Формализация определения ошибок при статическом символьном выполнении

В.К. Кошелев <vedun@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. Данная работа посвящена формализации понятия ошибочной ситуации при статическом анализе исходного кода, основанном на символьном выполнении. При использовании методов символьного выполнения для статического анализа необходимо пересматривать критерий выдачи ошибок, так как оригинальный критерий приводит к чрезмерному числу ложных срабатываний. Для решения этой проблемы предлагаются альтернативные определения ошибочной ситуации, сообщающие об ошибке только в том случае, когда она происходит на некотором множестве значений входных переменных. Примерами таких множеств являются, например, множество значений входных переменных, при которых управление пройдет через заданную точку программы, или множество значений, при которых управление пройдет по заданному пути в графе потока управления. В данной работе рассматриваются различные способы задания таких множеств начальных значений. Анализируются полученные таким образом определения ошибочных ситуаций. Приводятся алгоритмы, обнаруживающие данные ошибочные ситуации, а также доказывается их соответствие. Рассматривается практическое применение приведённых определений ошибочных ситуаций, а именно: классификация срабатываний инструментов статического анализа; учет неизвестных контракта вызова анализируемой функции; использование определения ошибочной ситуации в качестве запроса к SMT-решателю для поиска точного решения в соответствии с данными определением.

Ключевые слова: статический анализ; определение ошибочной ситуации; символьное выполнение.

DOI: 10.15514/ISPRAS-2016-28(5)-6

Для цитирования: Кошелев В.К. Формализация определения ошибок при статическом символьном выполнении. Труды ИСП РАН, том 28, вып. 5, 2016, стр. 105-118. DOI: 10.15514/ISPRAS-2016-28(5)-6

1. Введение

В настоящее время при разработке программного обеспечения широко используются методы автоматического поиска дефектов в программном коде. Одним из таких методов является статический анализ исходного кода

программы. Преимуществом статического анализа является возможность обнаруживать дефекты на путях выполнения, не покрытых при тестировании.

К сожалению, в общем случае задачи поиска дефектов при помощи статического анализа неразрешимы в силу теоремы Райса. На практике статические анализаторы ищут частные случаи ошибочных ситуаций. От того, ситуации какого типа обнаруживает статический анализатор, зависит как число пропущенных ошибок, так и количество ложных срабатываний. Анализаторы, не выдающие ложных срабатываний, способны обнаружить лишь простые дефекты. Не пропускающие ошибок анализаторы, напротив, оказываются неприменимыми для анализа промышленных программ из-за большого числа ложных срабатываний. Для эффективного использования статического анализа для промышленных программ необходимо использовать определение ошибочной ситуации, позволяющее обнаружить широкий спектр ошибочных ситуаций, сохраняя при этом высокий процент истинных срабатываний.

Один из сценариев использования статического анализатора предполагает регулярный анализ проектов во время разработки. Для исполнения этого сценария к статическому анализу предъявляется ряд требований. Во-первых, статический анализатор должен быть способен проводить анализ функции, не имея информации о точках её вызова. Это необходимо как для анализа ещё недописанного кода, так и для анализа библиотек. Во-вторых, при определении понятия ошибочной ситуации анализ должен учитывать наличие в коде вызовов неизвестных функций.

К сожалению, в существующих работах по статическому анализу [1] [2] [3], авторы акцентируют внимание на построении анализа, оставляя за скобками определение ошибочной ситуации. Целью данной работы является разработка формальных определений ошибочных ситуаций, позволяющих выдавать предупреждения с различной степенью уверенности относительно неизвестных значений.

Один из вариантов определения ошибочной ситуации, учитывающей контексты вызова и внешние функции, был рассмотрен в работе [4], посвящённой поиску внутрипроцедурной ошибки переполнения буфера. Данная работа формализует и расширяет определение ошибочной ситуации, использующееся в работе [4] для произвольных детекторов ошибок, основанных на методе символьного выполнения. Рассматриваются примеры реальных детекторов ошибок в программах на языке C#, включая доказательство соответствия с данными определениями ошибочных ситуации.

2. Статический анализ с помощью символьного выполнения

Символьное выполнение [5] широко используется для автоматической генерации тестов. В этом случае выполнение начинается с точки входа

программы, а входные данные программы считаются символьными. От символьного выполнения требуется подобрать входные данные, на которых произойдет ошибка. Пусть e – шаг символьного выполнения, P_e – предикат пути для шага e, а Err_e - условие возникновения ошибки на шаге e, тогда наличие ошибки на шаге e задается следующей формулой:

$$(1)\ \exists \bar{s} : P_e(\bar{s}) \wedge Err_e(\bar{s}),\quad \bar{s} \text{ – вектор символьных переменных}$$

Однако задача статического анализа существенно отличается от задачи автоматической генерации ошибочных входных данных. Ключевым отличием является отсутствие необходимости в предоставлении набора данных, на которых произойдет ошибка.

