О РЕАЛИЗАЦИИ ВЫЧИСЛЕНИЙ В ЗАДАЧАХ АНАЛИЗА ПРОГРАММ, ОПРЕДЕЛЕННЫХ НАД ВЕКТОРНЫМИ ПРОСТРАНСТВАМИ

С.М. Львов

Інститут кібернетики ім. В.М. Глушкова НАНУ, відділ аспірантури,
Проспект Академіка Глушкова, 40, 03680, МСП Київ - 187, тел.: (044) 266 20 08, 266 02 08, факс (044) 266 74 18,
email: askserg@pisem.net

Анотація

В даній роботі розглянуті основні вимоги користувача до програмної системи, яка автоматично генерує програмні інваріанти для визначеного класу програм. Аналізуються також методи реалізації основних алгоритмів такої системи.

Аннотация

В настоящей работе рассмотрены основные пользовательские требования к программной системе, автоматически генерирующей программные инварианты для определенного класса программ. Анализируются также методы реализации основных алгоритмов такой системы.

Annotation

In the present work the main user requirements to the program system are viewed, which automatically generates program invariants for certain program class. Also methods of basic algorithms realization of such system are analyzed.

Ключевые слова.

Анализ программ, инварианты программ.

Обоснование

Методы потокового анализа последовательных программ в настоящее время достаточно хорошо изучены. Сейчас активно исследуются методы потокового анализа программ, основанные на сетях Петри как модели управления вычислениями и методах анализа последовательных программ, основанных на свойствах алгебр данных.

Отметим, однако, что эти исследования носят в основном теоретический характер, в то время как проблемам реализации и вычислительного эксперимента уделяется недостаточно внимания. Очевидно, что реализация и проведение вычислительного эксперимента для задач анализа асинхронных дискретных динамических систем будет способствовать как теоретическим исследованиям, так и разработке производственных систем автоматизации проектирования многопоточных приложений.

В работах [1-3] были развиты теоретические методы решения задачи поиска инвириантных соотношений в программах, определенных над многими классическими алгебрами данных. Вместе с этим экспериментальные исследования практической пригодности этих методов по существу отсутствуют. Поэтому задача построения программной системы, в которой эти методы были бы реализованы, является актуальной. Цель такого экпериментального исследования – поиск эффективных реализаций алгоритмов поиска инвариантов для практически важных алгебр данных. К таким алгебрам относятся, например, векторные пространства [4], многочлены многих переменых [5].

Модели программ и методы построения инвариантов

В [4] описаны алгоритмы поиска линейных инвариантных равенств для программ, алгеброй данных которых являются векторные пространства. Для решения этой задачи использовался метод потокового анализа, называемый методом нижней аппроксимации, а в качестве модели программ - U-Y схемы программ над памятью, интерпретированные над алгебрами даных. Отметим, что эта модель представляет неструктурированную программу.

В [5] описаны алгоритмы поиска полиномиальных инвариантных равенств для программ, алгеброй данных которых являются многочлены многих переменных. В [5] в качестве модели программы по существу использовались алгоритмические алгебры Глушкова, представляющие структурированные программы. Алгоритм вычисления полиномиальных инвариантных равенств основывался на свойстве нетеровости колец полиномов. В работе приведено лишь доказательство его существования.

Ниже мы определим единую теоретическую модель программы и унифицируем изложение метода поиска инвариантных равенств для этих подходов.

Модель программы

В качестве моделей программ нам будет удобно использовать формулы пропозициональной динамической логики (ПДЛ), интерпретированные над областью данных. Логика ПДЛ основывается на соответствии модальных операторов каждой команде некоторого языка программирования. При такой интерпретации множество **W** рассматривается как множество возможных состояний вычислений, а отношение **wRw'** говорит о том, что вычисления начинаются в состоянии **w** и заканчиваются в состоянии **w'**.

Если программа недетерминирована (например, включает операторы распараллеливания), то она может давать разные результаты. Формула □**A** означает в этом случае, что все выполнения программы заканчиваются тем, что формула **A** истинна. Фомула ◊**A** означает, что хотя бы одно выполнение программы заканчивается тем, что истинна формула **A**.

