LinuxTesting.org

SMT-формулы с неинтерпретируемыми функциями для уточнения предикатной абстракции в CPAchecker

Mikhail Mandrykin
mandrykin@ispras.ru

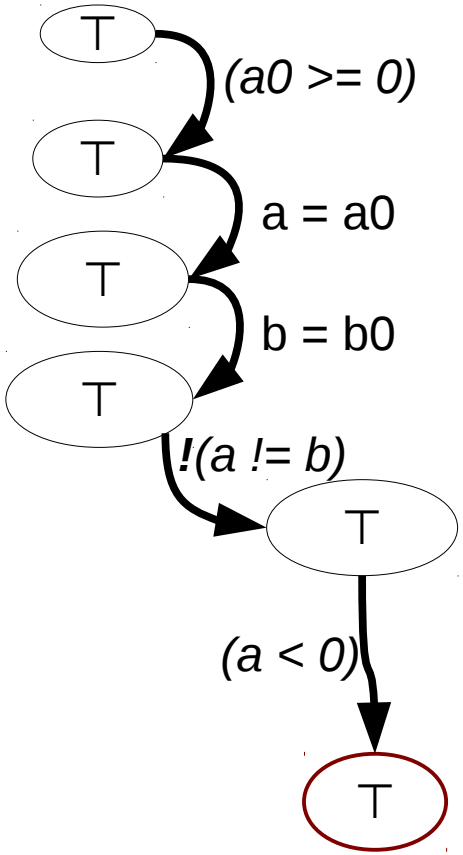
ISP RAS
Institute for System Programming of the Russian Academy of Sciences

Проверка достижимости (пример)

```
int a0, b0;
__VERIFIER_assume(a0 >= 0);
int a = a0, b = b0;

while (a != b)
    if (a > b) a -= b;
    else b -= a;

if (a < 0)
    ERROR: goto ERROR;
```

Проверка достижимости (пример)

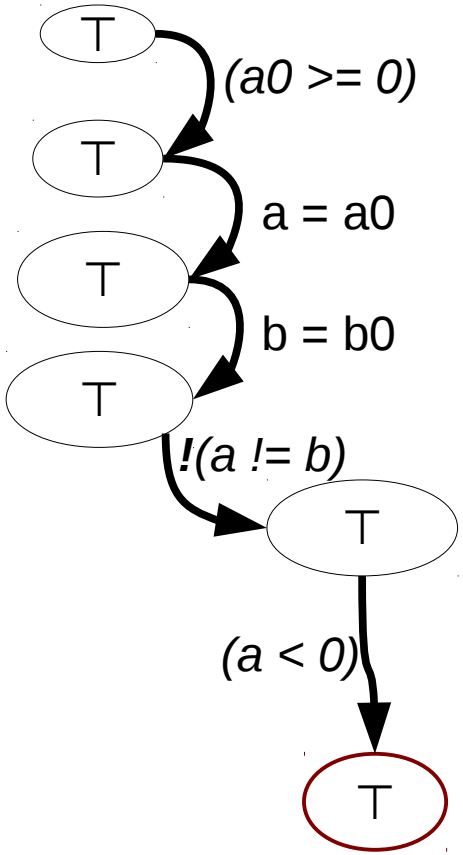
```
int a0, b0;
__VERIFIER_assume(a0 >= 0);
int a = a0, b = b0;

while (a != b)
    if (a > b) a -= b;
    else b -= a;

if (a < 0)
    ERROR: goto ERROR;
```



Проверка выполнимости пути

```
(a0 >= 0)
a = a0
b = b0
!(a != b)
(a < 0)
```

$$a0_1 >= 0 \wedge$$
$$a_1 = a0_1 \wedge$$
$$b_1 = b0_1 \wedge$$
$$\neg(a_1 \neq b_1) \wedge$$
$$(a_1 < 0)$$



Проверка выполнимости пути

```
(a0 >= 0)
a = a0
b = b0
!(a != b)
(a < 0)
```

$a0_1 >= 0 \wedge$
$a_1 = a0_1 \wedge$
$b_1 = b0_1 \wedge$
$\neg(a_1 \neq b_1) \wedge$
$(a_1 < 0)$

$a0_1 >= 0 \wedge a_1 = a0_1 \wedge b_1 = b0_1 \wedge \neg(a_1 \neq b_1) \wedge (a_1 < 0)$ – ✘ $(unsat, \models \neg f)$

