Семинар отдела
29 января 2012 г.

LinuxTesting.org

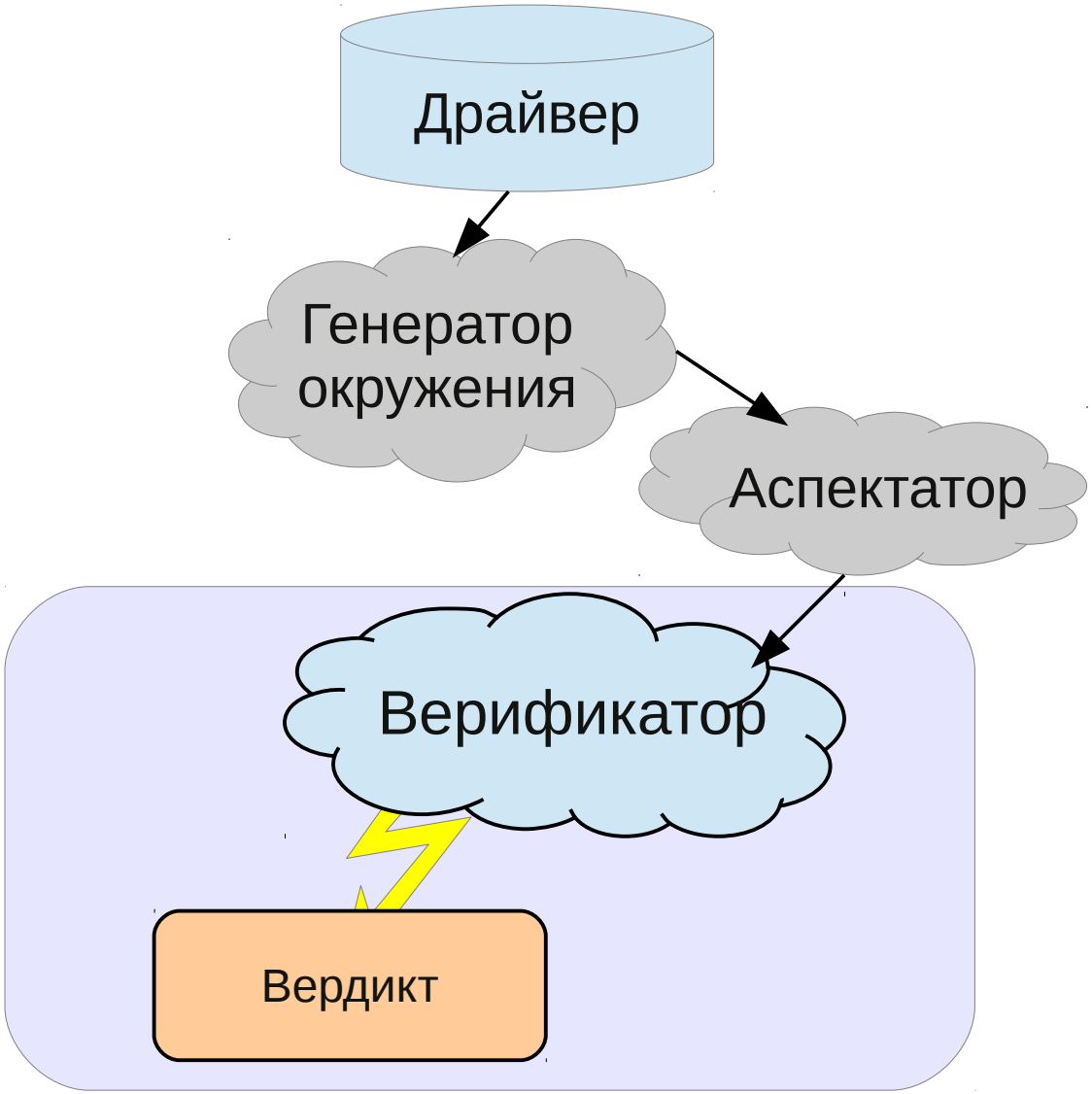
Моделирование памяти при верификации программ методом CEGAR

Мандрыкин Михаил
mandrykin@ispras.ru

ISP RAS

Institute for System Programming of the Russian Academy of Sciences

Схема работы LDV



Код на входе инструмента

```
int probe()
{
    //...
    int x, y;
    int z;
    if (x > y)
        z = x - y;
    else
        z = y - x;
    ldv_assert(z >= 0);
    //...
}

int main()
{
    //...
    probe();
    //...
}
```

```
void ldv_assert(int condition)
{
    if (!condition)
    ERROR: goto ERROR;
}
```

ошибочная метка

точка входа

Задача инструмента

Проверка достижимости ошибочной метки

Достижима ли ошибочная метка при ***каком-либо*** из возможных вариантов исполнения программы?

Комбинаторный взрыв

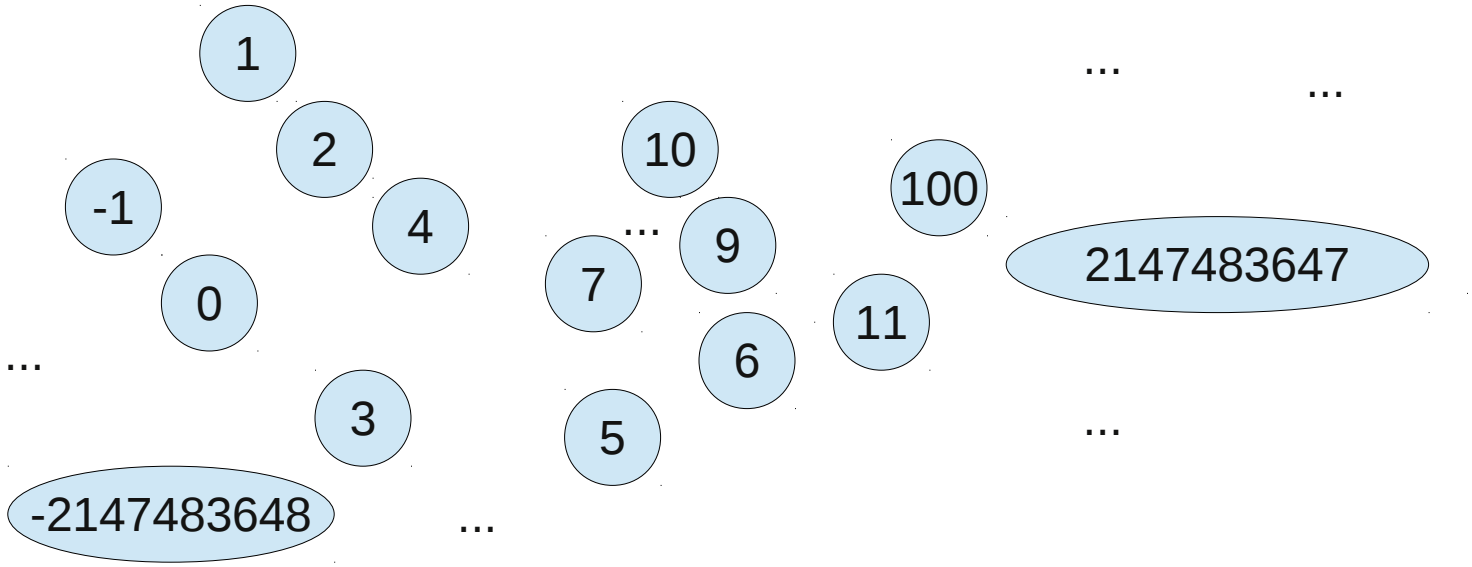
```
int x, y;
int z;
if (x > y)
   z = x - y;
else
   z = y - x;
if (!(z >= 0))
ERROR: goto ERROR;
```

Число начальных состояний:

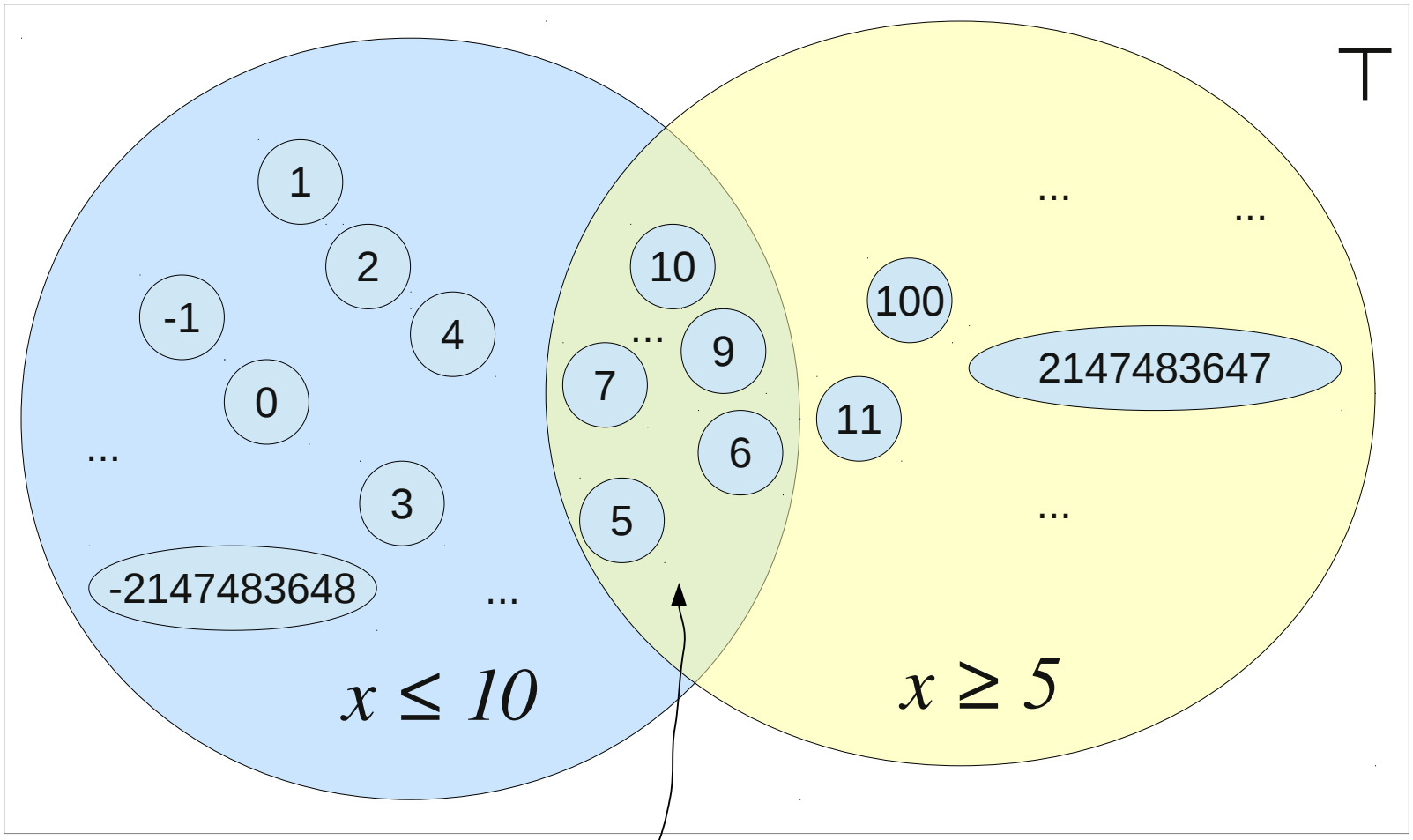
$(2^{32})^3 > 7{,}9 \cdot 10^{28}$ –
несколько миллиардов лет на перебор

Предикатная абстракция (1)

Значения переменной x



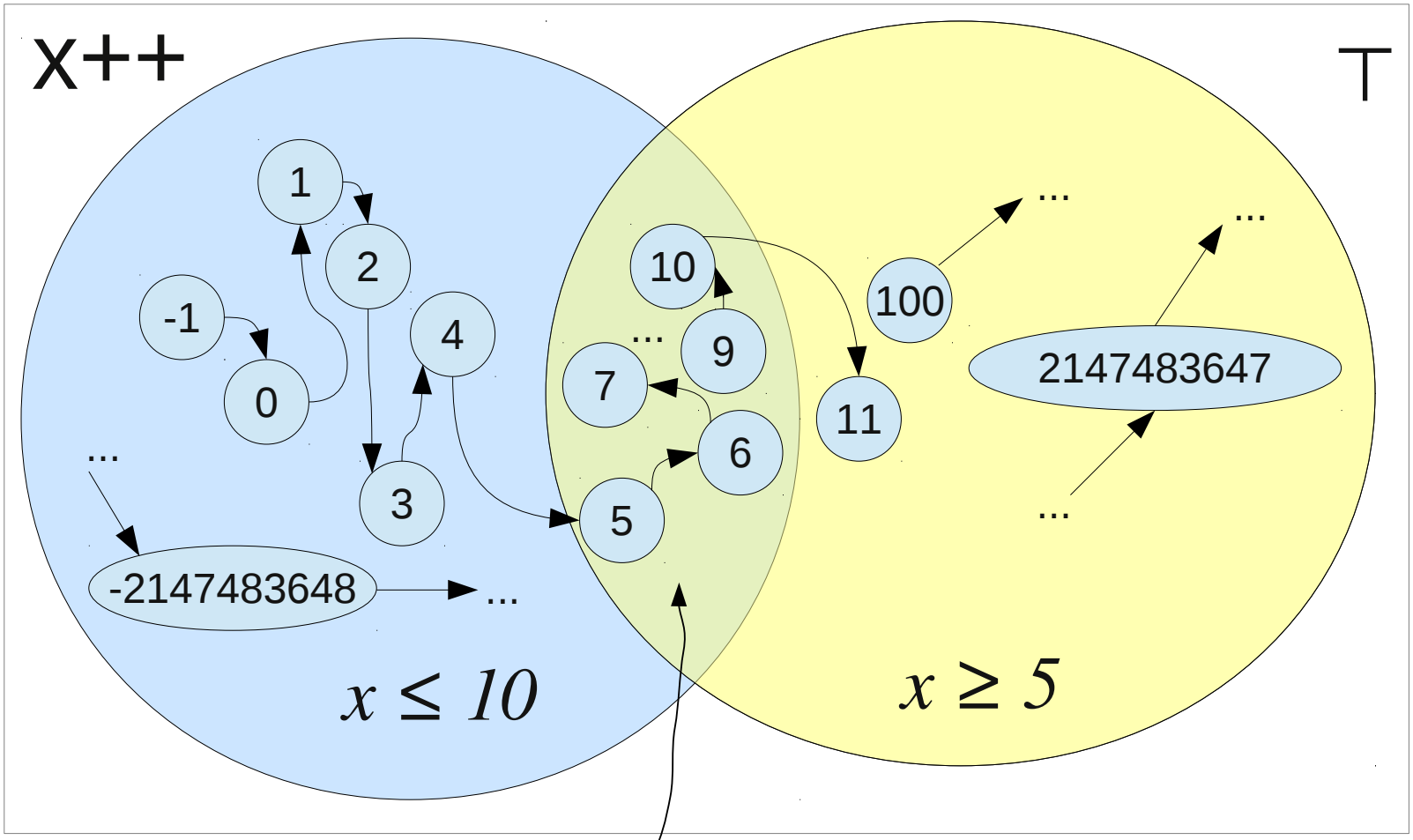
Предикатная абстракция (2)



$x \geq 5 \wedge x \leq 10$

Абстрактные соятояния

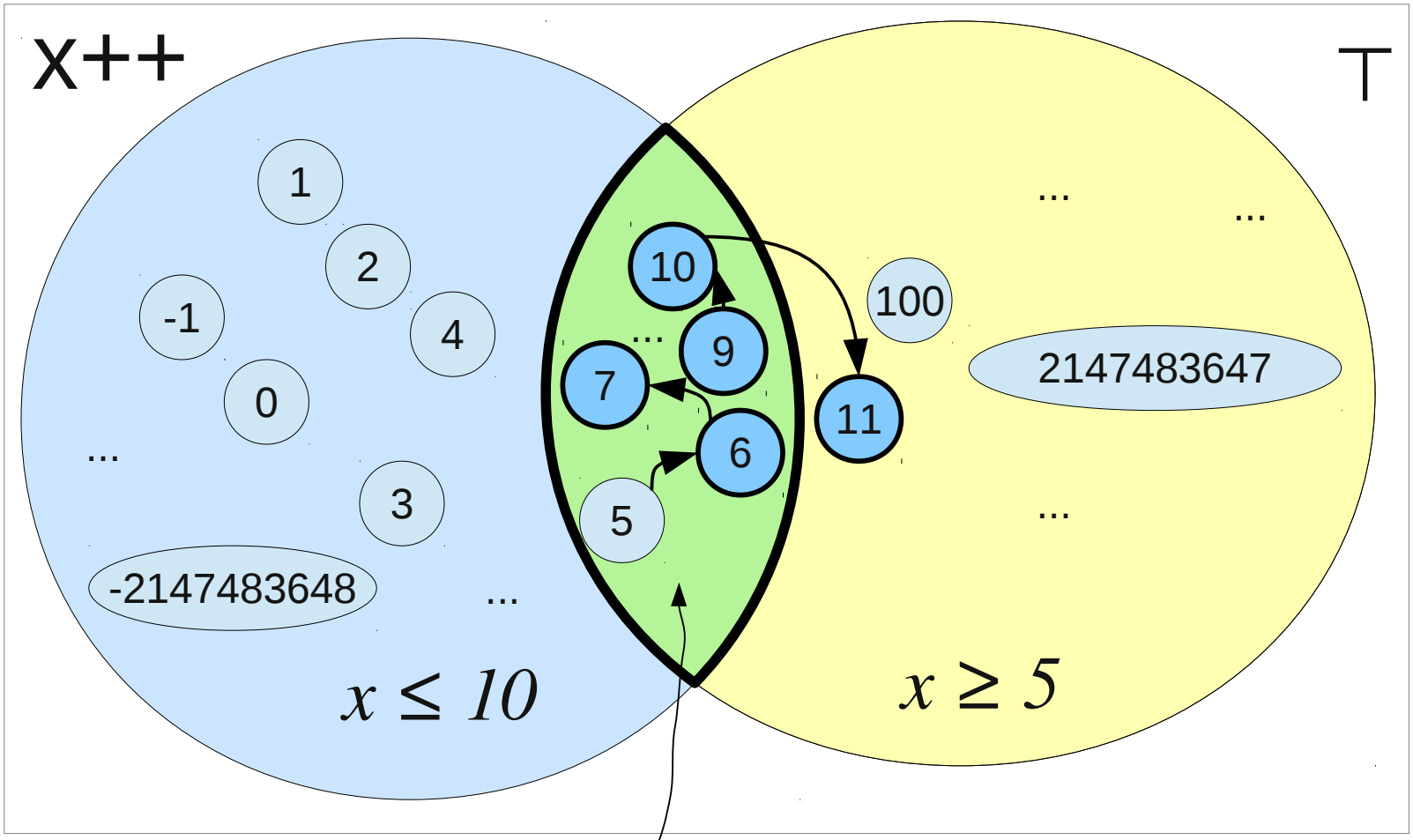
Предикатная абстракция (3)



