Коротко о Scalar Evolution
как LLVM справляется с проверками диапазонов

Макс Казанцев

О чём поговорим

- Коротко о LLVM и LLVM IR
- Проверки диапазонов и издержки на них
- Scalar Evolution и индукционные переменные
- Оптимизации для удаления проверок диапазонов

О докладчике

- 11 лет в компиляторах (JVM, LLVM, нейросети)
- Вклад в LLVM Open-Source Project
  - ~950 патчей всего
  - Top-3 контрибьютор в Scalar Evolution
- Пишу статьи о компиляторах на Хабре
  - https://habr.com/ru/users/xortator/
- Читаю лекции в НГУ
- Работаю в Сбере

Коротко о LLVM
и LLVM IR

Мы живём в эпоху LLVM

В наше время «написать свой язык программирования» обычно означает «написать свой фронтэнд для трансляции своего языка в LLVM IR»

Компиляция в LLVM

Пример LLVM IR

```c
int example(int n, int *arr) {
    int sum = 0;
    for (int i = 0; i < n; i++)
        sum += i * arr[i];
    return sum;
}
```

```
define i32 @example(i32 %n, ptr %arr) {
entry:
  %cmp5 = icmp sgt i32 %n, 0
  br i1 %cmp5, label %for.body.preheader, label %for.cond.cleanup

for.body.preheader:
  %wide.trip.count = zext nneg i32 %n to i64
  br label %for.body

for.cond.cleanup:
  %sum.0.lcssa = phi i32 [ 0, %entry ], [ %add, %for.body ]
  ret i32 %sum.0.lcssa

for.body:
  %indvars.iv = phi i64 [ 0, %for.body.preheader ], [ %indvars.iv.next, %for.body ]
  %sum.06 = phi i32 [ 0, %for.body.preheader ], [ %add, %for.body ]
  %arrayidx = getelementptr inbounds i32, ptr %arr, i64 %indvars.iv
  %0 = load i32, ptr %arrayidx, align 4
  %1 = trunc i64 %indvars.iv to i32
  %mul = mul nsw i32 %0, %1
  %add = add nsw i32 %mul, %sum.06
  %indvars.iv.next = add nuw nsw i64 %indvars.iv, 1
  %exitcond.not = icmp eq i64 %indvars.iv.next, %wide.trip.count
  br i1 %exitcond.not, label %for.cond.cleanup, label %for.body
}
```

Устройство LLVM IR

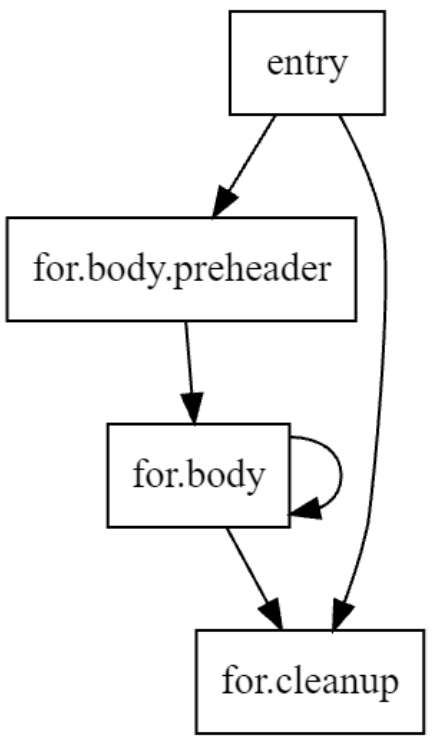
Функция состоит из базовых блоков, наполненных инструкциями.

Весь блок исполняется подряд, после чего управление может быть передано в другой блок.

```
define i32 @example(i32 %n, ptr %arr) {
entry:
  %cmp5 = icmp sgt i32 %n, 0
  br i1 %cmp5, label %for.body.preheader, label %for.cond.cleanup

for.body.preheader:
  %wide.trip.count = zext nneg i32 %n to i64
  br label %for.body

for.cond.cleanup:
  %sum.0.lcssa = phi i32 [ 0, %entry ], [ %add, %for.body ]
  ret i32 %sum.0.lcssa

for.body:
  %indvars.iv = phi i64 [ 0, %for.body.preheader ], [ %indvars.iv.next, %for.body ]
  %sum.06 = phi i32 [ 0, %for.body.preheader ], [ %add, %for.body ]
  %arrayidx = getelementptr inbounds i32, ptr %arr, i64 %indvars.iv
  %0 = load i32, ptr %arrayidx, align 4
  %1 = trunc i64 %indvars.iv to i32
  %mul = mul nsw i32 %0, %1
  %add = add nsw i32 %mul, %sum.06
  %indvars.iv.next = add nuw nsw i64 %indvars.iv, 1
  %exitcond.not = icmp eq i64 %indvars.iv.next, %wide.trip.count
  br i1 %exitcond.not, label %for.cond.cleanup, label %for.body
}
```