Синтаксис ПДЛ

ПДЛ впервые была введена Фишером и Леднером. ПДЛ играет такую же роль в динамической логике, какую классическое исчисление высказываний в классической логике предикатов. ПДЛ описывает свойства взаимодействия между программами и высказываниями, которые не зависят от области данных вычислений.

Синтаксис ПДЛ включает два сорта выражений:

- программы **p, q, r**, …;
- высказывания (или формулы) $\varphi, \psi, \ldots$.

Если φ, ψ формулы, а **p**, **q** - программы, то

$\varphi \vee \psi, \neg\varphi, < p > \varphi$ - формулы;

p;q, **p|q**, **p***, φ**?** – программы.

Операция **p;q** означает последовательную композицию, **p|q** – недетерминированный выбор, **p*** - итерацию и φ**?** - проверку условия. Интуитивные значения этих конструкций таковы:

- **<** p **>** φ = "возможно выполнение **p** и это выполнение заканчивается в состоянии, в котором истинна формула φ",
- **p;q** = "Выполнить **p**, а потом выполнить **q**",
- **p|q** = "Выбрать недетерминированно **p** или **q** и выполнить его",
- **p*** = "Выполнить p повторяя его недетерминированное число раз",
- φ**?** = "Проверка условия φ, вывести true, если она истинна или false, если она ложна".

Пропозициональные операторы $\wedge, \rightarrow, \Leftrightarrow, 1, 0$ определяются обычным образом. Они дополняются такими операторами:

- **skip** = true?,
- **fail** = false?,
- [p] φ = $\neg < p > \neg\varphi$,
- **if** $\varphi_1 \rightarrow p_1 \| \ldots \| \varphi_n \rightarrow p_n$ **fi** = $\varphi_1?; p_1 | \ldots | \varphi_n?; p_n$
- **do** $\varphi_1 \rightarrow p_1 \| \ldots \| \varphi_n \rightarrow p_n$ **od** = $(\varphi_1?; p_1 | \ldots | \varphi_n?; p_n)^*; (\neg\varphi_1? \wedge \ldots \wedge \neg\varphi_n?)$
- **if** φ **then** p **else** q **fi** = **if** $\varphi \rightarrow p \| \neg\varphi \rightarrow q$ **fi** = $\varphi?; p | \neg\varphi?; q$
- **while** φ **do** p **od** = **do** $\varphi \rightarrow p$ **od** = $(\varphi?; p)^*; \neg\varphi?$
- **repeat** p **until** $\varphi = p$; **while** $\neg\varphi$ **do** p **od** = $p; (\neg\varphi?; p)^*; \varphi?$
- $\{\varphi\}\, p\, \{\psi\} = \varphi \rightarrow [p]\psi;$

где <p> и [p] является модальными операторами. Конструкции **if||fi** и **do||od** являются альтернативными охраняемыми командами (quarded command) и итеративными охраняемыми командами соотвественно. Конструкция $\{\varphi\}\, p\, \{\psi\}$ - это класическое хоаровское утверждение частичной корректности.

В исходных для нас работах [3, 4] при построении множеств инвариантов условия в операторах управления игнорируются. Это означает, что формула ПДЛ, описывающая программу, редуцируется до формулы в сигнатуре **p;q**, **p|q**, **p*,** т.е. игнорируется операция φ**?**. Таким образом, система автоматической генерации инвариантов должна:

- получать на входе исходный текст программы в структурированных операторах управления **if .. then .., if .. then .. else .., while .. do .., repeat..until .. ;**
- трансформировать этот текст в формулу ПДЛ в сигнатуре **p;q**, **p|q**, **p*,** одновременно формируя словарь вычисляющих операторов;

- используя формулу программы и словарь вычисляющих операторов, генерировать множество инвариантов этой программы.

Переписывающие правила процедуры редуцирования имеют вид:

if ϕ then p = p | I, // **I –** символ тождественного вычисляющего оператора

if ϕ then p else q = p | q, (1)

while ϕ do p = p*,

repeat p until ϕ = p; p*.

