УДК 004.052.42:004.415.5

АЛГОРИТМ ИНТЕРВАЛЬНОГО АНАЛИЗА ДЛЯ ОБНАРУЖЕНИЯ ДЕФЕКТОВ В ИСХОДНОМ КОДЕ ПРОГРАММ

В. М. Ицыксон,
канд. техн. наук, доцент
М. Ю. Моисеев,
старший преподаватель
В. А. Цесько,
аспирант
А. В. Захаров,
аспирант
М. Х. Ахин,
магистрант
Санкт-Петербургский государственный политехнический университет

Предлагается алгоритм интервального анализа с интерпретацией условий в операторах ветвления, предназначенный для обнаружения дефектов в исходном коде программ на языке С. Особенностью данного алгоритма является формирование оценки вероятности наличия дефекта. Рассматриваются способы управления точностью и полнотой анализа. Применение предложенного алгоритма иллюстрируется на примере программы.

Ключевые слова *— статический анализ, интервальный анализ, обнаружение дефектов, программные дефекты.*

Введение

Одной из важнейших проблем при промышленной разработке программных систем является обеспечение требуемого качества. Наличие ошибок в программных системах существенно влияет на такие характеристики качества как функциональность, надежность, эффективность, сопровождаемость и другие [1]. Обнаружение и исправление программных ошибок является трудоемким процессом, который в общем случае не может быть автоматизирован.

Можно выделить особый класс нефункциональных программных ошибок, которые вносятся на этапе разработки. Будем называть такие ошибки программными дефектами. Обнаружение программных дефектов может выполняться автоматически с использованием различных методов и алгоритмов статического анализа.

В данной статье рассматриваются вопросы автоматического обнаружения дефектов на основе статического анализа исходного кода программ на языке С. Выбор языка С обусловлен тем, что он широко используется при разработке прикладного и системного программного обеспечения (ПО), а также ПО для встраиваемых систем. Кроме того, выбор языка С определяется отсутствием в нем автоматического управления ресурсами и контроля инициализации переменных, а также наличием потенциально опасных конструкций, таких как конструкции адресной арифметики, вызова функций через указатели и возможность неконтролируемого приведения типов.

Существует несколько классов алгоритмов статического анализа, используемых при обнаружении дефектов: анализ указателей [2], интервальный анализ [3], анализ типов [4]. По данным ряда авторитетных организаций [5, 6], одними из наиболее распространенных дефектов в программах на языке С являются переполнение буфера и использование неинициализированных переменных. Для обнаружения перечисленных дефектов целесообразно использовать методы интервального анализа.

Интервальный анализ обеспечивает получение интервалов значений для объектов программы. К объектам программы относятся переменные, элементы массивов и поля структур, а так-

же объекты в памяти, адресуемые через указатели. В данной статье предлагается алгоритм интервального анализа для обнаружения дефектов в программах на языке С.

Для оценки качества алгоритмов анализа введем понятие эффективности обнаружения дефектов. Эффективность обнаружения дефектов — комплексное свойство, отражающее качество получаемого результата, ресурсоемкость, применимость для программ различного размера и класса. Показателями эффективности обнаружения дефектов являются *полнота* и *точность* результатов анализа. Полнота результатов анализа — это показатель, характеризующий долю обнаруживаемых дефектов среди всех дефектов в программе. Точность результатов анализа — это показатель, характеризующий долю истинных дефектов среди всех обнаруженных дефектов.

Сформулируем общие требования к алгоритму интервального анализа. Алгоритм должен:

- формировать интервалы значений объектов в отдельных конструкциях программы;
- учитывать неинициализированные объекты;
- анализировать условия в операторах ветвления;
- обеспечивать возможность управления точностью и полнотой анализа.

Интервальный анализ

Существует несколько теорий, на которых базируются методы статического анализа [3]: абстрактная интерпретация, системы типов и эффектов, теория решеток. В качестве математического аппарата для интервального анализа предлагается использовать теорию решеток [7], поскольку известны эффективные алгоритмы поиска решения [3] в рамках этой теории.

Теория решеток

Решеткой называется частично упорядоченное множество L, для каждого подмножества X которого определена единственная точная верхняя грань $sup(X)$ и единственная точная нижняя грань $inf(X)$.

