i_1+1}^{j_1} \,||\, S_i^j},$$

$$\frac{\forall S_i^j : S_{i_1}^{j_1} \,||\, S_i^j \vee S_{i_2}^{j_1} \,||\, S_i^j}{[\phi(S_{i_1}^{j_1}, S_{i_2}^{j_1}); S_{i_3}^{j_1}] : S_{i_3}^{j_1} \,||\, S_i^j},$$

$$\frac{\forall S_i^j \in C : C \in C(S_{i_1}^{j_1}), \exists a \in m_C(S_{i_1}^{j_1}) : a > 0}{[lock(m) | wait(m)] : S_{i_1+1}^{j_1} \,||\, S_i^j}.$$

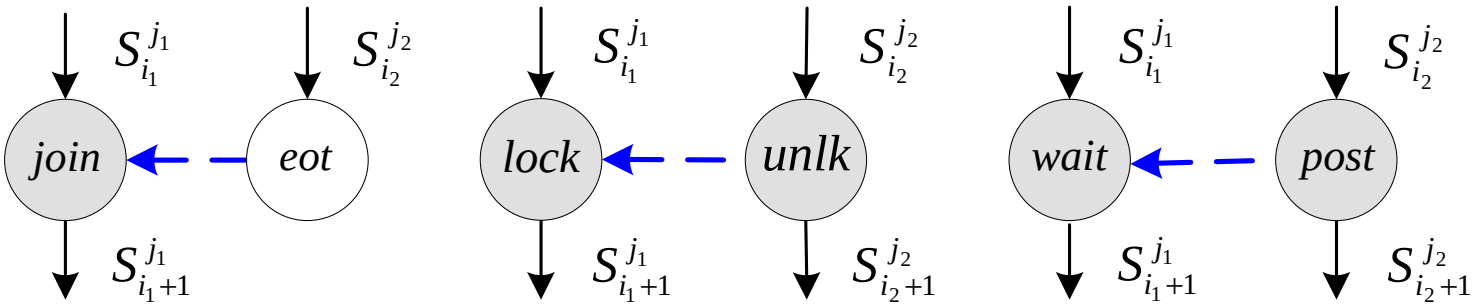


Алгоритм построения отношений синхронизации

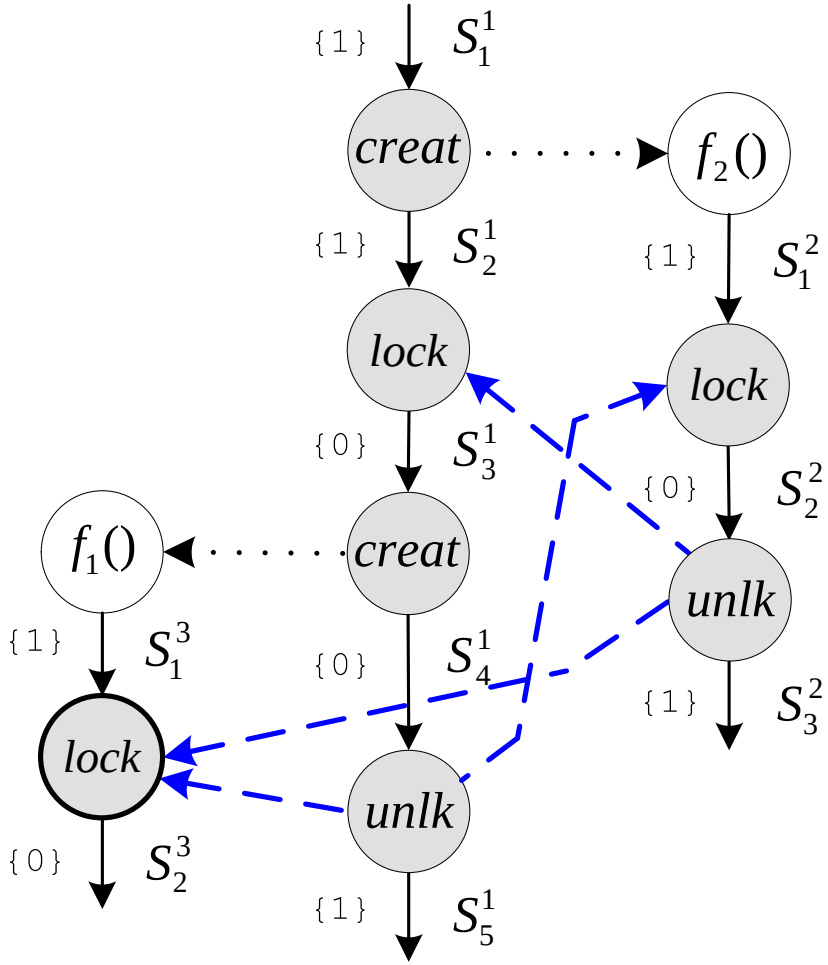
$$\frac{-}{[join(t)]: S_{i_2}^{j_2} \rightarrow S_{i_1+1}^{j_1}},$$

$$\frac{S_{i_1}^{j_1} \,||\, S_{i_2}^{j_2}}{[lock(m)]: S_{i_2}^{j_2} \rightarrow S_{i_1+1}^{j_1}},$$

$$\frac{\exists C : C \in C\left(S_{i_1}^{j_1}\right), \exists a \in m_C\left(S_{i_1}^{j_1}\right) : a = 0,\ S_{i_2}^{j_2} \in C}{[wait(m)]: S_{i_2}^{j_2} \rightarrow S_{i_1+1}^{j_1}}$$



