УДК 004.421.6

М.С. Львов

СТАТИЧЕСКИЙ АНАЛИЗ ФИЗИЧЕСКИХ РАЗМЕРНОСТЕЙ ПЕРЕМЕННЫХ ПРОГРАММ И ЕГО РЕАЛИЗАЦИЯ В АЛГЕБРАИЧЕСКОМ ПРОГРАММИРОВАНИИ

В работе рассматривается класс «физических» программ – т. е. программ, осуществляющих физические вычисления. Некоторые переменные таких программ имеют физический смысл, определяемый их физическими размерностями. Приведен алгоритм статического анализа исходного программного кода, проверяющий правильность использования переменных в соответствии с их физическими размерностями. Используются алгебраические модели программ, дополненные спецификациями физических размерностей входных и выходных переменных. Алгоритм интерпретирует эту модель, используя системы соотношений типа равенств из стандартной системы физических размерностей и семантику операторов алгебраической модели объектного языка программирования. Алгоритм реализован средствами алгебраического программирования системы APS-1.

Введение

Компьютерные программы, в которых реализованы физические вычисления, должны удовлетворять нескольким специфическим свойствам. Одно из таких свойств заключается в корректном использовании размерных физических величин.

Вычисления в «физических» программах основаны на формулах физических законов, переменные которых, кроме имени, типа и значения, имеют еще одну характеристику – физическую размерность. Современные языки программирования не используют по умолчанию ни типизации переменных по их физическому смыслу, ни, тем более, физических размерностей констант и переменных. Даже языки со строгой типизацией, поддерживающие перегрузку арифметических операций, не вполне пригодны для определения специальных физических типов данных. В самом деле, эти типы определяются векторами физических размерностей, поэтому вычисление типа значения арифметического выражения требует вычислений в векторном пространстве, определенном одной из стандартных систем физических размерностей (например, Си, таблица). Поэтому ошибки в программах, связанные с неправильным использованием физических размерностей, не выявляются ни во время компиляции, ни во время исполнения программы.

Таким образом, возникает задача разработки программы, проверяющей правильность использования переменных по их физическим размерностям. Пользователь должен специфицировать все входные и выходные переменные программы их физическими размерностями, а программа должна проверить правильность этой спецификации.

Алгоритм анализа программного кода опирается на соотношения типа равенств – физические законы и правила систем физических размерностей, которые естественно представят в виде систем переписывающих правил. Например, соотношение *Вольт=Ампер*Ом* – закон Ома в системе Си. Это означает, что если в программе есть оператор присваивания `I:=A*R`, то переменная I должна иметь такую же физическую размерность, как и выражение $A*R$.

Практика показывает, что многие алгоритмы анализа программ могут быть эффективно реализованы средствами алгебраического программирования, т. е. в виде совокупности систем переписывающих правил. Такие алгоритмы описываются особенно просто, если использовать алгебраические модели объектных программ. Таким образом, объект исследования – приложения методологии алгебраического программирования к задачам статического анализа программ, а предмет

исследования – разработка алгебраической программы анализа правильности использования «физических» переменных в программах.

1. Постановка проблемы

Пусть $P(W_P)$ – математическая модель программы и W_P – множество ее переменных. Физической размерностью переменной $x \in W_P$ назовем элемент $r \in Rat^n$ векторного пространства Rat^n над полем рациональных чисел, а физической величиной (интерпретированной переменной) – пару $\langle x, r \rangle$. Физической интерпретацией $Int(W_P)$ программы $P(W_P)$ назовем множество физических величин: $Int(W_P) = \{\langle x, r \rangle : \forall x \in W_P\}$. Требуется построить алгоритмическую функцию

$$PhInt(P, Int) = \begin{cases} (P^*, Int^*), \text{ если Int}(W_P) \\ \text{определена корректно;} \\ \omega - \text{в противном случае.} \end{cases}$$

Функция $PhInt(P, Int)$ должна быть описана средствами алгебраического программирования, т. е. в виде систем переписывающих правил. Предполагается, что исходный программный код программы P синтаксически правилен и его математическая модель построена правильно. Задача построения математической модели исходной программы здесь не рассматривается.