$x \geq 5 \wedge x \leq 10$

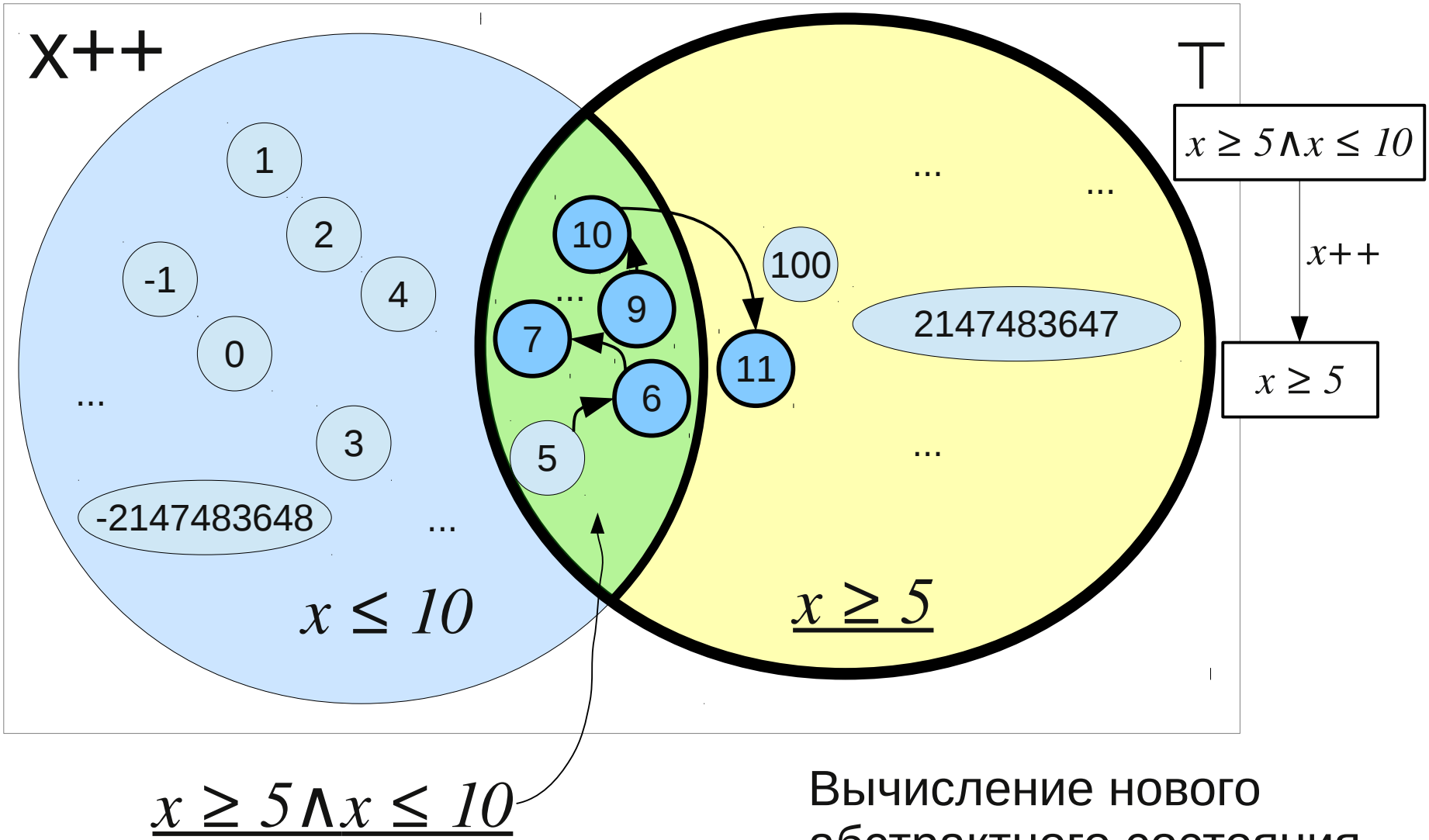
Переходы между состояниями

Предикатная абстракция (4)



$x \geq 5 \wedge x \leq 10$ Переход из абстрактного состояния

Предикатная абстракция (5)

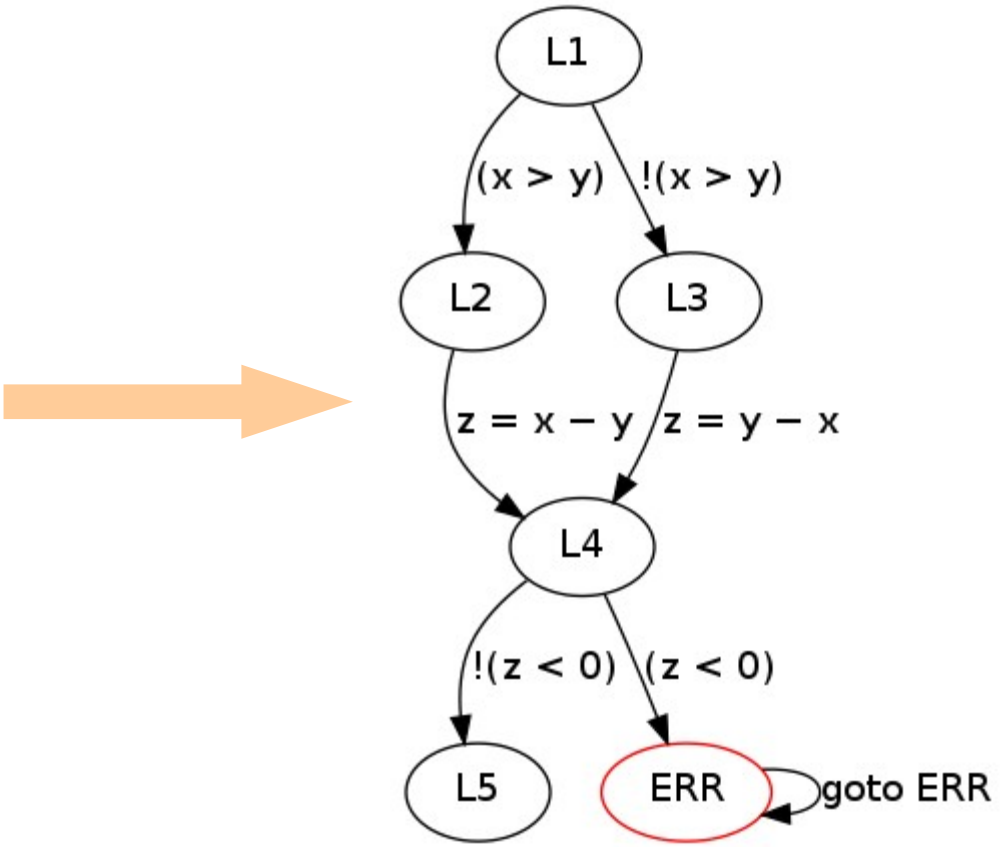


Вычисление нового абстрактного состояния

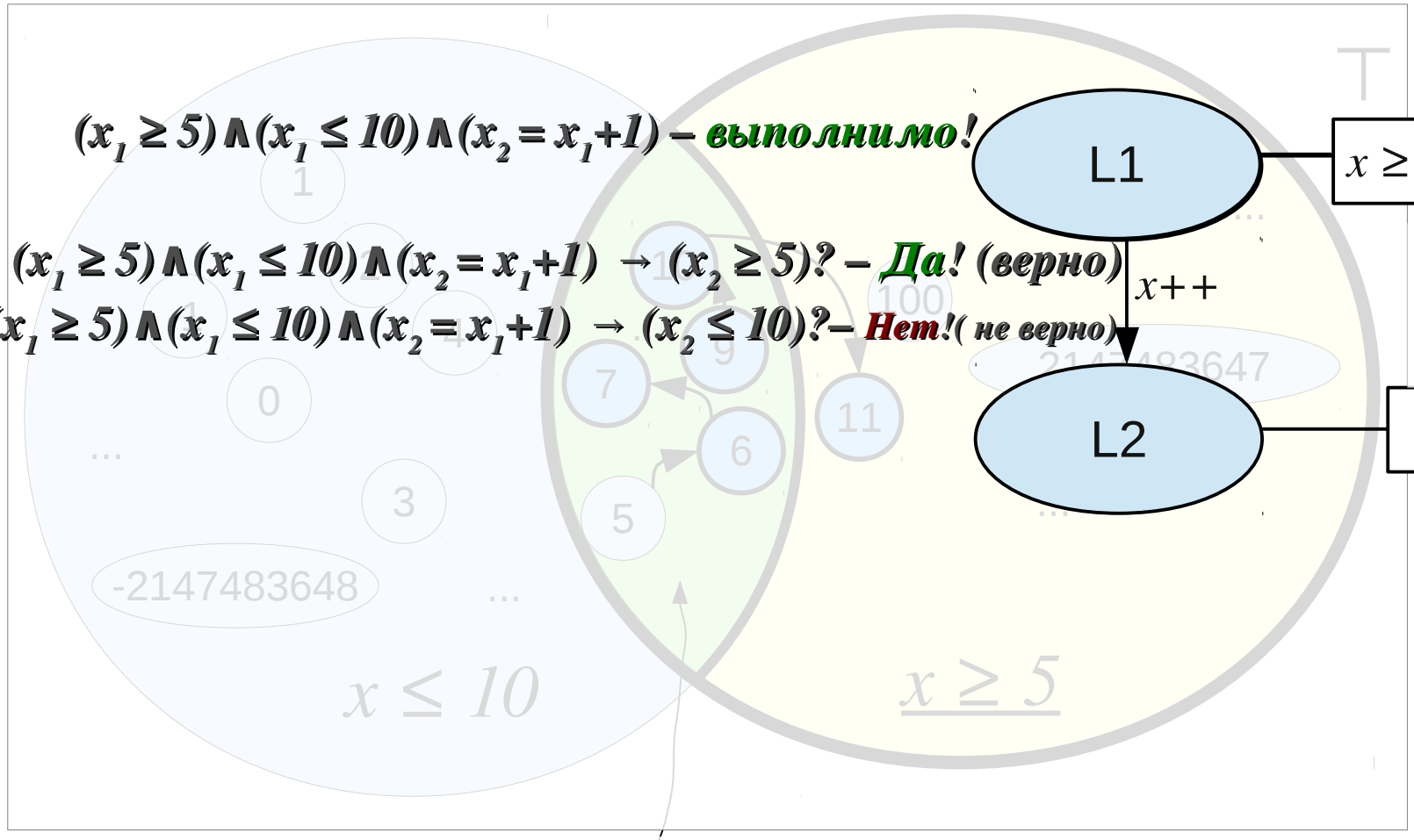
Построение ГПУ

```
L1: if (x > y)
L2:    z = x - y;
    else
L3:    z = y - x;

L4: if (!(z >= 0))
ERROR: goto ERROR;
L5:
```