Пример определения отношений



Для lock после блока S_1^3

$\langle S_4^1, S_1^2 \rangle, \langle S_4^1, S_3^2 \rangle,$

$\langle S_5^1, S_1^2 \rangle, \langle S_5^1, S_2^2 \rangle, \langle S_5^1, S_3^2 \rangle \in C(S_1^3)$

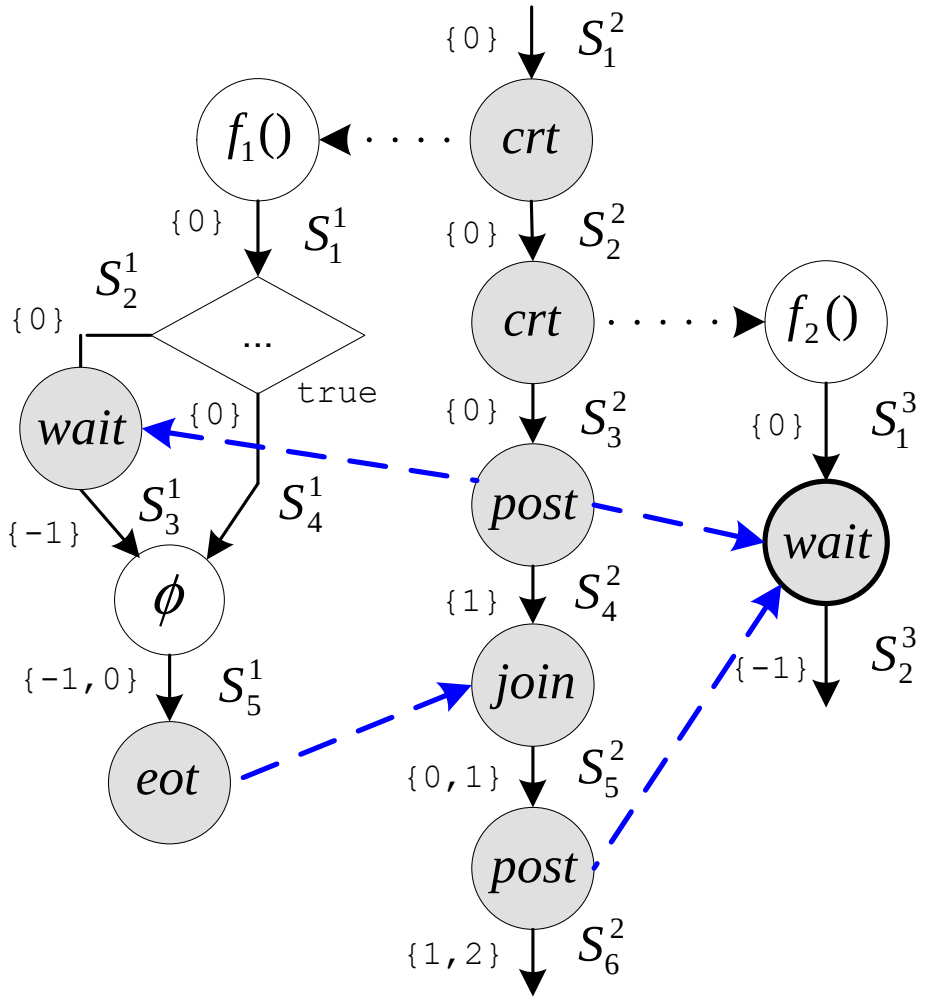
$\langle S_5^1, S_1^2 \rangle, \langle S_5^1, S_3^2 \rangle$

$S_2^3 \,||\, S_5^1, S_1^2, S_3^2$

$S_1^3 \,||\, S_4^1, S_2^2$

$S_4^1 \to S_2^3, S_2^2 \to S_2^3$

Пример определения отношений



Для конструкции wait после S_1^3

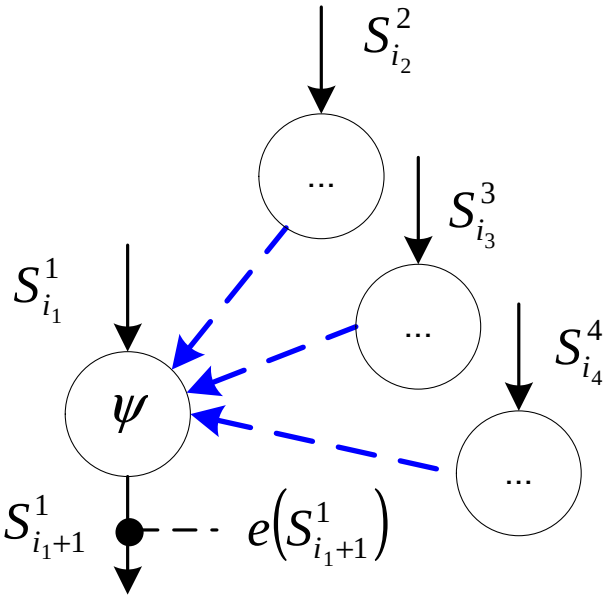
$$\langle S_3^2, S_1^1\rangle, \langle S_3^2, S_2^1\rangle, \langle S_3^2, S_4^1\rangle, \langle S_3^2, S_5^1\rangle,$$
$$\langle S_4^2, S_1^1\rangle, \langle S_4^2, S_2^1\rangle, \langle S_4^2, S_3^1\rangle, \langle S_4^2, S_4^1\rangle,$$
$$\langle S_4^2, S_5^1\rangle, \langle S_5^2, S_v^1\rangle, \langle S_6^2, S_v^1\rangle \in C(S_1^3)$$

$$\langle S_4^2, S_1^1\rangle, \langle S_4^2, S_2^1\rangle, \langle S_4^2, S_4^1\rangle,$$
$$\langle S_4^2, S_5^1\rangle, \langle S_5^2, S_v^1\rangle, \langle S_6^2, S_v^1\rangle$$
$$S_2^3 \,||\, S_1^1, S_2^1, S_4^1, S_5^1, S_v^1, S_4^2, S_5^2, S_6^2$$

$$\langle S_3^2, S_1^1\rangle, \langle S_5^2, S_v^1\rangle$$
$$S_3^2 \to S_2^3, S_5^2 \to S_2^3$$

