VPT 2017

Fifth International Workshop on

Verification and Program Transformation

April 29th, 2017, Uppsala, Sweden

Software Model Checking of Linux Device Drivers

Alexey Khoroshilov
khoroshilov@ispras.ru

LinuxTesting.org

ISP RAS

Institute for System Programming of the Russian Academy of Sciences

Linux Verification Center

LinuxTesting.org

Institute for System Programming of Russian Academy of Sciences
VERIFICATION CENTER
OF THE OPERATING SYSTEM
Linux
SUS
LSB
POSIX

founded in 2005

- User Space Model Based Testing
- Application Binary/Program Interface Stability
- **Linux Driver Verification Program**
- Linux File System Verification
- Deductive Verification of Operating Systems

Why Device Drivers?

- There is a demand
  - A big source of issues in kernel
- Realistic for software model checking
  - Limited usage of complex data structures
  - Very limited usage of floating point arithmetics
  - Small number of recursive functions
  - Limited size of drivers
- Academic interest
  - Big set of uniform but diverse source code

Commit Analysis(*)

- All patches in stable trees (2.6.35 – 3.0) for 1 year:
  - 26 Oct 2010 – 26 Oct 2011
- 3101 patches overall

(*) Khoroshilov A.V., Mutilin V.S., Novikov E.M. Analysis of typical faults in Linux operating system drivers. Proceedings of the Institute for System Programming of RAS, volume 22, 2012, pp. 349-374. (In Russian)
http://ispras.ru/ru/proceedings/docs/2012/22/isp_22_2012_349.pdf
Raw data: http://linuxtesting.org/downloads/ldv-commits-analysis-2012.zip

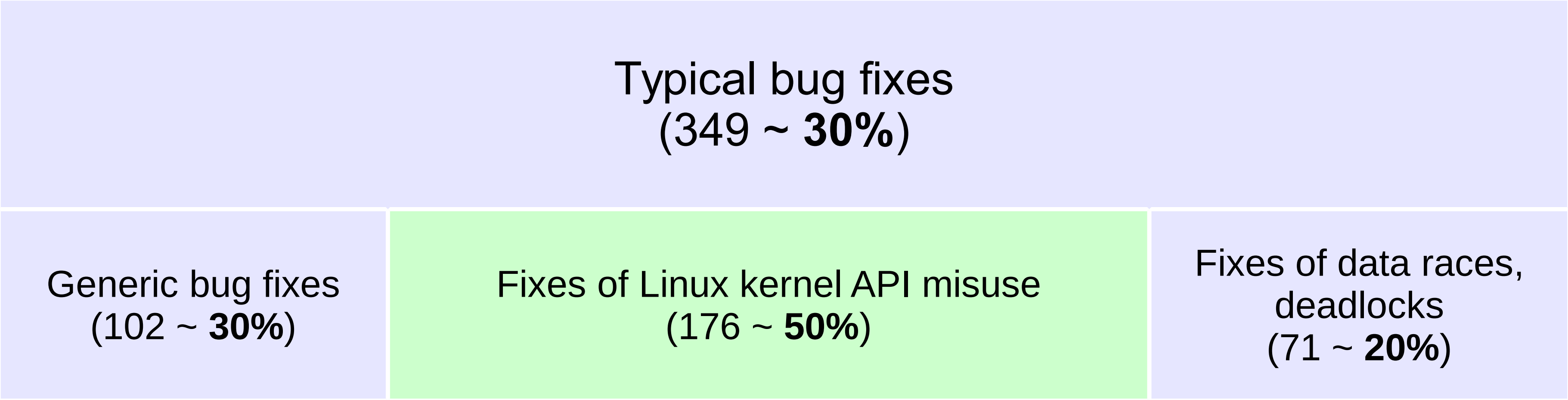
Commit Analysis

- All patches in stable trees (2.6.35 – 3.0) for 1 year:
  - 26 Oct 2010 – 26 Oct 2011
- 3101 patches overall

<table>
<tr><td colspan="2">Unique commits to drivers
(1503 ~ <b>50%</b>)</td></tr>
<tr><td>Support of a new functionality
(321 ~ <b>20%</b>)</td><td>Bug fixes
(1182 ~ <b>80%</b>)</td></tr>
</table>

Commit Analysis

- All patches in stable trees (2.6.35 – 3.0) for 1 year:
  - 26 Oct 2010 – 26 Oct 2011
- 3101 patches overall



Typical bug fixes
(349 ~ **30%**)

Generic bug fixes
(102 ~ **30%**)

Fixes of Linux kernel API misuse
(176 ~ **50%**)

Fixes of data races, deadlocks
(71 ~ **20%**)

Taxonomy of Typical Bugs

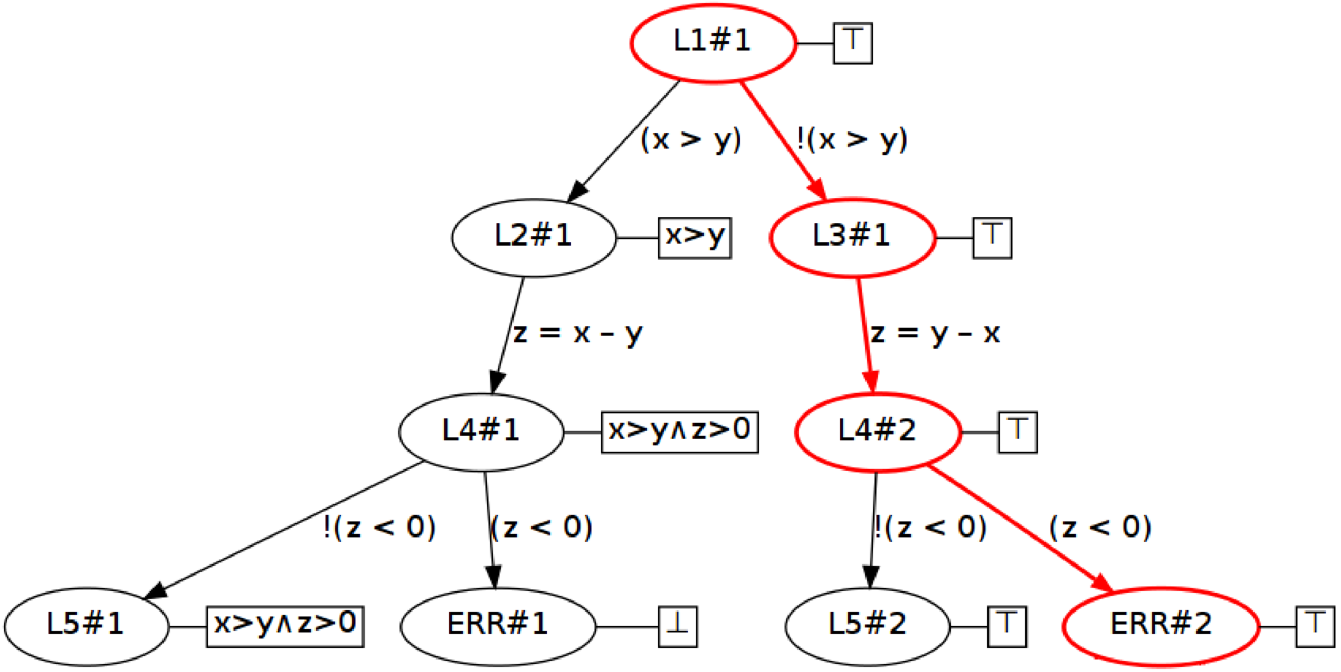
Rule classes	Types	Number of bug fixes	Percents	Cumulative total percents
Correct usage of the Linux kernel API (176 ~ 50%)	Alloc/free resources	32	~18%	~18%
	Check parameters	25	~14%	~32%
	Work in atomic context	19	~11%	~43%
	Uninitialized resources	17	~10%	~53%
	Synchronization primitives in one thread	12	~7%	~60%
	Style	10	~6%	~65%
	Network subsystem	10	~6%	~71%
	USB subsystem	9	~5%	~76%
	Check return values	7	~4%	~80%
	DMA subsystem	4	~2%	~82%
	Core driver model	4	~2%	~85%
	Miscellaneous	27	~15%	100%
Generic (102 ~ 30%)	NULL pointer dereferences	31	~30%	~30%
	Alloc/free memory	24	~24%	~54%
	Syntax	14	~14%	~68%
	Integer overflows	8	~8%	~76%
	Buffer overflows	8	~8%	~83%
	Uninitialized memory	6	~6%	~89%
	Miscellaneous	11	~11%	100%
Synchronization (71 ~ 20%)	Races	60	~85%	~85%
	Deadlocks	11	~15%	100%

Reach ability

Software Model Checking

- Reachability problem

entry point



error location

Verification Tools World

```
int main(int argc,char* argv[])
{
  ...
  other_func(var);
  ...
}
```

```
void other_func(int v)
{
    ...
    assert( x != NULL);
}
```

Device Driver World

```
int usbpn_open(struct net_device *dev) { ... };
int usbpn_close(struct net_device *dev) { ... };
struct net_device_ops usbpn_ops = {
      .ndo_open = usbpn_open, .ndo_stop = usbpn_close
};
int usbpn_probe(struct usb_interface *intf, const struct usb_device_id *id){
      dev->netdev_ops = &usbpn_ops;
      err = register_netdev(dev);
}
void usbpn_disconnect(struct usb_interface *intf){...}

struct usb_driver usbpn_struct = {
      .probe = usbpn_probe, .disconnect = usbpn_disconnect,
};
int __init usbpn_init(void){ return usb_register(&usbpn_struct);}
void __exit usbpn_exit(void){usb_deregister(&usbpn_struct );}

module_init(usbpn_init);
module_exit(usbpn_exit);
```

No explicit calls to init/exit procedures

Device Driver World

```
int usbpn_open(struct net_device *dev) { ... };
int usbpn_close(struct net_device *dev) { ... };
struct net_device_ops usbpn_ops = {
      .ndo_open = usbpn_open, .ndo_stop = usbpn_close
};
int usbpn_probe(struct usb_interface *intf, const struct usb_device_id *id){
      dev->netdev_ops = &usbpn_ops;
      err = register_netdev(dev);
}
void usbpn_disconnect(struct usb_interface *intf){...}

struct usb_driver usbpn_struct = {
      .probe = usbpn_probe, .disconnect = usbpn_disconnect,
};
int __init usbpn_init(void){ return usb_register(&usbpn_struct);}
void __exit usbpn_exit(void){usb_deregister(&usbpn_struct );}

module_init(usbpn_init);
module_exit(usbpn_exit);
```