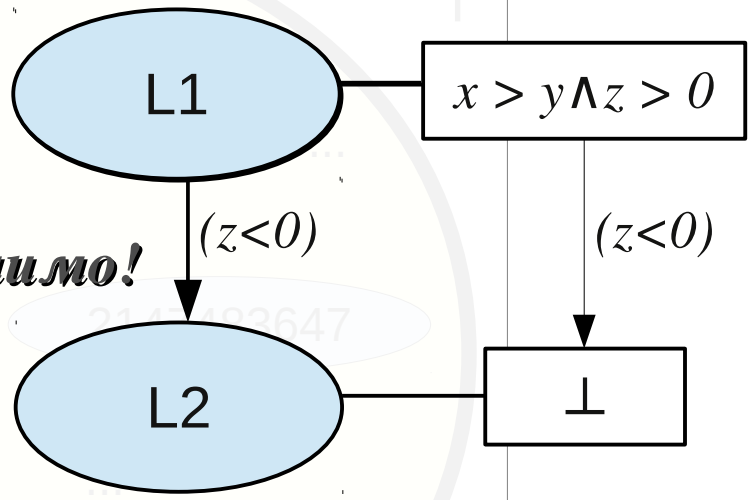


Предикатная абстракция (6)



Предикатная абстракция (7)

$(x_1 > y_1) \wedge (z_1 > 0) \wedge (z_1 < 0)$ – ***не выполнимо!***



$x \leq 10$

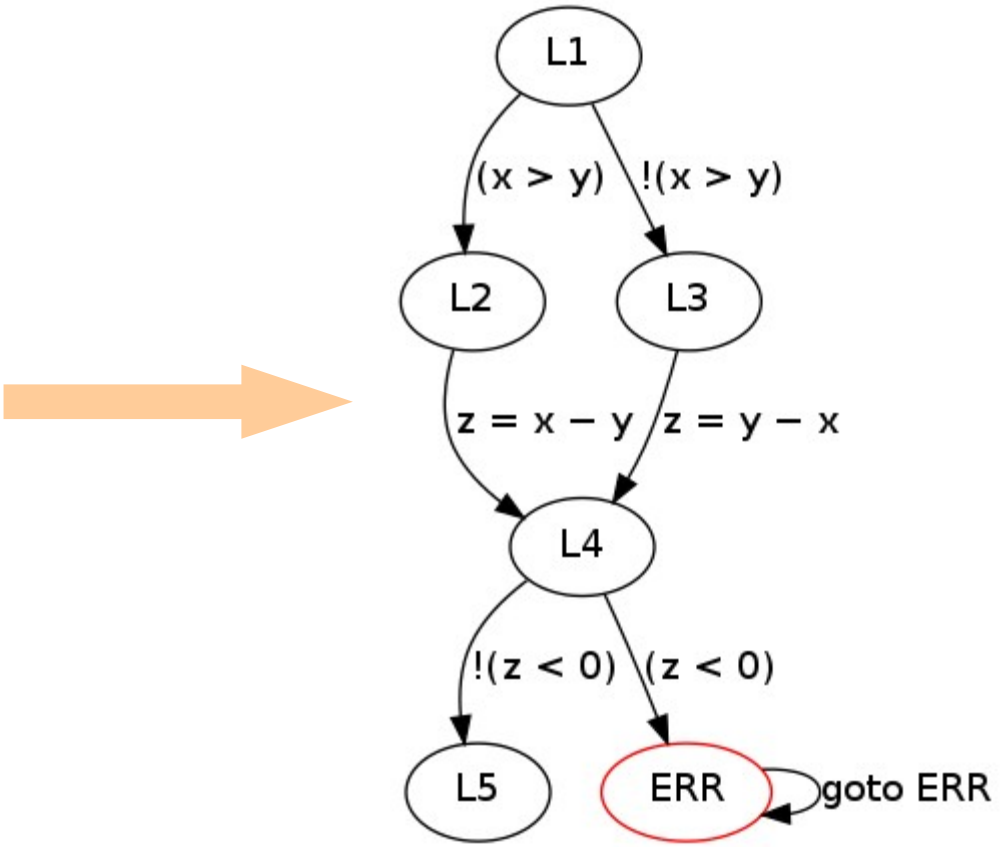
$x \geq 5$

$x \geq 5 \wedge x \leq 10$

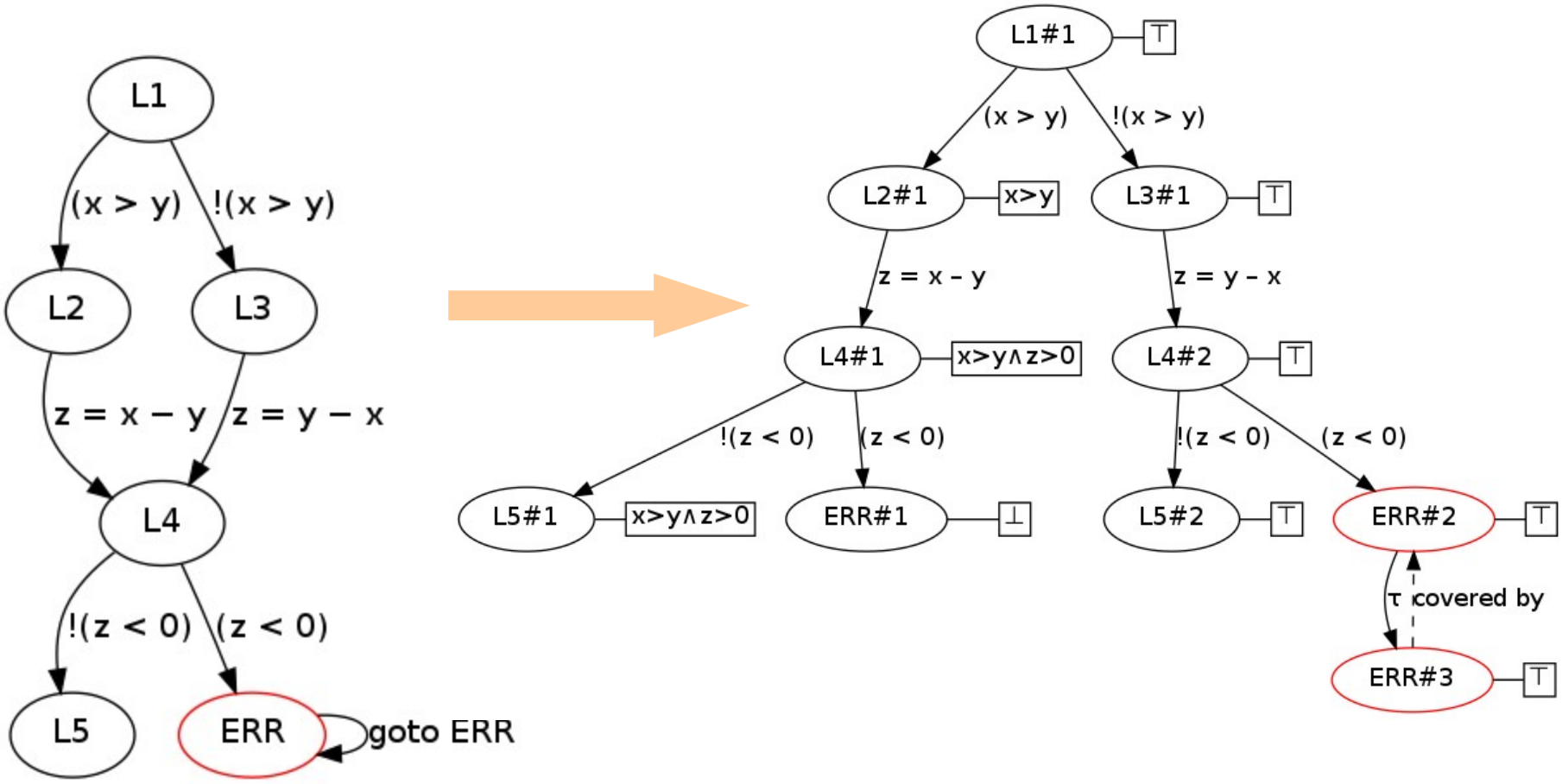
Построение ГПУ

```
L1: if (x > y)
L2:    z = x - y;
    else
L3:    z = y - x;

L4: if (!(z >= 0))
ERROR: goto ERROR;
L5:
```