Алгоритм учета взаимного влияния потоков программы

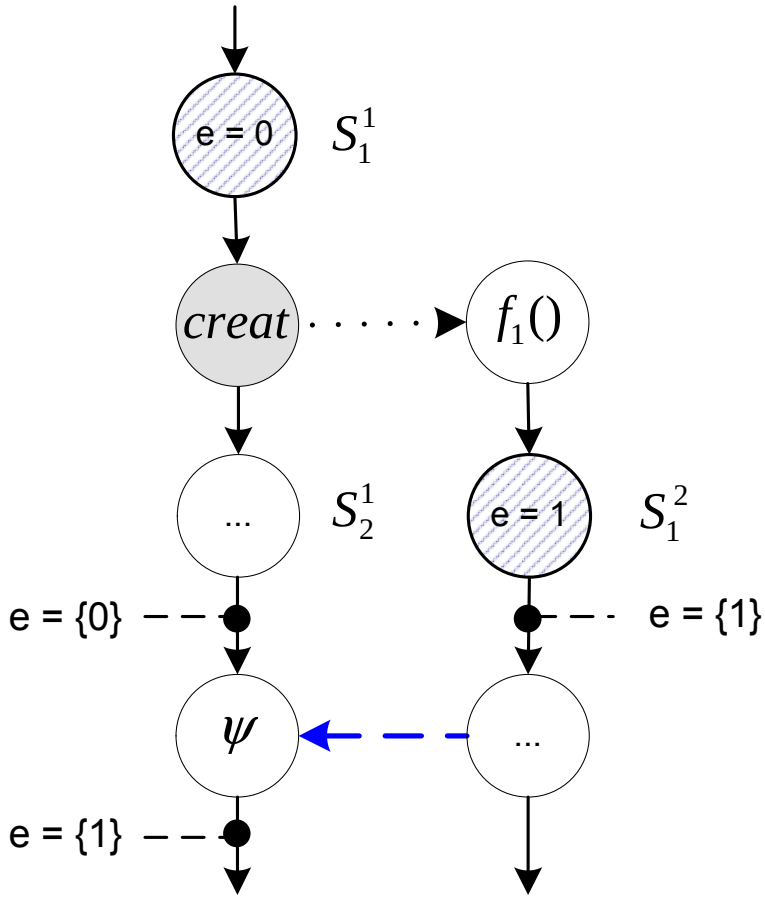
- Алгоритм учета взаимного влияния потоков распространяет изменения значений разделяемых объектов между потоками программы
- Для этого используются ***ψ*-функции**
- Каждая *ψ*-функция имеет один вход из текущего потока и одни или несколько входов из других потоков
- В *ψ*-функциях выполняется объединение значений разделяемых объектов со всех входов



$$e(S^1_{i_1+1}) = e(S^1_{i_1}) \cup e(S^2_{i_2}) \cup \ldots$$

Актуальные значения в ψ-функциях

- Значения на некоторых входах ψ-функции могут оказаться неактуальными
- Определение актуальных значений в ψ-функциях позволяет повысить точность получаемых результатов

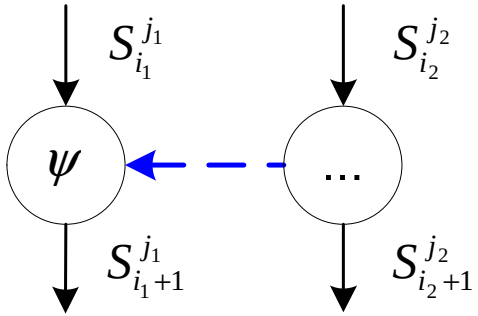


Определение актуальных значений

- Алгоритм определения актуальных значений для ψ-функции с двумя входами

$$\frac{\forall e \in E_{j_1}}{e\left(S_{i_1+1}^{j_1}\right) = e\left(S_{i_1}^{j_1}\right)}, \quad \frac{\forall e \in E_{j_2}}{e\left(S_{i_1+1}^{j_1}\right) = e\left(S_{i_2}^{j_2}\right)},$$

$$\frac{\forall e \in E_{j_1, j_2}}{e\left(S_{i_1+1}^{j_1}\right) = e\left(S_{i_1}^{j_1}\right) \cup e\left(S_{i_2}^{j_2}\right)}$$