Математическая модель класса исследуемых программ описана в разделе 3. Функция $PhInt(P, Int)$, определяющая и проверяющая корректность интерпретированной программы, описана в разделе 4. Отметим, что заключительная программа P^* – программа без циклов такая, что $PhInt(P^*, Int) = (P^*, Int^*)$. Заключительная интерпретация Int^* содержит вычисленные физические размерности локальных переменных.

2. Обзор литературы

Как показано в [1], рассматриваемая задача решается методами статического анализа программного кода. Причины, по которым необходим статический анализ для верификации программ этого типа – так называемого программного обеспечения критического оборудования обсуждаются в [2–4]. Еще один класс такого программного обеспечения – программы учебного назначения. Таким образом, задача актуальна.

Технологии алгебраического программирования [5, 6] успешно используются для разработки программного обеспечения во многих предметных областях [7–9], в том числе и в задачах статического анализа программ [10]. Практика показала, что в качестве модели объектной программы удобно использовать выражения в алгебре алгоритмов Глушкова [11].

В настоящей статье описано построение алгебраической функции, эффективно решающей задачу статического анализа правильности применения «физических» переменных в программах. В качестве модели объектной программы используется выражение в алгебре алгоритмов Глушкова. Тем самым показано, что в терминах алгебраического программирования алгоритм по существу представляет собой спецификации объектной программы в виде систем переписывающих правил и одновременно эффективную программу статического анализа программного кода, работающую «на лету».

3. Модель программы

3.1. Модель памяти программы. Пусть

$X = (x_1, x_2, ..., x_k)$ – вектор входных переменных программы P;

$Y = (y_1, y_2, ..., y_m)$ – вектор выходных переменных программы P;

$Z = (z_1, z_2, ..., z_l)$ – вектор локальных переменных программы P.

Памятью программы назовем множество $W = X \cup Y \cup Z$.

Физической размерностью переменной $x \in W$ назовем вектор $r = (r_1, ..., r_n)$ n-мерного векторного пространства Rat^n, где Rat – поле рациональных чисел.

Таблица. Основные и производные единицы системы Си

Название и обозначение величины		Единица измерения	Обозначение	Формула	Показатели степени					
					м	кг	с	А	К	кд
Длина	L	Метр	M	L	1					
Масса	M	Килограмм	Kg	M		1				
Время	T	Секунда	S	T			1			
Сила электр. тока	I	Ампер	A	I				1		
Термодинамическая температура	T	Кельвин	K	T					1	
Сила света	I_v	Кандела	Cd	J						1
Сила	F	Ньютон	N	F = ma	1	1	−2			
Давление	P	Паскаль	Pa	P = F/S	−1	1	−2			
Работа, энергия	A	Джоуль	J	A = F·L	2	1	−2			
Импульс	P		kg·m/s	p = m·v	1	1	−1			
Мощность	P	Ватт	W	P = A/t	2	1	−3			
Электрический заряд	Q	Кулон	C	q = I·t			1	1		
Электрическое напряжение, электрический потенциал	U	Вольт	V	U = A/q	2	1	−3	−1		

Физической величиной u назовем пару $u =< x, r >,\ x \in W,\ r \in Rat^n$. Если u – физическая величина, то, по определению, x – ее идентификатор, а r – физическая размерность.

Определение размерности физической величины связывает единицу измерения **(**идентификатор размерности) с вектором показателей степеней основных физических величин системы физических размерностей, выбранной в качестве стандартной. Фрагмент такого определения представлен в таблице. Отметим, что на практике можно расширить язык спецификаций таблицей имен единиц измерения физических величин. Например, вместо спецификации $< W, (2,1,-3,0,0,0) >$ можно использовать $< W, Bamm >$.