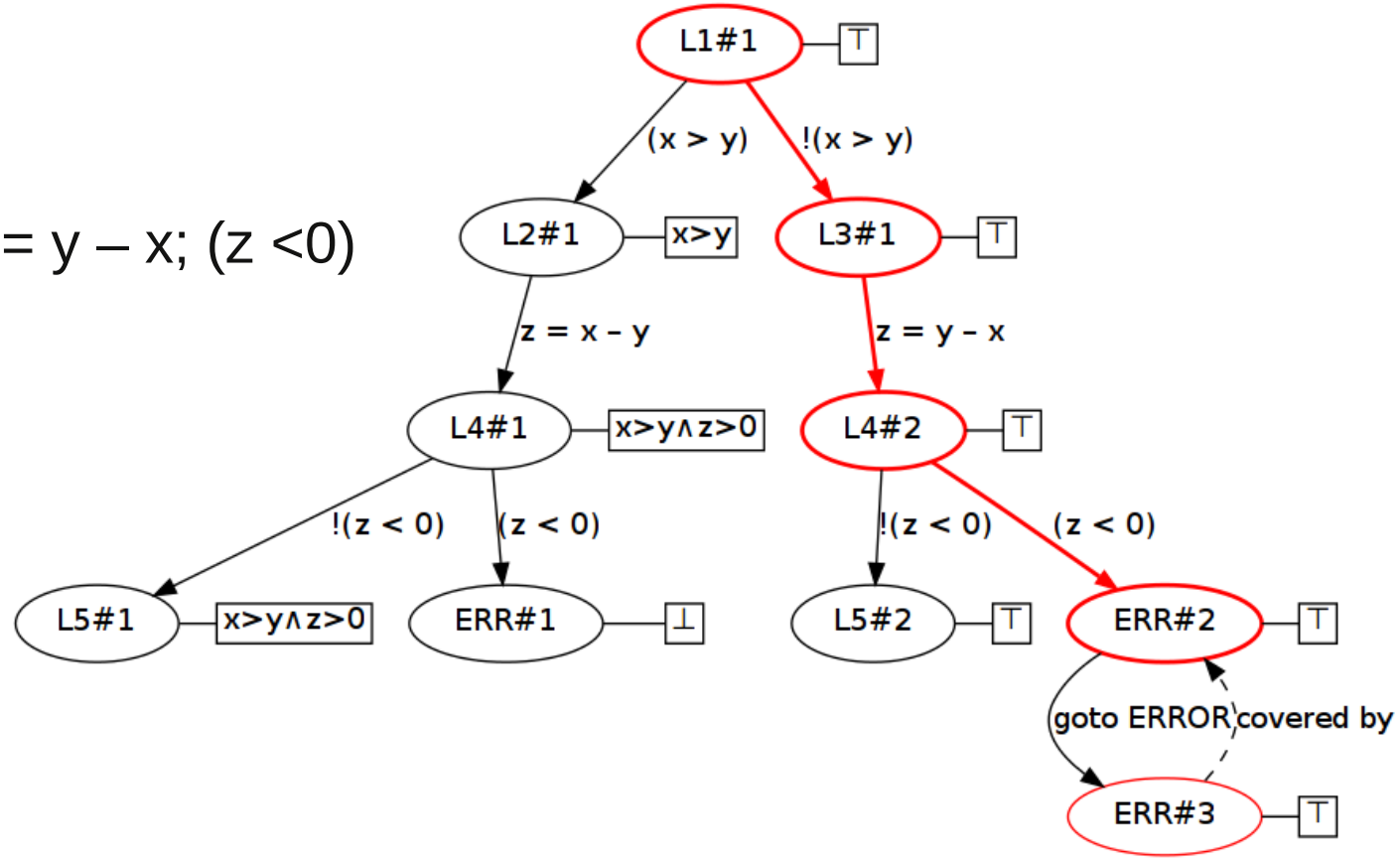


Построение АДД



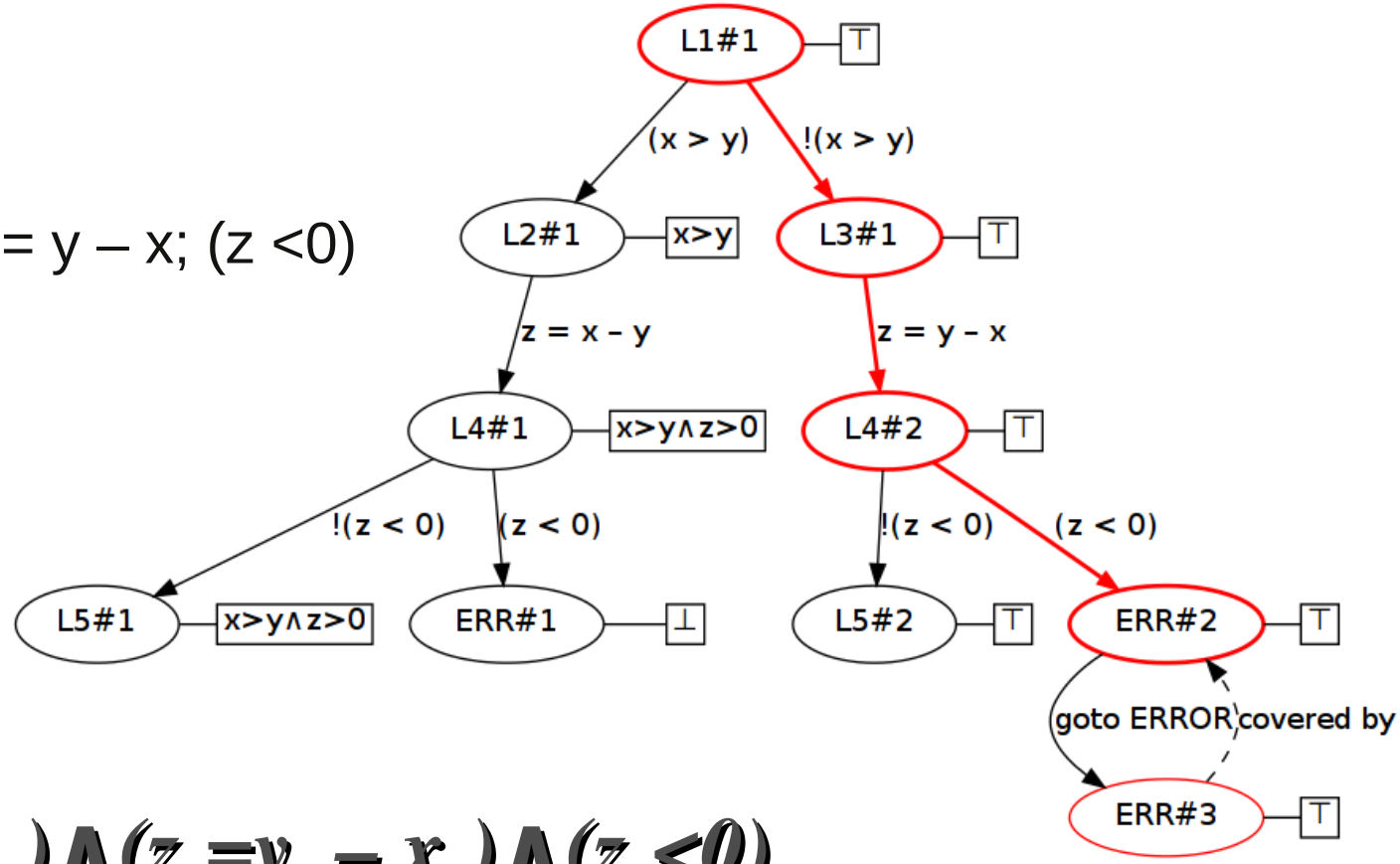
Контрпример

!(x > y); z = y – x; (z <0)



Формула пути

!(x > y); z = y – x; (z <0)