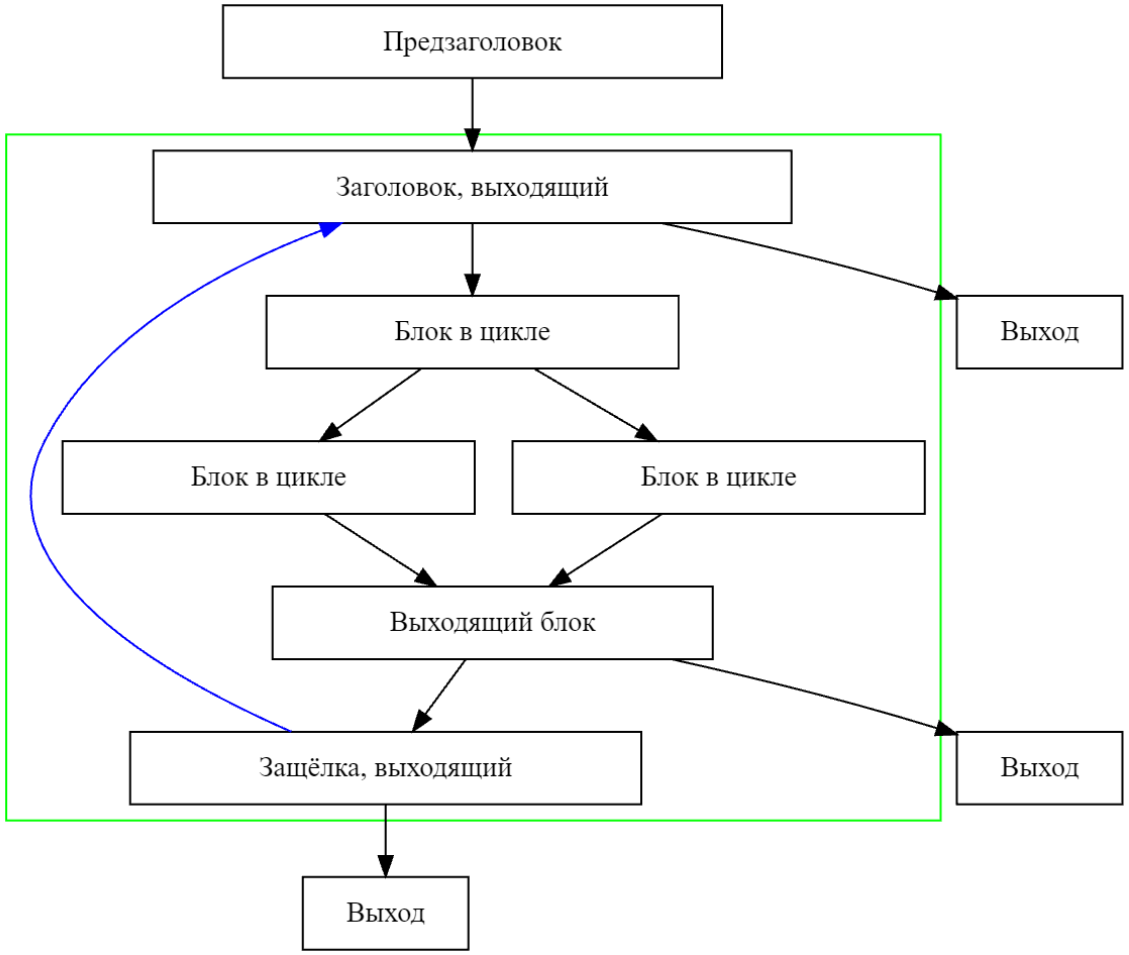
Устройство LLVM IR

Базовые блоки и переходы между ними образуют граф потока управления.




```
define i32 @example(i32 %n, ptr %arr) {
entry:
  %cmp5 = icmp sgt i32 %n, 0
  br i1 %cmp5, label %for.body.preheader, label %for.cond.cleanup

for.body.preheader:
  %wide.trip.count = zext nneg i32 %n to i64
  br label %for.body

for.cond.cleanup:
  %sum.0.lcssa = phi i32 [ 0, %entry ], [ %add, %for.body ]
  ret i32 %sum.0.lcssa

for.body:
  %indvars.iv = phi i64 [ 0, %for.body.preheader ], [ %indvars.iv.next, %for.body ]
  %sum.06 = phi i32 [ 0, %for.body.preheader ], [ %add, %for.body ]
  %arrayidx = getelementptr inbounds i32, ptr %arr, i64 %indvars.iv
  %0 = load i32, ptr %arrayidx, align 4
  %1 = trunc i64 %indvars.iv to i32
  %mul = mul nsw i32 %0, %1
  %add = add nsw i32 %mul, %sum.06
  %indvars.iv.next = add nuw nsw i64 %indvars.iv, 1
  %exitcond.not = icmp eq i64 %indvars.iv.next, %wide.trip.count
  br i1 %exitcond.not, label %for.cond.cleanup, label %for.body
}
```

Устройство цикла

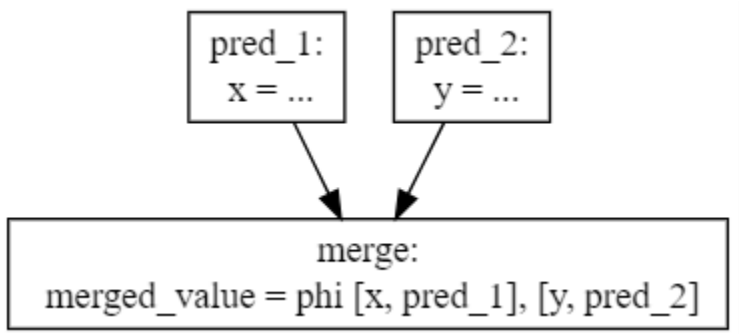


- Предзаголовок (Preheader) – блок вне цикла, из которого есть переход внутрь цикла
- Заголовок (Header) – блок в цикле, в который мы входим извне
- Обратное ребро (Backedge) – ребро изнутри цикла в заголовок
- Защёлка (Latch) – блок, откуда исходит обратное ребро
- Выходящий блок (Exiting Block) – блок, после которого можно выйти из цикла

Ф-узлы (Ф-функции, финоды)

В LLVM значения присваиваются лишь единожды (SSA-форма) и не могут быть переприсвоены. Для того, чтобы слить значения из разных блоков, используется Ф-узел.

Переменные-счётчики представляются Ф-узлами, сливающими стартовое и инкрементированное значение.



Ф-узлы (Ф-функции, финоды)

В LLVM значения присваиваются лишь единожды (SSA-форма) и не могут быть переприсвоены. Для того, чтобы слить значения из разных блоков, используется Ф-узел.

Переменные-счётчики представляются Ф-узлами, сливающими стартовое и инкрементированное значение.

```
define i32 @example(i32 %n, ptr %arr) {
entry:
  %cmp5 = icmp sgt i32 %n, 0
  br i1 %cmp5, label %for.body.preheader, label %for.cond.cleanup

for.body.preheader:
  %wide.trip.count = zext nneg i32 %n to i64
  br label %for.body

for.cond.cleanup:
  %sum.0.lcssa = phi i32 [ 0, %entry ], [ %add, %for.body ]
  ret i32 %sum.0.lcssa

for.body:
  %indvars.iv = phi i64 [ 0, %for.body.preheader ], [ %indvars.iv.next, %for.body ]
  %sum.06 = phi i32 [ 0, %for.body.preheader ], [ %add, %for.body ]
  %arrayidx = getelementptr inbounds i32, ptr %arr, i64 %indvars.iv
  %0 = load i32, ptr %arrayidx, align 4
  %1 = trunc i64 %indvars.iv to i32
  %mul = mul nsw i32 %0, %1
  %add = add nsw i32 %mul, %sum.06
  %indvars.iv.next = add nuw nsw i64 %indvars.iv, 1
  %exitcond.not = icmp eq i64 %indvars.iv.next, %wide.trip.count
  br i1 %exitcond.not, label %for.cond.cleanup, label %for.body
}
```

Проверки диапазонов
и издержки* на них

* В рамках данного доклада под издержками, ценой, стоимостью и т.п. понимаются только дополнительные расходы процессорного времени на совершение тех или иных операций, а не их финансовое выражение.

Индукционные переменные

```
int example(int n, int *arr) {
    int sum = 0;
    for (int i = 0; i < n; i++)
        sum += i * arr[i];
    return sum;
}
```

Переменная, значения которой в цикле образуют арифметическую прогрессию, называется индукционной переменной.

$$IV_n = start + n * step$$

Простейший пример индукционной переменной – счётчик цикла.

Индукционные переменные

```
int example(int start, int step, int end) {
    ...
    for (int i = start; i < end; i += step) {
        // i = start + step * num_iter
        int j = i + 10;
        // j =  (start + 10) + step * num_iter
        int k = j * 2;
        // k = (2 * start + 20) + (step * 2) * num_iter
        int w = i + k;
        // w = (3 * start + 30) + (step * 3) * num_iter
        ...
    }
    ...
}
```

Проверки диапазонов

Если мы хотим иметь предсказуемое поведение (например, исключение) при доступе к некорректному индексу массивоподобной структуры, то нам нужны проверки диапазонов.

- Могут быть зашиты в язык (массивы в Java)
- Могут быть имплементированы в стандартном классе (std::vector::at)
- Могут быть в пользовательском коде

Проверки диапазонов

```
class Square {
    int arr_1[];
    int arr_2[];
    int arr_3[];

    void do_magic(int start, int end, int step) {
        for (int i = start; i < end; i += step) {
            arr_1[i - 2] = arr_2[i * 2] + arr_3[i + 7];
        }
    }
}
```

Проверки диапазонов

```
class Square {
    int arr_1[];
    int arr_2[];
    int arr_3[];

    void do_magic(int start, int end, int step) {
        for (int i = start; i < end; i += step) {
            if (i * 2 < 0 || i * 2 >= arr_2.length)
                throw ...
            int tmp_1 = load(arr_2, i * 2);
            if (i + 7 < 0 || i + 7 >= arr_3.length)
                throw ...
            int tmp_2 = load(arr_3, i + 7);
            int tmp_3 = tmp_1 + tmp_2;
            if (i - 2 < 0 || i - 2 >= arr_1.length)
                throw ...
            store(arr_1, tmp_3, i - 2);
        }
    }
}
```

01 Скорее всего, проверки диапазонов падать не будут

02 Но не выполнять их вовсе – нельзя (это требование спецификации языка)

03 Хочется делать таких проверок как можно меньше, а ненужные проверки – убрать совсем