Таким образом, семантика программы определяется семантикой вычисляющих операторов, т.е. интерпретацией программы над некоторой областью данных. Понятие интерпретации программы над областью данных мы сейчас уточним.

Программы, интерпретированные над алгебрами данных

Пусть ***A*** – (многосортная) алгебра с (главной) сигнатурой операций $\mathbf{\Sigma} = \langle \sigma_1, \ldots, \sigma_k \rangle$ и (главным) носителем **A**. Пусть, далее, $\mathbf{X} = \langle x_1, \ldots, x_n \rangle$ - вектор переменных, которые мы интерпретируем как переменные программы. Для краткости мы будем называть **X** памятью программы, а векторы $\mathbf{a} = \langle a_1, \ldots, a_n \rangle$, $a_i \in \mathbf{A}$ - состояниями памяти. Множество $\mathbf{U} = \mathbf{A}^n$ образует пространство - универсум состояний памяти программы. Вычисляющий оператор – это отображение **f : U → U**, определенное покоординатно системой присвоений

$$x_1 := f_1(\mathbf{X}), \ldots, x_n := f_n(\mathbf{X}) \quad (2)$$

Вычисляющий оператор мы часто будем записывать в векторной форме **X := f(X)**, а вектор координатных функций (вычислений) – в форме $\mathbf{f} = \langle f_1, \ldots, f_n \rangle$. Отметим, что в таких обозначениях вычисляющий оператор интерпретируется как одновременное присвоение вектору переменных значений – правых частей покоординатных операторов присвоений.

Мы будем говорить, что программа **p** интерпретирована над векторным пространством, если ***A*** – некоторое поле (тело, область целостности), а все ее вычисляющие операторы имеют вид

$$\mathbf{X := F*X + b}, \quad (3)$$

где **F** – линейный оператор, действующий в векторном пространстве **U,** а $\mathbf{b} \in \mathbf{U}$.

Мы будем говорить, что программа **p** интерпретирована над полиномиальным кольцом, если ***A*** – некоторое поле (тело, область целостности), а все ее вычисляющие операторы имеют вид

$$\mathbf{X := F(X)}, \quad (4)$$

где $f_i \in \boldsymbol{A}\,\mathbf{[X]}$, т.е. многочлены с коэффициентами из ***A.***

Таким образом, программы, интерпретированные над линейными пространствами, на каждом шаге вычислений осуществляют линейные преобразования универсума состояний памяти, а программы, интерпретированные над полиномиальными кольцами – полиномиальные преобразования универсума состояний памяти.

Инварианты программ

Многочлен $\mathbf{i(X) \in F[X]}$ называется полиномиальным инвариантом программы **p,** если $\Box$ **{True} p {i(X)=0}.**

Это означает, что при любом начальном состоянии памяти $\mathbf{a} = \langle a_1, \ldots, a_n \rangle$, если программа **p** завершает работу, для заключительного состояния **b** (т.е. такого, что **a{p}b)** выполняется равенство **i(b) = 0.**

Легко видеть, что множество всех полиномиальных инвариантов программ, интерпретированных над полиномиальным кольцом **A[X]**, образует идеал этого кольца, который мы будем обозначать через $\mathbf{I_p}$. В дальнейшем мы будем рассматривать множество всех элементов $\mathbf{I_p}$, степени которых ограничены некоторой константой M. Множество таких многочленов образует векторное пространство, которое обозначается через $I_p^{(M)}$. Понятно, что $I_p^{(M)} \subset I_p$.

Теоретические результаты

Основной результат [М. Львов], на котором основываются алгоритмы анастоящей работы, состоит в следущем:

Теорема 1. Пусть **p** - программа, интерпретированная над полиномиальным кольцом **A[X]**, и $I_p^{(M)} = \{i \mid i \in I_p, \deg\ i \le M\}$.

Тогда:

а) Проблема принадлежности $i \in I_p^{(M)}$ алгоритмически разрешима.

б) Проблема построения базиса векторного пространства $I_p^{(M)}$ алгоритмически разрешима.

