ВПнМ/Теормин

Материал из ESyr's Wiki.

Содержание

- 3.2 Подходы к верификации программ. Тестирование и имитационное моделирование. Область применения, плюсы и минусы. Проблема полноты тестового покрытия.
  - 3.2.1 Проблема полноты тестового покрытия
- 3.3 Подходы к верификации программ. Доказательство теорем. Область применения, плюсы и минусы.
- 3.4 Подходы к верификации программ. Статический анализ исходного кода программ. Область применения, плюсы и минусы.
- 3.5 Подходы к верификации программ. Верификация программ на моделях. Процесс верификации программы при помощи её модели. Область применения, плюсы и минусы.
- 3.6 Верификация на моделях. История развития верификации программ на моделях. Схема верификации программ на моделях. Классы проверяемых свойств правильности программы.
- 3.7 Верификация при помощи Spin. Задание свойств состояний.
- 3.8 Верификация при помощи Spin. Задание свойств последовательностей состояний. Циклы бездействия. Ограничения справедливости.
- 3.9 Верификация при помощи Spin. Задание свойств последовательностей состояний. Утверждения о невозможности. Трассовые ассерты.
- 3.10 Верификация при помощи Spin. Принцип верификации нарушения свойств. Контрпримеры. Процесс верификации при помощи Spin. Использование LTL в Spin.

- 4 Система Spin и язык Promela
  - 4.1 Система Spin. Процесс моделирования и верификации при помощи системы Spin. Конечность моделей на Promela. Асинхронное выполнение моделей. Недетерминированный поток управления. Понятие выполнимости оператора.
  - 4.2 Язык Promela. Основные компоненты модели на языке Promela. Процессы, локальные и глобальные объекты данных, каналы сообщений.
  - 4.3 Язык Promela. Механизмы взаимодействия процессов в языке Promela. Глобальные переменные, каналы сообщений, явная синхронизация.
  - 4.4 Язык Promela. Основные операторы языка Promela. Операторы-выражения, присваивания.
  - 4.5 Язык Promela. Основные операторы языка Promela. Отладочная печать, операторы skip, true, run, assert.
  - 4.6 Язык Promela. Чередование (интерливинг) операторов. Внешний и внутренний недетерминизм. Управление выполнимостью операторов.
  - 4.7 Язык Promela. Задание потока управления последовательного процесса. Управляющие конструкции if, do. Организация внутреннего недетерминизма.
  - 4.8 Язык Promela. Каналы сообщений. Операторы отправки и приёма сообщений. Тип mtype. синхронная и асинхронная передача сообщений.
  - 4.9 Язык Promela. Каналы сообщений. Вспомогательные операции с каналами сообщений.
  - 4.10 Язык Promela. Основные типы данных. Область видимости данных.

Моделирование и абстракция

Моделирование программ. Понятие состояния. Потенциальные и достижимые состояния. Требования к модели. Процесс построения модели.

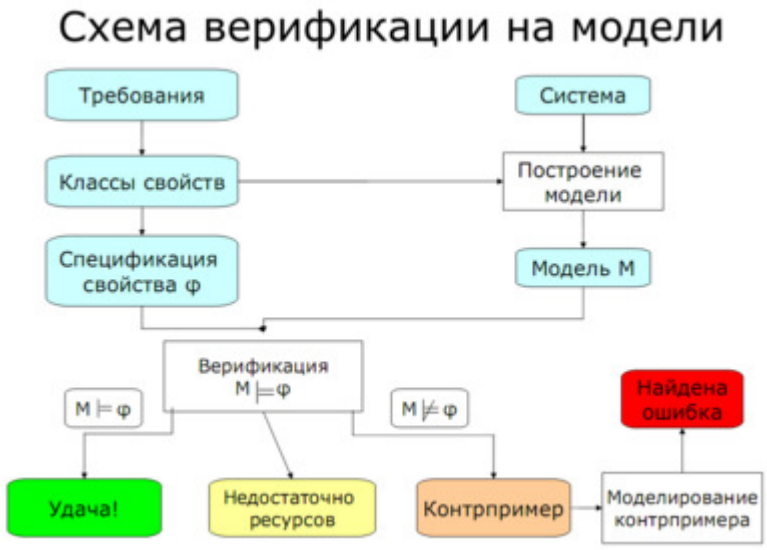

Схема верификации на моделях (*Лекция 2, слайд 3*)

Состояние программы - совокупность значений переменных и управления, связанных с некоторой моделью программы.

Модель - упрощённое описание реальности, выполненное с определенной целью.

- с каждым объектом может быть связано несколько моделей
- каждая модель отражает свой аспект реальности

Аспекты модели:

- *простота* - модель должна быть проще, чем реальность
- *корректность* - не расходиться с реальностью
- *адекватность* - соответствовать решаемой задаче

Построение модели

1. формализация требований (постановка задачи моделирования)
2. выбор языка моделирования
3. абстракция системы до модели с учётом требований

Построение модели в строгом смысле (*Лекция 2, слайд 38*)

Моделирование программ. Размеченные системы переходов. Детерминизм и недетерминизм. Вычисления и трассы. Свойства линейного времени. Выполнимость свойства на трассе.

