

Теория вычислительных процессов

Process calculus

Введение. Верификация программ

ИВТ и ПМ
ЗабГУ

2021

Outline

Введение

Верификация программ

Контрактное программирование

Вопросы

Ссылки

github.com/ivtipm/ProcessCalculus

Примерная структура курса

- ▶ Верификация программ
- ▶ Сети Петри
- ▶ Параллельное и асинхронное программирование

Теоретическое программирование

Теоретическое программирование (Programming language theory) – математическая дисциплина, изучающая математические абстракции программ, трактуемых как объекты, выраженные на формальном языке, обладающие определенной информационной и логической структурой и подлежащие исполнению на автоматических устройствах.

Основные направления теоретического программирования

- ▶ Математические основы программирования.
- ▶ Теория схем программ *строение программы, взаимодействие составляющих ее компонентов*
- ▶ Семантическая теория программ *методы формального описания семантики программ, семантические методы преобразования и доказательства утверждений о программах*
- ▶ Теория вычислительных процессов и структур (теория параллельных вычислений) *разработку и обоснование новых методов программирования, прежде всего методов программирования параллельных процессов*
- ▶ Прикладные задачи теоретического программирования *разработка и обоснование алгоритмов трансляции и алгоритмов автоматической оптимизации программ*

Outline

Введение

Верификация программ

Контрактное программирование

Вопросы

Ссылки

Верификация программ

Верификацией называется проверка соответствия программы (или некоторого промежуточного результата проектирования: прототипа, модели и т.п.) предъявляемым к ней требованиям (проектной документации).

Верификация программ

Зачем?

- ▶ В 1982 г. канадской фирмой Atomic Energy of Canada Limited был запущен в серию медицинский аппарат «Therac-25», предназначенный для лучевой терапии — облучения раковой опухоли. В «редких» ситуациях из-за ряда ошибок, допущенных при проектировании и реализации встроенной системы управления, интенсивность облучения возрастала на 2 порядка и более. За время эксплуатации прибора (период с июня 1985 г. по январь 1987 г.) как минимум два пациента умерли, а несколько человек остались инвалидами

Верификация программ

Зачем?



Верификация программ

Зачем?



Верификация программ

Зачем?

- ▶ 4 июня 1996 г. на 39-й секунде после старта взорвалась ракета-носитель «Ариан 5», разработанная Европейским космическим агентством. Бортовая система частично использовала ПО предыдущей версии ракеты, «Ариан 4», в частности модуль управления инерциальной навигационной системой⁴. При выполнении преобразования из 64-битного числа с плавающей точкой, представляющего наклон ракеты, в 16-битное целое возникло переполнение, которое не было обработано модулем. При разработке модуля предполагалось, что переполнение невозможно из-за физических ограничений ракеты, но прогресс, как известно, не стоит на месте — в «Ариан 5» использовались новые двигатели.

Верификация программ

Зачем?

- ▶ 13 марта 2019 года компания Boeing приостановила полеты самолетов Boeing 737 MAX из-за двух авиакатастроф. Причиной аварий стали ошибки в программном обеспечении системы управления полетом самолета.

Boeing во время аудита обнаружила ошибки в обновленном ПО для самолетов модели 737 MAX

Верификация программ

Корректность программ критически важна в

- ▶ системах управления опасным производством (например атомные станции)
- ▶ медицинские устройства
- ▶ бортовые системы управления (самолёты, космические аппараты, автомобили)
- ▶ электронная коммерция

Верификация программ

Способы проверки корректности программы

- ▶ Инспекция кода – просмотр и рецензирование исходных кодов
- ▶ Статический анализ
- ▶ Тесты
Не гарантируют 100% корректность программы. Тесты относительно просто реализовать на практике.
- ▶ Доказательство корректности математическими методами.
Гарантирует абсолютную корректность. На практике труднее в использовании чем тесты (не вполне поддается автоматизации)

Верификация программ

Далее в примерах будут сделаны упрощения (использован простой язык программирования while):

