Eliminate Tail Calls

01

Выполнили студенты гр. ИВ-722: Булатов, Булах, Конфедератова, Шефер

Хвостовая рекурсия

частный случай рекурсии, при котором рекурсивный вызов единственен и является последней инструкцией метода. При этом метод, вызвавший “сам себя”, должен немедленно возвратить результат рекурсивного вызова, не производя над ним предварительно никаких преобразований. Разумеется, последнее требование относится только к методам, возвращающим значение, отличное от void

02

рекурсивный поиск суммы от 1 до n

```
int recSum(int n) {
    if (n == 0)
        return 0;
    return n + recSum(n - 1);
}
```

тот же алгоритм с применением хвостовой рекурсии

```
int tailRecSum(int n, int sum) {
    if (n == 0)
        return sum;
    return tailRecSum (n - 1, sum + n);
}
```

03

Задачи оптимизации

Tail Call Elimination преобразует рекурсивные вызовы текущей функции, за которыми следует инструкция возврата с ветвлением, создавая цикл. Этот проход также реализует расширения базового алгоритма:

- Тривиальные инструкции между вызовом и возвратом не препятствуют выполнению преобразования

- Преобразует функции, которые являются хвостовыми рекурсивными, для использования переменной-накопителя.

- TRE выполняется, если функция возвращает результат, возвращенный вызовом, или если функция возвращает константу времени выполнения при всех выходах из функции.

- Если проход может доказать, что вызываемые объекты не имеют доступа к своему кадру стека вызывающего абонента, они помечаются как подходящие для исключения хвостового вызова (генератором кода).

Проход оптимизации

FunctionPass *llvm::createTailCallEliminationPass() - открытый интерфейс к проходу **Tail Call Elimination,** который возвращает **new TailCallElim()**

```
struct TailCallElim : public FunctionPass {
    TailCallElim() : FunctionPass(ID);
    void getAnalysisUsage(AnalysisUsage &AU);
    bool runOnFunction(Function &F);
}
```

05

```
#include "llvm/ADT/Statistic.h"

STATISTIC(NumEliminated, "Number of tail calls removed");
STATISTIC(NumRetDuped,   "Number of return duplicated");
STATISTIC(NumAccumAdded, "Number of accumulators introduced");
```

```
===-------------------------------------------------------------------------===
                          ... Statistics Collected ...
===-------------------------------------------------------------------------===

1 tailcallelim - Number of tail calls removed
1 tailcallelim - Number of return duplicated

===-------------------------------------------------------------------------===
```

06

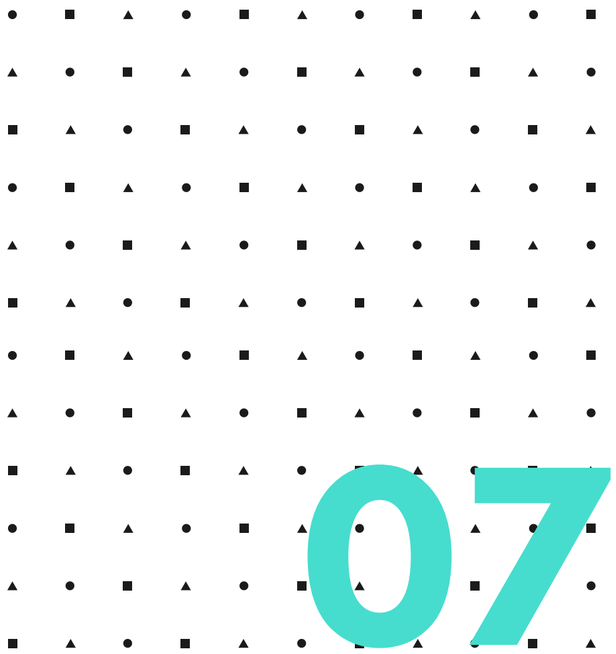
Проход оптимизации можно логически разбить на две взаимодополняющие части: Анализ и Преобразование

Анализ

осуществляет проход по коду функции, собирает информацию о созданных переменных, помечает вызовы функций для дальнейшего исключения

Преобразование

используя информацию, полученную в результате анализа, удаляет помеченные вызовы функций, создает тело цикла (базовый блок) в конце которого описывается условие перехода (phi)