Доказательство. Пусть $y_1,...,y_m$ все вычисляющие операторы, встречающиеся в программе p, и $y_i = (X := F_i(X))$.

Доказательство утверждения а) будем вести индукцией по структуре редуцированной программы p (1), т.е. по структуре ее ПДЛ формулы в сигнатуре **p;q**, **p|q**, **p*,** в которой вычисляющие операторы – суть атомы $y_1,...,y_m$.

Пусть **w = $y_{i1};y_{i2};\ldots;y_{ik}$** – ПДЛ-программа, представляющая собою последовательность вычисляющих операторов. Мы будем говорить, что **w ∈ p**, если **p** допускает вычисления, описанные **w**. Мы будем также называть **w** словом **p**. Собственно программа **p** ассоциируется тогда со множеством слов, принадлежащих ей (множество слов – это множество всевозможных путей вычислений **p**).

Введём следующие обозначения. Через i(w), $i \in A[X]$, обозначим многочлен $i(F_k(F_{k-1}(...(F_1(X))...))$, а через $i(p)$ - множество многочленов вида i(w), $w \in p$. Идеал, порожденный множеством $i(p)$, обозначим через $(i(p))$.

Мы покажем, что для каждой программы p существует и может быть эффективно построено конечное подмножество слов $W = W(i,p)$ такое, что $(i(p)) = (i(W))$. Поскольку **i** - инвариант программы **p** тогда и только тогда, когда **i**(w)=0 для любого $w \in p$, проверка инвариантности **i** сведётся к проверке равенств i(w)=0, $w \in W$.

1. (базис индукции) $p = y_k$. Тогда $W(i,p) = \{y_k\}$.

2. (шаг индукции)

а). $p = p_1 \mid p_2$

$$W(i, p_1 \mid p_2) = W(i,p_1) \cup W(i,p_2) \qquad (5)$$

б) $p = p_1; p_2$

Пусть $W(i,p_2) = \{v_1,...,v_l\}$. Поскольку из $w \in p$ следует $w = w_1;w_2$, *где* $w_i \in p$, имеем $p(w) = p(w_1;w_2) = p(w_2(w_1))$, откуда

$$W(i, p_1; p_2) = W(i(v_1), p_1) \cup ... \cup W(i(v_l), p_1) \qquad (6)$$

в) $p = \{p_1\}$

Тогда $p = I \cup p_1 \cup p_1^2 \cup ...$, где I – обозначение тождественного вычисляющего оператора. Рассмотрим последовательность идеалов:

$$(i) \subseteq (i, i(p_1)) \subseteq ... \subseteq (i, i(p_1)), ..., i(p_1^m)) \subseteq ...$$

Поскольку кольцо A[X] нетерово, эта последовательность стабилизируется на некотором номере m_0. Покажем, что m_0 - наименьшее натуральное число такое, что

$$i(p_1^{m_0+1}) \subseteq (i, i(p_1), ..., i(p_1^{m_0})) \qquad (7)$$

В самом деле, из (7) следует $i(p_1^{m_0+2}) \subseteq (i(p_1), ..., i(p_1^{m_0+1})) \subseteq (i, i(p_1), ..., i(p_1^{m_0}))$

поэтому из равенства $(i,...,i(p_1^{m_0})) = (i,...,i(p_1^{m_0}), i(p_1^{m_0+1}))$ следуют равенства $(i,...,i(p_1^{m_0})) = (i,...,i(p_1^{m_0+k}))$ для любого натурального k, откуда

$$W(i,p) = W(i,e) \cup W(i,p_1) \cup ... \cup W(i,p_1^{m_0}) \qquad (8)$$

Поскольку проблема принадлежности $i \in (q_1,...,q_l)$ алгоритмически разрешима, формулы (6) - (8) описывают рекурсивный алгоритм построения $W(i,p)$. Утверждение а) доказано.