- ▶ Нет оператора goto
- ▶ Только целый тип данных
- ▶ Нет массивов и других сложных типов данных
- ▶ Нет функций и процедур
- ▶ Есть любые нужные «операции»

Верификация программ

Рассматриваемые операторы

- ▶ Пустой оператор: **skip**. Ничего не делает.
- ▶ Оператор прерывания **abort**, аварийно прерывает работу программы; результат работы программы не определён.
- ▶ Оператор присваивания $x := E$, где x – переменная, E – арифметическое выражение.
- ▶ Последовательное выполнение операторов
begin $S1$; $S2$ **end**
где $S1$ и $S2$ – произвольные операторы.
Если не возникает двусмысленности, операторные скобки **begin** и **end** можно опускать: $S1 ; S2 ; \dots ; S_n$.

Верификация программ

Рассматриваемые операторы

- ▶ Условный оператор

if B **then** S1 **else** S2

B – логическое выражение.

Краткую форма условного оператора `if B then S1,`
считая ее сокращением полной формы `if B then S 1`
`else skip`

- ▶ Оператор цикла

while B **do** S

B – условное выражение, а S – оператор тела цикла

Состояния

- ▶ **Состояние** программы – значение её переменных.
- ▶ Программа начинает работу в **начальном состоянии**
- ▶ Программа заканчивает работу в **конечном состоянии**
- ▶ Результат работы может оказаться неопределенным
*программа «зациклилась», выполнила abort, попыталась
вычислить некорректное значение*

Состояния

- ▶ программу можно рассматривать как функцию, которая преобразует начальное состояние переменных в конечное
- ▶ функция может быть неопределенной на некоторых состояниях
- ▶ функция может быть неоднозначной (не детерминированной)

Состояния

Состояний у программы может быть большое количество

Удобнее рассматривать не отдельные состояния, а **множества состояний**

Множество состояний описывается предикатом, аргументы которого – переменные

Состояния

Пример

- ▶ предикат $P(x, y) = x < y$ описывает множество состояний переменных программы, в которых значение переменной x меньше значения переменной y .
- ▶ Если в программе используются три переменные x , y и z , то состояние $x=3, y=4, z=0$ входит в множество P
- ▶ состояние $x=5, y=4, z=2$ в него не входит

Спецификация программы

Спецификация программы (тройка Хоара):

$$P\{S\}Q$$

- ▶ P – предусловие
- ▶ S – программа
- ▶ Q – постусловие

программа удовлетворяет спецификации $P\{S\}Q$, если программа S такова, что если она начинает свою работу в некотором состоянии, принадлежащем множеству состояний P , то она непременно закончит работу, причем заключительное состояние будет принадлежать множеству состояний Q

Спецификация

- ▶ Пара PQ называется **программным контрактом**.
- ▶ Если верна некоторая спецификация программы, то верна также и спецификация с более сильным предусловием и спецификация с более слабым постусловием

Аксиоматическая семантика

Далее будет описана так называемая аксиоматическая семантика – способ формализации программы, позволяющей анализировать её корректность.

Сильные и слабые предикаты

- ▶ Предикат X **сильнее**, чем Y , если Y является логическим следствием X т. е. $(X \rightarrow Y)^1$

Например $x > 0$ слабее чем $(x = 1) \vee (x = 2) \vee (x = 3)$

a	b	$a \rightarrow b, a \leq b$
0	0	1
0	1	1
1	0	0
1	1	1

¹символом \rightarrow обозначается импликация

Аксиоматическая семантика

$P(x) = (x = 1) \vee (x = 2) \vee (x = 3)$ – сильный предикат

$Q(x) = x > 0$ – слабый предикат

$P(x)$	$Q(x)$	$P(x) \rightarrow Q(x)$
0	0	1
0	1	1
1	0	0
1	1	1

Сильные и слабые предикаты

- ▶ Предикат X **сильнее**, чем Y , если Y является логическим следствием X т. е. $(X \rightarrow Y)^2$
Например $x > 0$ слабее чем $(x = 1) \vee (x = 2) \vee (x = 3)$
- ▶ чем слабее предикат тем большее число состояний программы он характеризует
- ▶ Пустому множеству состояний соответствует тождественно ложный предикат, обозначаемый F .
- ▶ Это самый сильный из всех возможных предикатов, поскольку $F \rightarrow P$ верно для любого P

