Двусторонняя унификация программ и ее применение для задач рефакторинга

Т.А. Новикова
Казахстанский филиал МГУ им. М.В. Ломоносова, Астана, Казахстан
taniaelf@mail.ru
В.А.Захаров
ИСП РАН, Москва, Россия
zakh@cs.msu.su

Аннотация. Задача унификации пары подстановок θ_1 и θ_2 состоит в вычислении такой пары подстановок η' и η'', чтобы композиции $\theta_1\eta'$ и $\theta_2\eta''$ были равны. По существу, задача унификации подстановок равносильна задаче решения линейных уравнений вида $\theta_1 X = \theta_2 Y$ в полугруппе подстановок. Но некоторые линейные уравнения над подстановками также можно рассматривать как новые варианты задачи унификации. В этой статье мы вводим понятие двусторонней унификации как процесса преобразования одной заданной подстановки θ_1 к другой заданной подстановке θ_2 при помощи композиции, применяемой как справа, так и слева к подстановке θ_1. Иначе говоря, задача двусторонней унификации состоит в решении уравнений вида $X\theta_1 Y = \theta_2$. Двусторонняя унификация подстановок может быть использована при решении одной из задач реорганизации (рефакторинга) программ – выделения в заданном фрагменте кода тела библиотечной процедуры с целью последующей замены выделенного участка кода на вызов этой процедуры. В статье исследован вопрос о сложности задачи двусторонней унификации подстановок. Установлено, что эта задача является NP-полной. Доказательство NP-трудности задачи двусторонней унификации проводится путем сведения к ней NP-полной задачи правильного расположения домино в прямоугольной области плоскости. В статье также сформулирована и исследована задача двусторонней унификации программ в модели программ первого порядка с отношением логико-термальной эквивалентности. Доказано, что сформулированная задача двусторонней унификации программ также является NP-полной.

Ключевые слова: программа, рефакторинг, логико-термальная эквивалентность, подстановка, композиция, унификация, сложность, проблема домино.

1. Введение

Данная статья продолжает серию работ [1-4], в которых исследуется применимость методов и алгоритмов теории унификации [5] для решения задач оптимизации и верификации программ. В статьях [2] предложены

эффективные алгоритм проверки логико-термальной эквивалентности в ранее известной модели последовательных императивных программ [6,7], сложность которого существенно меньше сложности ранее известных алгоритмов (см. [8,9]). На основе этого алгоритма в статье [3] был предложен полиномиальный по времени алгоритм унификации программ, т.е. приведения двух программ к общему виду за счет выбора подходящих инициализаций входных переменных. Ранее задача унификации программ не исследовалась. В обоих случаях эффективность предложенных алгоритмов анализа и преобразования программ была обусловлена использованием быстрых алгоритмов унификации (вычисления наиболее общих примеров) и антиунификации (вычисления наиболее специальных шаблонов) заданных выражений (термов и атомов).

В данной работе рассматривается еще одна задача анализа и преобразования программ, относящаяся к области реорганизации (рефакторинга) программ [10]. Цель реорганизации программы состоит в том, чтобы за счет эквивалентных преобразований, сохраняющих функциональные свойства программы, привести ее к такому виду, который облегчает понимание программы, имеет более простую структуру, меньший размер, лишен избыточных конструкций и т. п. В частности, одной из задач реорганизации программ является задача обнаружения и устранения клонов. Содержательно, программный клон – это совокупность фрагментов программы, осуществляющих «похожие» преобразования данных. Корректное определение программного клона, сопровождаемое эффективным методом обнаружения клонов, позволяет проводить упрощения программного код путем замены нескольких больших фрагментов программы вызовом одной и той же процедуры или макроса [12]. Такое преобразование сокращает размер программы. Но оно также и облегчает понимание программы, поскольку понимание поведения нескольких разных фрагментов программы теперь сводится к пониманию поведения одной процедуры. Устранение клонов также упрощает тестирование и анализ программы, поскольку однородные ошибки, присущие фрагментам одного и того же клона, можно обнаружить и устранить при анализе поведения одной процедуры.

Попытки разработать и реализовать подходящие средства обнаружения и устранения клонов предпринимались во многих работах [13]. Главную трудность здесь составляют два вопроса: как определить понятие схожести вычислений фрагментов программ (определение клона), и как обнаружить фрагменты, имеющие сходное поведение (выявление клона). В частности, в статье [3] был предложен следующий подход к решению этих вопросов.

Предположим, что на множестве программ введено некоторое отношение эквивалентности и выделен некоторый класс «простых» программ Π. Тогда одно из возможных определений клона можно сформулировать, введя отношение Π-подобия программ. Заданная пара программ π_1 и π_2 считается Π-подобной, если существует такая программа π_0 и две такие пары программ

(ρ_1, λ_1) и (ρ_2, λ_2) из класса Π, для которых последовательная композиция программ $\rho_1; \pi_0; \lambda_1$ эквивалентна π_1, а последовательная композиция программ $\rho_2; \pi_0; \lambda_2$ эквивалентна π_2. Программу π_0 указанного вида, назовем Π-*ядром* пары программ π_1, π_2. Пары программ (ρ_1, λ_1) и (ρ_2, λ_2) в приведенном определении могут мыслиться как интерфейсы, преобразующие формат представления входных и выходных данных. Можно предполагать, что преобразования такого рода выполняются программами, имеющими сравнительно простое устройство. Тогда наличие ядра у пары программ означает, что эти программы вычисляют схожие функции и могут считаться подобными.

Аналогичным образом понятие ядра можно распространить и на целые семейства программ. Тогда Π-клоном называется семейство программ, имеющих Π-ядро. При обнаружении клона мы можем ввести в программе новую процедуру α, телом которой служит фрагмент π_0, и все фрагменты клона π_i , $1 \le i \le n$, заменить вызовом этой процедуры в составе композиций вида $\rho_i; call\ \alpha;\ \lambda_i$.

Для реализации этого подхода нужно уметь решать задачу проверки подобия программ: для произвольной заданной пары программ π_1 и π_2 выяснить существование ядро относительно заданного семейства интерфейсных программ Π. А решение этой задачи, в свою очередь, опирается на решение более простой задачи: для заданной пары программ π_1 и π_0 требуется выяснить, существует ли такая пара программ (ρ, λ) из класса Π, для которой программа π_1 эквивалентна композиции программ $\rho; \pi_0; \lambda$. Поскольку интерфейсные программы ρ, λ приводят программы π_1 и π_0 к общему виду, пару (ρ, λ) назовем *двусторонним унификатором* программ π_1 и π_0.

Понятие двусторонней унификации вводится нами впервые. Оно является естественным развитием широко известной концепции унификации для логических выражений, содержащих переменные. Задача унификации выражений (формул, атомов, термов) языка предикатов первого порядка состоит в том, чтобы для заданной пары выражений $E_1(x_1, \dots, x_n)$ и $E_2(x_1, \dots, x_n)$ отыскать такую подстановку $\theta = \{x_1/t_1, \dots, x_n/t_n)\}$, для которой выражения $E_1(x_1, \dots, x_n)\theta$ и $E_2(x_1, \dots, x_n)\theta$ становятся синтаксически одинаковыми. Впервые задачу унификации исследовал Дж. Робинсон в статье [13] в рамках разработки метода резолюций для систем автоматического доказательства теорем. В дальнейшем метод резолюций послужил отправной точкой для разработки концепции логического программирования, и алгоритмы унификации фактически стали основным средством вычисления логических программ. За прошедшие годы задача унификации была детально исследована. Было обнаружено, что унификация, помимо логического программирования, может быть использована при разработке систем автоматического доказательства теорем, обработке текстов естественных и

формальных языков, построении систем переписывания термов (см. обзор [14]). Был разработан широкий спектр эффективных алгоритмов унификации [15-20], имеющих почти линейную сложность, а также были найдены подходящие структуры данных для практической реализации этих алгоритмов.

