LinuxTesting.org

Linux Device Driver Verification

Алексей Хорошилов
Вадим Мутилин

ISP RAS

Institute for System Programming of the Russian Academy of Sciences

Статический анализ

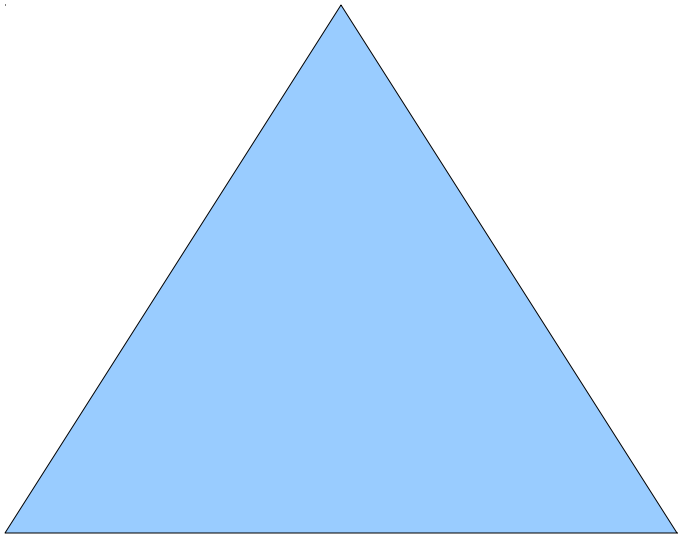
Выявляемые ошибки

- Ошибки общего вида
- Специфичные для предметной области

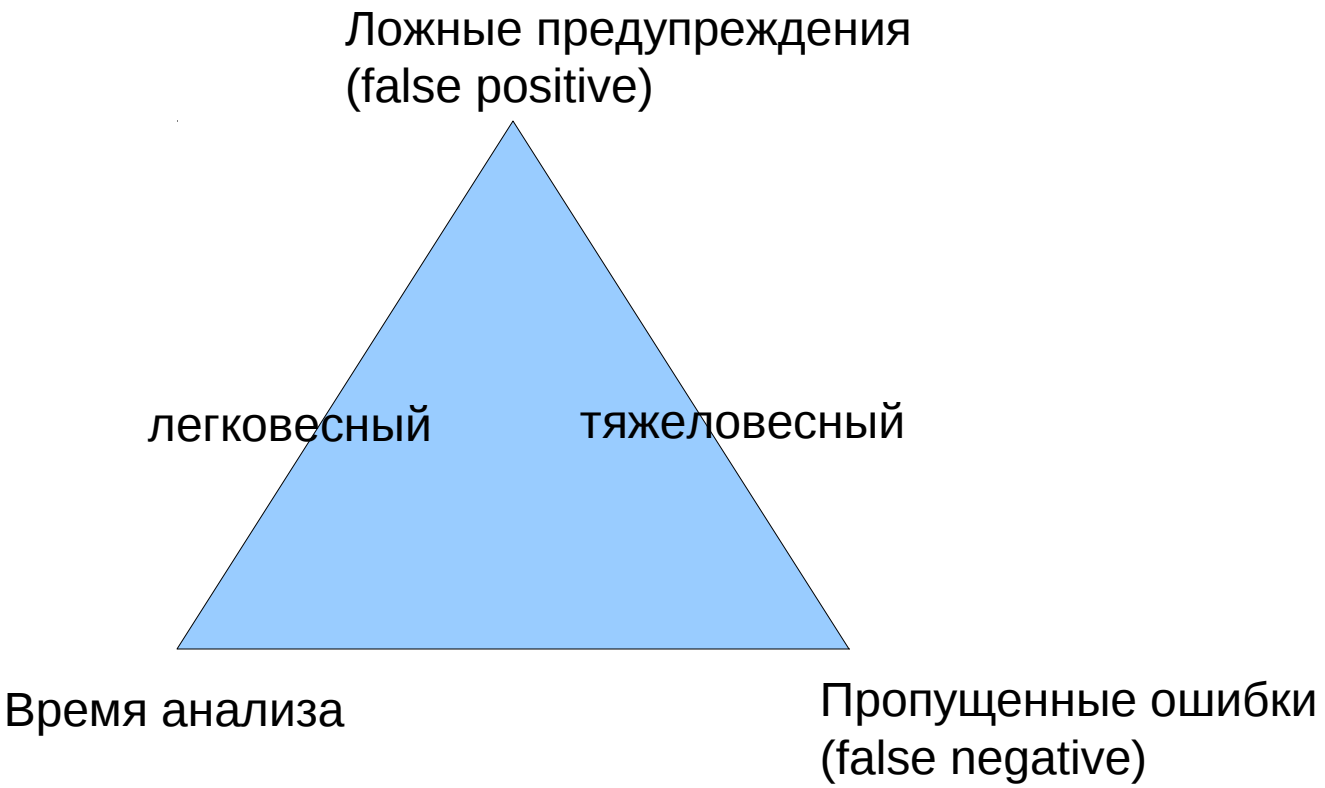
Качество анализа

- Ложные предупреждения (false positive)
- Пропущенные ошибки (false negative)

Статический анализ



Статический анализ



Легковесные инструменты

Коммерческие:

- Coverity
- KLOCwork
- CodeSonar
- PCLint
- Microsoft PREfast
- PolySpace
- SPARROW

Академические:

- Saturn
- FindBugs
- Splint
- ARCHER
- Coccinelle

Тяжеловесный анализ

На основе ижображения с http://engineer.org.in

Тяжеловесные инструменты

Коммерческие:

- Microsoft SDV

Академические:

- DDVerify (CMU)
- Avinux (U. Tuebingen)
- CBMC (U. Oxford)
- SATABS (U. Oxford)
- CPAChecker (U. Passua)
- BLAST
- ARMC (A. Rybalchenko)

Индустриальное качество

- Минимальное вмешательство человека
- Приемлемое качество верификации
  - Обнаружение реальных ошибок при
  - приемлемом % ложных предупреждений
  - И приемлемом времени работы
- Парсер языка Си совместимый с gcc/cl.exe
- Удобный анализ результатов

Microsoft Static Driver Verifier

We've created a number of things to do rich static analysis. We actually went out and **bought for a little over $30 million a company** that was in the business of building those kinds of tools, and we said now we want you to focus on applying these tools to large-scale software systems, the kind of system we have in the source code of Windows or Office, and see how far we can get on this.

…...

We call the system that does this kind of proof, it's a model-checking system. You describe the constraints, including things as simple as nobody should acquire the lock if they've already acquired it, nobody should release it if they haven't acquired it, certain things about the multi-threading aspect of the code that you want to make sure work very well. And you describe those things literally, in this case in the C code itself, and then the analyzer goes through and reduces the program, takes away anything that doesn't affect the path analysis that it's trying to go through to determine is there some path through the program that violates the constraints.