Издержки на выполнение проверок

```
class Square {
    int arr_1[];
    int arr_2[];
    int arr_3[];

    void do_magic(int start, int end, int step) {
        for (int i = start; i < end; i += step) {
            if (i * 2 < 0 || i * 2 >= arr_2.length)
                throw ...
            int tmp_1 = load(arr_2, i * 2);
            if (i + 7 < 0 || i + 7 >= arr_3.length)
                throw ...
            int tmp_2 = load(arr_3, i + 7);
            int tmp_3 = tmp_1 + tmp_2;
            if (i - 2 < 0 || i - 2 >= arr_1.length)
                throw ...
            store(arr_1, tmp_3, i - 2);
        }
    }
}
```

01 Имеют собственную стоимость: минимум 1 ассемблерная инструкция на проверку

02 Мешают высокоуровневым оптимизациям: цикл с проверками сложно векторизовать

03 Увеличивается размер кода – больше кэш-линий, меньше аппаратных оптимизаций

Упрощённый вид проверок

```
class Square {
    int arr_1[];
    int arr_2[];
    int arr_3[];

    void do_magic(int start, int end, int step) {
        for (int i = start; i < end; i += step) {
            if (!(i * 2 <u arr_2.length))
                throw ...
            int tmp_1 = load(arr_2, i * 2);
            if (!(i + 7 <u arr_3.length))
                throw ...
            int tmp_2 = load(arr_3, i + 7);
            int tmp_3 = tmp_1 + tmp_2;
            if (!(i - 2 <u arr_1.length))
                throw ...
            store(arr_1, tmp_3, i - 2);
        }
    }
}
```

<u означает сравнение беззаковых чисел. Основано на том, что отрицательные числа, если их трактовать как беззнаковые – это большие положительные числа (больше SINT_MAX).

Это работает, только если length не может быть отрицательным или больше SINT_MAX.

Scalar Evolution

и индукционные переменные

Дисклеймер

LLVM IR имеет механизм, позволяющий сказать, что знаковое переполнение ведёт к UB (поведение C++). А также имеет механизм, чтобы сказать, что при переполнении всё считается по модулю 2^N (поведение Java).

Далее будем считать, что знаковые переполнения не ведут к неопределённому поведению и всё считается по модулю 2^N (UB несколько усложнит картину).

Что такое Scalar Evolution

Анализ, который представляет результаты исполнения инструкций LLVM IR в виде математических формул.

Фокусируется на том, что было посчитано, а не на том, как именно его посчитали.

Формулы в Scalar Evolution

```
define i32 @example(i32 %x, i32 %y) {
    %a = mul i32 %x, %x
    %b = add i32 %a, %y
    %c = mul i32 %b, %b
    %d = sub i32 %c, %b
    %e = sub i32 %d, %c
    %f = add i32 %e, %a
    ret i32 %f
}
```

```
Printing analysis 'Scalar Evolution Analysis' for function 'example':
Classifying expressions for: @example
  %a = mul i32 %x, %x
  -->  (%x * %x) U: full-set S: full-set
  %b = add i32 %a, %y
  -->  ((%x * %x) + %y) U: full-set S: full-set
  %c = mul i32 %b, %b
  -->  (((%x * %x) + %y) * ((%x * %x) + %y)) U: full-set S: full-set
  %d = sub i32 %c, %b
  -->  (((%x * %x) + %y) * (-1 + (%x * %x) + %y)) U: full-set S: full-set
  %e = sub i32 %d, %c
  -->  (-1 * ((%x * %x) + %y)) U: full-set S: full-set
  %f = add i32 %e, %a
  -->  (-1 * %y) U: full-set S: full-set
Determining loop execution counts for: @example
Compiler returned: 0
```

```
opt -passes='print<scalar-evolution>'
```

https://godbolt.org/z/sx9GnMKKe

Формулы в Scalar Evolution

```
define i32 @example(i32 %x, i32 %y) {
    %a = mul i32 %x, %x
    %b = add i32 %a, %y
    %c = mul i32 %b, %b
    %d = sub i32 %c, %b
    %e = sub i32 %d, %c
    %f = add i32 %e, %a
    ret i32 %f
}
```

```
Printing analysis 'Scalar Evolution Analysis' for function 'example':
Classifying expressions for: @example
  %a = mul i32 %x, %x
  -->  (%x * %x) U: full-set S: full-set
  %b = add i32 %a, %y
  -->  ((%x * %x) + %y) U: full-set S: full-set
  %c = mul i32 %b, %b
  -->  (((%x * %x) + %y) * ((%x * %x) + %y)) U: full-set S: full-set
  %d = sub i32 %c, %b
  -->  (((%x * %x) + %y) * (-1 + (%x * %x) + %y)) U: full-set S: full-set
  %e = sub i32 %d, %c
  -->  (-1 * ((%x * %x) + %y)) U: full-set S: full-set
  %f = add i32 %e, %a
  -->  (-1 * %y) U: full-set S: full-set
Determining loop execution counts for: @example
Compiler returned: 0
```

```
opt -passes='print<scalar-evolution>'
```

https://godbolt.org/z/sx9GnMKKe

Формулы в Scalar Evolution

```
define i32 @example(i32 %x, i32 %y) {
    %a = mul i32 %x, %x
    %b = add i32 %a, %y
    %c = mul i32 %b, %b
    %d = sub i32 %c, %b
    %e = sub i32 %d, %c
    %f = add i32 %e, %a
    ret i32 %f
}
```

```
Printing analysis 'Scalar Evolution Analysis' for function 'example':
Classifying expressions for: @example
  %a = mul i32 %x, %x
  -->  (%x * %x) U: full-set S: full-set
  %b = add i32 %a, %y
  -->  ((%x * %x) + %y) U: full-set S: full-set
  %c = mul i32 %b, %b
  -->  (((%x * %x) + %y) * ((%x * %x) + %y)) U: full-set S: full-set
  %d = sub i32 %c, %b
  -->  (((%x * %x) + %y) * (-1 + (%x * %x) + %y)) U: full-set S: full-set
  %e = sub i32 %d, %c
  -->  (-1 * ((%x * %x) + %y)) U: full-set S: full-set
  %f = add i32 %e, %a
  -->  (-1 * %y) U: full-set S: full-set
Determining loop execution counts for: @example
Compiler returned: 0
```

```
opt -passes='print<scalar-evolution>'
```

https://godbolt.org/z/sx9GnMKKe

Формулы в Scalar Evolution

```
define i32 @example(i32 %x, i32 %y) {
    %a = mul i32 %x, %x
    %b = add i32 %a, %y
    %c = mul i32 %b, %b
    %d = sub i32 %c, %b
    %e = sub i32 %d, %c
    %f = add i32 %e, %a
    ret i32 %f
}
```

```
Printing analysis 'Scalar Evolution Analysis' for function 'example':
Classifying expressions for: @example
  %a = mul i32 %x, %x
  -->  (%x * %x) U: full-set S: full-set
  %b = add i32 %a, %y
  -->  ((%x * %x) + %y) U: full-set S: full-set
  %c = mul i32 %b, %b
  -->  (((%x * %x) + %y) * ((%x * %x) + %y)) U: full-set S: full-set
  %d = sub i32 %c, %b
  -->  (((%x * %x) + %y) * (-1 + (%x * %x) + %y)) U: full-set S: full-set
  %e = sub i32 %d, %c
  -->  (-1 * ((%x * %x) + %y)) U: full-set S: full-set
  %f = add i32 %e, %a
  -->  (-1 * %y) U: full-set S: full-set
Determining loop execution counts for: @example
Compiler returned: 0
```

```
opt -passes='print<scalar-evolution>'
```

