Применение алгебры подстановок для унификации программ

В.А.Захаров (ИСП РАН), Т.А. Новикова (ф-т ВМК МГУ им. М.В. Ломоносова)
{zakh@cs.msu.su} *{taniaelf@mail.ru}*

Аннотация. Для решения многих задач системного программирования, к числу которых относятся задачи реорганизации программ, деобфускации программ, выявления уязвимостей в программном коде и др., желательно иметь инструментальное средство, позволяющее обнаруживать фрагменты программ, имеющие сходное поведение. Современные средства обнаружения программных клонов позволяют выявлять лишь фрагменты программ, имеющие сходное синтаксическое устройство, поскольку более глубокий семантический анализ программ сталкивается с алгоритмической неразрешимостью проблемы функциональной эквивалентности программ. Для того чтобы избежать алгоритмически трудных задач проверки функциональной эквивалентности, авторы настоящей статьи предлагают воспользоваться более сильным разрешимым отношением эквивалентности программ – логико-термальной эквивалентностью, – введенной в 1972 г. В.Э. Иткиным. В данной статье разработан новый алгоритм проверки логико-термальной эквивалентности программ, основанный на операции вычисления точной нижней грани в решетке конечных подстановок. На основе этого алгоритма авторам статьи удалось также решить задачу логико-термальной унификации программ, которая состоит в построении для двух заданных фрагментов программного кода такой процедуры, которая представляет собой наиболее общую специализацию этих двух фрагментов.

Ключевые слова: программа, обнаружение клонов, эквивалентность, подстановка, унификация, антиунификация.

1. Введение

Результаты статистических исследований, о которых говорится в работах [1], [2], [3], свидетельствуют о том, что суммарный объём клонов (фрагментов программ, имеющих сходное синтаксическое устройство) в больших проектах обычно составляет 7–20%, а в некоторых случаях доходит и до 50%. Наличие большого числа клонов приводит к увеличению стоимости поддержки кода, к

возрастанию ресурсов, необходимых для компиляции и хранения скомпилированной программы, а также к увеличению вероятности возникновения новых и распространения существующих ошибок [4,5]. Поэтому от избыточного дублирования кода целесообразно избавляться при помощи существующих методов реорганизации (рефакторинга) кода, например, при помощи процедурной абстракции, также называемой “Выделением метода” (Extract Method) в терминологии Фаулера [6]. В статье [7] предложен метод процедурной абстракции, позволяющий для заданной совокупности программных клонов построить процедуру, вызов которой заменяет каждый из выделенных клонов. Однако применение этого метода требует предварительного выделения подобных фрагментов программ (клонов). К сожалению, специалисты в области анализа программного обеспечения ещё не выработали общепринятого формального определения понятия клона. В отличие от многих других широко используемых в программировании понятий (например, процедуры, алгоритма) в настоящее время не существует строгого математического определения понятия клона. Все исследователи сходятся в том, что клоном называются фрагменты, имеющие незначительные синтаксические отличия друг от друга, но до сих пор еще не сложилось общего мнения о том, какие именно синтаксические отличия можно признать незначительными.

В обзоре [8] описано большое разнообразие методов выделения клонов на разных уровнях представления программы (текстовое представление, абстрактное синтаксическое дерево, граф потоков управления и данных). Однако при всем разнообразии этих методов они ограничиваются только поверхностным синтаксическим анализом программ и полностью игнорируют проверку семантических (функциональных) свойств программ. Вместе с тем, именно на основании анализа функций, вычислимых программами, можно дать строгое математическое определение отношения подобия фрагментов программ и разработать более изощренные алгоритмы выделения клонов. Задачу выделения программных клонов можно сформулировать, например, так: для пары программ π_1, π_2, реализующих функции $\Phi_1(x)$ и $\Phi_2(x)$, требуется выяснить, существует ли такая программа π_0 (программа-шаблон), реализующая функцию $\Phi_0(x)$, для которой равенства $\Phi_1(x) = \psi_1(\Phi_0(\varphi_1(x)))$ и $\Phi_2(x) = \psi_2(\Phi_0(\varphi_2(x)))$ выполняются для некоторых функций ψ_i, φ_i, $i = 1,2$, вычислимых очень простыми программами; эти программы осуществляют инициализацию переменных и последующую специализацию вычисленного результата. В этом случае программу-шаблон π_0 можно выделить в отдельный модуль (процедуру) и

заменить обе программы π_1, π_2 соответствующими вызовами этой процедуры.

Нетрудно видеть, что в самой общей постановке задача распознавания клонов всегда имеет вырожденное решение – в качестве шаблона π_0 можно выбрать пустую программу, – и поэтому интерес представляют лишь такие варианты задачи, в которых на функции инициализации φ_i и специализации ψ_i налагаются дополнительные ограничения. В настоящей статье рассматривается один вариант задачи распознавания клонов, в котором на статус программы-шаблона π_0 претендует одна из двух анализируемых программ π_1, π_2, а в качестве процедуры инициализации переменных разрешается использовать только последовательную композицию операторов присваивания вида $x_1 := t_1; x_2 := t_2; \ldots x_n := t_n$. В этом случае программа-шаблон (например, π_1) становится процедурой, а вторая программа (в данном случае π_2) заменяется вызовом этой процедуры с набором параметров $(t_1, t_2, \ldots, t_n)$. В такой постановке задача распознавания клонов является частным случаем более общей задачи унификации программ.

В задаче унификации выражений E_1 и E_2 требуется отыскать пару наиболее общих эквивалентных примеров $E_1' = E_1\theta_1$ и $E_2' = E_2\theta_2$ этих выражений. Чаще всего эквивалентность примеров E_1' и E_2' предполагает синтаксическое совпадение этих выражений. Интерес к задаче синтаксической унификации был обусловлен, в первую очередь, разработкой и применением метода резолюций в логике предикатов первого порядка [9] и развитием на его основе парадигмы логического программирования. Фактически, алгоритм унификации – это основной механизм вычисления логических программ. Задача унификации изучалась также и для языков более высокого порядка; в этих языках эта задача оказалась алгоритмически неразрешимой (см. [10]). Наряду с задачей синтаксической верификации во многих приложениях интерес представляет также и более сложная задача семантической унификации (или E-унификации); в этом случае эквивалентность выражений E_1' и E_2' предполагает равенство этих выражений в тех или иных аксиоматических теориях. Некоторые результаты исследований задачи семантической унификации в различных логических и алгебраических теориях представлены в работе [11]. В настоящей статье впервые сформулирована и исследована задача семантической унификации последовательных императивных программ. Эта задача состоит в том, чтобы

для произвольной пары программ π_1, π_2 вычислить такую пару инициализирующих последовательностей операторов θ_1, θ_2 вида $x_1 := t_1; x_2 := t_2; \ldots x_n := t_n$, для которой последовательные композиции программ $\theta_1; \pi_1$ и $\theta_2; \pi_2$ оказываются эквивалентными.

Очевидно, что алгоритмическая разрешимость задачи унификации программ возможна лишь в том случае, когда разрешимо то отношение эквивалентности программ, в рамках которого сформулирована эта задача. Известно, однако, что функциональная эквивалентность программ, семантика которых задана в свободных алгебраических системах, неразрешима [12]. Более того, как было показано в статье [13] нерекурсивно всякое невырожденное отношение эквивалентности программ, которое определяется на основании вычислений программ в интерпретациях (алгебраических системах). Поэтому в настоящей статье нами было выбрано одно из наиболее слабых разрешимых неинтерпретационных отношений эквивалентности программ – отношение логико-термальной эквивалентности.

