Доказательство корректности программ

Кондратьев Дмитрий Александрович

Институт систем информатики им. А. П. Ершова СО РАН
Новосибирский Государственный Университет

Что такое ошибка в программе?

Вместе с программой всегда присутствует еще один объект – спецификация этой программы.

Спецификация описывает то, что должна делать программа.

Если спецификации на первый взгляд не видно, то нужно знать, что спецификация все равно где-то есть (может быть, в Ваших мыслях?).

Ошибка – это когда программа, делает не то, что описывает спецификация этой программы.

Примеры спецификаций программы:

- Речь заказчика ПО (нужно записать в виде текста).
- Задание в задачнике по программированию.
- Текст технического задания на ПО, оформленный по стандарту ГОСТ 19.201-78 (замечательный вариант!).
- Формальная спецификация (самый лучший вариант!)

От тестирования программного обеспечения к формальной верификации программ

Известно, что тестирование не может гарантировать корректность и надежность программного обеспечения.

Следовательно, тестирование не может гарантировать корректность и надежность такого программного обеспечения, которое реализует искусственный интеллект.

Гарантировать корректность и надежности программного обеспечения может только формальная верификация программ.

Формальная верификация программ

Входными данными формальной верификации являются программа и ее спецификации, описывающие те свойства программы, выполнение которых нужно проверить.

Формальная верификация позволяет доказать, что программа корректна относительно своих спецификаций.

Основные виды формальной верификации:

- дедуктивная верификация
- проверка на модели (model checking).

Актуальность формальной верификации искусственного интеллекта

Формальную верификацию можно применить и к программному обеспечению, реализующему искусственный интеллект, чтобы сделать искусственный интеллект доверенным.

В отличие от тематики формальной верификации программного обеспечения общего назначения, тематика формальной верификации искусственного интеллекта стала активно развиваться только последние несколько лет.

Итого, проблема формальной верификации искусственного интеллекта, основанного на нейронных сетях, является актуальной.

Дедуктивная верификация программного обеспечения

Дедуктивная верификация позволяет свести задачу проверки корректности программы относительно спецификаций к задаче проверки истинности утверждений о том, что программа корректна относительно спецификаций

Такие утверждения являются логическими формулами и называются условиями корректности программы

Если все условия корректности являются истинными, то программа корректна относительно спецификаций

Дедуктивная верификация основана на генерации условий корректности верифицируемой программы

Входными данными для генерации условий корректности являются верифицируемая программа и ее спецификации

Истинность условий корректности проверяется с помощью программных систем для доказательства теорем

Этапы дедуктивной верификации программ

1. Задание спецификаций верифицируемой программы
2. Генерация условий корректности для верифицируемой программы и ее спецификаций
3. Проверка истинности условий корректности с помощью программных систем для доказательства теорем (например, с помощью SMT-решателей Z3, CVC5 или с помощью интерактивных систем доказательства теорем Coq, ACL2, Isabelle/HOL)
4. Если все условия корректности истинны, то верифицируемая программа корректна относительно спецификаций.
5. Если истинность какого-либо из условий корректности не удается доказать или удается доказать ложность какого-либо из условий корректности, то наступает этап локализации ошибок.

Рассмотрим основной этап дедуктивной верификации программ, этап генерации условий корректности

Пример задания спецификаций программы

Верифицируемая программа:

$$\text{if } (\text{x} == 0)\ \{\text{y} = 1;\}\ \text{else}\ \{\text{y} = 2;\}$$

Предусловие (логическая формула, описывающая ограничения на входные данные программы):

$$\mathit{true}$$

Постусловие (логическая формула, описывающая ограничения на связь входных и выходных данных программы):

$$((x = 0) \to (y = 1)) \wedge ((x \neq 0) \to (y = 2))$$

Генерация условий корректности программ

Генерация условий корректности программ основана на логике Хоара

В логике Хоара программа и ее спецификации представляются в виде тройки Хоара:

$$\{P\}\ S\ \{Q\}$$

где

- P — предусловие (логическая формула)
- S — программа
- Q — постусловие (логическая формула)

Частичная корректность тройки Хоара $\{P\}\ S\ \{Q\}$ означает, что "если предусловие P истинно перед исполнением фрагмента программы S, и, если исполнение S завершилось, тогда постусловие Q выполняется после его завершения".