https://godbolt.org/z/sx9GnMKKe

Формулы в Scalar Evolution

```
define i32 @example(i32 %x, i32 %y) {
    %a = mul i32 %x, %x
    %b = add i32 %a, %y
    %c = mul i32 %b, %b
    %d = sub i32 %c, %b
    %e = sub i32 %d, %c
    %f = add i32 %e, %a
    ret i32 %f
}
```

```
Printing analysis 'Scalar Evolution Analysis' for function 'example':
Classifying expressions for: @example
  %a = mul i32 %x, %x
  -->  (%x * %x) U: full-set S: full-set
  %b = add i32 %a, %y
  -->  ((%x * %x) + %y) U: full-set S: full-set
  %c = mul i32 %b, %b
  -->  (((%x * %x) + %y) * ((%x * %x) + %y)) U: full-set S: full-set
  %d = sub i32 %c, %b
  -->  (((%x * %x) + %y) * (-1 + (%x * %x) + %y)) U: full-set S: full-set
  %e = sub i32 %d, %c
  -->  (-1 * ((%x * %x) + %y)) U: full-set S: full-set
  %f = add i32 %e, %a
  -->  (-1 * %y) U: full-set S: full-set
Determining loop execution counts for: @example
Compiler returned: 0
```

```
opt -passes='print<scalar-evolution>'
```

https://godbolt.org/z/sx9GnMKKe

Формулы в Scalar Evolution

```
define i32 @example(i32 %x, i32 %y) {
    %a = mul i32 %x, %x
    %b = add i32 %a, %y
    %c = mul i32 %b, %b
    %d = sub i32 %c, %b
    %e = sub i32 %d, %c
    %f = add i32 %e, %a
    ret i32 %f
}
```

```
Printing analysis 'Scalar Evolution Analysis' for function 'example':
Classifying expressions for: @example
  %a = mul i32 %x, %x
  -->  (%x * %x) U: full-set S: full-set
  %b = add i32 %a, %y
  -->  ((%x * %x) + %y) U: full-set S: full-set
  %c = mul i32 %b, %b
  -->  (((%x * %x) + %y) * ((%x * %x) + %y)) U: full-set S: full-set
  %d = sub i32 %c, %b
  -->  (((%x * %x) + %y) * (-1 + (%x * %x) + %y)) U: full-set S: full-set
  %e = sub i32 %d, %c
  -->  (-1 * ((%x * %x) + %y)) U: full-set S: full-set
  %f = add i32 %e, %a
  -->  (-1 * %y) U: full-set S: full-set
Determining loop execution counts for: @example
Compiler returned: 0
```

```
opt -passes='print<scalar-evolution>'
```

https://godbolt.org/z/sx9GnMKKe

Формулы в Scalar Evolution

```
define i32 @example(i32 %x, i32 %y) {
    %a = mul i32 %x, %x
    %b = add i32 %a, %y
    %c = mul i32 %b, %b
    %d = sub i32 %c, %b
    %e = sub i32 %d, %c
    %f = add i32 %e, %a
    ret i32 %f
}
```

```
Printing analysis 'Scalar Evolution Analysis' for function 'example':
Classifying expressions for: @example
  %a = mul i32 %x, %x
  -->  (%x * %x) U: full-set S: full-set
  %b = add i32 %a, %y
  -->  ((%x * %x) + %y) U: full-set S: full-set
  %c = mul i32 %b, %b
  -->  (((%x * %x) + %y) * ((%x * %x) + %y)) U: full-set S: full-set
  %d = sub i32 %c, %b
  -->  (((%x * %x) + %y) * (-1 + (%x * %x) + %y)) U: full-set S: full-set
  %e = sub i32 %d, %c
  -->  (-1 * ((%x * %x) + %y)) U: full-set S: full-set
  %f = add i32 %e, %a
  -->  (-1 * %y) U: full-set S: full-set
Determining loop execution counts for: @example
Compiler returned: 0
```

```
opt -passes='print<scalar-evolution>'
```

https://godbolt.org/z/sx9GnMKKe

Формулы в Scalar Evolution

LLVM трактует одни и те же числа и как знаковые, и как беззнаковые. Соответственно, у них могут быть знаковые (S) и беззнаковые (U) диапазоны значений.

full-set означает диапазон, включающий все возможные значения.

```
Printing analysis 'Scalar Evolution Analysis' for function 'example':
Classifying expressions for: @example
  %a = mul i32 %x, %x
  -->  (%x * %x) U: full-set S: full-set
  %b = add i32 %a, %y
  -->  ((%x * %x) + %y) U: full-set S: full-set
  %c = mul i32 %b, %b
  -->  (((%x * %x) + %y) * ((%x * %x) + %y)) U: full-set S: full-set
  %d = sub i32 %c, %b
  -->  (((%x * %x) + %y) * (-1 + (%x * %x) + %y)) U: full-set S: full-set
  %e = sub i32 %d, %c
  -->  (-1 * ((%x * %x) + %y)) U: full-set S: full-set
  %f = add i32 %e, %a
  -->  (-1 * %y) U: full-set S: full-set
Determining loop execution counts for: @example
Compiler returned: 0
```

Расчёт диапазонов

```
define i32 @example(i32 %x, i32 %y) {
    %a = udiv i32 %x, 4
    %b = add i32 %a, 1000
    %c = add i32 %b, %a
    ret i32 %a
}
```

```
Printing analysis 'Scalar Evolution Analysis' for function 'example':
Classifying expressions for: @example
  %a = udiv i32 %x, 4
  -->  (%x /u 4) U: [0,1073741824) S: [0,1073741824)
  %b = add i32 %a, 1000
  -->  (1000 + (%x /u 4))<nuw><nsw> U: [1000,1073742824)
                                    S: [1000,1073742824)
  %c = add i32 %b, %a
  -->  (1000 + (2 * (%x /u 4))<nuw><nsw>)<nuw>
                                    U: [1000,-2147482649)
                                    S: [-2147483648,2147483647)
Determining loop execution counts for: @example
Compiler returned: 0
```

```
opt -passes='print<scalar-evolution>'
```

https://godbolt.org/z/rzhG87TqT

Расчёт диапазонов

Флаги nuw/nsw означают, что SCEV доказал, что при выполнении данной операции (сложение, умножение и т.п.) не произойдёт беззнакового/знакового переполнения.

`U: [1000,-2147482649)`
На самом деле это большая положительная константа (ЭТО ФИЧА). Нужно добавить 2^{32}.

```
Printing analysis 'Scalar Evolution Analysis' for function 'example':
Classifying expressions for: @example
  %a = udiv i32 %x, 4
  -->  (%x /u 4) U: [0,1073741824) S: [0,1073741824)
  %b = add i32 %a, 1000
  -->  (1000 + (%x /u 4))<nuw><nsw> U: [1000,1073742824)
                                    S: [1000,1073742824)
  %c = add i32 %b, %a
  -->  (1000 + (2 * (%x /u 4))<nuw><nsw>)<nuw>
                                    U: [1000,-2147482649)
                                    S: [-2147483648,2147483647)
Determining loop execution counts for: @example
Compiler returned: 0
```

Явное указание диапазонов

```
define i32 @example(ptr %array) {
  %length.ptr = getelementptr i32, ptr %array, i32 4
  %len = load i32, ptr %length.ptr, !range !0
  ret i32 %len
}

!0 = !{i32 0, i32 2147483647}
```

```
Printing analysis 'Scalar Evolution Analysis' for function 'example':
Classifying expressions for: @example
  %length.ptr = getelementptr i32, ptr %array, i32 4
  -->  (16 + %array) U: full-set S: full-set
  %len = load i32, ptr %length.ptr, align 4, !range !0
  -->  %len U: [0,2147483647) S: [0,2147483647)
Determining loop execution counts for: @example
Compiler returned: 0
```

```
opt -passes='print<scalar-evolution>'
```

https://godbolt.org/z/9nTfqnPn1

Количество итераций в цикле

```
void example(int start, int step, int end) {
    for (int i = start; i < end; i += step) {
        use(i);
    }
}
```

Если условие выхода из цикла зависит от индукционной переменной, то это позволяет относительно просто оценить количество его итераций.