Две программы π_1 и π_2 считаются логико-термально эквивалентными, если для любой синтаксически допустимой трассы tr' в одной из программ существует такая трасса tr'' 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. Логико-термальная (л.-т.) эквивалентность была введена в статье [14]. Интерес к л.-т. эквивалентности программ был обусловлен двумя ее свойствами:

1) л.-т. эквивалентность программ π_1 и π_2 влечет функциональную эквивалентность этих программ [14],

2) отношение л.-т. эквивалентности программ разрешимо за полиномиальное время [15].

Было установлено также, что л.-т. эквивалентность – это одно из наиболее слабых рекурсивных отношений эквивалентности программ, аппроксимирующих отношение функциональной эквивалентности.

В настоящей статье представлены два основных результата изучения задачи унификации программ:

1) новый алгоритм проверки л.-т. эквивалентности программ, использующий операции композиции и антиунификации в алгебре конечных подстановок,

2) алгоритм л.-т. унификации программ, разработанный на основе алгоритма проверки л.-т. эквивалентности программ.

Статья состоит из 7 разделов. В разделе 2. введены основные понятия теории конечных подстановок. Во разделах 3 и 4 в терминах теории конечных подстановок определена модель стандартных схем программ и логико-термальная эквивалентность программ. В разделах 5 и 6 описан алгоритм проверки л.т. эквивалентности программ и алгоритм унификации программ. В заключении обсуждаются направления дальнейших исследований задачи унификации программ.

2. Алгебра конечных подстановок

Рассмотрим конечный алфавит, состоящий из множества функциональных символов F, множества предикатных символов P и множества предметных переменных Var. Множество термов $Term(F,Var)$ – это наименьшее множество, удовлетворяющее следующим двум условиям: 1) $Var \subseteq Term(F,Var)$, и 2) если $f^{(n)} \in F$ и $\{t_1,t_2,\ldots,t_n\} \subseteq Term(F,Var)$, то $f^{(n)}(t_1,t_2,\ldots,t_n) \subseteq Term(F,Var)$. Атомарная формула (или просто атом) – это всякое выражение вида $p^{(m)}(t_1,t_2,\ldots,t_n)$, где $p^{(m)} \in P$, а $t_1,t_2,\ldots,t_n$ – термы. Множество атомарных формул обозначим записью $Atom(F,Var)$. Записи Var_t и Var_A обозначают множество переменных, входящих в терм t и атом A соответственно.

Пусть X и Y – два конечных множества переменных. Подстановкой назовем всякое отображение $\theta : X \to Term(F,Y)$, сопоставляющее каждой переменной из X некоторый терм из $Term(F,Y)$. Множество всех таких подстановок условимся обозначать записью $Subst(X,F,Y)$. Если $X = \{x_1,x_2,\ldots,x_n\}$ и $\theta(x_i) = t_i$ для всех i, $1 \le i \le n$, то подстановка θ однозначно определяется множеством (списком) пар $\{x_1/t_1, x_2/t_2, \ldots, x_n/t_n\}$. Если $t_i = x_i$, то пара (связка) $\{x_i/t_i\}$ может быть опущена в этом списке. Запись Var_θ будет использоваться для обозначения множества переменных $\bigcup_{i=1}^{n} Var_{t_i}$, входящих в состав всех термов подстановки θ. Результатом применения подстановки θ к терму

$t, t \in Term(F,X)$, называется терм $t\theta$, получающийся одновременной заменой в t каждой переменной x_i термом $\theta(x_i)$. Аналогично определяется результат применения подстановки к атому. Выражение $E\theta$, полученное в результате применения к выражению E подстановки θ, будем называть примером выражения E, специализированного подстановкой θ. Композиция $\theta\eta$ подстановок $\theta \in Subst(X,F,Y)$, $\eta \in Subst(Y,F,Z)$ – это подстановка из множества $Subst(X,F,Z)$, которая определяется равенством $\theta\eta(x) = (\theta(x))\eta$ для каждой переменной $x, x \in X$. Всякая биекция $\theta : Y \to Y$ называется переименованием.

На множестве подстановок $Subst(X,F,Y)$ определим отношение предпорядка $\prec$ и отношение эквивалентности $\approx$: для пары подстановок θ_1, θ_2 отношение $\theta_1 \prec \theta_2$ выполняется, если существует такая подстановка $\eta \in Subst(Y,F,Y)$, что $\theta_2 = \theta_1\eta$, и отношение $\theta_1 \approx \theta_2$ выполняется, если $\theta_2 = \theta_1\rho$ для некоторого переименования ρ. Отношение предпорядка $\prec$ индуцирует на множестве классов эквивалентности $Subst(X,F,Y)/\approx$ отношение частичного порядка $\le$. Частично упорядоченное множество $(Subst(X,F,Y)/\approx, \le)$ образует решетку, наименьшим элементом которой является класс эквивалентности, порожденный так называемой пустой подстановкой $\{x_1/y_1, x_2/y_2, \ldots, x_n/y_n\}$. Для того чтобы сделать решетку подстановок полной, добавим к множеству подстановок в качестве наибольшего элемента специальную мнимую подстановку τ, удовлетворяющую равенствам $\tau\theta = \theta\tau = \tau$ и $E_1\tau = E_2\tau$ для любой подстановки θ и выражений E_1, E_2. Операция взятия точной верхней грани в этой решетке называется унификацией подстановок и обозначается символом $\uparrow$, а операция взятия точной нижней грани называется антиунификацией подстановок и обозначается символом $\downarrow$. Решетка $(Subst(X,F,Y)/\approx, \le)$ удовлетворяет условию обрыва убывающих цепей. Далее записи $\theta_1 \uparrow \theta_2$ и $\theta_1 \downarrow \theta_2$ будем использовать для обозначения произвольной подстановки из классов эквивалентности $\theta_1^{\approx} \uparrow \theta_2^{\approx}$ и $\theta_1^{\approx} \downarrow \theta_2^{\approx}$. Также для любой пары подстановок $\theta', \theta' \in Subst(X',F,Y)$ и $\theta'', \theta'' \in Subst(X'',F,Y)$ в том случае, если $X' \cap X'' = \varnothing$, мы будем использовать запись $\theta' \cup \theta''$ для обозначения подстановки η, которая

представляет собой теоретико-множественное объединение связок подстановок θ' и θ''. Решетка подстановок была подробно исследована в статьях [16], [17]. В частности, установлено, что операция композиции обладает следующими двумя важными свойствами:

1. свойством монотонности: $\theta_1 \prec \theta_2 \quad \Rightarrow \quad \eta\theta_1 \prec \eta\theta_2$
2. свойством левой дистрибутивности относительно операции антиунификации: $\eta(\theta_1 \downarrow \theta_2) = (\eta\theta_1) \downarrow (\eta\theta_2)$.

Именно эти свойства композиции подстановок обеспечивают корректность преложенного нами алгоритма проверки логико-термальной эквивалентности.

Операции унификации и антиунификации подстановок – это основные алгебраические операции, которые будут использоваться для решения задачи унификации программ. Поэтому в этом разделе мы приведем также два наиболее эффективных алгоритма вычисления этих операций в решетке подстановок.

2.1. Алгоритм унификации подстановок

Описанный ниже алгоритм унификации подстановок был предложен в работе [18]. В том случае, если множества термов из области значений подстановок задаются ориентированными ациклическими графами, этот алгоритм имеет линейную сложность как по времени, так и по объему используемой памяти.

Пусть заданы две подстановки $\theta' = \{x_1 / t'_1, x_2 / t'_2, \ldots, x_n / t'_n\}$ и $\theta'' = \{x_1 / t''_1, x_2 / t''_2, \ldots, x_n / t''_n\}$, принадлежащие множествам $Subst(X, F, Y')$ и $Subst(X, F, Y'')$ соответственно таким, что $Y' \cap Y'' = \varnothing$. Для того чтобы вычислить их унификацию $\theta' \uparrow \theta''$ составим систему термальных уравнений

$$t'_1 = t''_1$$
$$t'_2 = t''_2$$
$$\vdots$$
$$t'_n = t''_n$$

и будем применять в произвольном порядке к этой систем уравнений, до тех пор пока это возможно, следующие 6 правил переписывания уравнений.