Callback interface procedures registration

No explicit calls to init/exit procedures

Device Driver World

```
int usbpn_open(struct net_device *dev) { ... };
int usbpn_close(struct net_device *dev) { ... };
struct net_device_ops usbpn_ops = {
      .ndo_open = usbpn_open, .ndo_stop = usbpn_close
};
int usbpn_probe(struct usb_interface *intf, const struct usb_device_id *id){
      dev->netdev_ops = &usbpn_ops;
      err = register_netdev(dev);
}
void usbpn_disconnect(struct usb_interface *intf){...}

struct usb_driver usbpn_struct = {
      .probe = usbpn_probe, .disconnect = usbpn_disconnect,
};
int __init usbpn_init(void){ return usb_register(&usbpn_struct);}
void __exit usbpn_exit(void){usb_deregister(&usbpn_struct );}

module_init(usbpn_init);
module_exit(usbpn_exit);
```

Callback interface procedures registration

No explicit calls to init/exit procedures

Active Driver Environment Model (1)

```
int main(int argc,char* argv[])
{
  usbpn_init()
  for(;;) {
    switch(*) {
     case 0: usbpn_probe(*,*,*);break;
     case 1: usbpn_open(*,*);break;
     ...
    }
  }
  usbpn_exit();
}
```

Active Driver Environment Model (2)

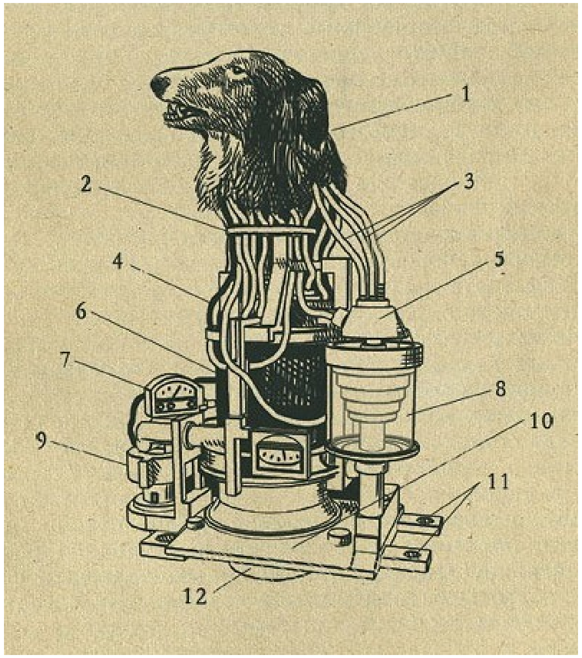
- Order limitation
    - open() after probe(), but before remove()
- Implicit limitations
    - read() only if open() succeed
- and it is specific for each class of drivers

Active Driver Environment Model (3)

- Precise
  - Complete - to avoid missing bugs
  - Correct - to avoid false alarms

Active Driver Environment Model (3)

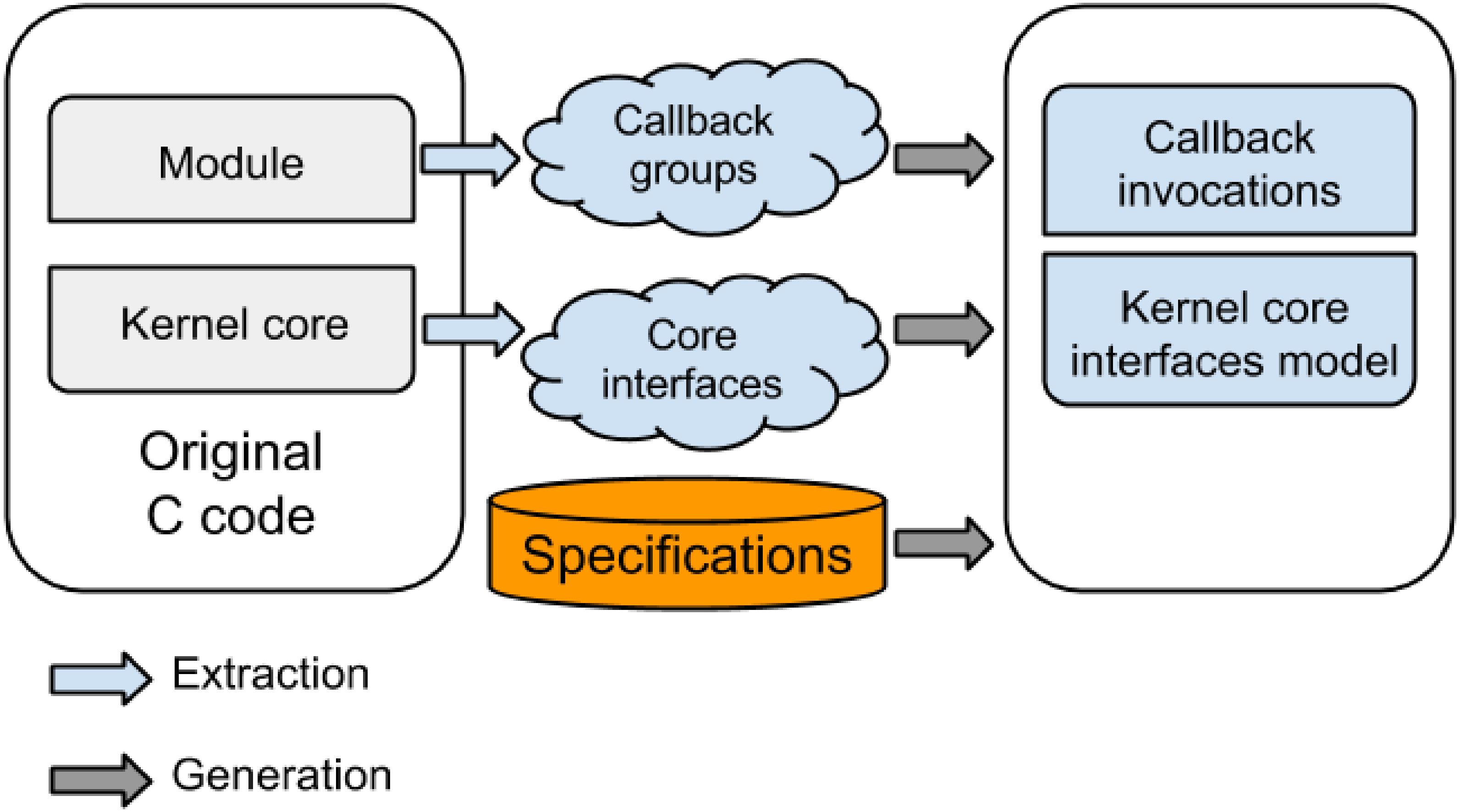
- Precise
  - Complete - to avoid missing bugs
  - Correct - to avoid false alarms
- Simple enough



Active Driver Environment Model (4)

- >20 000 callback registrations
- of 800 different types
- changes across Linux kernel versions

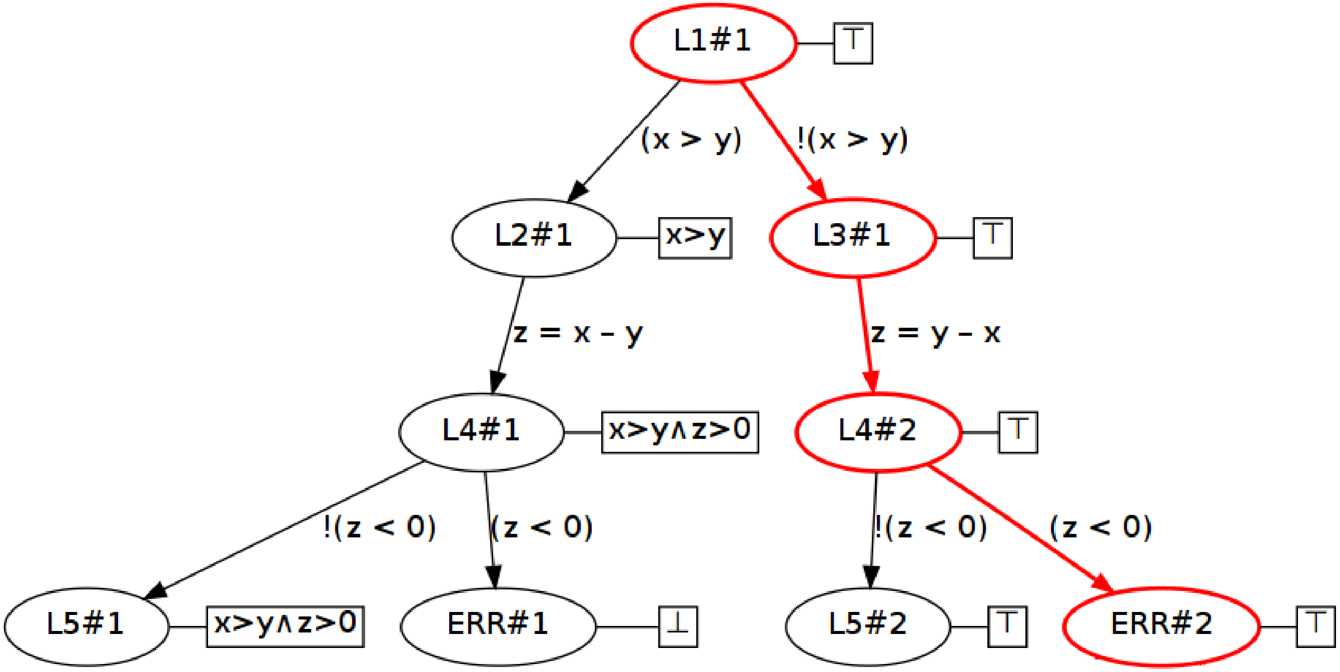
LDV Approach - EMG



Model Checking and Linux Kernel

- Reachability problem

entry point ✓ DONE



error location

Error Location?

```
int f(int y)
{
struct urb *x;

  x = usb_alloc_urb(0,GFP_KERNEL);
  ...
  usb_free_urb(x);

  return y;
}
```

Error Location?

```
int f(int y)
{
struct urb *x;

  x = usb_alloc_urb(0,GFP_KERNEL); // allocate new URB
  ...
  usb_free_urb(x); // deallocate URB: assert(x is NULL or previously allocated URB)

  return y;
}


  …
  // after module exit: assert( all allocated URBs are deallocated)
```

Instrumentation

```
int f(int y)
{
struct urb *x;

  x = usb_alloc_urb(0,GFP_KERNEL);

  ...

  usb_free_urb(x);

  return y;
}
```

```
set URBS = empty;

int f(int y)
{
struct urb *x;

  x = usb_alloc_urb();
  add(URBS, urb);
  ...
  assert(contains(URBS, x));
  usb_free_urb(x);
  remove(URBS, urb);

  return y;
}
  …
  // after module exit
  assert(is_empty(URBS));
```