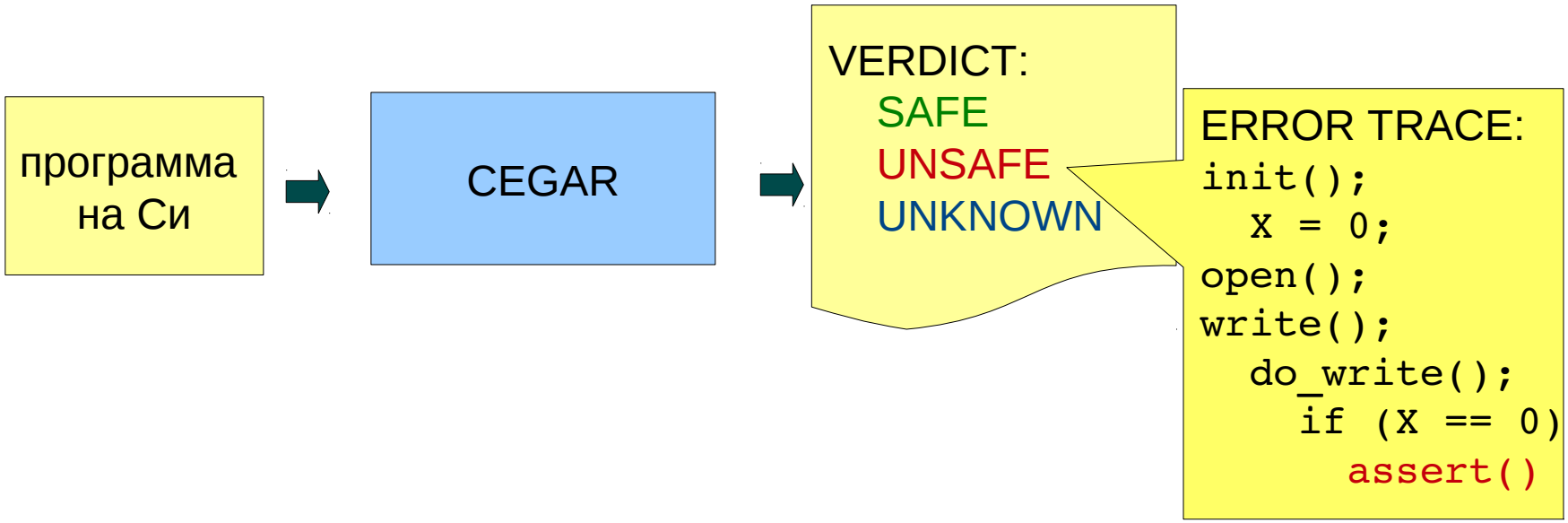
The initial domain we applied this in was in device drivers.

Bill Gates at
17th Annual ACM Conference on Object-Oriented Programming, Systems, Languages and Application

CEGAR

Counter-**E**xample **G**uided

Abstraction **R**efinement



Формулировка правил (1)

```
spinlock x;
int f(int y)
{
  lock(x);
  ...
  unlock(x);
  return y;
}
```

```
int x_locked = 0;
int f(int y)
{
  assert(x_locked == 0);
  x_locked = 1;
  ...
  assert(x_locked == 1);
  x_locked = 0;
  return y;
}
```

Формулировка правил (2)

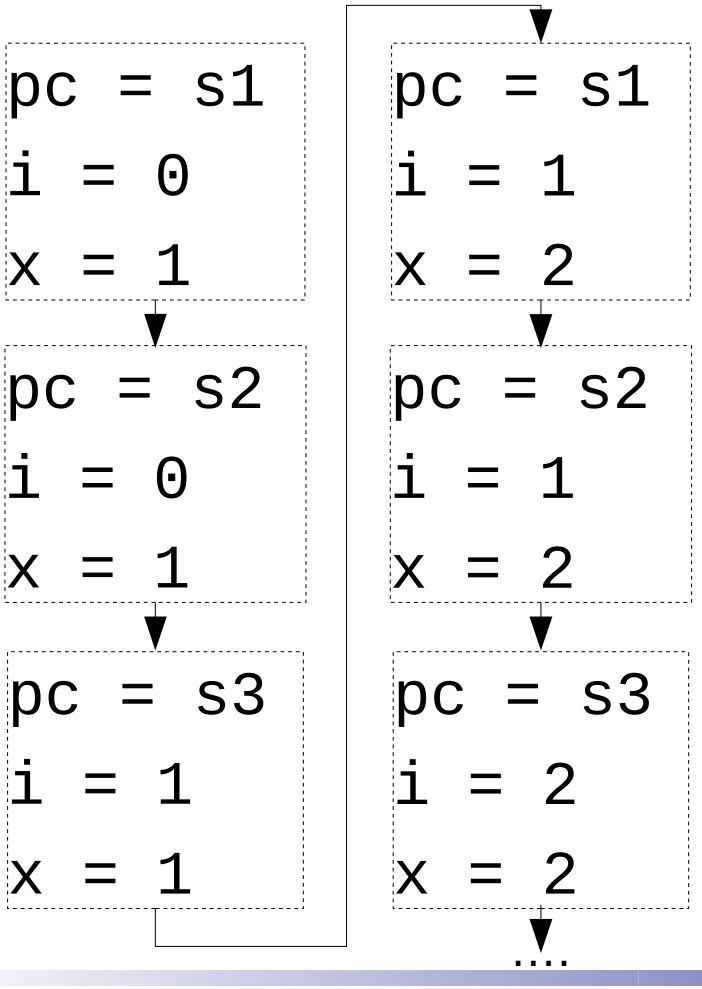
```
int main()
{
  ...
  assert(x > 0);



  ...
}
```

```
int main()
{
  ...
  if(!(x > 0)) {
    ERROR:
    abort();
  }
  ...
}
```

Пример

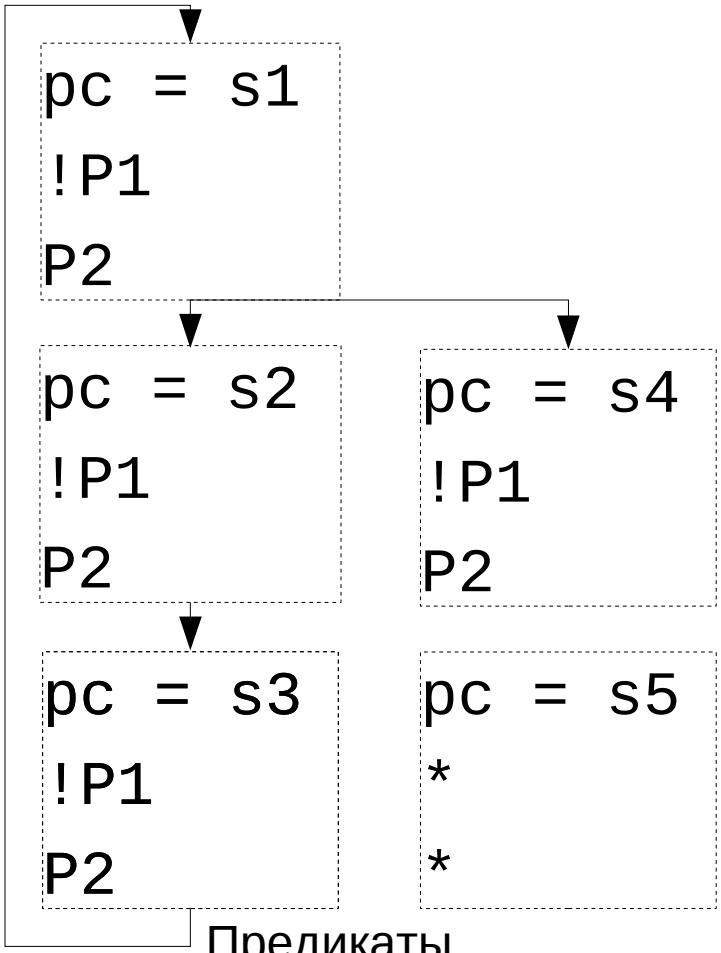
```
int main() {
  int i = 0;
  int x = 0;

  while (i < 100) {
    i++;
    x = x + i;
  }
  assert(x > 0);
}
```