Отсутствие этой необходимости позволяет рассматривать каждую функцию, представленную в программе, в качестве точки входа. В этом случае аргументы анализируемой функции и состояние памяти в момент её вызова параметризируются набором символьных переменных. Данный подход используется в ряде работ, включая [1], [2], [3]. Однако, в этом случае условие наличие ошибки (1) не может быть использовано ввиду чрезмерного количества ложных срабатываний. Рассмотрим следующий пример.

```
1) public class A {
2)     public int X;
3)     public static int Foo(A a, bool five) {
4)       if (five)
5)           return a.X;
6)       else
7)          return 5;
8)     }
9) }
```

Напишем условие ошибки NullReferenceException для точки (5).

$$\exists a, five : five \ \wedge a = null$$

Данная формула имеет решение $a = null, five = true$, однако функция Foo может иметь неявные предусловия, запрещающее переменной a иметь значение $null$. На практике, использование условия ошибки (1) для поиска NullReferenceException приведет к обнаружению ошибки практически в каждой функции.

Статический анализ приходится проводить, не имея полной информации о контракте анализируемой функции, включая её точное предусловие. Из-за чего, предположение о том, что неизвестные переменные никак не связаны между собой, слишком часто нарушается на практике.

Для учета неизвестного контракта функции предлагается ввести гибкое определение ошибки в конкретной точке программы, позволяющее в зависимости от целей анализатора описывать различные классы ошибочных ситуаций.

Предполагается, что анализ проводится над некоторым набором путей выполнения. Множество таких путей удобно описывать при помощи графа развертки. Ациклический ориентированный связанный граф с выделенными истоком и стоком $G' = \langle V', E', v'_{entry}, v'_{exit}\rangle$, будем называть графом развертки для ГПУ $G = \langle V, E, v_{entry}, v_{exit}\rangle$, если найдется функция соответствия $\psi : V' \to V$, такая что выполнено:

$$(2)\ \psi\left(v'_{entry}\right) = v_{entry} \quad \psi(v'_{exit}) = v_{exit}$$

$$(3)\ \langle v', u'\rangle \in E' \to \langle \psi(v'), \psi(u')\rangle \in E$$

$$(4)\ \langle v', u'\rangle \in E' \wedge \langle v', w'\rangle \in E' \wedge u' \neq w' \ \to \psi(u') \neq \psi(w')$$

Графом развертки, например, является дерево выполнений программы, проходящих по обратным рёбрам не более k-раз. Так как размер такого дерева экспоненциально зависит от размера исходной программы, на практике используются ациклические графы. Пример алгоритма построения ациклического графа развертки приведен в работе [6].

Будем говорить, что путь $l = \langle v_0 = v_{entry}, v_1, v_2, \dots, v_n\rangle$ от истока на ГПУ принадлежит к графу развертки G' (обозначим как $\psi(l) \in G'$), если найдется такой путь $l' = \langle v'_0 = v'_{entry}, v'_1, v'_2, \dots, v'_n\rangle$ в G', что $\forall_{i\in[0,n]}\psi(v'_i) = v_i$. Заметим, что исходя из свойства (4), если путь l' существует, то он единственный.

Тогда символьное выполнение должно производить поиск ошибок среди путей $l \in G$, таких, что $\psi(l) \in G'$. Учитывая введённую параметризацию, ход выполнения в рассматриваемой модели полностью определяется значением символьных переменных.

3. Определения ошибочных ситуаций

Пусть Σ – множество значений вектора символьных переменных $\bar{s}$. Тогда рассмотрим множество $\{\Sigma_i\} \subseteq 2^{\Sigma}$, где $\Sigma_i \subseteq \Sigma$. Элемент Σ_i назовём абстракцией, а $\{\Sigma_i\}$ – множеством абстракций. Будем говорить, что в программе в точке B содержится ошибка, если найдется такая абстракция Σ_i, что на всех наборах значений переменных $\sigma \in \Sigma_i$ произойдет ошибка в точке B.