Логика Хоара

Генерация условий корректности основана на правилах вывода логики Хоара. Схема правила вывода в логике Хоара:

$$\frac{\{P_1\}\ S_1\ \{Q_1\},\ \ldots\ \{P_n\}\ S_n\ \{Q_n\},\ \gamma_1,\ \ldots\ \gamma_m}{\{P\}\ S\ \{Q\}}$$

где

- $\{P_1\}\ S_1\ \{Q_1\},\ \ldots\ \{P_n\}\ S_n\ \{Q_n\}$ — n посылок правила вывода в виде троек Хоара
- $\gamma_1,\ \ldots\ \gamma_m$ — m посылок правила вывода в виде условий корректности (логических формул).
- $\{P\}\ S\ \{Q\}$ — заключение правила вывода (тройка Хоара)

Генерация условий корректности происходит в направлении от заключений правил вывода к посылкам правил вывода. Генерация условий корректности порождает дерево и останавливается, когда все листья полученного дерева представляют собой условия корректности.

Правила вывода для базовых конструкций языка

Правило вывода для пустой программы:

$$\frac{P \rightarrow Q}{\{P\}\text{emptyProgram}\{Q\}}$$

где $\rightarrow$ обозначает импликацию.

Правило вывода для присваивания переменной:

$$\frac{\{P\}\ \text{prog};\ \{Q(var \leftarrow expr)\}}{\{P\}\ \text{prog};\ \text{var=expr}\ \{Q\}}$$

где $Q(var \leftarrow expr)$ обозначает замену всех вхождений var в Q на $expr$, prog обозначает остальную часть программы.

Правило вывода для *if*:

$$\frac{\{P \wedge B\}\ \text{S}_1;\ \text{prog}\ \{Q\},\ \{P \wedge \neg B\}\ \text{S}_2;\ \text{prog}\ \{Q\}}{\{P\}\ \text{if B then S}_1\ \text{else S}_2;\ \text{prog}\ \{\text{Q}\}}$$

где $\wedge$ обозначает конъюнкцию, $\neg$ – отрицание.

Такие правила вывода задают формальную семантику конструкций языка программирования

Пример генерации условий корректности

$$\{true\}\ \text{if}\ (\text{x} == 0)\ \{\text{y} = 1;\}\ \text{else}\ \{\text{y} = 2;\}$$
$$\{((x = 0) \to (y = 1)) \wedge ((x \neq 0) \to (y = 2))\}$$

Используем правило вывода для *if*

- $\{true \wedge (x = 0)\}\ \text{y} = 1;$
  $\{((x = 0) \to (y = 1)) \wedge ((x \neq 0) \to (y = 2))\}$
  Используем правило вывода для присваивания
  $\{true \wedge (x = 0)\}\{((x = 0) \to (1 = 1)) \wedge ((x \neq 0) \to (1 = 2))\}$
  Используем правило вывода для пустой программы и получим условие корректности: $(true \wedge (x = 0)) \to (((x = 0) \to (1 = 1)) \wedge ((x \neq 0) \to (1 = 2)))$
- $\{true \wedge (x \neq 0)\}\ \text{y} = 2;$
  $\{((x = 0) \to (y = 1)) \wedge ((x \neq 0) \to (y = 2))\}$
  Используем правило вывода для присваивания
  $\{true \wedge (x \neq 0)\}\{((x = 0) \to (2 = 1)) \wedge ((x \neq 0) \to (2 = 2))\}$
  Используем правило вывода для пустой программы и получим условие корректности: $(true \wedge (x \neq 0)) \to (((x = 0) \to (2 = 1)) \wedge ((x \neq 0) \to (2 = 2)))$

Формальная спецификация

Формальная спецификация – математические формулы, описывающие связь входных и выходных данных программы.

Рассмотрим задание формальной спецификации на примере аварии миссии «Луна-25».

Отчет Роскосмоса (https://www.roscosmos.ru/39790/):

«наиболее вероятной причиной аварии «Луны-25» стало нештатное функционирование бортового комплекса управления ... из-за возможного попадания в один массив данных команд с различными приоритетами их исполнения прибором».

Поможем Роскосмосу избежать подобных аварий в будущем!

Наша цель – чтобы все миссии Роскосмоса в будущем были успешными

Формальная спецификация на примере «Луна-25»

Пусть приоритетами являются значения элементов в массиве команд.

Неправильно сработала программа, проверяющая, все ли элементы массива равны.

Зададим формальную спецификацию такой программы.

Пусть возвращаемое значение программы, проверяющей, равны ли все элементы, хранится в переменной `result`.

Зададим спецификацию в виде формулы, описывающей значение выходной переменной `result`.

Спецификация, описывающая значение выходной переменной `result`

Пусть массив команд a имеет длину n. Тогда формула, описывающая значение выходной переменной `result` имеет вид

$$((\forall j\ ((0 \leq j < n) \rightarrow a[0] == a[j])) \rightarrow result == 1) \wedge (\exists j\ (0 \leq j < n \wedge a[0] != a[j])) \rightarrow result == 0)$$

где

- $\forall$ обозначает "для всех";
- $\exists$ обозначает "существует".
- $\rightarrow$ обозначает импликацию "если ... то".
- $\wedge$ обозначает конъюнкцию "и".