Лекция 2, Слайды 39-50

Размеченная система переходов (LTS)

$$TS = \langle S, Act, \xrightarrow{a}, s_0, AP, L \rangle$$

- S - множество состояний
- Act - множество действий
- *τ - невидимое действие*
- $\xrightarrow{a} \subseteq S \times Act \times S$ - тотальное отношение переходов. Тотальность означает, что из каждого состояния ведёт какое-то действие.
- $s_0 \in S$ - начальное состояние, либо *I* - множество начальных состояний
- AP - множество атомарных высказываний
- $L : S \to 2^{AP}$ - функция разметки

S, Act - конечные или счётные множества

$$\langle s, a, s' \rangle \in \xrightarrow{a} \equiv s \xrightarrow{a} s'$$

Пример LTS: Лекция 2, слайды 40-41

Прямые потомки

- $Post(s, a) = \{s' \in S, s \xrightarrow{a} s'\}$ - такие состояния s', которые непосредственно вытекают из s через переход a
- $Post(s) = \bigcup_{a \in Act} Post(s, a)$ - все возможные состояния s', которые непосредственно вытекают из s

Система $TS = \langle S, Act, \xrightarrow{a}, I, AP, L \rangle$ **детерминирована**:

- **по действиям** тогда и только тогда, когда
  - $|I| \leqslant 1$
  - $\forall s \in S, \forall a \in Act \Rightarrow |Post(s,a)| \leqslant 1$ (количество потомков не больше одного)
- **по атомарным высказываниям**
  - $|I| \leqslant 1$
  - $\forall s \in S, \forall A \in 2^{AP} \Rightarrow |Post(s) \cap \{s' \in S, L(s') = A\}| \leqslant 1$ (количество **одинаково размеченных** потомков не больше одного)

Недетерминизм - это фича! Полезен для:

- моделирования параллельного выполнения в режиме чередования (интерливинга)
  - позволяет не указывать скорость выполнения процессов
- моделирования прототипа системы
  - не ограничивает реализацию заданным порядком выполнения операторов
- построения абстракции реальной системы
  - модель может быть построена по неполной информации

Вычисления

1. **Конечный фрагмент вычисления σ** системы переходов TS - это конечная последовательность чередующихся состояний и действий, заканчивающаяся состоянием: $\sigma = s_0 a_1 s_1 a_2 s_2 \ldots a_n s_n, \forall i \in [0, n) \Rightarrow s_i \xrightarrow{a_{i+1}} s_{i+1}$
2. **Бесконечный (максимальный) фрагмент вычисления ρ** - $\rho = s_0 a_1 s_1 a_2 s_2 \ldots, \forall i \geqslant 0 \Rightarrow s_i \xrightarrow{a_{i+1}} s_{i+1}$
3. **Начальный фрагмент вычисления** - фрагмент вычисления, для которого $s_0 \in I$
4. **Вычисление** - начальный максимальный фрагмент вычисления (описывает последовательность состояний и действий)

Достижимое состояние (из начального) в системе переходов TS - такое состояние $s \in S$, для которого существует конечный фрагмент вычисления $s_0 a_1 s_1 a_2 s_2 \ldots a_n s_n = s$

Reach(TS) - множество всех достижимых состояний в TS

Трасса $tr = L(s_0)L(s_1)\ldots \in (2^{AP})^{\omega}$

Свойства линейного времени

- Свойство φ определяет набор допустимых трасс: $\varphi \subseteq (2^{AP})^{\omega}$
- Система переходов TS удовлетворяет свойству линейного времени φ, если: $TS \models \varphi \quad \Leftrightarrow \quad Traces(TS) \subseteq \varphi$
- по определению программа удовлетворяет свойству φ, если её система переходов удовлетворяет этому свойству: $P \models \varphi \quad \equiv \quad TS(P) \models \varphi$.

Моделирование программ. Графы программ. Статическая и операционная семантика.

Граф программы – формальное описание текста программы.

- D_p -- единый абстрактный домен данных.
- P -- программа.
  - V_p -- множество переменных программы(Var).
- $v \in Var$
  - $dom(v) = D_p^v \subseteq D_p$ -- каждая переменная принадлежит какому-либо домену
- n -- подстановка. $n : V_p \to D_p, \forall v \in Var\ n(v) \in D_p^v$
- $Cond(V_p)$ -- Набор булевых условий над V_p
  - формулы пропозициональной логики
  - условия на переменные
- $Eval(Var)$ -- множество значений переменных. Собственно, это и есть подстановка.
- Эффект операторов: $Effect : Act \times Eval(Var) \to Eval(Var)$

Граф программы:

$$PG = \langle Loc, Act, Effect, \to, Loc_0, g_0 \rangle$$

- Loc -- множество точек
  - $Loc_0 \in Loc$ -- множество начальных точек
- Act -- множество действий
- $\to \subseteq Loc \times (Cond(V_p) \times Act) \times Loc$ -- отношение перехода ($Cond(V_p)$ -- это фактически страж оператора)
- *Effect* -- функция эффекта
- $g_0 \in Cond(V_p)$ -- начальное условие

Получение TS из PG: раскрутка графа

- Состояние в TS -- это точка программы и текущая подстановка
- Начальное состояние -- исходная точка, удовлетворяющая начальному условию
- Атомарные высказывания в TS:
  - находимся в точке программы l
  - значение переменной x принадлежит некоторому множеству и это множество является подмножеством множеств значений этой переменной.
- Состояния $\langle l, n \rangle$ размечаются высказыванием о том, что мы находимся в точке программы l и всеми высказываниями, истинными в n
- Если в графе программы есть дуга из l в *l'* со стражем g и действием a и в некоторой подстановке n выполняется страж g, то в системе переходов, которая соответствует этой программе будет присутствовать дуга из состояния $\langle l, n \rangle$ в состояние $\langle l', Effect(a, n) \rangle$ по действию a.