Если не случится переполнений, то количество итераций в этом примере оценивается как $\left\lceil \frac{end - start}{step} \right\rceil$

Циклы с несколькими условиями выхода

```
bool should_exit(int iter);

void example(int start, int end, int n) {
    for (int i = start; i < end; i += 2) { // Можно оценить
        if (2 * i + 10 > n)                // Можно оценить
            break;
        if (should_exit(i))                // Нельзя оценить
            break;
    }
}
```

Если в цикле несколько условий выхода, то возможно, что для некоторых из них можно получить точную или верхнюю оценку числа итераций, после которых цикл обязан будет выйти.

Расчёт числа итераций цикла

```
; for (int i = 0; i < end; i++) {
;   if (i >= n) return -1;
;   if (i >= m) return -2;
; }
; return 0;
define i32 @example(ptr %p_n, ptr %p_m, i32 %end) {
entry:
  %n = load i32, ptr %p_n, !range !{i32 1, i32 100}
  %m = load i32, ptr %p_m, !range !{i32 20, i32 50}
  br label %loop
loop:
  %iv = phi i32 [0, %entry], [%iv.next, %backedge]
  %check_n = icmp ult i32 %iv, %n
  br i1 %check_n, label %checked_1, label %exit_1
checked_1:
  %check_m = icmp ult i32 %iv, %m
  br i1 %check_m, label %backedge, label %exit_2
backedge:
  %iv.next = add i32 %iv, 1
  %loop_cond = icmp ult i32 %iv.next, %end
  br i1 %loop_cond, label %loop, label %exit_0
...
}
```

```
opt -passes='print<scalar-evolution>'
```

https://godbolt.org/z/69Y5dfYGc

```
Loop %loop: <multiple exits> backedge-taken count is
    ((-1 + (1 umax %end)) umin %n umin %m)
  exit count for loop: %n
  exit count for checked_1: %m
  exit count for backedge: (-1 + (1 umax %end))
Loop %loop: constant max backedge-taken count is i32 49
```

Мы выйдем из цикла через блок loop, если придём туда на итерации номер n.

Мы выйдем из цикла через блок checked_1, если придём туда на итерации номер m.

Мы выйдем из цикла через блок backedge, если придём туда на итерации номер
(-1 + (1 umax %end)).

Всего в цикле будет не более 49 итераций.

Расчёт числа итераций цикла

```
; for (int i = 0; i < end; i++) {
;   if (i >= n) return -1;
;   if (foo(i)) return -2;
; }
; return 0;
define i32 @example(ptr %p_n, i32 %end) {
entry:
  %n = load i32, ptr %p_n, !range !{i32 1, i32 100}
  br label %loop
loop:
  %iv = phi i32 [0, %entry], [%iv.next, %backedge]
  %check_n = icmp ult i32 %iv, %n
  br i1 %check_n, label %checked_1, label %exit_1
checked_1:
  %check_foo = call i1 @foo(i32 %iv)
  br i1 %check_foo, label %backedge, label %exit_2
backedge:
  %iv.next = add i32 %iv, 1
  %loop_cond = icmp ult i32 %iv.next, %end
  br i1 %loop_cond, label %loop, label %done
...
}
```

```
Loop %loop: <multiple exits> Unpredictable backedge-taken count.
  exit count for loop: %n
  exit count for checked_1: ***COULDNOTCOMPUTE***
  exit count for backedge: (-1 + (1 umax %end))
Loop %loop: constant max backedge-taken count is i32 99
Loop %loop: symbolic max backedge-taken count is
    ((-1 + (1 umax %end)) umin %n)
  symbolic max exit count for loop: %n
  symbolic max exit count for checked_1: ***COULDNOTCOMPUTE***
  symbolic max exit count for backedge: (-1 + (1 umax %end))
```

```
opt -passes='print<scalar-evolution>'
```

https://godbolt.org/z/Wjqr6nKxM

Расчёт числа итераций цикла

```
; for (int i = 0; i < end; i++) {
;   if (i >= n) return -1;
;   if (foo(i)) return -2;
; }
; return 0;
define i32 @example(ptr %p_n, i32 %end) {
entry:
  %n = load i32, ptr %p_n, !range !{i32 1, i32 100}
  br label %loop
loop:
  %iv = phi i32 [0, %entry], [%iv.next, %backedge]
  %check_n = icmp ult i32 %iv, %n
  br i1 %check_n, label %checked_1, label %exit_1
checked_1:
  %check_foo = call i1 @foo(i32 %iv)
  br i1 %check_foo, label %backedge, label %exit_2
backedge:
  %iv.next = add i32 %iv, 1
  %loop_cond = icmp ult i32 %iv.next, %end
  br i1 %loop_cond, label %loop, label %done
...
}
```

```
Loop %loop: <multiple exits> Unpredictable backedge-taken count.
  exit count for loop: %n
  exit count for checked_1: ***COULDNOTCOMPUTE***
  exit count for backedge: (-1 + (1 umax %end))
Loop %loop: constant max backedge-taken count is i32 99
Loop %loop: symbolic max backedge-taken count is
    ((-1 + (1 umax %end)) umin %n)
  symbolic max exit count for loop: %n
  symbolic max exit count for checked_1: ***COULDNOTCOMPUTE***
  symbolic max exit count for backedge: (-1 + (1 umax %end))
```

Мы не смогли определить, на какой итерации мы выйдем через блок checked_1.

Из-за этого мы не знаем, сколько всего итераций в цикле.

```
opt -passes='print<scalar-evolution>'
```

https://godbolt.org/z/Wjqr6nKxM

Расчёт числа итераций цикла

```
; for (int i = 0; i < end; i++) {
;   if (i >= n) return -1;
;   if (foo(i)) return -2;
; }
; return 0;
define i32 @example(ptr %p_n, i32 %end) {
entry:
  %n = load i32, ptr %p_n, !range !{i32 1, i32 100}
  br label %loop
loop:
  %iv = phi i32 [0, %entry], [%iv.next, %backedge]
  %check_n = icmp ult i32 %iv, %n
  br i1 %check_n, label %checked_1, label %exit_1
checked_1:
  %check_foo = call i1 @foo(i32 %iv)
  br i1 %check_foo, label %backedge, label %exit_2
backedge:
  %iv.next = add i32 %iv, 1
  %loop_cond = icmp ult i32 %iv.next, %end
  br i1 %loop_cond, label %loop, label %done
...
}
```

```
Loop %loop: <multiple exits> Unpredictable backedge-taken count.
  exit count for loop: %n
  exit count for checked_1: ***COULDNOTCOMPUTE***
  exit count for backedge: (-1 + (1 umax %end))
Loop %loop: constant max backedge-taken count is i32 99
Loop %loop: symbolic max backedge-taken count is
    ((-1 + (1 umax %end)) umin %n)
  symbolic max exit count for loop: %n
  symbolic max exit count for checked_1: ***COULDNOTCOMPUTE***
  symbolic max exit count for backedge: (-1 + (1 umax %end))
```