3.2. Модель вычислений. Оператор присваивания имеет вид

$$v := f(u_1, u_2, \ldots, u_j),\ u_1, u_2, ..., u_j, v \in W. \quad (1)$$

Алгебраические выражения в правых частях операторов присваивания определены в сигнатуре термов (операций)

$$()+(),\ ()-(),\ ()*(),\ ()/(),()\wedge n,\ \sqrt{()},\ \sqrt[n]{()}. \quad (2)$$

Физическая размерность выражения правой части оператора присваивания определяется следующими правилами:

```
<x,r> + <y,r> = <x + y, r>,        (3)
<x,r> - <y,r> = <x - y, r>,
C*<x,r> = <C*x, r>,// Умножение на безразмерную константу
<x, r1>*<y, r2> = <x*y, r1 + r2>,
<x,r1>/<y,r2> = <x*y, r1 - r2>,
<x,r>^n = <x^n, n*r>,
Sqrt(<x, r>) = <Sqrt(x), r/2>,
Sqrtn(<x, r>) = <Sqrtn(x), r/n>.
```

Отметим, что аддитивные операции над физическими величинами определены частично – только в случае, когда физические размерности обеих операндов совпадают. Иначе в выражении допущена ошибка, сигнализируемая специальным

символом ω (wrong), которая выявляется правилами

$$\texttt{r1<>r2} \rightarrow \texttt{<x,r1> + <y,r2> = ω}, \quad (3')$$
$$\texttt{r1<>r2} \rightarrow \texttt{<x,r1> - <y,r2> = ω}.$$

Физической интерпретацией оператора присваивания (1) называется пара $(< v, r_v >, < f(u_1,...,u_n), r_f >)$, вычисленная по правилам (3, 3′). Физической интерпретацией оператора присваивания (1) называется корректной, если r_v определена и $r_v = r_f$.

В противном случае возможны три варианта:

v1. r_v не определена (переменная v не интерпретирована).

v2. Переменная v интерпретирована повторно.

v3. В программе допущена ошибка интерпретации.

Условия в операторах управления программы – суть бескванторные формулы прикладной логики предикатов в сигнатуре термов (2), сигнатуре атомарных предикатов-условий $() > (), \ () = (), \ () \geq ()$ и сигнатуре логических связок $() \& (), () \vee (), \neg()$.

Физические размерности атомарных условий определяются правилами

$$\begin{aligned}
&\texttt{<x,r> > <y,r> = x > y},\\
&\texttt{<x,r> = <y,r> = x = y},\\
&\texttt{<x,r> ≥ <y,r> = x ≥ y},\\
&\texttt{r}_1\texttt{<> r}_2 \rightarrow \texttt{<x,r}_1\texttt{> > <y,r}_2\texttt{> = ω},\\
&\texttt{r}_1\texttt{<> r}_2 \rightarrow \texttt{<x,r}_1\texttt{> = <y,r}_2\texttt{> = ω},\\
&\texttt{r}_1\texttt{<> r}_2 \rightarrow \texttt{<x,r}_1\texttt{> ≥ <y,r}_2\texttt{> = ω}.
\end{aligned} \quad (4)$$

Таким образом, сравнимы физические выражения одинаковой физической размерности. В противном случае в сравнении допущена ошибка, сигнализируемая символом ω. Поэтому физическая интерпретация атомарного условия называется корректной, если она не равна ω. Физическая интерпретация составного условия называется корректной, если корректны все физические интерпретации ее атомарных условий.

Замечание. В настоящей версии алгоритма анализа использование различных, но сравнимых размерностей приводит к генерации признака ошибки. Так, размерности *секунда* и *минута* считаются различными, и алгоритм анализа сгенерирует значение ω. Таким образом, если в формуле используются две величины сравнимых размерностей, пользователь сам должен решить, допущена ошибка или различие в этих размерностях учтено домножением на соответствующую константу.

3.3. Модель управления. Операторы управления программы – условные операторы и операторы циклов

If U **then** P, **If** U **then** P **else** Q,
While U **do** P,
Repeat P **until** U.

с общепринятой семантикой. Таким образом, мы ограничиваемся структурированными программами (процедурами, функциями). В соответствии с определениями выражения в алгебре алгоритмов Глушкова в дальнейшем используется следующая алгебраическая и функциональная формы записи операторов управления:

P; Q // Последовательное выполнение

If U then P ~ $\underset{U}{P \vee I}$, **If(U,P)**

If U then P else Q ~ $\underset{U}{P \vee Q}$, **Ifelse(U,P,Q)**

While U do P ~ $\underset{U}{\{P\}}$, **While(U,P)**

Repeat P until U. ~ $\underset{U}{\{P\}}$**.Repeat(U,P)**

Множество всех операторов присваивания в программе P обозначим $Ass(P)$, а множество всех условий – $Cond(P)$.