Системы переходов графов программ Операционная семантика -- строгое описание того как из графа программы получить ее систему переходов. Описывается это все при помощи правил вывода.
TS(PG) -- система переходов графа программы $PG = \langle Loc, Act, Effect, \rightarrow, Loc_0, g_0 \rangle$ задается сигнатурой $\langle S, Act, \rightarrow, I, AP, L \rangle$

- $S = Loc \times Eval(V_p)$ (декартово произведение точек программы на всевозможные подстановки)
- $\rightarrow: S \times Act \times S$ с соответствующим правилом вывода $\dfrac{l \xrightarrow{g:a} l' \wedge (n \models g)}{\langle l, n \rangle \xrightarrow{a} \langle l', Effect(a, n) \rangle}$

- Множество начальных состояний системы переходов описывается как множество состояний, в которых точка программы принадлежит начальным точкам, а на подстановках выполняется начальное условие графа программы:
  - $I = \{\langle l, n \rangle : l \in Loc_0, n \models g_0\}$
- Множество атомарных высказываний -- это объединение множества точек программы и всевозможных булевых высказываний над переменными программы:
  - $AP = Loc \cup Cond(V_P)$
- Состояния вида $\langle l, n \rangle$ размечаются высказываниям о точке программы, в которой мы находимся и всеми высказываниями из множества всевозможных высказываний, которые верны в этой подстановке:
  - $L(\langle l, n \rangle) = \{l\} \cup \{g \in Cond(V_P) : n \models g\}.$

Пример: Лекция 4, слайд 16

Параллелизм. Чередование систем переходов.

Лекция 4, слайды 21-24

- Действия независимых процессов чередуются.
- Порядок выполнения процессов не известен.

Чередование:

- эффект от параллельного выполнения независимых действий a и b равен эффекту от их последовательного выполнения в произвольном порядке:
  - $Effect(a \;|||\; b, n) = Effect((a; b) + (b; a), n)$
    - ||| -- оператор чередования
    - ; -- оператор последовательного выполнения
    - \+ -- оператор недетерминированного выбора

Пример: Лекция 4, слайд 23

Чередование систем переходов

Пусть $TS_i = \langle S_i, Act_i, \rightarrow_i, AP_i, I_i, L_i \rangle \quad , \quad i \in \{1, 2\}$

Тогда чередование этих систем
$TS_1 \mid\mid\mid TS_2 = \langle S_1 \times S_2, Act_1 \cup Act_2, \rightarrow, AP_1 \cup AP_2, I_1 \times I_2, L \rangle$, где

- $L(s_1, s_2) = L_1(s_1) \cup L_2(s_2)$
- оператор $\rightarrow$ определяется как
  - $\dfrac{s_1 \rightarrow_1 s_1'}{(s_1, s_2) \rightarrow (s_1', s_2)}$
  - $\dfrac{s_2 \rightarrow_2 s_2'}{(s_1, s_2) \rightarrow (s_1, s_2')}$

Параллелизм. Чередование графов программ. Случаи без разделяемых переменных и с разделяемыми переменными.

Лекция 4, слайды 25—28

Для графов программ PG_1 (над V_1) и PG_2 (над V_2) без разделяемых переменных (т. е. $V_1 \cap V_2 = \varnothing$):

- формула $TS(PG_1) \mid\mid\mid TS(PG_2)$ достоверно описывает параллельную композицию PG_1 и PG_2
- в случае с разделяемыми переменными это не так (см. лекцию 4, слайд 26).

Пусть

$$PG_i = \langle Loc_i, Act_i, Effect_i, \rightarrow_i, Loc_{0,i}, g_{0,i} \rangle, \ i \in \{1, 2\}.$$

Граф $PG_1 \mid\mid\mid PG_2$ над $V_1 \cup V_2$ определяется так:

$$PG_1 \mid\mid\mid PG_2 = \langle Loc_1 \times Loc_2, Act_1 \cup Act_2, Effect, \rightarrow, Loc_{0,1} \times Loc_{0,2}, g_{0,1} \wedge g_{0,2} \rangle,$$

где отношение перехода $\rightarrow$ определяется следующими правилами вывода:

$$\frac{l_1 \overset{g:\alpha}{\rightarrow}_1 l_1'}{\langle l_1, l_2 \rangle \overset{g:\alpha}{\rightarrow} \langle l_1', l_2 \rangle} \text{ и } \frac{l_2 \overset{g:\alpha}{\rightarrow}_2 l_2'}{\langle l_1, l_2 \rangle \overset{g:\alpha}{\rightarrow} \langle l_1, l_2' \rangle},$$

а $Effect(\alpha, \eta) = Effect_i(\alpha, \eta)$, если $\alpha \in Act_i$.

Пример: лекция 4, слайд 28. В указанном примере
$TS(PG_1) \mid\mid\mid TS(PG_2) \neq TS(PG_1 \mid\mid\mid PG_2)$

Параллелизм. Синхронный параллелизм. Рандеву.

- распределённые программы выполняются параллельно
- в распределённой программе нет разделяемых переменных

Передача сообщений в распределённых программах:

- синхронная передача сообщений (рандеву)
- асинхронная передача сообщений (каналы)

Синхронный обмен сообщениями:

- Процессы вместе выполняют синхронизированные действия
- Взаимодействие процессов - одновременно

Рандеву

- $TS_i = \langle S_i, Act_i, \xrightarrow{a}_i, I_i, AP_i, L_i \rangle \quad i = \{1, 2\}$
- $H \subseteq Act_1 \cap Act_2$ -- набор синхронизированных действий.
  - действия из H должны быть синхронизированны
  - действия не из H независимы и могут чередоваться

Тогда $TS_1 ||_H TS_2 = \langle S_1 \times S_2, Act_1 \cup Act_2, \rightarrow, I_1 \times I_2, AP_1 \cup AP_2, L \rangle$, где