Aspect-Oriented Notation

```
// Model state: set of allocated URBs
set URBS = empty;

// Model functions
struct urn * ldv_usb_alloc_urb(void)
{
  void *urb;
  urb = ldv_alloc();
  if (urb) {
    add(URBS, urb);
  }
  return urb;
}

void ldv_usb_free_urb(struct urb *urb)
{
  if (urb) {
    remove(URBS, urb);
  }
}
```

```
// Pointcut declarations
pointcut USB_ALLOC_URB:
  call(struct urb *usb_alloc_urb(int, gfp_t));
pointcut USB_FREE_URB:
  call(void usb_free_urb(struct urb *));

// Update model state
around: USB_ALLOC_URB {
  return ldv_usb_alloc_urb();
}
around: USB_FREE_URB {
  ldv_usb_free_urb($arg1);
}

// Assertions
before: USB_FREE_URB {
  assert(contains(URBS, $arg1));
}
after: MODULE_EXIT {
  assert(is_empty(URBS));
}
```

Instrumentation

```
int f(int y)
{
struct urb *x;

  x = usb_alloc_urb(0,GFP_KERNEL);
  ...
  usb_free_urb(x);

  return y;
}
```

➡

```
int f(int y)
{
struct urb *x;

  x = ldv_usb_alloc_urb();
  ...
  assert(contains(URBS, x));
  ldv_usb_free_urb(x);

  return y;
}

  ...
  // after module exit
  assert(is_empty(URBS));
```

Rule Instrumentor

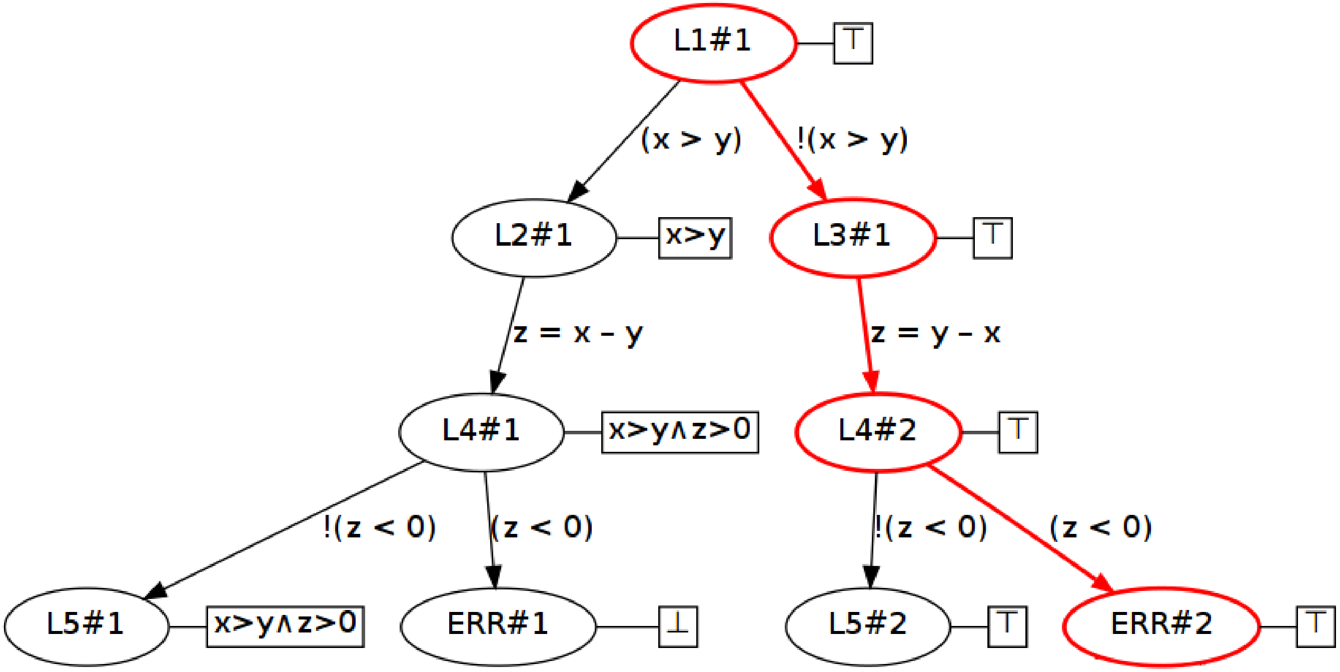
- **CIF** – C Instrumentation Framework
  - gcc-based aspect-oriented programming tool for C language
  - available under GPLv3:

    http://forge.ispras.ru/projects/cif

Model Checking and Linux Kernel

- Reachability problem

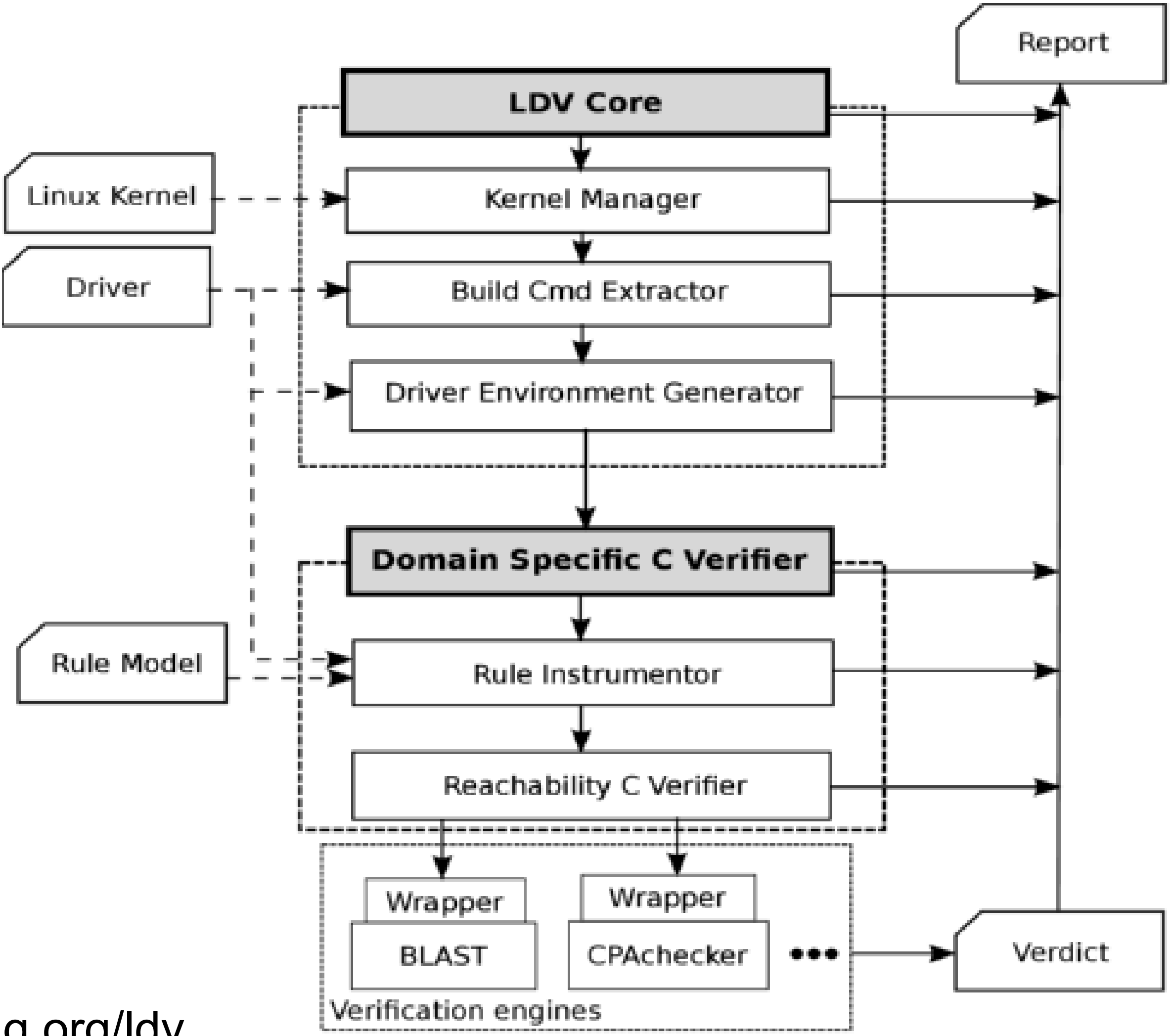
entry point ✓ DONE



error location

✓ DONE

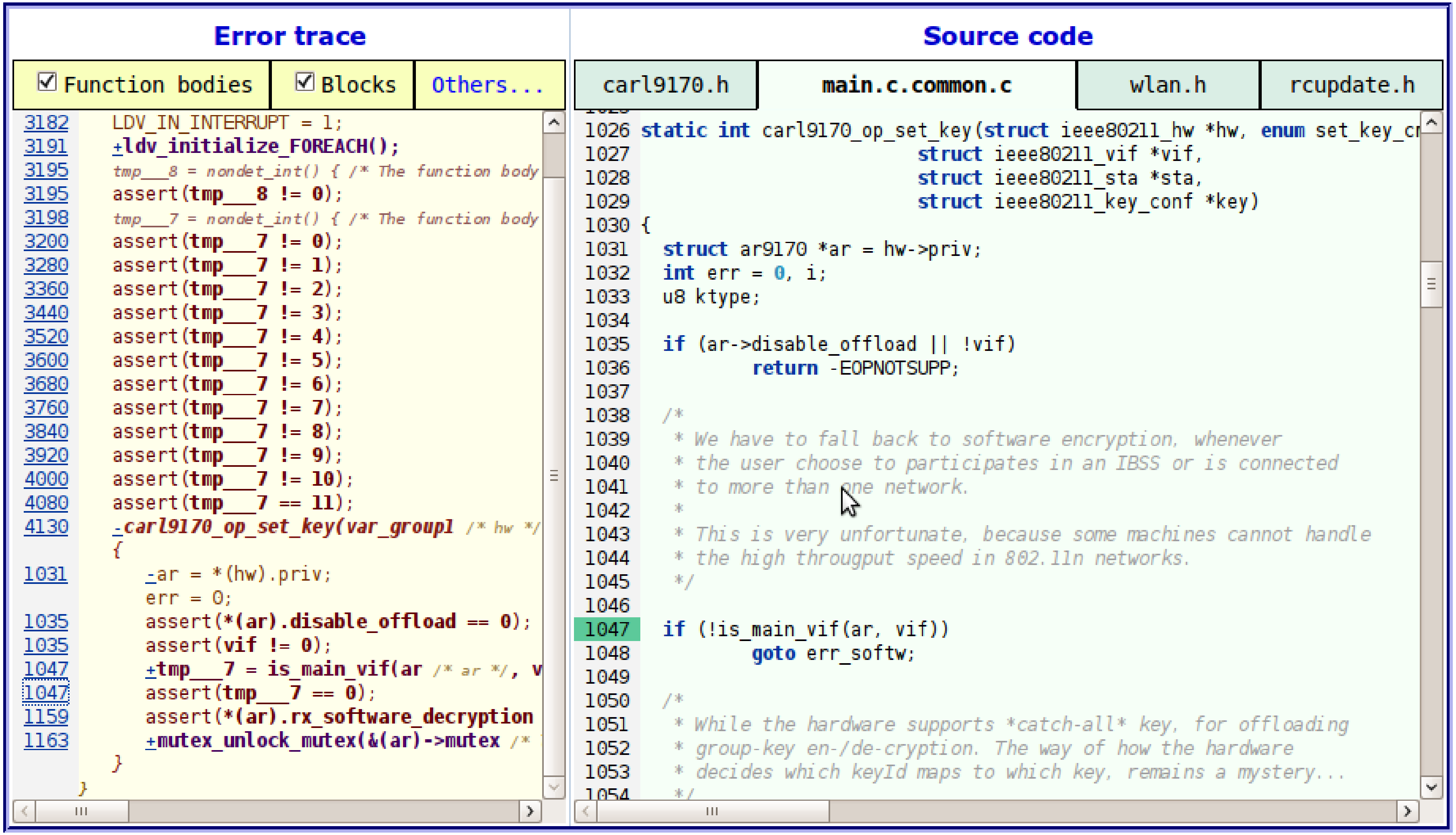
Linux Driver Verification



http://linuxtesting.org/ldv

Error Trace Visualizer

Rule: Mutex lock/unlock



```
3182    LDV_IN_INTERRUPT = 1;
3191    +ldv_initialize_FOREACH();
3195    tmp___8 = nondet_int() { /* The function body
3195    assert(tmp___8 != 0);
3198    tmp___7 = nondet_int() { /* The function body
3200    assert(tmp___7 != 0);
3280    assert(tmp___7 != 1);
3360    assert(tmp___7 != 2);
3440    assert(tmp___7 != 3);
3520    assert(tmp___7 != 4);
3600    assert(tmp___7 != 5);
3680    assert(tmp___7 != 6);
3760    assert(tmp___7 != 7);
3840    assert(tmp___7 != 8);
3920    assert(tmp___7 != 9);
4000    assert(tmp___7 != 10);
4080    assert(tmp___7 == 11);
4130    -carl9170_op_set_key(var_group1 /* hw */
        {
1031        -ar = *(hw).priv;
            err = 0;
1035        assert(*(ar).disable_offload == 0);
1035        assert(vif != 0);
1047        +tmp___7 = is_main_vif(ar /* ar */, v
1047        assert(tmp___7 == 0);
1159        assert(*(ar).rx_software_decryption
1163        +mutex_unlock_mutex(&(ar)->mutex /*
        }
    }
```

```
static int carl9170_op_set_key(struct ieee80211_hw *hw, enum set_key_cr
                               struct ieee80211_vif *vif,
                               struct ieee80211_sta *sta,
                               struct ieee80211_key_conf *key)
{
	struct ar9170 *ar = hw->priv;
	int err = 0, i;
	u8 ktype;

	if (ar->disable_offload || !vif)
		return -EOPNOTSUPP;

	/*
	 * We have to fall back to software encryption, whenever
	 * the user choose to participates in an IBSS or is connected
	 * to more than one network.
	 *
	 * This is very unfortunate, because some machines cannot handle
	 * the high througput speed in 802.11n networks.
	 */

	if (!is_main_vif(ar, vif))
		goto err_softw;

	/*
	 * While the hardware supports *catch-all* key, for offloading
	 * group-key en-/de-cryption. The way of how the hardware
	 * decides which keyId maps to which key, remains a mystery...
	 */
```

Recent Experiment

All modules Linux 3.14 (4368), 32 rules

CPAChecker BAM VA+PA, EMG conservative

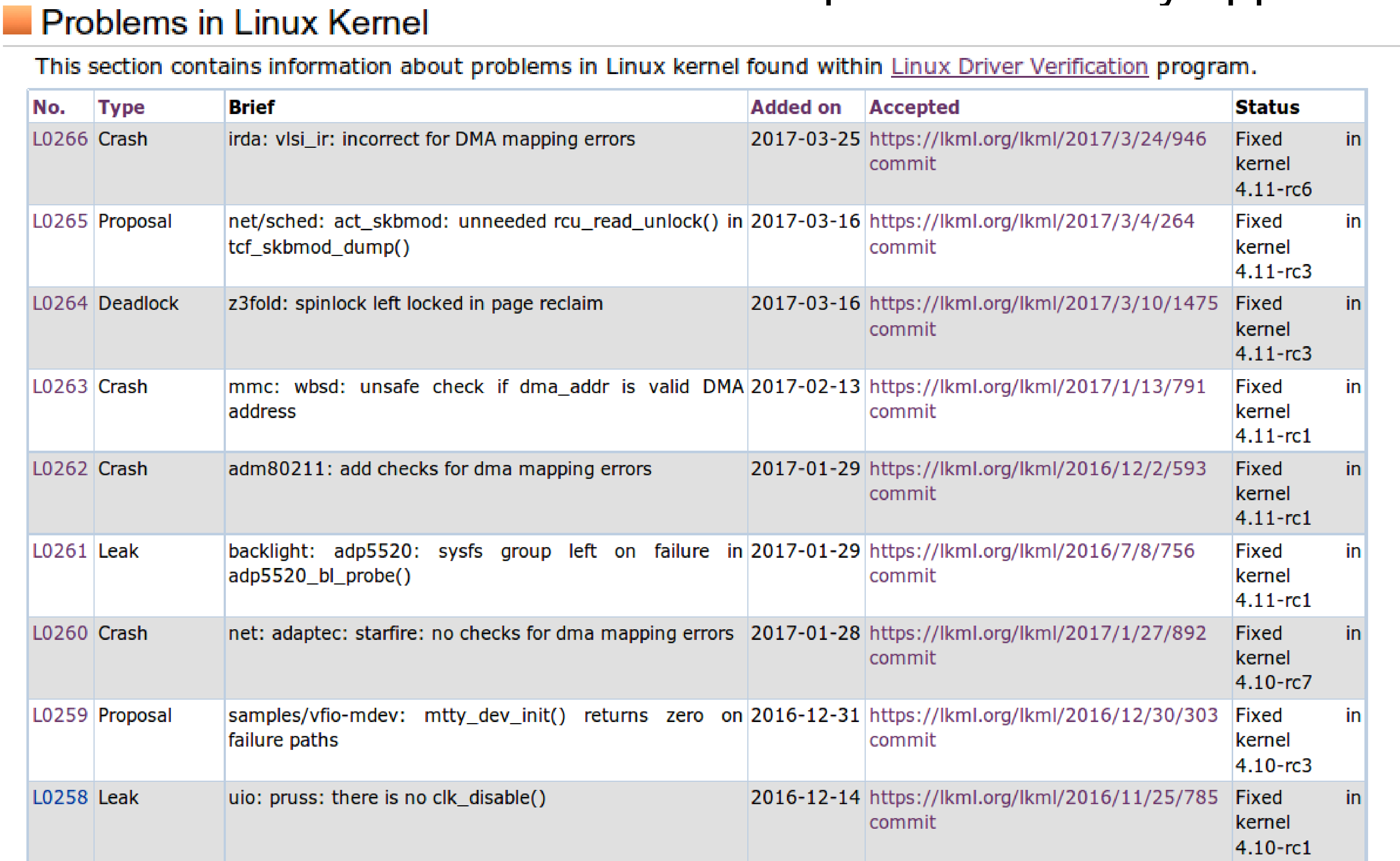
- 417 UNSAFEs
  - 116 TRUE 28%
  - 301 FALSE 72%

Kernel model:	97	32%
Verifier:	82	27%
Rules:	46	15%
EMG:	41	14%
LDV:	35	11%

Bugs Found

http://linuxtesting.org/results/ldv

>250 patches already applied



Problems in Linux Kernel

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

No.	Type	Brief	Added on	Accepted	Status
L0266	Crash	irda: vlsi_ir: incorrect for DMA mapping errors	2017-03-25	https://lkml.org/lkml/2017/3/24/946 commit	Fixed in kernel 4.11-rc6
L0265	Proposal	net/sched: act_skbmod: unneeded rcu_read_unlock() in tcf_skbmod_dump()	2017-03-16	https://lkml.org/lkml/2017/3/4/264 commit	Fixed in kernel 4.11-rc3
L0264	Deadlock	z3fold: spinlock left locked in page reclaim	2017-03-16	https://lkml.org/lkml/2017/3/10/1475 commit	Fixed in kernel 4.11-rc3
L0263	Crash	mmc: wbsd: unsafe check if dma_addr is valid DMA address	2017-02-13	https://lkml.org/lkml/2017/1/13/791 commit	Fixed in kernel 4.11-rc1
L0262	Crash	adm80211: add checks for dma mapping errors	2017-01-29	https://lkml.org/lkml/2016/12/2/593 commit	Fixed in kernel 4.11-rc1
L0261	Leak	backlight: adp5520: sysfs group left on failure in adp5520_bl_probe()	2017-01-29	https://lkml.org/lkml/2016/7/8/756 commit	Fixed in kernel 4.11-rc1
L0260	Crash	net: adaptec: starfire: no checks for dma mapping errors	2017-01-28	https://lkml.org/lkml/2017/1/27/892 commit	Fixed in kernel 4.10-rc7
L0259	Proposal	samples/vfio-mdev: mtty_dev_init() returns zero on failure paths	2016-12-31	https://lkml.org/lkml/2016/12/30/303 commit	Fixed in kernel 4.10-rc3
L0258	Leak	uio: pruss: there is no clk_disable()	2016-12-14	https://lkml.org/lkml/2016/11/25/785 commit	Fixed in kernel 4.10-rc1

Lessons Learnt

Lessons Learnt

- Language features support

gcc extensions, e.g. zero-length arrays

```
struct line {
       int length;
       char contents[0];
     };
```

etc.

