Московский государственный университет
имени М. В. Ломоносова

Lomonosov Moscow
State University

Факультет вычислительной математики и кибернетики
Кафедра математической кибернетики

Курсовая работа студента 318 группы
Балашова Александра Владимировича

Тема курсовой работы:
«О нормальной форме простых программ над двоичными деревьями»

Научный руководитель:

к.ф.-м.н., доцент

Подымов В. В.

Москва, 2023

Содержание

1 Введение

В работе рассматривается задача проверки эквивалентности программ и решение этой задачи, основанное на составлении адекватного тестового покрытия.[2] Адекватным тестовым покрытием назовем конечное подмножество множества для которых и рассматриваются программы. Задача же заключается в проверке совпадения результатов работы программ на адекватном тестовом покрытии. Эта задача неразрешима для многих языков программ [1], таких как Тьюринг-полные языки, язык примитивно рекурсивных функций и даже язык полиномов с целочисленными коэффициентами. В [2] предлагается рассмотреть класс простых программ для которых эта задача разрешима. В этих программах нет циклов, рекурсий, есть условный оператор и простейшие действия над деревьями. Программы называются простыми, потому что существует язык программ, где уже есть циклы.[3] Двоичные деревья - распространённая и простая структура данных, и при этом достаточно полезная: такие деревья можно использовать, например, для представления булевых значений, натуральных чисел и списков. В работе рассматривается класс простых программ на них. А так же рассмотрим задачу проверки эквивалентности таких программ. Цель данной работы состоит в исследовании и улучшении алгоритма проверки эквивалентности программ на двоичных деревьях, предлагающегося в [2].

2 Основные понятия

2.1 Простые программы над двоичными деревьями

Будем рассматривать двоичные деревья, у которых не может отсутствовать левое или правое поддерево. Синтаксис *деревьев* [2] зададим формой Бэкуса-Науэра

Определение 2.1 *$T ::= nil \mid (T\ .\ T)$, где nil - пустое дерево, T - двоичное дерево.*

Рассматриваются непустые корневые упорядоченные двоичные деревья. nil - это дерево из одной вершины. $(T_1.T_2)$ - это дерево, первым (левым) поддеревом которого является T_1, и вторым (правым) - T_2. Такие деревья могут быть полезны, так как с их помощью можно выразить булевы значения, натуральные числа и списки[2]:

- $\lceil \bullet \rceil_{Bool} : Bool \to T$ - отображение из множества булевых значений в множество деревьев T.
  - $\lceil false \rceil_{Bool} = nil$ — false отобравается в дерево из одной вершины.
  - $\lceil true \rceil_{Bool} = (nil.nil)$ — true отображается в дерево из трех вершин, высоты 1.

- $\lceil \bullet \rceil_{N \cup \{0\}} : N \cup \{0\} \to T$ - отображение из множества натуральных чисел в множество деревьев T.
  - $\lceil 0 \rceil_{N \cup \{0\}} = nil$ — 0 отобравается в дерево из одной вершины.
  - $\lceil n{+}1 \rceil_{N \cup \{0\}} = (nil.\lceil n \rceil_N)$ — любое число больше 0 отображается в дерево, у которого левое поддерево состоит из одной вершины, а правое - из дерева, соответствующего предыдущему числу.

- $\lceil \bullet \rceil_{List(X)} : List(X) \to T$ — отображение из множества списков состоящих из элементов типа X, имеющего представление деревьями, в множество деревьев T.
  - $\lceil [] \rceil_{List(X)} = nil$ — пустой список отобравается в дерево из одной вершины.
  - $\lceil [x_1, x_2, ..., x_n] \rceil_{List(X)} = (\lceil x_1 \rceil_X . \lceil [x_2, ..., x_n] \rceil_{List(X)})$ — любой непустой список отображается в дерево, у которого левое поддерево состоит из дерева соответствующего

первому элементу списка, а правое - из дерева соответствующего списку без первого элемента.

Введем понятие программы над описанными выше деревьями. Как и в случае с деревьями, сделаем это с помощью БНФ.