- $L(\langle s_1, s_2 \rangle) = L_1(s_1) \cup L_2(s_2)$
- $\rightarrow$ определяется как:
  - интерливинг для $\alpha \notin H \ : \ \dfrac{s_1 \xrightarrow{a}_1 s_1'}{\langle s_1, s_2 \rangle \xrightarrow{a} \langle s_1', s_2 \rangle}$ и $\dfrac{s_2 \xrightarrow{a}_2 s_2'}{\langle s_1, s_2 \rangle \xrightarrow{a} \langle s_1, s_2' \rangle}$
  - рандеву для $\alpha \in H \ : \ \dfrac{s_1 \xrightarrow{a}_1 s_1' \ \wedge \ s_2 \xrightarrow{a}_2 s_2'}{\langle s_1, s_2 \rangle \xrightarrow{a} \langle s_1', s_2' \rangle}$

Пример рандеву: Лекция 4, слайд 32

Синхронный параллелизм

- $TS_i = \langle S_i, Act_i, \xrightarrow{a}_i, I_i, AP_i, L_i \rangle \quad i = \{1, 2\}$
- $Act_1 \times Act_2 \rightarrow Act \quad : \quad (\alpha, \beta) \rightarrow \alpha * \beta$

Тогда $TS_1 \times TS_2 = \langle S_1 \times S_2, Act, \rightarrow, I_1 \times I_2, AP_1 \cup AP_2, L \rangle$, где

- $L(\langle s_1, s_2 \rangle) = L_1(s_1) \cup L_2(s_2)$
- $\rightarrow$ определяется как: $\dfrac{s_1 \xrightarrow{\alpha}_1 s_1' \ \wedge \ s_2 \xrightarrow{\beta}_2 s_2'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha * \beta} \langle s_1', s_2 \rangle}$

Параллелизм. Асинхронный параллелизм. Системы с каналами. Операционная семантика.

Лекция 4. Слайды 36-39, 43-46

- $c \in Chan$ -- Процессы взаимодействуют с помощью **каналов**, представляющих собой FIFO буфера
- *dom*(*c*) -- Каналы типизированы по передаваемым сообщениям
- *cap*(*c*) -- ёмкость канала. если *cap*(*c*) = 0, то взаимодействие сводится к рандеву
- действия по обмену сообщениями:
  - *c*!*v* -- запись *v* в конец канала *c*. Выполняется только если буфер не полон (< *cap*(*c*))
  - *c*?*x* -- чтение в *x* из начала канала *c*. Выполняется только если буфер не пуст (> *cap*(*c*))
  - формально
    $$Comm = \{c!v, c?x \mid c \in Chan, v \in dom(c), x \in V_p, dom(c) \subseteq dom(x)\}$$

Системы с каналами.

- Граф программы *PG* над (*Var*,*Chan*) задаётся $PG = \langle Loc, Act, Effect, \rightarrow, Log_0, g_0 \rangle$, где $\rightarrow \subseteq Loc \times (Cond(Var) \times Act) \times Loc \ \cup \ Loc \times Comm \times Loc$
- Система с каналами *CS* над $(\cup_{1 \leqslant i \leqslant n} PG_i, Chan)$ задаётся как $CS = [PG_1 | PG_2 | \ldots | PG_n]$, где PG_i — граф программы над (Var_i,*Chan*)

При асинхронной передаче сообщений (при *cap*(*c*) > 0):

- процесс P_i может выполнить $l_i \xrightarrow{c!v} l_i'$, только если в *c* хранится меньше *cap*(*c*) сообщений;
- процесс P_j может выполнить $l_j \xrightarrow{c?x} l_j'$, только если *c* не пуст, после чего первый элемент *v* извлекается из *c* и присваивается *x* (атомарно).

Оценка ξ значения канала *c* — это отображение канала на последовательность значений $\xi : Chan \rightarrow dom(c)^*$, такое что длина последовательности не превосходит ёмкости канала $len(\xi(c)) \leq cap(c)$, и при этом $\xi(c) = v_1 v_2 \ldots v_k$ означает, что v_1 — верхнее сообщение в буфере.

$$\xi[c = v_1 v_2 \ldots v_k](c') = \begin{cases} \xi(c'), & c \neq c' \\ v_1 v_2 \ldots v_k, & c = c' \end{cases}$$ *Кто понимает смысл этой хренотени, опишите, плз. Overrider 18:28, 22 мая 2009 (UTC)*

Исходная оценка $\xi_0(c) = \epsilon, \forall c \in Chan$

Операционная семантика: лекция 4, слайды 44—46

Абстракция. Абстракция трасс. Абстракция системы переходов. Необходимое и достаточное условие корректности LTS модели.

Представим трассу в форме интерпретации I: $I(tr) = \langle \mathbb{N}, \leqslant, \xi \rangle$

- $\mathbb{N}$ - множество натуральных чисел
- $\leqslant$ - отношение порядка на $\mathbb{N}$
- $\xi : \mathbb{N} \times AP \to \{true, false\}, \ \forall n > 0, p \in AP \ \Rightarrow \ \xi(n,p) = true \Leftrightarrow p \in L(s)$
  (для каждого порядкового номера говорит истенен или ложен заданный на нем предикат)

Рассмотрим трассы tr и tr' такие, что

- $I(tr) = \langle \mathbb{N}, \leqslant, \xi \rangle, \ \ \xi : \mathbb{N} \times AP = \{true, false\}$
- $I(tr') = \langle \mathbb{N}, \leqslant, \xi' \rangle, \ \ \xi : \mathbb{N} \times AP' = \{true, false\}$