Состояние программы




```
   int main() {
     int i = 0;
     int x = 1;

s1:  while (i < 100) {
s2:    i++;
s3:    x = x + i;
     }
s4:  if (!(x > 0))
s5:    ERROR: abort();
}
```

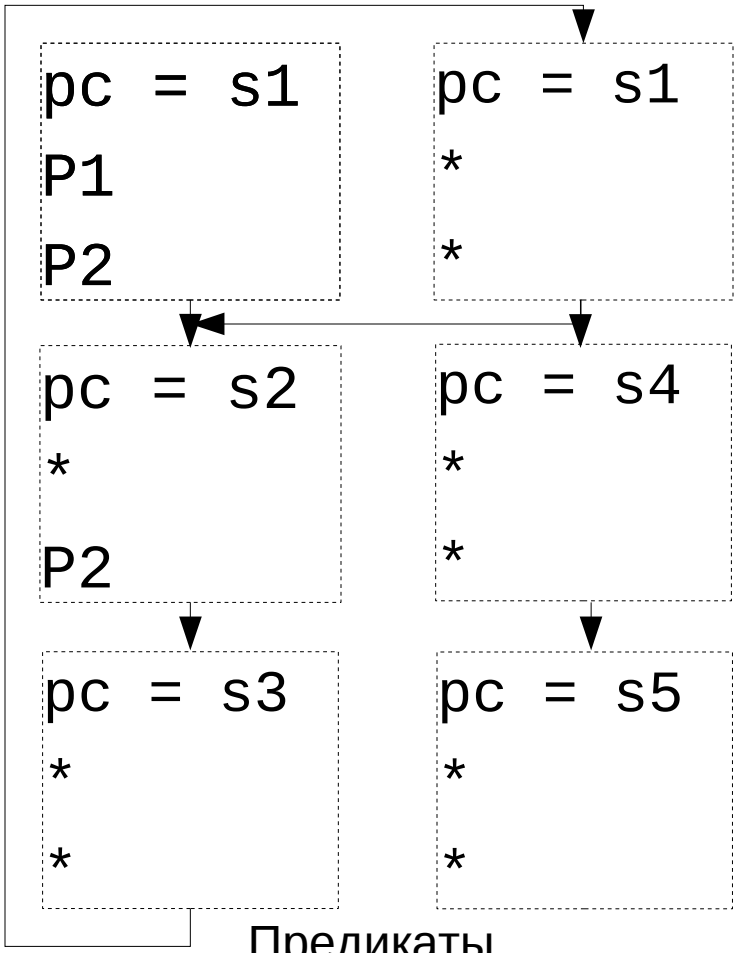
Предикатная абстракция



Предикаты
P1: i<0
P2: x>0

```
   int main() {
     int i = 0;
     int x = 1;

s1:  while (i < 100) {
s2:    i++;
s3:    x = x + i;
     }
s4:  if (!(x > 0))
s5:    ERROR: abort();
}
```

Предикатная абстракция (2)



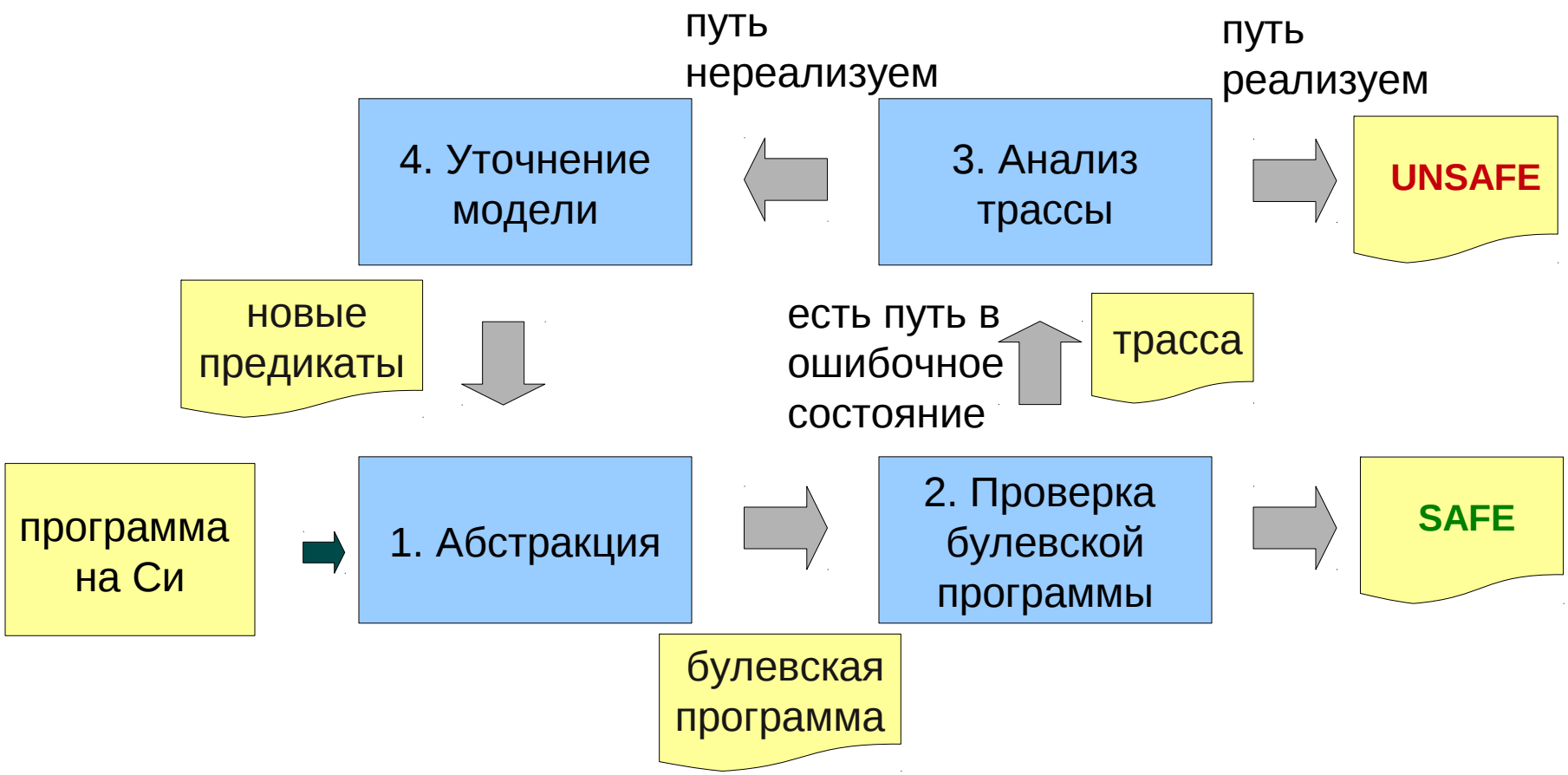

```
   int main() {
     int i = 0;
     int x = 1;

s1:  while (i < 100) {
s2:    i++;
s3:    x = x + i;
     }
s4:  if (!(x > 0))
s5:    ERROR: abort();
}
```

Предикаты

P1: i<100

P2: x>0

CEGAR


путь
нереализуем
путь
реализуем
4. Уточнение
модели
3. Анализ
трассы
UNSAFE
новые
предикаты
есть путь в
ошибочное
состояние
трасса
программа
на Си
1. Абстракция
2. Проверка
булевской
программы
SAFE
булевская
программа

