Московский Государственный Университет им. М.В. Ломоносова
факультет Вычислительной математики и кибернетики
кафедра системного программирования

Конструирование компиляторов

Исключение бесполезного кода

Москва
2015

1 Постановка задачи

Бесполезный код - множество инструкций программы, не влияющих на внешнее поведение и результат. К бесполезному коду относятся:

- *Мертвый код* - выполнение не влияет на внешнее поведение и результат.
- *Недостижимый код* - не может быть выполнен в ходе работы программы.

Требуется обнаружить и удалить мертвый и недостижимый код из программы.

2 Описание алгоритма Mark&Sweep

Классический алгоритм удаления мертвого кода работает по аналогии с mark-sweep алгоритмами сборщиков мусора, где в качестве данных выступает промежуточное представление.

Как и mark-sweep сборщики мусора, алгоритм удаления мертвого кода выполняется в два прохода.

2.1 Проход *Mark*

В начале первого прохода необходимо очистить все пометки и пометить *критические(critical)* инструкции как полезные.

Для приведенного в заданиях промежуточного представления *критическими* являются:

- инструкции, возвращающие управление из процедуры,
- инструкции ввода-вывода,
- инструкции, влияющие на значения в доступных извне процедуры областях памяти.

Все такие инструкции в начале прохода *Mark* помещаются в *Worklist*.

Algorithm 1 Процедура Mark

```
procedure MARK
    WorkList ← ∅
    for all operation i do
        clear i's mark
        if i is critical then
            mark i
            WorkList ← WorkList ∪ {i}
        end if
    end for
    while WorkList ≠ ∅ do
        remove i from WorkList
        (assume i is x ← y op z)
        if def(y) is not marked then
            mark def(y)
            WorkList ← WorkList ∪ {def(y)}
        end if
        if def(z) is not marked then
            mark def(z)
            WorkList ← WorkList ∪ {def(z)}
        end if
        for all block b ∈ RDF(block(i)) do
            let j be the branch that ends b
            if j is unmarked then
                mark j
                WorkList ← WorkList ∪ {j}
            end if
        end for
    end while
end procedure
```

Затем пока *Worklist* не пуст выполняется следующее:

- очередная иструкция I извлекается из *Worklist*,

- для каждого операнда x инструкции I:
  - для каждого определения x, если соответствующая инструкция

$$x \leftarrow y \; op \; z$$

ещё не помечена, то она помечается и помещается в *Worklist*,

- для каждого базового блока B из $RDF(block(I))$: Пусть j - последняя инструкция B. Это ветвление. Та ветка, которая ведет в $block(I)$, если она ещё не помечена, помечается и добавляется в *Worklist*.

2.2 Проход *Sweep*

Далее проход *Sweep* для каждой непомеченной инструкции делает следующее:

- если это одна из ветвей оператора ветвления, то она заменяется на инструкцию `goto` на ближайший помеченный постдоминатор,
- иначе, если это не безусловный переход, то инструкция удаляется.

Algorithm 2 Процедура Sweep

```
procedure SWEEP
    for all operation i do
        if i is unmasked then
            if i is branch then
                rewrite i with a jump
                to i's nearest marked
                postdominator
            end if
            if i is not a jump then
                delete i
            end if
        end if
    end for
end procedure
```