Будем говорить, что трасса tr' является **абстракцией трассы** tr ($tr \leqslant tr'$), если

1. $AP' \subseteq AP$
2. $\exists \alpha : \mathbb{N} \to \mathbb{N}$ такое, что
   - α - неубывающая функция: $\forall n, k \in \mathbb{N}, n \leqslant k \ \ \Rightarrow \ \ \alpha(n) \leqslant \alpha(k)$
   - $\forall n \in \mathbb{N}, p \in AP' \ \ \Rightarrow \ \ \xi(n,p) = \xi'(\alpha(n), p)$

Пример абстракции трассы: Лекция 2, слайд 53

Необходимое условие корректности модели -
$\forall tr \in Traces(TS(P)) \ \exists tr' \in Traces(TS(M)) \ : \ tr \leqslant tr'$, где

- *P* - система
- *M* - модель этой системы

При этом, если φ - некоторое свойство системы, то $M \models \varphi \ \Rightarrow \ P \models \varphi$ выполняется тогда и только тогда, когда верно условие корректности модели.

Абстракция. Абстракция системы переходов. Достаточное условие корректности LTS модели. Адекватность LTS модели.

Абстракция системы переходов -- картинка на 4 слайде 3-й лекции.

Достаточное условие корректности LTS модели.

Пусть у нас имеются две системы переходов, TS^1 и TS^2 -- для системы и модели соответственно:

$$TS^i = \langle S^i, Act^i, \rightarrow_i, I^i, AP^i, L^i \rangle$$

Достаточное условие корректности:

- Алфавит предикатов модели включен в алафвит предикатов системы: $AP^2 \subseteq AP^1$
- Задано отображение $a : S^1 \to S^2$. На отображение накладываются следующие условия:

    - Оно преобразует начальное состояние системы в начальное состояние модели: $s_0^2 = a(s_0^1)$
    - Каждому переходу из системы должен соответствовать переход в модели: $s_1^1 \to_1 s_2^1 \Rightarrow a(s_1^1) \to_2 a(s_2^1)$
- Метки на состояниях модели должны состоять только из предикатов модели: $\forall s \in S^1 : L^2(a(s)) = L^1(s) \cap 2^{AP^2}$

Необходимое условие адекватности модели свойствам правильности: алфавит предикатов свойств правильности включен в алфавит предикатов модели — $AP_\phi \subseteq AP_M$. Условие не является достаточным (см. примеры, лекция 3, слайды 4—5).

Абстракция. Абстракция графов программ. Отношение слабой симуляции.

Лекция 10, слайды 8 - 11

Программа PG' *корректно моделирует* программу PG тогда и только тогда, когда система переходов TS(PG') корректно моделирует систему переходов TS(PG).

Будем говорить, что PG' моделирует PG, если

- в PG' присутствуют переменные, соответствующие наблюдаемым переменным PG
- все действия PG, влияющие на наблюдаемые переменные, отражены в модели *(наблюдаемые операторы)*
- модель корректно воспроизводит возможные последовательности изменения значений наблюдаемых переменных, присутствующих в PG

Отношение слабой симуляции не сохраняет количество шагов между состояниями. В связи с этим, не сохраняются свойства, не инвариантные к прореживанию (LTL: оператор neXt).

Логика LTL, автоматы Бюхи

Свойства правильности. Формулирование требований правильности программы. Двойственность. Типы свойств.

Лекция 5, слайды 2-14.

Требования правильности — утверждения о возможных и невозможных вариантах выполнения программы.

Двойственность :

- если какое-то утверждение невозможно, то обратное — неизбежно
- если какое-то утверждение неизбежно, то обратное — невозможно
- при помощи логики от одного можно переходить к другому при помощи отрицания

Способы описания свойств правильности:

- свойства достижимых состояний (свойства безопасности)
- свойства последовательности состояний (свойства живучести)
- в Promela:
    - свойства состояний

        - asserts:
            - локальные ассерты процессов
            - инварианты системы процессов
        - метки терминальных состояний
            - задаём допустимые точки останова прочесса
    - свойства последовательностей состояний
        - метки прогресса — чтобы найти циклы бездействия
        - утверждения о невозможности (never claims) — например, LTL формулы
        - трассовые ассерты — используются для описания правильных последовательностей выполнения операторов отправки и приема сообщения

Свойства правильности. Свойства безопасности и живучести. Проверка таких свойств. Примеры свойств.

Типы свойств:

- свойства безопасности
    - ничего плохого никогда не произойдет
    - пример: инвариант системы (например, x всегда меньше y);
    - задача верификатора -- найти те вычисления, которые ведут к нарушению безопасности.
- свойства живучести
    - рано или поздно произойдет что-то хорошее
    - пример: “отзывчивость” (например, если отправлен запрос, то рано или поздно будет сгенерирован ответ)
    - задача верификатора – найти вычисления, в которых это “хорошее” может откладываться до бесконечности.

ps. автор терминов – Лесли Лампорт; см. *Лекция 5, слайд 4*.

Автоматы Бюхи. Конечные автоматы. Проход автомата. Язык автомата.

Лекция 6, слайды 8 - 15

Конечный автомат описывается сигнатурой: $\langle S, s_0, L, F, T \rangle$, где

- S -- множество сотояний
- $s_0 \in S$ -- множество начальных состояний
- L -- конечное множество меток
- $F \subseteq S$ -- множество терминальных состояний
- $T \subseteq S \times L \times S$ -- отношение перехода на состояниях

Детерминизм и недетерминизм

Конечный автомат называется **детерминированным**, если по метке и исходному состоянию можно однозначно определить целевое состояние:

$$\forall s \in S, \forall l \in L : ((s, l, s_1) \in A.T \wedge (s, L, s_2) \in A.T \;\Rightarrow\; s_1 = s_2)$$

В противном случае автомат называется **недетерминированным**.