Тяжеловесные инструменты

Коммерческие:

- **Microsoft SDV**

Академические:

- **DDVerify** (CMU)
- Avinux (U. Tuebingen)
- CBMC (U. Oxford)
- **SATABS** (U. Oxford)
- **CPAChecker** (U. Passua)
- **BLAST**
- **ARMC** (A. Rybalchenko)

Microsoft Static Driver Verifier

- Включен в состав Microsoft Windows Driver Developer Kit (DDK) в 2006 году
- Непрерывно развивается:
  - Виды интерфейсов:
    WDM (2006) → WDM, NDIS, KMDF (2010)
  - Число правил:
    43 (2006) → 200 (2010)
  - % ложных срабатываний:
    30%(2008) → 5% (2010)

Microsoft Static Driver Verifier

- Включен в состав Microsoft Windows Driver Developer Kit (DDK) в 2006 году
- Непрерывно развивается:
  - Время работы на одном драйвере:
    ??? → до 2-3 часов (2010)
  - TIMEOUT и MEMORY_LIMIT вердикты:
    5% для KMDF драйверов (за счет аннотаций)
  - Размер обрабатываемого драйвера:
    до 50 тыс. строк

Microsoft Static Driver Verifier

Результаты

- 33 critical bugs in the WDK sample drivers
- 53 critical bugs in kernel-mode drivers

Тяжеловесный анализ

На основе изображения с http://engineer.org.in

Тяжеловесные инструменты

Коммерческие:

- **Microsoft SDV**

Академические:

- **DDVerify (CMU)**
- **Avinux (U. Tuebingen)**
- CBMC (U. Oxford)
- SATABS (U. Oxford)
- CPAChecker (U. Passua)
- BLAST
- ARMC (A. Rybalchenko)

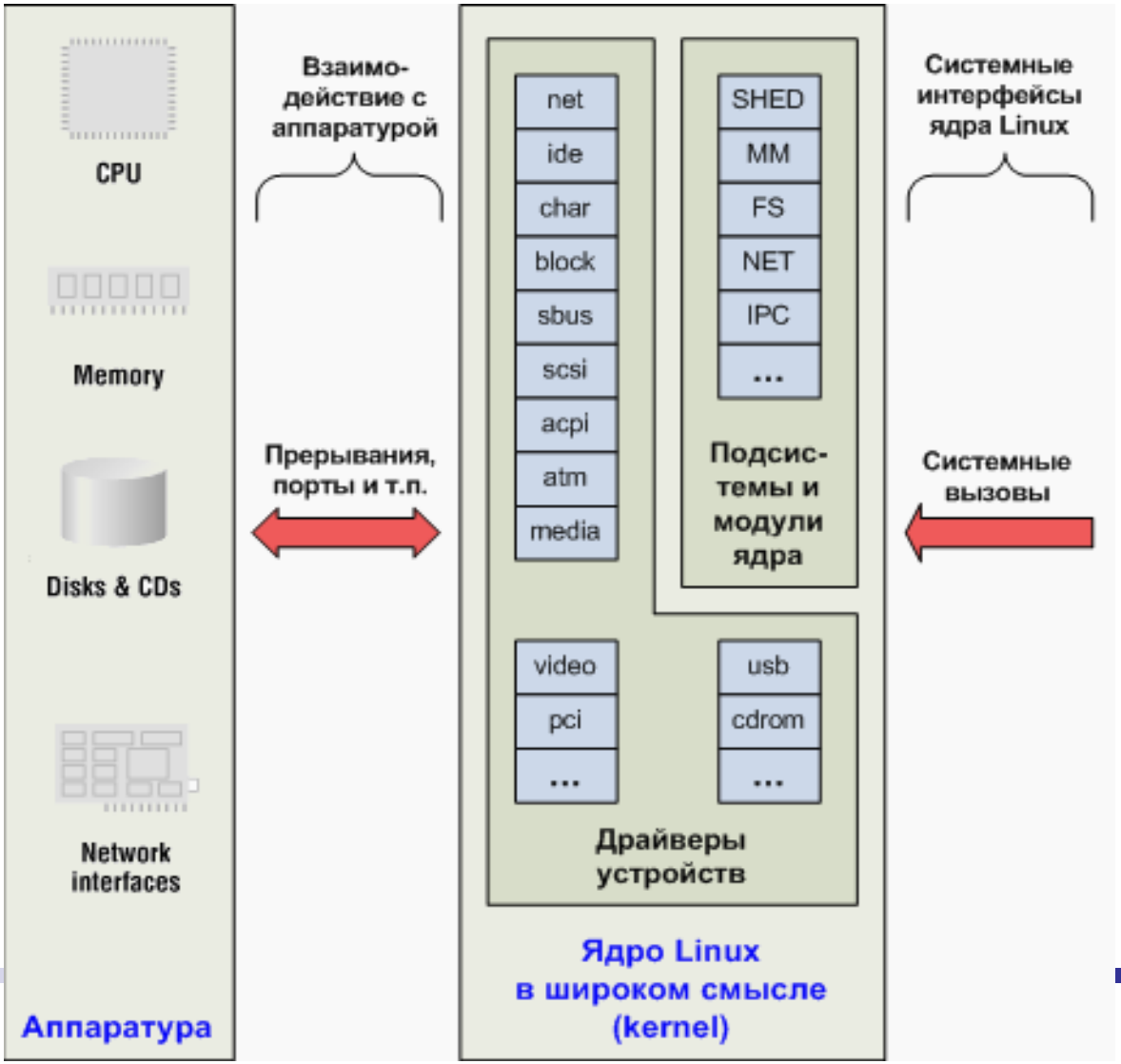
А почему драйвера?

- Основной источник проблем в ядре ОС
- Ограниченное использование сложных структур данных
- Практически нет арифметики с плавающей запятой
- Немного рекурсивных функций
- Небольшой размер драйверов

Особенности верификации драйверов

- Внутренняя асинхронность
- Вместо функции main асинхронные обработчики событий

Linux Kernel



Linux Device Driver

```
static struct pci_driver DAC960_pci_driver = {
        .name           = "DAC960",
        .id_table       = DAC960_id_table,
        .probe          = DAC960_Probe,
        .remove         = DAC960_Remove,
};

static int DAC960_init_module(void)
{
        int ret;

        ret =  pci_register_driver(&DAC960_pci_driver);
#ifdef DAC960_GAM_MINOR
        if (!ret)
                DAC960_gam_init();
#endif
        return ret;
}
...

module_init(DAC960_init_module);
module_exit(DAC960_cleanup_module);
```

Генерация псевдо-main

```
int main(int argc,char* argv[])
{
  init_module()
  for(;;) {
    switch(*) {
     case 0: driver_probe(*,*,*);break;
     case 1: driver_open(*,*);break;
     ...
    }
  }
  exit_module();
}
```