Обозначим как $P_B^{\Sigma_i}(\bar{s})$ формулу от символьных переменных, задающую условие того, что $\bar{s}$ принадлежит абстракции Σ_i и управление дойдет до точки B. Как $Er_B(\bar{s})$ обозначим условие того, что в точке B произойдет ошибка. Тогда дадим определение ошибки для абстракции Σ_i следующим образом следующим образом:

$$(5)\quad Err_B^{\Sigma_i} = \left(\exists \bar{s}\left(P_B^{\Sigma_i}(\bar{s})\right)\right) \wedge \forall \bar{s}\ \left(P_B^{\Sigma_i}(\bar{s}) \rightarrow\ Err_B(\bar{s})\right)$$

Тогда наличие ошибки в точке B определим, как существования абстракции, на которой произойдёт ошибка:

$$(6)\ \exists \Sigma_i :\ Err_B^{\Sigma_i}$$

Данную формулу будем называть общим определением ошибки, $\langle\{\Sigma_i\}, Err_B\rangle$ - определением ошибки, полученным подстановкой $\{\Sigma_i\}$ и Err_B в общее определение ошибки. В зависимости от выбора множества абстракций $\{\Sigma_i\}$ будут различаться множества обнаруживаемых ошибок. Рассмотрим два множества абстракций $\boldsymbol{\Sigma}' = \{\Sigma_i'\}$ и $\boldsymbol{\Sigma}'' = \{\Sigma_j''\}$, $\boldsymbol{\Sigma}', \boldsymbol{\Sigma}'' \in 2^{\Sigma}$ таких, что:

$$(7)\ \forall i\ \exists \bar{\boldsymbol{j}}: \Sigma_i' = \bigcup_{j\in\bar{\boldsymbol{j}}} \Sigma_j''$$

Пусть $Err_B^{\boldsymbol{\Sigma}'}$- множество ошибок в точке B для множества абстракций $\boldsymbol{\Sigma}'$, $Err_B^{\boldsymbol{\Sigma}''}$ - для $\boldsymbol{\Sigma}''$, тогда если (7) верно, то:

$$(8)\ \forall B :\ Err_B^{\boldsymbol{\Sigma}'} \subseteq Err_B^{\boldsymbol{\Sigma}''}$$

Для доказательства (8) заметим, что если верна $Err_B^{\Sigma_i'}$, то для всех $\Sigma_j'' \subseteq \Sigma_i'$, при условии достижимости $\mathrm{P}_{\mathrm{B}}^{\Sigma_j''}(\bar{s})$, будет верна и $Err_B^{\Sigma_j''}$. Так как $P_B^{\Sigma_i'}(\bar{s})$ – выполнима, а $\Sigma_i' = \bigcup_{j\in\bar{\boldsymbol{j}}}\Sigma_j''$, то среди Σ_j'' найдется такой $\Sigma_{j'}''$, что $P_B^{\Sigma_{j'}''}(\bar{s})$ – выполнима. Тогда подставляя $i = j'$ в (6) получим верную формулу.

Таким образом, определение ошибки (6) позволяет проводить сравнение различных методов обнаружения ошибок. Рассмотрим несколько используемых на практике вариантов задания абстракций $\{\Sigma_i\}$.

Рассмотрим множество Σ, пронумеруем все его элементы $\Sigma = \{\sigma_i\}$. Зададим $\Sigma_i = \sigma_i$. Тогда для данного варианта разбиения формула (6) принимает вид:

$$(9)\ \exists \sigma_i: P_B(\sigma_i) \wedge Err_B(\sigma_i)$$

Формула (9) эквивалента формуле (1), т.к. задает значения символьных переменных параметризации. Таким образом формула (6) для разбиения $\Sigma_i = \sigma_i$ формулирует задачу символьного выполнения для автоматической генерации тестов.

С другой стороны, в качестве множества абстракции можно взять один элемент $\Sigma_1 = \Sigma$. Тогда формула (6) примет вид:

$$(10)\ \exists s\left(P_B(\bar{s})\right) \wedge \forall s\ \left(P_B(\bar{s}) \rightarrow\ Err_B(\bar{s})\right)$$

Формула (10) утверждает, что если точка B такова, что она достижима хотя бы на одном конкретном состоянии и в ней всегда происходит ошибка, то точка B ошибочна. Данное определение является самым строгим из рассматриваемых, поэтому ему свойственен пропуск реальных ошибок.

По числу обнаруживаемых ошибок между абстракциями $\Sigma_i = \sigma_i$ и $\Sigma_i = P_B$ находится абстракция путей выполнения. Идея данной абстракции основана на предположении, что контракт функции не запрещает пути выполнения в графе развертки. Иными словами, для каждого пути в графе развертки найдется способ запустить рассматриваемую функцию таким образом, чтобы управление прошло именно на данном пути.