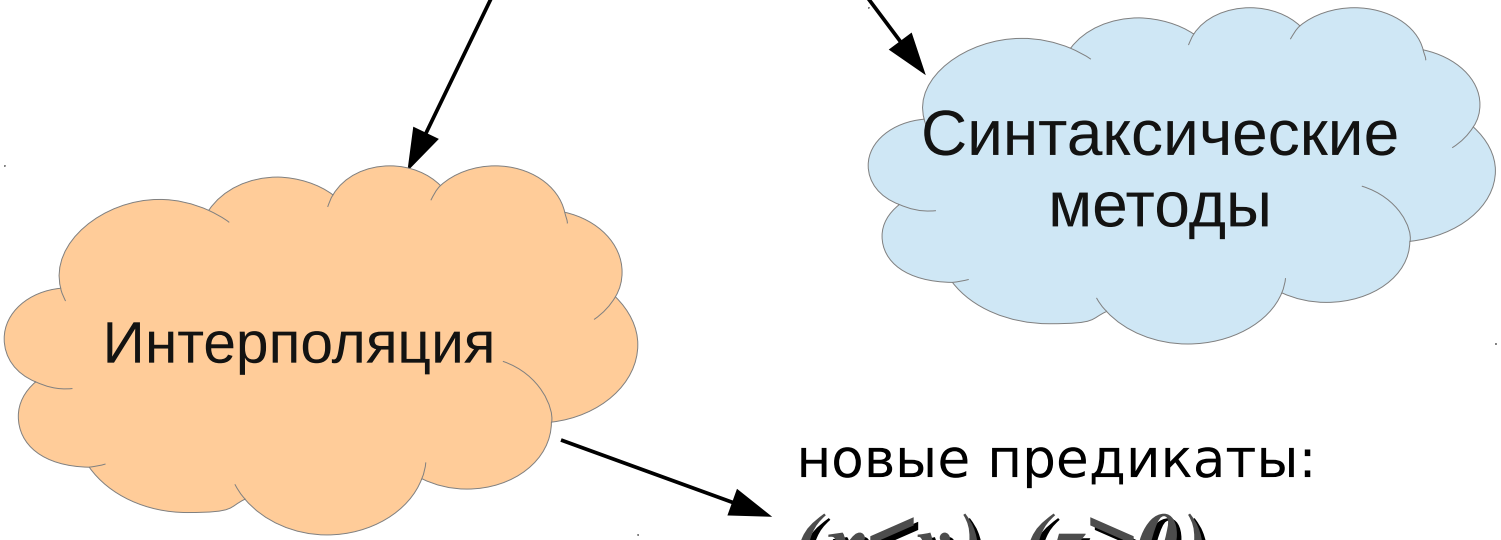


$$\neg(x_1 > y_1) \wedge (z_2 = y_1 - x_1) \wedge (z_2 < 0)$$

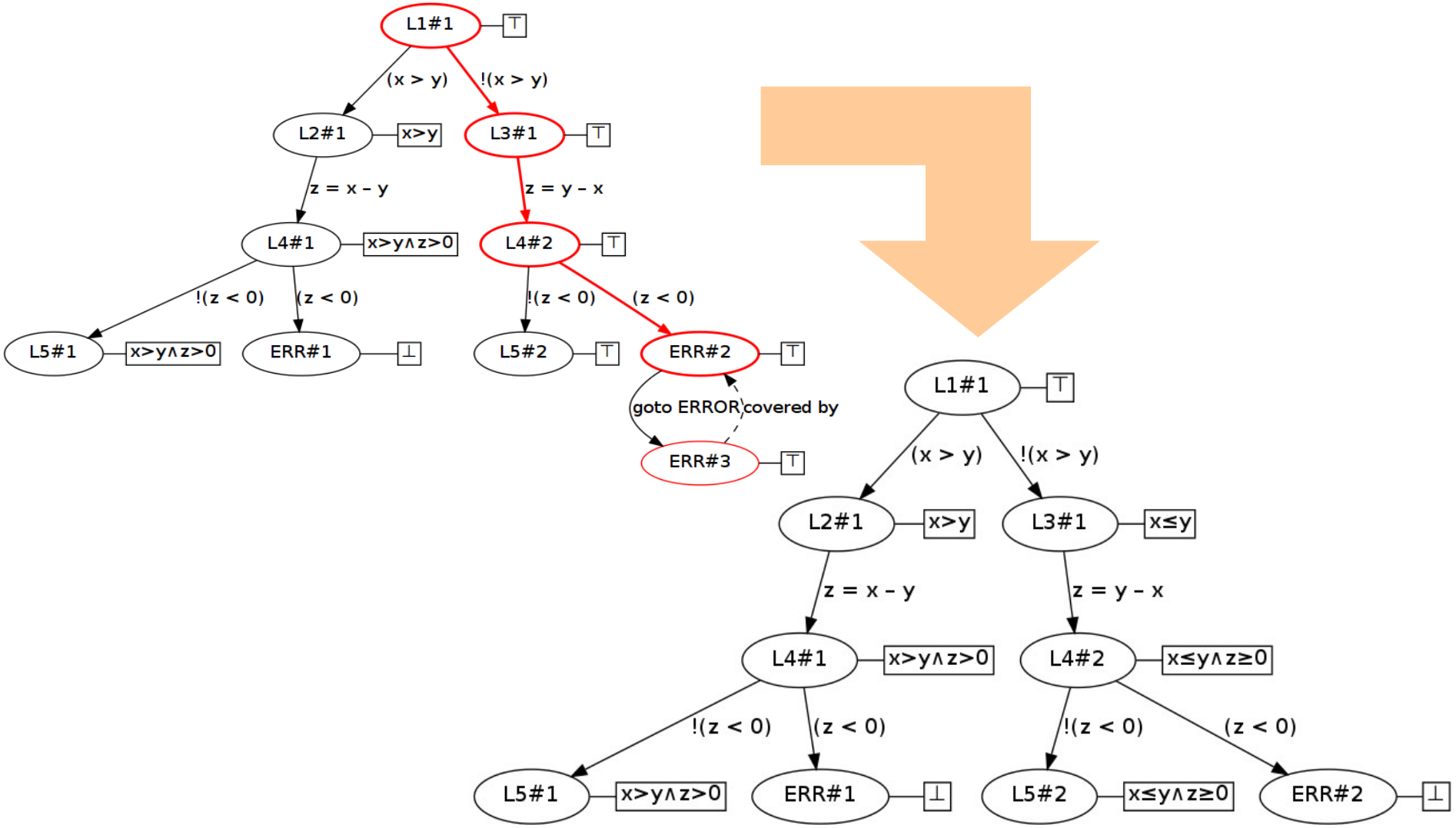
Получение предикатов

!(x > y); z = y – x; (z <0)

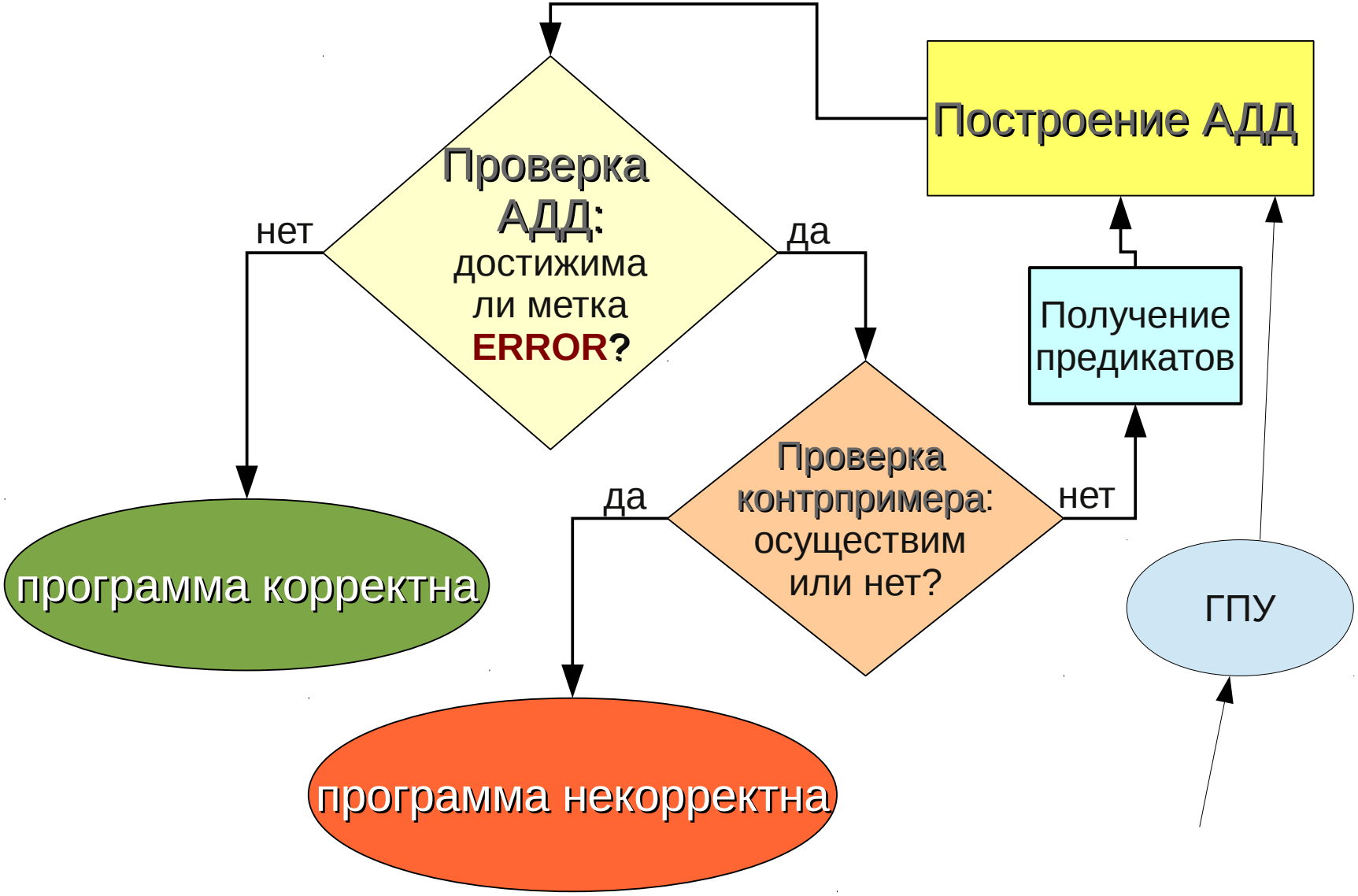
$\neg(x_1 > y_1) \wedge (z_2 = y_1 - x_1) \wedge (z_2 < 0)$ – **не** выполнима



Уточнение абстракции



Подход CEGAR



Подход CEGAR

Couner-Example Guilded Abstraction Refinement – уточнение абстракции по контрпримеру

- Построение графа потока управления программы
- Разворачивание ГПУ в дерево достижимости
- В вершинах дерева хранятся логические формулы над некоторым набором предикатов
- Набор предикатов выводится во время верификации – по контрпримерам
- Чаще всего используются SMT-решатели