Анализ

- **static bool canTRE(Function &F)** - сканирует указанную функцию на предмет инструкций alloca. Если он содержит какие-либо динамические аллоки, возвращает **false**

- **void walk(Value *Root)** - Начинает поиск с корневого значения, чтобы отметить вызовы, которые используют значение или производное значение в **AllocaUsers**, а также места, откуда оно может уйти в **EscapePoints**.

```
struct AllocaDerivedValueTracker {
  void walk(Value *Root) {
    SmallVector<Use *, 32> Worklist;
    SmallPtrSet<Use *, 32> Visited;

    auto AddUsesToWorklist = [&](Value
*V) {
      for (auto &U : V->uses()) {
        if (!Visited.insert(&U).second)
          continue;
        Worklist.push_back(&U);
      }
  };
...
```

09

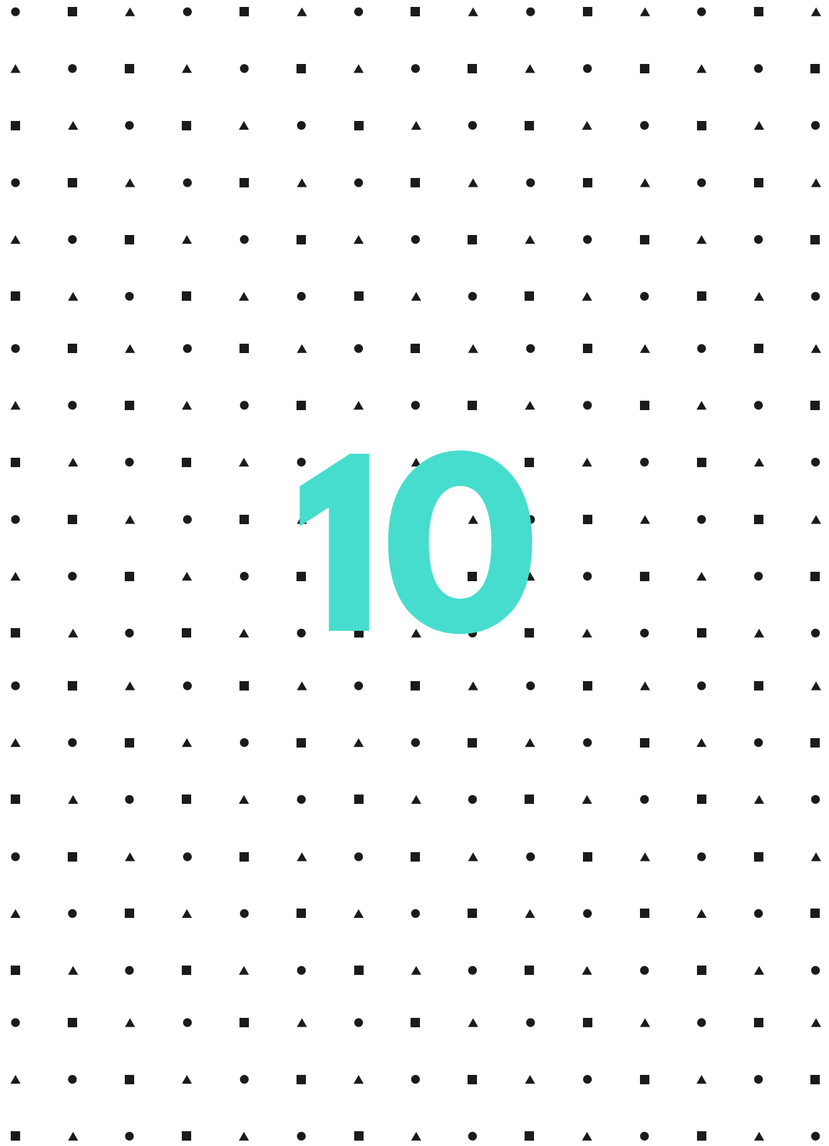
- **void callUsesLocalStack(CallBase &CB, bool IsNocapture)** — функция добавляет значения в два списка:
- **AllocaUsers** - вызовы alloc пользователя
- **EscapePoints** - такие call, которые могут записывать что-то в память на стеке, исключение их из CallBase может привезти к утечке значения

```
AddUsesToWorklist(Root);

while (!Worklist.empty()) {
  Use *U = Worklist.pop_back_val();
  Instruction *I = cast<Instruction>(U->getUser());

  switch (I->getOpcode()) {
    case Instruction::Call:
    case Instruction::Invoke: {
      auto &CB = cast<CallBase>(*I);
      if (CB.isArgOperand(U) &&
CB.isByValArgument(CB.getArgOperandNo(U)))
        continue;
        bool IsNocapture = CB.isDataOperand(U) &&
CB.doesNotCapture(CB.getDataOperandNo(U));
        callUsesLocalStack(CB, IsNocapture);
        if (IsNocapture) {
          continue;
        }
        break;
      }
...
```