3.4. Физическая интерпретация программы определяется множеством физических величин

$$W^R = < X^R, Y^R, Z^R >,$$
$$X^R = << x_1, r_{11} >, ..., < x_k . r_{1k} >> ,$$
$$Y^R = << y_1, r_{21} >, ..., < y_m . r_{2m} >> ,$$
$$Z^R = << z_1, r_{31} >, ..., < z_l . r_{3l} >> .$$

Исходной физической спецификацией называется множество

$$W_0^R =< X^R, Y^R >.$$

Это множество задается пользователем.

Будем считать, что исходные физические интерпретации не могут изменяться в процессе вычислений. Если одна из переменных множества $X \cup Y$ интерпретируется повторно в результате выполнения оператора присваивания, ее новая интерпретация должна совпадать с исходной. Иначе в программе допущена ошибка ω физической интерпретации.

Заметим, что в языках программирования повторная интерпретация любых переменных допустима, если выполнен контроль типов.

Вектор $\boldsymbol{Z}$ локальных переменных не специфицирован. Таким образом, для исходной интерпретации переменных из $\boldsymbol{Z}$ алгоритм использует специальный символ α. Исходная интерпретация переменной $z \in Z$ имеет вид $< z, \alpha >$. $Z^\alpha = \{< z, \alpha > \mid z \in Z\}$.

Исходной физической интерпретацией программы P называется множество $W_\alpha^R =< X^R, Y^R, Z^\alpha >$. Обозначим это множество $Int^{(0)}$. Для вычисления физической интерпретации программы P с исходной интерпретацией $Int^{(0)}$ введем специальную функцию $PhInt(P, Int)$, отображающую программу P с исходной интерпретацией $Int^{(0)}$ в ее заключительную физическую интерпретацию:

$$PhInt : P \times W^R \to P \times W^R.$$

Если $P(W)$ – программа над памятью $W = X \cup Y \cup Z$, не содержащая ни одного оператора, $PhInt(P, W_o^R) = (P, W_\alpha^R)$.

Пусть P – программа, представленная в алгебраической форме. Путем w в программе P назовем последовательность операторов присваивания и условий $s_1 s_2 ... s_N$, выполняемых последовательно при исполнении программы:

$$w = s_1 s_2 ... s_N,\ s_j \in Ass(P) \cup Cond(P).$$

Точное определение пути – это индуктивное определение по структуре формулы программы. Оно будет сформулировано при доказательстве основной теоремы.

Программу P будем ассоциировать с множеством ее путей точно так же, как регулярное выражение ассоциируется с регулярным языком – множеством слов, определенных этим выражением. Поэтому имеет смысл отношение $w \in P$.

Пусть $Int^{(0)}$ – исходная физическая интерпретация. Выполнение программы P вдоль пути w преобразует интерпретацию $Int^{(0)}$ в интерпретацию $Int^{(w)}$.

Физическая интерпретация пути w в программе P называется корректной, если $Int^{(w)} \neq \omega$. Физическая интерпретация программы P называется корректной, если все интерпретации $Int^{(w)}$, $w \in P$ корректны.

Правила вычисления интерпретации пути заданы функциями $PhIntExpr$, $PhIntCond$, $PhIntAss$ и правилами интерпретации последовательного выполнения, сформулированными ниже.

4. Алгоритм и алгебраическая программа анализа физических размерностей

4.1. Основные функции алгебраической программы. На предварительном этапе алгоритм заменяет в специфицированной программе P все переменные своими исходными физическими интерпретациями из Int, используя систему переписывающих правил, определенную таблицей размерностей физических величин стандартной системы физических размерностей (см. таблицу).

Вычисления физических интерпретаций арифметических выражений и условий осуществляют функции $PhIntExpr(f, Int)$, $PhIntCond(U, Int)$, основанные на системах переписывающих правил, расширяющих соответственно (3), (4).