²символом \rightarrow обозначается импликация

Задачи верификации

- ▶ если задана некоторая спецификация $P\{S\}Q$, то можно задаться целью проверить ее истинность
- ▶ если задана программа S и предусловие P , то можно попытаться найти какое-либо постусловие Q такое, чтобы спецификация $P\{S\}Q$ была истинной. Следует искать наиболее сильное постусловие.
- ▶ если задана программа S и постусловие Q , то можно попытаться найти (слабейшее) предусловие P такое, что спецификация $P\{S\}Q$ будет истинной.

weakest precondition

Если задана программа S и постусловие Q , то слабое предусловие для этой программы и этого постусловия может быть получено с помощью специальной функции – преобразователя предикатов ***wp (weakest precondition)***

которая и должна выдавать требуемый результат

$$P = wp(S, Q)$$

Далее приведём аксиомы описывающие правила вывода предусловия P

Аксиомы

- ▶ Аксиома исключенного чуда:
 $wp(S, F) \Leftrightarrow F$ для любого оператора S .
не существует никакого состояния, начавшись в котором программа может гарантированно завершить свою работу в «никаком» состоянии.
- ▶ $wp(skip, Q) \Leftrightarrow Q$ для любого предиката Q .
Семантика пустого оператора: пустой оператор и в самом деле не меняет состояния переменных.
- ▶ $wp(abort, Q) \Leftrightarrow F$ для любого предиката Q .
Семантика оператора прерывания. Действительно, не существует такого состояния, начав работу в котором оператор прерывания смог бы завершить работу в некотором состоянии, поэтому данная аксиома также «разумна».

Аксиомы

► $wp(S, Q_1 \wedge Q_2) \Leftrightarrow wp(S, Q_1) \wedge wp(S, Q_2).$

аксиома позволяет получать слабое предусловие для конъюнкции постусловий, если известны слабые предусловия для отдельных членов этой конъюнкции. Обосновать эту аксиому можно, исходя из смысла преобразователя предикатов wp .

► $wp(S, Q_1) \vee wp(S, Q_2) \Rightarrow wp(S, Q_1 \vee Q_2).$

Если программа не детерминированная, то эквивалентности (\Leftrightarrow) может и не быть.

► $wp(\text{begin } S_1; S_2 \text{ end}, Q) \Leftrightarrow wp(S_1, wp(S_2, Q))$

Аксиомы

Оператор присваивания

- ▶ меняет состояние переменных программы таким образом, что ровно одна переменная меняет свое значение
- ▶ оператор присваивания $x := E$ всегда заканчивается нормально за исключением случаев, когда правая часть не может быть вычислена.
- ▶ $D(E)$ – область определения E – предикат, задающий множество состояний переменных, в каждом из которых правая часть присваивания вычисляется нормально.

$$wp(x := E, Q) \Leftrightarrow D(E) \wedge Q\{x/E\}$$

$Q\{x/E\}$ обозначает результат подстановки в предикат Q выражения E вместо всех вхождений переменной x .

Аксиомы

Оператор присваивания. Пример

Рассмотрим простой оператор $x := x + 1$
и постусловие $x > 0$.

Тогда согласно аксиоме:

$$wp(x := x + 1, x > 0) = D(x + 1) \wedge x + 1 > 0$$

Аксиомы

Оператор присваивания. Пример

Рассмотрим простой оператор $x := x + 1$
и постусловие $x > 0$.

Тогда согласно аксиоме:

$$wp(x := x + 1, x > 0) = D(x + 1) \wedge x + 1 > 0$$

Область определения выражения $x + 1$ есть условие $x < \text{maxint}$,
где maxint – максимальное целое значение.