static bool markTails(Function &F, ...) - помечает **tail call**, как кандидата для последующего удаления

static bool canMoveAboveCall(Instruction *I, CallInst *CI, ...) - Вернет истину, если можно безопасно переместить указанную инструкцию после вызова в перед вызовом

static bool canTransformAccumulatorRecursion(Instruction *I, ..) - проверяет возможность преобразование рекурсии с использованием переменной аккумулятора



Преобразование

TailRecursionEliminator - класс отвечает за смену всех хвостовых рекурсивных вызовов на циклы и включает след.функции:

- **CallInst *TailRecursionEliminator::findTRECandidate(BasicBlock *BB, ..)** - поиск в базовом блоке вызов call, который будет подходить для оптимизации
- **bool TailRecursionEliminator::processBlock(BasicBlock &BB, bool CannotTailCallElimCallsMarkedTail)** — создает базовый блок, который является телом нового цикла, который получился в результате устранения хвостовой рекурсии
- **void TailRecursionEliminator::insertAccumulator(Instruction *AccRecInstr)** — инициализация и вставка переменной аккумулятора, в которой будет храниться результат функции
- **bool TailRecursionEliminator::eliminateCall(CallInst *CI)** — удаление всех хвостовых вызовов из IR представления

Примеры оптимизации

программы, реализующей алгоритм вычисления факториала

Получение читаемой формы IR без оптимизации (.ll файл)

$ clang-10 -S -O3 -emit-llvm -mllvm -disable-llvm-optzns factorial.c -o no_opt_factorial.ll