3 Пример

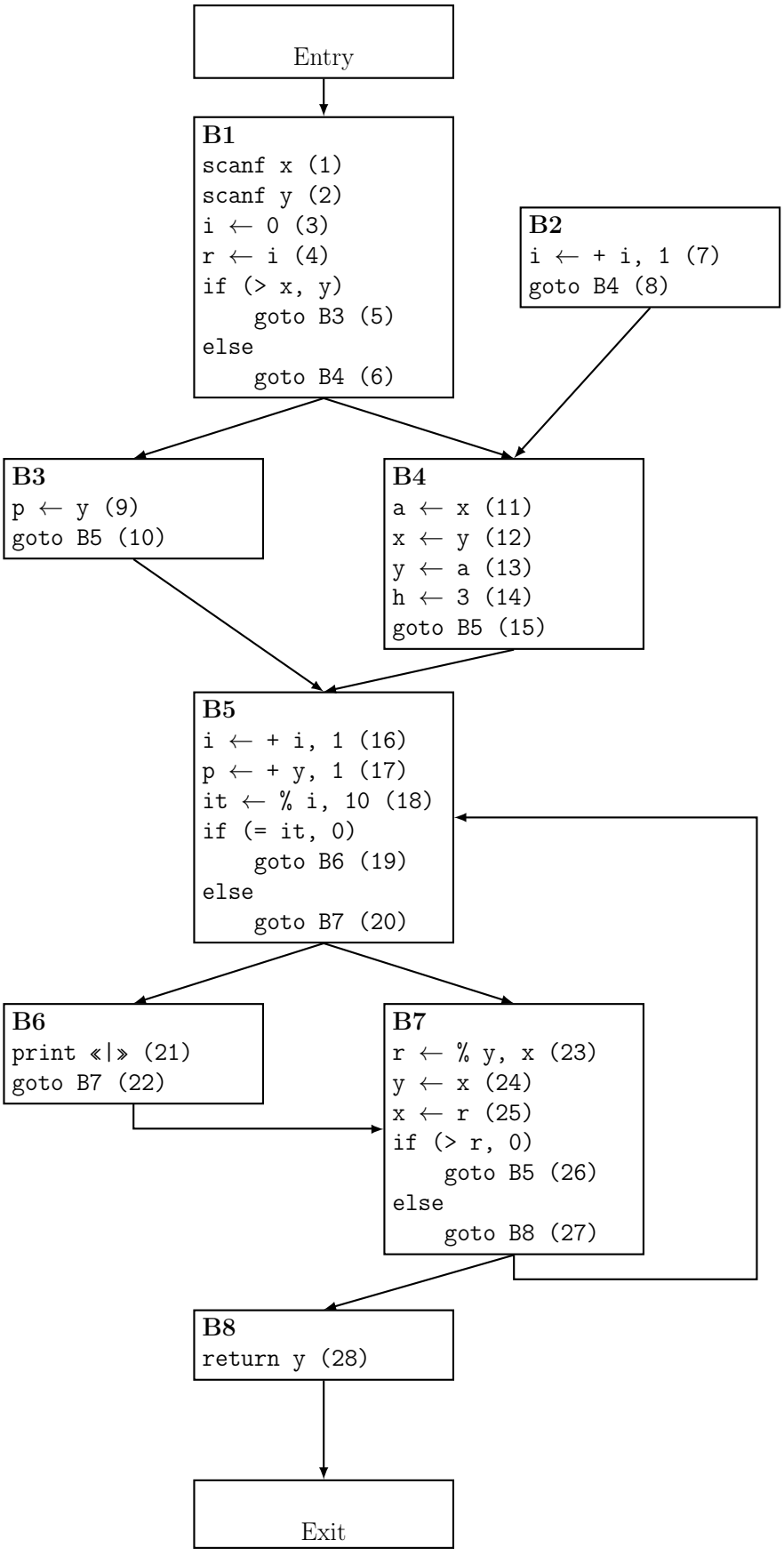

Рис. 1: Граф потока управления

3.1 Удаление мертвого кода

3.1.1 Построение обратной границы доминирования

Таблица 1: Обратная граница доминирования

Block n	RDF(n)
$B1$	$\emptyset$
$B2$	$\emptyset$
$B3$	$\{B1\}$
$B4$	$\{B1\}$
$B5$	$\{B7\}$
$B6$	$\{B5\}$
$B7$	$\emptyset$
$B8$	$\emptyset$

3.1.2 Проход *Mark*

1. В начале первого прохода помечаются и добавляются в *Worklist* критические инструкции.
   Worklist: ***(1), (2), (21), (28)***
   Marked: *(1), (2), (21), (28)*

2. Из *Worklist* вынимается инструкция *(1)*. $RDF(B1) = \emptyset$
   Worklist: *(2), (21), (28)*
   Marked: *(1), (2), (21), (28)*

3. Из *Worklist* вынимается инструкция *(2)*. $RDF(B1) = \emptyset$
   Worklist: *(21), (28)*
   Marked: *(1), (2), (21), (28)*

4. Из *Worklist* вынимается инструкция *(21)*. Помечается и добавляется в *Worklist* ветка блока $B5 \in RDF(B6)$, по которой поток управления идет в $B6$ – *(19)*.

