

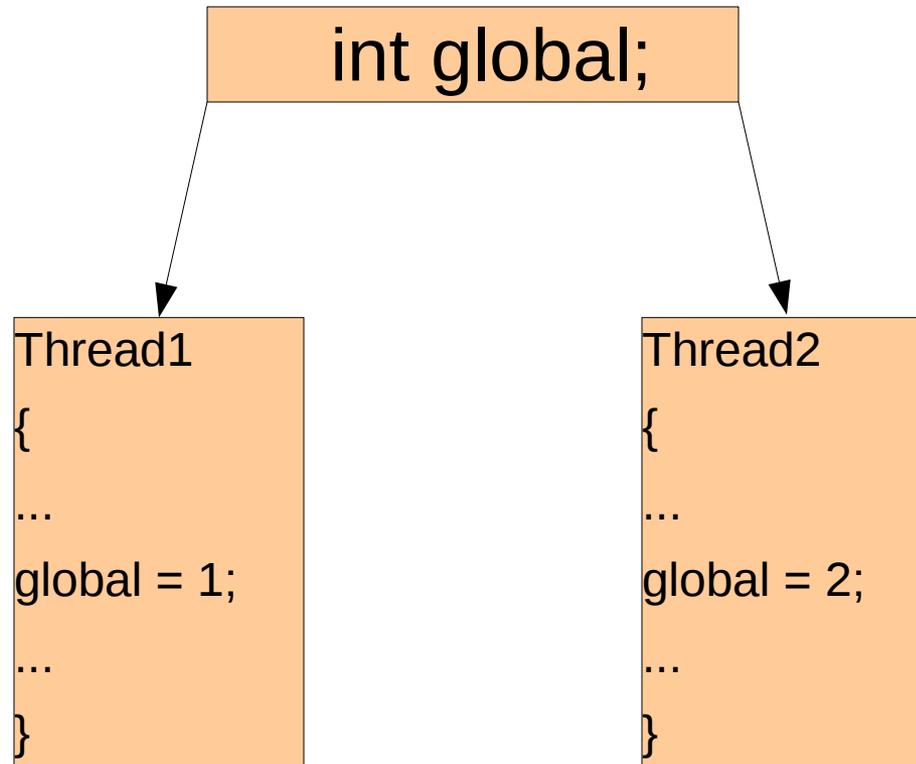
Поиск состояний гонок методами статического анализа

Андрианов Павел
andrianov@ispras.ru



Institute for System Programming of the Russian Academy of Sciences

Состояние гонки



Существующие инструменты

- Динамические (Eraser, Intel Thread Checker, Sun Thread Analyzer)
- Статические (Coverity Prevent, KlocWork K7, Locksmith, Prefast)
- Проверка на основе моделей (SLAM)

Особенности задачи

- Специфика ядра операционной системы
- Масштаб анализируемого кода
- Критичность ошибки