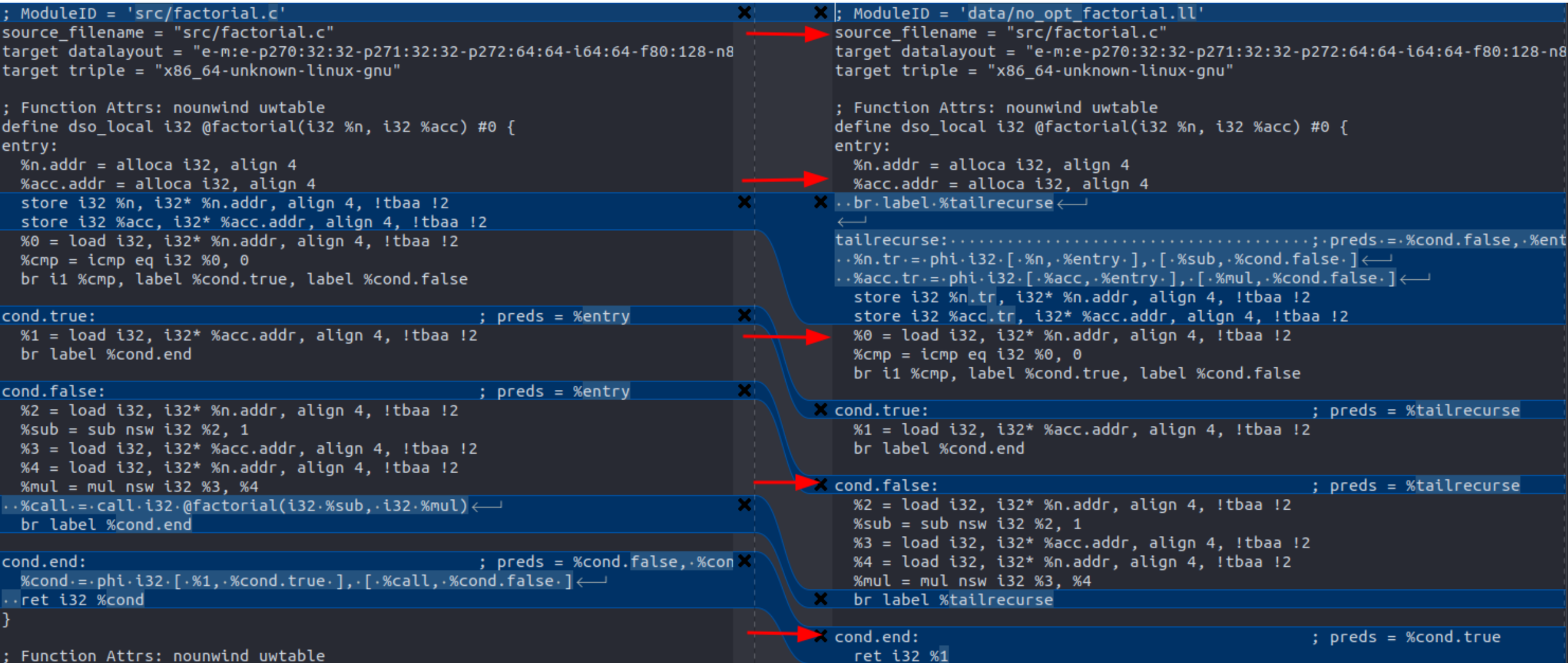
Применение Tail Call Elimination

$ opt --tailcallelim -S no_opt_factorial.ll -o opt_factorial.ll -time-passes -stats

Factorial.c (вычисление факториала)

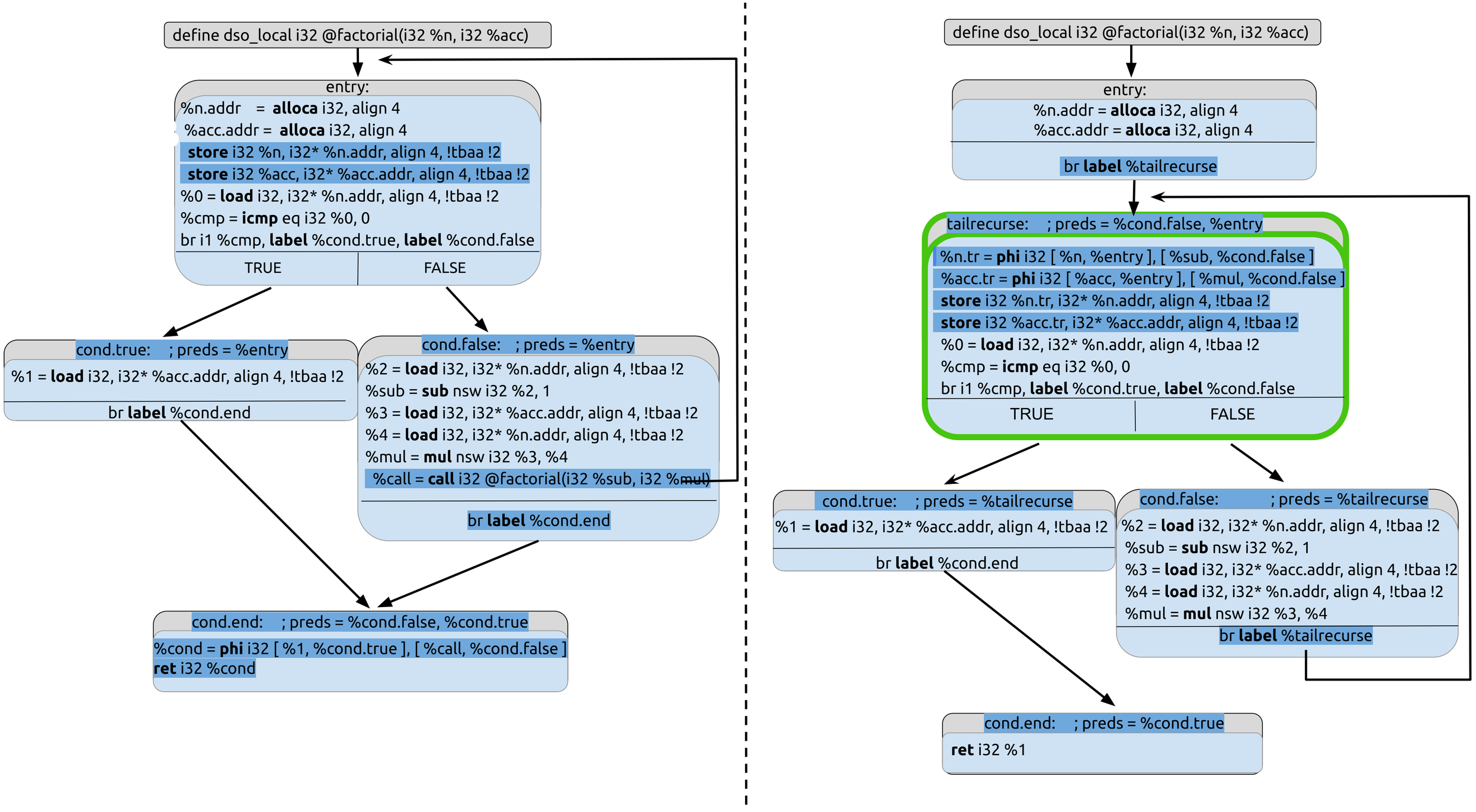
```
int factorial (int n, int acc) {
    return (n==0) ? acc : factorial(n - 1, acc * n);
}

int main () {
   factorial(10, 1);
   return 0;
}
```