Наиболее важным параметром в формулировке задачи унификации программ является отношение эквивалентности программ. Содержательный смысл задачи унификации требует, чтобы это отношение аппроксимировало отношение функциональной эквивалентности. В то же время, для эффективного решения задачи унификации необходимо, чтобы это выбранное отношение эквивалентности было разрешимым. Обоим требованиям удовлетворяет отношение логико-термальной (л-т) эквивалентности, введенное в статье [8]. Две программы π_1 и π_2 считаются л-т эквивалентными, если для любой синтаксически допустимой трассы $trace'$ в одной из программ существует такая трасса $trace''$ в другой программе, что в обеих трассах логические условия (предикаты) проверяются в одной и той же последовательности для одних и тех же наборов значений переменных. Как было установлено в [8], л-т эквивалентность программ влечет их функциональную эквивалентность в любой интерпретации базовых операций и предикатов. В статье [9] показано, что проверку л-т эквивалентности программ можно провести за время, полиномиально зависящее от размеров программ. Еще более быстрый алгоритм проверки л-т эквивалентности программ был предложен в статьях [2].

Цель настоящей статьи – оценить сложность задачи двусторонней унификации программ. Основным результатом статьи является теорема, показывающая, что задача проверки двусторонней унифицируемости программ относительно логико-термальной эквивалентности является NP-полной задачей.

Содержание статьи таково. Во втором разделе приведены определения основных понятий алгебры конечных подстановок [5,21], в терминах которых сформулирована задача двусторонней унификации для конечных подстановок. В этом же разделе показано, что эта задача может быть решена недетерминированным алгоритмом за полиномиальное время. Чтобы установить NP-полноту задачи двусторонней унификации подстановок, в третьем разделе рассмотрена задача о правильном расположении домино (мозаики) на ограниченной области плоскости. Эта задача, известная под названием Bounded Tiling Problem, является NP-полной задачей; она была сформулирована и исследована в статье [22]. Эта задача часто используется в качестве эталонной NP-полной задачи в теории сложности. В четвертом разделе показано, что задача о правильном расположении домино сводится к задаче проверки двусторонней унифицируемости подстановок; тем самым устанавливается NP-полнота исследуемой задачи. На основании этого результата в пятом разделе статьи показано, что задача двусторонней

унифицируемости в модели последовательных императивных программ с отношением логико-термальной эквивалентности также является NP–полной. В заключении статьи обсуждаются практические способы вычисления двусторонних унификаторов программ и методы выделения ядра у пар подобных программ.

2. Двусторонняя унификация конечных подстановок

Определим понятие конечной подстановки первого порядка и операции композиции подстановок. Пусть задано некоторое множество функциональных символов F. Символы U, X, Y, Z будем использовать для обозначения конечных множеств переменных. Множество термов $Term[X]$ над множеством переменных X определяется общепринятым для языка первого порядка образом.

Пусть $X = \{x_1, \dots, x_n\}$ и $Y = \{y_1, y_2, \dots\}$. Тогда *X-Y-подстановкой* называется всякое отображение $\theta: X \to Term[Y]$. Каждая подстановка может быть описана конечным множеством *связок* $\theta = \{x_1/\theta(x_1), \dots x_n/\theta(x_n)\}$. Переменные множества Y будем называть *входными переменными*, а переменные множества X – *выходными переменными* подстановки θ. Множество всех X-Y-подстановок обозначим записью $Subst[X, Y]$. Применение подстановки θ к терму $t(x_1, \dots, x_n)$ дает в результате терм $t\theta = t\big(\theta(x_1), \dots, \theta(x_n)\big)$, который получается из терма t одновременной заменой всех вхождений каждой переменной x_i, $1 \le i \le n$, на терм $\theta(x_i)$. *Композицией* X-Y-подстановки θ и Y-Z-подстановки η называется такая X-Z-подстановка ξ, которая удовлетворяет равенству $x\xi = (x\theta)\eta$ (или, иначе, $\xi(x) = \big(\theta(x)\big)\eta$) для каждой переменной x, $x \in X$. Для обозначения композиции подстановок θ и η будем использовать запись $\theta\eta$. Поскольку равенство $t(\theta\eta) = (t\theta)\eta$ справедливо для любого терма $t, t \in Term[X]$, операция композиции подстановок является ассоциативной. Будем называть X-X-подстановку θ *переименованием*, если отображение θ является биекцией на множестве переменных X. Две подстановки θ_1, θ_2 из семейства $Subst[Z, X]$ считаются эквивалентными, если $\theta_1 = \theta_2\rho$ для некоторого X-X-переименования ρ. Если подстановка θ_1 является композицией подстановок θ_2 и η, то подстановку θ_1 назовем *примером* подстановки θ_2, а подстановку θ_2 будем называть *шаблоном* подстановки θ_1.

Пусть заданы X-Y-подстановка θ_0 и Z-U-подстановка θ_1. Тогда пару η' и η'' из множеств $Subst[Z, X]$ и $Subst[Y, U]$ соответственно назовем *двусторонним унификатором* пары (θ_0, θ_1) тогда и только тогда, когда выполняется равенство $\eta'\theta_0\eta'' = \theta_1$. Задача двусторонней унификации состоит в том, чтобы для заданной пары подстановок указанного типа θ_0 и θ_1 вычислить двусторонний унификатор (η', η'') пары (θ_0, θ_1).

Отметим некоторые особенности задачи двусторонней унификации. Прежде всего, нетрудно увидеть, что эта задача фактически состоит в решении уравнения $\boldsymbol{X}'\theta_0\boldsymbol{X}'' = \theta_1$ относительно неизвестных подстановок $\boldsymbol{X}', \boldsymbol{X}''$ из семейств $Subst[Z, X]$ и $Subst[Y, U]$соответственно. Неявно заданные уравнения подобного вида ранее уже исследовались в теории унификации. Так, например, традиционная задача унификации выражений в языке первого порядка (см. [5]) может быть представлена как задача решения уравнений вида $\theta_0\boldsymbol{X}' = \theta_1\boldsymbol{X}''$. Задача односторонней унификации возникает во многих разделах математической логики, алгебры, теории вычислений, программирования, теории искусственного интеллекта. Подробнее с описанием прикладных возможностей односторонней унификации можно ознакомиться в статье [14]. Для ее решения было предложено немало эффективных алгоритмов (см. [15-20]), некоторые из которых вычисляют односторонний унификатор за почти линейное время относительно размера описания подстановок, представленных в виде размеченных ориентированных ациклических графов. Уравнения другого вида $\boldsymbol{X}'\theta_0 = \boldsymbol{X}''\theta_1$ также неявно возникали в работах [23], в которых исследовалась проблема эквивалентности в одном классе последовательных программ. Было показано, что решение этих уравнений можно вычислить за полиномиальное время. В связи с эти представляет интерес не только сама задача двусторонней унификации подстановок, но также и более общая задача проверки разрешимости и вычисления решения уравнений более общего вида над подстановками. Не исключено, что некоторые из этих уравнений могут иметь прикладное значение.

Следует заметить, что задача двусторонней унификации, в отличие от задачи односторонней унификации, является асимметричной, поскольку и сами исходные подстановки θ_0, θ_1, и подстановки двустороннего унификатора η', η'' относятся к разным типам и играют разную роль в содержательной интерпретации задачи унификации. Подстановки θ_0, θ_1 моделируют вычисления программ, а подстановки η'' и η' выполняют инициализацию входных данных и специализацию результатов вычисления. Поэтому, как следует из определения двусторонней унификации, термы подстановки η'' не могут непосредственно влиять на входные переменные подстановки η', но только через термы подстановки θ_0.

И, наконец, задача двусторонней унификации для подстановок θ_0, θ_1 может иметь несколько разных решений. Например, в случае $\theta_0 = \{x_1/f(y_1, y_2), x_2/y_3\}$, $\{\theta_1 = \{z/f(f(u,u), f(u,u))\}$ двусторонними унификаторами являются пары $(\eta' = \{z/f(x_1, x_1)\}, \eta'' = \{y_1/u, y_2/u\})$, $(\eta' = \{z/f(f(x_2, x_2), x_1)\}, \eta'' = \{y_1/u, y_2/u\})$ и $(\eta' = \{z/x_1\}, \eta'' = \{y_1/f(u,u), y_2/f(u,u)\})$. Однако множество попарно неэквивалентных двусторонних унификаторов заданной пары подстановок θ_0, θ_1 конечно, поскольку первая компонента η' каждого такого

унификатора является шаблоном подстановки θ_1, а число попарно неэквивалентных шаблонов всякой подстановки конечно.