Для построения абстракций путей выполнения, каждому булевому выражению в графе развертки сопоставим свою булеву переменную $\{b_j\}$. По построению, каждая из булевых переменных зависит параметризации, т.е. $b_j = b_j(\bar{\boldsymbol{s}})$, тогда:

$$(11)\ \Sigma_i = \bigwedge_j \begin{cases} b_j, \text{если } \left\lfloor \frac{i}{2^j} \right\rfloor \equiv 0\ (mod\ 2) \\ \neg b_j, \text{ иначе} \end{cases} = \bar{\boldsymbol{b}}_i$$

Таким образом, каждая Σ_i задает значения всех булевых выражений, которые в свою очередь однозначно определяет путь выполнения в графе развертки. Формула (6) для данной абстракции имеет следующий вид:

$$(12)\ \exists \bar{\boldsymbol{b}}: \left(\exists \bar{s}\left(P_B^{\bar{b}}(\bar{s})\right)\right) \wedge \forall \bar{s}\ \left(P_B^{\bar{b}}(\bar{s}) \rightarrow\ Er\ _B(\bar{s})\right)$$

Где $P_B^{\bar{b}}(\bar{s}) = \left(\forall i\ b_i = b_i(\bar{s})\right) \wedge P_B$

Заметим, что для введённых абстракций верно следующее соотношение:

$$(13)\ Err^{\Sigma_i = P_B} \subseteq Err^{\Sigma_i = \bar{\boldsymbol{b}}_i} \subseteq Err^{\Sigma_i = \sigma_i}$$

Между абстракциями $\Sigma_i = P_B$ и $\Sigma_i = \bar{\boldsymbol{b}}_i$ находятся абстракции k критических точек. При использовании абстракции k критических точек точка B считается ошибочным в том случае, если найдутся k точек графа развертки таких, что любой путь проходящий через них и через B приводит к ошибке. Как можно заметить, при достаточно большом числе k, данная абстракция будет эквивалентна абстракции $\Sigma_i = \bar{\boldsymbol{b}}_i$. Рассмотрим случай $k = 1$. Пронумеруем все точки в графе развертки – $\{B_i\}$, тогда $\Sigma_i = P_{B_i}$, а $P_B^{B_i} = P_{B_i} \wedge P_B$. Тогда общая формула ошибки записывается следующим образом:

$$(14)\ \exists B_i: \left(\exists \bar{s}\left(P_B^{B_i}(\bar{s})\right)\right) \wedge \forall \bar{s}\ \left(P_B^{B_i}(\bar{s}) \rightarrow\ Err_B(\bar{s})\right)$$

Учитывая последнюю введённую абстракцию, напишем итоговое соотношение:

$$(15)\ Err^{\Sigma_i = \Sigma} \subseteq Err^{\Sigma_i = P_{Bi}} \subseteq Err^{\Sigma_i = \bar{\boldsymbol{b}}_i} \subseteq Er\ ^{\Sigma_i = \sigma_i}$$

4. Примеры ошибочных ситуаций

Рассмотрим на примерах разницу между представленными выше определениями ошибок.

```
1) public string Example1(object obj)
2) {
3)     obj = null;
4)     return obj.ToString();
5) }

1) public string Example2(object obj, bool a)
2) {
3)     if (a) {
4)         obj = null;
5)     }
6)     return obj.ToString();
7) }

1) public string Example3(object obj, bool a, bool b)
2) {
3)     if (a) {
4)         obj = null;
5)     }
6)     if (b) {
7)          return obj.ToString();
8)     }
9)     return null;
10)}

1) public string Example4(object obj, bool a, bool b, bool c)
2) {
3)    if (a) {
4)        obj = null;
5)    }
6)    if (b) {
7)         obj = new object();
8)    }
9)    if (c) {
10)          return obj.ToString();
11)    }
12)    return null;
13)}
```



```
1) public string Example5(object obj)
2) {
3)    return obj.ToString();
4) }
```

Листинг 1. Примеры различных ошибочных ситуаций.

Listing 1. Examples of defects.

На листинге 1 приведены пять различных потенциально ошибочных ситуаций. В качестве условия ошибки рассмотрим условие $obj == null$. Тогда Example1 будет являться ошибочным для всех абстракций.