1. Уравнение $f(s'_1, s'_2, \ldots, s'_m) = f(s''_1, s''_2, \ldots, s''_m)$ замещается системой уравнений

$$s'_1 = s''_1$$
$$s'_2 = s''_2$$
$$\vdots$$
$$s'_m = s''_m$$

2. Уравнение $f(s'_1, s'_2, \ldots, s'_m) = g(s''_1, s''_2, \ldots, s''_k)$ в том случае, если $f \neq g$, завершает работу алгоритма с результатом $\theta' \uparrow \theta'' = \tau$;
3. Уравнение (тождество) $t = t$ исключается из системы;
4. Уравнение $t = y$ в том случае, если y – это переменная, а t – это терм, отличный от переменной, замещается уравнением $y = t$;
5. Уравнение $y = t$ в том случае, если y – это переменная, а t – это терм, содержащий y и отличный от y, завершает работу алгоритма с результатом $\theta' \uparrow \theta'' = \tau$;
6. Уравнение $y = t$ в том случае, если y – это переменная, t – это терм, не содержащий y, и переменная y содержится еще хотя бы в одном уравнении системы, требует применения подстановки $\{y / t\}$ ко всем остальным уравнениям системы (их левым и правым частям);

В том случае, когда образуется система уравнений, к которой неприменима ни одно из указанных выше правил, эта система будет состоять из уравнений $y_1 = t_1, \ldots, y_k = t_k$, в которых все переменные $y_1, \ldots, y_k$ попарно отличны и не содержатся в термах $t_1, \ldots, t_k$. Тогда $\theta' \uparrow \theta'' = \theta'\{y_1 / t_1, \cdots, y_k / t_k\} = \theta''\{y_1 / t_1, \cdots, y_k / t_k\}$.

Операция унификации возникает при решении задачи унификации, которая состоит в вычислении наиболее общего примера двух выражений E', E''. Если $E' = p(t_1, \ldots, t_n)$, $E' = p(s_1, \ldots, s_n)$ – это атомарные формулы, и при этом $Var_{E'} \cap Var_{E''} = \varnothing$, то наиболее общим примером этих выражений является атом $E_0 = p(x_1, \ldots, x_n)\theta_0$, где $\theta_0 = \{x_1 / t_1, \ldots, x_n / t_n\} \uparrow \{x_1 / s_1, \ldots, x_n / s_n\}$.

Приведенный выше алгоритм унификации можно применять и для решения более общей задачи унификации конечных множеств пар выражений $\{(E'_1, E''_1), (E'_2, E''_2), \ldots, (E'_m, E''_m)\}$. Эта задача состоит в вычислении

подстановки θ, применение которой приводит каждую пару выражений к общему виду. В этом случае работа алгоритма унификации начинается с системы уравнений $E'_1 = E''_1, E'_2 = E''_2, \ldots, E'_m = E''_m$.

2.2. Алгоритм антиунификации подстановок

Операция антиунификации подстановок имеет гораздо меньшую область применения, чем операция унификации. Антиунификация была введена Г. Плоткиным в статье [19] и активно использовалась для решения задач суперкомпиляции в статье [20]. В работе [21] было показано применение операции антиунификации подстановок для вычисления инвариантов программ и выделения синтаксических клонов. Оптимальный по времени алгоритм вычисления антиунификации подстановок, заданных ориентированными ациклическими графами, описан в статье [22]. Ниже приводится более простой алгоритм антиунификации подстановок, заимствованный из статьи [23].

Пусть заданы две подстановки $\theta' = \{x_1 / t'_1, x_2 / t'_2, \ldots, x_n / t'_n\}$ и $\theta'' = \{x_1 / t''_1, x_2 / t''_2, \ldots, x_n / t''_n\}$ из множества $Subst(X, F, Y)$, и пусть $Z = Y \setminus (\bigcup_{i=1}^{n} Var_{t'_i} \cup \bigcup_{i=1}^{n} Var_{t''_i})$ – множество вспомогательных переменных, не входящих в состав термов из области значений подстановок θ' и θ''. Антиунификация $\theta' \downarrow \theta''$ вычисляется итеративным алгоритмом, который строит неубывающую последовательность подстановок $\eta_0, \eta_{1,\ldots}$, преобразуя систему аннотированных уравнений, порожденных подстановками θ' и θ''. Каждое уравнение системы помечается некоторой вспомогательной переменной из множества Z. В начале работы алгоритма система аннотированных уравнений имеет вид

$$\begin{aligned} z_1 &: \; t'_1 = t''_1 \\ z_2 &: \; t'_2 = t''_2 \\ &\vdots \\ z_n &: \; t'_n = t''_n \end{aligned}$$

и $\eta_0 = \{x_1 / z_1, x_2 / z_2, \ldots, x_n / z_n\}$. Далее на каждом i-ом шаге работы алгоритма к этой систем уравнений, до тех пор пока это возможно, в

произвольном порядке применяются следующие 2 правила переписывания уравнений и вычисления последовательности подстановок $\eta_0, \eta_1, \ldots$.

1. Аннотированное уравнение $z : f(s'_1, s'_2, \ldots, s'_m) = f(s''_1, s''_2, \ldots, s''_m)$ замещается системой уравнений

$$\begin{aligned} z_{N+1} &: \; s'_1 = s''_1 \\ z_{N+2} &: \; s'_2 = s''_2 \\ &\vdots \\ z_{N+m} &: \; s'_m = s''_m \end{aligned}$$

где $z_{N+1}, z_{N+2}, \ldots z_{N+m}$ – переменные из множества Z, ранее не использованные для разметки других уравнений; при этом $\eta_i = \eta_{i-1}\{z / f(z_{N+1}, z_{N+2}, \ldots z_{N+m})\}$.

2. Если в системе есть пара одинаковых аннотированных уравнений $z' : t' = s'$ и $z'' : t'' = s''$, то одно из уравнений (например, $z' : t' = s'$) удаляется из системы; при этом $\eta_i = \eta_{i-1}\{z' / z''\}$.

Как только будет построена такая система аннотированных уравнений $z_{M+1} : s'_1 = s''_1, \ldots, z_{M+k} : s'_k = s''_k$, к которой не применимо ни одно из указанных двух правил, алгоритм прекращает работу. Результатом его работы является последняя из построенных подстановок $\eta_j = \theta' \downarrow \theta''$. Пары подстановок $\{z_{M+j} / s'_j\}$ и $\{z_{M+j} / s''_j\}$, $j = 1, \ldots, k$, соответствующие уравнениям сформировавшейся в конце работы алгоритма системы аннотированных уравнений, мы будем называть аннотирующими подстановками для вспомогательных переменных $z_{M+1}, \ldots, z_{M+k}$; для них справедливы равенства $\theta' = (\theta' \downarrow \theta'')\{z_{M+1} / s'_1, \ldots, z_{M+k} / s'_k\}$, $\theta'' = (\theta' \downarrow \theta'')\{z_{M+1} / s''_1, \ldots, z_{M+k} / s''_k\}$.

Операция антиунификации возникает при вычислении наиболее специального шаблона двух выражений E_1, E_2, т.е. такого выражения E_0, примером которого является как E_1, так и E_2. Если $E_1 = p(t_{1,}, \ldots, t_n)$, $E_1 = p(s_{1,}, \ldots, s_n)$ – это атомарные формулы, то наиболее специальным шаблоном этих выражений является атом $E_0 = p(x_{1,}, \ldots, x_n)\theta_0$, где $\theta_0 = \{x_1 / t_1, \ldots, x_n / t_n\} \downarrow \{x_1 / s_1, \ldots, x_n / s_n\}$.