Интерполяция

```
(a0 >= 0)
a = a0
b = b0
!(a != b)
(a < 0)
```

$$a0_1 >= 0 \wedge$$
$$a_1 = a0_1 \wedge$$
$$b_1 = b0_1 \wedge$$
$$\neg(a_1 \neq b_1) \wedge$$
$$(a_1 < 0)$$

$$a0_1 >= 0 \wedge a_1 = a0_1 \wedge b_1 = b0_1 \wedge \neg(a_1 \neq b_1) \wedge (a_1 < 0) - \text{✘}(unsat, \models \neg f)$$

$$\underline{a0_1} >= 0 \rightarrow \underline{a0_1} >= 0 \rightarrow \neg(a_1 = \underline{a0_1} \wedge b_1 = b0_1 \wedge \neg(a_1 \neq b_1) \wedge (a_1 < 0))$$

$p_1 \equiv a0 \geq 0$

Интерполяция

```
(a0 >= 0)
a = a0
b = b0
!(a != b)
(a < 0)
```

$$a0_1 >= 0 \wedge$$
$$a_1 = a0_1 \wedge$$
$$b_1 = b0_1 \wedge$$
$$\neg(a_1 \neq b_1) \wedge$$
$$(a_1 < 0)$$

$$a0_1 >= 0 \wedge a_1 = a0_1 \wedge b_1 = b0_1 \wedge \neg(a_1 \neq b_1) \wedge (a_1 < 0) - \text{✘}(unsat, \models \neg f)$$

$$a0_1 >= 0 \wedge \underline{a_1} = a0_1 \rightarrow \neg \underline{a_1} < 0 \rightarrow \neg(b_1 = b0_1 \wedge \neg(\underline{a_1} \neq b_1) \wedge (\underline{a_1} < 0))$$

$p_2 \equiv a < 0$

Проверка достижимости 2

```
int a0, b0;
__VERIFIER_assume(a0 >= 0);
int *a = &a0, *b = &b0;

while (*a != *b)
    if (*a > *b) *a -= *b;
    else *b -= *a;

if (*a < 0)
    ERROR: goto ERROR;
```

```
(a0 >= 0)
a = &a0
b = &b0
(*a != *b)
(*a > *b)
*a -= *b
!(*a != *b)
(*a < 0)
```

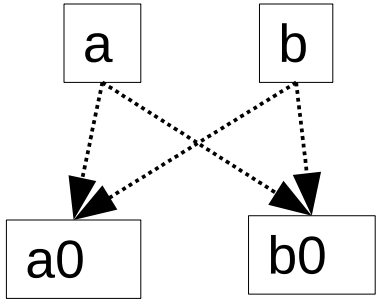
Введение переменных

```
(a0 >= 0)
a = &a0
b = &b0
(*a != *b)
(*a > *b)
*a -= *b
!(*a != *b)
(*a < 0)
```

*a

*b

$\&a0 > 0 \wedge$
$\&b0 > 0 \wedge$
$\&a0 \neq \&b0 \wedge$
$a0_1 >= 0 \wedge$
$a_1 = \&a0 \wedge$
$*a_1 = a0_1 \wedge$
$b_1 = \&b0 \wedge$
$*b_1 = b0_1 \wedge$
$(*a_1 \neq *b_1) \wedge$
$(*a_1 > *b_1) \wedge$
$*a_2 = *a_1 - *b_1 \wedge$
$(a_1 = \&a0 \wedge a0_2 = *a_1 - *b_1 \vee a_1 \neq \&a0 \wedge a0_2 = a0_1) \wedge$
$(a_1 = \&b0 \wedge b0_2 = *a_1 - *b_1 \vee a_1 \neq \&b0 \wedge b0_2 = b0_1) \wedge$
$(a_1 = b_1 \wedge *b_2 = *a_1 - *b_1 \vee a_1 \neq b_1 \wedge *b_2 = *b_1) \wedge$
$\neg(*a_2 \neq *b_2) \wedge$
$(*a_2 < 0)$



Общие правила

- `pa = e; // pa : T *(*), pa_i, *pa_j(, **pa_k, ...)`

  $pa_{i+1}=e \wedge *pa_{j+1} = *e(\wedge **pa_{k+1} = **e \wedge ...)$

- `*pa = e; // pa : T *, a^i_ia, *pa^j_ja, **ppa^k_ka,...`

$$*pa_{i+1}=e \wedge \bigwedge_{(ai:T)}(pa=\&a^i \wedge a^i_{ia+1}= e \vee pa\neq\&a^i \wedge a^i_{ia+1}=a^i_{ia}) \wedge$$
$$\bigwedge_{(paj:T*)}(pa=pa^j \wedge pa^j_{ja+1}= e \vee pa\neq pa^j \wedge pa^j_{ja+1}=pa^j_{ja}) \wedge$$
$$\bigwedge_{(ppaj:T**)}(ppa=ppa^k \wedge pa^k_{ka+1}= e \vee ppa\neq ppa^k \wedge ppa^k_{ka+1}=ppa^k_{ka}) \wedge$$
$$...$$

closure (depth = 3)