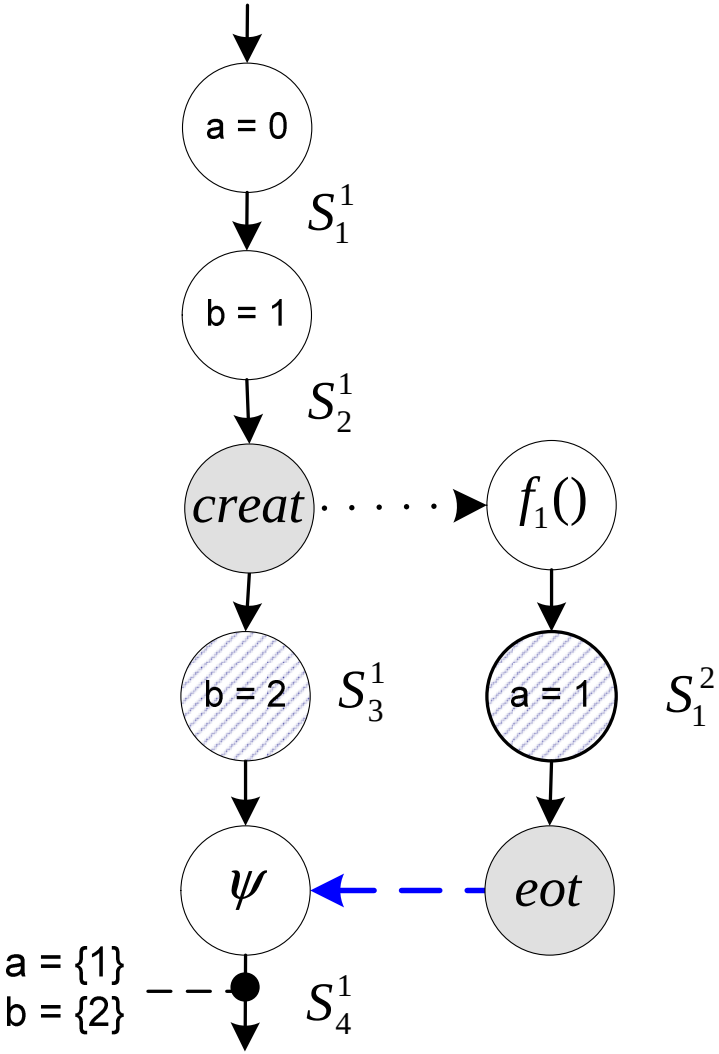


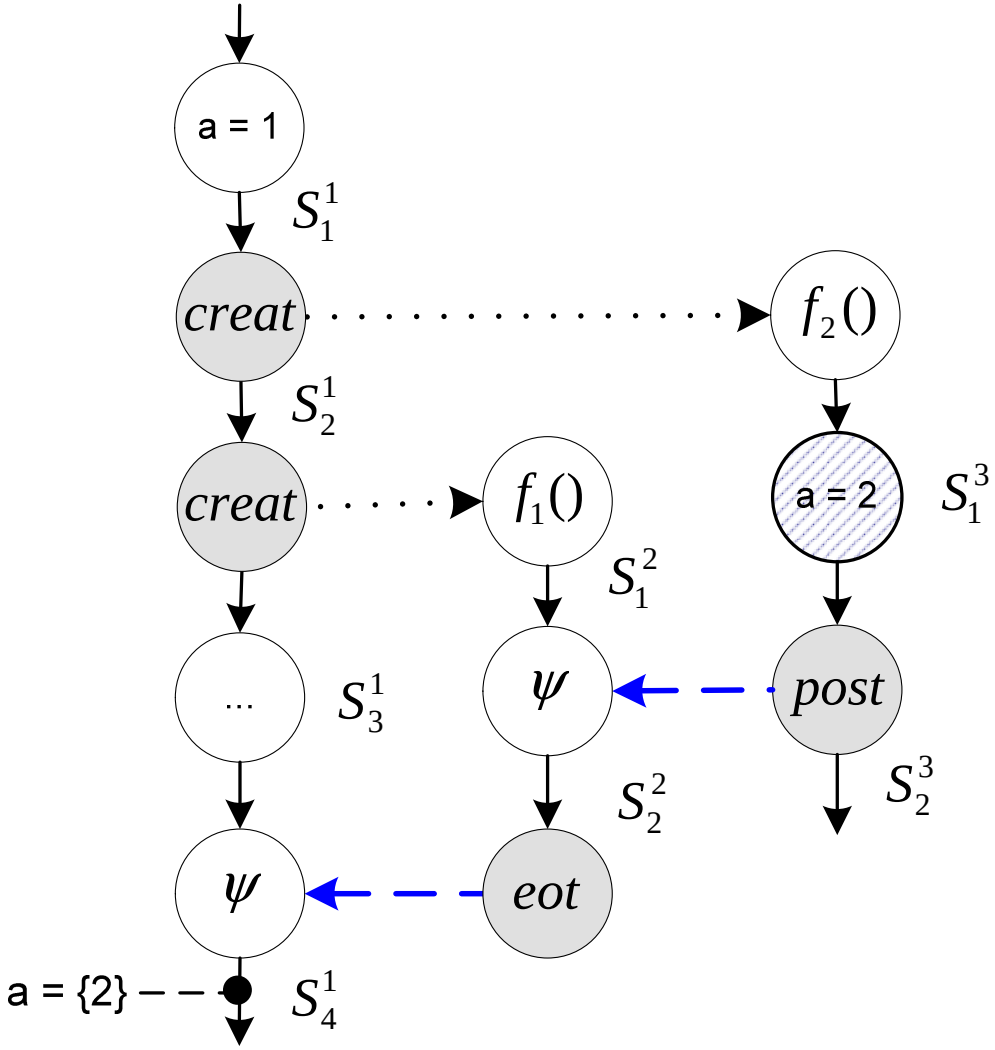
- где $e\left(S_i^j\right)$ – значение разделяемого объекта e в блоке S_i^j,

  E_{j_1} (E_{j_2}) – множество разделяемых объектов, значение которых менялось в блоках параллельных $S_{i_2}^{j_2}$ ($S_{i_1}^{j_1}$) и не параллельных $S_{i_1}^{j_1}$ ($S_{i_2}^{j_2}$),