Благодаря наличию оценок для каких-то из выходов, мы можем сказать, что мы выйдем из цикла **не позднее**, чем min(n, max(end, 1) – 1)).

```
opt -passes='print<scalar-evolution>'
```

https://godbolt.org/z/Wjqr6nKxM

Расчёт числа итераций цикла

```
; for (int i = 0; i < end; i++) {
;   if (i >= n) return -1;
;   if (foo(i)) return -2;
; }
; return 0;
define i32 @example(ptr %p_n, i32 %end) {
entry:
  %n = load i32, ptr %p_n, !range !{i32 1, i32 100}
  br label %loop
loop:
  %iv = phi i32 [0, %entry], [%iv.next, %backedge]
  %check_n = icmp ult i32 %iv, %n
  br i1 %check_n, label %checked_1, label %exit_1
checked_1:
  %check_foo = call i1 @foo(i32 %iv)
  br i1 %check_foo, label %backedge, label %exit_2
backedge:
  %iv.next = add i32 %iv, 1
  %loop_cond = icmp ult i32 %iv.next, %end
  br i1 %loop_cond, label %loop, label %done
...
}
```

```
Loop %loop: <multiple exits> Unpredictable backedge-taken count.
  exit count for loop: %n
  exit count for checked_1: ***COULDNOTCOMPUTE***
  exit count for backedge: (-1 + (1 umax %end))
Loop %loop: constant max backedge-taken count is i32 99
Loop %loop: symbolic max backedge-taken count is
    ((-1 + (1 umax %end)) umin %n)
  symbolic max exit count for loop: %n
  symbolic max exit count for checked_1: ***COULDNOTCOMPUTE***
  symbolic max exit count for backedge: (-1 + (1 umax %end))
```

В цикле будет не более 99 итераций.

```
opt -passes='print<scalar-evolution>'
```

https://godbolt.org/z/Wjqr6nKxM

Индукционные переменные

```
; for (int i = 0; i < end; i++) {
;   if (i >= n) return -1;
;   if (foo(i)) return -2;
; }
; return 0;
define i32 @example(ptr %p_n, i32 %end) {
entry:
  %n = load i32, ptr %p_n, !range !{i32 1, i32 100}
  br label %loop
loop:
  %iv = phi i32 [0, %entry], [%iv.next, %backedge]
  %check_n = icmp ult i32 %iv, %n
  br i1 %check_n, label %checked_1, label %exit_1
checked_1:
  %check_foo = call i1 @foo(i32 %iv)
  br i1 %check_foo, label %backedge, label %exit_2
backedge:
  %iv.next = add i32 %iv, 1
  %loop_cond = icmp ult i32 %iv.next, %end
  br i1 %loop_cond, label %loop, label %done
...
}
```

```
 %iv = phi i32 [ 0, %entry ], [ %iv.next, %backedge ]
  -->  {0,+,1}<nuw><nsw><%loop> U: [0,100) S: [0,100)
...
%iv.next = add i32 %iv, 1
  -->  {1,+,1}<nuw><nsw><%loop> U: [1,101) S: [1,101)

В общем виде: {start,+,step} IV(n) = start + step * n
```

```
opt -passes='print<scalar-evolution>'
```

https://godbolt.org/z/Wjqr6nKxM

Индукционные переменные

```
define i32 @example(ptr %p_n, i32 %end) {
entry:
  %n = load i32, ptr %p_n, !range !0
  br label %loop

loop:
  %iv = phi i32 [0, %entry], [%iv.next, %loop]
  %iv_1 = add i32 %iv, %n
  %iv_2 = mul i32 %iv_1, 20
  %iv_3 = add i32 %iv_2, %iv
  %iv.next = add i32 %iv, 1
  %loop_cond = icmp ult i32 %iv.next, %end
  br i1 %loop_cond, label %loop, label %done

done:
  ret i32 0
}

!0 = !{i32 1, i32 100}
```

```
%iv = phi i32 [ 0, %entry ], [ %iv.next, %loop ]
 -->  {0,+,1}<nuw><%loop>
      U: [0,-1) S: [0,-1)
      Exits: (-1 + (1 umax %end))
 %iv_1 = add i32 %iv, %n
 -->  {%n,+,1}<nw><%loop>
      U: full-set S: full-set
      Exits: (-1 + (1 umax %end) + %n)
 %iv_2 = mul i32 %iv_1, 20
 -->  {(20 * %n)<nuw><nsw>,+,20}<%loop>
      U: [0,-3) S: [-2147483648,2147483645)
      Exits: (-20 + (20 * (1 umax %end)) + (20 * %n)<nuw><nsw>)
 %iv_3 = add i32 %iv_2, %iv
 -->  {(20 * %n)<nuw><nsw>,+,21}<%loop>
      U: full-set S: full-set
      Exits: (-21 + (20 * %n)<nuw><nsw> + (21 * (1 umax %end)))
```

```
opt -passes='print<scalar-evolution>'
```

https://godbolt.org/z/je7z5T11r

Оптимизации

удаляем проверки диапазонов

Простой случай

Некоторые проверки диапазонов не падают никогда, и это можно доказать во время компиляции. В этом случае компилятор попытается просто удалить эту проверку.

Пример 1: бег до длины массива

```
void example(int[] array) {
    for (int i = 0; i < array.length; i++) {
        array[i] = 10;
    }
}
```

Пример 1: бег до длины массива

```
declare void @throw_oob_exception(i32 %index, i32 %len) noreturn

define void @example(ptr %data.ptr, ptr %len.ptr) {
entry:
  %len = load i32, ptr %len.ptr, !range !0
  %entry_cond = icmp slt i32 0, %len
  br i1 %entry_cond, label %loop, label %done

loop:
  %iv = phi i32 [0, %entry], [%iv.next, %latch]
  %rc = icmp ult i32 %iv, %len
  br i1 %rc, label %latch, label %throw_oob

latch:
  %el.ptr = getelementptr i32, ptr %data.ptr, i32 %iv
  store i32 10, ptr %el.ptr
  %iv.next = add i32 %iv, 1
  %loop_cond = icmp slt i32 %iv.next, %len
  br i1 %loop_cond, label %loop, label %done

done:
  ret void

throw_oob:
  call void @throw_oob_exception(i32 %iv, i32 %len)
  unreachable
}

!0 = !{i32 0, i32 2147483647}
```

```
Determining loop execution counts for: @example
Loop %loop: <multiple exits> backedge-taken count is
  ((-1 + %len)<nsw> umin %len)
  exit count for loop: %len
  exit count for latch: (-1 + %len)<nsw>
Loop %loop: constant max backedge-taken count is i32 2147483645
Loop %loop: symbolic max backedge-taken count is ((-1 + %len)<nsw>
umin %len)
  symbolic max exit count for loop: %len
  symbolic max exit count for latch: (-1 + %len)<nsw>
Loop %loop: Trip multiple is 1
Compiler returned: 0
```

Проверка никогда не упадёт, если цикл сделает **строго меньше** итераций, чем нужно для выхода по проверке.

```
opt -passes='print<scalar-evolution>'
```

https://godbolt.org/z/rMoc1ExGT

Пример 1: бег до длины массива

```
declare void @throw_oob_exception(i32 %index, i32 %len) noreturn

define void @example(ptr %data.ptr, ptr %len.ptr) {
entry:
  %len = load i32, ptr %len.ptr, !range !0
  %entry_cond = icmp slt i32 0, %len
  br i1 %entry_cond, label %loop, label %done

loop:
  %iv = phi i32 [0, %entry], [%iv.next, %latch]
  %rc = icmp ult i32 %iv, %len
  br i1 %rc, label %latch, label %throw_oob

latch:
  %el.ptr = getelementptr i32, ptr %data.ptr, i32 %iv
  store i32 10, ptr %el.ptr
  %iv.next = add i32 %iv, 1
  %loop_cond = icmp slt i32 %iv.next, %len
  br i1 %loop_cond, label %loop, label %done

done:
  ret void

throw_oob:
  call void @throw_oob_exception(i32 %iv, i32 %len)
  unreachable
}

!0 = !{i32 0, i32 2147483647}
```

opt -passes='print<scalar-evolution>'

https://godbolt.org/z/rMoc1ExGT

```
Determining loop execution counts for: @example
Loop %loop: <multiple exits> backedge-taken count is
  ((-1 + %len)<nsw> umin %len)
  exit count for loop: %len
  exit count for latch: (-1 + %len)<nsw>
Loop %loop: constant max backedge-taken count is i32 2147483645
Loop %loop: symbolic max backedge-taken count is ((-1 + %len)<nsw>
umin %len)
  symbolic max exit count for loop: %len
  symbolic max exit count for latch: (-1 + %len)<nsw>
Loop %loop: Trip multiple is 1
Compiler returned: 0
```

((-1 + %len)<nsw> umin %len) <u %len?