Для доказательства утверждения б) необходимо рассмотреть многочлен $i^*(X) = a_0 X_1^M + ... + a_{\varphi(M)}$ степени M с неопределенными коэффициентами и для него построить множество $W(i^*, p)$. Поскольку кольцо $A[a_0,...,a_{\varphi(M)}, X]$ также нетерово, это можно осуществить с помощью вышеизложенного алгоритма.

Пусть $W(i^*, p) = \{i_1, ..., i_l\}$, $i_k \in A[a_0, ..., a_{\varphi(M)}, X]$. Система равенств $i_1 \equiv i_2 \equiv ... \equiv i_l \equiv 0$ определяет систему линейных уравнений над F с неизвестными $a_0, ..., a_{\varphi(M)}$, фундаментальная система решений которой задаёт искомый базис A_S^M .

Основные задачи и алгоритмы

Из доказательства теоремы 1 а) видно, что основной алгоритмической проблемой, которую нужно решить при реализации соответствующего алгоритма, является проблема

$$i(p_1^{m+1}) \subseteq (i, i(p_1), ..., i(p_1^m))$$

Иными словами, задача сводится к тому, чтобы распознать принадлежность некоторого многочлена полиномиальному идеалу, заданному своим базисом. Эта задача является классической задачей теории полиномиальных идеалов, и для ее решения используется построение так называемого базиса Гребнера полиномиального идеала.

Таким образом, все идеалы, которые строятся в алгоритме теоремы 1а), должны быть представлены своими базисами Гребнера. В дальнейшем мы будем обозначать тот факт, что идеал **I** представлен базисом Гребнера $\mathbf{f_1, \ldots, f_k}$ равенством $\mathbf{I = (f_1, \ldots, f_k)_{Gr}}$.

Базис Гребнера идеала **I** мы будем обозначать через **Gr(I)**.

Наш алгоритм теоремы 1а) должен решать следующие задачи:

1. Для $\mathbf{I = (f_1, \ldots, f_k)_{Gr}}$, $\mathbf{J = (g_1, \ldots, g_l)_{Gr}}$ найти $\mathbf{I \cap J = (h_1, \ldots, h_m)_{Gr}}$

2. Для $\mathbf{I = (f_1, \ldots, f_k)_{Gr}}$, вычисляющего оператора **X := H(X)**, найти $\mathbf{J = (g_1, \ldots, g_l)_{Gr}}$, т.е. найти $\mathbf{Gr((f_1(H(X)), \ldots, f_k(H(X))))}$.

3. Для $\mathbf{I = (f_1, \ldots, f_k)_{Gr}}$, $\mathbf{J = (g_1, \ldots, g_l)_{Gr}}$ распознать $\mathbf{I \subseteq J}$.

Задачи 1 и 3 – классические задачи теории полиномиальных идеалов, решение которых в терминах базисов Гребнера хорошо известно. Для решения задачи 2 можно воспользоваться общим алгоритмом пополнения, который строит базис Гребнера по произвольному конечному множеству полиномов.

Рассмотрим теперь вопросы, связанные с реализацией алгоритма теоремы 1б). Отметим, что из метода доказательства следует, что искать инварианты можно, задавая их в виде полиномиальных форм, т.е. многочленов «специального» вида с неопределенными коэффициентами. Например, можно определить общий вид искомых инвариантов как линейную комбинацию нескольких многочленов с неопределенными коэффициентами:

$$\mathbf{I(X) = a_1 * P_1(X) + \ldots + a_k * P_k(X)} \qquad \mathbf{(9)}$$

Если, например, требуется искать линейные инварианты, следует положить

$$\mathbf{P_1 = x_1, \ldots, P_n = x_n, P_{n+1} = 1.} \qquad \mathbf{(10)}$$

Можно, к примеру, искать все инварианты вида $\mathbf{I(X) = a_1 * x_1^2 + \ldots + a_n * x_n^2}$ и т.п.