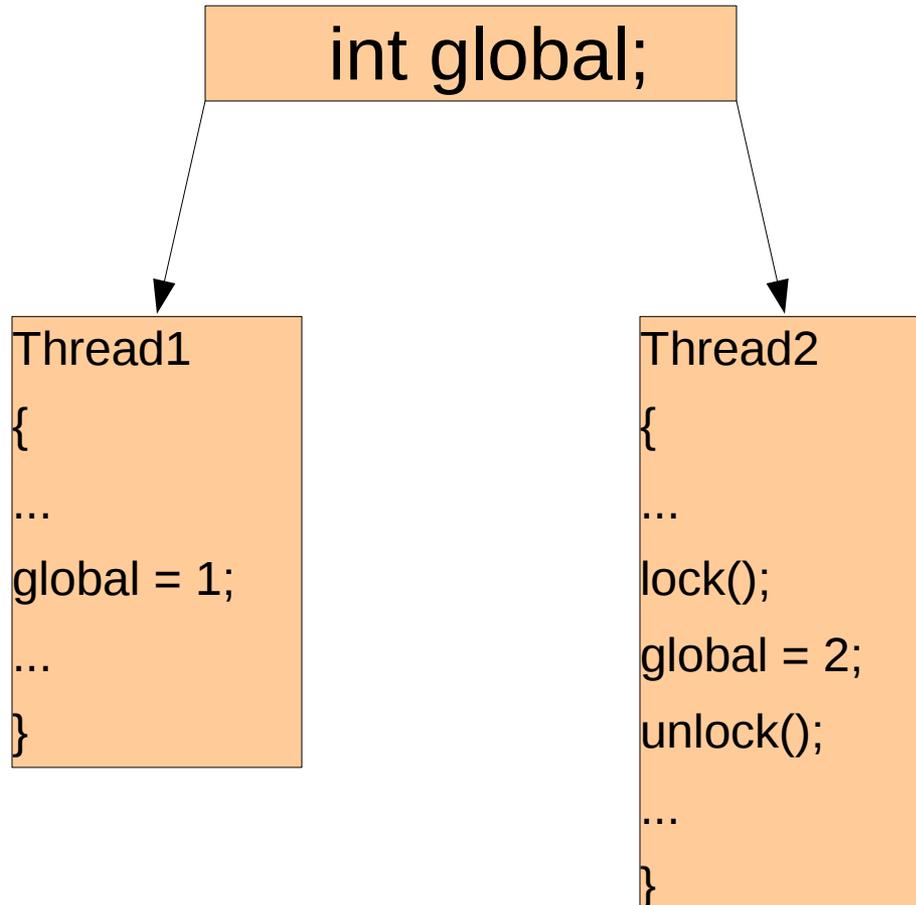
Некоторые понятия

- Переменная – локальные, глобальные и поля структур
- Механизм синхронизации (блокировка)
- Использование переменной – доступ к данным на чтение или запись

Состояние гонки

- *Потенциальным состоянием гонки* будем считать ситуацию, в которой доступ к одной и той же переменной происходит с **разным непересекающимся** набором блокировок, при этом одно из обращений является записью.

Состояние гонки



Пример

```
int func1() {  
    struct A *s;  
    int local;  
    s->f = ...  
    lock();  
    global = ...  
    local = ...  
    unlock();  
}
```

```
int func2() {  
    int local;  
    ...  
    local = global;  
}
```