Example2 является ошибочным для всех абстракций, кроме $\Sigma_i = \Sigma$. Однако, если в качестве условия ошибки использовать не условие равенства $null$ в точке разыменования (6), а условие разыменования в точке присваивания (4), то определение ошибки при $\Sigma_i = \Sigma$ также обнаружит ошибку.

Example3 также является ошибочным для всех абстракций, кроме $\Sigma_i = \Sigma$, однако в отличие от Example2 определение $\Sigma_i = \Sigma$ не обнаружит ошибку в данном случае при переносе ошибочной точки в присваивание.

В Example4 ошибку обнаружат только абстракции $\Sigma_i = \overline{\boldsymbol{b}}_i$ и $\Sigma_i = \sigma_i$. Абстракция $\Sigma_i = P_{B_i}$ не найдет ошибку, т.к. критической точкой для данной абстракции является точка (4), однако между точками (4) и (10) существует путь (4)-(7)-(10) не содержащий ошибок. Следовательно, по определению (14) данная ситуация не является ошибочной.

Наконец, в Example5 ошибку обнаружит только абстракция $\Sigma_i = \sigma_i$.

5. Примеры алгоритмов поиска ошибочных ситуаций

Рассмотрим классификацию алгоритмов поиска ошибок относительно введённых определений. Рассмотрим некоторый алгоритм поиска ошибок A, обозначим за Err_A множество обнаруживаемых им ошибок для данного графа развертки. Тогда для упорядоченного набора абстракций

$$(16)\ Err^{\Sigma_i^0} = \emptyset \subseteq Err^{\Sigma_i^1} \subseteq Err^{\Sigma_i^2} \subseteq \cdots Err^{\Sigma_i^N} \subseteq Err^{\Sigma_i = \sigma_i}$$

будем говорить, что A относится к классу $Err^{\Sigma_i^k}$, если $Err_A \subseteq Err^{\Sigma_i^k}$, но $Err_A \nsubseteq Err^{\Sigma_i^{k-1}}$. Заметим, что в данном случае допускается пропуск ошибок класса $Err^{\Sigma_i^k}$, однако алгоритм A должен находить некоторый набор ошибок, таких что они принадлежат $Err^{\Sigma_i^k}$, но не содержатся в $Err^{\Sigma_i^{k-1}}$.

Заметим, что набор ошибок, обнаруживаемых алгоритмом A, может не совпадать с набором выдаваемых им предупреждений. Для предупреждения w_A будем говорить, что оно является ошибкой, если $w_A \in Err^{\Sigma_i = \sigma_i}$.

Рассмотрим построенные во второй главе абстракции:

$$(17)\ Err^{\Sigma_i^0} = \emptyset \subseteq Err^{\Sigma_i=\Sigma} \subseteq Err^{\Sigma_i=P_{Bi}} \subseteq Err^{\Sigma_i=\bar{\boldsymbol{b}}_i} \subseteq Err^{\Sigma_i=\sigma_i}$$

Примером алгоритма поиска ошибок, относящихся к классу $\Sigma_i = \Sigma$ является анализ критических рёбер, реализованный в анализаторе Svace [7]. В качестве условия возникновения ошибки для поиска разыменования нулевого указателя в программах на C/C++ используется следующее условие:

$$(18) Err_x = (x = null) \wedge (dereference(x))$$

Где $dereference(x)$ – условие того, что символьное значение x будет использовано в операции разыменование далее в графе развертки. Условие $dereference(x)$ может быть вычислено с помощью обратного анализа графа развертки.

Алгоритм, использующийся в анализаторе Svace для поиска разыменования нулевого указателя, основан на вычислении для каждой точки условий $null(x) = \{T, F\},\ deref(x) = \{T, F\}$. Условие $null(x)$ истинно в том случае, если в данной точке символьная переменная x всегда равна $null$. Соответственно, $deref(x)$ истинен в том случае, если символьная переменная x будет гарантированно разыменована. Тогда, если найдется такая точка B в графе развертки G', что для какой-либо переменной x одновременно верны $null(x) \wedge deref(x)$, то в точке B происходит ошибка.

Покажем, что данный алгоритм относится к классу $\Sigma_i = \Sigma$. Для этого достаточно показать, что если данный алгоритм находит ошибку, то она соответствует определению $\langle \Sigma_i = \Sigma, Err_x \rangle$. Действительно, если для точки B верно одновременно $null(x) \wedge deref(x)$, то на любом пути выполнения будет верно $Err_x = (x = null) \wedge (dereference(x))$. Однако определение $\langle \Sigma_i = \Sigma, Err_x \rangle$ предполагает, что точка B достижима из точки входа. Данный алгоритм не проводит анализ достижимости, поэтому он будет выдавать предупреждения даже в случае недостижимого кода. Предупреждения в недостижимом коде не содержатся в $Err^{\Sigma_i=\sigma_i}$, поэтому они не включаются в общее множество ошибок алгоритма, следовательно ошибки данного алгоритма включены в $\langle \Sigma_i = \Sigma, Err_x \rangle$.