  E_{j_1, j_2} – разделяемые объекты, не входящие в E_{j_1} и E_{j_2}

Примеры определения актуальных значений



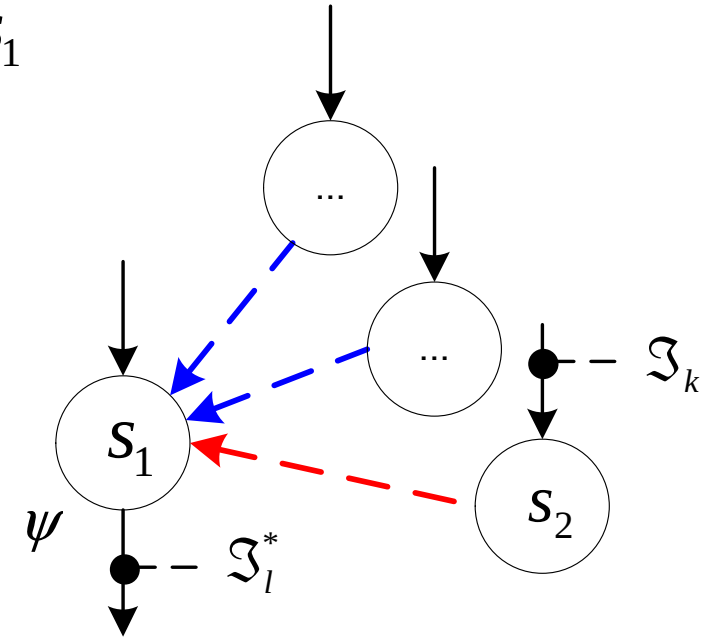


Итеративный алгоритм

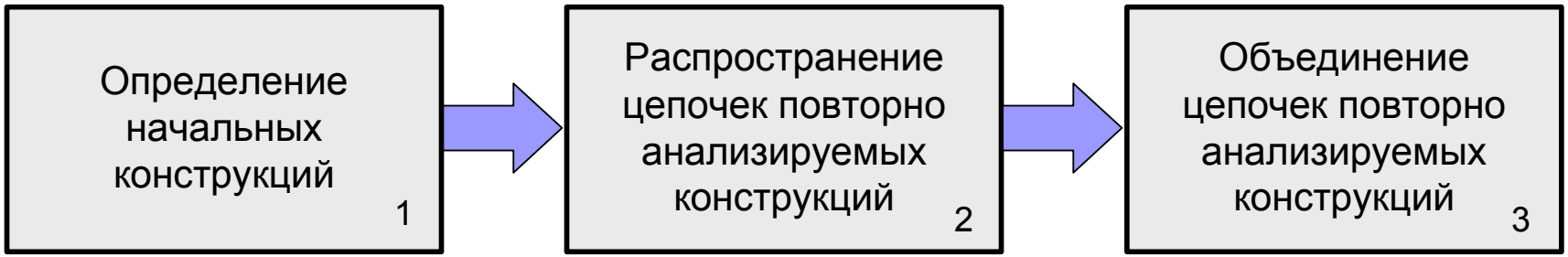
- Для получения полных результатов, при анализе конструкций блока программы необходимо учитывать информацию, полученную при анализе конструкций блоков других потоков, которые могут выполняться до этого блока
- Для программ с несколькими порядками выполнения блоков не существует единственной последовательности анализа конструкций, обеспечивающей получение полных результатов
- **Итеративный алгоритм** – выполняет повторный анализ конструкций, исходные данные для которых изменились в результате анализа конструкций других потоков программы

Итеративный алгоритм

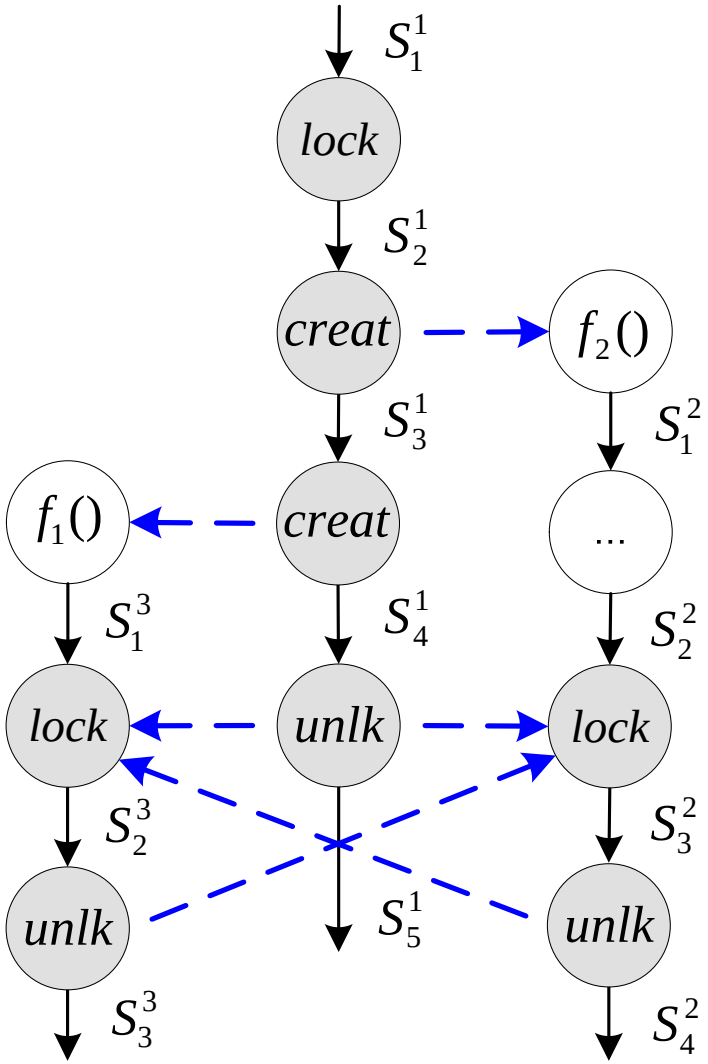
- Повторный анализ начинается в конструкции S_1 `lock`, `wait`, `join`, `state`, для которой найдена новая конструкция S_2 (`unlock`, `post`, `eot`, ...)
  - S_2 имеет соответствующий тип
  - S_1 и S_2 выполняются для одного и того же потока/объекта синхронизации
  - для блоков перед конструкциями S_1 и S_2 имеется отношение параллельности