Lessons Learnt

- Language features support
- No premature UNKNOWN

```
Error: Unsupported C feature (recursion) in line 60858:
tmp = gma_power_begin( tmp24, tmp25);
(CallstackTransferRelation.getAbstractSuccessors)
```

Bug Finder vs. Safe Prover

Lessons Learnt

- Language features support
- No premature UNKNOWN
- Efficiently ignore irrelevant details

Ten of thousands irrelevant transitions vs. dozens of relevant ones

Lessons Learnt

- Language features support
- No premature UNKNOWN
- Efficiently ignore irrelevant details
- Engineering matters

BLAST

Berkeley

Lazy

Abstraction

Software
Verification

Tool

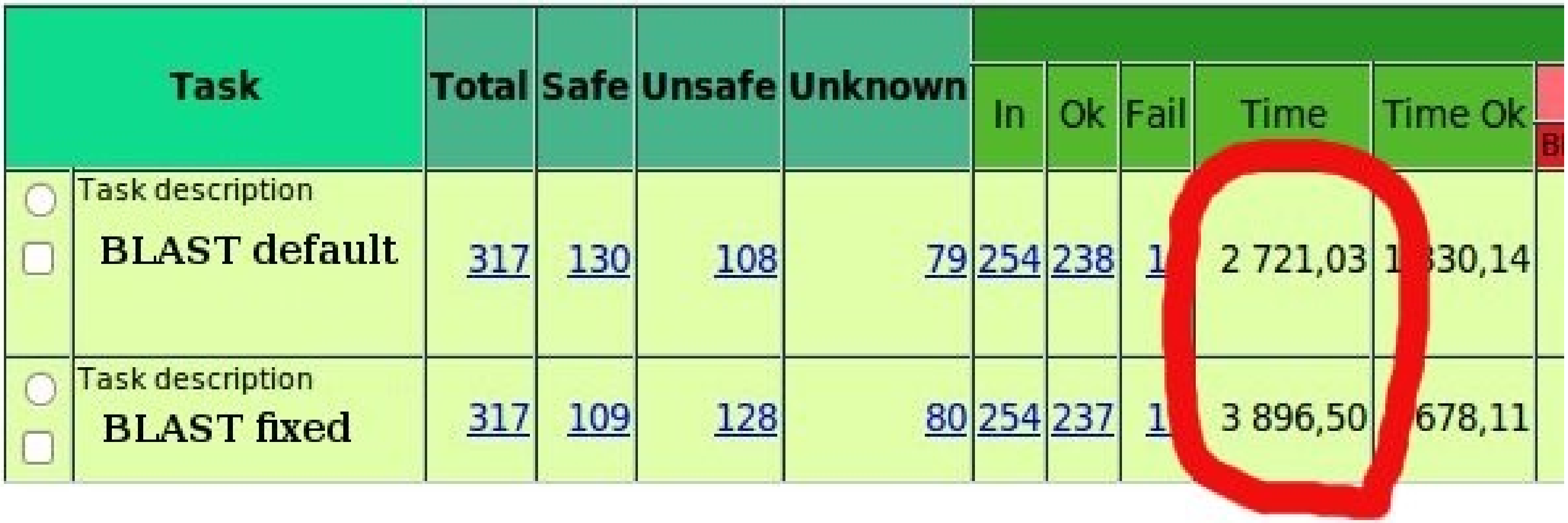
BLAST is a software model checker for C programs.

It uses counterexample-driven automatic abstraction refinement to construct an abstract model which is model checked for safety properties.

Fix Bugs in BLAST Algorithm

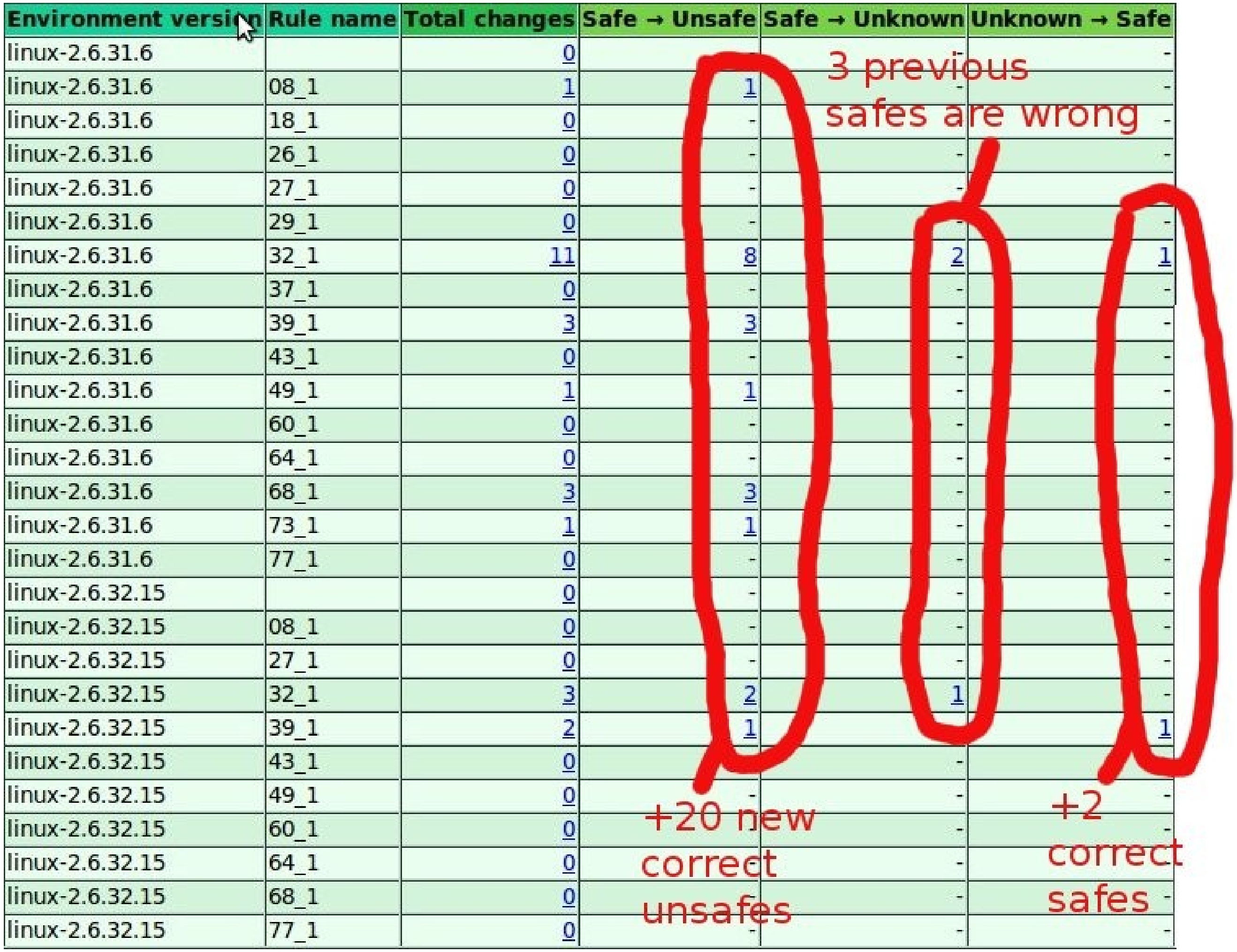
Run BLAST with lattice option,
which enables Configurable Program Analysis (CPA)

- BLAST default – without fix
- BLAST fixed – CPA implementation fixed



Task	Total	Safe	Unsafe	Unknown						
					In	Ok	Fail	Time	Time Ok	Bl
○ □ Task description **BLAST default**	317	130	108	79	254	238	1	2 721,03	1 [illegible]30,14	
○ □ Task description **BLAST fixed**	317	109	128	80	254	237	1	3 896,50	[illegible]678,11	

Fix Bugs in BLAST (Details)



Environment version	Rule name	Total changes	Safe → Unsafe	Safe → Unknown	Unknown → Safe
linux-2.6.31.6		0	-	-	-
linux-2.6.31.6	08_1	1	1	-	-
linux-2.6.31.6	18_1	0	-	-	-
linux-2.6.31.6	26_1	0	-	-	-
linux-2.6.31.6	27_1	0	-	-	-
linux-2.6.31.6	29_1	0	-	-	-
linux-2.6.31.6	32_1	11	8	2	1
linux-2.6.31.6	37_1	0	-	-	-
linux-2.6.31.6	39_1	3	3	-	-
linux-2.6.31.6	43_1	0	-	-	-
linux-2.6.31.6	49_1	1	1	-	-
linux-2.6.31.6	60_1	0	-	-	-
linux-2.6.31.6	64_1	0	-	-	-
linux-2.6.31.6	68_1	3	3	-	-
linux-2.6.31.6	73_1	1	1	-	-
linux-2.6.31.6	77_1	0	-	-	-
linux-2.6.32.15		0	-	-	-
linux-2.6.32.15	08_1	0	-	-	-
linux-2.6.32.15	27_1	0	-	-	-
linux-2.6.32.15	32_1	3	2	1	-
linux-2.6.32.15	39_1	2	1	-	1
linux-2.6.32.15	43_1	0	-	-	-
linux-2.6.32.15	49_1	0	-	-	-
linux-2.6.32.15	60_1	0	-	-	-
linux-2.6.32.15	64_1	0	-	-	-
linux-2.6.32.15	68_1	0	-	-	-
linux-2.6.32.15	77_1	0	-	-	-

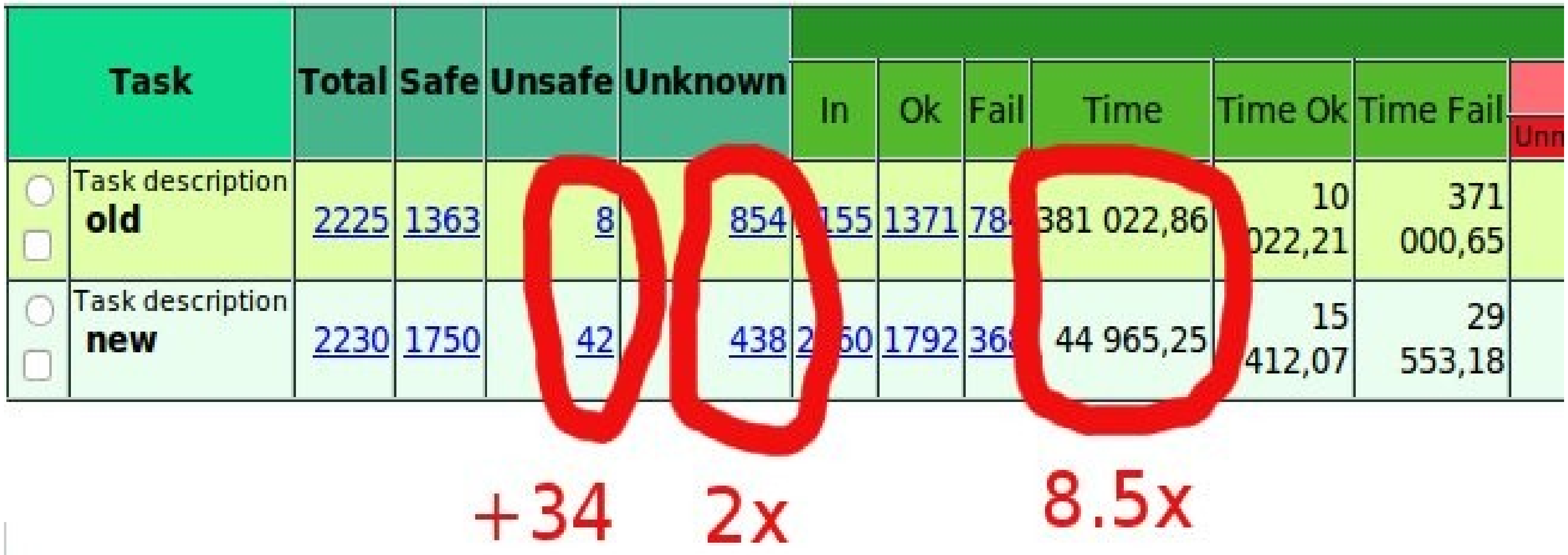
3 previous
safes are wrong

+20 new
correct
unsafes

+2
correct
safes

Speed Up BLAST

Old – BLAST 2.5, 2008
New – BLAST 2.6, 2011



	Task	Total	Safe	Unsafe	Unknown	In	Ok	Fail	Time	Time Ok	Time Fail	Unn
○ □	Task description **old**	2225	1363	8	854	2155	1371	784	381 022,86	10 022,21	371 000,65	
○ □	Task description **new**	2230	1750	42	438	2160	1792	368	44 965,25	15 412,07	29 553,18	