Рассмотрим алгоритм поиска ошибки разыменования нулевого указателя по определению $\langle \Sigma_i = \bar{\boldsymbol{b}}_i, Err_{Deref(x)} = (x = null) \rangle$. Пусть для каждой вершины графа развертки посчитаны условия $Null_B^x(\bar{\boldsymbol{b}})$, такие, что $Null_B^x(\bar{\boldsymbol{b}}) \to x = null$. Запись $Null_B^x(\bar{\boldsymbol{b}})$ означает, что условие ошибки зависит от значений булевых выражений

Алгоритм выдает в точке B ошибку в том случае, если формула разрешима:

$$(19)\ Null_B^x(\bar{\boldsymbol{b}}_i(\bar{\boldsymbol{s}})) \wedge P_B(\bar{\boldsymbol{s}}).$$

Покажем, что все ошибки найденные данным алгоритмом являются ошибками по определению определения $\langle \Sigma_i = \bar{\boldsymbol{b}}_i, Err_{Deref(x)} = (x = null) \rangle$.

Так как формула $Null_B^x(\bar{\boldsymbol{b}}(\bar{\boldsymbol{s}})) \wedge P_B(\bar{\boldsymbol{s}})$ разрешима, то найдется такое $\bar{\boldsymbol{s}}'$, что $Null_B^x(\bar{\boldsymbol{b}}(\bar{\boldsymbol{s}}')) \wedge P_B(\bar{\boldsymbol{s}}')$ - верно. Пусть $\bar{\boldsymbol{b}}' = \bar{\boldsymbol{b}}(\bar{\boldsymbol{s}}')$, тогда подставив в формулу (12) $\bar{\boldsymbol{b}} = \bar{\boldsymbol{b}}'$ получим:

$$(20)\ \Big(\exists \bar{\boldsymbol{s}} \Big((\forall i\ b_i' = b_i'(\bar{\boldsymbol{s}})) \wedge P_B(\bar{\boldsymbol{s}})\Big)\Big) \wedge \forall \bar{\boldsymbol{s}}\ \Big((\forall i\ b_i' = b_i'(\bar{\boldsymbol{s}})) \wedge P_B(\bar{\boldsymbol{s}}) \to x = null\Big)$$

Заметим, что первый конъюнкт формулы (20) верен, т.к. взяв $\bar{\boldsymbol{s}} = \bar{\boldsymbol{s}}'$ получим верное равенство. Осталось доказать, что:

$$(21)\ \forall \bar{\boldsymbol{s}}\ \Big((\forall i\ b_i' = b_i(\bar{\boldsymbol{s}})) \wedge P_B(\bar{\boldsymbol{s}}) \to x = null\Big)$$

Для этого докажем следующее утверждение:

$$(22)\ (\forall i\ b_i' = b_i(\bar{\boldsymbol{s}})) \wedge P_B(\bar{\boldsymbol{s}}) \to\ Null_B^x(\bar{\boldsymbol{b}}) \wedge P_B(\bar{\boldsymbol{s}})$$

Так как $\bar{\boldsymbol{s}}'$ является решением для (20), а $\bar{\boldsymbol{b}}' = \bar{\boldsymbol{b}}(\bar{\boldsymbol{s}}')$, то $Null_B^x(\bar{\boldsymbol{b}}')$ - верно. Условие $(\forall i\ b_i' = b_i(\bar{\boldsymbol{s}}))$ означает, что $\bar{\boldsymbol{b}}' = \bar{\boldsymbol{b}}(\bar{\boldsymbol{s}}')$, отсюда получаем, что $(\forall i\ b_i' = b_i(\bar{\boldsymbol{s}})) \to\ Null_B^x(\bar{\boldsymbol{b}})$. Следовательно, исходная импликация (22) верна. Учитывая, что из $Null_B^x(\bar{\boldsymbol{b}}) \wedge P_B(\bar{\boldsymbol{s}}) \to x = null$, получаем, что формула (21) также верна. Таким образом доказано, что результаты данного алгоритма включены в $\langle \Sigma_i = \bar{\boldsymbol{b}}_i, Err_{Deref(x)} = (x = null) \rangle$.