Аксиомы

Оператор присваивания. Пример

Рассмотрим простой оператор $x := x + 1$
и постусловие $x > 0$.

Тогда согласно аксиоме:

$$wp(x := x + 1, x > 0) = D(x + 1) \wedge x + 1 > 0$$

Область определения выражения $x + 1$ есть условие $x < \text{maxint}$,
где maxint – максимальное целое значение.
искомое слабейшее предусловие может быть задано
предикатом $-1 < x < \text{maxint}$

Аксиомы

Условный оператор

$$\begin{aligned} & wp(\text{if } B \text{ then } S_1 \text{ else } S_2, Q) \\ \Leftrightarrow & (B \rightarrow wp(S_1, Q)) \wedge (\neg B \rightarrow wp(S_2, Q)) \end{aligned}$$

Символом \rightarrow обозначается импликация ($a \rightarrow b$ то же самое что и $\neg a \vee b$)

Аксиомы

Условный оператор. Пример

$$\begin{aligned}wp(M, z = \max(x, y)) &\Leftrightarrow \\wp(M, z = x \wedge x \geq y \vee z = y \wedge y \geq x) &\Leftrightarrow \\(x < y \rightarrow wp(z := y, z = x \wedge x \geq y \vee z = y \wedge y \geq x)) \wedge \\(x \geq y \rightarrow wp(z := x, z = x \wedge x \geq y \vee z = y \wedge y \geq x)) &\Leftrightarrow \\(x \geq y \vee y = x \wedge x \geq y \vee y = y \wedge y \geq x) \wedge \\(x < y \vee x = x \wedge x \geq y \vee x = y \wedge y \geq x) &\Leftrightarrow \\x \geq y \vee x = y \vee y \geq x\end{aligned}$$

- Последний полученный предикат, очевидно, всегда истинен.

Аксиомы

Условный оператор. Пример

$$\begin{aligned}wp(M, z = \max(x, y)) &\Leftrightarrow \\wp(M, z = x \wedge x \geq y \vee z = y \wedge y \geq x) &\Leftrightarrow \\(x < y \rightarrow wp(z := y, z = x \wedge x \geq y \vee z = y \wedge y \geq x)) \wedge \\(x \geq y \rightarrow wp(z := x, z = x \wedge x \geq y \vee z = y \wedge y \geq x)) &\Leftrightarrow \\(x \geq y \vee y = x \wedge x \geq y \vee y = y \wedge y \geq x) \wedge \\(x < y \vee x = x \wedge x \geq y \vee x = y \wedge y \geq x) &\Leftrightarrow \\x \geq y \vee x = y \vee y \geq x\end{aligned}$$

- ▶ Последний полученный предикат, очевидно, всегда истинен.
- ▶ Значит, слабейшим предусловием для данной программы является тождественно истинный предикат Т

Аксиомы

Оператор цикла

while B do S

- ▶ Определяется последовательность предикатов P_i , задающие слабойшие предусловия, при которых цикл завершится не более чем за i шагов, и результирующее состояние будет удовлетворять постусловию Q
- ▶ $P_0 = \neg B \wedge Q$ (цикл завершается за 0 шагов)
- ▶ $P_{i+1} = P_0 \vee (B \wedge wp(S, P_i))$ (цикл завершается не более, чем за $i+1$ шаг)

Аксиомы

Оператор цикла

$$\text{wp (while B do S, Q)} \Leftrightarrow \exists i (i \geq 0 \rightarrow P_i)$$

Аксиомы

Оператор цикла

$$\text{wp (while B do S, Q)} \Leftrightarrow \exists i (i \geq 0 \rightarrow P_i)$$

должно найтись такое неотрицательное i , что цикл завершится не более чем за i шагов, причем заключительное состояние будет удовлетворять условию Q

Аксиомы

Оператор цикла

$$\text{wp} (\text{while } B \text{ do } S, Q) \Leftrightarrow \exists i (i \geq 0 \rightarrow P_i)$$

должно найтись такое неотрицательное i , что цикл завершится не более чем за i шагов, причем заключительное состояние будет удовлетворять условию Q

применить аксиому непосредственно в большинстве случаев невозможно, поскольку невозможно выписать всю последовательность условий, да еще и сформулировать, когда по крайней мере одно из этих условий будет истинным

Аксиомы

Оператор цикла. Пример

`while x < 5 do x := x + 1`

постусловие $x = 5$

Какое будет слабейшее предусловие?