- В первом случае, вообще говоря, нужны те же дизъюнкции, что и во втором, только для *closure_depth - 1*

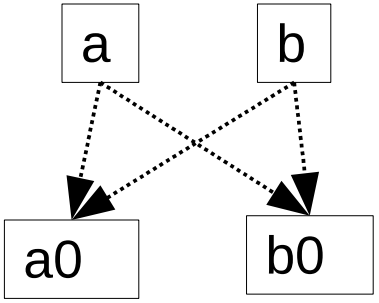
Введение переменных

```
(a0 >= 0)
a = &a0
b = &b0
(*a != *b)
(*a > *b)
*a -= *b
!(*a != *b)
(*a < 0)
```

*a

*b

$\&a0 > 0 \wedge$
$\&b0 > 0 \wedge$
$\&a0 \neq \&b0 \wedge$
$a0_1 >= 0 \wedge$
$a_1 = \&a0 \wedge$
$*a_1 = a0_1 \wedge$
$b_1 = \&b0 \wedge$
$*b_1 = b0_1 \wedge$
$(*a_1 \neq *b_1) \wedge$
$(*a_1 > *b_1) \wedge$
$*a_2 = *a_1 - *b_1 \wedge$
$(a_1 = \&a0 \wedge a0_2 = *a_1 - *b_1 \vee a_1 \neq \&a0 \wedge a0_2 = a0_1) \wedge$
$(a_1 = \&b0 \wedge b0_2 = *a_1 - *b_1 \vee a_1 \neq \&b0 \wedge b0_2 = b0_1) \wedge$
$(a_1 = b_1 \wedge *b_2 = *a_1 - *b_1 \vee a_1 \neq b_1 \wedge *b_2 = *b_1) \wedge$
$\neg(*a_2 \neq *b_2) \wedge$
$(*a_2 < 0)$



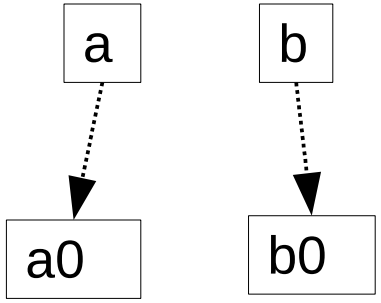
Анализ алиасов

```
(a0 >= 0)
a = &a0
b = &b0
(*a != *b)
(*a > *b)
*a -= *b
!(*a != *b)
(*a < 0)
```

*a

*b

$\&a0 > 0 \wedge$
$\&b0 > 0 \wedge$
$\&a0 \neq \&b0 \wedge$
$a0_1 >= 0 \wedge$
$a_1 = \&a0 \wedge$
$*a_1 = a0_1 \wedge$
$b_1 = \&b0 \wedge$
$*b_1 = b0_1 \wedge$
$(*a_1 \neq *b_1) \wedge$
$(*a_1 > *b_1) \wedge$
$*a_2 = *a_1 - *b_1 \wedge$
$(a_1 = \&a0 \wedge a0_2 = *a_1 - *b_1 \vee a_1 \neq \&a0 \wedge a0_2 = a0_1) \wedge$
$(a_1 = \&b0 \wedge b0_2 = *a_1 - *b_1 \vee a_1 \neq \&b0 \wedge b0_2 = b0_1) \wedge$
$(a_1 = b_1 \wedge *b_2 = *a_1 - *b_1 \vee a_1 \neq b_1 \wedge *b_2 = *b_1) \wedge$
$\neg(*a_2 \neq *b_2) \wedge$
$(*a_2 < 0)$



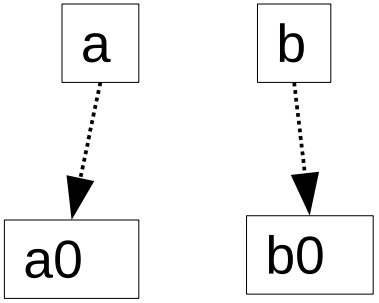
Анализ алиасов

```
(a0 >= 0)
a = &a0
b = &b0
(*a != *b)
(*a > *b)
*a -= *b
!(*a != *b)
(*a < 0)
```

*a

*b

$\&a0 > 0 \wedge$
$\&b0 > 0 \wedge$
$\&a0 \neq \&b0 \wedge$
$a0_1 >= 0 \wedge$
$a_1 = \&a0 \wedge$
$*a_1 = a0_1 \wedge$
$b_1 = \&b0 \wedge$
$*b_1 = b0_1 \wedge$
$(*a_1 \neq *b_1) \wedge$
$(*a_1 > *b_1) \wedge$
$*a_2 = *a_1 - *b_1 \wedge$
$(a0_2 = *a_1 - *b_1) \wedge$
$\neg(*a_2 \neq *b_1) \wedge$
$(*a_2 < 0)$