Определение 2.2 *[2] $E ::= I \mid hd \mid tl \mid nil \mid cons(E, E) \mid E \circ E \mid ifnil(E, E, E)$, где E - программа над деревьями T. Смысл операций I, hd, tl, nil, cons(), ∘, ifnil() описан ниже.*

Опишем семантику программ: каждая программа в квадратных скобках обозначает отображение из T в T: $[\bullet] : T \to T$

1. $[I](x) = x$ — программа, возвращающая входное дерево без изменений.
2. $[hd](t_1.t_2) = t_1$ — программа, возвращающая левое поддерево.
3. $[tl](t_1.t_2) = t_2$ — программа, возвращающая правое поддерево.
4. $[nil](x) = nil$ — программа, для любого дерева возвращающая nil.
5. $[cons(e_1, e_2)](x) = ([e_1](x).[e_2](x))$ — программа, возвращающая новое дерево, состоящее из поддеревьев, полученных путем применения e_1 и e_2 к дереву x.
6. $[e_1 \circ e_2](x) = [e_1]([e_2](x))$ — программа, которая последовательно применяет к дереву сначала e_2, а потом к результату применяет e_1. Назовем ее последовательной композицией.
7. $[ifnil(e_1, e_2, e_3)](x) = [e_2](x)$, если $[e_1](x) = nil$.
8. $[ifnil(e_1, e_2, e_3)](x) = [e_3](x)$, если $[e_1](x) = (t_1.t_2)$ — ifnil возвращает результат применеия e_2 к входному дереву, если результат применения e_1 к исходному дереву возвращает nil, и возвращает результат применения e_3 к входному дереву в другом случае.

Итого операциями программы считаются: I, hd, tl, nil, cons(), ∘ и ifnil(). Будем считать, что сложность каждой операции равна 1.

Определение 2.3 $|p_n|$ - *сложность программы p_n - суммарная сложность всез операций программы p_n*

2.2 Нормальная форма программ

Определение 2.4 *[2]* $E^{nf} ::= nil \mid cons(E^{nf}, E^{nf}) \mid sel_1 \circ ... \circ sel_n \mid ifnil(sel_1 \circ ... \circ sel_n, E^{nf}, E^{nf}),\ sel_i \in \{hd, tl\}, i \in [1, n]$*, где пустое множество композиций* sel *значит* I

Опишем тождества приведения программ к нормальной форме

$$\mathrm{T}_1 : I \circ e = e \circ I = e$$

$$\mathrm{T}_2 : sel \circ cons(e_1, e_2) = e_i$$

$$\mathrm{T}_3 : nil \circ e = nil$$

$$\mathrm{T}_4 : cons(e_1, e_2) \circ e_3 = cons(e_1 \circ e_3, e_2 \circ e_3)$$

$$\mathrm{T}_5 : e \circ ifnil(e_1, e_2, e_3) = ifnil(e_1, e \circ e_2, e \circ e_3)$$

$$\mathrm{T}_6 : ifnil(e_1, e_2, e_3) \circ e = ifnil(e_1 \circ e, e_2 \circ e, e_3 \circ e)$$

$$\mathrm{T}_7 : ifnil(nil, e_1, e_2) = e_1$$

$$\mathrm{T}_8 : ifnil(cons(e_h, e_t), e_1, e_2) = e_2$$

$$\mathrm{T}_9 : ifnil(ifnil(e_1, e_2, e_3), e'_1, e'_2) = ifnil(e_1, ifnil(e_2, e'_1, e'_2), ifnil(e_3, e'_2, e'_3))$$

2.3 Теорема об эквивалентности программ в нормальной форме

Введем понятие глубины $depth$ [2] дерева.

Определение 2.5 *Глубиной дерева* depth *будем называть такое отображение* $depth : T \to N$*, что* $depth(nil) = 0$*,* $depth(t_1.t_2) = 1 + max(depth(t_1), depth(t_2))$
Введем также понятие дерева глубины не более чем N.

Определение 2.6 *[2]* $T_N = \{t \in T \ / \ depth(t) \leq N\}$

Так же нам потребуется определение программ в нормальной форме глубины N

Определение 2.7 *[2]* $E_N^{nf} ::= nil \mid cons(E_N^{nf}, E_N^{nf}) \mid sel_1 \circ ... \circ sel_n \mid ifnil(sel_1 \circ ... \circ sel_n, E_N^{nf}, E_N^{nf})$, $n \le N$

Теорема 2.1 *[2] Для любого натурального n и двух любых программ из множества программ в нормальной форме, глубины не более чем N, если результат этих программ одинаков на всех деревьях глубины не более чем N+1, то результат таких программ будет одинаков на всех деревьях, то есть они равны в смысле тестирования.*

2.4 Схема проверки эквивалентности двух программ

Определение 2.8 *[2]* $nf : E \to E^{nf}$*, отображение преобразующее программу в ее нормальную форму*

Стоит отметить, что реализация алгоритма, преобразовывающего программу в нормальную форму, не была явно представлена автором статьи [2]. Этот алгоритм будет представлен в 4.1.