ISPRAS BLAST 2.6 Release Notes

Speedup ranges from **8 times** on small-sized programs to **30 times** on medium-sized programs

- Logarithmic algorithm for useful-blocks (significantly speedup of trace analysis)
- Improved integration with SMT solvers
  - efficient string concatenation
  - caching of converted formulae
  - optimization of CVC3 options for BLAST use cases
- Formulae normalization moved to solvers since solvers do it faster
- Alias analysis speedup
  - must-aliases are handled separately and faster than may-aliases
  - removed unnecessary debug prints from alias iteration (even a check for debug flag impacts performance significantly in hot places)
- BLAST-specific tuning of OCaml virtual machine options

Lessons Learnt

- Language features support
- No premature UNKNOWN
- Efficiently ignore irrelevant details
- Engineering matters
- Theory improvement matters

Model Checking

Clarke/Emerson, Sifakis (1981) *Iterative fixpoint post computation*

```
Reached, Frontier := { $s_0$ }
while Frontier != $\emptyset$ do
  remove s from Frontier
  for each $s' \in$ post( s ) do



       if !stop(s', Reached) add s' to Reached, Frontier
return Reached
```

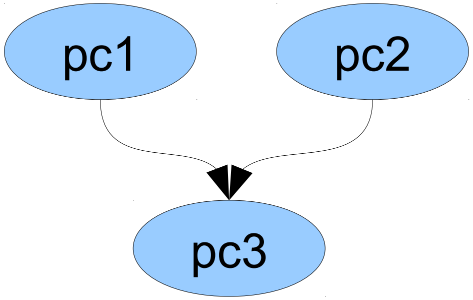
Configurable Program Analysis

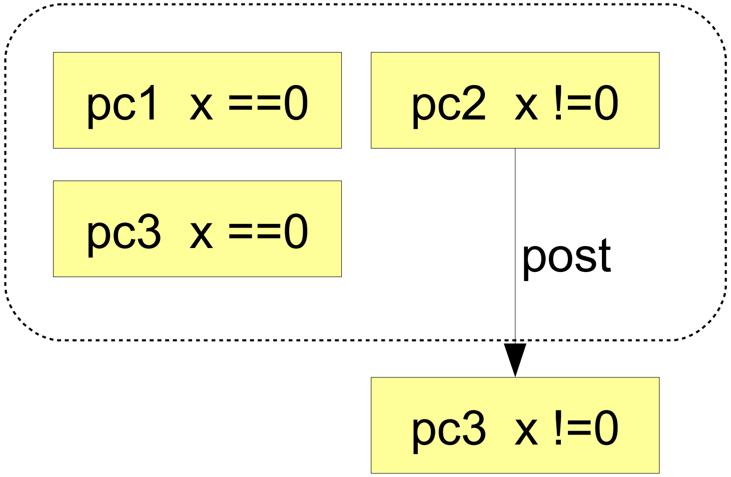
D.Beyer, T.A.Henzinger, G. Theoduloz (2007)

CPA = (**D**, **post**, **merge**, **stop**)

```
$Reached$, $Frontier$ := { $s_0$ }
while $Frontier$ != $\emptyset$ do
  remove $s$ from $Frontier$
  for each $s' \in \underline{\textbf{post}}(\ s\ )$ do
    for each $s'' \in Reached$ do
       $s''_{new}$ := $\underline{\textbf{merge}}(\ s', s''\ )$        // require: $s''_{new} \sqsubseteq \underline{\textbf{merge}}(\ s', s''\ )$
       if $s''_{new}$ != $s''$ then
         replace $s''$ in $Reached$, $Frontier$ by $s''_{new}$
       if !$\underline{\textbf{stop}}(s', Reached)$ add $s'$ to $Reached$, $Frontier$
return $Reached$
```

Data Flow vs Model Checking





pc1 x ==0

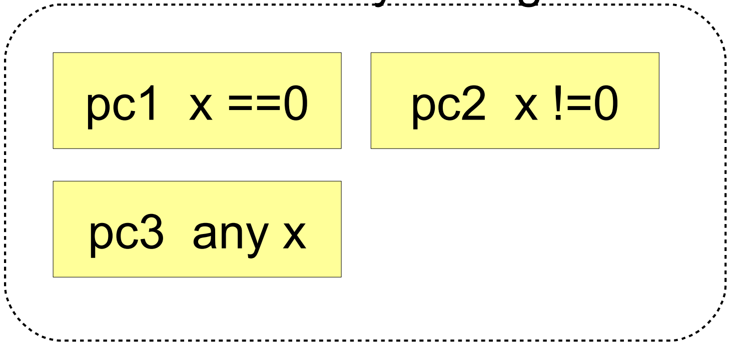
pc2 x !=0

pc3 x ==0

post

pc3 x !=0

Data flow: always merge

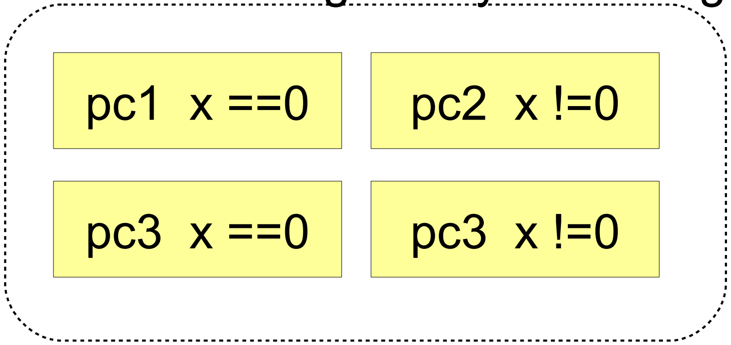


pc1 x ==0

pc2 x !=0

pc3 any x

Model checking: always not merge



pc1 x ==0

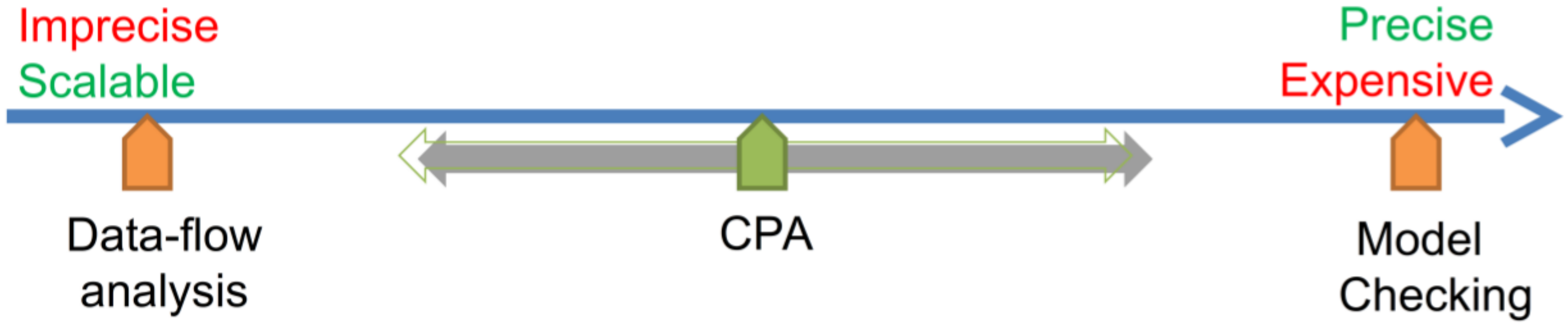
pc2 x !=0

pc3 x ==0

pc3 x !=0

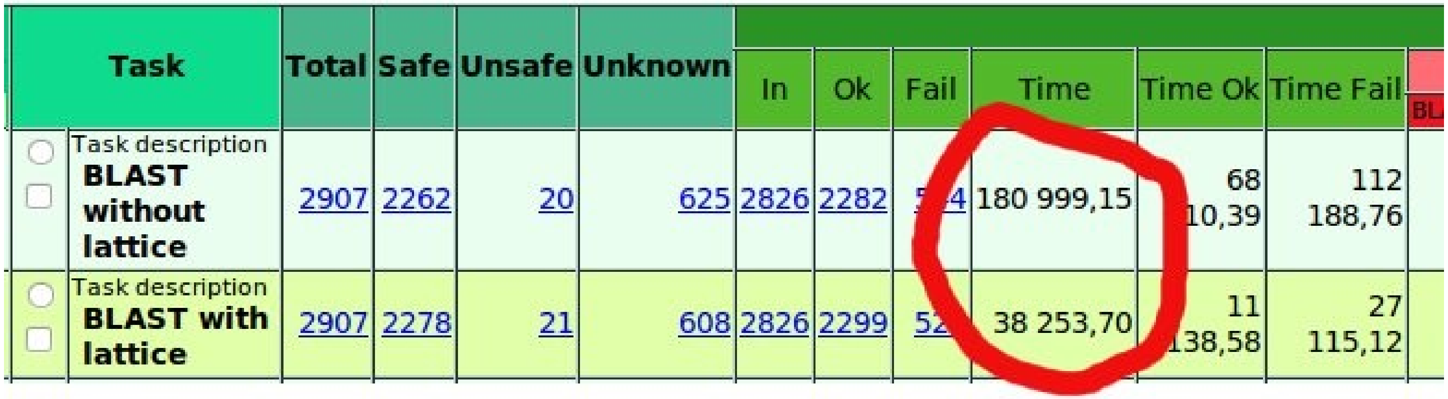
Configurable Program Analysis

D.Beyer, T.A.Henzinger, G. Theoduloz (2007)



BLAST with CPA enabled

- BLAST without lattice option
- BLAST with lattice option, which enables CPA



	Task	Total	Safe	Unsafe	Unknown	In	Ok	Fail	Time	Time Ok	Time Fail	BL
○ □	Task description **BLAST without lattice**	2907	2262	20	625	2826	2282	544	180 999,15	68 10,39	112 188,76	
○ □	Task description **BLAST with lattice**	2907	2278	21	608	2826	2299	527	38 253,70	11 138,58	27 115,12	

4.8x

Lessons Learnt

- Language features support
- No premature UNKNOWN
- Efficiently ignore irrelevant details
- Engineering matters
- Theory improvement matters
- Engineering vs. Theory improvement