$$\Im_l^* = \Im_l \bigcup \Im_k$$



Пример работы итеративного алгоритма



Последовательности выполнения блоков

$S_4^1, S_3^2, S_2^3 \quad S_4^1, S_2^3, S_3^2$

Для блока S_3^2

$S_3^2 \,||\, S_5^1, S_1^3 \quad S_4^1 \rightarrow S_3^2$

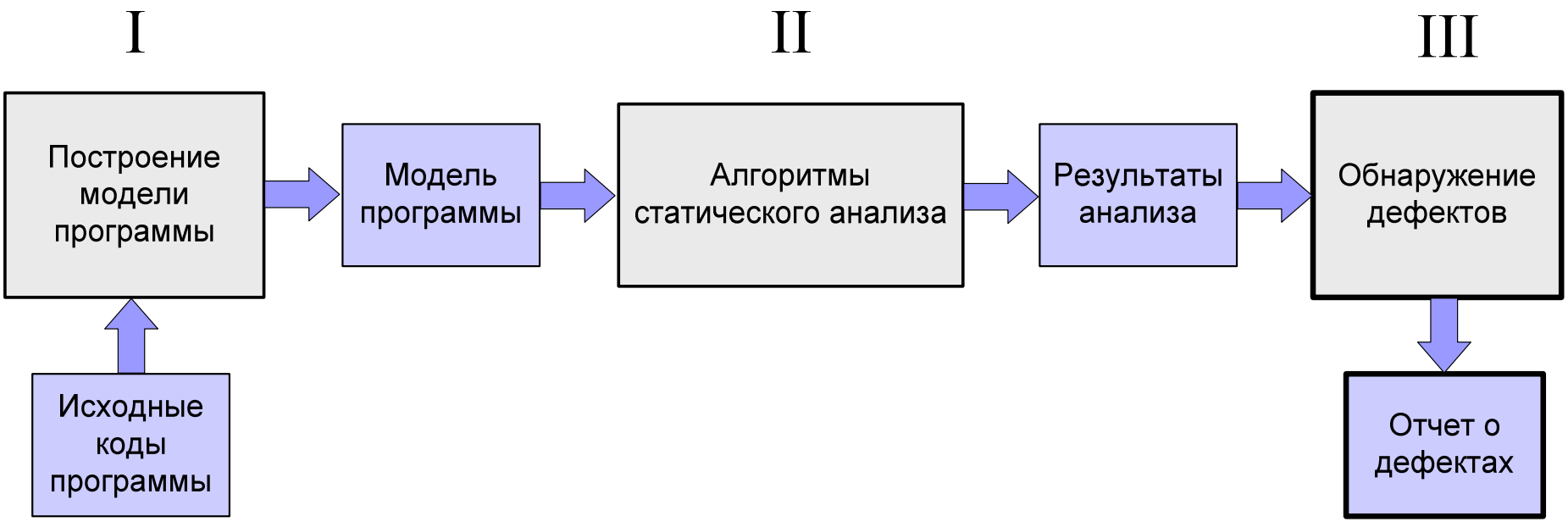
Для блока S_2^3

$S_2^3 \,||\, S_5^1, S_1^2, S_2^2, S_4^2 \quad S_4^1 \rightarrow S_2^3, S_3^2 \rightarrow S_2^3$

Повторный анализ конструкции `lock` после S_2^2

$S_3^2 \,||\, S_3^3 \quad S_2^3 \rightarrow S_3^2$

Обнаружение программных дефектов на основе статического анализа



Пример обнаружение дефекта

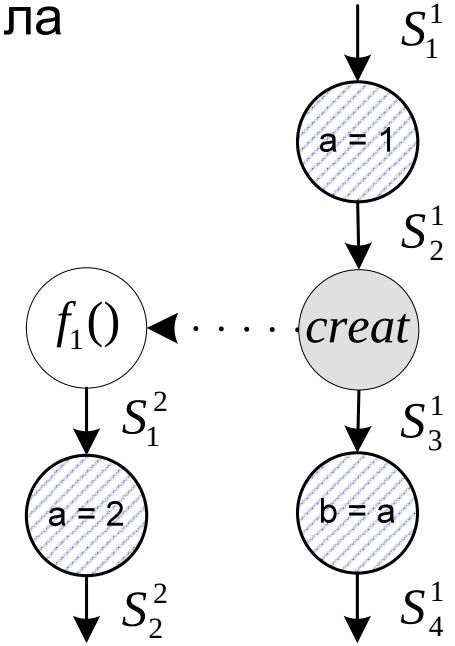
```
int* p;
…
void f() {
  …
  pthread_mutex_lock(&m);
  free(p);
  pthread_mutex_unlock(&m);
}

int main() {
  p = malloc(...);
  pthread_create(&t,NULL,f,NULL);
  …
  pthread_mutex_lock(&m);
  *p = 1;
  pthread_mutex_unlock(&m);
  …
}
```

Обнаружение ошибок Data race

- Для обнаружения этих ошибок используются правила

$$\frac{\exists S_{i_1}^{j_1}, S_{i_2}^{j_2}, e : S_{i_1}^{j_1} \,||\, S_{i_2}^{j_2}, e \in Use\left(S_{i_1}^{j_1}\right), e \in Def\left(S_{i_2}^{j_2}\right)}{DEFECT_{RACE}},$$

$$\frac{\exists S_{i_1}^{j_1}, S_{i_2}^{j_2}, e : S_{i_1}^{j_1} \,||\, S_{i_2}^{j_2}, e \in Def\left(S_{i_1}^{j_1}\right), e \in Def\left(S_{i_2}^{j_2}\right)}{DEFECT_{RACE}}$$



$e \in Def\left(S_i^j\right)$ – факт изменения значения объекта в блоке программы,

$e \in Use\left(S_i^j\right)$ – факт использования значения объекта в блоке программы.