Когда затрагиваются вопросы сложности задач и алгоритмов их решения, большое значение имеют способы описания задачи и структуры данных, с которыми работают алгоритмы. Например, задача односторонней унификации имеет экспоненциальную сложность, если для представления термов используются размеченные деревья, но эта же задача решается за почти линейное время, если для представления термов использовать размеченные ориентированные ациклические графы. Как будет видно из последующих разделов статьи, выбор представления термов не оказывает существенного влияния на сложность задачи двусторонней унификации. Для определенности мы будем в дальнейшем считать, что все термы в подстановках θ_0, θ_1 имеют древесное представление. Условимся обозначать записью T_θ множество размеченных деревьев (лес), представляющих все термы подстановки θ.

Очевидно, что в этом случае задача проверки двусторонней унифицируемости подстановок (θ_0, θ_1) может быть решена недетерминированным алгоритмом за полиномиальное время. Для решения этой задачи достаточно:

1. Провести недетерминированно образом по два сечения в каждом из деревьев множества T_{θ_1}, разделив лес T_{θ_1} на три части T', T_0 и T''; при этом листовые вершины каждой из частей T', T_0 совмещаются с корневыми вершинами частей T_0 и T'' соответственно;
2. Согласованно приписать переменные из множеств X и Y всем листовым вершинам фрагментов T' и T_0 соответственно (одинаковые переменные могут быть приписаны разным листовым вершинам, только если эти листья совместимы с корневыми вершинами одинаковых деревьев лежащих в следующем слое разреза).
3. Проверить, что все помеченные таким образом деревья из среднего фрагмента T_0, представляют термы из подстановки θ_0.

Очевидно, что проверить согласованность разметки листовых вершин, а также включение термов из фрагмента T_0 в древесное представление подстановки T_{θ_0} можно за время, полиномиальное относительно размеров подстановок T_{θ_0} и T_{θ_1}. Таким образом, справедлива

Лемма 1. Задача проверки двусторонней унифицируемости подстановок (θ_0, θ_1) принадлежит классу сложности NP.

NP-трудность этой задачи будет обоснована в разделе 4 путем сведения к ней ограниченного варианта проблемы домино (bounded tiling problem), которая формально описана в разделе 3.

3. Проблема ограниченного домино

Проблема домино состоит в том, чтобы покрыть заданную область плоскости четырехсторонними квадратными домино заданных типов так, чтобы смежные стороны двух соседних квадратов имели одинаковую окраску. Впервые эта задача была рассмотрена в статье [24]. Сложность этой задачи существенно зависит от формы покрываемой области. Например, задача покрытия всей плоскости посредством домино заданных типов алгоритмически неразрешима; в статье [25] было установлено, что к этому варианту проблемы домино сводится проблема (не)останова машин Тьюринга. В статье [22] было показано, что задача правильного покрытия прямоугольной области является NP-полной. В дальнейшем этот вариант проблемы домино под названием Bounded Tiling Problem широко использовался в математической логике и теории вычислений для доказательства NP-полноты многих задач. В частности, в монографии [26] проблема домино выступает в роли центральной задачи теории сложности вычислений. Мы также используем проблему домино для доказательства NP-полноты задачи двусторонней унифицируемости.

Неформальное описание ограниченного варианта проблемы домино таково. Предположим, что задано некоторое конечное множество домино $\{T_1, T_2, \dots, T_L\}$, представляющих собой квадраты единичного размера, каждая сторона которых окрашена в какой-либо цвет. Рассмотрим прямоугольник размера $n \times m$, стороны которого разделены на отрезки единичной длины. Предположим, что каждый из этих сегментов также окрашен какой-то цвет. Задача состоит в том, чтобы выяснить, можно ли расположить в указанном прямоугольнике домино заданных типов так, чтобы смежные стороны соседних домино имели одинаковую окраску, и чтобы стороны домино, прилегающие к границе прямоугольника, имели ту же окраску, что и сегменты границы, смежные с этими сторонами.

Опишем теперь эту задачу формально. Пусть задано конечное множество *цветов* $Colours = \{1,2, \dots, K\}$. *Домино* – это четверка цветов $tile = \langle a_1, a_2, a_3, a_4 \rangle$. Для выделения компонентов домино $tile$ (в указанном порядке) будем использовать записи $tile[0,-1], tile[-1,0], tile[0,1], tile[1,0]$, которые обозначают соответственно окраску северной, восточной, южной и западной сторон домино. Прямоугольная область размера $n \times m$ – это множество пар $Area = \{(i,j) : 0 \le i \le n+1, 0 \le j \le m+1\}$; элементы этого множества называются *квадратами*. Множество квадратов $Inter = \{(i,j) : 1 \le i \le n, 1 \le j \le m\}$ называется *внутренней областью* прямоугольника. *Граница* прямоугольника – это множество пар $Border = Area \setminus Interior$. Два квадрата (i_1, j_1) и (i_2, j_2) прямоугольника $Area$ считаются *соседними*, если $|i_1 - i_2| + |j_1 - j_2| = 1$. *Граничными условиями* называется всякое отображение $B: Border \to Colours$. Равенство $B(i,j) = a$ означает, что сторона граничного квадрата (i,j), прилегающая к внутренней области

прямоугольника, окрашена в цвет a. Пусть задано конечное множество типов домино $Tiles = \{tile_1, \dots, tile_L\}$. Тогда *покрытием* прямоугольника $Area$ называется всякое отображение $T: Inter \to Tiles$. Для заданного граничного условия B прямоугольника $Area$ покрытие T называется -*правильным*, если оно удовлетворяет следующим требованиям:

1. Для каждой пары соседних внутренних квадратов (i_1, j_1) и (i_2, j_2) справедливо равенство $T(i_1, j_1)[i_1 - i_2, j_1 - j_2] = T(i_2, j_2)[i_2 - i_1, j_2 - j_1]$: смежные стороны соседних домино одинаково окрашены;
2. Для каждого внутреннего квадрата (i_1, j_1), соседом которого является граничный квадрат (i_2, j_2), справедливо равенство $T(i_1, j_1)[i_1 - i_2, j_1 - j_2] = B(i_2, j_2)$: сторона домино, прилегающая к границе области, имеет ту же окраску, что и сегмент этой границы.

Примером ограниченной проблемы домино называется набор $BT = (n, m, Tiles, B)$. Этот пример считается *допустимым,* если существует -правильное покрытие прямоугольной области $Area$ размера $n \times m$ домино из множества $Tiles$. Задача Bounded Tiling Problem состоит в том, чтобы для произвольного примера ограниченной проблемы домино определить, является ли этот пример допустимым. В статье [22] показано, что проблема ограниченного домино является NP-полной.

4. NP-полнота проблемы двусторонней унифицируемости подстановок

Пусть задан произвольный пример ограниченной проблемы домино $BT = (n, m, Tiles, B)$, где $Tiles = \{tile_1, \dots, tile_L\}$, $Colours = \{1,2, \dots, K\}$. Для доказательства NP-полноты проблемы двусторонней унифицируемости подстановок покажем, что для примера BT ограниченной проблемы домино можно построить такую пару подстановок (θ_0, θ_1), которые двусторонне унифицируемы в том и только том случае, когда этот пример является допустимым. Связки X-Y-подстановки θ_0 представляют граничные условия B, а также все возможные примеры размещений домино в различных квадратах области $Area$. А Z-U-подстановка θ_1 состоит из единственной связки, которая представляет описание правильного покрытия прямоугольной области одноцветными домино, все стороны которых окрашены в цвет K. Двусторонний унификатор (η', η'') описывает решение примера BT: связки подстановки η' описывают некоторое покрытие T области $Area$, а вторая компонента унификатора η'' проверяет правильность предложенного

покрытия. Проверка правильности покрытия состоит в том, что подстановка η'' предпринимает попытку «перекрасить» стороны домино, увеличивая цвет каждой пары смежных сторон домино на одну и ту же величину. При таком способе «перекраски» достичь монохроматического покрытия удается в том и только том случае, когда исходное покрытие было правильным.