Определение 2.9 *Назовем* нормальным параметром *программы e число* $N \in N : e^{nf} = nf(e) \in E_N^{nf}$*, но* $e^{nf} = nf(e) \notin E_{N-1}^{nf}$

Приведем схему, позволяющую проверить, эквивалентны ли две программы e_1, e_2 [2]

1. Найдем наименьшее N такое, что $nf(e_1), nf(e_2) \in E_N^{nf}$
2. Проверим что для всех $t \in T_{N+1}$ выполняется $[e_1](t) = [e_2](t)$

В первом пункте автором [2] неявно предполагается, что поиск такого требуемого N заключается в построении нормальных форм программ e_1, e_2 и явном нахождении наибольшего числа подряд идущих операций композиций в этих формах. Сложность схемы будет зависеть от количества деревьев, глубины не более чем N+1, а их количество равно $c^{2^{N+2}}$, где c = 1.2259... [2] Сложность схемы получилась суперэкспоненциальной, а соответственно неэффективной.

3 Постановка задачи

1. Явно описать алгоритм приведения простой программы над двоичными деревьями, не вполне явно приведённый в [2], и использующийся для проверки эквивалентности программ так, как это рассказано в разделе 2.4, и оценить сложность нормальной формы, получающейся по этому алгоритму, относительно числа операций исходной программы в худшем случае.

2. Предложить алгоритм нахождения нормального параметра программы e, более эффективный по сравнению с упомянутым в разделе 2.4.

3. Оценить нормальный параметр программы для уточнения оценки сложности алгоритма, изложенного в разделе 2.4.

4 Основная часть

4.1 Алгоритм преобразования программы к нормальной форме

Алгоритм A: $A(p) = p_{nf}$, p - программа на двоичных деревьях, p_{nf} - эквивалентная программа в нормальной форме

A:

1. Пока в программе р есть тождества вида $T_1 - T_6$ применяется любое из применимых преобразований

   - Пока в программе р есть выражения вида $I \circ e$, $e \circ I$ выполняем преобразование T_1
   - Пока в программе р есть выражения вида $tl \circ cons(e_1, e_2)$, $hd \circ cons(e_1, e_2)$ выполняем преобразование T_2
   - Если в программе р есть выражения вида $nil \circ e$ выполняем преобразование T_3
   - Если в программе р есть выражения вида $cons(e_1, e_2) \circ e_3$ выполняем преобразование T_4
   - Если в программе р есть выражения вида $e \circ ifnil(e_1, e_2, e_3)$ выполняем преобразование T_5
   - Если в программе р есть выражения вида $ifnil(e_1, e_2, e_3) \circ e$ выполняем преобразование T_6

   Итогом работы этой части алгоритма будет программа p_1

2. Пока в программе p_1 есть тождества вида $T_7 - T_9$ применяется любое из применимых преобразований

   - Пока в программе p_1 есть выражения вида $ifnil(nil, e_1, e_2)$ выполняем преобразование T_7
   - Пока в программе p_1 есть выражения вида $ifnil(cons(e_h, e_t), e_1, e_2)$ выполняем преобразование T_8
   - Пока в программе p_1 есть выражения вида $ifnil(ifnil(e_1, e_2, e_3), e'_1, e'_2)$ выполняем преобразование T_9

Результатом работы алгоритма будет программа $p_2 \in E^{nf}, p_2 = nf(p)$

4.2 Оценка увеличения сложности программы при переходе к нормальной форме

Как можно заметить из оценки сложности алгоритма, проверки эквивалентности двух программ, авторы не оценивают сложность поиска нормального параметра программы. Самый очевидный способ нахождения нормального параметра программы - привести обе программы к нормальной форме и посмотреть на них. В связи с этим оценим то, во сколько раз увеличится размер программы при переходе к нормальной форме, согласно алгоритму из раздела 4.1.

Теорема 4.1 *В худшем случае нормальная форма nf(p) произвольной программы p имеет размер* $\Omega(2^{|p|})$.

Доказательство Покажем это

Рассмотрим такую последовательность программ $p_1, p_2, \ldots, p_n, \ldots$:

$p_1 : ifnil(ifnil(tl, tl, hd), tl, hd)$

$p_n : ifnil(ifnil(tl, tl, hd), tl, p_{n-1}), \forall n \geq 2$

Заметим, что $|p_1| = 7$ и $|p_n| = |p_{n-1}| + 6$, $n \geq 2$, а значит, $|p_n| = 1 + 6 * n$, $n \geq 1$

Пусть p_i^{nf} - нормальная форма программы p_i, $i \geq 1$. Тогда верно следующее:

$p_1^{nf} = ifnil(tl, ifnil(tl, tl, hd), ifnil(hd, tl, hd))$

$p_n^{nf} = ifnil(tl, ifnil(tl, tl, p_{n-1}^{nf}), ifnil(hd, tl, p_{n-1}^{nf}))$

При этом $|p_1^{nf}| = 10$ и $|p_n^{nf}| = 8 + 2 * |p_{n-1}^{nf}|$ для $n \geq 2$, а значит, $|p_n^{nf}| = 8 + 2 * 8 + 2 * 2 * 8 + \ldots + 2^{n-1} * 8 + 2^n = 9 * 2^n - 8$ для $n \geq 1$.