Пример обнаружение Data race

```
int i = 0;
…
void f() {
  if (…) {
    i = 1;
  }
}
void g() {
  if (…) {
    i++;
  }
}

int main() {
  pthread_create(&t1,NULL,f,NULL);
  pthread_create(&t2,NULL,g,NULL);
  …
}
```

Оценка вычислительной сложности

- Наибольшую сложность имеют правила, использующие допустимые комбинации – правила для конструкций `join`, `lock`, `wait`, `state`

$$O\left(n^k \cdot k^2 \cdot \log(k \cdot n)\right)$$

- Оценка вычислительной сложности с учетом итеративного алгоритма

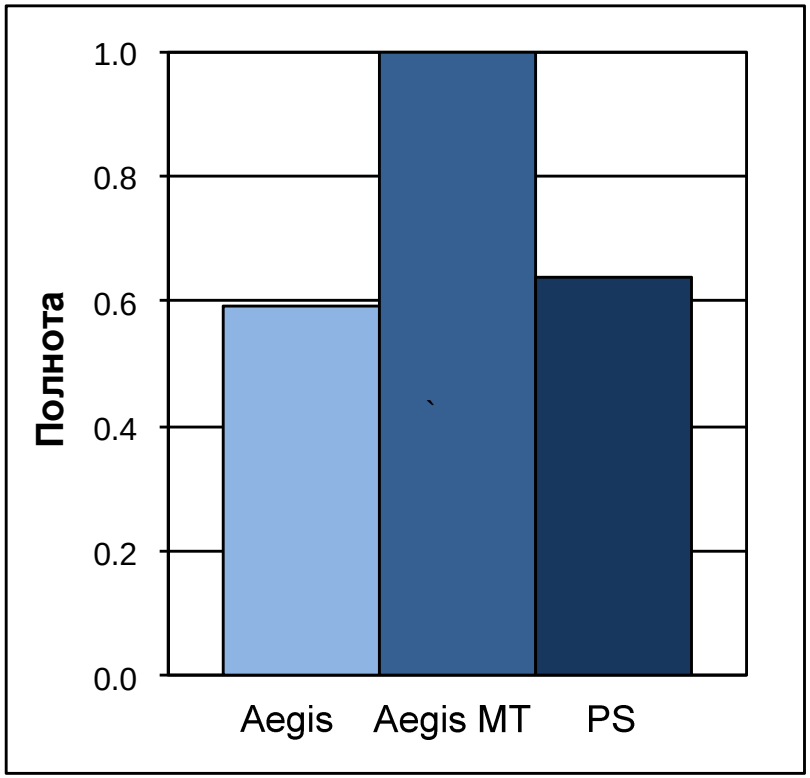
$$O\left(n_S \cdot \left(n^k \cdot k^2 \cdot \log(k \cdot n)\right) \cdot n_I\right) \quad n_I = O\left(n \cdot k \cdot n_S\right) \quad \overline{n}_I = 3.1$$

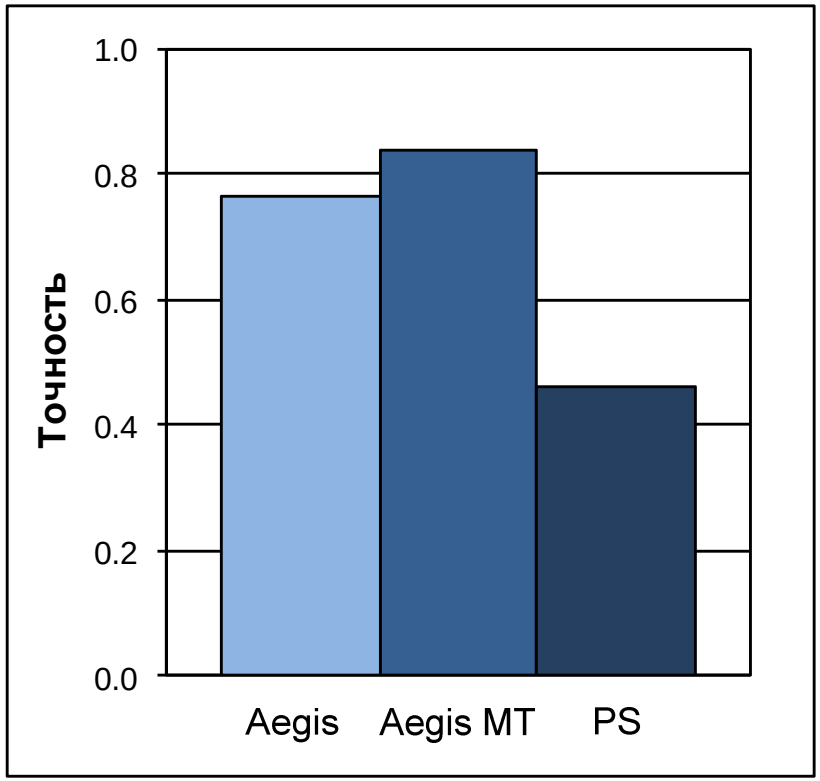
- Итоговая оценка $O\left(n_S^2 \cdot n^{k+1}\right)$
  - k – число потоков
  - n – число параллельных блоков в каждом потоке
  - n_S – число анализируемых конструкций синхронизации в программе
  - n_I - среднее количество выполнений повторного анализа конструкции за счет итеративного алгоритма