Аксиомы

Оператор цикла. Пример

`while $x < 5$ do $x := x + 1$`

постусловие $x = 5$

Какое будет слабейшее предусловие?

$x \leq 5$

Аксиомы

Оператор цикла. Пример

Доказательство:

$$P_0 = (x = 5)$$

Докажем (по индукции), что $P_i = (5 - i \leq x \leq 5)$

Пусть теперь утверждение справедливо для P_i .

Докажем, что тогда оно справедливо и для P_{i+1} .

Действительно, поскольку $P_{i+1} = P_0 \wedge (B \wedge wp(S, P_i))$, то по индукционному предположению

$P_{i+1} = (x = 5) \wedge (x < 5 \wedge 5 - i \leq x + 1 \leq 5)$, что и в самом деле эквивалентно выражению $5 - (i + 1) \leq x \leq 5$.

Теперь очевидно, что истинное P_i найдется при некотором i тогда и только тогда, когда $x \leq 5$.

Теорема о цикле

позволяет «угадать» предусловие и доказать, что оно и в самом деле может служить предусловием в спецификации цикла

если существуют предикат I (называемый в дальнейшем инвариантом цикла) и целочисленная функция f от переменных программы, принимающая только неотрицательные значения (называемая в дальнейшем функцией спуска) такие, что выполнены следующие 4 условия.

1. $I \wedge \neg B \Rightarrow Q$
 2. $I \wedge B \Rightarrow wp(S, I)$
 3. $f = 0 \Rightarrow \neg B$
 4. $f = V \wedge B \Rightarrow wp(S, f < V)$
- то $I \Rightarrow wp(\mathbf{while\ B\ do\ S}, Q)$.

Теорема о цикле

Пример

Для цикла `while $x < 5$ do $x := x + 1$` и постусловия

- ▶ $x = 5$ можно на основании теоремы доказать, что условие $x \leq 5$ действительно является предусловием цикла, если рассматривать его как инвариант цикла.
- ▶ условие (1) превращается в $x \leq 5 \wedge x \geq 5 \rightarrow x = 5$,
- ▶ условие (2) превращается в $x \leq 5 \wedge x < 5 \rightarrow x + 1 \leq 5$,
- ▶ в качестве функции спуска можно взять функцию $f(x) = |5 - x|$

Outline

Введение

Верификация программ

Контрактное программирование

Вопросы

Ссылки

Контрактное программирование

Контрактное программирование (design by contract (DbC)) – метод проектирования при котором для всех частей системы определяются точные, проверяемые, формальные спецификации.

- ▶ Идею контрактного программирования предложил Бертран Мейер в 90-х.
- ▶ Впервые реализовано в языке программирования Eiffel³, разработанного Мейером

³используется в точи числе Northrop-Grumman

Контрактное программирование

- ▶ *Клиент* вызывает метод (функцию) *поставщика*.
- ▶ **предусловия** (preconditions) – конкретные обязательства клиента перед вызовом метода
- ▶ **постусловия** (postconditions) – конкретные свойства, которые должны присутствовать после выполнения метода (обязательства поставщика)
- ▶ обязательства поставщика по выполнению конкретных свойств – **инвариантов** (invariants)

Контрактное программирование

- ▶ Если на вход поступили данные удовлетворяющие входному условию контракта, то метод гарантирует соблюдение выходного условия
- ▶ Если входные данные НЕ удовлетворяют первому условию, то ничего не гарантируется

Контрактное программирование

Примеры

anadea.info/ru/blog/contract-na-kachestvo

eiffel.org/doc/solutions/Design_by_Contract_and_Assertions

Контрактное программирование

Инварианты могут быть описаны не только для отдельных методов или функций но и для класса целиком.