Подход CEGAR

Преимущества

- проверка программ с циклами за счет операции покрытия
- хорошая масштабируемость (до 20 – 30 тыс. строк кода)

Недостатки – низкая точность для

- побитовых операций
- целочисленной арифметики

  (метка ERROR недостижима!)
- массивов, структур, указателей
- адресной арифметики

Подход BMC

Bounded Model Checking

- Ограничение числа итераций циклов
- Кодирование путей выполнения в виде логических формул
- Побитовая точность
- Указатели моделируются через массивы или функции (используя возможности SAT-реш.)

```
if (x > y)
    z = x - y;
else
    z = y - x;

if (!(z >= 0))
ERROR: goto ERROR;
```


$$((BV\_GT(x_1, y_1) \wedge z_2 = x_1 - y_1) \vee \neg(BV\_GT(x_1, y_1) \wedge z_2 = y_1 - x_1)) \wedge \neg BV\_GT(z_2, BV\_ZERO)$$

SAT (x_1=2147483647, y_1= –2147483648, z_2=–1)

Подход BMC

Преимущества – точное моделирование

- побитовых операций
- целочисленной арифметики (`int`)
- массивов, структур, указателей
- адресной арифметики

Недостатки

- ограниченное число итераций циклов
- плохо масштабируется на большие программы (в том числе из-за использования SAT)

CEGAR и BMC

	CEGAR	BMC
побитовые операции	крайне неточно	точно
целочисленная арифметика	только линейная, как математические целые	точное побитовое моделирование
массивы, структуры, указатели	частично поддерживаются через анализ алиасов	поддерживается большинство возможностей C
адресная арифметика	не поддерживается	поддерживается
масштабируемость	до 20-30 тыс. строк	для драйверов плохо применим

Результаты соревнований SV-COMP'2013

	Competition candidate	BLAST 2.7.1	CPAchecker-Explicit 1.1.10	CPAchecker-SeqCom 1.1.10	UFO 2012-10-22	ESBMC 1.20	LLBMC 2012-10-23	Predator 2012-10-20
	Representing Jury Member	Vadim Mutilin	Stefan Löwe	Philipp Wendler	Arie Gurfinkel	Lucas Cordeiro	Carsten Sinz	Tomas Vojnar
	Affiliation	Moscow, Russia	Passau, Germany	Passau, Germany	Pittsburgh, USA	Manaus, Brazil	Karlsruhe, Germany	Brno, Czechia
Битовые операции	BitVectors 32 files, max score: 60	--	16 86 s	**17** **190 s**	--	**24** **480 s**	**60** **36 s**	-75 95 s
	ControlFlowInteger 94 files, max score: 146	93 7 100 s	**143** **1 200 s**	**141** **3 400 s**	**146** **450 s**	90 * 17 000 s	--	-27 650 s
Драйверы	DeviceDrivers64 1237 files, max score: 2419	**2 338** **2 400 s**	**2 340** **9 700 s**	2 186 30 000 s	**2 408** **2 500 s**	2 233 46 000 s	--	0 0 s
Указатели	HeapManipulation 28 files, max score: 48	--	22 30 s	**22** **29 s**	--	--	**32** **310 s**	**40** **2.3 s**
	MemorySafety 36 files, max score: 54	--	0 0 s	0 0 s	--	**3** **1 300 s**	**24** **38 s**	**52** **61 s**

CEGAR

BMC

Shape-анализ

Shape-анализ

Анализ состояния объектов кучи с помощью shape-графов

- Вершины – переменные программы (указатели) и объекты в куче
- Дуги – отоношение “указывает на”, “равно”, “не равно”,..
- Есть операции покрытия и слияния shape-графов
- Специальные шаблоны графов (для связных списков)
- Решатели обычно не используются (используются встроенные алгоритмы)

Shape-анализ

Преимущества – поддерживает

- проверку безопасности работы с памятью
- связные списки любой длины
- структуры
- адресную арифметику

Недостатки – не поддерживает

- целые числа (очень ограниченно)
- битовые операции

Анализ указателей в BLAST

Анализ Андерсена, отдельная переменная для каждого поля каждой структуры

- Полезен во многих случаях!
  - `kfree_skb(info->rx_skb);`
  - `info->rx_skb = NULL;`

```
...
dtl1_receive(info);
```

  - `dtl1_receive(dtl1_info_t *info)`

```
{
  ...
  if (info->rx_skb)
    kfree_skb(info->rx_skb);
```

Здесь info – *мэй*-алиасы

Вызов недостижим

Указатели в драйверах

- ```
  for (i = 0; i < RIONET_RX_RING_SIZE; i++)
      kfree_skb(rnet->rx_skb[i]);

  // RIONET_RX_RING_SIZE defaults to 128
  ```
- ```
  while (db->rx_avail_cnt) {
      kfree_skb(db->rx_ready_ptr->rx_skb_ptr);
      db->rx_ready_ptr = db->rx_ready_ptr->next_rx_desc;
      db->rx_avail_cnt--;
   }
  // db->rx_avail_cnt <= RX_DESC_CNT == 32
  ```
- ```
  nf_conntrack_put_reasm(skb->nfct_reasm);
  ```

Массивы

Списки

Алиасинг поля и структуры
  ```

Указатели в драйверах (2)

```
static void bdx_tx_free_skbs(struct bdx_priv *priv)
{
    struct txdb *db = &priv->txdb;
    while (db->rptr != db->wptr) {
        kfree_skb(db->rptr->addr.skb);
        ++db->rptr;
    }
}
```

Адресная арифметика

- В **`struct`** `bdx_priv` *26* полей. *11* из них – структуры или указатели на структуры, в некоторых по *25* и более полей, из которых некоторые – структуры... И при передаче параметра мы должны обновить каждое подполе в каждом мэй-алиасе `priv`

Анализ указателей

BLAST и другие подходы

Инструмент/подход	Указат.	Структуры	Масс.	Рекурсивные структуры данных	Адресная арифм.	Эффективн.
BLAST с *"глубиной замыкания"*	+	± (finite depth)	-	-	-	+
Оптимищация **BLAST** – *"бесконечной глубиной замыкания"*	+	+	-	-	-	+
BLAST с *ленивым shape-анализом ("BLAST 3.0")*	+	+	-	+	-	?
Bounded Model Checking	+	+	+	± (finite depth)	+	-
CPAchecker с предикатной абстракцией (текущая реализация)	+	-	-	-	-	+
Наш подход (неинтерпретируемые функции)	+	+	+	± (finite depth)	+	?

Предложенный подход (1)

Какова *основная* идея?

- Очень ***простая***, *не эффективная, но* ***точная*** модель памяти:
- M – память – отображение адреса в значение
- M1, M2 – состояния памяти