((-1 + %len)<nsw> <u %len OR %len <u %len

((-1 + %len)<nsw> <u %len

%len != 0

Вывод предикатов

Scalar Evolution умеет анализировать вышестоящие (доминирующие) условия и искать среди них нужные факты. Если нужно доказать неравенство A < B, он пытается свести его к более простым неравенствам и вывести их из условий вышестоящих проверок.

Также нужные факты могут выводиться напрямую из диапазонов значений (если их сравнивают с константами).

Пример 1: бег до длины массива

```
declare void @throw_oob_exception(i32 %index, i32 %len) noreturn

define void @example(ptr %data.ptr, ptr %len.ptr) {
entry:
  %len = load i32, ptr %len.ptr, !range !0
  %entry_cond = icmp slt i32 0, %len
  br i1 %entry_cond, label %loop, label %done

loop:
  %iv = phi i32 [0, %entry], [%iv.next, %latch]
  %rc = icmp ult i32 %iv, %len
  br i1 %rc, label %latch, label %throw_oob

latch:
  %el.ptr = getelementptr i32, ptr %data.ptr, i32 %iv
  store i32 10, ptr %el.ptr
  %iv.next = add i32 %iv, 1
  %loop_cond = icmp slt i32 %iv.next, %len
  br i1 %loop_cond, label %loop, label %done

done:
  ret void

throw_oob:
  call void @throw_oob_exception(i32 %iv, i32 %len)
  unreachable
}

!0 = !{i32 0, i32 2147483647}
```

```
opt -passes='print<scalar-evolution>'
```

https://godbolt.org/z/rMoc1ExGT

```
Determining loop execution counts for: @example
Loop %loop: <multiple exits> backedge-taken count is
  ((-1 + %len)<nsw> umin %len)
  exit count for loop: %len
  exit count for latch: (-1 + %len)<nsw>
Loop %loop: constant max backedge-taken count is i32 2147483645
Loop %loop: symbolic max backedge-taken count is ((-1 + %len)<nsw>
umin %len)
  symbolic max exit count for loop: %len
  symbolic max exit count for latch: (-1 + %len)<nsw>
Loop %loop: Trip multiple is 1
Compiler returned: 0
```

`0 <s %len -> %len != 0`

Если найдено вышестоящее условие, из которого следует то, что мы ищем, то утверждение считается доказанным.

Пример 1: бег до длины массива

```
declare void @throw_oob_exception(i32, i32) #0

define void @example(ptr %data.ptr, ptr %len.ptr) {
  %len = load i32, ptr %len.ptr, align 4, !range !0
  %entry_cond = icmp slt i32 0, %len
  br i1 %entry_cond, label %loop.preheader, label %done
loop.preheader:
  br label %loop
loop:
  %iv = phi i32 [ %iv.next, %latch ], [ 0, %loop.preheader ]
  br i1 true, label %latch, label %throw_oob
latch:
  %el.ptr = getelementptr i32, ptr %data.ptr, i32 %iv
  store i32 10, ptr %el.ptr, align 4
  %iv.next = add nuw nsw i32 %iv, 1
  %loop_cond = icmp ult i32 %iv.next, %len
  br i1 %loop_cond, label %loop, label %done.loopexit
done.loopexit:
  br label %done
done:
  ret void
throw_oob:  %iv.lcssa = phi i32 [ 0, %loop ]
  call void @throw_oob_exception(i32 %iv.lcssa, i32 %len)
  unreachable
}
attributes #0 = { noreturn }
!0 = !{i32 0, i32 2147483647}
```

В результате можно просто удалить проверку, заменив её на true.

```
opt -passes='indvars'
```

https://godbolt.org/z/Yov33bzhv

Пример 2: бег до нуля с неизвестного старта

```
void example(int[] array, int start) {
    for (int i = start; i >= 0; i--) {
        array[i] = 10;
    }
}
```

Проверка может либо упасть на первой итерации, либо никогда.

Пример 2: бег до нуля с неизвестного старта

```
declare void @throw_oob_exception(i32 %index, i32 %len) noreturn

define void @example(ptr %data.ptr, ptr %len.ptr, i32 %start) {
entry:
  %len = load i32, ptr %len.ptr, !range !0
  %entry_cond = icmp sge i32 %start, 0
  br i1 %entry_cond, label %loop, label %done

loop:
  %iv = phi i32 [%start, %entry], [%iv.next, %latch]
  %rc = icmp ult i32 %iv, %len
  br i1 %rc, label %latch, label %throw_oob

latch:
  %el.ptr = getelementptr i32, ptr %data.ptr, i32 %iv
  store i32 10, ptr %el.ptr
  %iv.next = add i32 %iv, -1
  %loop_cond = icmp sge i32 %iv.next, 0
  br i1 %loop_cond, label %loop, label %done

done:
  ret void

throw_oob:
  call void @throw_oob_exception(i32 %iv, i32 %len)
  unreachable
}

!0 = !{i32 0, i32 2147483647}
```

`opt -passes='print<scalar-evolution>'`

https://godbolt.org/z/E5GrfT3fr

```
Classifying expressions for: @example
  %len = load i32, ptr %len.ptr, align 4, !range !0
  -->  %len U: [0,2147483647) S: [0,2147483647)
  %iv = phi i32 [ %start, %entry ], [ %iv.next, %latch ]
  -->  {%start,+,-1}<%loop>
  %el.ptr = getelementptr i32, ptr %data.ptr, i32 %iv
  -->  {((4 * (sext i32 %start to i64))<nsw> + %data.ptr),+,-4}<nw><%loop>
  %iv.next = add i32 %iv, -1
  -->  {(-1 + %start),+,-1}<%loop>
Determining loop execution counts for: @example
…
Loop %loop: symbolic max backedge-taken count is %start
  symbolic max exit count for loop: ***COULDNOTCOMPUTE***
  symbolic max exit count for latch: %start
```

Мы не можем доказать **{%start,+,-1} <u %len**.

Можем ли мы доказать **{%start,+,-1} <u %len**, начиная со второй итерации?

Мы на второй итерации знаем, что **%start <u %len**.

Если докажем, что **{%start,+,-1} <u %start**, начиная со второй итерации, то задача решена.