Захват блокировки

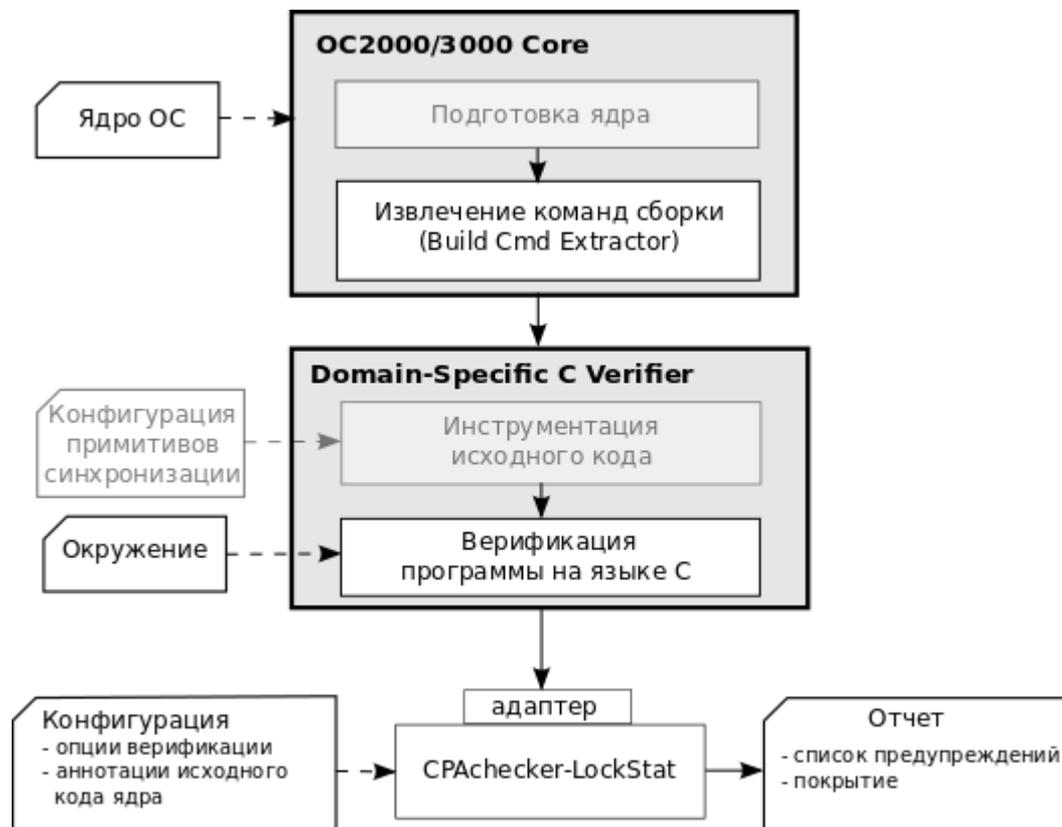
Освобождение блокировки

Доступ к переменной на запись

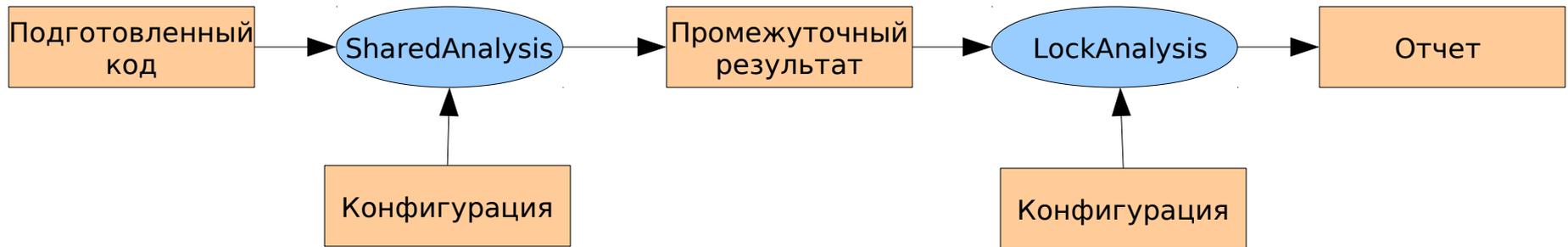
Доступ к переменной на чтение

Возможная гонка

Архитектура решения

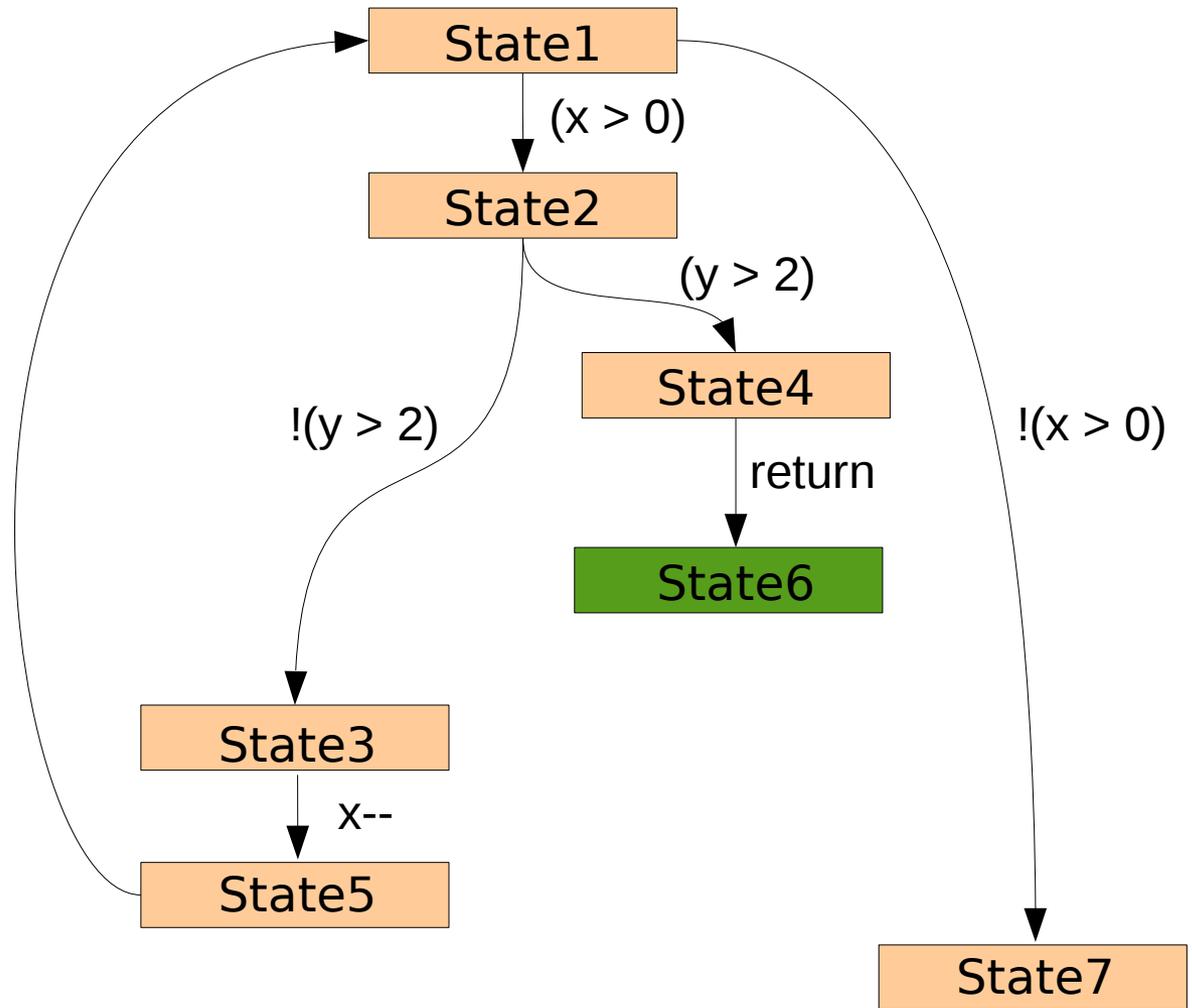


Структура инструмента



Общая идея

```
while (x > 0) {  
  if (y > 2) {  
    return;  
  }  
  x--;  
}
```