3. Стандартные схемы программ

Стандартные схемы программ были введены в статье [12] в качестве математической модели для решения задач верификации и оптимизации последовательных императивных программ. Стандартная схема программ представляет собой конечную систему переходов (ориентированный размеченный граф), вершины которой соответствуют операторам ветвления, проверяющим выполнимость логических условий (предикатов), а переходы между вершинами (дуги графа) соответствуют линейным участкам программы, на которых выполняются последовательности операторов присваивания. Каждое логическое условие описывается атомарной формулой. Каждый оператор присваивания вида $y := t$ может быть ассоциирован с подстановкой $\{y/t\}$; таким образом, линейному участку программы, состоящему из операторов $y_1 := t_1; y_2 := t_2; \ldots, y_n := t_n;$, может быть сопоставлена композиция подстановок $\theta = \{y_1/t_1\}\{y_2/t_2\}\cdots\{y_n/t_n\}$, которая адекватно отражает в рамках алгебры подстановок вычислительный эффект выполнения операторов этого линейного участка.

Формальное определение стандартной схемы программ (далее просто программы) таково. Пусть заданы два конечных множества переменных $X = \{x_1, \ldots, x_n\}$ и $Y = \{y_1, \ldots, y_m\}$. Переменные множества X – это входные переменные (параметры) программы, переменные множества Y – это внутренние (локальные) переменные программы. Помимо этих двух множеств переменных при выполнении операции антиунификации подстановок мы будем использовать множество вспомогательных переменных Z.

Программой над множеством переменных X, Y называется размеченная система переходов $\pi = \langle X, Y, V, entry, exit, \to, B, \lambda_0 \rangle$, в которой

V – это конечное множество точек программы, включающее точку входа $entry$ и точку выхода $exit$,

$B : V \to Atom(F, Y)$ – это функция привязки, сопоставляющая каждой точке программы атомарную формулу (логическое условие, проверяемое в этой точке), $\to : (V \setminus \{exit\}) \times \{0,1\} \to Subst(Y, F, Y) \times V$ – это функция переходов, которая для каждой точки программы, отличной от точки выхода, и для каждого истинностного значения логического условия в этой точке указывает подстановку, описывающую результат выполнения линейного участка программы, и точку программы, в которую передается управление после выполнения этого линейного участка,

$\lambda_0 \in Subst(Y, F, X)$ –подстановка, инициализирующая локальные переменные программы.

В дальнейшем при обращении к функции переходов вместо записи $\to (v, \Delta) = (\theta, u)$ мы будем использовать более привычную запись $v \xrightarrow{\Delta, \theta} u$.

Чтобы подчеркнуть особую роль некоторых вершин программы, условимся считать, что точке выхода программы $exit$ приписана атомарная формула вида $OUT(y_{i_1}, y_{i_2}, \ldots, y_{i_k})$, которая выделяет выходные переменные среди локальных переменных программы. В этом случае можно ввести определение вычисления программы в заданной интерпретации на заданной оценке входных переменных (см. в [24]).

Пусть задана произвольная интерпретация $I = \langle D, \overline{F}, \overline{P} \rangle$ сигнатуры (F, P). Оценкой множества переменных Var в интерпретации I называется всякое отображение $\overline{d} : Var \to D$, сопоставляющее каждой переменной элемент из области интерпретации. Если $Var = \{z_1, \ldots, z_k\}$, то для обозначения оценки δ будем использовать запись $\{z_1 \leftarrow \overline{d}(z_1), \ldots, z_k \leftarrow \overline{d}(z_k)\}$. Для обозначения множества всех оценок переменных Var в интерпретации I будем использовать запись $Val(Var, I)$.

Для каждой интерпретации $I = \langle D, \overline{F}, \overline{P} \rangle$ и оценки $\overline{d}, \overline{d} \in Val(X, I)$, обычным образом определяются элемент области интерпретации $t[\overline{d}]$, являющийся значением терма $t, t \in Term(X, F)$, и истинностное значение $A[\overline{d}]$ атомарной формулы $A, A \in Atom(X, F)$. Всякая подстановка $\theta, \theta \in Subst(X, F, Y)$, в интерпретации I преобразует множество оценок переменных $Val(X, I)$ в множество оценок переменных $Val(Y, I)$ следующим образом: если $\theta = \{y_1/t_1, y_2/t_2, \ldots, y_n/t_n\}$, то $\theta(\overline{d}) = \{y_1 \leftarrow t_1[\overline{d}], y_2 \leftarrow t_2[\overline{d}] \ldots, y_n \leftarrow t_n[\overline{d}]\}$.

Для каждой программы $\pi = \langle X, Y, V, entry, exit, \to, B, \lambda_0 \rangle$ и интерпретации I оценки переменных $Val(X, I)$ играют роль входных

данных, а оценки данных $Val(Y,I)$ выступают роли состояний данных вычислений программы. Вычисление программы π в интерпретации I для оценки входных переменных $\bar{d}, \bar{d} \in Val(X,I)$ – это максимальная последовательность

$$comp(\pi, I, \bar{d}) = (v_0, \bar{d}_0), (v_1, \bar{d}_1), (v_2, \bar{d}_2), \ldots, (v_i, \bar{d}_i), (v_{i+1}, \bar{d}_{i+1}), \ldots ,$$

удовлетворяющая следующим условиям:

1) $v_0 = entry, \quad \bar{d}_0 = \lambda_0(\bar{d})$;

2) для любого $i, i \geq 1$, в программе π существует переход $v_i \xrightarrow{\Delta,\theta} v_{i+1}$, где $\Delta = B(v_i)(\bar{d}_i)$, и при этом $\bar{d}_{i+1} = \theta(\bar{d}_i)$.

Если вычисление $comp(\pi, I, \vec{d})$ бесконечно, то результат его не определен. Если вычисление $comp(\pi, I, \vec{d})$ завершается парой $(exit, \bar{d}_N)$, и точке выхода приписана атомарная формула $OUT(y_{i_1}, y_{i_2}, \ldots, y_{i_k})$, то результатом вычисления является набор значений переменных $(\bar{d}_N(y_{i_1}), \bar{d}_N(y_{i_2}), \ldots, \bar{d}_N(y_{i_k}))$. Таким образом, для каждой интерпретации I значением программы π в этой интерпретации является частичная функция $\Phi_{\pi,I} : Val(X,I) \to D^k$, осуществляющая отображение оценок входных переменных в наборы значений переменных на выходе программы. Программы π' и π'' считаются функционально эквивалентными, если для любой интерпретации I эти программы вычисляют одинаковые функции, т.е. $\Phi_{\pi',I} =: \Phi_{\pi'',I}$.

В работе [12] было доказано, что определенная таким образом функциональная эквивалентность программ алгоритмически неразрешима. Последующие исследования показали, что любое невырожденное отношение эквивалентности программ, определяемое на основе вычислений в интерпретации, неразрешимо даже для очень простых сигнатур (F,P). В связи с этим в статье [14] была предложена иная разновидность программной эквивалентности, – логико-термальная эквивалентность, – которая занимает промежуточное положение между семейством чисто семантических отношений эквивалентности программ, наподобие функциональной эквивалентности, и семейством чисто синтаксических отношений эквивалентности, наподобие изоморфизма программ.

4. Логико-термальная эквивалентность программ

В отличие от функциональной эквивалентности, опирающейся на семантическое понятие вычисления программы в интерпретации, определение логико-термальная эквивалентности программ основывается лишь на сопоставлении синтаксических характеристик анализируемых программ – множеств трасс, ведущих из входов программ в их выходы.