Пример 2: бег до нуля с неизвестного старта

```
declare void @throw_oob_exception(i32 %index, i32 %len) noreturn

define void @example(ptr %data.ptr, ptr %len.ptr, i32 %start) {
entry:
  %len = load i32, ptr %len.ptr, !range !0
  %entry_cond = icmp sge i32 %start, 0
  br i1 %entry_cond, label %loop, label %done

loop:
  %iv = phi i32 [%start, %entry], [%iv.next, %latch]
  %rc = icmp ult i32 %iv, %len
  br i1 %rc, label %latch, label %throw_oob

latch:
  %el.ptr = getelementptr i32, ptr %data.ptr, i32 %iv
  store i32 10, ptr %el.ptr
  %iv.next = add i32 %iv, -1
  %loop_cond = icmp sge i32 %iv.next, 0
  br i1 %loop_cond, label %loop, label %done

done:
  ret void

throw_oob:
  call void @throw_oob_exception(i32 %iv, i32 %len)
  unreachable
}

!0 = !{i32 0, i32 2147483647}
```

opt -passes='print<scalar-evolution>'

https://godbolt.org/z/E5GrfT3fr

```
Classifying expressions for: @example
  %len = load i32, ptr %len.ptr, align 4, !range !0
  -->  %len U: [0,2147483647) S: [0,2147483647)
  %iv = phi i32 [ %start, %entry ], [ %iv.next, %latch ]
  -->  {%start,+,-1}<%loop>
  %el.ptr = getelementptr i32, ptr %data.ptr, i32 %iv
  -->  {((4 * (sext i32 %start to i64))<nsw> + %data.ptr),+,-4}<nw><%loop>
  %iv.next = add i32 %iv, -1
  -->  {(-1 + %start),+,-1}<%loop>
Determining loop execution counts for: @example
…
Loop %loop: symbolic max backedge-taken count is %start
  symbolic max exit count for loop: ***COULDNOTCOMPUTE***
  symbolic max exit count for latch: %start
```

{%start,+,-1} <u %start, если шаг отрицательный и не происходит переполнений ниже нуля.

Из symbolic max backedge-taken count знаем, что {%start,+,-1} не опускается ниже нуля (т.е. не становится огромным положительным числом).

Следовательно, начиная со второй итерации проверка не падает.

Пример 2: бег до нуля с неизвестного старта

```
define void @example(ptr %data.ptr, ptr %len.ptr, i32 %start) {
  %len = load i32, ptr %len.ptr, align 4, !range !0
  %entry_cond = icmp sge i32 %start, 0
  br i1 %entry_cond, label %loop.preheader, label %done

loop.preheader:
  %rc.first_iter = icmp ult i32 %start, %len
  br label %loop

loop:
  %iv = phi i32 [ %iv.next, %latch ], [ %start, %loop.preheader ]
  br i1 %rc.first_iter, label %latch, label %throw_oob

latch:
  %el.ptr = getelementptr i32, ptr %data.ptr, i32 %iv
  store i32 10, ptr %el.ptr, align 4
  %iv.next = add nsw i32 %iv, -1
  %loop_cond = icmp sge i32 %iv.next, 0
  br i1 %loop_cond, label %loop, label %done.loopexit
done.loopexit:
  br label %done
done:
  ret void
throw_oob:
  %iv.lcssa = phi i32 [ %start, %loop ]
  call void @throw_oob_exception(i32 %iv.lcssa, i32 %len)
  unreachable
}
```

opt -passes='indvars'

https://godbolt.org/z/EM47GGb1K

Пример 2: бег до нуля с неизвестного старта

```
define void @example(ptr %data.ptr, ptr %len.ptr, i32 %start) {
  %len = load i32, ptr %len.ptr, align 4, !range !0
  %entry_cond = icmp sge i32 %start, 0
  br i1 %entry_cond, label %loop.preheader, label %done

loop.preheader:                                   ; preds = %entry
  %rc.first_iter = icmp ult i32 %start, %len
  br i1 %rc.first_iter, label %loop, label %throw_oob

loop:                                             ; preds = %loop.preheader, %loop
  %iv = phi i32 [ %iv.next, %loop ], [ %start, %loop.preheader ]
  %el.ptr = getelementptr i32, ptr %data.ptr, i32 %iv
  store i32 10, ptr %el.ptr, align 4
  %iv.next = add nsw i32 %iv, -1
  %loop_cond = icmp sge i32 %iv.next, 0
  br i1 %loop_cond, label %loop, label %done

done:                                             ; preds = %loop, %entry
  ret void

throw_oob:                                        ; preds = %loop.preheader
  %iv.lcssa = phi i32 [ %start, %loop.preheader ]
  call void @throw_oob_exception(i32 %iv.lcssa, i32 %len)
  unreachable
}
```

```
opt -passes='loop(indvars,simple-loop-unswitch),simplifycfg‘
```

https://godbolt.org/z/EscfdTWqT

Пример 3: может упасть в конце

```
void example(int[] arr1, int[] arr2, int start, int end) {
    for (int i = start; i < end; i++) {
        arr1[i + 3] = arr2[i - 2];
    }
}
```

Первые несколько итераций могут пройти, но потом проверка может упасть (или не упасть).

Пример 3: может упасть в конце

```
void example(int[] arr1, int[] arr2, int start, int end) {
    for (int i = start; i < end; i++) {
        arr1[i + 3] = arr2[i - 2];
    }
}
```

Построим неравенства, описывающие условия прохождения проверок диапазонов:

$$0 \leq i + 3$$
$$i + 3 < arr1.length$$
$$0 \leq i - 2$$
$$i - 2 < arr2.length$$

Пример 3: может упасть в конце

```
void example(int[] arr1, int[] arr2, int start, int end) {
    for (int i = start; i < end; i++) {
        arr1[i + 3] = arr2[i - 2];
    }
}
```

Оставим индукционную переменную в одной стороне неравенства, всё остальное – в другой.

$$-3 \leq i$$
$$i < arr1.length - 3$$
$$2 \leq i$$
$$i < arr2.length + 2$$

Пример 3: может упасть в конце

```
void example(int[] arr1, int[] arr2, int start, int end) {
    for (int i = start; i < end; i++) {
        arr1[i + 3] = arr2[i - 2];
    }
}
```

Пересечём все эти области и получим безопасный диапазон

$$\max(-3, 2) \leq i$$
$$i < \min(arr1.length - 3, arr2.length + 2)$$

Пример 3: может упасть в конце

```
void example(int[] arr1, int[] arr2, int start, int end) {
    int safe_start = max(-3, 2);
    int safe_end = min(arr1.length - 3, arr2.length + 2);
    int i;
    for (i = start; i < safe_start; i++) {
        arr1[i + 3] = arr2[i - 2];
    }
    for (i = safe_start; i < safe_end; i++) {
        // Тут проверки диапазонов не генерируются
        arr1[i + 3] = arr2[i - 2];
    }
    for (int i = safe_end; i < end; i++) {
        arr1[i + 3] = arr2[i - 2];
    }
}
```

Пасс: irce (Inductive Range Check Elimination)

Пример 4: есть возможность деоптимизации

```
void example(int[] arr, int start, int end) {
    for (int i = start; i < end; i++) {
        arr[i] = 10;
    }
}
```

Деоптимизация – возможность выйти из скомпилированного кода и продолжить исполнение в безопасной (потенциально – более медленной) среде. Например, Java (и подобные языки) может в любой момент уйти в интерпретатор.

Пример 4: есть возможность деоптимизации

```
void example(int[] arr, int start, int end) {
    for (int i = start; i < end; i++) {
        arr[i] = 10;
    }
}
```

Идея: если не упадёт ни на первой, ни на последней итерации, и на пути нет переполнений, то достаточно проверить только краевые условия.

Пример 4: есть возможность деоптимизации

```
void example(int[] arr, int start, int end) {
    if (start < 0 || start >= arr.length || end < 0 || end >= arr.length) {
        // Идём считать в интерпретаторе
        deoptimize();
    }
    for (int i = start; i < end; i++) {
        // Проверки не нужны
        arr[i] = 10;
    }
}
```

Мы можем уйти в интерпретатор очень рано, но считается, что это будет происходить редко (или вообще не будет происходить).

Название оптимизации: Loop Predication

Заключение

Мы узнали

- Что такое LLVM IR
- Что такое Scalar Evolution
- Что можно делать при помощи Scalar Evolution
- Как LLVM борется с проверками диапазонов

Контакты

- Мой Telegram: @xortator
- Хабр: https://habr.com/ru/users/xortator/

Спасибо за внимание!