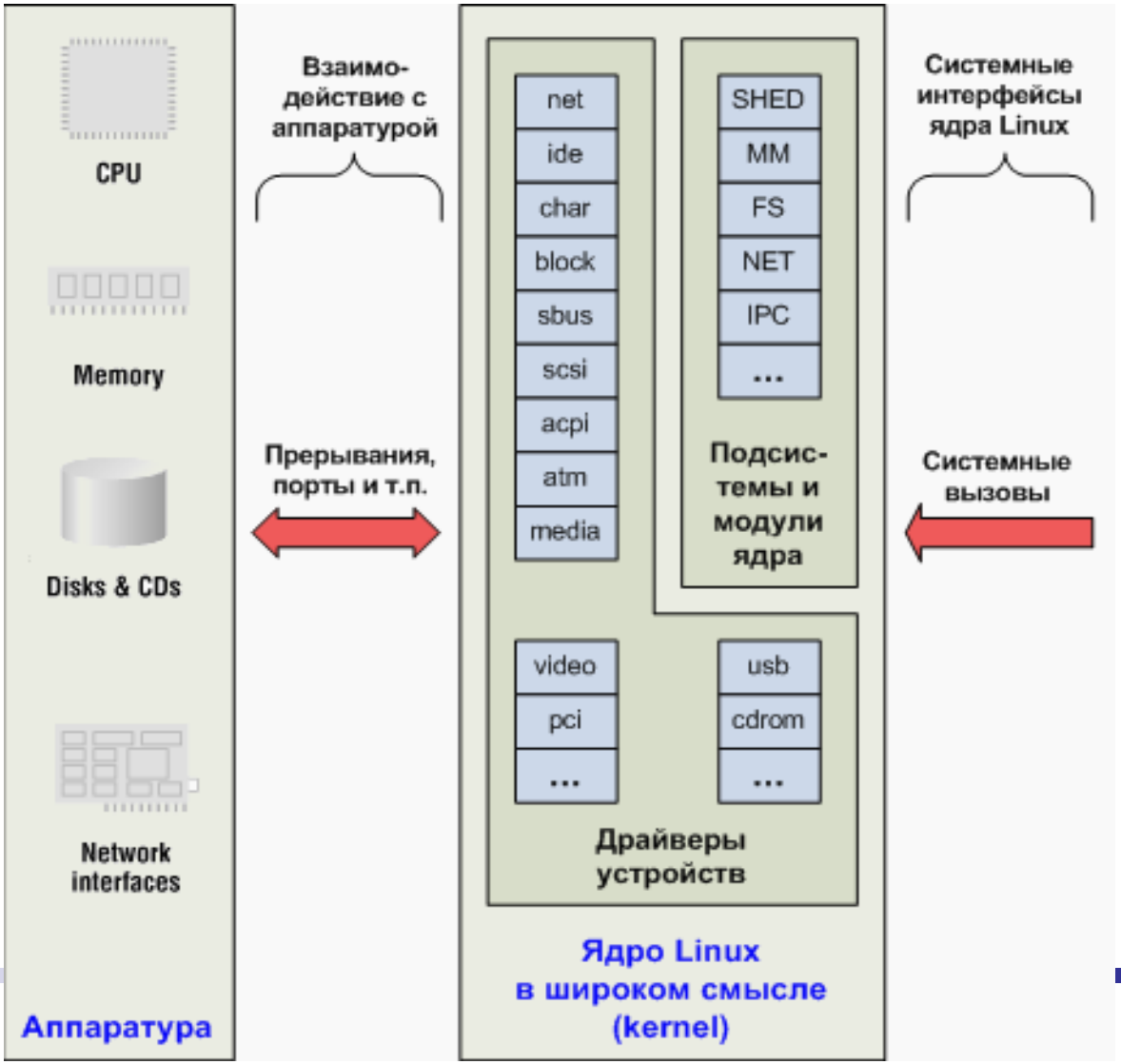
Генерация псевдо-main (2)

- Ограничения на порядок
  - open() после probe(), но до remove()
- Неявные ограничения
  - read() только, если open() был успешен
- и всё это уникально для каждого типа устройств

Особенности верификации драйверов

- Внутренняя асинхронность
- Вместо функции main асинхронные обработчики событий
- Нефиксированный интерфейс с ядром (Linux)

Linux Kernel



stable_api_nonsense.txt

This is being written to try to explain why Linux does not have a binary kernel interface, nor does it have a stable kernel interface. Please realize that this article describes the **in kernel** interfaces, not the kernel to userspace interfaces. The kernel to userspace interface is the one that application programs use, the syscall interface. That interface is **very** stable over time, and will not break.

Executive Summary

You think you want a stable kernel interface, but you really do not, and you don't even know it. What you want is a stable running driver, and you get that only if your driver is in the main kernel tree. You also get lots of other good benefits if your driver is in the main kernel tree, all of which has made Linux into such a strong, stable, and mature operating system which is the reason you are using it in the first place.

Greg Kroah-Hartman

Статистика по ядру Linux

- Статистика изменений в ядре [*]
  - 4.02 изменений в час
    (в среднем за 2005-2010)
  - Строки кода: 9 058 добавляется, 4 495 удаляется, 1 978 изменяется каждый день
    (в среднем за 2009-2010)
- Поддерживаемые архитектуры
  - Более 18

[*] Kroah-Hartman G, Corbet J, McPherson A (2010) Linux Kernel Development
http://www.linuxfoundation.org/docs/lf_linux_kernel_development_2010.pdf

Тяжеловесные инструменты

Коммерческие:

- **Microsoft SDV**

Академические:

- **DDVerify (CMU)**
- **Avinux (U. Tuebingen)**
- CBMC (U. Oxford)
- SATABS (U. Oxford)
- CPAChecker (U. Passua)
- BLAST
- ARMC (A. Rybalchenko)

Сравнение инструментов верификации драйверов

	SDV (Windows)	**Avinux** (Linux)	**DDVerify** (Linux)
Генерация модели окружения	по аннотации	ручная	для 3-х типов
Стандартная сборка	+	±	–
Независимость от версии ядра	±	+	–
Визуализация результатов	+	–	–
Добавление новых правил	+	+	±
Добавление новых инструментов	–	–	–