CPAchecker

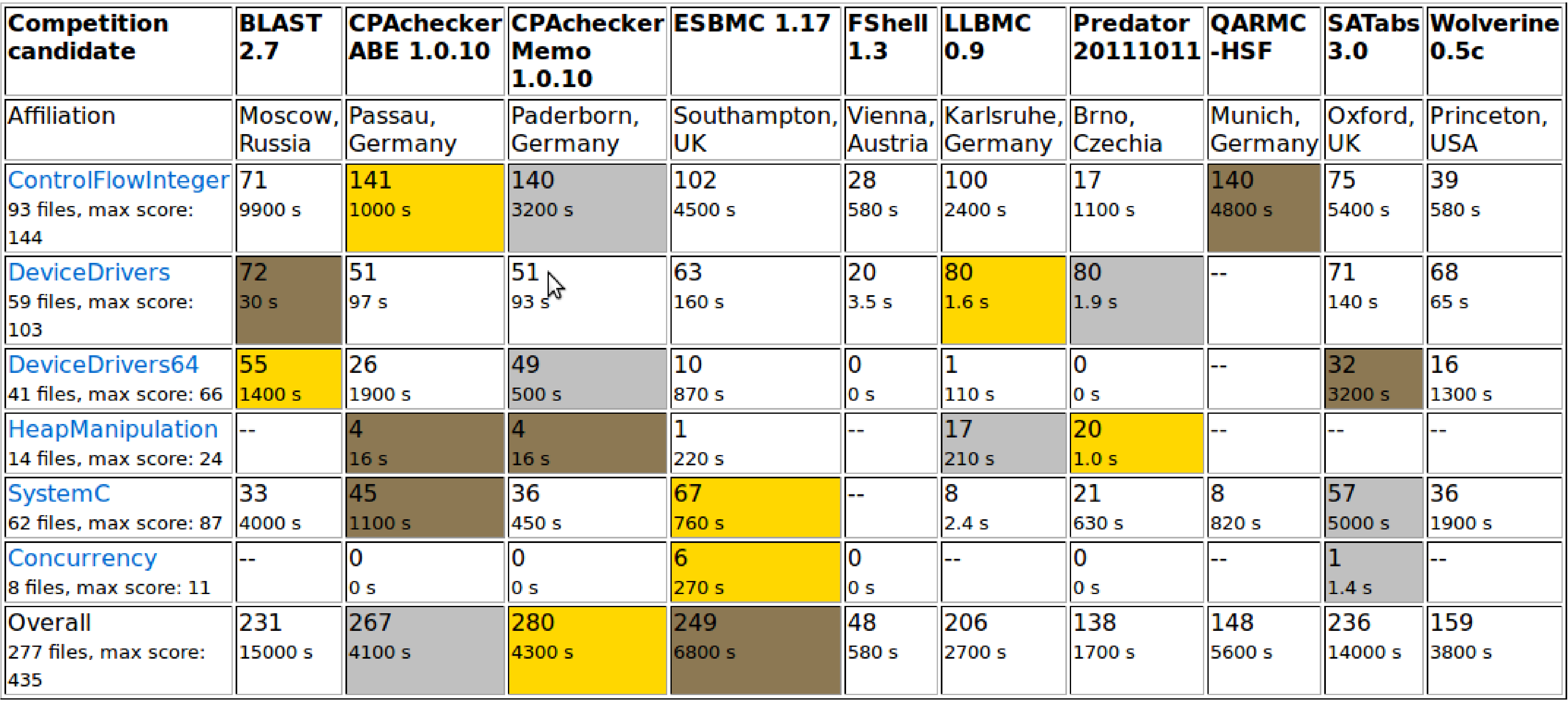
- Modular framework for software verification
  - Written in Java
  - Open source: Apache 2.0 License
  - Over 40 contributors so far
    from ~8 universities/institutions
  - ~300 000 lines of code
    (170 000 without blanks and comments)
  - Started 2007

http://cpachecker.sosy-lab.org

Lessons Learnt

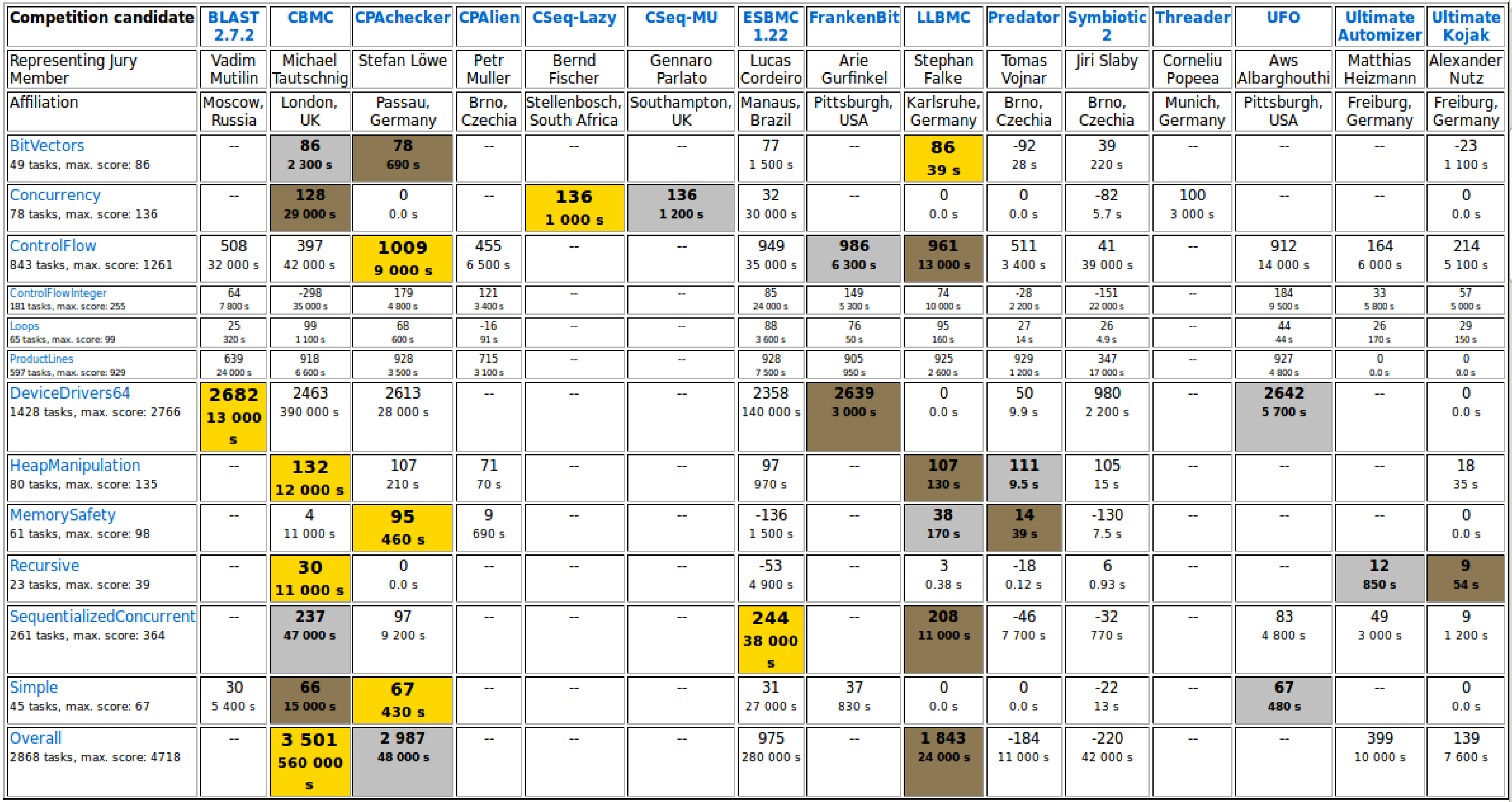
- Language features support
- No premature UNKNOWN
- Efficiently ignore irrelevant details
- Engineering matters
- Theory improvement matters
- Engineering vs. Theory improvement
- Usability

SVCOMP'12 Results



Competition candidate	BLAST 2.7	CPAchecker ABE 1.0.10	CPAchecker Memo 1.0.10	ESBMC 1.17	FShell 1.3	LLBMC 0.9	Predator 20111011	QARMC -HSF	SATabs 3.0	Wolverine 0.5c
Affiliation	Moscow, Russia	Passau, Germany	Paderborn, Germany	Southampton, UK	Vienna, Austria	Karlsruhe, Germany	Brno, Czechia	Munich, Germany	Oxford, UK	Princeton, USA
ControlFlowInteger 93 files, max score: 144	71 9900 s	141 1000 s	140 3200 s	102 4500 s	28 580 s	100 2400 s	17 1100 s	140 4800 s	75 5400 s	39 580 s
DeviceDrivers 59 files, max score: 103	72 30 s	51 97 s	51 93 s	63 160 s	20 3.5 s	80 1.6 s	80 1.9 s	--	71 140 s	68 65 s
DeviceDrivers64 41 files, max score: 66	55 1400 s	26 1900 s	49 500 s	10 870 s	0 0 s	1 110 s	0 0 s	--	32 3200 s	16 1300 s
HeapManipulation 14 files, max score: 24	--	4 16 s	4 16 s	1 220 s	--	17 210 s	20 1.0 s	--	--	--
SystemC 62 files, max score: 87	33 4000 s	45 1100 s	36 450 s	67 760 s	--	8 2.4 s	21 630 s	8 820 s	57 5000 s	36 1900 s
Concurrency 8 files, max score: 11	--	0 0 s	0 0 s	6 270 s	0 0 s	--	0 0 s	--	1 1.4 s	--
Overall 277 files, max score: 435	231 15000 s	267 4100 s	280 4300 s	249 6800 s	48 580 s	206 2700 s	138 1700 s	148 5600 s	236 14000 s	159 3800 s