Замена переменной ее физической величиной реализована подстановкой в структуры данных арифметических выражений и условий вместо переменных указателей на соответствующие физиче-

ские величины в списке физических величин W^R. Таким образом, $< x,r >$ реализована как указатель на структуру (x, r) – элемент списка. Такая реализация позволяет изменять вектор r одновременно во всех вхождениях переменой x в программу. Системы переписывающих правил интерпретирующих функций на внешнем языке Aplan системы программирования APS имеют вид:

```
PhIntExpr := rs(x,y,r,r1,r2 C){
 IsConst(C)→ C = <Const, 0>,
 <x,α> = return(ω),
 <x,r> + <y,r> = <x + y, r>,
 r1<>r2 → <x,r1>+<y,r2> = return(ω),
<x,r> - <y,r> = <x − y, r>,
 r1<>r2 → <x,r1>-<y,r2> = return(ω),
 <x, r1>*<y, r2> = <x*y, r1+r2>,
 <x,r1>/<y,r2> = <x*y, r1 − r2>,
 <x,r>^n = <x^n, n*r>,
 Sqrt(<x, r>) = <Sqrt(x), r/2>,
 Sqrtn(<x, r>) = <Sqrtn(x), r/n>
};

PhIntCond := rs(x,y,r1,r2,L,R){
 L > R = PhIntExpr(L)> PhIntExpr(R),
 L = R = PhIntExpr(L)= PhIntExpr(R),
 L ≥ R = PhIntExpr(L)≥ PhIntExpr(R),
 r1<>r2 →<x,r1> > <y,r2> = return(ω),
 r1<>r2 →<x,r1> = <y,r2> = return(ω),
 r1<>r2 →<x,r1> ≥ <y,r2> = return(ω)
};
```

Правила вычисления физической интерпретации оператора присваивания имеют вид:

```
PhIntAss := rs(u, f, r, r1, r2){
//вариант v1
<u, α> := <f, r> = <u, r>,
// вариант v2
u∈Z)→<u, r1> := <f, r2>=<u, r2>,
// вариант v3
(u∈X∪Y)&(r1<>r2)→<u,r1> :=
                  <f, r2> = return(ω)
};
```

Определим теперь физические интерпретации для каждого оператора управления.

// **RO.** Прерывание и обработка исключительной ситуации

```
PhInt(P, ω) = return(ω);
```

// **R1.** Интерпретация оператора присваивания

```
PhInt(u:=v, Int)=PhIntAss(u:=v, Int);
```

// **R2.** Интерпретация последовательного выполнения

```
PhInt(P;Q, Int) = PhInt(Q,
PhInt(P,Int));
```

// **R3.** Интерпретация неполного ветвления. Интерпретации ветвей должны быть равны

```
PhIntCond(U, Int)= ω →
PhInt(If(U,P), Int) = return(ω),
PhInt(P, Int)<> Int → PhInt(If(U,P),
Int) = return(ω),
PhInt(If(U,P), Int) = Int;
```

// **R4.** Интерпретация полного ветвления. Интерпретации ветвей должны быть равны

```
PhIntCond(U)== ω →
PhInt(Ifelse(U,P,Q),Int) = return(ω),
(PhInt(P)<> PhInt(Q))→
PhInt(Ifelse(U,P,Q),Int) = return(ω),
PhInt(Ifelse(U,P,Q),Int) =
                        PhInt(P, Int);
```

// **R5.** Интерпретация цикла While

```
PhIntCond(U)= ω → PhInt(While(U,P),
Int) = return(ω),
PhInt(P, Int)<> Int →
PhInt(While(U,P), Int) = return(ω),
PhInt(While(U,P), Int) = PhInt(P);
```

// **R6.** Интерпретация цикла Repeat. Выполнение ***P*** изменяет интерпретацию ***U***.

```
PhInt(P, Int))= ω →
PhInt(Repeat(U,P), Int) = return(ω),
PhIntCond(U,PhInt(P,Int))=ω→PhInt(Re
peat(U,P),Int) = return(ω),
PhInt(P,Int)<>PhInt(P,PhInt(P,Int))
→PhInt(Repeat(U,P),Int)= return(ω),
PhInt(Repeat(U,P), Int) = PhInt(P);
```

Физическая интерпретация *Int* программы ***P*** равна $PhInt(P, Int^{(0)})$. Отметим, что для функции *PhInt* выполняются следующие основные свойства:

1) только два правила системы `PhIntAss` изменяют интерпретацию, не прерывая работы алгоритма ввиду ошибки интерпретации – это правила вариантов 1 и 2;

2) правила интерпретации условий `PhIntCond` не изменяют корректных интерпретаций;

3) правила интерпретации операторов управления **R3**–**R6** требуют, чтобы интерпретации вдоль различных путей вычислений, определяемых этими операторами, совпадали. Поскольку интерпретации переменных из множества $X \cup Y$ зафиксированы в спецификациях, это относится к переменным из ***Z***. Точнее, если $s_{11}, s_{12}, ..., s_{1j1}$, $s_{21}, s_{22}, ..., s_{2j2}$ – два пути в одном операторе управления, вычисления вдоль каждого из путей могут менять интерпретации локальных переменных, но результирующие интерпретации должны совпасть.

4.2. Основная теорема. Теорема. Алгоритм, определенный правилами *PhIntExpr*, *PhIntCond*, *PhIntAss*, **R0**–**R6** правильно вычисляет физическую интерпретацию программы ***P***.

Доказательство. Мы покажем, что $PhInt(P, Int^{(0)}) = \omega$ тогда и только тогда, когда в программе ***P*** существует путь ***w*** такой, что $Int^{(w)} = \omega$. Доказательство будем вести индукцией по структуре формулы программы.

1. Предположим, что

$$PhInt(P, Int^{(0)}) = \omega.$$

Анализ правил алгоритма *PhInt* показывает, что ошибка, если она обнаруживается алгоритмом, генерируется вдоль путей, получаемых выполнением операторов в каждом цикле некоторое минимально необходимое количество раз. Назовем такие пути *простыми*. Точное определение простых путей будет задано формулами в индуктивных шагах доказательства.

а) (базис индукции) $P = s,\ Ass(P) = \{s\}$. Путь $w = s$ является простым.

б) $P = Q_1; Q_2$. По правилу **R2**

$$\mathtt{PhInt}(Q_1;Q_2, \mathtt{Int}^{(0)}) = \mathtt{PhInt}(Q_2, \mathtt{PhInt}(Q_1, \mathtt{Int}^{(0)})) = \omega.$$

Тогда либо $\mathtt{PhInt}(Q_1, \mathtt{Int}^{(0)}) = \omega$ и по правилу **R0**

$$\mathtt{PhInt}(Q_2, \omega) = \omega,$$

либо $\mathtt{PhInt}(Q_1, \mathtt{Int}^{(0)}) \neq \omega$ и $\mathtt{PhInt}(Q_2, \mathtt{PhInt}(Q_1, \mathtt{Int}^{(0)})) = \omega$.

Если $\mathtt{PhInt}(Q_1, \mathtt{Int}^{(0)}) = \omega$, по предположению индукции существует простой путь $w_1 \in Q_1$ такой, что $Int^{(w_1)} = \omega$. Тогда для любого простого пути $w_2 \in Q_2$ искомым является простой путь $w_1 w_2$.

Если $\mathtt{PhInt}(Q_1, \mathtt{Int}^{(0)}) \neq \omega$, по предположению индукции, все интерпретации $Int^{(w_1)}$, $w_1 \in Q_1$ корректны. Следовательно, корректны интерпретации вдоль простых путей из Q_1. Любой путь из P начинается с некоторого пути w_1 из Q_1. Выберем все простые пути из Q_2. По предположению индукции, ошибка должна быть обнаружена вдоль некоторого простого пути w_2. Тогда ошибка интерпретации в P обнаруживается вдоль простого пути $w = w_1 w_2$.

Простые пути задаются формулой $P = Q_1; Q_2$;

в) $P = Q \underset{U}{\vee} I$. По правилу **R3,** ошибка обнаруживается либо в интерпретации условия, либо в интерпретации программы Q. Если интерпретация условия корректна, по предположению индукции ошибка обнаруживается вдоль простого пути из Q. В противном случае программа Q может не выполняться ни одного раза. Поскольку правила интерпретации условий не меняют корректных

интерпретаций, простые пути задаются формулой $P = U \vee Q$;