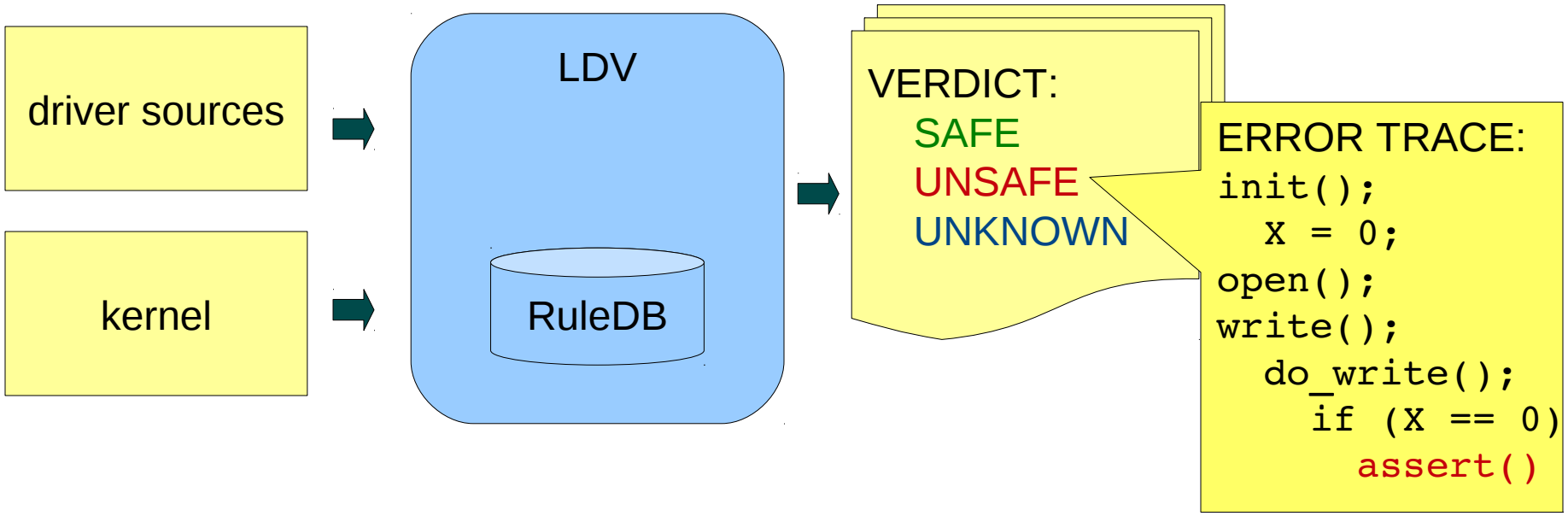
Linux Driver Verification

Стартовал силами ИСП РАН в 2007 году

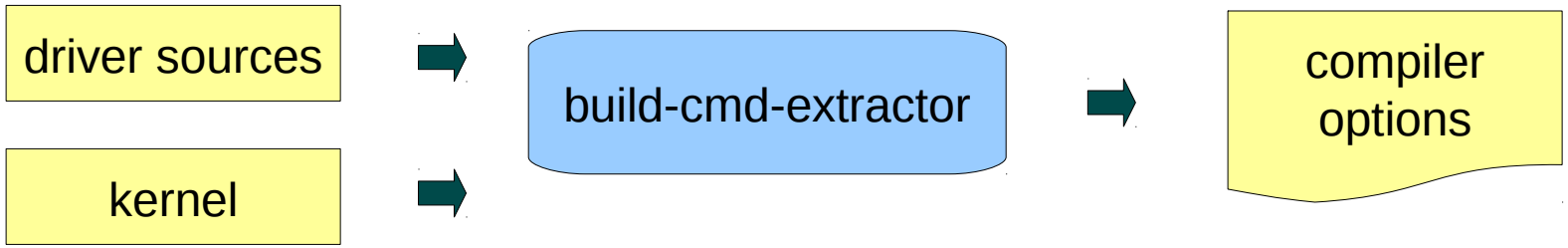
Цели:

- Разработать инструменты для обеспечения надежности драйверов Linux
- Построить открытую площадку индустриального уровня для анализа и развития инструментов верификации

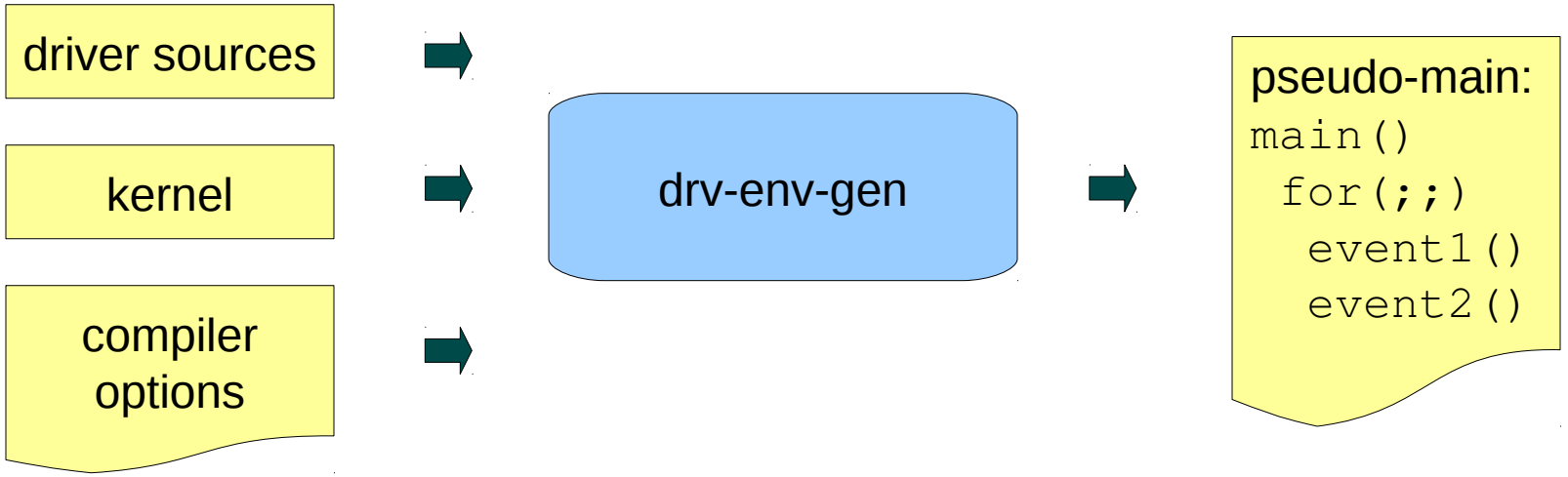
Схема работы LDV в целом



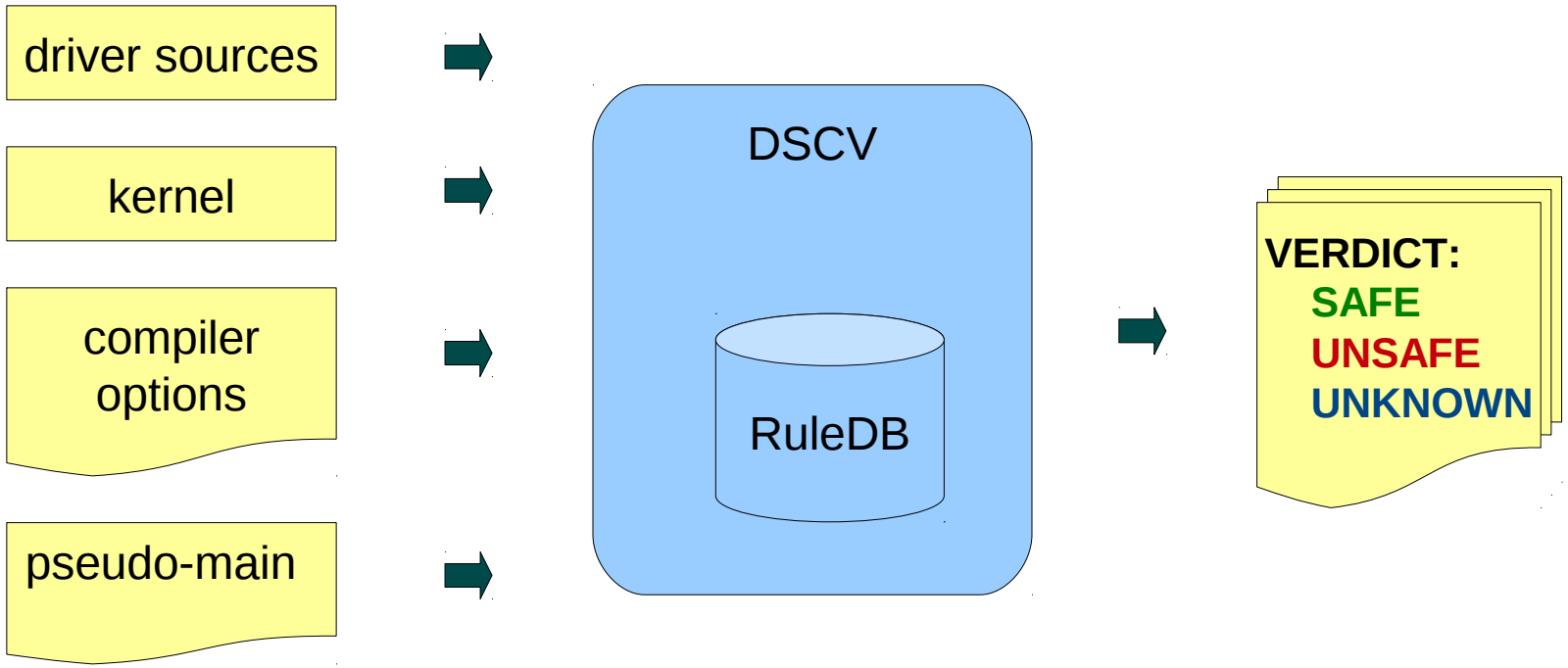
1. Извлечение информации об опциях сборки