Слева IR представление без оптимизации, справа - оптимизированное



```
; ModuleID = 'src/factorial.c'
source_filename = "src/factorial.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8
target triple = "x86_64-unknown-linux-gnu"

; Function Attrs: nounwind uwtable
define dso_local i32 @factorial(i32 %n, i32 %acc) #0 {
entry:
  %n.addr = alloca i32, align 4
  %acc.addr = alloca i32, align 4
  store i32 %n, i32* %n.addr, align 4, !tbaa !2
  store i32 %acc, i32* %acc.addr, align 4, !tbaa !2
  %0 = load i32, i32* %n.addr, align 4, !tbaa !2
  %cmp = icmp eq i32 %0, 0
  br i1 %cmp, label %cond.true, label %cond.false

cond.true:                                        ; preds = %entry
  %1 = load i32, i32* %acc.addr, align 4, !tbaa !2
  br label %cond.end

cond.false:                                       ; preds = %entry
  %2 = load i32, i32* %n.addr, align 4, !tbaa !2
  %sub = sub nsw i32 %2, 1
  %3 = load i32, i32* %acc.addr, align 4, !tbaa !2
  %4 = load i32, i32* %n.addr, align 4, !tbaa !2
  %mul = mul nsw i32 %3, %4
  %call = call i32 @factorial(i32 %sub, i32 %mul)
  br label %cond.end

cond.end:                                         ; preds = %cond.false, %con
  %cond = phi i32 [ %1, %cond.true ], [ %call, %cond.false ]
  ret i32 %cond
}

; Function Attrs: nounwind uwtable
```

```
; ModuleID = 'data/no_opt_factorial.ll'
source_filename = "src/factorial.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8
target triple = "x86_64-unknown-linux-gnu"

; Function Attrs: nounwind uwtable
define dso_local i32 @factorial(i32 %n, i32 %acc) #0 {
entry:
  %n.addr = alloca i32, align 4
  %acc.addr = alloca i32, align 4
  br label %tailrecurse

tailrecurse:                                      ; preds = %cond.false, %ent
  %n.tr = phi i32 [ %n, %entry ], [ %sub, %cond.false ]
  %acc.tr = phi i32 [ %acc, %entry ], [ %mul, %cond.false ]
  store i32 %n.tr, i32* %n.addr, align 4, !tbaa !2
  store i32 %acc.tr, i32* %acc.addr, align 4, !tbaa !2
  %0 = load i32, i32* %n.addr, align 4, !tbaa !2
  %cmp = icmp eq i32 %0, 0
  br i1 %cmp, label %cond.true, label %cond.false

cond.true:                                        ; preds = %tailrecurse
  %1 = load i32, i32* %acc.addr, align 4, !tbaa !2
  br label %cond.end

cond.false:                                       ; preds = %tailrecurse
  %2 = load i32, i32* %n.addr, align 4, !tbaa !2
  %sub = sub nsw i32 %2, 1
  %3 = load i32, i32* %acc.addr, align 4, !tbaa !2
  %4 = load i32, i32* %n.addr, align 4, !tbaa !2
  %mul = mul nsw i32 %3, %4
  br label %tailrecurse

cond.end:                                         ; preds = %cond.true
  ret i32 %1
```