Чтобы определить формально подстановки θ_0 и θ_1 формально, мы введем множество функциональных символов F, включающее

- двухместный функциональный символ $g^{(2)}$; с его помощью будет построен терм, описывающий область покрытия $Area$;
- шестиместный функциональный символ $h^{(6)}$; с его помощью строятся термы, представляющие домино и граничные квадраты области покрытия;
- одноместный функциональный символ $f^{(1)}$; он позволяет строить термы (нумералы), представляющие натуральные числа, которыми обозначаются цвета и квадраты области покрытия.

Подстановки θ_0 и θ_1 оперируют над следующими множествами переменных

1. $X = X' \cup X''$, где
   - $X' = \{x'_{i,j} : (i,j) \in Border\}$: каждая переменная $x'_{i,j}$ привязана к граничному квадрату (i,j) области $Area$,
   - $X'' = \{x''_{i,j,\ell} : (i,j) \in Interior, 1 \le \ell \le L\}$: каждая переменная $x''_{i,j,\ell}$ ассоциируется с размещением домино $tile_\ell$ в квадрате (i,j) внутренней части области $Area$;
2. $Y = \{y_0\} \cup Y'$, где

$$Y' = \{y_{i_1,j_1,i_2,j_2} : 0 \le i_1 \le i_2 \le n+1, 0 \le j_1 \le j_2 \le m+1, |i_1 - i_2| + |j_1 - j_2| = 1\}$$

   - : каждая переменная y_{i_1,j_1,i_2,j_2} ассоциирована с парой соседних квадратов (i_1,j_1) и (i_2,j_2) области покрытия Area;
   - y_0 – вспомогательная переменная, используемая для нумерации координат квадратов;

3. $Z = \{z\}$, и $U = \{u\}$.

При помощи функционального символа $f^{(1)}$ определим рекурсивно *нумералы* – термы вида $f_n(y)$ – для каждого целого неотрицательного n следующим образом: $f_0(y) = y$ и $f_{n+1}(y) = f(f_n(y))$ для каждого n, $n \geq 0$. Очевидно, что для любых n, m верно $f_n(f_m(y)) = f_{n+m}(y)$. Число n будем называть *показателем* нумерала $f_n(y)$.

Термы, которые служат описаниями граничных условий B и размещений домино во внутренних квадратах области покрытия, определяются так.

Если граничный квадрат (i, j) располагается в одном из углов прямоугольной области покрытия, т.е. $(i, j) \in \{(0,0), (n+1,0), (n+1, m+1), (0, m+1)\}$, то этому квадрату в подстановке θ_0 сопоставляется связка $x_{i,j}/t_{i,j}$, где

$$t_{i,j} = h\left(f_i(y_0), f_j(y_0), f_K(y_0), f_K(y_0), f_K(y_0), f_K(y_0)\right).$$

Этот терм призван обозначить то, что все стороны указанного квадрата окрашены в цвет K.

Если граничный квадрат не лежит в углах прямоугольника, т.е. $(i, j) \in Border \setminus \{(0,0), (n+1,0), (n+1, m+1), (0, m+1)\}$, и при этом $B(i, j) = k$, то существует единственный соседний с ним квадрат (i', j') из внутренней области. Пусть y_{i_1,j_1,i_2,j_2} – переменная из множества Y', ассоциированная с парой соседних квадратов (i, j), (i', j'). Тогда граничные условия для квадрата (i, j) в подстановке θ_0 описываются связкой $x_{i,j}/t_{i,j}$, где

$$t_{i,j} = h\left(f_i(y_0), f_j(y_0), f_K(y_0), f_K(y_0), f_K(y_0), f_k\left(y_{i_1,j_1,i_2,j_2}\right)\right).$$

Терм $t_{i,j}$ призван обозначить тот факт, что сегмент границы, являющийся одной из сторон квадрата $t_{i,j}$, окрашен в цвет k, а все остальные стороны этого граничного квадрата окрашены в цвет K.

Если квадрат (i, j) лежит во внутренней области $Interior$ прямоугольника $Area$, то у этого квадрата есть в точности четыре соседних с ним квадрата, расположенных вверху, справа, внизу и слева от квадрата (i, j). Пусть y_{i_1,j_1,i'_1,j'_1}, y_{i_2,j_2,i'_2,j'_2}, y_{i_3,j_3,i'_3,j'_3}, и y_{i_4,j_4,i'_4,j'_4} - это четыре переменные из множества Y', которые ассоциированы с указанными четырьмя парами соседних квадратов, включающих квадрат (i, j). Тогда для каждого домино $tile_\ell = \langle k_1, k_2, k_3, k_4 \rangle$ из набора $Tiles$ введем в подстановку θ_0 связку $x_{i,j,\ell}/t_{i,j,\ell}$, где

$$t_{i,j,\ell} = h\left(f_i(y_0), f_j(y_0), f_{k_1}\left(y_{i_1,j_1,i'_1,j'_1},\right), f_{k_2}\left(y_{i_2,j_2,i'_2,j'_2}\right), f_{k_3}\left(y_{i_3,j_3,i'_3,j'_3}\right), f_{k_4}\left(y_{i_4,j_4,i'_4,j'_4}\right)\right).$$

Этот терм призван обозначать тот факт, что в квадрат (i,j) внутренней области может быть вставлено домино $tile_\ell$ с присущей этому типу домино раскраской сторон.

Располагая описанными выше тремя типами связок, определим подстановку θ_0 как множество всех указанных выше связок, т.е.

$$\theta_0 = \{x_{i,j}/t_{i,j} : (i,j) \in Border\} \cup \{x_{i,j,\ell}/t_{i,j,\ell} : (i,j) \in Interior, 1 \le \ell \le L\}.$$

Заметим, что каждая из переменных множества Y' присутствует сразу в нескольких термах подстановки θ_0, в качестве аргумента нумералов. Для каждого вхождения переменной $y, y \in Y'$, в некоторый терм подстановки θ_0 *глубиной* этого вхождения будем называть максимальный показатель тех нумералов, аргументом которых является это вхождение.

Используя только функциональный символ $g^{(2)}$, построим произвольный терм t_{area} местности $(n+2)(m+2)$ (т.е. имеющий $(n+2)(m+2)$ листовых вершин в древесном представлении). Каждый аргумент этого терма соответствует одному из квадратов в области покрытия $Area$. Для каждого квадрата (i,j) в области $Area$ введем терм $\hat{t}_{i,j} = h(f_i(u), f_j(u), f_K(u), f_K(u), f_K(u), f_K(u))$. Фактически, этот терм соответствует расположению в квадрате (i,j) домино со сторонами, монохроматически окрашенными в цвет K. Определим подстановку

$$\theta_1 = \{z/t_{area}(\hat{t}_{0,0}, \hat{t}_{0,1}, \dots, \hat{t}_{n+1,m+1})\}.$$

Эта подстановка означает, что вся область $Area$ заполнена монохроматически окрашенными домино указанного вида.

Лемма 2.

Пример ограниченной проблемы домино $BT = (n, m, Tiles, B)$ допустим тогда и только тогда, когда пара подстановок (θ_0, θ_1), определенных выше, двусторонне унифицируема.