Пусть L — некоторая конечная решетка, а $f(x)$ — монотонная функция на L:

$$\forall x_1, x_2 \in L: \quad x_1 \subseteq x_2 \Rightarrow f(x_1) \subseteq f(x_2).$$

Наименьшей неподвижной точкой функции $f(x)$ называется минимальный элемент x_{LFP} решетки L, для которого справедливо

$$f(x_{LFP}) = x_{LFP}.$$

Любая монотонная функция имеет единственную наименьшую неподвижную точку.

Рассмотрим систему уравнений вида $x_i = f_i(x_1, ..., x_N)$. Пусть $x_i \in L$ и все функции f_i: $L^N \to L$ монотонны, тогда вектор функций $F = (f_1, ..., f_N)$ является монотонной функцией $L^N \to L^N$. Исходная система может быть записана в виде $X = F(X)$, где X — элемент L^N. Наименьшая неподвижная точка для функции $F(X)$ является решением данной системы уравнений.

Основные определения

При проведении интервального анализа формируется система уравнений над решеткой L_{INT}. Уравнения строятся для вершин графа потока управления (Control Flow Graph, CFG) программы по определенным правилам. Неизвестными в уравнениях являются состояния программы S^l в соответствующих вершинах CFG. Состояние программы соответствует некоторому элементу решетки L_{INT}.

Элементами L_{INT} являются подмножества множества всех возможных кортежей $\langle a, i\rangle$, где a — объект программы, i — интервал принимаемых им значений. Отношением порядка на решетке L_{INT} является отношение включения подмножеств:

$$A \subseteq B \quad \Leftrightarrow \quad \forall \langle a, i\rangle \in A \Rightarrow \langle a, i\rangle \in B,$$
$$\langle a, i\rangle = \langle b, j\rangle \quad \Leftrightarrow \quad (a \equiv b) \wedge (i = j).$$

Интервал i обладает следующими атрибутами: i^{low} — нижняя граница интервала; i^{high} — верхняя граница интервала.

Кроме обычных интервалов вводится специальный интервал $i_{noninit}$, соответствующий неинициализированному значению, который в дальнейшем будет использоваться для обнаружения дефектов.

Предлагается расширить кортежи дополнительным атрибутом r. Атрибут r является оценкой вероятности существования кортежа $\langle a, i\rangle$ в вершине CFG (далее — вероятность кортежа) и используется при обнаружении дефектов. Для этого атрибута выполняется условие нормировки:

$$\forall a: \langle a, i, r_j\rangle \in H, \ \sum_j r_j = 1,$$

где H — произвольный горизонтальный срез CFG. Горизонтальный срез представляет собой множество вершин, которые принадлежат альтернативным путям выполнения программы. Для каждого возможного пути срез H содержит ровно одну вершину CFG. Отметим, что атрибут r вычисляется в уравнениях, но не входит в решетку L_{INT} и, следовательно, не влияет на монотонность функций.

Обеспечение условий нормировки для циклов в CFG является отдельным вопросом. В силу ограниченного объема статьи анализ циклов в CFG остается за рамками рассмотрения.

Обозначим тип переменой a как T_a. Для обозначения максимального и минимального значения, которое могут принимать переменные типа T_a, будем использовать T_a^{high} и T_a^{low} соответственно.

Правила построения уравнений

Рассмотрим правила построения уравнений для вершин CFG, содержащих конструкции программы. Типы конструкций, для которых строятся уравнения, будем называть интерпретируемыми.

Правило построения уравнения для конструкции объявления переменной заключается в добавлении к предыдущему состоянию программы кортежа, состоящего из объявляемой переменной a и интервала $i_{noninit}$, при этом вероятность кортежа равна 1:

$$[declare(T\ a)]^l:\ \ S^l = \hat{S}^l \cup \langle a, i_{noninit}, 1\rangle,$$

где $\hat{S}^l$ — множество кортежей всех непосредственно предшествующих состояний для вершины l.

При выходе переменной из области видимости удаляются все кортежи для данной переменной:

$$[undeclare(a)]^l:\ \ S^l = \\ = \hat{S}^l \setminus \bigcup_{\forall\langle x,i,r\rangle\in\hat{S}^l: x=a} \langle a, i, r\rangle.$$

Обозначим сумму вероятностей кортежей для объекта a в конструкции l как R_a^l:

$$R_a^l = \sum_{\forall\langle x,i,r\rangle\in\hat{S}^l: x=a} r.$$

Правило для оператора присваивания, в правой части которого стоит константа, порождает уравнение

$$[a = C]^l:\ \ S^l = \hat{S}^l \setminus \bigcup_{\forall\langle x,i,r\rangle\in\hat{S}^l: x=a} \langle a, i, r\rangle \cup \\ \cup \langle a, [C, C], R_a^l\rangle.$$

В этом уравнении из предыдущего состояния программы удаляются все кортежи, содержащие объект программы a, и добавляется кортеж для объекта a с соответствующим интервалом и вероятностью R_a^l.