Worklist: *(28),* ***(19)***
Marked: *(1), (2), (19), (21), (28)*

5. Из *Worklist* вынимается инструкция *(28)*. Помечаются и добавляются в *Worklist* инструкции, вычисляющие её операнды: *(13), (24)*. $RDF(B8) = \emptyset$
Worklist: *(19),* ***(13), (24)***
Marked: *(1), (2), (13), (19), (21), (24), (28)*

6. Из *Worklist* вынимается инструкция *(19)*. Помечаются и добавляются в *Worklist* инструкции, вычисляющие её операнды: *(18)*.Помечается и добавляется в *Worklist* ветка блока $B7 \in RDF(B5)$, по которой поток управления идет в $B5$ – *(26)*.
Worklist: *(13), (24),* ***(18), (26)***
Marked: *(1), (2), (13), (18), (19), (21), (24), (26), (28)*

7. Из *Worklist* вынимается инструкция *(13)*. Помечаются и добавляются в *Worklist* инструкции, вычисляющие её операнды: *(11)*. Помечается и добавляется в *Worklist* ветка блока $B1 \in RDF(B4)$, по которой поток управления идет в $B4$ – *(6)*.
Worklist: *(24), (18), (26),* ***(6), (11)***
Marked: *(1), (2), (6), (11), (13), (18), (19), (21), (24), (26), (28)*

8. Из *Worklist* вынимается инструкция *(24)*. Помечаются и добавляются в *Worklist* инструкции, вычисляющие её операнды: *(12), (25)*. $RDF(B7) = \emptyset$.
Worklist: *(18), (26), (6), (11),* ***(12), (25)***
Marked: *(1), (2), (6), (11), (12), (13), (18), (19), (21), (24), (25), (26), (28)*

9. Из *Worklist* вынимается инструкция *(18)*. Помечаются и добавляются в *Worklist* инструкции, вычисляющие её операнды: *(3), (7), (16)*. Ветви, ведущие из $RDF(B5)$ в $B5$, также уже помечены.
Worklist: *(26), (6), (11), (12), (25),* ***(3), (7), (16)***
Marked: *(1), (2), (3), (6), (7), (11), (12), (13), (16), (18), (19), (21), (24), (25), (26), (28)*

10. Из *Worklist* вынимается инструкция *(26)*. Помечаются и добавляются в *Worklist* инструкции, вычисляющие её операнды: *(4), (23)*. Ветви, ведущие

из $RDF(B5)$ в $B5$, также уже помечены.
Worklist: *(6), (11), (12), (25), (3), (7), (16),* ***(4), (23)***
Marked: *(1), (2), (3), (4), (6), (7), (11), (12), (13), (16), (18), (19), (21), (23), (24), (25), (26), (28)*

11. Из *Worklist* вынимается инструкция *(6)*. Инструкции, вычисляющие её операнды, уже помечены. $RDF(B1) = \emptyset$.
Worklist: *(11), (12), (25), (3), (7), (16), (4), (23)*
Marked: *(1), (2), (3), (4), (6), (7), (11), (12), (13), (16), (18), (19), (21), (23), (24), (25), (26), (28)*

12. Из *Worklist* вынимается инструкция *(11)*. Инструкции, вычисляющие её операнды, уже помечены. Ветви, ведущие из $RDF(B4)$ в $B4$, также уже помечены.
Worklist: *(12), (25), (3), (7), (16), (4), (23)*
Marked: *(1), (2), (3), (4), (6), (7), (11), (12), (13), (16), (18), (19), (21), (23), (24), (25), (26), (28)*

13. Из *Worklist* вынимается инструкция *(12)*. Инструкции, вычисляющие её операнды, уже помечены. Ветви, ведущие из $RDF(B4)$ в $B4$, также уже помечены.
Worklist: *(25), (3), (7), (16), (4), (23)*
Marked: *(1), (2), (3), (4), (6), (7), (11), (12), (13), (16), (18), (19), (21), (23), (24), (25), (26), (28)*