Доказательство. 1) Пусть пример BT допустим. Тогда существует B-правильное покрытие T области $Area$ домино из семейства $Tiles$. Рассмотрим это правильное покрытие, и для каждой пары соседних квадратов $(i,j), (i',j')$ внутренней области прямоугольника, где $i \le i', j \le j'$, обозначим записью $c(i,j,i',j')$ тот цвет, которым окрашены смежные стороны домино $T(i,j)$ и $T(i',j')$, размещенных при покрытии T в указанных квадратах. Точно такой же записью обозначим тот общий цвет, в который окрашены сторона домино и смежный с этой стороной сегмент границы прямоугольника. Согласно описанию устройства термов вида $t_{i,j,\ell}$ каждое вхождение переменной $y_{i,j,i',j'}$ в термы $t_{i,j,T(i,j)}$ и $t_{i',j',T(i',j')}$ имеет одну и ту же глубину $c(i,j,i',j')$. Тогда двусторонним унификатором пары подстановок (θ_0, θ_1) является пара (η', η''), где

$$\eta' = \{z/t_{area}(x_{0,0}, x_{0,1}, \dots, x_{0,m+1}, x_{1,0}, x_{1,1,T(1,1)}, \dots, x_{1,m,T(1,m)}, x_{1,m+1}, \dots, x_{n+1,m+1})\},$$
$$\eta'' = \{y_0/u, y_{0,1,1,1}/f_{K-c(0,1,1,1)}(u), \dots, y_{i,j,i',j'}/f_{K-c(i,j,i',j')}(u), \dots\}.$$

В подстановке η' в терме t_{area} на место каждого аргумента, соответствующего квадрату (i,j) покрываемого прямоугольника, поставлена либо переменная $x_{i,j}$, если этот квадрат располагается на границе области, либо переменная $x_{i,j,T(i,j)}$, если этот квадрат располагается во внутренней части области. В последнем случае переменная $x_{i,j,T(i,j)}$ указывает, что в этот квадрат должно быть помещено домино $tile_{T(i,j)}$. В подстановке η'' вместо каждой переменной, соответствующей паре смежных сторон двух квадратов области $Area$, подставлен нумерал, дополняющий тот цвет, в который окрашены смежные стороны, располагающиеся в этих квадратах домино, до максимального цвета K.

Нетрудно убедиться, принимая во внимание B-правильность покрытия T, что имеет место равенство $\theta_1 = \eta'\theta_0\eta''$.

2) Пусть для некоторой пары подстановок (η', η'') справедливо равенство $\theta_1 = \eta'\theta_0\eta''$. Взглянув на устройство семейства размеченных деревьев, представляющих подстановку θ_1, можно заметить, что на каждой ветви вначале следуют функциональные символы g, затем функциональный символ h, и в заключении следуют функциональные символы f. Кроме того, для каждого квадрата (i,j) в терме подстановки θ_1 содержится единственный подтерм вида $h(f_i(y_0), f_j(y_0), \dots)$. В подстановке θ_0 все термы содержат только функциональные символы h и f. Таким образом, подстановка η' должна иметь вид

$$\eta' = \{z/t_{area}(x_{0,0}, x_{0,1}, \dots, x_{0,m+1}, x_{1,0}, x_{1,1,\ell_{1,1}}, \dots, x_{1,m,\ell_{1,m}}, x_{1,m+1}, \dots, x_{n+1,m+1})\},$$

а подстановка η'' должна иметь вид

$$\eta'' = \{y_0/u, y_{0,1,1,1}/f_{k_{0,1,1,1}}(u), \dots, y_{i,j,i',j'}/f_{k_{i,j,i',j'}}(u), \dots\}$$

Рассмотрим покрытие T, в котором для каждого квадрата (i,j) имеет место равенство $T(i,j) = \ell_{i,j}$ тогда и только тогда, когда в терме подстановки η' содержится переменная $x_{i,j,\ell_{i,j}}$. Покажем, что это покрытие является B-правильным.

Допустим противное. Тогда окраска двух смежных сторон каких-то двух соседних домино (или окраска стороны одного из домино и смежного с этой стороной сегмента границы) должны быть разными. Без ограничения общности ограничимся рассмотрением первого из этих двух вариантов нарушения правильности покрытия. Предположим, что несогласованность окраски сторон проявляется для домино, располагающихся в квадратах $(i_1, j_1), (i_2, j_2)$ внутренней области прямоугольника, где $i_1 \le i_2, j_1 \le j_2$. Это

означает, что в термах $t_{i_1,j_1,\ell_{i_1,j_1}}$ и $t_{i_2,j_2,\ell_{i_2,j_2}}$ вхождения разделяемой этими термами переменной y_{i_1,j_1,i_2,j_2} имеют разную глубину. Поэтому и в том терме, который представляет единственную связку композиции подстановок $\eta'\theta_0 = \{z/t_{area}(x_{0,0},\dots,x_{n+1,m+1})\theta_0\}$, оба вхождения переменной y_{i_1,j_1,i_2,j_2} имеют разную глубину. Но тогда и в композиции $\eta'\theta_0\eta''$ нумералы, располагающиеся в тех позициях подтермов $t_{i_1,j_1,\ell_{i_1,j_1}}$и $t_{i_2,j_2,\ell_{i_2,j_2}}$, которые соответствуют окраске сторон домино, будут разными. А это противоречит тому, что $\theta_1 = \eta'\theta_0\eta''$ и при этом в терме подстановки θ_1 показатели всех нумералов, обозначающих окраску сторон домино, одинаковы и равны K.

Таким образом, предложенное покрытие T является B-правильным. QED

Лемма 3. Проблема ограниченного домино $log - space$ сводима к проблеме двусторонней унифицируемости подстановок.

Доказательство. Из описания подстановок θ_0 и θ_1, приведенных в этом разделе, видно, что для всякого примера проблемы ограниченного домино $BT = (n, m, Tiles, B)$ соответствующую этому примеру пару подстановок (θ_0, θ_1) можно построить посредством детерминированного алгоритма, использующего объем памяти, пропорциональный логарифму размера описания примера. QED

Из лемм 1 и 3 вытекает

Теорема 1. Проблема двусторонней унифицируемости подстановок является NP-полной.

Здесь стоит отметить, что данная теорема является завершающим утверждением, дающим полную картину сложности проблемы разрешимости уравнений вида $\boldsymbol{X}_1^{\sigma_1}\theta\boldsymbol{X}_2^{\sigma_2} = \boldsymbol{X}_3^{\sigma_3}\eta\boldsymbol{X}_4^{\sigma_4}$ в полугруппе конечных подстановок первого порядка, где $\sigma_i \in \{0,1\}$ и при этом $\boldsymbol{X}^1 = \boldsymbol{X}$ и $\boldsymbol{X}^0 = \varepsilon$, а ε – тождественная подстановка (нейтральный элемент полугруппы). Действительно, уравнения вида $\boldsymbol{X_1\theta X_2 = X_3\eta X_4}$, $\boldsymbol{\theta X_2 = X_3\eta X_4}$ и $\boldsymbol{X_1\theta = X_3\eta X_4}$ имеют очевидные тривиальные решения вида $(\boldsymbol{X}_1 = \eta, \boldsymbol{X_2} = \boldsymbol{X}_3 = \varepsilon, \boldsymbol{X}_4 = \theta)$, $(\boldsymbol{X_2} = \eta, \boldsymbol{X}_3 = \theta, \boldsymbol{X}_4 = \varepsilon)$ и $(\boldsymbol{X}_1 = \eta, \boldsymbol{X}_3 = \theta, \boldsymbol{X}_4 = \varepsilon)$ соответственно. Уравнения вида $\boldsymbol{\theta X_2 = \eta X_4}$ и $\theta\boldsymbol{X_2} = \eta$ соответствуют проблеме унификации и, как показано в работах [16], разрешимы за почти линейное время. Уравнения вида $\boldsymbol{X_1\theta = X_3\eta}$ и $\boldsymbol{X_1\theta} = \eta$ были исследованы в статье [23] в связи с изучением проблемы эквивалентности в одном классе последовательных программ. Эти уравнения разрешимы за полиномиальное время. И, как установлено в теореме 1, лишь для уравнений вида $\boldsymbol{X_1\theta X_2} = \eta$ задача их разрешимости является NP-полной.

5. Двусторонняя унификация программ

Задача двусторонней унификации может быть обобщена и распространена на программы. В этом разделе статьи мы сформулируем проблему двусторонней

унификации для одной модели последовательных императивных программ и покажем, что эта задача также является NP-полной.