Правило для оператора присваивания $a = b$ состоит в удалении всех кортежей для объекта a и добавлении для него всех кортежей, которые имеются для объекта b:

$$[a = b]^l:\ \ S^l = \hat{S}^l \setminus \bigcup_{\forall\langle x,i,r\rangle\in\hat{S}^l: x=a} \langle a, i, r\rangle \cup \\ \cup \bigcup_{\forall\langle x,i,r\rangle\in\hat{S}^l: x=b} \left\langle a, i, \frac{r\cdot R_a^l}{R_b^l}\right\rangle.$$

Правило для оператора присваивания, в правой части которого стоит бинарная арифметическая (сложение, вычитание, умножение) или логическая (дизъюнкция, конъюнкция) операция, имеет вид

$$[a = b \otimes d]^l:\ \ S^l = \hat{S}^l \setminus \bigcup_{\forall\langle x,i,r\rangle\in\hat{S}^l: x=a} \langle a, i, r\rangle \cup \\ \cup \bigcup_{\substack{\forall\langle x,i,r\rangle,\langle y,j,p\rangle\in\hat{S}^l: \\ x=b,\ y=d,\ i\neq i_{noninit},\ j\neq i_{noninit}}} \langle a, \gamma(a,i,j), \hat{r}\rangle \cup \\ \cup \bigcup_{\substack{\forall\langle x,i,r\rangle,\langle y,j,p\rangle\in\hat{S}^l: \\ x=b,\ y=d,\ i=i_{noninit}\vee j=i_{noninit}}} \langle a, i_{noninit}, \hat{r}\rangle,$$

где $\hat{r} = \dfrac{r\cdot p\cdot R_a^l}{R_b^l\cdot R_d^l}$; функция $\gamma(a, i, j)$ возвращает интервал, получаемый в результате арифметической или логической операции над интервалами, которые соответствуют инициализированному значению объекта:

$$\gamma(a,i,j) = \left(\max\left(\min(G), T_a^{low}\right), \right. \\ \left. \min\left(\max(G), T_a^{high}\right)\right), \\ G = \left\{\left(i^{low}\otimes j^{low}\right),\ \left(i^{high}\otimes j^{low}\right),\right. \\ \left.\left(i^{low}\otimes j^{high}\right),\ \left(i^{high}\otimes j^{high}\right)\right\}.$$

Правило для операции деления сводится к правилу для умножения на выражение, обратное делителю:

$$[a = b/d]^l \Rightarrow [a = b\cdot d^{-1}]^l.$$

Кортежи для d^{-1} вычисляются по правилу

$$[d^{-1}]^l = \begin{cases} \langle d^{-1}, i, r\rangle, \\ \forall\langle x,i,r\rangle\in\hat{S}^l: x=d,\ i=i_{noninit}; \\ \left\langle d^{-1}, \beta\left(d^{-1},\ 1/i^{high},\ 1/i^{low}\right), r\right\rangle, \\ \forall\langle x,i,r\rangle\in\hat{S}^l: x=d,\ i\neq i_{noninit}\wedge(\{0\}\notin i); \\ \left\langle d^{-1}, \beta\left(d^{-1},\ T_d^{low},\ 1/i^{low}\right), r/2\right\rangle\cup \\ \cup\left\langle d^{-1}, \beta\left(d^{-1},\ 1/i^{high},\ T_d^{high}\right), r/2\right\rangle, \\ \forall\langle x,i,r\rangle\in\hat{S}^l: x=d,\ i\neq i_{noninit}\wedge(\{0\}\in i) \end{cases}$$

где функция $\beta(a, b_1, b_2)$ приводит значения b_1, b_2 к диапазону типа объекта a:

$$\beta(a, b_1, b_2) = \left(\max\left(b_1, T_a^{low}\right), \min\left(b_2, T_a^{high}\right)\right).$$

В стандартной библиотеке языка С существуют функции, которые возвращают неизвестное

значение. Примером такой функции является функция получения символа из стандартного потока ввода *getchar*(). Правило для этой функции выглядит следующим образом:

$$[a = getchar()]^l: \; S^l = $$
$$= \hat{S}^l \setminus \bigcup_{\forall\langle x,i,r\rangle \in \hat{S}^l : x=a} \langle a, i, r\rangle \cup$$
$$\cup \left\langle a, [T_a^{low}, T_a^{high}], R_a^l \right\rangle.$$

Для других подобных функций правила составляются аналогично.