Проходом a конечного автомата $\langle S, s_0, L, F, T \rangle$ называется такое *упорядоченное* и, возможно, бесконечное множеств переходов из T:

$$a = \langle (s_0, l_0, s_1), (s_1, l_1, s_2), ... \rangle \quad \forall i, i \geqslant 0; \ (s_i, l_i, s_{i+1}) \in T$$

Допускающим проходом конечного автомата A называется конечный проход a, финальный переход которого (s_{n-1}, l_{n-1}, s_n) ведёт в терминальное состояние.

Языком автомата A называется множество слов в алфавите A.L, соответствующих допускающим проходам автомата A

Автоматы Бюхи. Омега-допускание. Расширение автоматов Бюхи.

Лекция 6, слайды 19-20

Для любого бесконечного прохода σ конечного автомата A можно выделить два последовательных отрезка проходов:

- σ^+ -- конечный отрезок прохода σ, включающий в себя множество состояний, встречающихся конечное число раз
- σ^ω -- бесконечный хвост прохода σ, включающий в себя множество состояний, встречающихся бесконечное число раз

Допускающий проход по Бюхи (ω-допускание) конечного автомата A называется такой *бесконечный проход*, в котором по крайней мере одно терминальное состояние встречается бесконечное число раз: $\exists i \geqslant 0, (s_{i-1}, l_{i-1}, s_i) \in \sigma \quad : \quad (s_i \in A.F) \wedge (s_i \in \sigma^\omega)$

Расширение автоматов Бюхи.

- добавляем алфавит автомата меткой ε
- все конечные проходы расширяем до бесконечности меткой ε

Примечание. При помощи автоматов Бюхи удобно проверять свойства живучести.

Логика LTL. Синтаксис LTL. Семантика выполнимости формул. Сильный и слабый until.

Лекция 6, слайды 30 - 35

Особенности LTL:

- может использоваться для описания свойств как живучести, так и безопасности
- описывает свойства, которым должны удовлетворять линейные последовательности наблюдаемых состояний - трассы
- семантика LTL определена на бесконечных автоматах Бюхи. Для конечных проходов необходимо использовать расширение автомата.

Формула в LTL **f::=**

- **p, q, ...** — свойства состояний, включая **true** и **false**
- **(f)** — группировка при помощи скобок

- $\alpha\ f$ — унарные операторы
- $f_1\ \beta\ f_2$ — бинарные операторы

Операторы в LTL

- унарные
  - $\Box$([]) — всегда в будущем, т.е. $s_j \models \Box f \quad \Leftrightarrow \quad \forall j, j \geqslant i:\ s_j \models f$
  - $\Diamond$ (<>) — в конце концов, т.е. $s_j \models \Diamond f \quad \Leftrightarrow \quad \exists j, j \geqslant i:\ s_j \models f$
  - X (X) — в следующем состоянии, т.е. $s_j \models X f \quad \Leftrightarrow \quad i:\ s_{j+1} \models f$
  - $\neg$ (!) — логическое отрицание
- бинарные
  - $\wedge$ (&&) — логическое И
  - $\vee$ (||) — логическое ИЛИ
  - $\rightarrow$ (->) — логическая импликация
  - $\leftrightarrow$ (<->) — логическая эквивалентность
  - U (U) — до тех пор, пока (until)

Сильный Until:

- всегда e, до тех пор, пока не f, при этом f обязательно должно наступить
- $$s_i \models eUf \quad \Leftrightarrow \quad \begin{cases} \exists j, j \geqslant i:\ s_j \models f \\ \forall k, i \leqslant k < j:\ s_k \models e \end{cases}$$

Слабый Until:

- всегда e, до тех пор, пока не f, при этом не факт, что f наступает (тогда всегда e)
- $s_i \models e \cup f \quad \Leftrightarrow \quad s_i \models f \vee (s_i \models e \wedge s_{i+1} \models e \cup f)$

Выполнимость формул:

- Задаётся последовательность состояний прохода σ
- $\forall i, i \geqslant 0,\ \forall p \quad :\quad s_i \models p$ (**Внимание**!! *это слишком смахивает на бред, который должен быть интуитивно понятен. Если кто может - распишите подробнее выполнимость!!* может быть, здесь имеется в виду что в каждом состоянии определена формула, или что если есть формула, то она определена в каком-то состоянии. вообщем, wierd).

Логика LTL. Основные типы свойств LTL. Цикличность, стабильность, инвариант, гарантия, отклик, приоритет, корреляция.

Лекция 6, Слайды 38-39

Распространенные LTL-формулы

Формула	**Описание**	**Тип**
$\Box p$	всегда p	*инвариант*
$\Diamond p$	рано или поздно p	*гарантия*
$p \rightarrow \Diamond q$	если p, то рано или поздно q	*отклик*

	если p то затем постоянно q до тех пор,	
$p \rightarrow qUr$	пока рано или поздно не наступит r	*приоритет*
$\Box \diamond p$	всегда рано или позндно будет p	*цикличность (прогресс)*
$\diamond\Box p$	рано или позндно всегда будет p	*стабильность (бездействие)*
$\diamond p \rightarrow \diamond q$	если рано или поздно p, то рано или поздно q	*корреляция*

Логика LTL. Эквивалентные преобразования формул LTL.