Поддержка структур

```
struct  sa {
    int a, b;
} a;

struct sa *pa, *pb;
```

a.a

a.b

pa->a

pb->a

pa->b

pb->b

6 переменных

```
struct  sa {
    int *pa, **ppb;
} a;

struct sa *pa, *pb;
```

a.pa *a.pa

a.ppb *a.ppb **a.ppb

pa->a *pa->a pb->a ...

pa->b *pa->b **pa->b

15 переменных

Поддержка структур

```
struct  sa {
    int a, b;
};
struct  sb {
    struct sa *s;
} a;
struct sa *pa, *pb;
```

```
struct  l {
    struct l *next;
} list;

struct l *plist;
```

a.s	a.s->a	a.s->b
pa->s	pa->s->a	pa->s->b
pb->s	pb->s->a	pb->s->b

9 переменных

list.l	list.l->l	list.l->l->l	...
plist->l	plist->l->l	plist->l->l->l	

...

Оптимизация closure depth

- Можно ограничиваться лишь переменными, существенно используемыми в ***нижележащей части формулы***, таким образом достигая ***неограниченного*** *closure_depth* (*closure_depth* определяется фактическим максимальным числом разыменований)

- Выполнив ***преобразование***, можно добиться ***не более одного разыменования*** в операторе:

```
**a = **b;
```

↓

```
tp = *b;
t = *tp;
p = *a; //*a=&v1∧ *p=v ∨ *a=&v2∧ *p=v2 ∨ ... ∨ *a=p1∧*p=*p1 ∨...
*p = t;
```

После такого преобразования достаточно *closure_depth = 1*

Проблемы

- Для поддержки ***связных списков*** требуется выписывать формулу как ***предусловие*** (чтобы использовать оптимизацию)
- Для поддержки ***массивов*** дизъюнкции придется выписывать при каждом обращении:

`a[i] = 0` $i=0 \wedge a0_2=\mathbf{0} \wedge a1_2=a1_1 \wedge \ldots \wedge an_2=an_1 \vee$
$i=1 \wedge a0_2=a0_1 \wedge a1_2=\mathbf{0} \wedge \ldots \wedge an_2=an_1 \vee \ldots \vee$
$i=n \wedge a0_2=a0_1 \wedge a1_2=a1_1 \wedge \ldots \wedge an_2=\mathbf{0} \vee$
$i<0 \vee i \geq n$

`a[i]` $i=0 \wedge r=a0_1 \vee \; i=1 \wedge r=a1_1 \vee \; \ldots \vee \; i=n \wedge r=an_1, \; r$

- Для поддержки ***адресной арифметики*** число дизъюнкций еще больше – в квадрате
(`int *a; *(a+i)` – $\ldots \vee a=\&a+4 \wedge i=1 \wedge r=a5_1 \vee \ldots$)
- `int *a; struct s* s; a = &s->f;` – усложнение ***анализа алиасов***
- Большое число равенств, соответствующее ***присваиванию указателей***
(`struct s* s1, s2; s1 = s2;` – $\ldots \wedge s1 \rightarrow a=s2 \rightarrow a \wedge s1 \rightarrow b=s2 \rightarrow b \wedge \ldots$)

Преимущества

- ***Минимальные требования к решателю*** формул – из теорий – только линейная арифметика

Теория массивов

- Операции ***select*** (*[]*) и ***store*** (`a[i] = 0` – $a_2 = store(a_1, i, 0)$)

rnet`->`**`rx_skb`**`[i]->`**`users`**`--;` →

M2 = store(M1,

*M1[M1[**rnet**] + offsetof(**struct** rionet_private, **rx_skb**) + M1[i]] +*

*offsetof(**struct** sk_buff, **users**),*

*M1[M1[M1[**rnet**] + offsetof(**struct** rionet_private, **rx_skb**) + M1[i]] +*