Количество срабатываний, выдаваемых алгоритмом напрямую зависит от вида условия $Null_B^x$. Вообще говоря, условие $Null_B^x$ может быть построено точно, т.е. результаты такого алгоритма совпадут с $\langle \Sigma_i = \bar{\boldsymbol{b}}_i, Err_{Deref(x)x} = (x = null) \rangle$. На практике, в инструментах Svace [7] и SharpChecher [6], используются неполные реализации, пропускающие часть срабатываний по сравнению $\langle \Sigma_i = \bar{\boldsymbol{b}}_i, Err_{Deref(x)x} = (x = null) \rangle$. Однако, в соответствии с классификацией данные реализации всё равно относятся к $\langle \Sigma_i = \bar{\boldsymbol{b}}_i, Err_{Deref(x)x} = (x = null) \rangle$, т.к. обнаруживают ошибки в Example4.

В качестве примера алгоритма поиска $Err^{\Sigma_i=P_{Bi}}$, может быть использован алгоритм поиска $Err^{\Sigma_i=\bar{\boldsymbol{b}}_i}$, который в случае обнаружения ошибки дополнительно проверяет её принадлежность к классу $Err^{\Sigma_i=P_{Bi}}$.

5. Применения предложенной классификации

Предложенный метод построения определения ошибок может быть использован для введения определений с целью формализации новых подклассов ошибочных ситуаций. Примером такого класса ошибок являются ошибки, происходящие вне зависимости от условия заданного перехода. Мотивацией к введению такого класса ошибок является наличие ситуаций, в которых переход зависит от возвращаемого значения неизвестной функции.

Если ошибка возможно только на одной ветке такого перехода, то ошибку предлагается не выдавать.

Набор различных определений ошибочных ситуаций можно использовать для анализа и сравнения алгоритмов поиска ошибок. Наличие формальных определений ошибочных ситуаций позволяет проводить формальное доказательство соответствия предложенных алгоритмов анализа программ.

Кроме того, предложенные определения ошибок могут сами использованы как алгоритмы поиска ошибок. Построенные формулы, описывающие ошибочные ситуации, могут быть переданы сторонним SMT-решателям, которые проверят их на совместность. В случае, если формула является совместной, SMT-решатель предоставит явное решение, описывающее абстракцию Σ_i, на которой происходит ошибка. Данная абстракция может быть предъявлена пользователю, в качестве примера ошибочной ситуации.

Использование определений в качестве алгоритма поиска ошибок было применено в анализаторе Svace для поиска переполнения буфера. Предварительное тестирование данного подхода показало его применимость для обнаружения реальных ошибок в промышленных программах.

Наконец, использование разных определений ошибочных ситуаций может применяться для ранжирования предупреждений. Благодаря тому, что ошибочные ситуации вложены друг в друга, результаты работы алгоритма могут быть классифицированы по определениям ошибок, как наименьшее по включению определение, содержащее данную ошибку. Данная классификация является осмысленной, поскольку, как правило, меньшие по включению определения имеют более высокий процент истинных срабатываний.

6. Заключение

В данной работе рассмотрен вопрос формализации ошибочных ситуаций для статического символьного выполнения. Рассмотрена общая формула ошибочных ситуаций. Разобраны частные определения ошибочных ситуаций, а также алгоритмы их поиска, использующиеся в анализаторах кода Svace и SharpChecker. Разработана классификация алгоритмов поиска ошибок на основе набора определений ошибочных ситуаций. Для рассматриваемых алгоритмов поиска ошибок проведена их классификация, включая доказательство соответствия. Приведены примеры использования данной формализации на практике.

Список литературы

[1]. Y. Xie, A. Aiken. Saturn: A Scalable Framework for Error Detection Using Boolean Satisfiability ACM Trans. Program. Lang. Syst. 2007. Vol. 29, no. 3.

[2]. F. Ivančić, G. Balakrishnan, A. Gupta at al. Scalable and scope-bounded software verification in Varvel. Automated Software Engineering. 2015. Vol. 22, no. 4. pp. 517–559.

[3]. Babic D., Hu A.J. Calysto. Software Engineering, 2008. ICSE ’08. ACM/IEEE 30th International Conference on. 2008. May. pp. 211–220.

[4]. В.К. Кошелев, И.А. Дудина, В.И. Игнатьев, А.И. Борзилов. Чувствительный к путям поиск дефектов в программах на языке C# на примере разыменования нулевого указателя. Труды ИСП РАН, том 27, вып. 5, 2015 г., стр. 59-86. DOI: 10.15514/ISPRAS-2015-27(5)-5

[5]. J. King. Symbolic Execution and Program Testing. Commun. ACM. 1976. Vol. 19, no. 7. pp. 385–394.