г) $P = Q_1 \underset{U}{\vee} Q_2$. По правилу **R4,** ошибка обнаруживается либо в интерпретации условия, либо в интерпретации программ Q_1, Q_2. Если интерпретация условия корректна, по предположению индукции ошибка обнаруживается либо вдоль простого пути из Q_1, либо вдоль простого пути из Q_2. В противном случае программы Q_1, Q_2 могут не выполняться ни одного раза. Отметим, что после выполнения $P = Q_1 \underset{U}{\vee} Q_2$ интерпретации вдоль различных путей должны совпадать. Иначе обнаруживается ошибка интерпретации. Простые пути задаются формулой $P = U; Q_1 \vee Q_2$;

д) $P = \underset{U}{\{Q\}}$. По правилу **R5,** ошибка обнаруживается либо в интерпретации условия U, либо в интерпретации программы Q. Если интерпретация условия корректна, по предположению индукции ошибка обнаруживается вдоль простого пути из Q. В противном случае программа Q может не выполняться ни одного раза. Следовательно, при выполнении Q интерпретация не должна меняться. Простые пути задаются формулой $P = U \vee Q$;

е) $P = \underset{U}{\{Q\}}$. По правилу **R6,** ошибка обнаруживается либо в теле цикла Q, либо в интерпретации условия $\boldsymbol{U}$ после однократного выполнения тела цикла. Если ошибка не обнаружена, интерпретация программы P равна интерпретации тела цикла Q при исходной интерпретации. Повторное выполнение тела цикла Q не должно изменить интерпретации. По предположению индукции, ошибка обнаруживается вдоль простого пути из Q. Простые пути задаются формулой $P = Q; U \vee Q; Q$.

2. Предположим, что в программе P существует путь $\boldsymbol{w}$ такой, что $Int^{(w)} = \omega$.

а) (базис индукции) $P = s,\ Ass(P) = \{s\}$. В этом случае $w = s$. И $Int^{(w)}$, и $PhInt(P, Int^{(0)})$ вычисляется одними и теми же правилами **`PhIntAss`** (правило **R1**). Следовательно,

$$Int^{(w)} = PhInt(P, Int^{(0)}) = \omega.$$

Заметим, что в этом случае имеет место вариант **v3** правил, т. е. интерпретации левой и правой частей оператора s различны;

б) $P = Q_1; Q_2$ (см. п. б)) предыдущего доказательства;

в) $P = Q \underset{U}{\vee} I$. Тогда $P = U; Q \vee \neg U$. Поскольку U не меняет интерпретации, $\neg U$ также не меняет интерпретации. Наконец, интерпретации $U; Q$ и $\neg U$ должны совпасть. Следовательно, ошибка должна быть обнаружена вдоль путей $U \vee Q$;

г) $P = Q_1 \underset{U}{\vee} Q_2$. Рассуждения, аналогичные предыдущим, приводят к путям $U; Q_1 \vee Q_2$**;**

д) $P = \underset{U}{\{Q\}}$. Рассуждения, аналогичные п. в), приводят к путям $U \vee Q$;

е) $P = \underset{U}{\{Q\}}$,

$P = Q; U \vee Q; \neg U; Q; U \vee Q; U; Q; U; \neg U; Q \vee \ldots$

Поскольку интерпретации

$$Q; U,\ Q; \neg U; Q; U,$$

если они корректны, должны совпадать, то и интерпретации $Q, Q; Q$ должны совпадать. Если эти интерпретации совпадают, повторное вычисление тела Q вдоль пути $Q; Q$ не изменяет интерпретации P. Следовательно, и последующие вычисления $Q; Q; Q \ldots$ уже не изменят интерпретации P.

Теорема доказана.

5. Дискуссия

Основное свойство, на котором построено доказательство основной теоремы – свойство 3. Можно привести пример, в котором это свойство нарушено, а программа все-таки имеет корректную физическую интерпретацию. Однако, с нашей точки зрения, это плохой стиль программирования, когда, например, в одной ветви вычислений переменная T обозначает время, а в другой – той же переменной T обозначена температура.

Алгоритм анализа ориентирован на применение к процедурам и функциям. Однако нет никаких проблем распространения его на модули и программные системы.

Модель программы использует только простые переменные. Однако планируется распространить его и на составные типы данных, что требует усложнения модели памяти и анализа раздела типов.