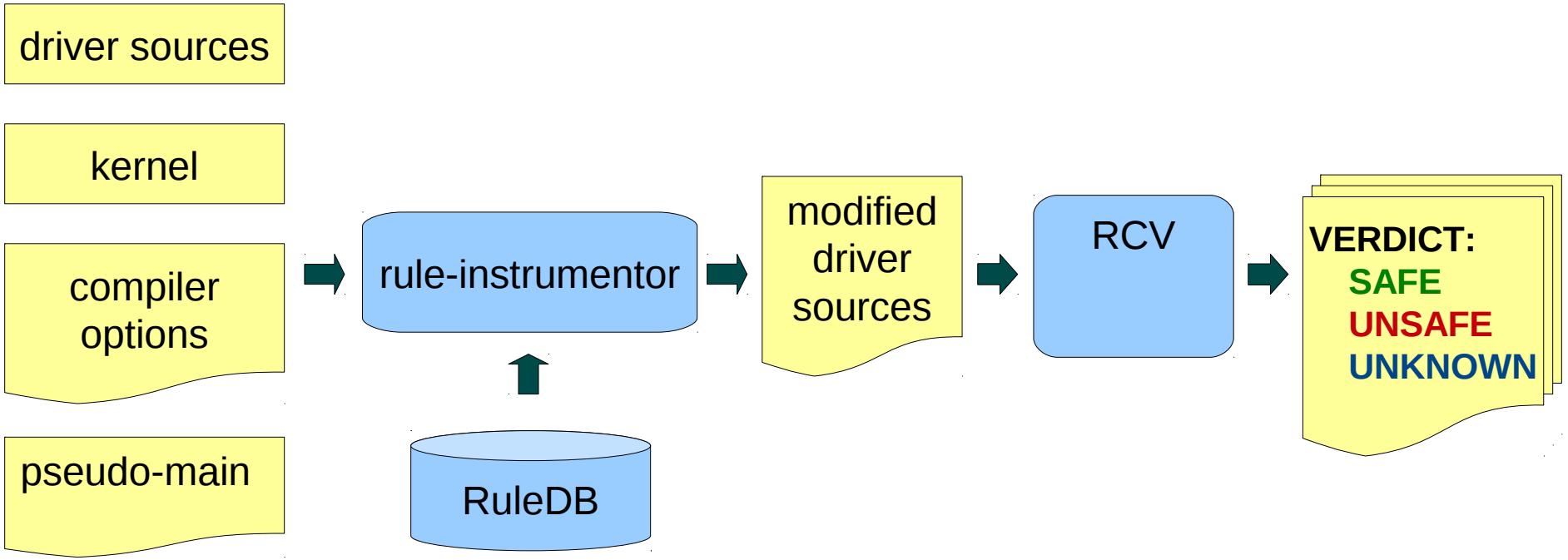
2. Генерация модели окружения



3. Верификация правил



Domain Specific C Verifier



Reachability C Verifier

В настоящее время две реализации:

- BLAST
- CPAChecker

Сравнение инструментов верификации драйверов

	SDV (Windows)	**Avinux** (Linux)	**DDVerify** (Linux)	**LDV** (Linux)
Генерация модели окружения	по аннотации	ручная	для 3-х типов	автома-тическая
Стандартная сборка	+	±	–	+
Независимость от версии ядра	±	+	–	+
Визуализация результатов	+	–	–	+
Добавление новых правил	+	+	±	+
Добавление новых инструментов	–	–	–	+

Текущие результаты

- База правил
  - Выявлено 80 правил
  - Формализовано 18 правил
- Найдено 25 ошибок в драйверах ядра

Обнаруженные ошибки

http://linuxtesting.org/results/ldv

- 25 ошибок, 16 принято и исправлено

Problems in Linux Kernel

This section contains information about problems in Linux kernel found within Linux Driver Verification program.

Click on a problem number for detailed description. Click on a column header to change the sorting order.

No.	Type	Brief	Added on	Accepted	Status
L0033	Crash	drivers/net/wireless/iwlwifi /iwl3945-base.c: mutex_unlock without mutex_lock	2010-12-14	commit 7ada88e5e5d7b465de8d0441b4a8d890a602074f	Fixed in 2.6.35
L0030	Crash	kernel/range.c: clean_sort_range() returns incorrect result for full array	2010-12-10	https://lkml.org/lkml/2010/11/5/264	Recognized as an error
L0029	Crash	drivers/media/radio/radio-gemtek-pci.c: mutex_lock imbalances	2010-09-14	commit fe643414dbf330d6d910e01edd48dd93dc6f2942, http://lkml.org/lkml/2009/7/13/320	Fixed in kernel 2.6.32
L0027	Crash	drivers/media/radio/radio-gemtek-pci.c: Double mutex_lock	2010-08-23	commit 3addbb8075c00e2a2408c192bd1002dead26b2aa	Fixed in kernel 2.6.32
L0026	Crash	drivers/net/3c505.c: Get spin_lock twice	2010-06-08	http://lkml.org/lkml/2010/6/7/139	Recognized as an error
L0025	Crash	drivers/mtd/mtd_blkdevs.c: Unsafe use of function module_put	2010-01-26	http://lkml.org/lkml/2010/1/12/246, commit 048d87199566663e4edc4880df3703c04bcf41d9	Fixed in kernel 2.6.35

Example of a report (L0009)

Date Wed, 7 Oct 2009 14:31:32 +0100

From Alan Cox <>

Subject Re: [BUG] isicom.c sleeping function called from invalid context

```
On Wed, 7 Oct 2009 17:15:14 +0000
Alexander Strakh <strakh@ispras.ru> wrote:
>       KERNEL_VERSION: 2.6.31
>       DESCRIBE:
> Driver drivers/char/isicom.c might sleep in atomic  context, because it calls
> tty_port_xmit_buf under spin_lock.
>
> ./drivers/char/isicom.c:
> 1307 static void isicom_hangup(struct tty_struct *tty)
> 1308 {
> ...
> 1315         spin_lock_irqsave(&port->card->card_lock, flags);
> 1316         isicom_shutdown_port(port);
> ...
>
> Path to might_sleep macro from isicom_hangup:
> 1. isicom_hangup calls spin_lock_irqsave (drivers/char/isicom.c:1315) and then
> calls isicom_shutdown_port.
> 2. isiscom_shutdown_port calls tty_port_free_xmit_buf at
> drivers/char/isicom.c:906
> 3. tty_port_free_xmit_buf calls mutex_lock at srivers/char/tty_port:48
>
> Found by Linux Driver Verification Project

Diagnosis is correct. I'll take a quick look at that one
```

Example of a report (L0014)

Date Mon, 12 Oct 2009 11:25:13 +0200 (CEST)

From Jiri Kosina <>

Subject Re: [BUG] hidraw.c: double mutex_lock

On Mon, 12 Oct 2009, iceberg wrote:

> KERNEL_VERSION: 2.6.31
> DESCRIBE:
> In driver ./drivers/hid/hidraw.c in function hidraw_read may be
> double mutex_lock:
>
> Path:
> 1. line 50: begin first iteration of "while(ret==0)"
> 2. line 52: first call to mutex_lock
> 3. inner loop "while (list->head == list->tail)" does not change state
> of mutex, because mutex_lock immediatelly follows mutex_unlock
> 4. if we go to the second iteration of "while(ret == 0)" in
> line 50 then there are second call to mutex_lock in line 52 (mutex
> aquired twice).
>
> Second iteration of loop "while(ret==0)" is possible if local variable
> ret is not changed at line 94: ret+=len - i.e. len==0;
> Variable len may be zero if hidraw_read is called with count==0 or
> list->buffer[list->tail].len == 0.