define dso_local i32 @factorial(i32 %n, i32 %acc)
entry:
%n.addr = alloca i32, align 4
%acc.addr = alloca i32, align 4
store i32 %n, i32* %n.addr, align 4, !tbaa !2
store i32 %acc, i32* %acc.addr, align 4, !tbaa !2
%0 = load i32, i32* %n.addr, align 4, !tbaa !2
%cmp = icmp eq i32 %0, 0
br i1 %cmp, label %cond.true, label %cond.false
TRUE
FALSE
cond.true: ; preds = %entry
%1 = load i32, i32* %acc.addr, align 4, !tbaa !2
br label %cond.end
cond.false: ; preds = %entry
%2 = load i32, i32* %n.addr, align 4, !tbaa !2
%sub = sub nsw i32 %2, 1
%3 = load i32, i32* %acc.addr, align 4, !tbaa !2
%4 = load i32, i32* %n.addr, align 4, !tbaa !2
%mul = mul nsw i32 %3, %4
%call = call i32 @factorial(i32 %sub, i32 %mul)
br label %cond.end
cond.end: ; preds = %cond.false, %cond.true
%cond = phi i32 [%1, %cond.true], [%call, %cond.false]
ret i32 %cond
define dso_local i32 @factorial(i32 %n, i32 %acc)
entry:
%n.addr = alloca i32, align 4
%acc.addr = alloca i32, align 4
br label %tailrecurse
tailrecurse: ; preds = %cond.false, %entry
%n.tr = phi i32 [%n, %entry], [%sub, %cond.false]
%acc.tr = phi i32 [%acc, %entry], [%mul, %cond.false]
store i32 %n.tr, i32* %n.addr, align 4, !tbaa !2
store i32 %acc.tr, i32* %acc.addr, align 4, !tbaa !2
%0 = load i32, i32* %n.addr, align 4, !tbaa !2
%cmp = icmp eq i32 %0, 0
br i1 %cmp, label %cond.true, label %cond.false
TRUE
FALSE
cond.true: ; preds = %tailrecurse
%1 = load i32, i32* %acc.addr, align 4, !tbaa !2
br label %cond.end
cond.false: ; preds = %tailrecurse
%2 = load i32, i32* %n.addr, align 4, !tbaa !2
%sub = sub nsw i32 %2, 1
%3 = load i32, i32* %acc.addr, align 4, !tbaa !2
%4 = load i32, i32* %n.addr, align 4, !tbaa !2
%mul = mul nsw i32 %3, %4
br label %tailrecurse
cond.end: ; preds = %cond.true
ret i32 %1

Пример неудачного применения оптимизации

альтернативный вариант реализации алгоритма вычисления факториала

Вариант с использованием тернарного оператора

```
int factorial (int n, int acc) {
    return (n==0) ? acc : factorial(n - 1, acc * n);
}
```

Вариант с использованием оператора if

```
int factorial (int n, int acc) {
    if (n == 0)
        return acc;
    return factorial(n - 1, acc * n);
}
```

Слева IR представление без оптимизации, справа - оптимизированное

```
define dso_local i32 @factorial(i32 %n, i32 %acc) #0
entry:
  %retval = alloca i32, align 4
  %n.addr = alloca i32, align 4
  %acc.addr = alloca i32, align 4
  store i32 %n, i32* %n.addr, align 4, !tbaa !2
  store i32 %acc, i32* %acc.addr, align 4, !tbaa !2
  %0 = load i32, i32* %n.addr, align 4, !tbaa !2
  %cmp = icmp eq i32 %0, 0
  br i1 %cmp, label %if.then, label %if.end

if.then:                                          ;
  %1 = load i32, i32* %acc.addr, align 4, !tbaa !2
  store i32 %1, i32* %retval, align 4
  br label %return

if.end:                                           ;
  %2 = load i32, i32* %n.addr, align 4, !tbaa !2
  %sub = sub nsw i32 %2, 1
  %3 = load i32, i32* %acc.addr, align 4, !tbaa !2
  %4 = load i32, i32* %n.addr, align 4, !tbaa !2
  %mul = mul nsw i32 %3, %4
  %call = call i32 @factorial(i32 %sub, i32 %mul)
  store i32 %call, i32* %retval, align 4
  br label %return

return:                                           ;
  %5 = load i32, i32* %retval, align 4
  ret i32 %5
}
```