Примеры

Контрактное программирование

- ▶ контрактное программирование предписывает начинать писать код с написания формальных утверждений корректности (assertions).
- ▶ Утверждения о предусловиях, инвариантах и постусловиях могут быть записываются в комментарии к методу или функции, как часть документации
- ▶ Затем кодируется проверка утверждений
- ▶ При нарушении контракта выполняется принцип fail-fast

Контрактное программирование

В объектно-ориентированном программировании контракт метода обычно включает следующую информацию:

- ▶ возможные типы входных данных и их значение;
- ▶ типы возвращаемых данных и их значение;
- ▶ условия возникновения исключений, их типы и значения;
- ▶ присутствие побочного эффекта метода;
- ▶ предусловия, которые могут быть ослаблены (но не усилены) в подклассах;
- ▶ постусловия, которые могут быть усилены (но не ослаблены) в подклассах;
- ▶ инварианты, которые могут быть усилены (но не ослаблены) в подклассах;
- ▶ (иногда) гарантии производительности, например, временная сложность или сложность по памяти

Контрактное программирование

Когда использовать?

- ▶ На этапе разработки и отладки программы
- ▶ Программа подготовленная для релиза не должна допускать нарушения контракта
- ▶ В этой версии программы контракты отключают для повышения производительности

Контрактное программирование

Поддержка в языках программирования

- ▶ C++. С помощью библиотеки [Boost.Contracts](#), ...
- ▶ C# (и другие языки платформы .NET). Code Contracts
- ▶ Java. [JML \(Java Modeling Language\)](#), [oval](#), ...
- ▶ Python [python-contracts](#), ...

Некоторые языки ориентированные на надёжность ПО

- ▶ Ada 2012

- ▶ SPARK (основан на Ada 2012)

используется в авиации (реактивные двигатели Rolls-Royce Trent, самолёты Eurofighter Typhoon и Бе-200, система UK NATS iFACTS) и для разработки космических систем (РН Вега, множество спутников). Также его применяют для разработки систем шифрования и кибербезопасности

- ▶ ACSL – ANSI/ISO C Specification Language

```
/*@ requires \valid(p);  
   @ assigns *p;  
   @ ensures *p == \old(*p) + 1;  
   @*/  
void incrstar (int *p);
```

The contract is given by the comment which starts with `/*@`. Its meaning is as follows:

the first line is a precondition: it states that function `incrstar` must be called with a pointer `p` that points to a safely allocated memory location. Second line is a frame clause, stating that function `incrstar` does not modify any memory location but the one pointed to by `p`. Finally, the `ensures` clause is a postcondition, which specifies that the value `*p` is incremented by one

Outline

Введение

Верификация программ

Контрактное программирование

Вопросы

Ссылки

Вопросы

- ▶ Что такое теоретическое программирование?
- ▶ Что такое верификация?
- ▶ Что такое тройка Хоара?
- ▶ Что такое состояние?
- ▶ Что такое предусловие и постусловие?
- ▶ Как произвести формальную верификацию?
- ▶ Что означает фраза: предикат X сильнее чем Y ? Что это означает для состояний?
- ▶ Верно ли что, для заданной программы и постусловий нужно искать слабое постусловие? А что на счёт предусловий?
- ▶ Что такое контрактное программирование?

Outline

Введение

Верификация программ

Контрактное программирование

Вопросы

Ссылки

- ▶ Курс лекций: Теория вычислительных процессов, Иркутский государственный университет путей сообщения, 2005.
- ▶ cc.dvo.ru/uchebnaya-literatura.html – Список литературы и слайды «Теория вычислительных процессов и структур»

Текущая тема:

- ▶ Введение в формальные методы верификации программ: учебное пособие / А. С. Камкин. – Москва: МАКС Пресс, 2018. – 272 с. (параграф 2.2.2. Аксиоматическая семантика и др.)
- ▶ хабр: Блог компании PVS-Studio

Материалы дисциплины
github.com/ivtipm/ProcessCalculus