Проведение экспериментов

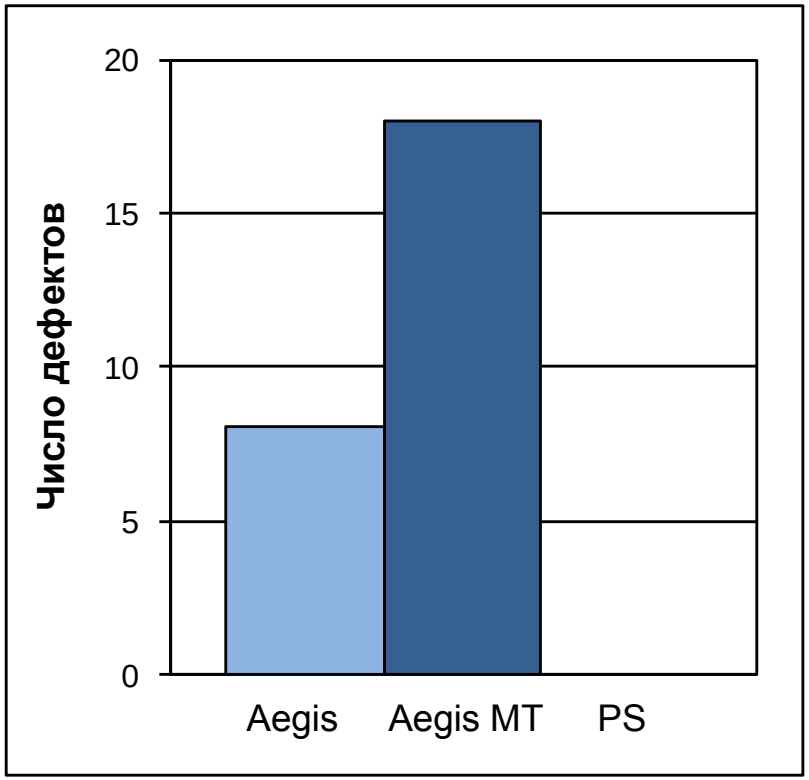
- Реализация разработанного подхода в средстве Aegis MT
- Для проверки полноты и оценки точности получаемых результатов проведены экспериментальные исследования
- Набор анализируемых программ
  - специальные тестовые программы – 140 программ
  - многоклиентская звуковая библиотека – 4 тестовых проекта
- Средства обнаружения дефектов
  - Aegis MT
  - Aegis
  - Parasoft C++ test

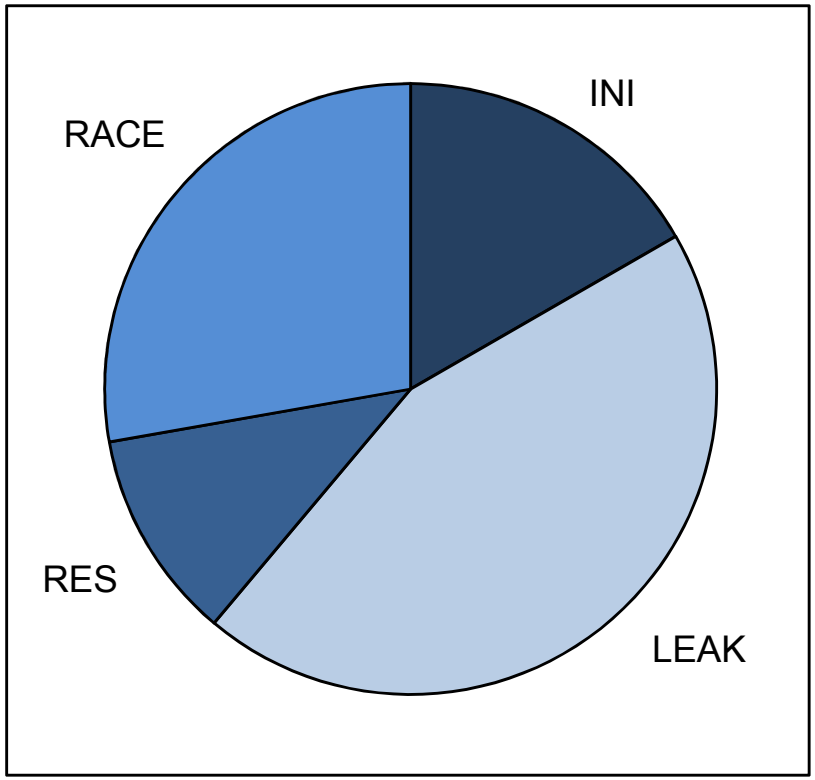
На наборе специальных тестовых программ





На тестовых проектах звуковой библиотеки





- Точность Aegis MT – **20%**

Вместо заключения

- Этапы развития методов анализа многопоточных программ
  - разработка отдельных методов (1988 – 2004)
  - выполнение разработанных методов с распространением получаемых результатов (2000 – 2010)
  - **комбинирование отдельных методов в единый подход (2008 – …)**
- Использование аппроксимаций
  - отсутствие учета конструкций синхронизации
  - ограничение на число анализируемых переключений контекстов
  - …
  - **объединение состояний программы в *Φ*-функциях**
  - **объединение порядков выполнения программы в *ψ*-функциях**