Последовательность переходов в программе π

$$tr = v_0 \xrightarrow{\Delta_0,\theta_0} v_1 \xrightarrow{\Delta_1,\theta_1} v_2 \xrightarrow{\Delta_2,\theta_2} \cdots v_N \xrightarrow{\Delta_N,\theta_N} v_{N+1},$$

в которой $v_0 = entry$, мы будем называть трассой в программе π, ведущей в точку v_{N+1}. Если $v_{N+1} = exit$, то трасса tr будет называться полной трассой. Множество всех полных трасс программы π обозначим записью $CompTr(\pi)$. Префикс длины $i, 0 \leq i \leq N+1,$ трассы tr будет обозначаться записью $tr|_{\leq i}$; при этом мы полагаем $tr|_{\leq 0} = v_0$ и $tr|_{\leq N+1} = tr$. Каждая трасса tr в программе π вычисляет подстановку $\theta[tr] = \theta_N \cdots \theta_2\theta_1\theta_0\lambda_0$, которая представляет собой композицию инициализирующей подстановки λ_0 и всех подстановок $\theta_0, \theta_1, \theta_2, \ldots \theta_N$, приписанных переходам этой трассы. Логико-термальная история (л.-т. история) полной трассы tr – это последовательность пар

$$lth(tr) = (B(v_0)\theta(tr|_{\leq 0}), \Delta_0), \ldots, (B(v_i)\theta(tr|_{\leq i}), \Delta_i), \ldots (B(v_N)\theta(tr|_{\leq N}), \Delta_N), (B(v_{N+1})\theta(tr), 0),$$

в которой первые компоненты пар – это примеры атомов $B(v_i)$, приписанных точкам трассы tr и специализированные подстановками $\theta(tr|_{\leq i})$, вычисленными в этих точках трассы, а вторые компоненты пар – это истинностные значения этих атомов, обеспечивающие прохождение трассы. Множество $Det(\pi) = \{lth(tr) : tr \in CompTr(\pi)\}$ л.-т. историй всех полных трасс программы π называется детерминантом программы. Программы π' и π'' считаются л.-т. эквивалентными, если $Det(\pi') = Det(\pi'')$.

Далее, не ограничивая общности, мы будем полагать, что в рассматриваемых программах через каждую точку проходит хотя бы одна полная трасса, т.е. в программах отсутствуют недостижимые и тупиковые точки. Программы такого вида будем называть редуцированными.

Логико-термальная эквивалентность программ была предложена В.Э. Иткиным в статье [14]. В этой же статье была доказана теорема, устанавливающая взаимосвязь отношений логико-термальной и функциональной эквивалентности программ.

Теорема [14]. Если программы π' и π'' л.-т. эквивалентны, то в любой интерпретации эти программы вычисляют одинаковые функции.

Эта теорема показывает, что для проверки функциональной эквивалентности двух программ достаточно проверить их л.-т. эквивалентность. На основании этой теоремы л.-т. эквивалентность программ можно использовать при решении задач верификации и оптимизации программ.

5. Алгоритм проверки логико-термальной эквивалентности программ

Первый алгоритм, разрешающий л.-т. эквивалентность стандартных схем программ был предложен в статье [14]. Сложность его оценивается двойной экспонентой, зависящей от размера проверяемых программ. Впоследствии были созданы более эффективные алгоритмы; в частности, в работе [15] был разработан алгоритм проверки л.-т. эквивалентности программ за время, полиномиальное относительно размеров программ. В этом разделе статьи описан еще более простой алгоритм проверки л.-т. эквивалентности программ. Мы сводим проверку л.-т. эквивалентности программ к вычислению точной нижней грани в регулярном множестве подстановок; это вычисление осуществляется итеративной процедурой, в которой применяются только операции композиции и антиунификации. Для завершения работы процедуре требуется совершить полиномиальное число шагов. Кроме того, после небольшой модификации эту процедуру можно использовать для решения задачи логико-термальной унификации программ.

Следующее необходимое и достаточное условие л.-т. эквивалентности программ вытекает непосредственно из определения этого отношения эквивалентности программ.

Теорема 1. Для того чтобы редуцированные программы $\pi' = \langle X, Y', V', entry', exit', \to', B', \lambda_0' \rangle$ и $\pi'' = \langle X, Y'', V'', entry'', exit'', \to'', B'', \lambda_0'' \rangle$ были л.-т. эквивалентными необходимо и достаточно, чтобы для любой конечной двоичной последовательности $\omega = \Delta_0, \Delta_1, \ldots, \Delta_N$, всякий раз, когда в одной из программ (например, в программе π') существует трасса

$$tr' = v_0' \xrightarrow{\Delta_0, \theta_0'} v_1' \xrightarrow{\Delta_1, \theta_1'} v_2' \xrightarrow{\Delta_2, \theta_2'} \cdots v_N' \xrightarrow{\Delta_N, \theta_N'} v_{N+1}',$$

переходы которой размечены символами последовательности ω в другой программе (в данном случае, в программе π'') существует трасса

$$tr'' = v_0'' \xrightarrow{\Delta_0, \theta_0''} v_1'' \xrightarrow{\Delta_1, \theta_1''} v_2'' \xrightarrow{\Delta_2, \theta_2''} \cdots v_N'' \xrightarrow{\Delta_N, \theta_N''} v_{N+1}'',$$

переходы которой размечены символами той же последовательности ω, и при этом выполняется равенство $B'(v_{N+1}')\theta[tr'] = B''(v_{N+1}'')\theta[tr'']$

Трассы tr' и tr'', о которых говорится в теореме 1, будем называть логически согласованными трассами. Таким образом, теорема 1 сводит проверку л.т. эквивалентности программ к анализу пар логически согласованных трасс в этих программах. Этот анализ осуществляется в графе согласованных трасс программ π' и π'', представляющем все пары логически согласованных трасс в этих программах.

Предположим, что требуется проверить логико-термальную эквивалентность двух программ

$\pi' = \langle X, Y', V', entry', exit', \to', B', \lambda_0' \rangle$ и

$\pi'' = \langle X, Y'', V'', entry'', exit'', \to'', B'', \lambda_0'' \rangle$, имеющих одно и то же множество входных переменных X и непересекающиеся множества локальных переменных Y', Y'', $Y' \cap Y'' = \varnothing$. Граф логически согласованных трасс этих программ $\Gamma[\pi', \pi''] = \langle V' \times V'', w_0, \Rightarrow, \lambda_0 \rangle$ устроен следующим образом Вершинами графа являются всевозможные пары $w = (v', v'')$ точек программ π' и π'', и каждой такой вершине приписана пара атомарных формул $(B'(v'), B''(v''))$. В графе $\Gamma[\pi', \pi'']$ особо выделена корневая вершина $w_0 = (entry', entry'')$. Дуги (переходы) графа $\Gamma[\pi', \pi'']$ определяются отношением переходов

$$\Rightarrow \subseteq V' \times V'' \times Subst(Y' \cup Y'', F, Y' \cup Y'') \times V' \times V''$$

Как и для программ, вместо записи $(v', v'', \theta, u', u'') \in \Rightarrow$ мы будем использовать более естественную запись отношения переходов $(v', v'') \overset{\theta}{\Rightarrow} (u', u'')$. Это отношение подчиняется следующему требованию:

$$(v', v'') \overset{\theta}{\Rightarrow} (u', u'') \Leftrightarrow \exists \Delta \in \{0,1\} : v' \xrightarrow{\Delta, \theta'} v'' \wedge v'' \xrightarrow{\Delta, \theta''} u'' \wedge \theta = \theta' \cup \theta''$$

для каждой пары точек (v', v'') и (u', u'') графа $\Gamma[\pi', \pi'']$ и подстановки $\theta, \theta \in Subst(Y' \cup Y'', F, Y' \cup Y'')$. И, наконец, $\lambda_0 = \lambda_0' \cup \lambda_0''$ – это инициализирующая подстановка графа логически согласованных трасс.

