Верификация: РК 1

2015-03-14

Содержание

1 Понятие верификации программного обеспечения

см http://www.ict.edu.ru/ft/005645/62322e1-st09.pdf, п. 1.1. обратите внимание на отличие терминов верификация и валидация. номера стандартов наизусть знать не надо :)

Верификация проверяет соответствие между нормами стандартов, описанием требований (техническим заданием) к ПО, проектными решениями, исходным кодом, пользовательской документацией и функционированием самого ПО. (проверка заданного ПО формально заданным требованиям.)

Валидация проверяет соответствие любых создаваемых или используемых в ходе разработки и сопровождения ПО артефактов нуждам и потребностям пользователей и заказчиков этого ПО, с учетом законов предметной области и ограничений контекста использования ПО.

2 Классификация методов верификации. Примеры конкретных способов, программные реализации в каждом из классов

TODO: программные реализации конкретных методов – добавлены, можно проверить

1. **Статические** – не требуют выполнения программы.

   (a) Экспертиза (code-review)

   (ПО: code collaborator, gerrit, merge-реквесты в гите и другие штуки)

   (b) Автоматический анализ style guide

   (ПО: встроенные плагины во всякие IDE, astyle – юзали в курсаче по smtp)

   Достоинства:

   - автоматизируются
   - время выполнения

   Недостатки:

   - большое число ложных срабатываний

2. **Формальные**

   (a) Дедуктивный анализ

   (ПО: Vampire и другие theorem-proovers)

   (b) Проверка моделей

   (ПО: SPIN(Promela), Java Pathfinder)

   Достоинства:

   - надежность
   - автоматизируются

- нет ложных срабатываний

Недостатки:

- вычислительная сложность
- время

3. **Динамические**

   (a) Мониторинг — контроль корректности работы программы в ходе её обычной работы. Частный случай — профилирование (измерение производительности).

   (ПО: valgrind, intel vtune (профилировщики), Graphite (реалтаймовый отрисовщик графиков), системные мониторы (диспетчер задач, top))

   (b) Тестирование — проверка на основе классов эквивалентности данных.

   (ПО: JUnit (фреймворки для юнит-тестов), генераторы юнит-тестов, например Pex — встроенная в VS тулза)

Достоинства:

- малое время верификации
- нет ложных срабатываний

Недостатки:

- ненадёжно
- низкий уровень автоматизации (относится к тестированию)
- большие временные затраты на составление тестов

3 Определение временной логики

Модальность — некоторая категория, характеризующая высказывание. Например, введём модальность F — «когда-нибудь в будущем», то Fполучено$_m$ будет трактоваться как «когда-нибудь в будущем сообщение m будет получено».

Модальная логика — классическая логика, расширенная некоторыми операторами модальности.

Темпоральные логики — логики, в которых истинностное значение логических формул зависит от момента времени, в котором вычисляются значения этих формул.

Основная идея — фиксировать *относительный порядок событий* (текущее, будущее и прошедшее время).

Предикат — это высказывание, принимающее значение либо «истина», либо «ложь» в зависимости от значений его аргументов.

Атомарный предикат — это утверждение, принимающее значение либо «истина», либо «ложь», от структуры которого мы абстрагируемся.

Темпоральные операторы:

- **F** $Fp : \exists t' \geq t : p \models t'$

  Пример: $\neg F \neg p = Gp$ — в будущем не будет такого момента, что p не верно

- **G** $Gp : \forall t' \geq t : p \models t'$

  Примеры:

  FGp — когда-то в будущем будет всегда истинно p

  GFp — в будущем p будет истинно бесконечное число раз

  $G(persil \Rightarrow Gpersil)$ — «Раз персил — всегда персил!»

- **U** (*Until*) — «до тех пор, пока не наступит нечто».

  Утверждение pUq истинно в момент времени t, если q истинно в некоторый будущий момент времени t', а во всём промежутке от t до t' истинно p: *когда-нибудь q, а до тех пор p.*

- **X** ($NextTime$)

  Утверждение Xq истинно в момент времени t, если q истинно в следующий момент $t+1$.

4 Синтаксис CTL и LTL формул

LTL — темпоральная логика линейного времени. Разработана для спецификации свойств параллельных технических систем. Оперирует атомарными предикатами.

Определение Формула темпоральной логики выполняется на последовательности миров, если истинна в начальном мире этой последовательности.

Определение Формула линейной темпоральной логики LTL это:

- атомарное утверждение (атомарный предикат): $p, q, ...$
- формулы LTL, связанные логическими операторами $\neg, \vee$
- формулы LTL, связанные темпоральными операторами X, U.

Прошлое в логике LTL не рассматривается.