Лекция 6, Слайды 40

$$\neg\Box p \Leftrightarrow \diamond\neg p \qquad p \cup (q \vee r) \Leftrightarrow (p \cup q) \vee (p \cup r)$$
$$\neg \diamond p \Leftrightarrow \Box\neg p \qquad (p \wedge q) \cup r \Leftrightarrow (p \cup r) \wedge (q \cup r)$$

$$\neg(pUq) \Leftrightarrow \neg q \cup (\neg p \wedge \neg q) \qquad pU(q \vee r) \Leftrightarrow (pUq) \vee (pUr)$$
$$\neg(p \cup q) \Leftrightarrow \neg qU(\neg p \wedge \neg q) \qquad (pUq) \cup r \Leftrightarrow (pUr) \wedge (qUr)$$

$$\Box(p \wedge q) \Leftrightarrow \Box p \wedge \Box q \qquad \Box \diamond (p \wedge q) \Leftrightarrow \Box \diamond p \wedge \Box \diamond q$$
$$\diamond(p \vee q) \Leftrightarrow \diamond p \vee \diamond q \qquad \diamond\Box(p \vee q) \Leftrightarrow \diamond\Box p \vee \diamond\Box q$$

Логика LTL. Оператор neXt. Свойства, инвариантные к прореживанию.

Лекция 7, слайды 22-25

Оператор *X* нужно использовать аккуратно:

- с его помощью делается утверждение о выполнимости формулы на непосредственных потомках текущего состояния,
- в распределённых системах значение оператора *X* неочевидно,
- поскольку алгоритм планирования процессов неизвестен, не стоит формулировать спецификацию в предположении о том,

какое состояние будет следующим,

- стоит ограничиться предположением о справедливости планирования.

Свойства, инвариантные к прореживанию

- Пусть f – трасса некоторого вычисления над пропозициональными формулами P,
  - по трассе можно определить, выполняется ли на ней темпоральная формула,
  - трассу можно записать в форме:
    - $f^{n_1}, f^{n_2}, f^{n_3}, \cdots$-- где значения пропозициональных формул на каждом интервале совпадают.

- Обозначим E(f) набор всех трасс, отличающихся лишь значениями n_1,n_2,n_3 (т.е. длиной интервалов).
  - E(f) называется расширением прореживания f.

- Свойство f, инвариантное к прореживанию, либо истинно для всех трасс из $E(\psi)$, либо ни для одной из них:
  - $\psi \models f \Rightarrow \forall v \in E(\psi), v \models f$
- истинность такого свойства не зависит от длины трассы, а только от порядка, в котором пропозициональные формулы меняют свои значения;
- Теорема: все формулы LTL без оператора *X* инвариантны к прореживанию.
- в рамках LTL без *X* можно описать все свойства, инвариантные к прореживанию

Логика LTL. Проверка выполнимости формул LTL при помощи автоматов Бюхи. Проверка LTL-формул в Spin.

Лекция 7, слайды 26-40

Связь LTL с автоматами Бюхи

- Удобно проверять допустимость трасс для некоторого автомата Бюхи;
- Удобно описывать свойства правильности при помощи темпоральных формул;
- Для всякой LTL-формулы f существует автомат Бюхи, который допускает те и только те прогоны, которым соответствуют трассы, на которых выполнима f;

Переход от LTL к автоматам

- Привести свойство правильности LTL к форме never языка Promela достаточно просто: нужно построить отрицание LTL формулы и поместить его в тело never:
  - f выполняется на всех вычислениях <=>
  - !f не выполняется ни на одном вычислении <=>
  - автомат never {!f} не допускает ни одного прогона, полученного в результате синхронного выполнения с системой

Логика LTL. Выразительная мощность LTL. Логики LTL + существование, CTL* и CTL. Сравнение выразительной мощности.

При помощи конструкции never можно описать любой ω-регулярный автомат над словами

Выразительная мощность LTL

по сравнению с конструкциями never

- LTL описывает подмножество этого языка:
  - всё, выразимое на LTL, может быть описано при помощи never,
  - при помощи never можно описать свойства, не выразимые на LTL
- Добавление одного квантора существования над одним пропозициональным символом расширяет выразительные способности LTL до всех омега-регулярных автоматов над словами.

(p) может быть истинным после выполнения системой чётного количества шагов, но никогда не истинно после нечётного.

$\exists$ t(t && [](t -> X!t) && [](!t -> Xt) && [](!t -> !p))

LTL-формула описывает свойство, которое должно выполняться на **всех** вычислениях, начинающихся из исходного остояния системы

Логика CTL*

- Логика ветвящегося времени:
  - использует кванторы ∀ и ∃ для обозначения трасс, на которых может выполняться свойство
  - использует F вместо <> и G вместо []

Логика CTL

Логика CTL – фрагмент логики CTL*, в котором кванторы могут встречаться только парами, причём в паре должны обязательно находиться один временной и один пространственный кванторы. Например: AG EF(p), A(p U q).

Выразительные возможности CTL* и CTL

- CTL* и CTL описывают подмножества w-регулярных автоматов над деревьями
  - автоматы над деревьями более выразительны, чем автоматы над словами (CTL-формула выполнима на дереве трасс, а не на одной трассе);
- CTL и LTL являются подмножествами CTL*;
- CTL и LTL не сравнимы по выразительной мощности (пересекаются, но не включают);
- на LTL можно описать свойства, не выразимые на CTL:
  - CTL не позволяет описать свойства вида []<>(p),
  - при помощи []<>(p) в LTL задаются ограничения справедливости;
- на CTL можно описать свойства, не выразимые на LTL:
  - на LTL нельзя описать свойства вида AGEF(p),
  - AGEF(p) используется для описания свойства reset: из любого состояния

система может перейти в нормальное

Верификация программ на моделях

Задача проверки правильности программ. Валидация. Верификация. Системы с повышенными требованиями к надёжности. Реактивные программы. Параллельные программы. Особенности верификации таких программ.