SVCOMP'14 Results



Competition candidate	BLAST 2.7.2	CBMC	CPAchecker	CPAlien	CSeq-Lazy	CSeq-MU	ESBMC 1.22	FrankenBit	LLBMC	Predator	Symbiotic 2	Threader	UFO	Ultimate Automizer	Ultimate Kojak
Representing Jury Member	Vadim Mutilin	Michael Tautschnig	Stefan Löwe	Petr Muller	Bernd Fischer	Gennaro Parlato	Lucas Cordeiro	Arie Gurfinkel	Stephan Falke	Tomas Vojnar	Jiri Slaby	Corneliu Popeea	Aws Albarghouthi	Matthias Heizmann	Alexander Nutz
Affiliation	Moscow, Russia	London, UK	Passau, Germany	Brno, Czechia	Stellenbosch, South Africa	Southampton, UK	Manaus, Brazil	Pittsburgh, USA	Karlsruhe, Germany	Brno, Czechia	Brno, Czechia	Munich, Germany	Pittsburgh, USA	Freiburg, Germany	Freiburg, Germany
BitVectors 49 tasks, max. score: 86	--	**86** **2 300 s**	**78** **690 s**	--	--	--	77 1 500 s	--	**86** **39 s**	-92 28 s	39 220 s	--	--	--	-23 1 100 s
Concurrency 78 tasks, max. score: 136	--	**128** **29 000 s**	0 0.0 s	--	**136** **1 000 s**	**136** **1 200 s**	32 30 000 s	--	0 0.0 s	0 0.0 s	-82 5.7 s	100 3 000 s	--	--	0 0.0 s
ControlFlow 843 tasks, max. score: 1261	508 32 000 s	397 42 000 s	**1009** **9 000 s**	455 6 500 s	--	--	949 35 000 s	**986** **6 300 s**	**961** **13 000 s**	511 3 400 s	41 39 000 s	--	912 14 000 s	164 6 000 s	214 5 100 s
ControlFlowInteger 181 tasks, max. score: 255	64 7 800 s	-298 35 000 s	179 4 800 s	121 3 400 s	--	--	85 24 000 s	149 5 300 s	74 10 000 s	-28 2 200 s	-151 22 000 s	--	184 9 500 s	33 5 800 s	57 5 000 s
Loops 65 tasks, max. score: 99	25 320 s	99 1 100 s	68 600 s	-16 91 s	--	--	88 3 600 s	76 50 s	95 160 s	27 14 s	26 4.9 s	--	44 44 s	26 170 s	29 150 s
ProductLines 597 tasks, max. score: 929	639 24 000 s	918 6 600 s	928 3 500 s	715 3 100 s	--	--	928 7 500 s	905 950 s	925 2 600 s	929 1 200 s	347 17 000 s	--	927 4 800 s	0 0.0 s	0 0.0 s
DeviceDrivers64 1428 tasks, max. score: 2766	**2682** **13 000 s**	2463 390 000 s	2613 28 000 s	--	--	--	2358 140 000 s	**2639** **3 000 s**	0 0.0 s	50 9.9 s	980 2 200 s	--	**2642** **5 700 s**	--	0 0.0 s
HeapManipulation 80 tasks, max. score: 135	--	**132** **12 000 s**	107 210 s	71 70 s	--	--	97 970 s	--	**107** **130 s**	**111** **9.5 s**	105 15 s	--	--	--	18 35 s
MemorySafety 61 tasks, max. score: 98	--	4 11 000 s	**95** **460 s**	9 690 s	--	--	-136 1 500 s	--	**38** **170 s**	**14** **39 s**	-130 7.5 s	--	--	--	0 0.0 s
Recursive 23 tasks, max. score: 39	--	**30** **11 000 s**	0 0.0 s	--	--	--	-53 4 900 s	--	3 0.38 s	-18 0.12 s	6 0.93 s	--	--	**12** **850 s**	**9** **54 s**
SequentializedConcurrent 261 tasks, max. score: 364	--	**237** **47 000 s**	97 9 200 s	--	--	--	**244** **38 000 s**	--	**208** **11 000 s**	-46 7 700 s	-32 770 s	--	83 4 800 s	49 3 000 s	9 1 200 s
Simple 45 tasks, max. score: 67	30 5 400 s	**66** **15 000 s**	**67** **430 s**	--	--	--	31 27 000 s	37 830 s	0 0.0 s	0 0.0 s	-22 13 s	--	**67** **480 s**	--	0 0.0 s
Overall 2868 tasks, max. score: 4718	--	**3 501** **560 000 s**	**2 987** **48 000 s**	--	--	--	975 280 000 s	--	**1 843** **24 000 s**	-184 11 000 s	-220 42 000 s	--	--	399 10 000 s	139 7 600 s

Lessons Learnt

- Language features support
- No premature UNKNOWN
- Efficiently ignore irrelevant details
- Engineering matters
- Theory improvement matters
- Engineering vs. Theory improvement
- Usability
- Source code coverage

Recent Experiment

All modules Linux 3.14 (4368), 32 rules

CPAChecker BAM VA+PA, EMG **conservative**

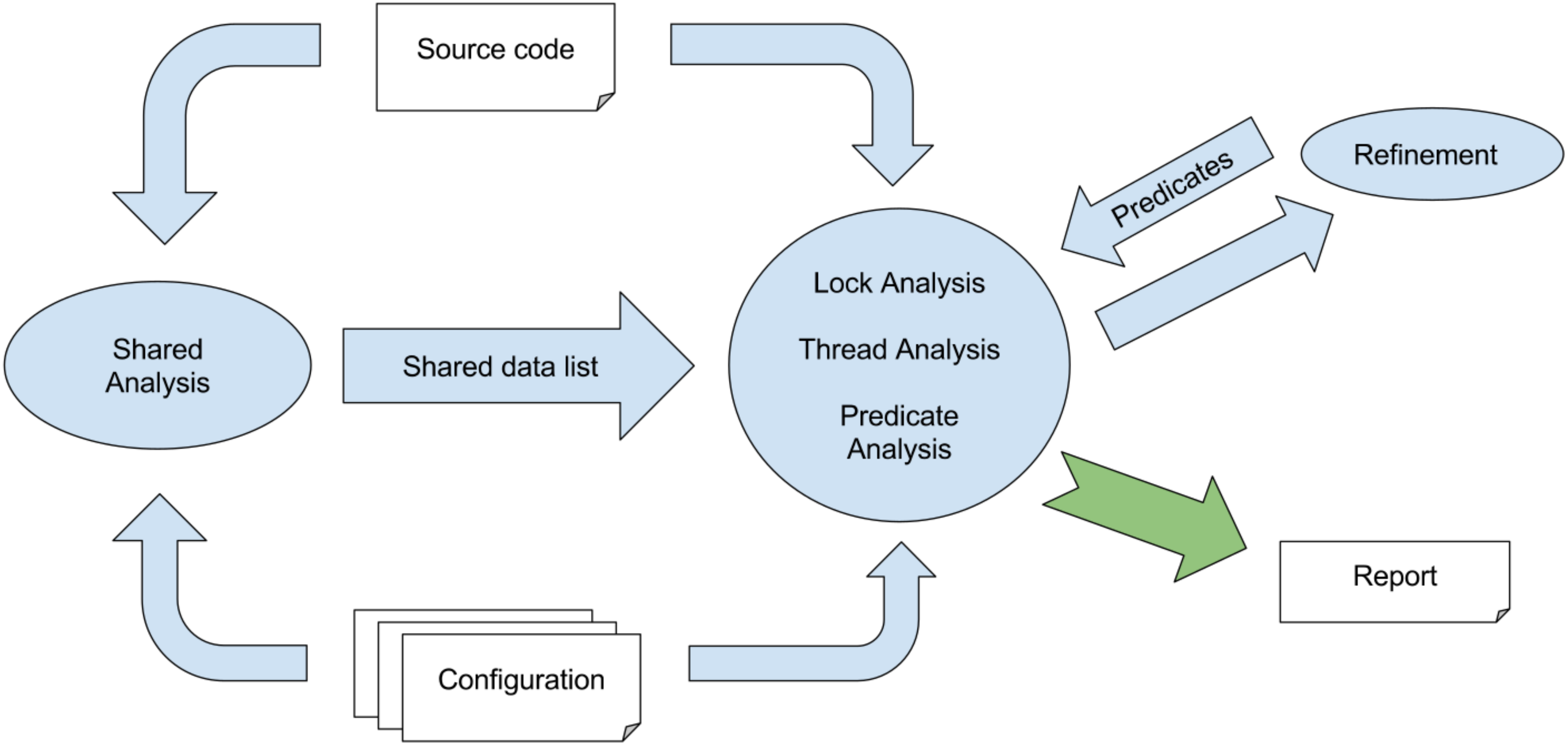
- 417 UNSAFEs
  - 116 TRUE 28%
  - 301 FALSE 72%

Kernel model:	97	32%
Verifier:	82	27%
Rules:	46	15%
EMG:	**41**	**14%**
LDV:	35	11%

Taxonomy of Typical Bugs

Rule classes	Types	Number of bug fixes	Percents	Cumulative total percents	
Correct usage of the Linux kernel API (176 ~ 50%)	Alloc/free resources	32	~18%	~18%	Reach ability
	Check parameters	25	~14%	~32%	
	Work in atomic context	19	~11%	~43%	
	Uninitialized resources	17	~10%	~53%	
	Synchronization primitives in one thread	12	~7%	~60%	
	Style	10	~6%	~65%	
	Network subsystem	10	~6%	~71%	
	USB subsystem	9	~5%	~76%	
	Check return values	7	~4%	~80%	
	DMA subsystem	4	~2%	~82%	
	Core driver model	4	~2%	~85%	
	Miscellaneous	27	~15%	100%	
Generic (102 ~ 30%)	NULL pointer dereferences	31	~30%	~30%	
	Alloc/free memory	24	~24%	~54%	
	Syntax	14	~14%	~68%	
	Integer overflows	8	~8%	~76%	
	Buffer overflows	8	~8%	~83%	
	Uninitialized memory	6	~6%	~89%	
	Miscellaneous	11	~11%	100%	
Synchronization (71 ~ 20%)	Races	60	~85%	~85%	CPALockator
	Deadlocks	11	~15%	100%	

CPALockator



Pavel Andrianov, Vadim Mutilin, Alexey Khoroshilov
"Predicate abstraction based configurable method for data race detection in Linux kernel"
In Proc. of TMPA-2017

CPALockator Intermediate Results

Linux kernel 4.5-rc1(drivers/ folder)

- 2219 warnings = 270 unsafe drivers
  - 55% - imprecision of environment model
  - 10% - simple memory model
  - 10% - operations with lists
  - 10% - other inaccuracies in our analysis
  - 15% - true races
- 290 true warnings = 32 bugs

Taxonomy of Typical Bugs

Rule classes	Types	Number of bug fixes	Percents	Cumulative total percents	
Correct usage of the Linux kernel API (176 ~ 50%)	Alloc/free resources	32	~18%	~18%	Reach ability
	Check parameters	25	~14%	~32%	
	Work in atomic context	19	~11%	~43%	
	Uninitialized resources	17	~10%	~53%	
	Synchronization primitives in one thread	12	~7%	~60%	
	Style	10	~6%	~65%	
	Network subsystem	10	~6%	~71%	
	USB subsystem	9	~5%	~76%	
	Check return values	7	~4%	~80%	
	DMA subsystem	4	~2%	~82%	
	Core driver model	4	~2%	~85%	
	Miscellaneous	27	~15%	100%	
Generic (102 ~ 30%)	NULL pointer dereferences	31	~30%	~30%	SMG
	Alloc/free memory	24	~24%	~54%	
	Syntax	14	~14%	~68%	
	Integer overflows	8	~8%	~76%	
	Buffer overflows	8	~8%	~83%	
	Uninitialized memory	6	~6%	~89%	
	Miscellaneous	11	~11%	100%	
Synchronization (71 ~ 20%)	Races	60	~85%	~85%	CPALockator
	Deadlocks	11	~15%	100%	

LinuxTesting.org

Thank you!

- Alexey Khoroshilov
- http://linuxtesting.org/ldv

ISP RAS

Institute for System Programming of the Russian Academy of Sciences