14. Из *Worklist* вынимается инструкция *(25)*. Инструкции, вычисляющие её операнды, уже помечены. $RDF(B7) = \emptyset$.
Worklist: *(3), (7), (16), (4), (23)*
Marked: *(1), (2), (3), (4), (6), (7), (11), (12), (13), (16), (18), (19), (21), (23), (24), (25), (26), (28)*

15. Из *Worklist* вынимается инструкция *(3)*. $RDF(B1) = \emptyset$.
Worklist: *(7), (16), (4), (23)*
Marked: *(1), (2), (3), (4), (6), (7), (11), (12), (13), (16), (18), (19), (21), (23), (24), (25), (26), (28)*

16. Из *Worklist* вынимается инструкция *(7)*. Инструкции, вычисляющие её операнды, уже помечены. $RDF(B2) = \emptyset$.
    Worklist: *(16), (4), (23)*
    Marked: *(1), (2), (3), (4), (6), (7), (11), (12), (13), (16), (18), (19), (21), (23), (24), (25), (26), (28)*

17. Из *Worklist* вынимается инструкция *(16)*. Инструкции, вычисляющие её операнды, уже помечены. Ветви, ведущие из $RDF(B5)$ в $B5$, также уже помечены.
    Worklist: *(4), (23)*
    Marked: *(1), (2), (3), (4), (6), (7), (11), (12), (13), (16), (18), (19), (21), (23), (24), (25), (26), (28)*

18. Из *Worklist* вынимается инструкция *(4)*. Инструкции, вычисляющие её операнды, уже помечены. $RDF(B1) = \emptyset$.
    Worklist: *(23)*
    Marked: *(1), (2), (3), (4), (6), (7), (11), (12), (13), (16), (18), (19), (21), (23), (24), (25), (26), (28)*

19. Из *Worklist* вынимается инструкция *(23)*. Инструкции, вычисляющие её операнды, уже помечены. $RDF(B7) = \emptyset$.
    Worklist:
    Marked: *(1), (2), (3), (4), (6), (7), (11), (12), (13), (16), (18), (19), (21), (23), (24), (25), (26), (28)*

20. $Worklist = \emptyset$. Первый проход *Mark* завершается.

3.1.3 Проход *Sweep*

Рассматриваем все непомеченные инструкции:

1. Инструкция *(5)*: ветка. Заменяется переходом на первый полезный постдоминатор.
   `goto B5`

2. Инструкция *(8)*: безусловный переход. Остается без изменений.

3. Инструкция *(9)*: не ветка и не безусловный переход. Удаляется.

4. Инструкция *(10)*: безусловный переход. Остается без изменений.

5. Инструкция *(14)*: не ветка и не безусловный переход. Удаляется.

6. Инструкция *(15)*: безусловный переход. Остается без изменений.

7. Инструкция *(17)*: не ветка и не безусловный переход. Удаляется.

8. Инструкция *(20)*: ветка. Заменяется переходом на первый полезный постдоминатор.
   `goto B7`

9. Инструкция *(22)*: безусловный переход. Остается без изменений.

10. Инструкция *(20)*: ветка. Заменяется переходом на первый полезный постдоминатор.
    `goto B8`