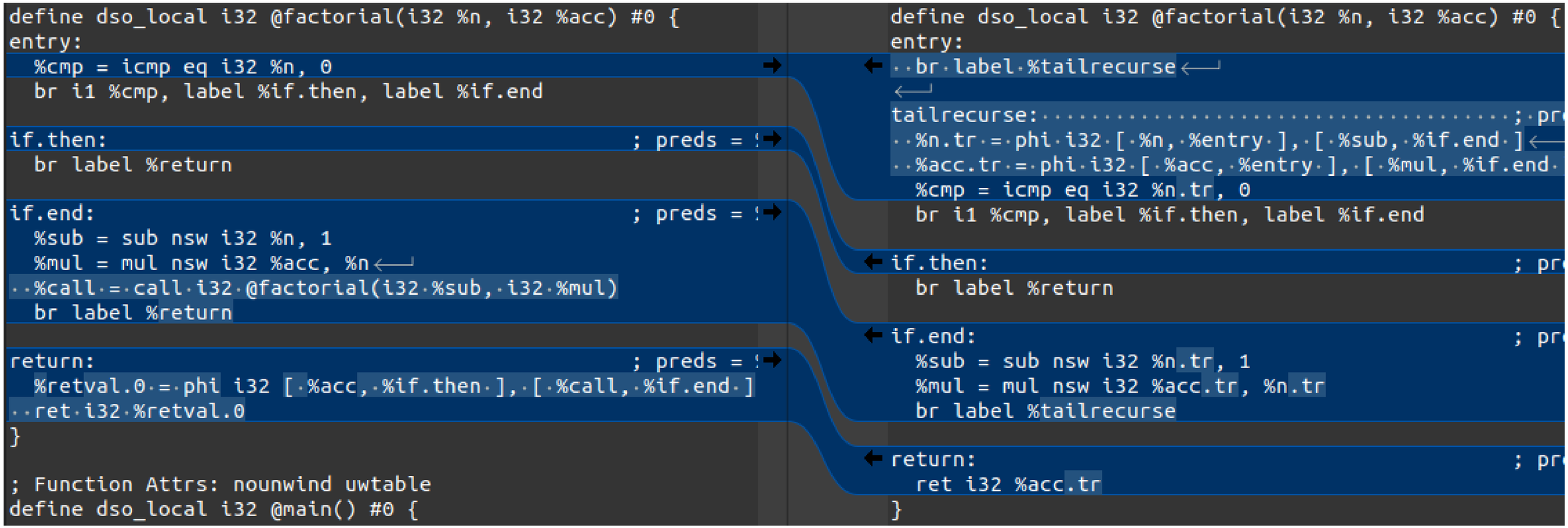
```
define dso_local i32 @factorial(i32 %n, i32 %acc) #0 {
entry:
  %retval = alloca i32, align 4
  %n.addr = alloca i32, align 4
  %acc.addr = alloca i32, align 4
  store i32 %n, i32* %n.addr, align 4, !tbaa !2
  store i32 %acc, i32* %acc.addr, align 4, !tbaa !2
  %0 = load i32, i32* %n.addr, align 4, !tbaa !2
  %cmp = icmp eq i32 %0, 0
  br i1 %cmp, label %if.then, label %if.end

if.then:                                          ; pre
  %1 = load i32, i32* %acc.addr, align 4, !tbaa !2
  store i32 %1, i32* %retval, align 4
  br label %return

if.end:                                           ; pre
  %2 = load i32, i32* %n.addr, align 4, !tbaa !2
  %sub = sub nsw i32 %2, 1
  %3 = load i32, i32* %acc.addr, align 4, !tbaa !2
  %4 = load i32, i32* %n.addr, align 4, !tbaa !2
  %mul = mul nsw i32 %3, %4
  %call = tail call i32 @factorial(i32 %sub, i32 %mul)
  store i32 %call, i32* %retval, align 4
  br label %return

return:                                           ; pre
  %5 = load i32, i32* %retval, align 4
  ret i32 %5
}
```