*offsetof(**struct** sk_buff, **users**)] - 1)*

Неинтерпретируемые функции

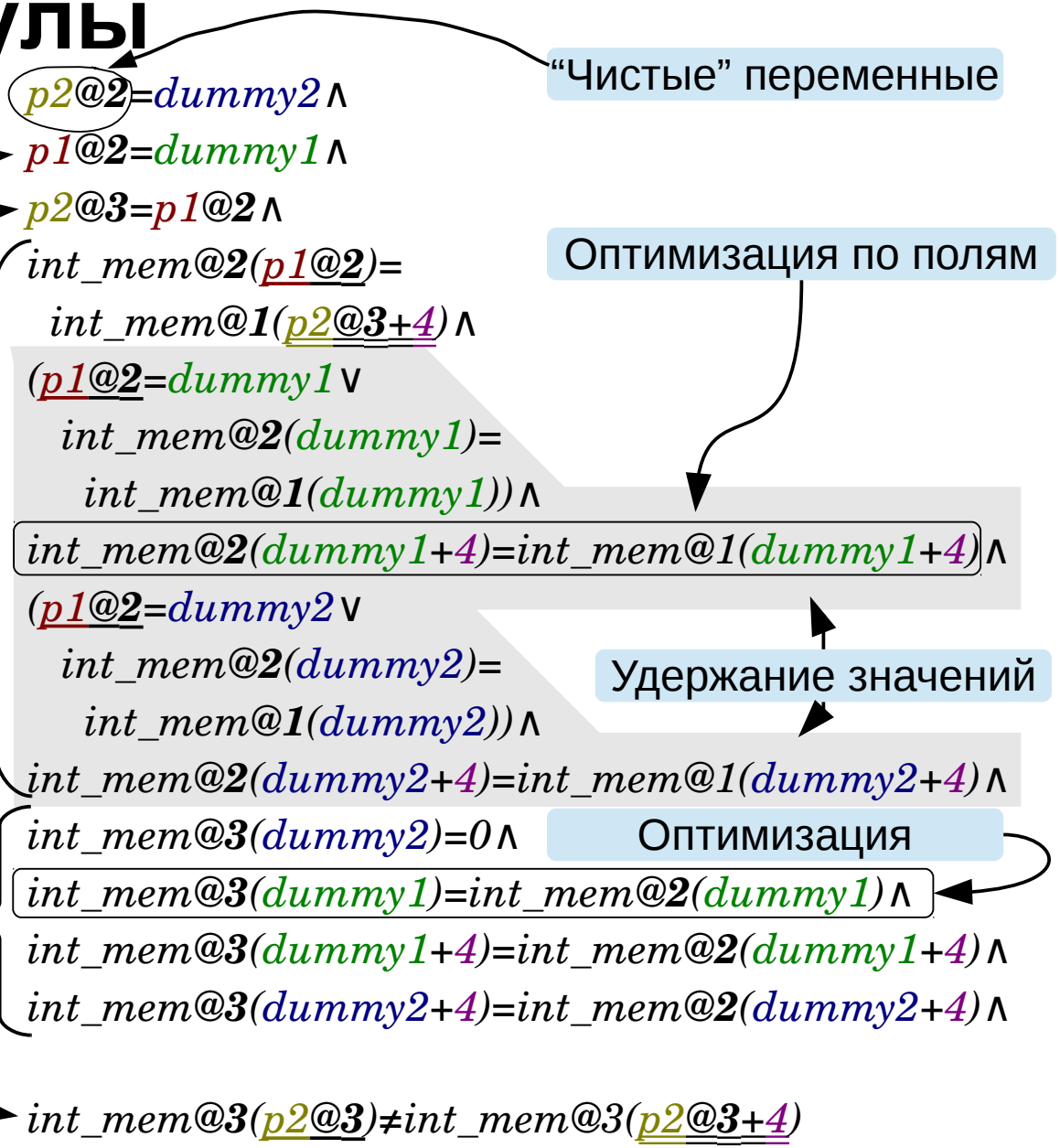
$$f(a)=b \wedge f(c)\neq b \wedge a=c$$

$$\forall a, b.\ a = b \rightarrow f(a)=f(b)$$

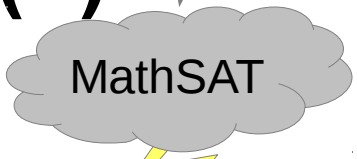
$$a=c \rightarrow f(a)=f(c) \rightarrow f(c)=b \wedge f(c)\neq b \equiv \bot$$

Пример формулы

```
struct {
    int a, b;
} dummy1, dummy2;

p2 = &dummy2;
p1 = &dummy1;
p2 = p1;
p1->a = p2->b;

dummy2.a = 0;

if (p2->a != p2->b)
    … // недостижимо
```



Пример формулы (2)



- Вердикт

unsat

- Интерполянты

(= (int_mem@3 p2@3) (int_mem@3 (+ p2@3 (to_real 4))))

↓

int_mem(p2) = int_mem(p2 + 4)