```
rnet->rx_skb[i]->users--; →
```

$$M2 = store(M1,$$

$$M1[M1[\texttt{rnet}] + offsetof(\texttt{struct rionet\_private}, \texttt{rx\_skb}) + M1[i]] + offsetof(\texttt{struct sk\_buff}, \texttt{users}),$$

$$M1[M1[M1[\texttt{rnet}] + offsetof(\texttt{struct rionet\_private}, \texttt{rx\_skb}) + M1[i]] + offsetof(\texttt{struct sk\_buff}, \texttt{users})] - 1)$$

Предложенный подход (2)

Совсем не эффективно...

- Большинство SMT-решателей не поддерживают интерполяцию массивов
- Поэтому используем ***неинтерпретируемые функции***
- Конгруэнтное замыкание: $a = b \rightarrow f(a) = f(b)$
- **Нет** операции *store(·,·,·)*
- $m_1(a_1) = 1$, $m_2(a_2) = 2$, $m_2(a_1) = ?$
- Приходится явно записывать сохранение старых значений
- $a_2 \neq a_1 \rightarrow m_2(a_1) = m_1(a_1)$ и т.д. для всех a_i

Предложенный подход (3)

Как мы представляем адреса областей пам.?

- Одна неинт. константа на одну область
- Адреса положительны $(b_i > 0)$
- Области не пересекаются:

  $B(b_i + k) = i,\ 0 \le k < s,$

  где s – размер области
- $b_i + k = b_j + l \rightarrow B(b_i + k) = B(b_j + l) \rightarrow i = j$
- Поэтому число таких равенств линейно зависит от числа областей

Предложенный подход (4)

Какие предлагаются оптимизации ?

- ***Типизирование***: одна функция на один простой тип данных

пр.:
```
char *,
long int,
struct sk_buff *,
...
```

- ***Чистые переменные*** – переменные, не имеющие алиасов

пр.: `int i;` // просто счетчик
// в коде **нигде** нет `&i'

- ***Оптимизация*** присваивания ***полей структур***, i.e. опускаем посыл импоикации, если смещения заранее не равны

пр.: обновление `skb1->next` *не может* повлиять ***ни на какой*** `skb2->prev`, хотя они и одного типа

Предложенный подход (5)

Дальнейшие оптимизации

- Инициализация ***константами***

напр. нулём: `kzalloc(sizeof(*info), GFP_KERNEL)`

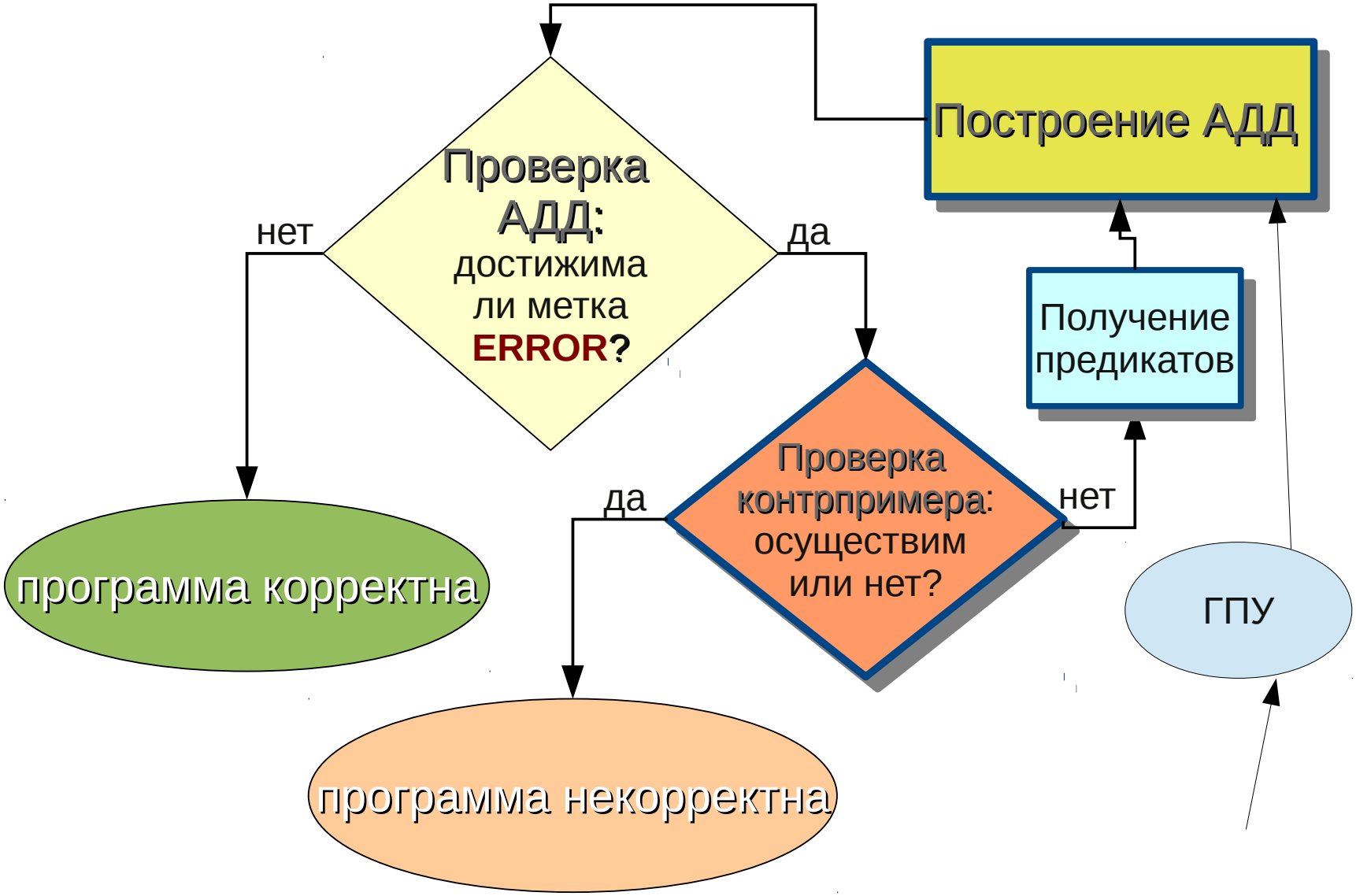
- ***Амортизация*** последовательности присваиваний

e.g.
```
for (i = 0; i < MAX_SKB_FRAGS + 1; i++) {
    lwords = 7 + (i * 3);
    ... /* pad it with 1 lword */
    txd_sizes[i].qwords = lwords >> 1;
    txd_sizes[i].bytes = lwords << 2;
}
```
// ***Нет чтений*** через указатели во всем цикле

// Поэтому обновим память только ***однажды после*** цикла!

- Предварительный анализ алиасов

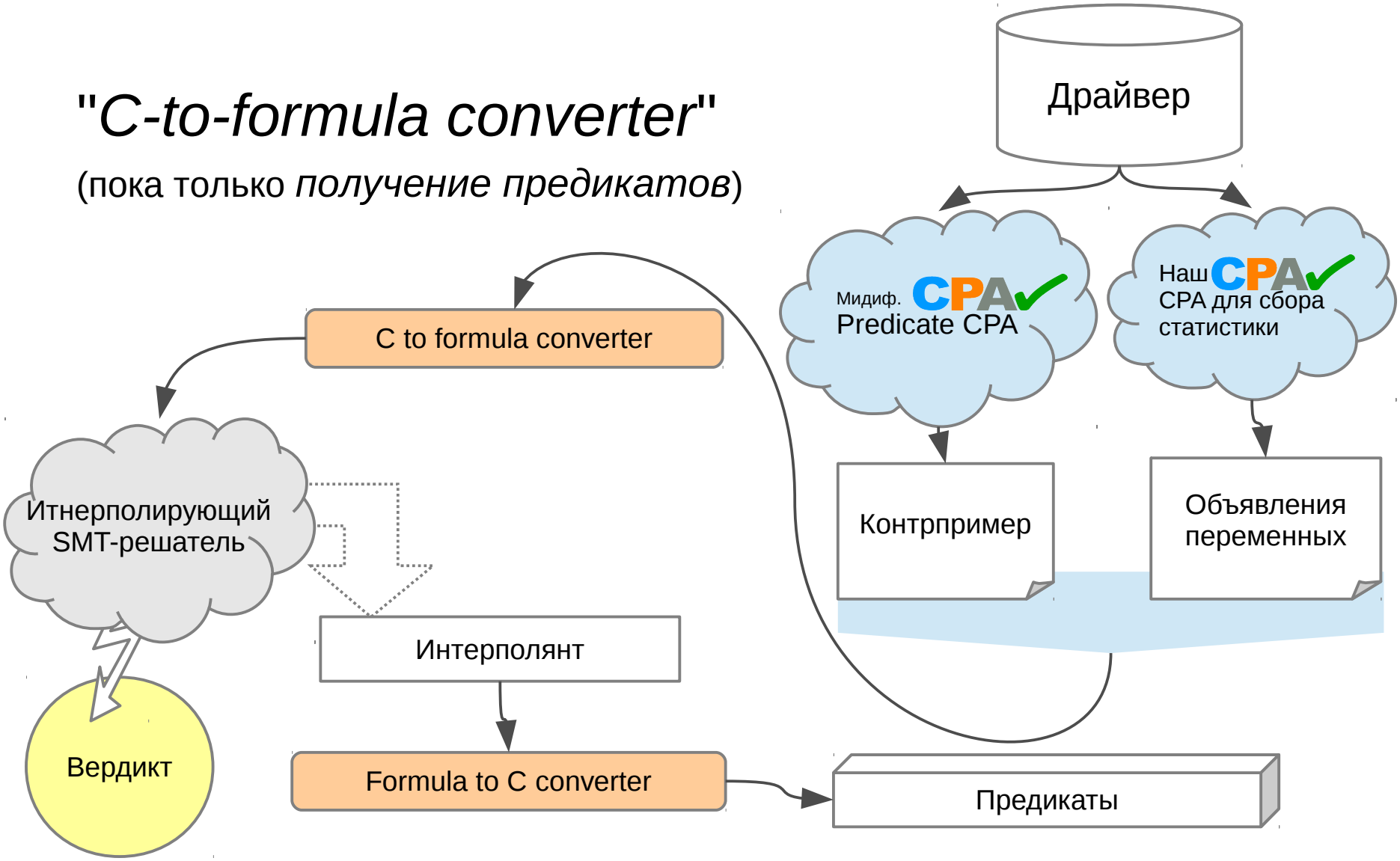
Цикл CEGAR


Построение АДД
Проверка АДД: достижима ли метка ERROR?
нет
да
Получение предикатов
Проверка контрпримера: осуществим или нет?
да
нет
программа корректна
программа некорректна
ГПУ

Прототип преобразователя

"*C-to-formula converter*"

(пока только *получение предикатов*)



Апробация

Размер формулы в KB

Драйвер	Без оптимизаций	Поля структур	Чистые переменные	Вместе
bluetooth/bpa10x.ko (2 skb)	448	369	311	220
bluetooth/dtl1_cs.ko (32 skb)	3700	2400	623	456
isdn/hysdn/hysdn.ko (20 skb)	726	474	101	95
hid/usbhid/usbkbd.ko (no rule model applied)	352	279	119	88
net/usb/cdc-phonet.ko (no rule model applied)	255	166	15	15

Апробация

Пример интерполянта и предикатов

```
true
(and
  (= usbpn_open!!i~1 0.0)
  (= usbpn_open!!dev~1 usbpn_open!!pnd~1))
(and
  (= usbpn_open!!i~1 0.0)
  (= usbpn_open!!dev~1 usbpn_open!!pnd~1))
(= (struct-urb-*~2 (+ (+ usbpn_close!!pnd~1 usbpn_close!!i~1) 66.0)) 0.0)
false
```


~60 операторов в ошибочном пути
Время интерполяции: 0.039s

```
true
usbpn_open::i == 0 && usbpn_open::dev == usbpn_open::pnd
usbpn_open::i == 0 && usbpn_open::dev == usbpn_open::pnd
usbpn_close::pnd->urbs[usbpn_close::i] == 0
false
```

LinuxTesting.org

Спасибо!

Мандрыкин Михаил
mandrykin@ispras.ru

ISP RAS

Institute for System Programming of the Russian Academy of Sciences