Маршрутом в графе логически согласованных трасс $\Gamma[\pi',\pi'']$ будем называть всякую последовательность дуг

$$path = (v'_0, v''_0) \overset{\theta_0}{\Rightarrow} (v'_1, v''_1) \overset{\theta_1}{\Rightarrow} (v'_2, v''_2) \overset{\theta_{21}}{\Rightarrow} \cdots \overset{\theta_N}{\Rightarrow} (v'_{N+1}, v''_{N+1}),$$

начинающуюся в корневой вершине $w_0 = (entry', entry'')$. Как и в случае программ, выражение $\theta[path]$ будет обозначать композицию $\theta_N \cdots \theta_2\theta_1\theta_0\lambda_0$, состоящую из инициализирующей подстановки λ_0 графа логически согласованных трасс и всех подстановок, приписанных дугам этого пути.

Из теоремы 1 и описания устройства графа логически согласованных трасс $\Gamma[\pi',\pi'']$ вытекает следующий критерий л.-т. эквивалентности программ π' и π''.

Теорема 2. Редуцированные программы π' и π'' л.-т. эквивалентны тогда и только тогда, когда для любого конечного маршрута $path$, ведущем в графе $\Gamma[\pi',\pi'']$ из корневой вершины $w_0 = (entry', entry'')$ в вершину $w = (v', v'')$, выполняется равенство $B'(v'_{N+1})\theta[path] = B''(v''_{N+1})\theta[path]$.

Поскольку в графе $\Gamma[\pi',\pi'']$ могут существовать вершины, в которые ведет бесконечно много маршрутов, условия проверки л.-т. эквивалентности программ, представленные в теореме 2, нуждаются в дальнейшем упрощении. Для этого воспользуемся следующей леммой.

Лемма 1. Для любой пары атомарных формул A_1, A_2 и для любой пары подстановок θ', θ'' справедливо соотношение

$$A_1\theta' = A_2\theta' \wedge A_1\theta'' = A_2\theta'' \quad \Leftrightarrow \quad A_1(\theta' \downarrow \theta'') = A_2(\theta' \downarrow \theta'')$$

Доказательство. ($\Rightarrow$) Если $A_1\theta' = A_2\theta'$ и $A_1\theta'' = A_2\theta''$, то это означает, что атомы A_1, A_2 унифицируемы и имеют наиболее общий унификатор μ, для которого верны неравенства $\mu \prec \theta'$ и $\mu \prec \theta''$. Операция антиунификации вычисляет точную нижнюю грань $\theta' \downarrow \theta''$ подстановок θ', θ'', и поэтому $\mu \prec \theta' \downarrow \theta''$. Следовательно, подстановка $\theta' \downarrow \theta''$

унифицирует атомы A_1, A_2, т.е. верно равенство $A_1(\theta' \downarrow \theta'') = A_2(\theta' \downarrow \theta'')$.

($\Leftarrow$) Очевидно в силу определения точной нижней грани подстановок. □

Для заданного графа $\Gamma[\pi',\pi'']$ логически согласованных трасс в программах π' и π'' мы будем использовать запись $PathSet(w)$ для обозначения множества всех маршрутов, ведущих из корня графа в вершину w. Тогда из теоремы 2 и леммы 1 следует

Теорема 3. Редуцированные программы π' и π'' л.-т. эквивалентны тогда и только тогда, когда для любой вершины $w = (v', v'')$ в графе $\Gamma[\pi',\pi'']$, выполняется равенство $B'(v')\theta[w] = B''(v'')\theta[w]$, где $\theta[w] = \downarrow_{path \in PathSet(w)} \theta[path]$.

Таким образом, задача проверки л.-т. эквивалентности программ сводится к задаче вычисления для каждой вершины w графа $\Gamma[\pi',\pi'']$ точной нижней грани множества всех подстановок, вычислимых на маршрутах, ведущих в вершину w. Для решения этой задачи предлагается следующая процедура глобальной разметки графа логически согласованных трасс.

5.1. Процедура глобальной разметки графа $\Gamma[\pi',\pi'']$

В этой процедуре глобальной разметки графа $\Gamma[\pi',\pi'']$ каждой вершине w этого графа приписывается подстановка η_w, которая приближает сверху искомую точную нижнюю грань $\theta[w]$; это приближение уточняется по ходу работы алгоритма. В начале работы процедуры корневой вершине $w_0 = (entry', entry'')$ приписывается подстановка $\eta_{w_0} = \lambda_0$, а всем остальным вершинам $w, w \neq w_0$, приписывается наибольшая в решетке подстановок мнимая подстановка τ. Далее, до тех пор пока это возможно, применяется следующее правило переписывания подстановок η_w, которыми помечены вершины графа:

если в графе $\Gamma[\pi',\pi'']$ существует дуга $u \overset{\theta}{\Rightarrow} w$, для которой не выполняется неравенство $\eta_w \prec \theta\eta_u$, то вершине w вместо подстановки η_w приписывается подстановка $\eta'_w = \eta_w \downarrow \theta\eta_u$.

Процедура переписывания завершает работу в том случае, если для всех дуг $u \overset{\theta}{\Rightarrow} w$ графа $\Gamma[\pi',\pi'']$ выполняется неравенство $\eta_w \prec \theta\eta_u$.

Теорема 4. Каковы бы ни были программы π' и π'' процедура глобальной разметки графа $\Gamma[\pi',\pi'']$ логически согласованных трасс в программах π' и π'' завершает свою работу и вычисляет в каждой вершине w подстановку $\eta_w = \theta[w] = \downarrow_{path \in PathSet(w)} \theta[path]$.

Доказательство. Свойство обрыва убывающих цепей в решетке подстановок гарантирует завершаемость работы процедуры переписывания. Воспользовавшись индукцией по числу шагов процедуры и применяя свойство левой дистрибутивности композиции подстановок относительно операции антиунификации $\eta(\theta_1 \downarrow \theta_2) = (\eta\theta_1) \downarrow (\eta\theta_2)$, можно показать, что на каждом шаге работы процедуры неравенство $\downarrow_{path \in PathSet(w)} \theta[path] \prec \eta_w$ выполняется для каждой вершины w графа $\Gamma[\pi',\pi'']$. Воспользовавшись индукцией по длине маршрута и применяя свойство монотонности композиции подстановок $\theta_1 \prec \theta_2 \;\Rightarrow\; \eta\theta_1 \prec \eta\theta_2$, можно показать, что для каждой вершины w графа $\Gamma[\pi',\pi'']$ и для каждого маршрута $path$, ведущего в вершину w, по окончании работы процедуры переписывания выполняется неравенство $\eta_w \prec \theta[path]$. Это означает, что вычисленная в конце работы процедуры глобальной разметки подстановка η_w удовлетворяет равенству $\eta_w = \downarrow_{path \in PathSet(w)} \theta[path]$. □

Таким образом, для проверки л.-т. эквивалентности программ π' и π'' достаточно построить граф $\Gamma[\pi',\pi'']$ логически согласованных трасс в программах π' и π'', применить описанную выше процедуру глобальной разметки и затем проверить для каждой вершины $w = (v',v'')$ в графе $\Gamma[\pi',\pi'']$ выполнимость равенства $B'(v')\eta_w = B''(v'')\eta_w$, где η_w – это

подстановка, вычисленная процедурой глобальной разметки для вершины w. Теоремы 2-4 гарантируют завершаемость и корректность предложенного алгоритма проверки л.-т.

Заметим также, что процедура глобальной разметки графа $\Gamma[\pi',\pi'']$, применяющая алгоритм антиунификации, вычисляет не только точные нижние грани подстановок $\theta[w] = \downarrow_{path \in PathSet(w)} \theta[path]$ для каждой вершины w, но также и конечные множества пар аннотирующих подстановок для вспомогательных переменных. Эти аннотирующие подстановки будут играть ключевую роль в решении задачи унификации программ.