Мы будем рассматривать модель последовательных императивных программ, которая была введена в статье [6] и подробно исследована в монографии [7]. Программы в этой модели строятся из операторов ввода и вывода, операторов присваивания и тестов. Операторы ввода и вывода лишь обозначают множества входных и выходных переменных. Семантика каждого оператора присваивания e вида $x{:=}t$, где x – некоторая переменная, а t – терм, определяется подстановкой $\theta_e = \{x/t\}$. Эффект последовательной композиции операторов присваивания $e_1; e_2; \dots e_k$ описывается композицией соответствующих подстановок $\theta_{e_k} \cdots \theta_{e_2}\theta_{e_1}$. Тесты представляют собой атомарные формулы вида $P(t_1, \dots, t_m)$, где P – предикатный символ, а $t_1, \dots, t_m$ – термы.

В модели последовательных программ такого вида введено отношение логико-термальной эквивалентности программ (см. [8]). Это отношение эквивалентности аппроксимирует отношение функциональной эквивалентности [7,8] и при этом разрешимо за полиномиальное время [2,9]. Разрешимость логико-термальной эквивалентности позволяет использовать ее для построения эффективных алгоритмов верификации, оптимизации и реорганизации (рефактроинга) последовательных программ. В рамках этой модели можно сформулировать одну из важных задач реорганизации программ – задачу замены фрагментов программ вызовами заданных процедур. В этом разделе мы покажем, что эту задачу можно решить при помощи методов двусторонней унификации подстановок.

Определим формально класс последовательных программ в рассматриваемой модели. Для этого выделим три типа переменных: конечное множество входные переменных $Y = \{y_1, \dots, y_n\}$, конечное множество выходных переменных $X = \{x_1, \dots, x_m\}$, и конечное множество вспомогательных переменных Var. Пусть задано некоторое множество предикатных символов P. Обозначим записью $Atom[Var]$ множество атомарных формул, которые строятся обычным образом из предикатных символов и термов из множества $Term[Var]$. Также введем две специальные атомарные формулы $Input(y_1, \dots, y_n)$ и $Output(x_1, \dots, x_m)$ над множествами входных и выходных переменных.

Модель программы представляет собой размеченный ориентированный граф π. Две вершины этого графа – входная вершина v_{in} и выходная вершина v_{out} – особо выделены. Входной вершине v_{in} приписана атомарная формула $Input(y_1, \dots, y_n)$; из этой вершины исходит единственная дуга, которой приписана подстановка θ_{in} из множества подстановок $Subst[Var, Y]$. Выходной вершине v_{out} приписана атомарная формула $Output(x_1, \dots, x_m)$; из этой вершины не исходит ни одной дуги. Каждой внутренней вершине v этого графа, т.е. вершине, отличной от входной и выходной вершин, приписана

некоторая атомарная формула A_v из множества $Atom[Var]$. Из каждой такой вершины v исходят две дуги, одна из которых помечена символом 0, а другая – символом 1. Эти дуги могут вести в любую вершину графа за исключением входной вершины v_{in}. Каждой дуге, ведущей в графе π из внутренней вершины u во внутреннюю вершину v, приписана подстановка θ_{uv} из множества $Subst[Var, Var]$. Каждой дуге, ведущей в графе π из внутренней вершины u в выходную вершину v_{out}, приписана подстановка $\eta_{u,out}$ из множества $Subst[X, Var]$. Предполагается также, что через каждую вершину графа $\pi(Var)$ проходит некоторый маршрут, ведущий из входа программы в ее выход.

В модели программ такого вида вершины графа – это точки проверки логических условий, дуги графа соответствуют линейным участкам программы, вычислительный эффект которых описывается подстановками, приписанными этим дугам. Во входе программы осуществляет инициализация вспомогательных переменных программы, а в выходной вершине собираются результаты вычислений в виде подстановок в выходные переменные.

Существуют разные способы определения функциональных возможностей моделей программ такого рода (см. [7]). Однако, как было показано в статье [27], любой вид эквивалентности программ, опирающийся на понятия интерпретации языка первого порядка, оказывается неразрешимым. Поэтому для получения эффективных алгоритмов, разрешающих вычислительные свойства программ, приходится использовать структурные виды эквивалентности программ. Для этого приходится рассматривать и сравнивать всевозможные синтаксически допустимые трассы в анализируемых программах. Одним из таких видов эквивалентности является логико-термальная эквивалентность (л-т эквивалентность), введенная в статье [8] и подробно исследованная в работах [2,9]. Определение этой эквивалентности таково.

Трассой в программе π называется маршрут, ведущий из входной вершины в выходную вершину программы; в этом маршруте указываются все пометки, приписанные дугам. Пусть задана некоторая трасса в программе π

$$\alpha = v_0 \xrightarrow{\theta_0} v_1 \xrightarrow{\sigma_1, \theta_1} v_2 \xrightarrow{\sigma_2, \theta_2} \cdots v_{n-2} \xrightarrow{\sigma_{n-2}, \theta_{n-2}} v_{n-1} \xrightarrow{\theta_{n-1}, \sigma_{n-1}} v_n$$

,

где $v_0 = v_{in}, \sigma_i \in \{0,1\}$ для всех $i, 1 \le i \le n-1, \theta_i \in Subst[Var, Var]$ для всех $i, 1 \le i \le n-2$, $\theta_{n-1} \in Subst[Z, Var], v_n = v_{out}$. Тогда последовательность пар

$$lth(\alpha) = \\ \left(A_{v_1}\theta_0, \sigma_1\right), \left(A_{v_2}\theta_1\theta_0, \sigma_2\right), \\ \ldots, \left(A_{v_{n-1}}\theta_{n-1}\cdots\theta_1\theta_0, \sigma_{n-1}\right), \left(A_{v_n}\theta_{n-1}\theta_{n-2}\cdots\theta_1\theta_0, 1\right)$$

называется *логико-термальной историей* (л-т историей) трассы α. Двоичная последовательность $\sigma_1, \sigma_2, \dots, \sigma_{n-1}$ называется *логической характеристикой* трассы α. Нетрудно видеть, что разные трассы в программе имеют разные характеристики. *Детерминантом* программы π называется множество $Det(\pi)$=$\{lth(\alpha)$: α – трасса в программе $\pi\}$. Две программы π_1 и π_2 считаются *логико-термально* (*л-т*) *эквивалентными*, если справедливо равенство $Det(\pi_1) = Det(\pi_2)$. В статьях [2,9] показано, что задача проверки л-т эквивалентности программ разрешима за полиномиальное время.

Опишем операцию применения подстановок к программам. Пусть заданы программа π над множествами входных переменных X и выходных переменных Y, а также две подстановки η' из семейства $Subst[Z, X]$ и η'' из семейства $Subst[U, Y]$, где $U = \{u_1, \dots, u_k\}$, $Z = \{z_1, \dots, z_\ell\}$. Тогда *результатом применения пары подстановок* (η', η'') *к программе* π является программа $\eta'\pi\eta''$, которая получается из программы π в результате следующих преобразований:

- во входной вершине атом $Input(y_1, \dots, y_n)$ заменяется атомом $Input(u_1, \dots, u_k)$ и подстановка θ_0, помечающая исходящую из этой вершины дугу, замещается подстановкой $\theta_0\eta''$;
- в выходной вершине атом $Ouput(x_1, \dots, x_m)$ заменяется атомом $Output(z_1, \dots, z_\ell)$, и для каждой дуги, ведущей в выходную вершину, подстановка θ, помечающая эту дугу, замещается подстановкой $\eta'\theta$.

Программу $\eta'\pi\eta''$ можно истолковывать как вызов процедуры с телом π, в котором инициализация входных параметров осуществляется подстановкой η'', а специализация выходных параметров осуществляется подстановкой η'.

Задача двусторонней унификации программ формулируется так: для заданной пары программ π_0 и π_1 вычислить пару подстановок (двусторонний унификатор) (η', η''), для которой программы $\eta'\pi_0\eta''$ и π_1 будут л-т эквивалентными. Одно из возможных решений задачи двусторонней унификации программ опирается на следующую лемму.