Оператор ветвления

Вершины CFG, содержащие операторы ветвления, распределяют входное множество кортежей $\hat{S}^l$ на два подмножества S^l_{true} и S^l_{false}. Условие в операторе ветвления может быть интерпретируемым или неинтерпретируемым. Для оператора ветвления с интерпретируемым условием начальные состояния программы для каждой из веток определяются следующим образом:

$$[if(cond)\, S_{true}\, else\, S_{false}]^l:$$
$$S^l_{true} = \eta\left(\hat{S}^l, cond\right),$$
$$S^l_{false} = \eta\left(\hat{S}^l, \overline{cond}\right),$$

где $\eta(S, cond)$ — функция, пересекающая кортежи из состояния S с условием $cond$.

Для интерпретируемого условия функция η определяет множество кортежей из S, которые удовлетворяют условию $cond$, и вычисляет оценку вероятности перехода для соответствующей ветки $true$ (R_{cond}) или $false$ ($R_{\overline{cond}}$). Оценка R_{cond} рассчитывается как сумма оценок r для всех кортежей, удовлетворяющих условию $cond$. После этого множество кортежей для S пересекается с условием $cond$, в результате чего формируется множество кортежей для S_{true}. Для кортежей, объекты которых не используются в условии $cond$, выполняется нормировка оценок вероятности:

$$\forall\langle a,i,r\rangle \in S_{true} : r \to r \cdot R_{cond},$$
$$\forall\langle a,i,r\rangle \in S_{false} : r \to r \cdot R_{\overline{cond}}.$$

Для неинтерпретируемых условий выполняется аналогичная процедура без расчета оценок вероятностей переходов: они принимаются равными 0,5.

Рассмотрим несколько примеров функции η для интерпретируемых условий. Для условия типа $a < C$ (переменная меньше константы) функция η имеет вид

$$\eta(S, a < C) =$$
$$= \bigcup_{\forall\langle x,i,r\rangle \in S : x=a,\, i \neq i_{noninit}} \left\langle a, i \cap [T_a^{low}, C), r \right\rangle \cup$$
$$\cup \bigcup_{\forall\langle x,i,r\rangle \in S : x=a,\, i = i_{noninit}} \langle a, i_{noninit}, r\rangle \cup$$
$$\cup \bigcup_{\forall\langle x,i,r\rangle \in S : x \neq a} \left\langle x, i, r \cdot R_{(a<C)} \right\rangle.$$

Для условия типа $a = C$ (переменная равна константе) можно предложить следующую форму функции η:

$$\eta(S, a = C) =$$
$$= \bigcup_{\forall\langle x,i,r\rangle \in S : x=a,\, i \neq i_{noninit}} \left\langle a, i \cap [C, C], r \right\rangle \cup$$
$$\cup \bigcup_{\forall\langle x,i,r\rangle \in S : x=a,\, i = i_{noninit}} \langle a, i_{noninit}, r\rangle \cup$$
$$\cup \bigcup_{\forall\langle x,i,r\rangle \in S : x \neq a} \left\langle x, i, r \cdot R_{(a=C)} \right\rangle.$$

В вершинах с несколькими входными дугами (ϕ-функциях) объединяются кортежи, полученные по всем входным дугам, в соответствии со следующим правилом:

$$\left[\phi(\ldots)\right]^l : S^l = \hat{S}^l.$$

Кортежи для одного объекта с равными интервалами объединяются; вероятность полученного кортежа равна сумме вероятностей исходных кортежей.

Все правила построения уравнений обеспечивают монотонность функций в правых частях уравнений, что гарантирует сходимость алгоритма интервального анализа.

Обнаружение дефектов

Для обнаружения дефектов в программе необходимо проанализировать решение системы уравнений. Анализ решения системы уравнений заключается в сопоставлении полученных интервалов значений с ограничениями для объектов, выраженными с помощью правил обнаружения дефектов.