6. Логико-термальная унификация программ

Условимся, что для каждой программы $\pi = \langle X, Y, V, entry, exit, \rightarrow, B, \lambda_0 \rangle$ и подстановки η запись $\pi\eta$ будет обозначать программу $\pi = \langle X, Y, V, entry, exit, \rightarrow, B, \theta_0\eta \rangle$, полученную из исходной программы заменой инициализирующей подстановки θ_0 композицией подстановок $\theta_0\eta$. Программа $\pi\eta$ называется примером программы π. Подстановка η называется логико-термальным унификатором пары программ π' и π'', если примеры $\pi'\eta$ и $\pi''\eta$ этих программ л.-т. эквивалентны. Унификатор η программ π' и π'' считается наиболее общим л.-т. унификатором, если для любого л.-т. унификатора μ этих программ выполняется неравенство $\eta \prec \mu$. Задача л.-т. унификации программ состоит в том, чтобы для любой заданной пары программ вычислить их наиболее общий л.-т. унификатор.

Задачу л.-т. унификации программ можно свести к задаче унификации множества пар атомарных формул, но это множество в общем случае может содержать бесконечно много пар атомов. Предположим, что имеются две программы $\pi' = \langle X', Y', V', entry', exit', \rightarrow', B', \lambda'_0 \rangle$ и $\pi'' = \langle X'', Y'', V'', entry'', exit'', \rightarrow'', B'', \lambda''_0 \rangle$, у которых множества входных переменных X', X'' и локальных переменных Y', Y'' попарно не пересекаются. Рассмотрим граф $\Gamma[\pi',\pi'']$ логически согласованных трасс в программах π' и π''. Как и прежде, запись $PathSet(w)$ обозначает множество всех маршрутов, ведущих из корня графа в вершину w.

Теорема 5. Подстановка η, $\eta \in Subst(X' \cup X'', F, X' \cup X'')$, является л.-т. унификатором программ π' и π'' тогда и только тогда, когда η является унификатором множества пар атомарных формул

$\{(B'(v')\theta[path],\ B''(v'')\theta[path]) \quad : \quad (v', v'') \in V' \times V'',\ path \in PathSet((v', v''))\}$.

Справедливость теоремы 5 следует из теоремы 3 и определения л.-т. унификатора программ. Таким образом, для унификации программ π' и π'' достаточно разработать способ унификации бесконечного множества пар атомарных формул. Решить эту задачу при помощи одного лишь алгоритма унификации, описанного в разделе 2.1, невозможно. Для ее решения мы воспользуемся комбинацией двух алгоритмов – процедурой глобальной разметки графа логически согласованных трасс и алгоритмом унификации конечного множества пар атомарных формул. Поочередно проводя глобальную разметку графа логически согласованных трасс и унификацию конечного числа пар атомарных формул, описанная ниже процедура л.-т. унификации программ формирует конечную последовательность подстановок $\eta_1, \eta_2, \ldots, \eta_N$, приближающих снизу л.-т. унификатор заданной пары программ η_N, который вычисляется на последнем этапе работы процедуры.

6.1. Процедура л.-т. унификации программ.

В начале первого этапа работы процедуры полагаем $\eta_1 = \{x'_1 / x'_1, \ldots, x'_n / x'_n, x''_1 / x''_1, \ldots, x''_m / x''_m\}$ (тождественная подстановка)

На каждом i-ом этапе работы процедуры строится граф $\Gamma[\pi'\eta_i, \pi''\eta_i]$ логически согласованных трасс в примерах $\pi'\eta_i, \pi''\eta_i$ программ π' и π''.

Затем к графу $\Gamma[\pi'\eta_i, \pi''\eta_i]$ применяется процедура глобальной разметки, которая вычисляет для каждой вершины w подстановку $\theta[w], \theta[w] \in Subst(Y' \cup Y'', F, X' \cup X'' \cup Z)$, и конечное множество пар аннотирующих подстановок $\{(\{z / s'\}, \{z / s''\}) \ : z \in Z\}$ для вспомогательных переменных, возникающих по ходу применения операции антиунификации. В графе $\Gamma[\pi'\eta_i, \pi''\eta_i]$ выделяем множество вершин $W_i = \{w \quad : \quad w = (v', v'') \wedge B'(v')\theta[w] \neq B''(v'')\theta[w]\}$, в которых

нарушается условие л.-т. эквивалентности примеров программ $\pi'\eta_i, \pi''\eta_i$, описанное в теореме 3.

Если $W_i = \varnothing$, то согласно теореме 3 примеры программ $\pi'\eta_i, \pi''\eta_i$ являются л.-т. эквивалентными, и подстановка η_i объявляется результатом работы процедуры л.-т. унификации программ π' и π''.

Если $W_i \neq \varnothing$, то к конечному множеству пар атомарных формул применяется модифицированный алгоритм унификации. На вход этого алгоритма поступает система уравнений

$$\{\ B'(v')\theta[w] = B''(v'')\theta[w] \quad : \quad w = (v', v'') \in W_i\}.$$

К этой системе, до тех пор пока это возможно, применяются 7 правил переписывания уравнений, заимствованных из алгоритма унификации. Первые 5 правил в точности совпадают с правилами 1-5 из раздела 2.1., в котором описан алгоритм выполнения операции унификации подстановок. Последние два правила представляют собой следующие модификации правила 6 из раздела 2.1:

6.1. Уравнение $x = t$ в том случае, если x – это входная переменная из множества $X' \cup X''$, t – это терм, не содержащий x, и переменная x содержится еще хотя бы в одном уравнении системы, требует применения подстановки $\{x / t\}$ ко всем остальным уравнениям системы (их левым и правым частям);

6.2. Уравнение $z = t$ в том случае, если z – это вспомогательная переменная из множества Z, снабженная парой аннотирующих подстановок $\{z / s'\}$ и $\{z / s''\}$, а t – это терм, не содержащий z, требует замены каждого уравнения системы $t' = t''$ (включая само уравнение $z = t$) парой уравнений $t'\{z / s'\} = t''\{z / s'\}$ и $t'\{z / s''\} = t''\{z / s''\}$.

Если ни одно из правил 2 или 5 «аварийного» завершения работы алгоритма унификации не применяется, то модифицированный алгоритм унификации прекращает свое выполнение, как только будет построена система уравнений, к которой неприменимо ни одно из перечисленных правил 1-5, 6.1, 6.2. Образованная в этом случае система уравнений состоит только из уравнений $x_1 = t_1, \ldots, x_k = t_k$, в которых все переменные $x_1, \ldots, x_k$ – это попарно различные входные переменные из множества $X' \cup X''$, не содержащиеся в

термах $t_1,\ldots,t_k$. Тогда результатом работы i-ого этапа процедуры л.-т. унификации объявляется подстановка $\eta_{i+1} = \eta_i\{x_1/t_1,\ldots,x_k/t_k\}$.

Теорема 6. Каковы бы ни были программы π' и π'', описанная выше процедура унификации программ завершает свое выполнение. Подстановка η_N, вычисленная этой процедурой, является наиболее общим унификатором программ π' и π''.

Доказательство. Чтобы убедиться в завершаемости описанной процедуры, достаточно заметить, что для последовательности подстановок $\eta_1,\eta_2,\ldots,\eta_N$, вычисленных на каждом этапе ее работы, имеет место строгое включение $Var_{\eta_{i+1}} \subset Var_{\eta_i}$. Следовательно, процедура не может иметь более $|X'|+|X''|$ этапов выполнения.