[6]. В. К. Кошелев, В. Н. Игнатьев, А. И. Борзилов. Инфраструктура статического анализа программ на языке C#. Труды ИСП РАН, том 28, вып. 1, 2016 г., стр. 21-40. DOI: 10.15514/ISPRAS-2016-28(1)-2

[7]. В.П. Иванников А.А. Белеванцев А.Е. Бородин В.Н. Игнатьев Д.М. Журихин А.И. Аветисян М.И. Леонов. Статический анализатор Svace для поиска дефектов в исходном коде программ. Труды ИСП РАН, том. 26, вып. 1. pp. 231–250. DOI: 10.15514/ISPRAS-2014-26(1)-7

[8]. И.А. Дудина, В.К. Кошелев, А.Е. Бородин. Поиск ошибок доступа к буферу в программах на языке C/C++. Труды ИСП РАН, том 28, вып. 4, 2016 г., стр. 149-168. DOI: 10.15514/ISPRAS-2016-28(4)-9

Formalization of Error Criteria for static symbolic execution

V.K. Koshelev <vedun@ispras.ru>
Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. This paper is devoted to the formalization of the error criteria for program static analysis, based on symbolic execution. Using the original error criteria of symbolic execution approach in program static analysis leads to an excessive number of false positives. To solve this problem, we propose an alternative definition of the error criteria. Proposed definition reports errors only if they occur on a certain set of input variables. Examples of such sets are the set of values of input variables in which control will pass through a given point of the program, or set of values in which the controls take place along a given path in the control flow graph. This paper discusses the various ways to specify such sets of initial values, including analysis of the final error criteria. We overview algorithms corresponding to the error criteria and prove their correctness. Finally, we consider the practical applications of the given error criteria, which include classification of the warnings generated by static analysis tools; taking into account unknown function contracting, especially preconditions; using the proposed error criteria as formulas for a SMT-solver. The latest application allows to get the precise solution of the particular error criteria, including the error trace.

Keywords: static analysis, error criteria, symbolic execution.

DOI: 10.15514/ISPRAS-2016-28(5)-6

For citation: V.K. Koshelev. Formalization of Error Criteria for static symbolic execution. Trudy ISP RAN/Proc. ISP RAS, 2016, vol. 28, issue 5, 2016, pp. 105-118 (in Russian). DOI: 10.15514/ISPRAS-2016-28(5)-6

References

[1]. Y. Xie, A. Aiken. Saturn: A Scalable Framework for Error Detection Using Boolean Satisfiability ACM Trans. Program. Lang. Syst. 2007. Vol. 29, no. 3.

[2]. F. Ivančić, G. Balakrishnan, A. Gupta at al. Scalable and scope-bounded software verification in Varvel. Automated Software Engineering. 2015. Vol. 22, no. 4. pp. 517–559.

[3]. Babic D., Hu A.J. Calysto. Software Engineering, 2008. ICSE '08. ACM/IEEE 30th International Conference on. 2008. May. pp. 211–220.

[4]. V. Koshelev, I. Dudina, V. Ignatyev, A. Borzilov. [Path-Sensitive Bug Detection Analysis of C# Program Illustrated by Null Pointer Dereference]. Trudy ISP RAN/Proc. ISP RAS, vol. 27, issue 5, 2015. pp. 59-86 (in Russian). DOI: 10.15514/ISPRAS-2015-27(5)-5

[5]. J. King. Symbolic Execution and Program Testing. Commun. ACM. 1976. Vol. 19, no. 7. pp. 385–394

[6]. V. Koshelev, V. Ignatyev, A. Borzilov. C# static analysis framework. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 1, 2016, pp. 21-40 (in Russian). DOI: 10.15514/ISPRAS-2016-28(1)-2

[7]. V.P. Ivannikov, A.A. Belevantsev, A.E. Borodin, V.N. Ignatiev, D.M. Zhurikhin, A.I. Avetisyan, M.I. Leonov. Static analyzer Svace for finding of defects in program source code. Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 1, 2014, pp. 231-250 (in Russian). DOI: 10.15514/ISPRAS-2014-26(1)-7

[8]. I. Dudina, V. Koshelev, A. Borodin. [Statically detecting buffer overflows in C/C++ Proceedings of the Institute for System Programming]. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 4, 2016, pp. 149-168 (in Russian).