Адаптивный статический анализ (Configurable Program Analysis)

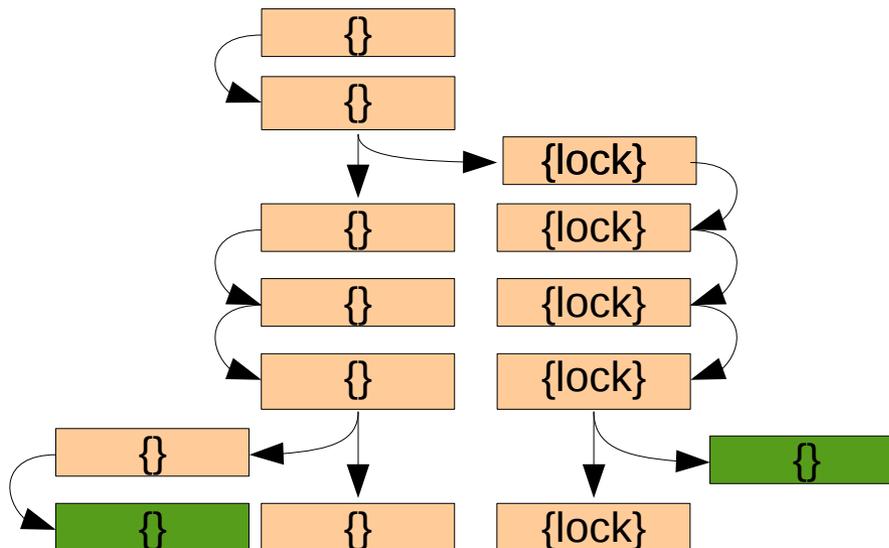
- Абстрактный домен – задает множество абстрактных состояний
- Отношение переходов – для каждого абстрактного состояния e определяет следующее состояние e' .
- Оператор слияния – объединяет информацию от двух абстрактных состояний
- Оператор останова – проверяет, покрывается ли состояние множеством других