Таким образом, $\frac{|p_n^{nf}|}{|p_n|} = \frac{9*2^n-8}{1+6*n}$, а значит, сложность программы при переходе к нормальной форме имеет порядок $\Omega(2^k)$, где k - сложность изначальной программы. Теорема доказана.

4.3 Алгоритм поиска миимального размера программы в нормальной форме

Для поиска нормального параметра программы не обязательно приводить программы к нормальной форме, ведь, как можно заметить, нормальный параметр программы зависит лишь от того, сколько hd и tl может быть в одной композиции в нормальной форме программы. В связи с этим можно попробовать оценить это число, исходя из вида исходной программы.

Определение 4.1 *Длина композиции* $\mathrm{len\_comp} : E^{nf} \to N$ *определяется следующим образом:* $\mathrm{len\_comp}(sel_1 \circ \ldots \circ sel_n) = n$.

Определение 4.2 *Максимальная глубина программы e - максимальная длина композиции по всем возможным композициям sel в программе e.*

Лемма 4.1 *В любом из тождеств* $T_1 - T_3$ *и* $T_7 - T_9$*, преобразующих программмму e к нормальной форме, на каждом шаге их применнеия к программе e максимальная глубина программы e не может увеличиться.*

Доказательство Вытекает из вида тождеств $T_1 - T_3$ и $T_7 - T_9$.

Лемма 4.2 *В любом из тождеств* $T_4 - T_6$*, преобразующих программу p к нормальной форме, на каждом шаге максимальная глубина программы p не может стать больше чем сумма максимальных глубин в подпрограммах с обоих сторон от композиции. То есть, если в программе p есть подпрограммы вида левой части тождества* T_4*, то максимальная глубина не может стать больше, чем сумма максимальных глубин подпрограмм* e_3 *и максимальной глубины подпрограмм* e_2*,* e_1*, если в программе p есть подпрограммы вида левой части тождества* T_5*, то максимальная глубина не может стать больше, чем сумма максимальных глубин подпрограмм e и максимальной глубины подпрограмм* e_3*,* e_2*,* e_1*, если в программе p есть подпрограммы вида левой части тождества* T_6*, то максимальная глубина не может стать больше, чем сумма максимальных глубин подпрограмм e и максимальной глубины подпрограмм* e_3*,* e_2*,* e_1*.*

Доказательство Вытекает из вида тождеств $T_4 - T_6$.

Теорема 4.2 *Нормальный параметр программы e не может быть больше количества композиций изначальной программы e + 1.*

Доказательство Следует из Леммы 4.1 и Леммы 4.2. Теорема доказана.

Опишем алгоритм вычисления верхней оценки нормального параметра программы.

Алгоритм $A1$: $A1(e) = n$, $e \in E$ - исходная программа, не обязательно в нормальной форме, $n \in N$ - оценка сверху нормального параметра программы.

A1:

1. Завести счетчик $count\_comp$ числа $\circ$ в программе, проставить в него изначальное значение 0.

2. Пройти по всем операциям op_i программы, если $op_i = \circ$, увеличиваем счетчик $count\_comp$ на единицу.

3. Вернуть $count\_comp + 1$ как результат работы алгоритма.

Теорема 4.3 *Сложность алгоритма A1(e) составляет* $O(n)$, *где* $n = |e|$.

Доказательство Алгоритм A1 заключается в просмотре всех операций программы e, соответственно и сложность его будет $O(n)$, где $n = |e|$. Теорема доказана.

5 Полученные результаты

1. Описан алгоритм преобразования программы к нормальной форме.

2. Показана неэффективность описанного алгоритма, и для этого оценена снизу экспонентой сложность нормальной формы относительно сложности исходной программы.

3. Предложен алгоритм вычисления нормального параметра программы, показана эффективность этого алгоритма линейной оценкой сверху.

4. Получена оценка сверху нормального параметра программы.

Список литературы

[1] Budd T. A., Angluin D.. Two notions of correctness and their relation to testing. // Acta Informatica. — 1982.

[2] Krustev D. Simple Programs on Binary Trees - Testing and Decidable Equivalence. // Fifth International Valentin Turchin Workshop on Metacomputation. — 2016.

[3] Krustev D. A simple supercompiler formally verified in Coq. // Proceedings of the Second International Workshop on Metacomputation in Russia. — 2010.