Good catch. I will fix that up by moving the mutex_lock() so that it's locked before the loop is entered.

Thanks,

--
Jiri Kosina
SUSE Labs, Novell Inc.

Тяжеловесный анализ

На основе изображения с http://engineer.org.in

SVACE vs LDV

Ядро 2.6.31.6, правило 32 (мьютексы)
9 реальных ошибок

SVACE	LDV
1 ошибка найдена	7 ошибок найдено
8 упущено	2 упущено
6 false positive (21)	1 false positive
~4 часов	~24 часов

Тяжеловесный анализ

На основе изображения с http://engineer.org.in

ldv-online

Online Linux Driver Verification Service (alpha)

Start Verification Verification History Rules

Start Verification

on x86_64 architecture

1. Ensure that drivers satisfy the following requirements:

- The driver is archived using gzip or bzip2 and has one of the following extensions: .tar.bz2, tar.gz, .tgz
- Archive should contain:
  o Makefile (written to be compiled with the kernel)
  + obj-m is mandatory
  o Sources needed by Makefile
- Archive should not contain generated files left from builds

2. Upload driver.

3. Wait for results.

Browse...

Start Verification

ldv-online (2)

Verification Report

Driver: test-0032-wl12xx-unsafe.tar.bz2
Timestamp: 2011-01-19 20:51:12
Verification architecture: x86_64

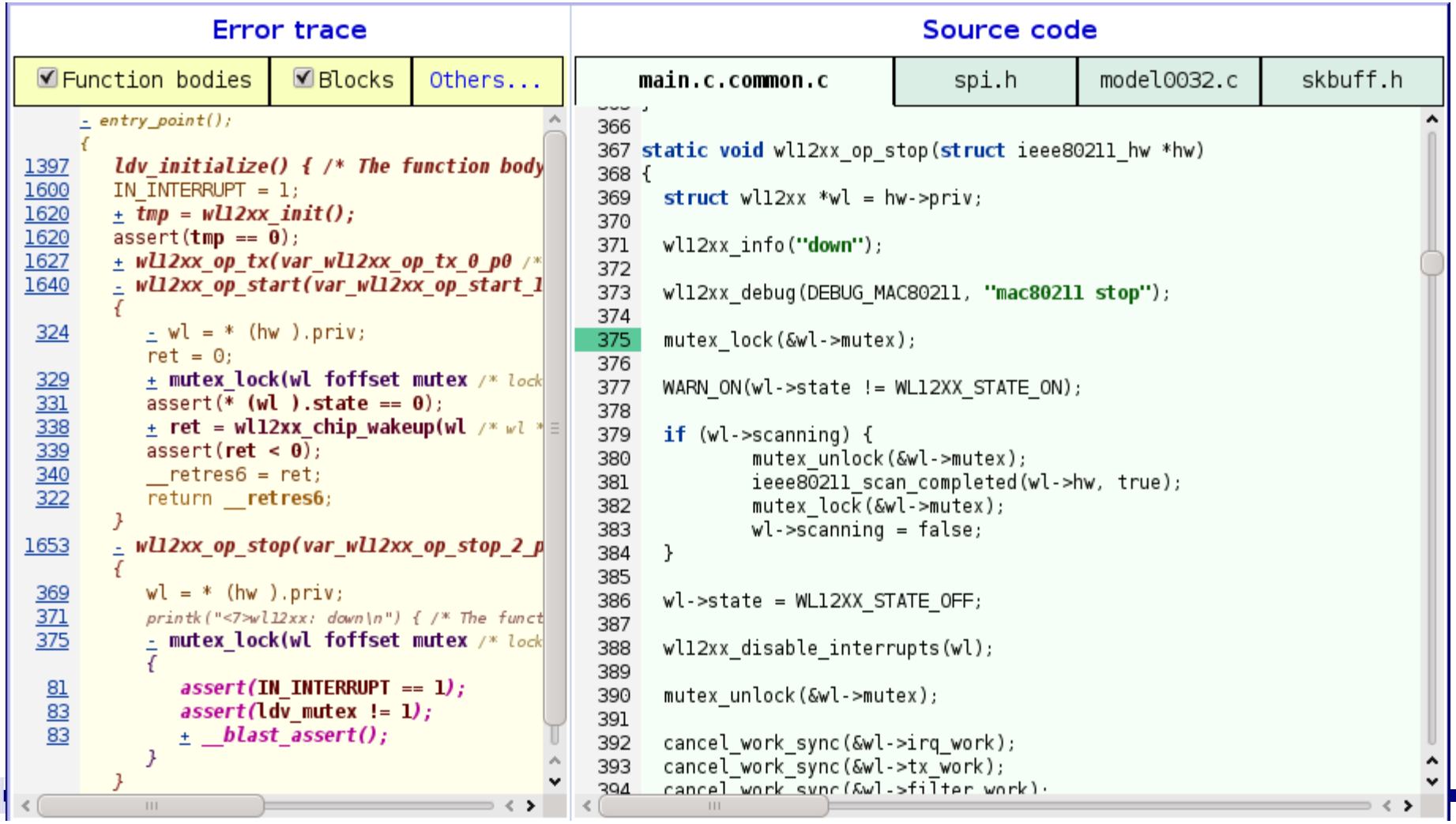
You can see **verification verdict** for each rule and linux kernel. Verdict may be:

- ***Safe*** - there is no mistakes for the given linux kernel and rule.
- ***Unsafe*** - driver may contain an error. You can see the error trace by clicking on the "Unsafe" link for the corresponding linux kernel and rule.
- ***Build failed*** - your driver is not compatible with the given linux kernel. In this case you may see the compile error trace by clicking on the "more details" link.
- ***Unknown*** - tools can not determine whether your driver *Safe* or *Unsafe*.
- ***Queued*** - the driver waits for the turn to verification.

7%

linux-2.6.32.12	
Rule	**Verdict**
Mutex lock/unlock	Unsafe
NOIO allocation under usb_lock	Safe
Module get/put	
PCI pool create/destroy, alloc/free	Queued
Delay in probe_irq on/off	Queued
Memory allocation inside spinlocks	Queued
Linked list double add	Queued
Usb alloc/free urb	Queued
Spinlocks lock/unlock	Queued

Визуализация трассы



Направления развития

- Масштабируемость
- Распараллеливание
- Расширение набора правил
- Качество анализа
- Альтернативные инструменты
- Увеличение скорости
- Улучшения в пользовательском интерфейсе

ISP RAS

LinuxTesting.org

Приглашаем к сотрудничеству!

khoroshilov@linuxtesting.org

Алексей Хорошилов

Вадим Мутилин

http://www.linuxtesting.org/project/ldv