Дефекты, выявляемые алгоритмами интервального анализа, можно условно разделить на две группы: ошибки переполнения буфера и ошибки отсутствия инициализации. Ошибки переполнения буфера включают обращение к массиву по неверному индексу, разыменование указателя, выведенного за границу массива, и другие.

Ошибка обращения к массиву по неверному индексу возникает в случае индексации по значению, выходящему за границы массива. Данный дефект обнаруживается следующим правилом:

$$[p[m]]^l: \ \exists(p,(a,k))\in\hat{S}^l:$$
$$\left[\left(\min_{\forall g\in\{m^{low},m^{high}\}}(k+g)<0\right)\vee\right.$$
$$\left.\vee\left(\max_{\forall g\in\{m^{low},m^{high}\}}(k+g)\geq a.size\right)\right],$$

где кортеж $(p,(a,k))$ связывает указатель p с k-м элементом объекта a и определяется в результате анализа указателей.

Правило обнаружения ошибок разыменования указателя, выведенного за границу массива, эквивалентно предыдущему случаю при индексе m, равном нулю:

$$[*p]^l: \ \exists(p,(a,k))\in\hat{S}^l:$$
$$[(k<0)\vee(k\geq a.size)].$$

Ошибка использования неинициализированной переменной в арифметических или логических операциях обнаруживается следующим правилом:

$$[a=b\otimes c]^l: \ \exists\langle b,i_{noninit},r\rangle\in$$
$$\in\hat{S}^l\vee\exists\langle c,i_{noninit},r\rangle\in\hat{S}^l,$$

где $\otimes$ — некоторая арифметическая или логическая операция.

Вероятность наличия дефекта

Приведенные правила последовательно применяются к состояниям программы в соответствующих вершинах CFG. Множество кортежей для некоторого объекта, проверяемого в правиле обнаружения дефекта, можно разделить на два непересекающихся подмножества — *DEF* и ¬*DEF*. В подмножество *DEF* входят кортежи, приводящие к срабатыванию правила, в подмножество ¬*DEF* — все остальные кортежи.

Возможны три ситуации:

1) подмножество *DEF* пусто — в рассматриваемой конструкции дефект отсутствует;

2) подмножество ¬*DEF* пусто — дефект гарантированно присутствует в данной конструкции;

3) оба подмножества *DEF* и ¬*DEF* не пусты — существует неопределенность наличия дефекта.

Первый и второй случаи однозначно свидетельствуют об отсутствии и наличии дефекта соответственно. Для принятия решения в третьем случае будем использовать оценку вероятности наличия дефекта для конструкции l, которая рассчитывается следующим образом:

$$\tilde{R}_a^l=\frac{\sum_{\forall\langle x,i,r\rangle\in DEF:x=a} r}{R_a^l}.$$

Полученное значение сравнивается с порогом принятия решения о наличии дефекта $\Pi\in[0, 1]$. При значении Π равном 1 фиксируются только гарантированные дефекты. При нулевом значении порога во всех подозрительных конструкциях программы фиксируются дефекты, что может привести к значительному числу ложных предупреждений. При помощи регулирования порога Π можно добиться баланса между точностью и полнотой получаемых результатов для анализируемой программы.

Одним из вариантов использования порога Π является его постепенное снижение в процессе анализа программы, что обеспечит обнаружение и исправление дефектов, начиная с наиболее вероятных. Возможность регулировки порога обнаружения дефектов может использоваться при анализе программ различного размера и класса.