Непересечение адресов

$(dummy1 > 0 \wedge dummy2 > dummy1 + 8)$

or

$(B(dummy1)=1 \wedge B(dummy1+4)=1 \wedge$

$B(dummy2)=2 \wedge B(dummy2+4)=2))$

```
(assert (! (and
(= p2@2 dummy2)
(= p1@2 dummy1)
(= p2@3 p1@2)
(= (int_mem@2 p1@2)
   (int_mem@1 (+ p2@3 4)))
(or (= p1@2 dummy1)
    (= (int_mem@2 dummy1)
       (int_mem@1 dummy1)))
(= (int_mem@2 (+ dummy1 4)) (int_mem@1 (+ dummy1 4)))
(= (int_mem@2 dummy2)
   (int_mem@1 dummy2)))
(= (int_mem@2 (+ dummy2 4)) (int_mem@1 (+ dummy2 4)))
(= (int_mem@3 dummy1) (int_mem@2 dummy1))
(= (int_mem@3 (+ dummy1 4)) (int_mem@2 (+ dummy1 4)))
(= (int_mem@3 (+ dummy2 4)) (int_mem@2 (+ dummy2 4)) )) :interpolation-group ig1))
(assert (! (and
  (not (= (int_mem@3 p2@3) (int_mem@3 (+ p2@3 4))))
  (> dummy1 0)
  (> dummy2 dummy1+8)) :interpolation-group ig2))
(check-sat)
(get-interpolant (ig1))
```

Сравнение подходов

Инструмент	Указатели	Структуры	Массивы	Рек. структуры (напр. списки)	Адресная арифметика	Масштабир.
BLAST с *"closure depth"*	+	± (конечная глубина)	-	-	-	+
Оптимизированный **BLAST** с *"бесконечным closure depth"*	+	+	-	± (конечная глубина)	-	+
BLAST с *lazy shape analysis ("BLAST 3.0")*	+	+	-	+	-	?
Bounded Model Checking	+	+	+	± (конечная глубина)	+	±
CPAchecker с предикатной абстракцией (переменные)	+	±	-	-	-	+
CPAchecker с предикатной абстракцией (неинт. функции)	+	+	± (огранич. размер)	± (конечная глубина)	+	+

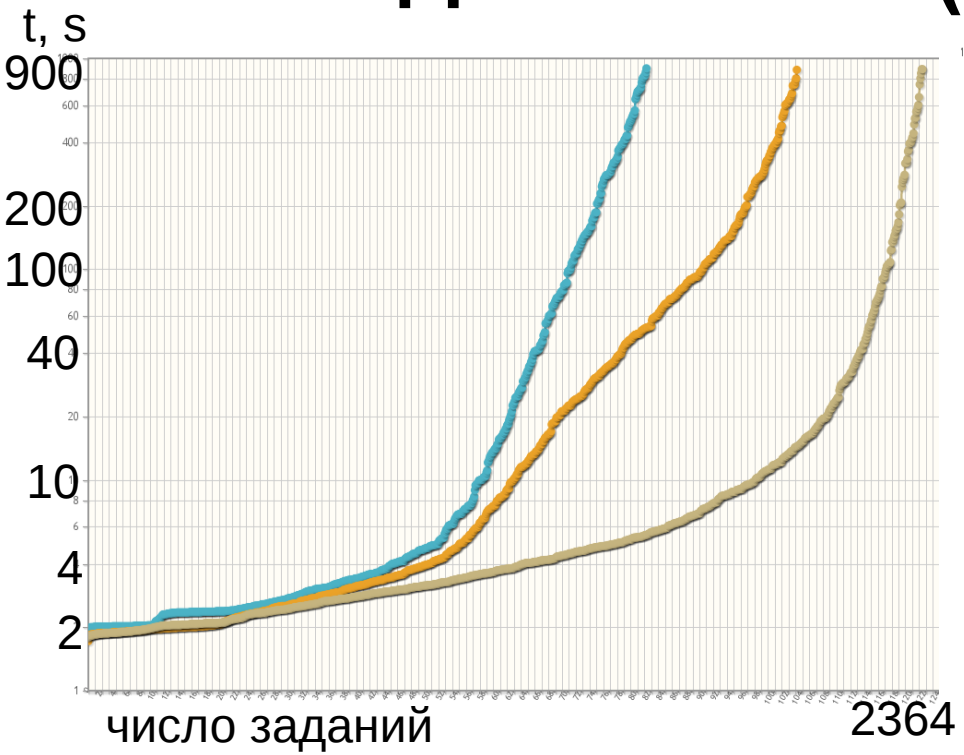
Результаты (30s, 15GB)