Лемма 4. Пусть (η', η'') – двусторонний унификатор программ π_0 и π_1. Предположим, что в программе π_0 имеется трасса

$$\alpha = v_0 \xrightarrow{\theta_0} v_1 \xrightarrow{\sigma_1, \theta_1} v_2 \xrightarrow{\sigma_2, \theta_2} \cdots v_{n-2} \xrightarrow{\sigma_{n-2}, \theta_{n-2}} v_{n-1} \xrightarrow{\sigma_{n-1}, \theta_{n-1}} v_n$$

.

Тогда в программе π_1 существует трасса

$$\beta = v'_0 \xrightarrow{\mu_0} v'_1 \xrightarrow{\sigma_1, \mu_1} v'_2 \xrightarrow{\sigma_2, \mu_2} \cdots v'_{n-2} \xrightarrow{\sigma_{n-2}, \mu_{n-2}} v'_{n-1} \xrightarrow{\sigma_{n-1}, \mu_{n-1}} v_n$$

,

имеющая ту же самую логическую характеристику, что и трасса α, и удовлетворяющая следующему равенству $\eta'\theta_{n-1}\theta_{n-2}\cdots\theta_1\theta_0\eta''=\mu_{n-1}\mu_{n-2}\cdots\mu_1\mu_0$.

Доказательство: Если программы $\eta'\pi_0\eta''$ и π_1 л-т эквивалентны, то $Det(\eta'\pi_0\eta'') = Det(\pi_1)$. Поскольку $lth(\alpha) \in Det(\eta'\pi_0\eta'')$, в программе π_1 существует такая трасса β, что $lth(\alpha) = lth(\beta)$. Значит, трасса β должна иметь ту же самую логическую характеристику, что и трасса α, и при этом последние пары в последовательностях $lth(\alpha)$ и $lth(\beta)$ должны быть одинаковыми. Отсюда следует указанное в формулировке леммы равенство композиций подстановок. QED

Теорема 2. Задача двусторонней унифицируемости программ относительно л-т эквивалентности является NP-полной.

Доказательство. NP-трудность этой задачи следует из теоремы 1, поскольку задача унифицируемости подстановок является частным случаем задачи унифицируемости программ. О принадлежности задачи двусторонней унифицируемости программ классу сложности NP свидетельствует следующий недетерминированный алгоритм решения этой задачи за полиномиальное время. В программе π_0 нужно выбрать кратчайшую трассу

$$\alpha = v_0 \xrightarrow{\theta_0} v_1 \xrightarrow{\sigma_1, \theta_1} v_2 \xrightarrow{\sigma_2, \theta_2} \cdots v_{n-2} \xrightarrow{\sigma_{n-2}, \theta_{n-2}} v_{n-1} \xrightarrow{\sigma_{n-1}, \theta_{n-1}} v_n$$

и вычислить композицию подстановок $\theta = \theta_{n-1}\theta_{n-2}\cdots\theta_1\theta_0$. Очевидно, что это вычисление можно осуществить за время линейное относительно размера программы π_0. Затем в программе π_1 нужно выбрать трассу

$$\beta = v'_0 \xrightarrow{\mu_0} v'_1 \xrightarrow{\sigma_1, \mu_1} v'_2 \xrightarrow{\sigma_2, \mu_2} \cdots v'_{n-2} \xrightarrow{\sigma_{n-2}, \mu_{n-2}} v'_{n-1} \xrightarrow{\sigma_{n-1}, \mu_{n-1}} v_n$$

,

имеющую ту же самую логическую характеристику, что и трасса α. Согласно лемме 4 такая трасса существует и определяется однозначно. Затем для трассы β нужно вычислить композицию подстановок $\mu = \mu_{n-1}\mu_{n-2}\cdots\mu_1\mu_0$. Это вычисление также осуществимо за время, полиномиальное от размера программы π_1. Далее, опираясь на лемму 4, нужно вычислить двусторонний унификатор (η', η'') подстановок θ и μ. Согласно лемме 1 это вычисление (угадывание) можно осуществить недетерминированным алгоритмом за полиномиальное время. В заключение нужно проверить л-т эквивалентность программ $\eta'\pi_0\eta''$ и π_1. Согласно результатам статей [2,9] такую проверку можно провести за время, полиномиальное относительно размеров этих программ. QED

6. Заключение

Результаты решения задачи двусторонней унификации, представленные в этой статье, приводят к нескольким задачам, связанным с реорганизацией программ.

NP-полнота задачи двусторонней унификации подстановок означает, что для ее решения целесообразно привлечь практические средства решения вычислительно трудных задач. В связи с этим возникает вопрос о выборе подходящих средств такого рода. Вероятно, что для этой цели пригодны программные системы решения проблемы выполнимости булевых формул. В этом случае нужно разработать подходящий метод сведения задачи двусторонней унификации подстановок к проблеме SAT.

Разрешимость задачи двусторонней унификации программ позволяет приступить к исследованию более трудной, но, вместе с тем, и практически более значимой задачи выделения общей части двух программ. Для заданной пары программ π_1 и π_2 требуется выяснить, существует ли такая программа π_0 , которая имеет двустороннюю унификацию как с программой π_1, так и с программой π_2 . В случае существования такой программы π_0, ее можно использовать в качестве отдельной процедуры и заменить каждую из программ π_1 и π_2 вызовом этой процедуры для специализированных подходящим образом значений параметров. Решение этой задачи позволит значительно продвинуться в направлении автоматизации рефакторинга программ.

Список литературы

[1] Захаров В.А., Новикова Т.А.. Применение алгебры подстановок для унификации программ // Труды Института системного программирования РАН, – 2011. – т. 21 – с. 141-166.

[2] Захаров В.А., Новикова Т.А.. Полиномиальный по времени алгоритм проверки логико-термальной эквивалентности программ. // Труды Института системного программирования РАН, – 2012. – т. 22 – с. 435-455.

[3] Захаров В.А., Новикова Т.А.. Унификация программ. // Труды Института системного программирования РАН, – 2012. – т. 23 – с. 455-476.

[4] Novikova T.A., Zakharov V.A. Is it possible to unify programs? // Proceedings of the 27-th International Workshop on Unification, Epic Series. – 2013. – v. 19 – p. 35-45.

[5] Baader F., Snyder W. Unification theory // In J.A. Robinson and A. Voronkov, editors, Handbook of Automated Reasoning. — 2001. — v. 1 — p. 447-533.

[6] Luckham D.C., Park D.M., Paterson M.S., On formalized computer programs // Journal of Computer and System Science — 1970. — v.4, N 3. — p. 220-249.

[7] Котов В.Е., Сабельфельд В.К. Теория схем программ. — М.:Наука, 1991. — 348 с.

[8] Иткин В.Э. Логико-термальная эквивалентность схем программ // Кибернетика. — 1972. — N 1. — с. 5-27.

[9] Сабельфельд В.К. Полиномиальная оценка сложности распознавания логико-термальной эквивалентности // ДАН СССР. — 1979. — т. 249, N 4. — с. 793-796.