Поскольку вычисляющие операторы оставляют неопределенные коэффициенты неподвижными, алгоритмы теоремы 1а) следует применять поккординатно к вектору многочленов ($\mathbf{P_1(X), \ldots P_k(X)}$), т.е. строить базисы Гребнера отдельно для каждого из многочленов $\mathbf{P_1(X), \ldots P_k(X)}$, а условие обрыва возрастающих цепочек п. в) алгоритма теоремы 1а) применять ко всему вектору идеалов $\mathbf{(I_1, \ldots, I_n)}$. Таким образом, построение конечного множества слов из символов $\mathbf{y_i}$ вычисляющих операторов при анализе программы $\mathbf{p = p_1^*}$ заканчивается при таком значении **m,** при котором стабилизируются все координаты вектора идеалов $\mathbf{(I_1, \ldots, I_n)}$.

Задача вычисления линейных инвариантов

Применим теперь эти рассуждения к задаче поиска линейных инвариантов в программых, интерпретированных над векторными пространствами. Кординаты соответствующего вектора идеалов задаются системами линейных комбинаций вектора $\mathbf{X=(x_1, \ldots, x_n, 1)}$ переменных программы с коэффициентами из основного поля.

$$\mathbf{I_p(X) = (i_1(X), \ldots, i_{n+1}(X)),} \quad \mathbf{i_k(X) = B_k * X}$$

Где $\mathbf{B_k}$ – матрица размера **l*(n+1)** с коэффициентами из основного поля, количество строк **l** которой, как мы покажем, зависит от шага вычислений.

На входе алгоритма вычисления линейных инвариантов программы, интерпретированной над векторным пространством, вектор идеалов задается формулой (10). Матрицы $\mathbf{B_k}$ имеют размеры **1*(n+1), $\mathbf{B_k} = \mathbf{e_k}$ =(0, ..., 1, ..., 0)** т.е. координатные векторы.

Метод теоремы 1а) исходит из того, что на каждом шаге вычислений заданы ПДЛ программы **p** и **q** , имеющие простой вид:

$$\mathbf{p = u_1 \mid u_2 \mid . . . \mid u_l}\ , \qquad \mathbf{q = v_1 \mid v_2 \mid . . . \mid v_m}$$

где $u_i = y_{i1};...;y_{ik_i}$, $v_j = y_{j1};...;y_{jk_j}$, т.е. представляют собой последовательности вычисляющих операторов, каждый из которых имеет вид (3).

На каждом шаге вычислений мы будем поддерживать перевычисления результирующих линейных преобразований, осуществляемых этими последовательностями. В самом деле, если дана последовательная программа $\mathbf{u = y_{i1};\ldots;y_{im}}$, $\mathbf{y_{ij} = F_j *X,}$ то результирующее линейное преобразование пространства состояний, осуществляемое этой программой, есть $\mathbf{F_u = F_m*F_{m-1}*\ldots*F_1}$.

1. Если данный шаг вычислений заключается в том, чтобы построить программу данного вида для **r = p | q,** то **r = $\mathbf{u_1}$ | $\mathbf{u_2}$ | . . . | $\mathbf{u_l}$ | $\mathbf{v_1}$ | $\mathbf{v_2}$ | . . . | $\mathbf{v_m}$.** Алгоритм объединяет (сливает) последовательности результирующих матриц программ **p и q,** получая последовательность результирующих матриц программы **r.**

2. Если данный шаг вычислений заключается в том, чтобы построить программу данного вида для **r = p ; q,** то **r = $\mathbf{u_1;v_1}$ | $\mathbf{u_2;v_1}$ | . . . | $\mathbf{u_i;v_j}$ | . . . | $\mathbf{u_l;v_m}$.** Алгоритм вычисляет последовательность результирующих матриц программы **r,** перемножая попарно результирующие матрицы программ **p** и **q**.