$\phi ::= p | \neg\phi | \phi \vee \phi | X\phi | \phi U \phi$

Остальные операторы выводимы:

- $true = p \vee \neg p$
- $false = \neg true$
- $\phi_1 \wedge \phi_2 = \neg(\neg\phi_1 \vee \neg\phi_2)$
- $\phi_1 \Rightarrow \phi_2 = \neg\phi_1 \vee \phi_2$
- $F\phi = trueU\phi$
- $G\phi = \neg F \neg\phi$

Примеры формул:

- "Пока живу, надеюсь": $G(\text{i\_live} \Rightarrow \text{i\_hope})$
- "Мы будем бороться, пока не победим": $(\text{we\_fight}U\text{we\_win}) \vee G\text{we\_fight}$
- "Сегодня он играет джаз, а завтра Родину продаст": $\text{play\_jazz} \Rightarrow X\text{betray}$

  или $G(\text{play\_jazz} \Rightarrow XF\text{betray})$ — если когда-нибудь начнёт играть джаз, то потом, рано или поздно, продаст Родину.

CTL* — расширенная темпоральная логика ветвящегося времени. В ней в дополнение к U и X введён квантор пути E. Т.о. в ней могут быть как формулы пути, так и формулы состояния.

Определение Формулы CTL* — это формулы состояний, они задаются следующей грамматикой:

- $\Phi ::= p|q|...$ — все атомарные предикаты
- $\Phi ::= \neg\Phi|\Phi \vee \Phi$ — формулы состояний
- $\Phi ::= E\Phi$ — формулы пути

Формулы пути задаются:

- $\phi ::= \Phi$ — формулы состояний
- $\phi ::= \neg\phi|\phi \vee \phi$ — связанные булевыми операторами
- $\phi ::= X\phi|\phi U\phi$ — связанные темпоральными операторами

Для удобства вводится квантор пути **A** (All): $A\phi = \neg E\neg\phi$ — "На всех путях из данного состояния формула пути ϕ истинна".

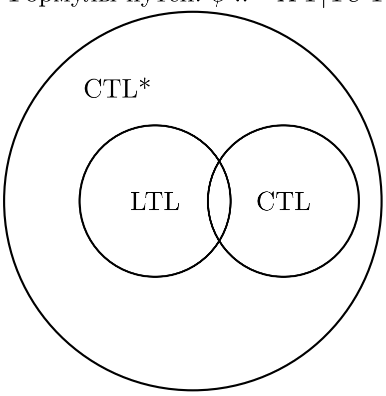
Также используются **F** и **G** (выводятся из **U**).

Любую формулу ϕ в логике LTL можно представить в логике CTL*, добавив квантор **A**: $A\phi$.

CTL — логика ветвящегося времени. Подмножество CTL*. Допустимы только формулы, в которых каждый темпоральный оператор (**X**, **U**, **F**, **G**) предваряется квантором пути (**E** или **A**). Это превращает каждую формулу пути в формулу, характеризующую состояние.