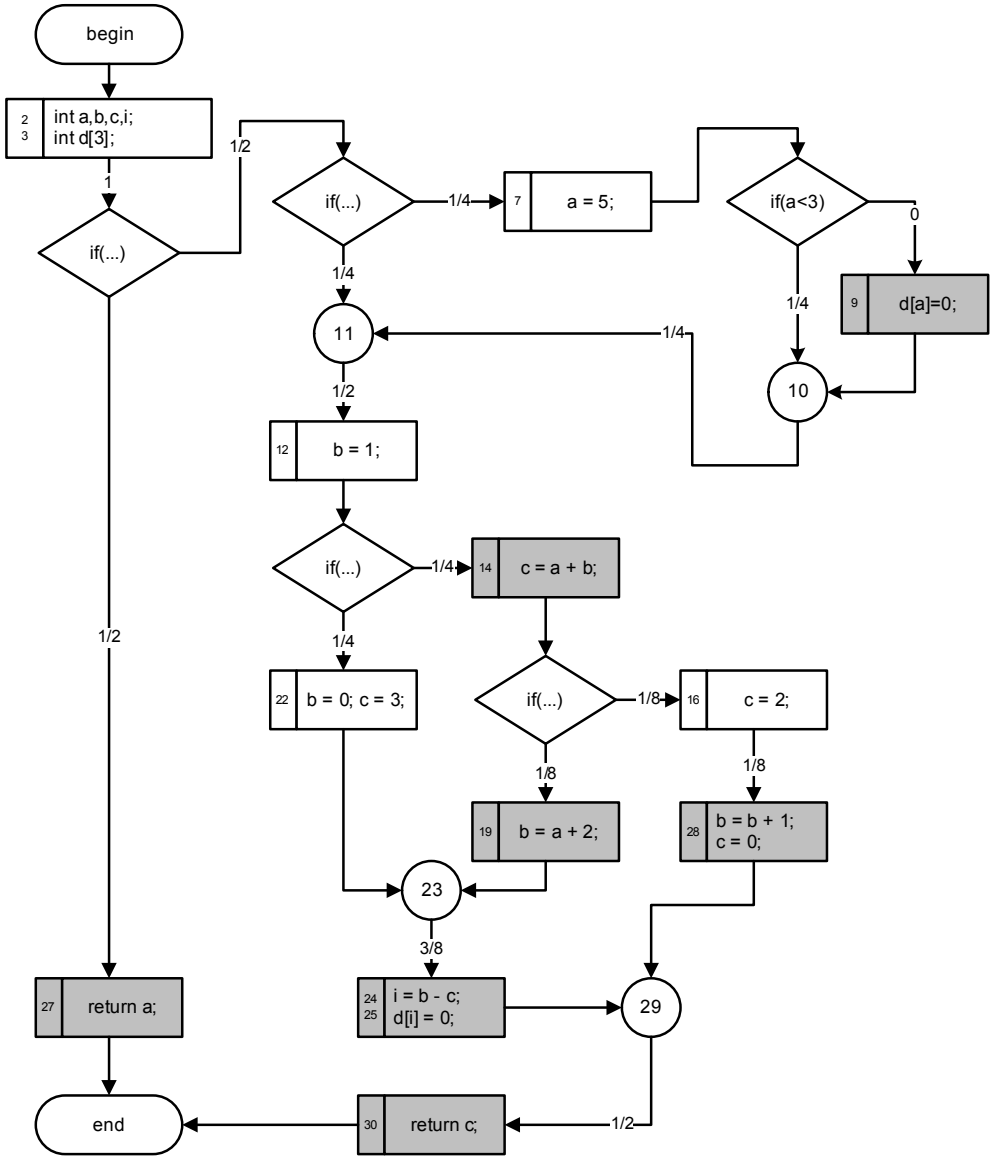
Применение предложенного метода

Продемонстрируем применение описанного алгоритма на примере простой программы, содержащей несколько дефектов.

Пример программы

Условия операторов ветвления, обозначенные многоточием, являются неинтерпретируемыми и не приводятся. В операторе ветвления в строке 8 содержится интерпретируемое условие.

```
 1  int main(void) {
 2      int a, b, c, i;
 3      int d[3];
 4
 5      if (...) {
 6          if (...) {
 7              a = 5;
 8              if (a < 3) {
 9                  d[a] = 0;
10              }
11          }
12          b = 1;
13          if (...) {
14              c = a + b;
15              if (...) {
16                  c = 2;
17                  goto L1;
18              } else {
19                  b = a + 2;
20              }
21          } else {
22              b = 0; c = 3;
23          }
24          i = b - c;
25          d[i] = 0;
26      } else {
27          return a;
28          L1: b = b + 1; c = 0;
29      }
30      return c;
31  }
```



■ *Граф потока управления для примера программы*

Граф потока управления программы представлен на рисунке. Дуги графа помечены оценками вероятностей. В вершинах графа указаны номера соответствующих строк программы. Конструкции программы, которые потенциально могут содержать дефекты, выделены.