Спецификация, описывающая значение входных переменных

Тогда формула, описывающая ограничение на длину массива имеет вид

$$0 < n$$

Предусловие и постусловие

Спецификация программы, проверяющей, все ли элементы массива равны, состоит из двух формул:

- Формула, описывающая свойства входных данных программы
- Формула, описывающая свойства выходных данных программы

Формулы таких же двух видов возникают при верификации и других программ.

Формула, описывающая свойства входных данных программы, называется предусловием программы и обозначается P.

Формула, описывающая свойства входных данных программы, называется предусловием программы и обозначается Q.

Верифицируемая программа на языке C:

```
int element_equality(int n, int* a)
{
    int result = 1;
    int i = 1;
    while ((i < n) && (result == 1))
    {
        if (a[0] != a[i])
        {
            result = 0;
        }
        i = i + 1;
    }
    return result;
}
```

Обычно ошибки в программе ищут тестированием. Но тестирование не позволяет доказать корректность (невозможно проверить все пути исполнения цикла программы).

От тестирования к дедуктивной верификации программ

Недостаток дедуктивной верификации программ: необходимость задавать формальные спецификации в виде математических формул (в отличие от тестирования).

Преимущества дедуктивной верификации программ: доказательство корректности программ (в отличие от тестирования).

Дедуктивная верификация программ

Дедуктивная верификация позволяет свести задачу проверки корректности программ относительно формальных спецификаций к задаче доказательства истинности специальных формул. Такие формулы называются условиями корректности программы.

Этапы дедуктивной верификации программ:

1. Задание формальных спецификаций программ.
2. Генерация условий корректности на основе программы и ее формальных спецификаций.
3. Доказательство условий корректности. На этом этапе можно использовать программные инструменты для автоматического доказательства теорем, например, Z3.

Рассмотрим, как сгенерировать условия корректности на основе программы и ее спецификаций.

Генерация условий корректности

Условие корректности имеет вид:

$$P \rightarrow wp(S, Q)$$

где

- P – предусловие;
- S – программа;
- Q – постусловие;
- wp – специальная функция (генератор условий корректности), зависящая от программы и постусловия.

Генератор условий корректности программ

$$wp(S_1; S_2,\ Q) = wp(S_1, wp(S_2, Q))$$

$$wp(emptyProgram,\ Q) = Q$$

$$wp(var = rval,\ Q) = Q(var \leftarrow rval))$$

$$wp(if\ B\ S_1\ else\ S_2,\ Q) =$$
$$(B \rightarrow wp(S_1,\ Q)) \wedge (\neg B \rightarrow wp(S_2,\ Q))$$

Генератор условий корректности для цикла

$$wp(\mathit{while}\ B\ \mathit{inv}\ I\ \mathit{do}\ S,\ Q) = I \wedge$$
$$(\forall y(B \wedge I \rightarrow wp(S, I))[v \leftarrow y]) \wedge (\forall y(\neg B \wedge I \rightarrow Q)[v \leftarrow y])$$

где

I – инвариант цикла

Инвариант цикла для программы, проверяющей, все ли элементы массива равны

$$
\begin{gathered}
((\forall j\ ((0 \leq j < i) \rightarrow a[0] == a[j])) \rightarrow result == 1 \\
\wedge \\
(\exists j\ (0 \leq j < i \wedge a[0]! = a[j])) \rightarrow result == 0 \\
\wedge \\
i \leq n)
\end{gathered}
$$

Условия корректности для программы, проверяющей, все ли элементы массива равны

Первое условие корректности:

$$
\begin{gathered}
0 < n \\
\rightarrow \\
((\forall j\ ((0 \leq j < 1) \rightarrow a[0] == a[j])) \rightarrow 1 == 1 \\
\wedge \\
(\exists j\ (0 \leq j < 1 \wedge a[0] != a[j])) \rightarrow 1 == 0 \\
\wedge \\
0 \leq n)
\end{gathered}
$$

Условия корректности для программы, проверяющей, все ли элементы массива равны

Второе условие корректности:

$$
\begin{gathered}
0 < n \\
\rightarrow \\
((((\forall j\ ((0 \leq j < i) \rightarrow a[0] == a[j])) \rightarrow result == 1 \\
\wedge \\
(\exists j\ (0 \leq j < i \wedge a[0]! = a[j])) \rightarrow result == 0 \wedge \\
i \leq n) \\
((i \geq n) \vee (result == 0))) \\
\rightarrow \\
((\forall j\ ((0 \leq j < n) \rightarrow a[0] == a[j])) \rightarrow result == 1 \\
\wedge \\
(\exists j\ (0 \leq j < n \wedge a[0]! = a[j])) \rightarrow result == 0))
\end{gathered}
$$