Формулы состояний: $\Phi ::= p|\neg\Phi|\Phi \vee \Phi|E\phi|A\phi$

Формулы путей: $\phi ::= X\Phi|\Phi U\Phi|F\Phi|G\Phi$

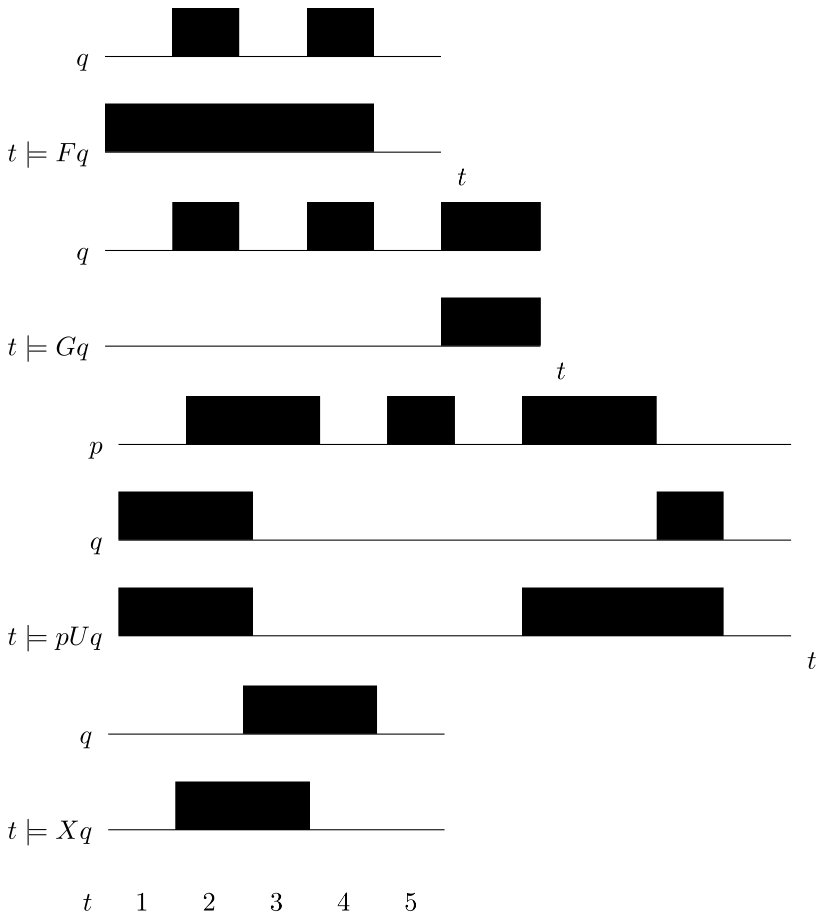


LTL — формулы пути. CTL — формулы состояний.

Примеры:

- $A(aU\neg b)$ — и CTL, и LTL
- $AG(p \Rightarrow Fq)$ — LTL
- $AG(p \Rightarrow EFq)$ — CTL

5 Временные диаграммы отношений F, G, U, X



6 Определение структуры Крипке

$$M = (S, S_0, R, AP, L)$$

S — конечное непустое множество состояний;

$S_0 \subseteq S$ — непустое множество начальных состояний;

$R = \{(s_1, s_2) : s_1, s_2 \in S\}$ — множество отношений, $(\forall s \in S)(\exists s' \in S)(s, s') \in R$ — из любого состояния есть переход;

$AP = \{p, q, r, t, ...\}$ — конечное множество атомарных предикатов;

$L : S \to 2^{AP}$ — функция пометок (разметки) для каждого состояния задает множество истинных в нем атомарных предикатов.

Определение. Бесконечная цепочка $\sigma = q_1 q_2 q_3 ...$ называется *вычислением структуры Крипке*, если $q_1 \in S_0$ и $\forall i \in \mathbb{N}$ $(q_i, q_{i+1}) \in R$. Отображение, в котором любому натуральному число сопоставляется состояние структуры Крипке.

Определение. Индуцированная вычислением $\sigma = q_1 q_2 q_3 ...$, бесконечная цепочка $L(\sigma) = L(q_1)L(q_2)L(q_3)...$ называется *траекторией структуры Крипке*. Отображение натуральных чисел в множество всех подмножеств атомарных предикатов.

Отличие от конечного автомата:

- Каждое состояние помечено множеством атомарных утверждений, истинных в этом состоянии.
- Отсутствуют пометки на переходах.
- Из каждого состояния существует хотя бы один переход (если это завершающее состояние - то в себя же, т.к. оперирует бесконечными вычислениями).

7 Определение автомата Бюхи

особое внимание уделите допускающим состояниям автомата: это главное отличие от КА

Автомат Бюхи можно считать недетерминированным КА, получающим на вход бесконечные слова. В отличие от КА, условие допустимости входной цепочки изменено так, чтобы можно было определять некоторые бесконечные цепочки (ω-слова) как допустимые.

$$B = (S, \Sigma, S_0, \delta, F)$$

- S — множество состояний;
- Σ — конечное множество элементов (алфавит);
- $S_0 \subseteq S$ — множество начальных состояний;
- $\delta : S \times \Sigma \to 2^S$ — отношение переходов;
- $F \subseteq S$ — множество допускающих состояний

Любое ω-слово над алфавитом Σ можно считать функцией $w : N \to \Sigma$, где $w(i)$ — i-ая буква. $\pi : N \to S$ — бесконечная последовательность состояний автомата, $\pi(i)$ — i-ое состояние. $inf(\pi)$ — множество элементов из S, которые встречаются в цепочке π бесконечно много раз.

Определение ω-слово w допускается, если существует такое π, что:

- $\pi(0) \in S_o$
- $\forall i >= 0 : \pi(i+1) \in \delta(\pi(i), w(i))$
- $\inf(\pi) \cap F \neq \varnothing$ — среди бесконечно повтояющихся состояний существует хотя бы одно допускающее

8 Алгоритм верификации CTL-формул на структуре Крипке.

здесь нужен алгоритм целиком для какого-то базиса CTL

Идея: для каждой подформулы заданной формулы определить, в каких состояниях заданной структуры Крипке эта подформула выполняется.

Для каждой подформулы:

1. Вычисляем множество состояний S_Ψ, в которых выполняется формула Ψ.
2. Вводим новый атомарный предикат p_Ψ.
3. Помечаем все состояния из S_Ψ этим предикатом.