Результаты интервального анализа

В результате выполнения базового интервального анализа было получено следующее решение:

$$S^2 = \left\{ \begin{matrix} (a, i_{noninit}, 1), (b, i_{noninit}, 1), \\ (c, i_{noninit}, 1), (i, i_{noninit}, 1) \end{matrix} \right\}$$

$$S^7 = \left\{ \begin{matrix} \left(a, 5, \frac{1}{4}\right), \left(b, i_{noninit}, \frac{1}{4}\right), \\ \left(c, i_{noninit}, \frac{1}{4}\right), \left(i, i_{noninit}, \frac{1}{4}\right) \end{matrix} \right\}$$

$$S^9 = \left\{ (b, i_{noninit}, 0), (c, i_{noninit}, 0), (i, i_{noninit}, 0) \right\}$$

$$S^{10} = \left\{ \begin{matrix} \left(a, 5, \frac{1}{4}\right), \left(b, i_{noninit}, \frac{1}{4}\right), \\ \left(c, i_{noninit}, \frac{1}{4}\right), \left(i, i_{noninit}, \frac{1}{4}\right) \end{matrix} \right\}$$

$$S^{11} = \left\{ \begin{matrix} \left(a, i_{noninit}, \frac{1}{4}\right), \left(a, 5, \frac{1}{4}\right), \left(b, i_{noninit}, \frac{1}{2}\right), \\ \left(c, i_{noninit}, \frac{1}{2}\right), \left(i, i_{noninit}, \frac{1}{2}\right) \end{matrix} \right\}$$

$$S^{12}=\left\{\begin{matrix}(a,i_{noninit},\frac{1}{4}),(a,5,\frac{1}{4}),(b,1,\frac{1}{2}),\\(c,i_{noninit},\frac{1}{2}),(i,i_{noninit},\frac{1}{2})\end{matrix}\right\}$$

$$S^{14}=\left\{\begin{matrix}(a,i_{noninit},\frac{1}{8}),(a,5,\frac{1}{8}),(b,1,\frac{1}{4}),\\(c,i_{noninit},\frac{1}{8}),(c,6,\frac{1}{8}),(i,i_{noninit},\frac{1}{4})\end{matrix}\right\}$$

$$S^{16}=\left\{\begin{matrix}(a,i_{noninit},\frac{1}{16}),(a,5,\frac{1}{16}),(b,1,\frac{1}{8}),\\(c,2,\frac{1}{8}),(i,i_{noninit},\frac{1}{8})\end{matrix}\right\}$$

$$S^{19}=\left\{\begin{matrix}(a,i_{noninit},\frac{1}{16}),(a,5,\frac{1}{16}),(b,i_{noninit},\frac{1}{16}),\\(b,7,\frac{1}{16}),(c,i_{noninit},\frac{1}{16}),\\(c,6,\frac{1}{16}),(i,i_{noninit},\frac{1}{8})\end{matrix}\right\}$$

$$S^{22}=\left\{\begin{matrix}(a,i_{noninit},\frac{1}{8}),(a,5,\frac{1}{8}),(b,0,\frac{1}{4}),\\(c,3,\frac{1}{4}),(i,i_{noninit},\frac{1}{4})\end{matrix}\right\}$$

$$S^{23}=\left\{\begin{matrix}(a,i_{noninit},\frac{3}{16}),(a,5,\frac{3}{16}),(b,i_{noninit},\frac{1}{16}),\\(b,0,\frac{1}{4}),(b,7,\frac{1}{16}),(c,i_{noninit},\frac{1}{16}),\\(c,3,\frac{1}{4}),(c,6,\frac{1}{16}),(i,i_{noninit},\frac{3}{8})\end{matrix}\right\}$$

$$S^{24}=\left\{\begin{matrix}(a,i_{noninit},\frac{3}{16}),(a,5,\frac{3}{16}),(b,i_{noninit},\frac{1}{16}),\\(b,0,\frac{1}{4}),(b,7,\frac{1}{16}),\\(c,i_{noninit},\frac{1}{16}),(c,3,\frac{1}{4}),(c,6,\frac{1}{16}),\\(i,i_{noninit},\frac{11}{96}),(i,-3,\frac{1}{6}),(i,-6,\frac{1}{24}),\\(i,4,\frac{1}{24}),(i,1,\frac{1}{96})\end{matrix}\right\}$$

$$S^{27}=\left\{\begin{matrix}(a,i_{noninit},\frac{1}{2}),(b,i_{noninit},\frac{1}{2}),\\(c,i_{noninit},\frac{1}{2}),(i,i_{noninit},\frac{1}{2})\end{matrix}\right\}$$

$$S^{28}=\left\{\begin{matrix}(a,i_{noninit},\frac{1}{16}),(a,5,\frac{1}{16}),\\(b,2,\frac{1}{8}),(c,0,\frac{1}{8}),(i,i_{noninit},\frac{1}{8})\end{matrix}\right\}$$