[10] Фаулер М. Рефакторинг. Улучшение существующего кода. — Символ-Плюс, 2008. — 432 с.
[11] Komondoor R., Horwitz S. Using slicing to identify duplication in source code // Proceedings of the 8th International Symposium on Static Analysis. — Springer-Verlag, 2001. — p. 40–56.
[12] Roy C. K., Cordy J. R. A survey on software clone detection research // Technical report TR 2007-541, School of Computing, Queen's University. — 2007. — v. 115.
[13] Robinson J.A. A machine-oriented logic based on the resolution principle // Journal of the ACM. 1965 — v. 12, N 1. — p. 23-41.
[14] Knight K. Unification: a multidisciplinary survey // ACM Computing Surveys — 1989. — v. 21 — N 1 — p. 93-124.
[15] Baxter L.D. An efficient unification algorithm // Technical Report CS-73-23, Dep. of Analysis and Comp. Sci., University of Waterloo, Ontario, Canada, 1973.
[16] Paterson M.S., Wegman M.N. Linear unification // The Journal of Computer and System Science. — 1978. — v. 16, N 2 — p. 158-167.
[17] Martelli A., Montanari U. An efficient unification algorithm //ACM Transactions on Program, Languages and Systems. — 1982. — v. 4, N 2 — p. 258-282.
[18] Stickel E.M. A unification algorithm for associative-commutative functions // Journal of the association for Computing Machinary. — 1981. — v. 28, N 5 — p. 423-434.
[19] Herold A., Sieckmann J. Unification in Abelean semigroups // Jornal of Automated Reasoning. — 1983. — p. 247-283.
[20] Lincoln P., Christian J. Adventures in associative-commutative unification //Journal of Symbolic Computation. — 1989. — v.8. — p. 393 — 416.
[21] Eder E. Properties of substitutions and unifications // Journal of Symbolic Computations. — v. 1. — 1985. — p. 31-46.
[22] Lewis C.H. Complexity of solvable cases of the decision problem for predicate calculus // Proceedings of the 19-th Annual Symposium on Foundations of Computer Science – 1978 - p. 35-47.
[23] Zakharov V.A. On the decidability of the equivalence problem for orthogonal sequential programs // Grammars. – 2000. – v. 2 - N 3. - p. 271-281.
[24] Wang Hao. Proving theorems by pattern recognition // Bell System Technical Journal. - 1961. - v. 40. - N 1. - p. 1-41.
[25] Berger R. The undecidability of domino problem // Memoirs of American Mathematical Society – v. 66.
[26] Lewis C.H., Papadimitriou C.H. Elements of the Theory of Computation – Prentice Hall, Englewood Cliffs, 1981.
[27] Itkin V.E., Zwinogrodski Z. On program schemata equivalence // Journal of Computer and System Science. – 1972. - v.6. - N 1. - p. 88-101.

Two-sided program unification and its application to program refactoring

T.A. Novikova
Kazakhstan Branch of Lomonosov Moscow State University, Astana, Kazakhstan
taniaelf@mail.ru
V.A.Zakharov
ISP RAS, Moscow, Russia
zakh@cs.msu.su

Annotation. It is generally accepted that to unify a pair of substitutions θ_1 and θ_2 means to find out a pair of substitutions η' and η'' such that the compositions $\theta_1\eta'$ and $\theta_2\eta''$ are the same. Actually, unification is the problem of solving linear equations of the form $\theta_1 X = \theta_2 Y$ in the semigroup of substitutions. But some other linear equations on substitutions may be also viewed as less common variants of unification problem. In this paper we introduce a two-sided unification as the process of bringing a given substitution θ_1 to another given substitution θ_2 from both sides by giving a solution to an equation $X\theta_1 Y = \theta_2$. Two-sided unification finds some applications in software refactoring as a means for extracting instances of library subroutines in arbitrary pieces of program code. In this paper we study the complexity of two-sided unification for substitutions and programs. In Section 1 we discuss the concept of unification on substitutions and programs, outline the recent results on program equivalence checking that involve the concept of unification and anti-unification, and show that the problem of library subroutine extraction may be viewed as the problem of two-sided unification for programs. In Section 2 we formally define the concept of two-sided unification on substitutions and show that the problem of two-unifiability is NP-complete (in contrast to P-completeness of one-sided unifiability problem). NP-hardness of two-sided unifiability problem is established in Section 4 by reducing to it the bounded tiling problem which is known to be NP-complete; the latter problem is formally defined in Section 3. In Section 5 we define two-sided unification of sequential programs in the first-order model of programs supplied with strong equivalence (logic&term equivalence) of programs and proved that this problem is NP-complete. This is the main result of the paper.

Keywords: program, strong equivalence, substitution, complexity, tiling problem, NP-completeness.

References

[1] Zakharov V.A., Novikova T.A. Primininie algebry podstanovok dlya unifikacii program [On the application of substitution algebra to program unification]. *Trudy ISP RAN* [The Proceedings of ISP RAS], 2011, vol. 21, p. 141-166 [in Russian].
[2] Zakharov V.A., Novikova T.A. Polynomialniy po vremeni algoritm proverki logiko-termalnoy ekvivalentnosti program [Polynomial time algorithm for checking strong

equivalence of program]. *Trudy ISP RAN* [The Proceedings of ISP RAS], 2012, vol. 22, p. 435-455 [in Russian].
[3] Zakharov V.A., Novikova T.A. Unifikaciya program [Program unification]. *Trudy ISP RAN* [The Proceedings of ISP RAS], 2012, vol. 23, p. 455-476 [in Russian].
[4] Novikova T.A., Zakharov V.A. Is it possible to unify programs? // Proceedings of the 27-th International Workshop on Unification, Epic Series. – 2013. – v. 19 – p. 35-45.
[5] Baader F., Snyder W. Unification theory. In J.A. Robinson and A. Voronkov, editors, Handbook of Automated Reasoning, 2001, v. 1, p. 447-533.
[6] Luckham D.C., Park D.M., Paterson M.S. On formalized computer programs // Journal of Computer and System Science. 1970, vol. 4, N 3, p. 220-249.
[7] Kotov V.E. Sabelfeld V.K. Teoriya skhem program [Theory of program schemata]. Moscow, "Nauka", 1991, 348 p.
[8] Itkin V.E. Logical-termal equivalence of program schemata. Proceedings of the International Sympoisum on Theoretical Programming, 1972, p. 127-143.
[9] Sabelfeld V.K. The Logic-Termal Equivalence is Polynomial-Time Decidable. Information Processing Letters, 1980, vol. 10, N 2, p. 57-62.
[10] Fauler M. Refactoring: Improving the design of existing code. 1999, Addison Wesley.
[11] Komondoor R., Horwitz S. Using slicing to identify duplication in source code // Proceedings of the 8th International Symposium on Static Analysis. Springer-Verlag, 2001, p. 40–56.
[12] Roy C. K., Cordy J. R. A survey on software clone detection research. Technical report TR 2007-541, School of Computing, Queen's University, 2007, vol. 115.
[13] Robinson J.A. A machine-oriented logic based on the resolution principle. Journal of the ACM, 1965, vol. 12, N 1, p. 23-41.
[14] Knight K. Unification: a multidisciplinary survey. ACM Computing Surveys. 1989, vol. 21, N 1, p. 93-124.
[15] Baxter L.D. An efficient unification algorithm. Technical Report CS-73-23, Dep. of Analysis and Comp. Sci., University of Waterloo, Ontario, Canada, 1973.
[16] Paterson M.S., Wegman M.N. Linear unification. The Journal of Computer and System Science. 1978, vol. 16, N 2, p. 158-167.
[17] Martelli A., Montanari U. An efficient unification algorithm. ACM Transactions on Program, Languages and Systems. 1982, vol. 4, N 2, p. 258-282.
[18] Stickel E.M. A unification algorithm for associative-commutative functions. Journal of the association for Computing Machinary. 1981, v. 28, N 5, p. 423-434.
[19] Herold A., Sieckmann J. Unification in Abelean semigroups. Journal of Automated Reasoning. 1983, p. 247-283.
[20] Lincoln P., Christian J. Adventures in associative-commutative unification. Journal of Symbolic Computation. 1989, vol. 8, p. 393-416.
[21] Eder E. Properties of substitutions and unifications. Journal of Symbolic Computations. vol. 1, 1985, p. 31-46.
[22] Lewis C.H. Complexity of solvable cases of the decision problem for predicate calculus. Proceedings of the 19-th Annual Symposium on Foundations of Computer Science, 1978, p. 35-47.
[23] Zakharov V.A. On the decidability of the equivalence problem for orthogonal sequential programs. Grammars. 2000, vol. 2, N 3, p. 271-281.
[24] Wang Hao. Proving theorems by pattern recognition. Bell System Technical Journal. 1961, vol. 40, N 1, p. 1-41.
[25] Berger R. The undecidability of domino problem. Memoirs of American Mathematical Society, v. 66.

[26] Lewis C.H., Papadimitriou C.H. Elements of the Theory of Computation – Prentice Hall, Englewood Cliffs, 1981.
[27] Itkin V.E., Zwinogrodski Z. On program schemata equivalence. Journal of Computer and System Science. 1972, vol. 6, N 1, p. 88-101.