3. Пусть, наконец, данный шаг вычислений заключается в том, чтобы построить программу данного вида для **r = $\mathbf{p^*}$** то алгоритм должен найти такое натуральное **m**, при котором $I_{p^8} = I_{p^m}$. Иными словами, **r = e | p | $\mathbf{p^2}$ |...| $\mathbf{p^m}$ | ...** и необходимо найти такой номер **m**, при котором $I_{p^{m+1}} \subseteq I_{p^{e|p|...|p^m}}$ Введем необходимые обозначения:

Пусть $\mathbf{u_1}$ есть $\mathbf{X:=F_1*X}$, . . . , $\mathbf{u_l}$ есть $\mathbf{X := F_l*X}$, где $\mathbf{F_j} = \langle \mathbf{f_{j1}, \ldots, f_{jn}} \rangle$ - линейные операторы результирующих вычисляющих операторов, построенные по словам $\mathbf{u_1, \ldots, u_l}$ и записанные в виде векторов линейных форм.

$$\mathbf{f_{jk}(X) = a_{jk1}*x_1 + \ldots + a_{jkn}*x_n + b_{jk}}$$

Наш алгоритм строит следующую последовательность $\mathbf{P^{(0)}, P^{(1)}, \ldots P^{(j)}, \ldots}$ векторов линейных форм $P_j = (L_1^{(j)},...,L_n^{(j)},L_{n+1}^{(j)})$, каждый член которой описывает совокупность линейных инвыриантных соотношений для $\mathbf{r^{(j)} = e \mid p \mid p^2 \mid \ldots \mid p^j \mid}$:

$$P_0 = (x_1,...,\ x_n,1),$$

$$P_1 = ((x_1,\ f_{11}(X),...,\ f_{l1}(X)),(x_2,\ f_{12}(X),...,\ f_{l2}(X))...(x_n,\ f_{1n}(X),...,\ f_{\ln}(X),1),$$

$$\ldots,$$

$$P_{j+1} = ((L_1^{(j)}(X),L_1^{(j)}(F_1(X)),...,L_1^{(j)}(F_l(X))),...,(L_n^{(j)}(X),L_n^{(j)}(F_1(X)),...,L_n^{(j)}(F_l(X))),1),$$

$$\ldots$$

В реализации каждая совокупность линейных форм $L_k^{(j)}, k = 1...n$ задается матрицей, которую, для простоты, мы будем об означать так же, как и соответствующую совокупность линейных форм. Таким образом, $L_k^{(j)}(X) = L_k^{(j)} * X$.

Для того, чтобы построить $L_k^{(j+1)} = (L_k^{(j)}, L_k^{(j)} * F_1,..., L_k^{(j)} * F_l)$, нужно вычислить все произведения $L_k^{(j)} * F_s,\ s = 1,...,l$ и построить матрицу, строками которой являются все строки матриц $L_k^{(j)}, L_s^{(j)} * F_l, s = 1,...,l$

Условием окончания вычислений являются равенства

$$L_k^{(j+1)} * X = 0 \Rightarrow L_k^{(j)} * X = 0,\ k = 1,...,n$$

Таким образом, на каждом шаге строится матрица $L_k^{(j+1)}$ и вычисляется ее ранг. Если ранг $L_k^{(j+1)}$ совпадает с рангом $L_k^{(j)}$, по данной координате стабилизация достигнута. Алгоритм заканчивает свою работу, когда достигнута стабилизация по всем координатным матрицам $L_k^{(j+1)}, k = 1,...,n$.