Базис: EX, AF, EU

for all $i : 0 < i <= |\Phi|$ **do**

  for all $\Psi \in Sub(\Phi) : |\Psi| = i$ — для всех подформул глубиной i **do**

    switch Ψ

      case p**:** $Sat_\Psi := \{s \in S | p \in L(s)\}$

      case $\neg p$**:** $Sat_\Psi := \{s \in S | p \notin L(s)\}$

      case $p \vee q$**:** $Sat_\Psi := \{s \in S | p \in L(s) \vee q \in L(s)\}$

      case EXp**:** $Sat_\Psi := Sat\_EX(p)$

      case AFp**:** $Sat_\Psi := Sat\_AF(p)$

      case $E(pUq)$**:** $Sat_\Psi := Sat\_EU(p, q)$

    for all $s : s \in Sat_\Psi$ **do**

      $L(s) := L(s) \cup \{p_\Psi\}$

    end for

  end for

end for

function SAT_EX(p) — возвращает множество состояний, на которых выполняется EXp

  $Y := \{s | (\exists s_1 : s \to s_1) p \in L(s1)\}$ — состояния, из которых есть переход в состояние, помеченное p

  return Y

end function

function SAT_AF(p) — возвращает множество состояний, на которых выполняется AFp

  $X := S$

  $Y := \{s \in S | p \in L(s)\}$ — состояния, помеченные p

  while $X \neq Y$ **do**

    $X := Y$

    $Y := Y \cup \{s | (\forall s_1 : s \to s_1) s_1 \in Y\}$ — добавить состояния, все преемники которых уже находятся в Y

  end while

  return Y

end function

function Sat_EU(p, q) — возвращает множество состояний, на которых выполняется $E(pUq)$

$W := \{s \in S | p \in L(s)\}$ — состояния, помеченные p

$X := S$

$Y := \{s \in S | q \in L(s)\}$ — состояния, помеченные q

while $X \neq Y$ **do**

$X := Y$

$Y := Y \cup (W \cap \{s | (\exists s_1 : s \to s_1) s_1 \in Y\})$ — добавить состояния, помеченные p и из которых есть переход в состояние, уже находящееся в Y

end while

return Y

end function

9 Общий алгоритм верификации LTL-формул.

Нужно знать весь алгоритм целиком, кроме самой сложной части — построение автомата Бюхи по LTL-формуле. Достаточно просто знать, что это возможно.

1. По заданной структуре Крипке M строится автомат Бюхе B_M, допускающий все возможные траектории в M;

   function make_buchi_machine($M = (S_M, S_{0M}, R, AP, L)$)

   $S_B := S_M$ — состояния M не изменяются;

   $S_{0B} := S_{0M}$ — множество начальных состояний M не изменяется;

   $\Sigma := 2^{AP}$ — алфавит автомата B_M содержит все подмножества атомарных предикатов M;

   $F := S_M$ — все состояния автомата B_M являются допускающими;

   $\delta := \varnothing$

   for all $(s, s') \in R$ **do**

   $\delta(s, L(s)) := s'$ — здесь $L(s) \in 2^{AP}$, значит $L(s) \in \Sigma$

   end for

   return $(S_B, \Sigma, S_{0B}, F, \delta)$

   end function

2. По LTL-формуле φ строится «контрольный» автомат Бюхи $B_{\neg\varphi}$, допускающий все ω-слова, на которых выполняется $\neg\varphi$;

3. Строится синхронная композиция автоматов $B_\otimes = B_M \otimes B_{\neg\varphi}$;

   function $\otimes(A = (S_A, \Sigma_A, S_{0A}, \delta_A, F_A),\ B = (S_B, \Sigma_B, S_{0B}, \delta_B, F_B))$

   $S_\otimes := \{(s_A, s_B) : s_A \in S_A \wedge s_B \in S_B\}$

   $S_{0\otimes} := \{(s_{0A}, s_{0B}) : s_{0A} \in S_{0A} \wedge s_{0B} \in S_{0B}\}$

   $\Sigma_\otimes := \Sigma_A \cup \Sigma_B$

   $\delta_\otimes := \varnothing$

for all $s_\otimes = (s_A, s_B) \in S_\otimes$ **do**

for all $a \in \Sigma_\otimes$ **do**

if $\delta_A(s_A, a) = s'_A \wedge \delta_B(s_B, a) = s'_B$ **then**

$\delta_\otimes(s_\otimes, a) := (s'_A, s'_B)$

end if

end for

end for

$F_\otimes := \{(s_{fA}, s_{fB}) : s_{fA} \in F_A \wedge s_{fB} \in F_B\}$

return $(S_\otimes, \Sigma_\otimes, S_{0\otimes}, \delta_\otimes, F_\otimes)$

end function

4. Формула φ выполняется для $M \iff L_{B_\otimes} = \varnothing$ (автомат $B_\otimes$ проверяется на наличие цикла, достижимого из начального состояния и включающего допускающее состояние, если такой цикл существует, то он выдается за контрпример — вычисление, не удовлетворяющее формуле φ).