Пример: LockStatisticsCPA

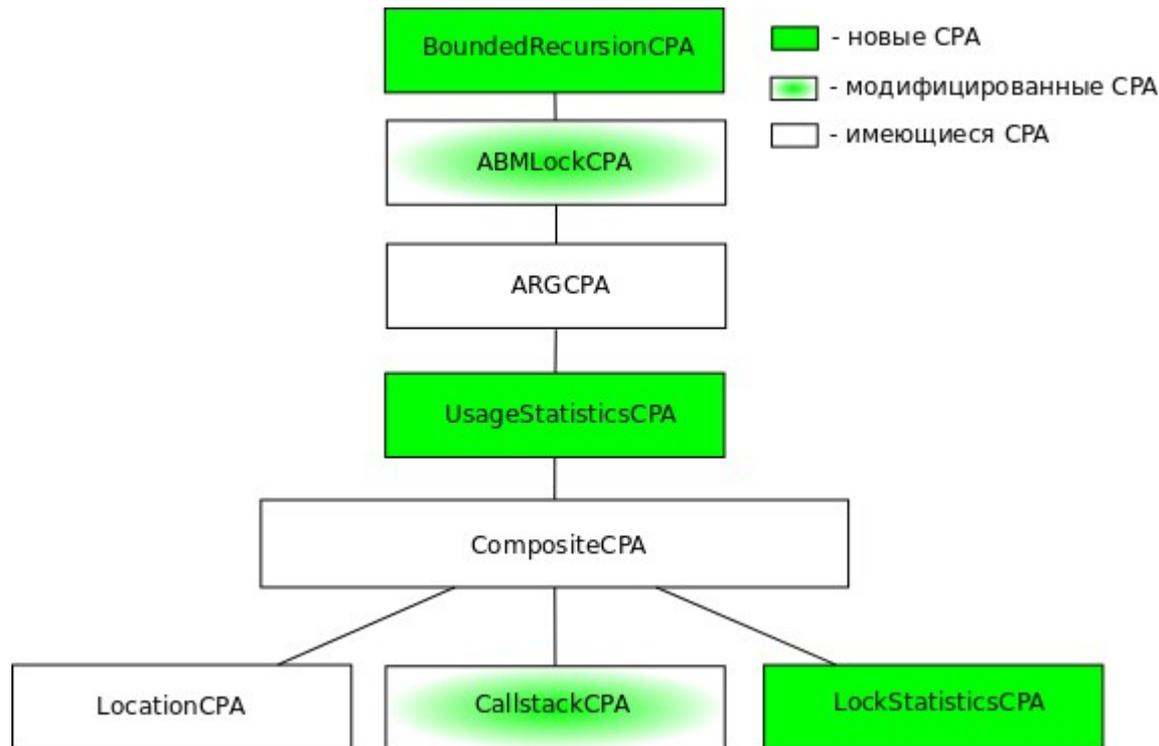
- Абстрактное состояние – множество захваченных блокировок
- Отношение переходов – изменяет состояние, если вызывается функция захвата или освобождения блокировки
- Оператор слияния – SEP, т.е. состояния никогда не объединяются
- Оператор останова – SEP, т.е. состояние покрыто тогда, когда в переданном множестве имеется равное ему состояние

Пример: LockStatisticsCPA

```
int global1, global2;  
int func() {  
    int a = 0;  
    if (global1) {  
        lock();  
    }  
    global2++;  
    if (global1) {  
        unlock();  
    }  
}
```