$$S^{29}=\left\{\begin{matrix}(a,i_{noninit},\frac{1}{4}),(a,5,\frac{1}{4}),(b,i_{noninit},\frac{1}{16}),\\(b,0,\frac{1}{4}),(b,2,\frac{1}{8}),(b,7,\frac{1}{16}),\\(c,i_{noninit},\frac{1}{16}),(c,0,\frac{1}{8}),(c,3,\frac{1}{4}),(c,6,\frac{1}{16}),\\(i,i_{noninit},\frac{23}{96}),(i,-3,\frac{1}{6}),(i,-6,\frac{1}{24}),\\(i,4,\frac{1}{24}),(i,1,\frac{1}{96})\end{matrix}\right\}$$

Применим описанные правила обнаружения дефектов к полученному решению. Оценки вероятностей наличия дефектов в потенциально опасных конструкциях программы таковы:

$$\tilde{R}^{9}=0,\tilde{R}^{14}=\frac{1}{2},\tilde{R}^{19}=\frac{1}{2},\tilde{R}^{24}=\frac{11}{36},$$
$$\tilde{R}^{25}=\frac{35}{36},\tilde{R}^{27}=1,\tilde{R}^{28}=0,\tilde{R}^{30}=\frac{1}{8}.$$

Полученные результаты позволяют сделать вывод о том, что в строках 9 и 28 дефекты отсутствуют, в строке 27 гарантированно присутствует дефект использования неинициализированной переменной. Принятие решения о наличии дефектов в других конструкциях программы зависит от выбранного порога П. Необходимо отметить, что в строке 25 присутствует дефект с оценкой вероятности, близкой к 1, но при обнаружении только гарантированных дефектов этот дефект будет пропущен. Кроме того, интерпретация условий, реализованная в предложенном алгоритме, позволила предотвратить ложное обнаружение дефекта переполнения буфера в строке 9. Полученные результаты являются корректными, в чем можно убедиться, проанализировав программу вручную.

Заключение

В статье предложен алгоритм, позволяющий обнаруживать такие распространенные дефекты в программах на языке С как переполнение буфера и использование неинициализированной переменной.

К достоинствам представленного алгоритма интервального анализа относится интерпретация условий в операторах ветвления. Интерпретация условий в некоторых случаях позволяет существенно повысить точность анализа. Особенностью алгоритма является формирование оценки вероятности проявления дефекта, позволяющей судить о точности полученного решения.

Сравнение оценок вероятностей наличия дефекта с порогом обнаружения позволяет управлять точностью и полнотой обнаружения дефектов. При этом все гарантированно присутствующие в программе дефекты будут обнаружены, однако часть истинных дефектов, для которых имеется неопределенность, может быть пропущена. Применение алгоритма продемонстрировано на примере.

Основным недостатком представленного алгоритма является потеря точности из-за объединения решений, полученных по отдельным путям. Можно предложить несколько способов преодоления указанного недостатка, в том числе извлечение и анализ зависимостей между объектами программы, проведение точного анализа путей для некоторых фрагментов программы.

Исследование выполнено в рамках работ по государственному контракту № 02.514.11.4081 «Исследование и разработка системы автоматического обнаружения дефектов в исходном коде программного обеспечения» Федерального агентства по науке и инновациям в рамках Федеральной целевой программы «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007–2012 годы».

Литература

1. **ГОСТ Р ИСО / МЭК 9126-93.** Информационная технология. Оценка программной продукции. Характеристики качества и руководство по их применению. — Введ. 01.07.94. М.: Госстандарт России: Изд-во стандартов, 2004. VI. 12 с.
2. **Steensgaard B.** Points-to Analysis in Almost Linear Time // Proc. of the 23rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL'96). ACM, 1996. P. 32–41.
3. **Nielson F., Nielson H. R., Hankin C.** Principles of Program Analysis. Corr. 2nd printing. Berlin: Springer, 2005. 452 p.
4. **Chakraborty S., Kumar R.** Precise Static Type Analysis for Object-Oriented Programs // ACM SIGPLAN Notices. N. Y.: ACM, 2007. Vol. 42, Issue 2. P. 17–26.
5. **http:**//www.securecoding.cert.org/confluence/display/seccode/CERT+C+Secure+Coding+Standard
6. **http:**//www.securityfocus.com
7. **Биркгоф Г.** Теория решеток. М.: Наука, 1984. 568 с.