Заключение

Значение функции $PhInt(P, Int^{(0)})$ вычисляется стандартной стратегией «снизу-вверх». Значения аргументов при этом не перевычисляются. Поэтому интерпретатор, описанный в работе, по спецификациям исходной программы строит программу без циклов P^* за один проход. Поэтому алгебраическая программа $PhInt$ эффективна – она проверяет правильность использования физических размерностей «на лету». Заметим, что алгоритм, приведенный в [1], строит и решает систему линейных уравнений, что не эффективно.

Проблема построения алгебраической модели объектной программы здесь не рассматривается. Однако, поскольку программы на языках процедурного программирования по существу представляют собой алгебраические объекты, эта проблема средствами алгебраического программирования может быть решена эффективно. Как показано в [2–4], задача анализа физических размерностей – одна из основных задач разработки программного обеспечения критического оборудования. Поэтому ее дальнейшее развитие имеет большое практическое значение.

Автор благодарит профессора Г.Н. Жолткевича за формулировку проблемы и полезные ее обсуждения, а также академика А.А. Летичевского за многолетнее сотрудничество в области алгебраического программирования.

1. *Briukhankov S.S., Konoriev B.M., Lvov M.S., Zholtkevitch H.N.* About the Static Variable Dimensions Analysis for Software of Critical Appliance (English) // Radioelectronic and Computing Systems. N 6(47). Kharkiv "KHAI". – 2010. – P. 186–191.
2. *Rushby J.M.* Formal Verification of Algorithms for Critical Systems // IEEE Trans. Soft. Eng. – 1993. – Vol. 19, N 1. – P. 13–23.
3. *Binkley D.* Source Code Analysis: A Road Map // Future of Software Engineering 2007. – IEEE: Proc. FOSE'07, 2007. – P. 89–105.
4. *Konorev B.M., Kharchenko V.S,. Chertkov G.N., Alekseev Yu.G., Manzhos Yu.S., Sergienko V.V.* Integrated Instrumental Environment for Critical-Mission Software Systems Assessment, in Russian // Kharkiv: I&C System Certification Centre. State Centre for Regulation of Supplies and Service of Quality of State Nuclear Regulatory Committee of Ukraine, 2005. – 258 p.
5. *Letichevsky A.A., Kapitonova J., Volkov V. and others.* Algebraic Programming System APS: User Manual // Glushkov Institute of Cybernetics, National Acad. of Sciences of Ukraine, Kiev, Ukraine, 1998. – 50 p.
6. *Kapitonova J., Letichevsky A., Lvov M., Volkov V.* Tools for Solving Problems in the Scope of Algebraic Programming // Lecture Notes in Computer Sciences. – N 958. – 1995. – P. 31–46.
7. *Lvov M.S.* Algebraic approach to the problem of solving systems of linear inequalities // Cybernetics and Systems Analysis. – 2010. – Vol. 46, N 2. – P. 326–338.
8. *Lvov M.S., Peschanenko V.S.* Trapezoid method for solving systems of linear inequalities and its implementation in insertion modeling // Cybernetics and Systems Analysis. – 2012. – Vol. 48, N 4. – P. 931–942.
9. *Letichevsky A., Letychevskyi O., Peschanenko V.* About One Efficient Algorithm for Reachability Checking in Modeling and Its Implementation // Communications in Computer

and Information Science. – 2012. – N 149. – P. 149–165.
10. *Lvov M.S.* Polynomial invariants for linear loops. // Cybernetics and Systems Analysis. – 2010. – Vol. 46, Issue 4. – P. 660–668.
11. *Глушков В.М., Цейтлин Г.Е., Ющенко Е.Л.* Алгебра. Языки. Программирование. 3-е изд., перераб. и доп. – Киев: Наук. думка, 1989. – 376 с.

Получено 21.11.2014

Об авторе:

Львов Михаил Сергеевич,
доктор физико-математических наук,
доцент, профессор кафедры информатики,
программной инженерии
и экономической кибернетики.

Место работы автора:

Херсонский государственный университет,
ул. 40-летия Октября, 27.
73000 Херсон, Украина.
Тел: +380552 32 67 81,
+380552 43 23 15.
E-mail: lvov@ksu.ks.ua