Конфигурация алгоритмов CPA в LockAnalysis



Локальность данных

```
int func(int *b) {
```

```
    int *a;
```

```
    a = malloc();
```

```
    *a = 1;
```

```
    a = b;
```

```
    *a = 1;
```

```
    a = &global;
```

```
    *a = 1;
```

```
}
```

a указывает на локальные данные

Обращение к локальным данным – не гонка

a указывает на неизвестные данные

Запись по неизвестному указателю – неизвестно

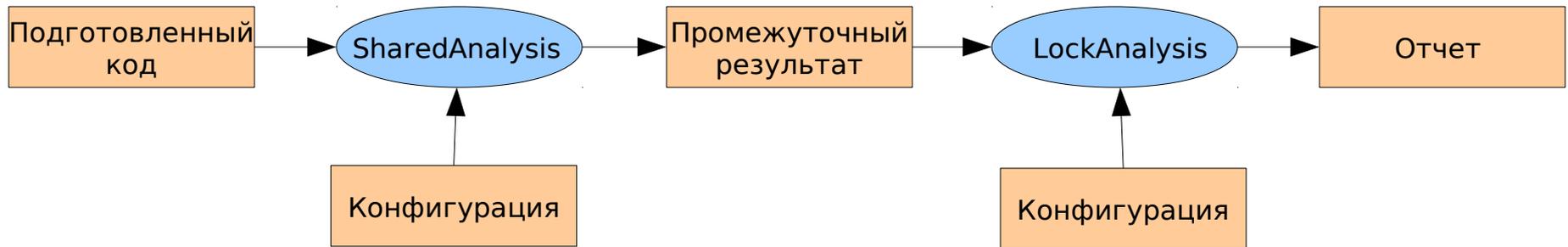
a указывает на глобальные данные

Обращение к разделяемым данным – гонка

Структура SharedCPA

- Абстрактное состояние – содержит информацию о локальности переменных, доступных в данной точке программы
- Отношение переходов – консервативно изменяет состояние
- Оператор слияния – JOIN, т.е. информация из двух состояний объединяется с учетом иерархии *global > null > local*
- Оператор останова - SEP

Структура инструмента



Отчет

Statistics	General	Unsafe
Global variables:	591	24
Simple:	411	20
Pointer:	180	4
Local variables:	1296	50
Simple:	0	0
Pointer:	1296	50
Structure fields:	1324	300
Simple:	1038	287
Pointer:	286	13
Total variables:	3211	374

Finded locks:

1. `global_lock lock()`
2. `global_lock pthread_mutex_lock(m->_mutex)(0)`
3. `global_lock pthread_mutex_lock(mutex)(0)`

List of unsafes:

Визуализация ошибочной трассы

int *a

The image displays a debugger window with two panes. The left pane, titled "Error trace", shows a call stack for a function named `pthread_mutex_lock()`. The stack includes `pthread_mutex_lock()`, `_f()`, `_g()`, and `_main()`. The right pane, titled "Source code", shows the source code for `Test.c`. The code includes a global variable `global`, a function `g(int *a)` that increments `*a`, a function `f(int *a)` that calls `g(a)` and returns the result, and a `main()` function that locks a mutex, calls `f(&global)`, calls `f(&global)` again, and then unlocks the mutex.

Error trace

```
Function bodies | Blocks | Others...
/*Number of usages:4*/
/*Two examples:*/
/*_____*/
/*global_lock pthread_mutex_lock(m)(0)*/
_main()
{
16 pthread_mutex_lock() { /* Function call is skipped d
17 _f()
   {
   9   _g()
   5   {
       *a = ...;
   }
}
/*_____*/
/*Without locks*/
_main()
17 {
   _f()
   {
   9   _g()
   5   {
       ... = *a;
   }
}
}
```

Source code

```
Test.c
1 #line 2 "/home/alpha/git/cpachecker/test/Test.c"
2 int global;
3
4 int g(int *a) {
5     (*a)++;
6 }
7
8 int f(int *a) {
9     int r = g(a);
10    return r;
11 }
12
13 int main() {
14     mutex m;
15     f(&global);
16     pthread_mutex_lock(m);
17     f(&global);
18     pthread_mutex_unlock(m);
19 }
```

Полученные результаты

- Покрытие 26% кода по функциям и 20% по строкам кода
- Ошибки, найденные при компиляции ядра и разборе исходного кода
- Более 10 ошибок, связанных с состоянием гонки



Спасибо за внимание

Вопросы?