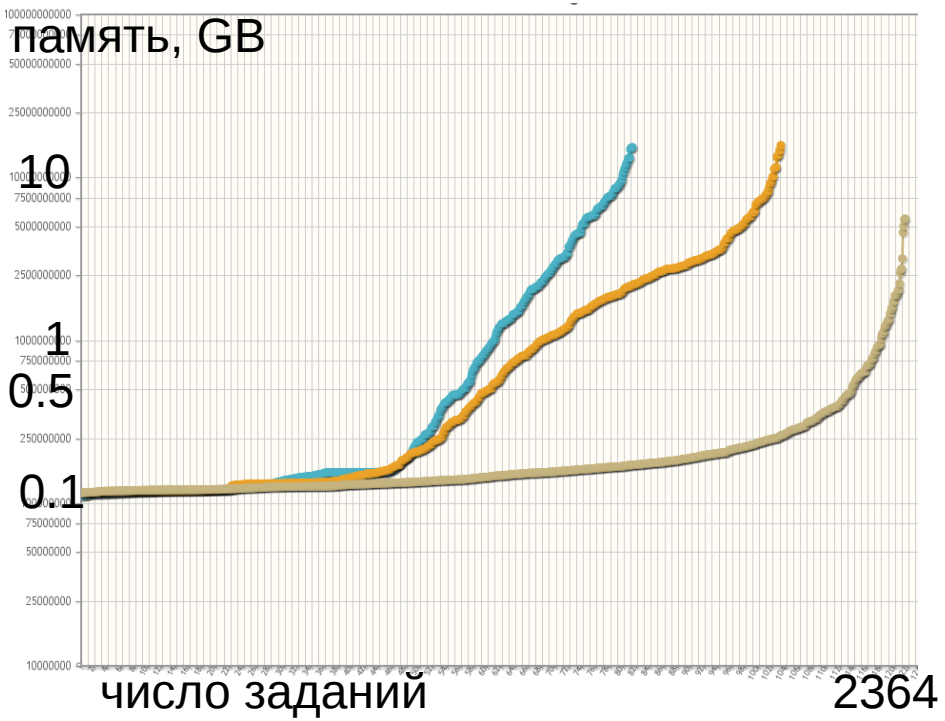
Категория	Max score/ Files	Конфигурации		
		BV/Variables	BV/UFs	Reals/UFs
LDV-regression+DDV (Feature checks category)	273 **159** files	132 **137** correct (**356.58**s total) (**20.134**GB total)	243 **150** correct (**518.43**s total) (**20.892**GB total)	251 **154** correct (**417.33**s total) (**20.172**GB total)
Heap manipulation	56 **35** files	-36 **20** correct (**120.00**s total) (**5.533**GB total)	6 **13** correct (**641.45**s total) (**7.246**GB total)	6 **13** correct (**497.55**s total) (**9.085**GB total)
Bit-vectors	63 **35** files	36 **20** correct (**655.65**s total) (**11.045**GB total)	32 **18** correct (**635.54**s total) (**10.883**GB total)	-88 **4** correct (**314.92**s total) (**9.903**GB total)
Control flow integer	148 **95** files	68 **42** correct (**1924.81**s total) (**216.279**GB total)	53 **41** correct (**1954.73**s total) (**25.036**GB total)	52 **44** correct (**1871.60**s total) (**36.316**GB total)

Результаты (900s, 15GB)

Категория	Max score/ Files	Конфигурации		
		Bit-precise-loops	Bit-precise-lf	Bit-precise-UF-lf
All LDV benchmarks	2840 **1474** files	1605 **814** correct (**360574.49**s total) (**9227.162**GB total)	2035 **1033** correct (**354960.08**s total) (**5211.142**GB total)	2364 **1216** correct (**231996.39**s total) (**1011.491**GB total)

Все задания LDV (900s, 15GB)





Bit-precise-loops

Bit-precise-lf

Bit-precise-UF-lf

SV-COMP'14

Competition candidate	BLAST 2.7.2	CBMC	CPAchecker	ESBMC 1.22	FrankenBit	LLBMC	Predator	UFO	Ultimate Automizer	Ultimate Kojak
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	**2642** **5 700 s**	--	0 0.0 s
HeapManipulation 80 tasks, max. score: 135	--	**132** **12 000 s**	107 210 s	97 970 s	--	**107** **130 s**	**111** **9.5 s**	--	--	18 35 s

Пример ложного срабатывания

- Неявное преобразование указателя (`T * ↔ void *`)

```
void dll_insert_slave(struct slave_item **dll) {
    struct slave_item *item = alloc_or_die_slave();
    ...
    *dll = item; // Indirect assignment through the pointer to struct slave_item *
                 // struct_slave_intem_*(...)=...
}
void dll_insert_master(struct slave_item **dll) { … }
void* dll_create_generic(void (*insert_fnc)()) {
    void *dll = NULL;
    insert_fnc(&dll); // Implicit type conversion from void *
    ...
    return dll; // Reading through the pointer to void *
                // ...=struct_void_*(...)
}
```

Один адрес, но разные функции

Планы

- Анализ указателей с разделением на регионы
- Оптимизация последовательных присваиваний – реже обновлять версию функции
- Слайсинг
- Доменные типы (использование битовых векторов и вещественных чисел в одной формуле)
- Поддержка объединений