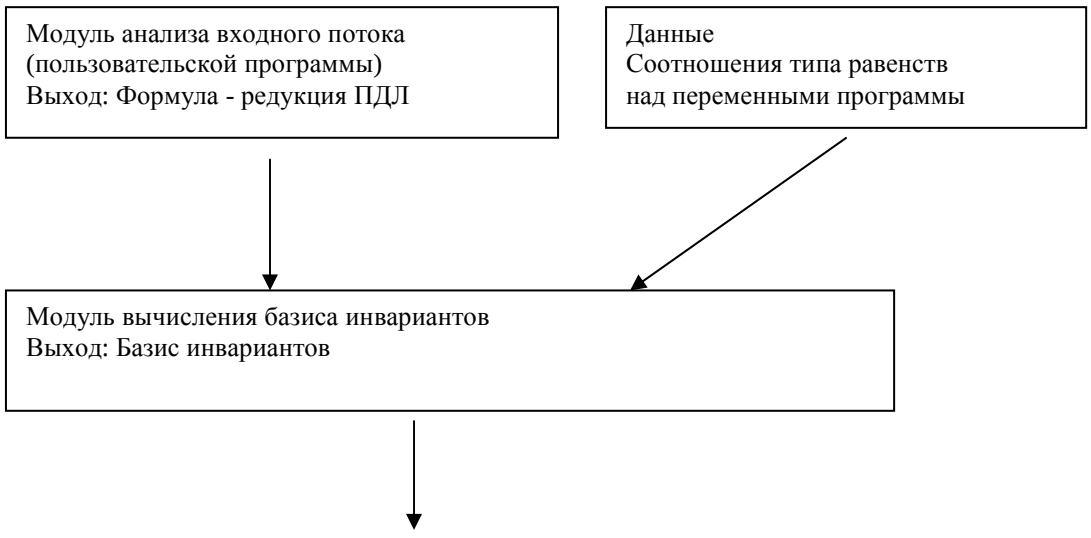
Для ускорения вычислений матрицы $L_k^{(j)}, k = 1...n$ нужно привести к верхней треугольной форме элементарными преобразованиями строк. Заметим, что эта форма и представляет базис Гребнера для нашого случая (алгоритм построения базиса Гребнера, примененный к набору линейных многочленов, дает именно это представление). Таким образом, на каждом шаге алгоритма матрицы $L_k^{(j)}, k = 1...n$, представлены в указанном виде, все нулевые строки вычеркнуты. Потому $L_k^{(j)}$ содержат не более **n** строк. Таким образом, $P_j = (L_1^{(j)},...,L_n^{(j)},0)$, где все $L_k^{(j)}$-верхние треугольные матрицы размеров $r_k \times n,\ r_k \le n$. Если на следующем (j+1)-ом шаге в результате вычислений (в.т.ч. приведения к верхнему треугольному виду) матрица $L_k^{(j+1)}$ совпадет с $L_k^{(j)}$, в дальнейших вычислениях **к**-тую координату вектора P_j можно не вычислять, т.к. по ней стабилизация уже достигнута. Эти рассуждения позволяют сделать вывод о том, что $m \le n$. Конечно, эта оценка сильно завышена, поскольку предполагается, что на каждом шаге итерации ровно одна из координатных матриц увеличивает свой ранг на 1. Можно предположить, что для реальних программ **m << n**.

Предположим, наконец, что в задаче вычисления линейных инвариантов для данной программы **p** конечное множество слов $\mathbf{u_1, u_2,\ldots,u_l}$ построено, а также найдены результирующие вычисляющие операторы $\mathbf{F_1, F_2, \ldots, F_l}$. Тогда осталось составить систему линейных уравнений от неизвестных $\mathbf{a = (a_1,\ldots,a_n, b)}$

$$\begin{cases} a * F_1 = 0 \\ a * F_2 = 0 \\ \ldots \\ a * F_l = 0 \end{cases}$$

и найти ее фундаментальные решения.

Блок схема программной системы вычисления инвариантов программ.



Литература

1. *Летичевский А.А*. Об одном подходе к анализу пограмм // Кибернетика. – 1979. - №6, С.1-8;
2. *Годлевский А.Б., Капитонова Ю.В., Кривой С.Л., Летичевский А.А*. Итеративные методы анализа программ // Кибернетика. – 1984. – №2, С.23-28;
3. *Капітонова Ю.В., Кривий С.Л., Летичевський О.А., Луцький Г.М., Печурін М.К.* Основи дискретної математики. Підручник. - К.: Наукова думка. – 2002. – 580 с.
4. *Кривой С.Л.* Об одном алгоритме поиска инвариантных соотношений в программах. // Кибернетика. – 1981. - №5. – С. 12 – 18.
5. *Львов М.С*. Инвариантные равенства малых степеней в программах, опеределенных над полем. // Кибернетика. – 1988. - №1. – С. 108 – 110.
6. *Бухбергер Б., Калме Ж., Калтофен Э*. Компьютерная алгебра: символьные и алгебраические вычисления: Пер. с англ. – М.: Мир, 1986. – 392 с.