3.1.4 Результат работы алгоритма Mark&Sweep

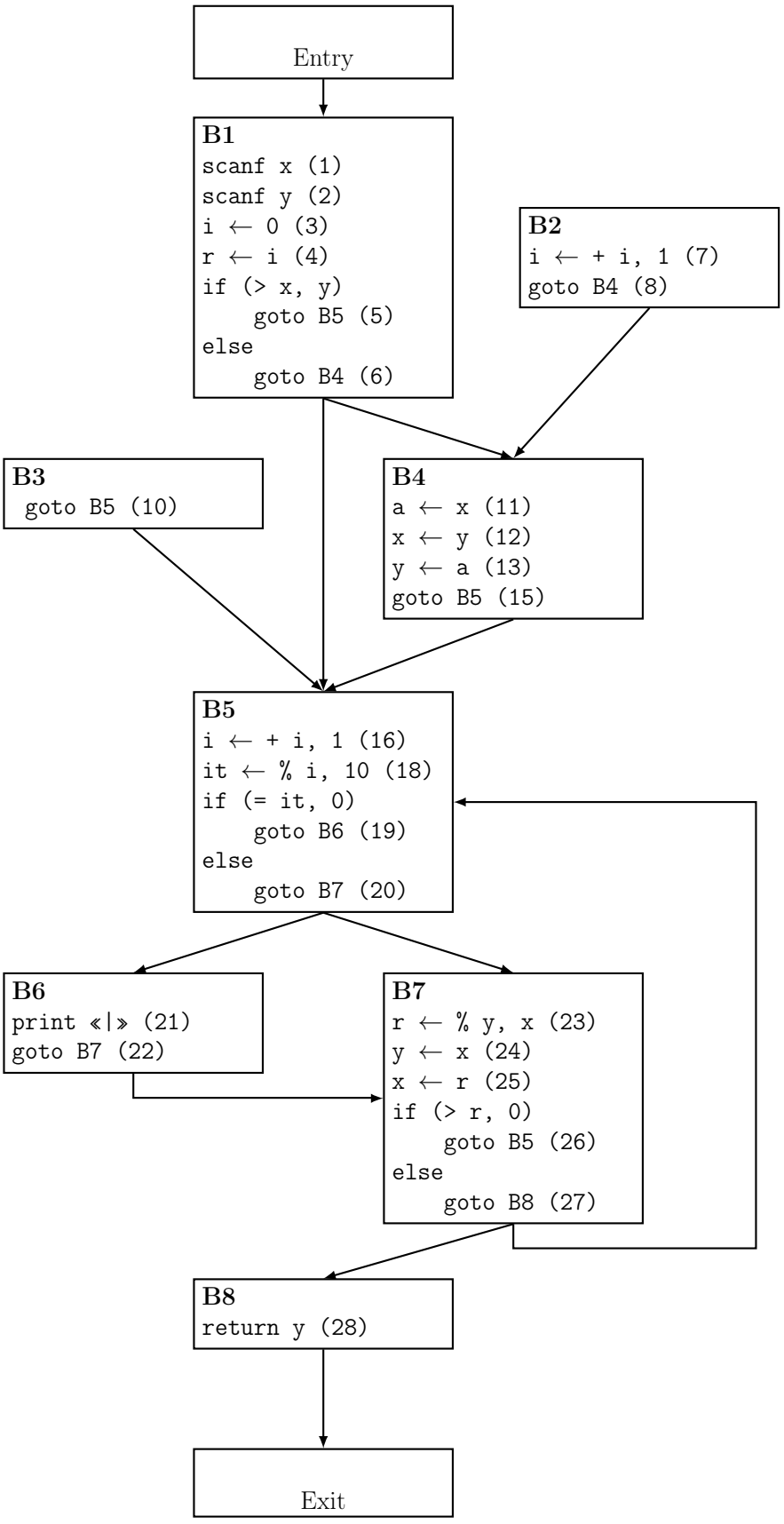

Рис. 2: Граф потока управления после применения алгортма Mark&Sweep

3.1.5 Замечания

1. Данный программа реализует алгоритм Евклида, при этом печатает на каждой десятой итерации символ «|». Значение счетчика итераций i не влияет на результат вычислений программы, но влияет на внешнее поведение, что и было обнаружено в ходе алгоритма и этот код был оставлен как полезный.

2. Тем не менее, внимательный читатель мог заметить, что инструкция *(4)* не является полезной, т.к. вычисляет мертвое значение переменной r, но она не была удалена. Этой неточности можно было бы избежать, если перед запуском алгоритма Mark&Sweep программа была бы приведена к SSA-форме.

3. Данный алгоритм не только не удаляет недостижимый код (блок $B2$ был недостижимым и остался в итоговом графе), но и порождает новый (блок $B3$ стал недостижимым). Поэтому необходимо дополнительно озаботиться удалением недостижимого кода.

3.2 Удаление недостижимого кода

Для удаления недостижимого кода необходимо обойти граф потока управления в глубину (или в ширину), начиная с блока Entry. Все непосещенные в процессе обхода блоки являются недостижимыми и подлежат удалению.