Третье условие корректности для программы, проверяющей, все ли элементы массива равны

$$
\begin{gathered}
0 < n \\
\rightarrow \\
((((\forall j\ ((0 \leq j < i) \rightarrow a[0] == a[j])) \rightarrow result == 1 \\
\wedge \\
(\exists j\ (0 \leq j < i \wedge a[0] != a[j])) \rightarrow result == 0 \wedge \\
i \leq n) \\
((i < n) \wedge (result == 1)) \\
a[0] == a[i]) \\
\rightarrow \\
((\forall j\ ((0 \leq j < i + 1) \rightarrow a[0] == a[j])) \rightarrow result == 1 \\
\wedge \\
(\exists j\ (0 \leq j < i + 1 \wedge a[0] != a[j])) \rightarrow result == 0 \\
\wedge \\
i + 1 \leq n))
\end{gathered}
$$

Четвертое условие корректности для программы, проверяющей, все ли элементы массива равны

$$
\begin{gathered}
0 < n \\
\rightarrow \\
((((\forall j\ ((0 \leq j < i) \rightarrow a[0] == a[j])) \rightarrow result == 1 \\
\wedge \\
(\exists j\ (0 \leq j < i \wedge a[0] != a[j])) \rightarrow result == 0 \wedge \\
i \leq n) \\
((i < n) \wedge (result == 1)) \\
a[0] \neq a[i]) \\
\rightarrow \\
((\forall j\ ((0 \leq j < i + 1) \rightarrow a[0] == a[j])) \rightarrow 0 == 1 \\
\wedge \\
(\exists j\ (0 \leq j < i + 1 \wedge a[0] != a[j])) \rightarrow 0 == 0 \\
\wedge \\
i + 1 \leq n))
\end{gathered}
$$

Итоги дедуктивной программы, проверяющей, все ли элементы массива равны

Так как все четыре условия корректности истинны (являются теоремами), то программа корректна относительно своих спецификаций.

Контест VeHa-2023

Два соревнования:

- Соревнование по дедуктивной верификации.
- Соревнование по model checking.

Тематика данных соревнований покрывает два главных направления формальной верификации программ:

- Дедуктивная верификация.
- Model checking.

Рассмотрим соревнование по дедуктивной верификации программ.

Ближайший аналог – соревнование VerifyThis

Задача по дедуктивной верификации программы, предназначенной для решения такой проблемы миссии "Луна-25", как проверка равенства всех приоритетов в массиве команд

Цель задачи состоит в том, чтобы задать формальные спецификации программы, предназначенной для решения такой проблемы миссии "Луна-25", как проверка равенства всех приоритетов в массиве команд, и провести автоматическую дедуктивную верификацию данной программы в системе C-lightVer.

Формальная верификация реализации хэш-функции «Стрибог» с «Группой Астра»

Приглашаются желающие участвовать в проекте "Формальная верификация реализации хэш-функции «Стрибог» с «Группой Астра» который запланирован 6-19 июля 2025 года на Большой Математической Мастерской в Новосибирском Академгородке. Задача проекта – выработать подход к построению формального автоматически проверяемого доказательства функциональной корректности одной из реализаций на языке C хэш-функции «Стрибог». Подробная информация о проекте доступна по следующей ссылке:
`https://bmm.mca.nsu.ru/project/116`

Литература

1. **Статья о нашей системе дедуктивной верификации C-lightVer:**
   Kondratyev D.A., Nepomniaschy V.A. Automation of C Program Deductive Verification without Using Loop Invariants // Programming and Computer Software. 2022. Volume 48. Issue 5. pp. 331–346. DOI: https://doi.org/10.1134/S036176882205005X
2. **Статья о нашем решении проблемы автоматизации дедуктивной верификации:**
   Kondratyev D. Implementing the Symbolic Method of Verification in the C-Light Project // Lecture Notes in Computer Science. 2018. Volume 10742. pp. 227–240. DOI: https://doi.org/10.1007/978-3-319-74313-4_17
3. **Статья о нашем решении проблемы инвариантов циклов:**
   Kondratyev D.A., Maryasov I.V., Nepomniaschy V.A. The Automation of C Program Verification by the Symbolic Method of Loop Invariant Elimination // Automatic Control and Computer Sciences. 2019. Volume 53. Issue 7. pp. 653–662. DOI: https://doi.org/10.3103/S0146411619070101
4. **Статья о нашем решении проблемы автоматизации доказательства условий корректности:**
   Kondratyev D., Maryasov I., Nepomniaschy V. Towards Automatic Deductive Verification of C Programs over Linear Arrays // Lecture Notes in Computer Science. 2019. Volume 11964. pp. 232–242. DOI: https://doi.org/10.1007/978-3-030-37487-7_20

Доказательство корректности программ

Кондратьев Дмитрий Александрович

Институт систем информатики им. А. П. Ершова СО РАН
Новосибирский Государственный Университет