Слева первоначальный вариант программы, справа - измененный. (Оба IR представления без оптимизации)

```
define dso_local i32 @factorial(i32 %n, i32 %acc) #0 {
entry:
  %n.addr = alloca i32, align 4
  %acc.addr = alloca i32, align 4
  store i32 %n, i32* %n.addr, align 4, !tbaa !2
  store i32 %acc, i32* %acc.addr, align 4, !tbaa !2
  %0 = load i32, i32* %n.addr, align 4, !tbaa !2
  %cmp = icmp eq i32 %0, 0
  br i1 %cmp, label %cond.true, label %cond.false

cond.true:                                        ; preds = %en
  %1 = load i32, i32* %acc.addr, align 4, !tbaa !2
  br label %cond.end

cond.false:                                       ; preds = %en
  %2 = load i32, i32* %n.addr, align 4, !tbaa !2
  %sub = sub nsw i32 %2, 1
  %3 = load i32, i32* %acc.addr, align 4, !tbaa !2
  %4 = load i32, i32* %n.addr, align 4, !tbaa !2
  %mul = mul nsw i32 %3, %4
  %call = call i32 @factorial(i32 %sub, i32 %mul)
  br label %cond.end

cond.end:                                         ; preds = %co
  %cond = phi i32 [ %1, %cond.true ], [ %call, %cond.false ]
  ret i32 %cond
}

; Function Attrs: nounwind uwtable
define dso_local i32 @main() #0 {
```

```
define dso_local i32 @factorial(i32 %n, i32 %acc) #0 {
entry:
  %retval = alloca i32, align 4
  %n.addr = alloca i32, align 4
  %acc.addr = alloca i32, align 4
  store i32 %n, i32* %n.addr, align 4, !tbaa !2
  store i32 %acc, i32* %acc.addr, align 4, !tbaa !2
  %0 = load i32, i32* %n.addr, align 4, !tbaa !2
  %cmp = icmp eq i32 %0, 0
  br i1 %cmp, label %if.then, label %if.end

if.then:                                          ; prec
  %1 = load i32, i32* %acc.addr, align 4, !tbaa !2
  store i32 %1, i32* %retval, align 4
  br label %return

if.end:                                           ; prec
  %2 = load i32, i32* %n.addr, align 4, !tbaa !2
  %sub = sub nsw i32 %2, 1
  %3 = load i32, i32* %acc.addr, align 4, !tbaa !2
  %4 = load i32, i32* %n.addr, align 4, !tbaa !2
  %mul = mul nsw i32 %3, %4
  %call = call i32 @factorial(i32 %sub, i32 %mul)
  store i32 %call, i32* %retval, align 4
  br label %return

return:                                           ; prec
  %5 = load i32, i32* %retval, align 4
  ret i32 %5
}
```

Слева IR представление измененной программы после оптимизации -sroa, справа - последующее применение оптимизации -tailcallelim



```
define dso_local i32 @factorial(i32 %n, i32 %acc) #0 {
entry:
  %cmp = icmp eq i32 %n, 0
  br i1 %cmp, label %if.then, label %if.end

if.then:                                          ; preds =
  br label %return

if.end:                                           ; preds =
  %sub = sub nsw i32 %n, 1
  %mul = mul nsw i32 %acc, %n
  %call = call i32 @factorial(i32 %sub, i32 %mul)
  br label %return

return:                                           ; preds =
  %retval.0 = phi i32 [ %acc, %if.then ], [ %call, %if.end ]
  ret i32 %retval.0
}

; Function Attrs: nounwind uwtable
define dso_local i32 @main() #0 {
```

```
define dso_local i32 @factorial(i32 %n, i32 %acc) #0 {
entry:
  br label %tailrecurse

tailrecurse:                                      ; pr
  %n.tr = phi i32 [ %n, %entry ], [ %sub, %if.end ]
  %acc.tr = phi i32 [ %acc, %entry ], [ %mul, %if.end
  %cmp = icmp eq i32 %n.tr, 0
  br i1 %cmp, label %if.then, label %if.end

if.then:                                          ; pr
  br label %return

if.end:                                           ; pr
  %sub = sub nsw i32 %n.tr, 1
  %mul = mul nsw i32 %acc.tr, %n.tr
  br label %tailrecurse

return:                                           ; pr
  ret i32 %acc.tr
}
```