Чтобы обосновать корректность процедуры, достаточно доказать, используя индукцию по числу этапов, справедливость следующих двух утверждений: 1) последовательность $\eta_1,\eta_2,\ldots,\eta_N$ является строго возрастающей последовательностью в решетке подстановок, и 2) для любого члена η_i этой последовательности и для любого л.-т. унификатора ρ программ π' и π'' выполняется неравенство $\eta_i \prec \rho$. Справедливость первого утверждения следует из определения последовательности подстановок $\eta_1,\eta_2,\ldots,\eta_N$. Для доказательства последнего из этих двух утверждений следует воспользоваться теоремой 5 и заметить, что согласно описанию процедуры л.-т. унификации каждая из подстановок η_{i+1} является наиболее общим унификатором непустого множества пар атомарных формул

$$\{(B'(v')\theta[path],\, B''(v'')\theta[path]) \;:\; (v',v'')\in W_i,\; path\in R\}$$

для некоторого подмножества маршрутов $R \subseteq PathSet((v',v''))$ в графе логически согласованных трасс в программах π' и π''.

Предложенный нами алгоритм л.-т. унификации программ можно применять для решения задачи процедурной абстракции. Предположим, что унификатором пары программ $\pi' = \langle X',Y',V',entry',exit',\to',B',\lambda_0'\rangle$ и $\pi'' = \langle X'',Y'',V'',entry'',exit'',\to'',B'',\lambda_0''\rangle$ является подстановка

$\eta_1 = \{x_1''/t_1,\ldots,x_m''/t_m\}$, в которой все термы $t_1,\ldots,t_m$ зависят только от переменных из множества X' (от входных параметров программы π'). Тогда программа π'' является примером программы π'. В этом случае программу π' можно объявить отдельной процедурой, а все вхождения программы π'' можно заменить вызовом этой процедуры $call\ \pi'(t_1,\ldots,t_m)$ со списком параметров $t_1,\ldots,t_m$.

7. Заключение

Основной результат этой статьи – это решение двух задач, возникающих при анализе последовательных императивных программ, – задачи проверки эквивалентности программ и задачи унификации программ. Для решения обеих задач был задействован математический аппарат алгебры подстановок с двумя основными операциями – операцией взятия точной верхней грани (унификация) и точной нижней грани (антиунификация) двух подстановок. На основе этих операций были предложены алгоритм проверки логико-термальной эквивалентности и алгоритм логико-термальной унификации программ. Эти алгоритмы можно использовать для решения многих задач семантического анализа и эквивалентных преобразований программ, включая задачи оптимизации, верификации и реорганизации программ,

Вместе с тем, практическое применение предложенных нами алгоритмов требует решения целого ряда вопросов, касающихся сложности рассмотренных задач. Вот лишь некоторые из них.

1. Оба операции над подстановками – унификации и антиунификация, – могут быть выполнены за время, пропорциональное размерам исходных данных. Однако наибольшая эффективность выполнения этих операций достигается на разных формах представления подстановок: для эффективного выполнения операции унификации желательно, чтобы термы были представлены в виде ориентированных размеченных ациклических графов, в то время как операция антиунифкации наилучшим образом приспособлена для работы с древесным представлением термов. Поскольку в алгоритме л.-т. унификации программ задействованы обе операции, необходимо отыскать такую форму представления термов, которая была бы оптимальной для обеих операций.
2. Для проверки л.-т. эквивалентности программ достаточно уметь вычислять любую (не обязательно точную) нижнюю грань двух подстановок при условии, что операция вычисления этой нижней грани будет обладать свойством левой дистрибутивности

относительно операции композиции подстановок, и для нее будет справедлива лемма 1. В связи с этим целесообразно изучить вопрос о выборе оптимальной по сложности вычисления операции взятия какой-либо нижней грани двух подстановок, которую можно использовать в процедуре глобальной разметки графа логически согласованных трасс для проверки л.-т. эквивалентности программ.

3. Предложенный нами алгоритм л.-т. унификации программ – это многопроходная процедура разметки графов логически согласованных трасс программ. Поэтому одно из направлений повышения эффективности алгоритма л.-т. унификации программ состоит в сокращении числа проходов этой процедуры.

Отметим также, что алгоритмическое решение задач проверки логико-термальной эквивалентности программ и выполнения логико-термальной унификации программ создает хорошие предпосылки для решения задачи, имеющей более важное прикладное значение, – вычисление наиболее специального шаблона двух программ (задачи логико-термальной антиунификации программ).

Список литературы

[1] Lague B., Proulx D., Mayrand J. et al. Assessing the benefits of incorporating function clone detection in a development process // Proceedings of the International Conference on Software Maintenance. — Washington, DC, USA: IEEE Computer Society, 1997.— p. 314–321.

[2] Baker B. S. On finding duplication and near-duplication in large software systems // Proceedings of the Second Working Conference on Reverse Engineering. — Washington, DC, USA: IEEE Computer Society, 1995. — p. 86–95.

[3] Kontogiannis K. A., Demori R., Merlo E. et al. Pattern matching for clone and concept detection // Journal of Automated Software Engineering.— 1996.— Vol. 3.— p. 108 -125.

[4] Li Z., Lu S., Myagmar S., Zhou Y.. Cp-miner: Finding copy-paste and related bugs in large-scale software code // IEEE Transactions on Software Engineering. — 2006. — v. 32, N 3.— p. 176–192.

[5] Johnson J. H. Identifying redundancy in source code using fingerprints // Proceedings of the Conference Centre for Advanced Studies on Collaborative research (CASCON). — IBM Press, 1993.— p. 171–183.

[6] Фаулер М. Рефакторинг. Улучшение существующего кода. — Символ-Плюс, 2008. — 432 с.

[7] 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.

[8] 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.

[9] Robinson J.A. A machine-oriented logic based on the resolution principle // Journal of the ACM. — 1965. — v. 12, N 1 — p. 23_41.

[10] Baader F., Snyder W. Unification theory. — Handbook of Automated Reasoning, v. I, Elsevier Science Publishers, 2001. — p. 447–533.

[11] Yelick K.A. Generalized approach to equational unification // Technical Report, Massachusetts Institute of Technology Cambridge, MA, USA, 1990.

[12] 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.

[13] Itkin V.E., Zwinogrodski Z. On program schemata equivalence // Journal of Computer and System Science. — 1972. — v. 6, N 1. — p. 88-101.

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

[15] Sabelfeld V.K. The logic-termal equivalence is polynomial-time decidable // Information Processing Letters. — 1980 — v. 10, N 2. — p. 57-62.

[16] Eder E. Properties of substitutions and unifications // Journal of Symbolic Computations. — v. 1. — 1985. — p. 31-46.

[17] Palamidessi C. Algebraic properties of idempotent substitutions // Lecture Notes in Computer Science — v. 443 — 1990. — p. 386-399.

[18] Martelli A., Montanari U. An Efficient Unification Algorithm // Transactions on Programming Languages and Systems (TOPLAS). — 1982. — v.4, N 2. — p. 258-282.

[19] Plotkin G.D. A note on inductive generalization // Machine Intelligence. — 1970. — v. 5, N 1. — p. 153-163.

[20] Sorensen M. H., Gluck. R. An algorithm of generalization in positive supercompilation // Proceedings of the 1995 International Symposium on Logic Programming._MIT Press. — 1995. — p. 465-479.

[21] Bulychev P., Kostylev E., Zakharov V. Anti-unification algorithms and their applications in program analysis // Proceedings of the 7th International Conference "Perspectives of System Informatics", June 15-19, 2009, Novosibirsk. — 2009. — p. 82-89.

[22] Захаров В.А., Костылев Е.В. О сложности задачи антиунификации // Дискретная математика. — 2008. — т. 20, N 1. — с. 131-144.

[23] Zakharov V.A. On the refinement of logic programs by means of anti-unification // Proceedings of the 2nd Panhellenic Logic Symposium, Delphi, Greece. — 1999. — p. 219-224.

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