В рассмотренном выше примере необходимо удалить блоки $B2$ и $B3$ (см. Рис. 3)

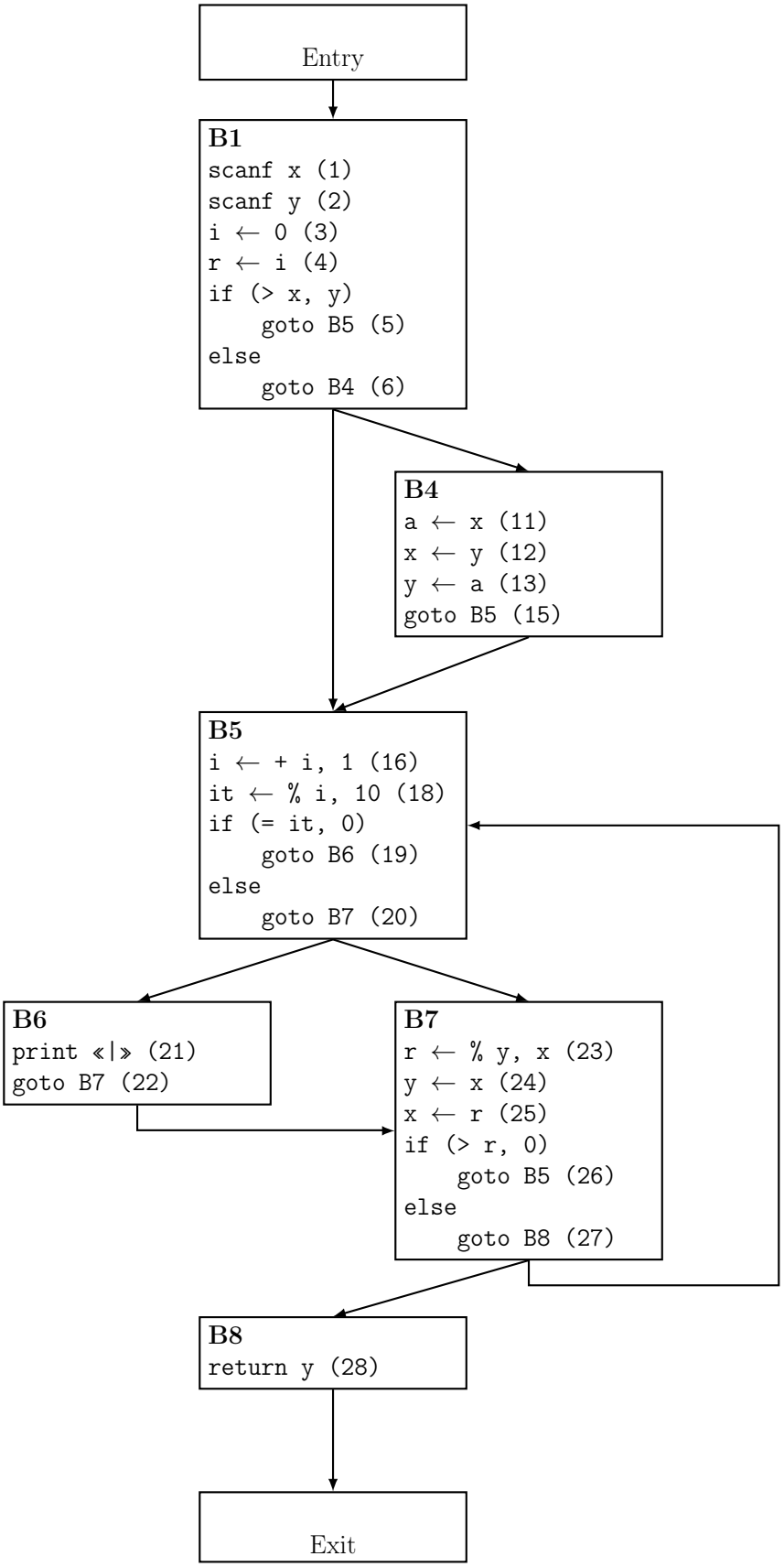

Рис. 3: Граф потока управления после удаления недостижимого кода

Список литературы

[1] Keith D. Cooper and Linda Torczon. *Engineering a Compiler, Second Edition.* Rice University, Houston, Texas, 2011

[2] Yming Zhang, Rebecca Smith, Linge Dai and Max Grossman, *Implementation of the Mark Sweep dead code elimination algorithm in LLVM.* https://yunmingzhang.files.wordpress.com/2013/12/dcereport-2.pdf