LinuxTesting.org

Спасибо за внимание

Mikhail Mandrykin
mandrykin@ispras.ru

ISP RAS
Institute for System Programming of the Russian Academy of Sciences

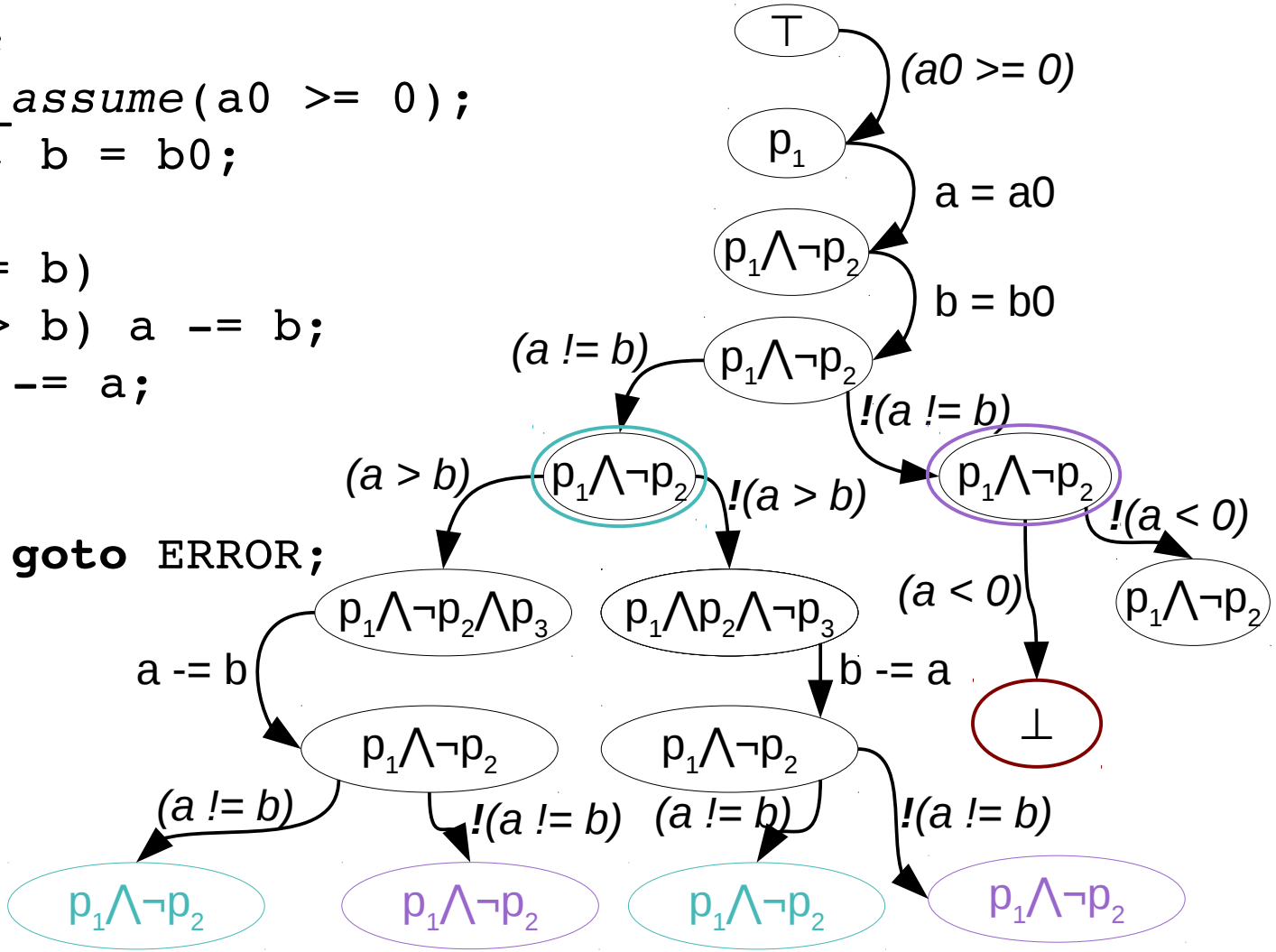
Проверка достижимости (пример)

```
int a0, b0;
__VERIFIER_assume(a0 >= 0);
int a = a0, b = b0;

while (a != b)
    if (a > b) a -= b;
    else b -= a;

if (a < 0)
    ERROR: goto ERROR;
```

$p_1 \equiv a0 \geq 0$

$p_2 \equiv a < 0$

$p_3 \equiv a > b$



Проверка достижимости (пример)

```
int a0, b0;
__VERIFIER_assume(a0 >= 0);
int a = a0, b = b0;

while (a != b)
    if (a > b) a -= b;
    else b -= a;

if (a < 0)
    ERROR: goto ERROR;
```