Валидация - исследование и обоснование того, что спецификация ПО и само ПО через реализованную в нём функциональность удовлетворяет ребованиям пользователей.

Верификация - исследование и обоснование того, что программа соответствует своей спецификации.

Верификация в общем случае алгоритмически неразрешима.

Методы верификации:

- "Полное" тестирование (*Лекция 1, слайды 14-22*)
- Имитационное моделирование (вики)
- Доказательство теорем (*слайды 27-29*)
- Статический анализ (*слайды 30-33*)

- Верификация на моделях (*слайды 34-38*)

Типы программ:

- Традиционные программы
  - завершимость
  - спецификация включает в себя описание входа/выхода программы
  - число состояний зависит от входных данных и переменных
- Реактивные программы
  - работают в бесконечном цикле
  - взаимодействуют с окружением
  - спецификация представляет собой пары стимул/реакция
- Параллельные программы
  - совместная работа нескольких компонент
  - невоспроизводимость тестов
  - ограниченные возможности по наблюдению

Подходы к верификации программ. Тестирование и имитационное моделирование. Область применения, плюсы и минусы. Проблема полноты тестового покрытия.

«Тестирование может показать присутствие ошибок, но не может показать их отсутствия» © Дейкстра.

Обоснование полноты тестового покрытия:

- метод «чёрного ящика» (ЧЯ) — полное покрытие входных данных,
- метод «прозрачного ящика» (ПЯ) — полное покрытие кода программы.

Плюсы применения тестирования:

- проверяется та программа, которая будет использоваться,
- не требуется знание/использование дополнительных инструментальных средств,
- удобная локализация ошибки.

Минусы применения тестирования:

- не всегда есть условия для тестирования системы,
- проблема с воспроизводимостью тестов (частичное решение — имитационное моделирование).

Проблема полноты тестового покрытия

- Чёрный ящик:
  - для последовательных программ сложно перебрать все входные данные,
  - для параллельных программ — очень сложно,
  - для динамических структур данных, взаимодействия с окружением — невозможно.
- Прозрачный ящик:
  - большой размер покрытия,
  - часто невозможно построить 100% покрытие,
  - полное покрытие не гарантирует отсутствия ошибок.

Итоги:

- Полный перебор входных данных — невозможен.
- Полнота покрытия кода не гарантирует правильности.
- Ошибка — ошибочное вычисление системы.
- Полнота в терминах возможных вычислений — хороший критерий.

Подходы к верификации программ. Доказательство теорем. Область применения, плюсы и минусы.

Основные пункты:

- система и её свойста - формулы
- задан набор аксиом и правил вывода
- строится доказательство свойства-теоремы
- таким образом, производится *качественный* анализ системы

Пример: Лекция 1, слайд 28

Достоинства:

- работа с бесконечными пространствами состояний
- даёт более глубокое понимание системы

Недостатки

- медленная скорость работы
- может потребоваться помощь человека (построение инвариантов циклов)
- в общем случае нельзя построить полную систему аксиом и правил вывода

Подходы к верификации программ. Статический анализ исходного кода программ. Область применения, плюсы и минусы.

Статистический анализ -- оцениваем для каждого состояния программы потенциально возможные значения переменных.

Пример: Лекция 1, Слайды 31-32

Особенности:

- анализ исходного текста без запуска программы
- в общем случае задача неразрешима

Достоинства:

- высокая скорость работы
- если ответ дан - ему можно верить

Недостатки:

- узкая область применения: компиляторы, анализ похожести кода, анализ безопасности
- ручная настройка при изменении применяемых свойств

Подходы к верификации программ. Верификация программ на моделях. Процесс верификации программы при помощи её модели. Область применения, плюсы и минусы.

Лекция 1, Слайды 34-38 , 45

Особенности:

- проверка свойств на конечной модели
- исчерпывающий поиск по пространству состояний
- свойства задаются в терминах значений предикатов состояний программы или последовательности этих значений

Пример: Лекция 1, слайды 35-36

Процесс верификации программ на моделях:

- моделирование
  - построение адекватной, корректной модели
  - фильтрация "лишних" состояний
- спецификация свойств
  - темпоральная логика
  - полнота свойств
- верификация
  - построение контр-примера
  - анализ контр-примера

Достоинтсва:

- хорошая автоматизация
- если модель конечна, корректна и адекватна данному свойству, то будет получен точный ответ
- выявление редких ошибок

Недостатки:

- работает только для конечных моделей

Области применения

- сетевые и криптографические протоколы
- протоколы работы кэш-памяти
- интегральные схемы
- стандарты
- встроенные системы
- драйвера
- и прочие программы на C

Верификация на моделях. История развития верификации программ на моделях. Схема верификации программ на моделях. Классы проверяемых свойств правильности программы.

Лекция 1, Слайды 40-44

История развития верификации программ на моделях:

- Флойд, 1967 – assertions, гипотеза о доказуемости корректности программы,
- Хоар, 1969 – пред- и пост-условия, триплеты Хоара (P ǀ S ǀ Q), логический вывод,
- Бойер, Мур, 1971 – первый автоматический прувер,
- Дейкстра, 1975 – Guarded Command Languages,
- Хоар, 1978 – взаимодействующие последовательные процессы (CSP).
- Пнуэли, 1977 – темпоральная логика LTL,
- Пнуэли, 1981 – логика ветвящегося времени (CTL),
- Кларк, Эмерсон, 1981 и Квили, Сифакис, 1982 – model checking (обход достижимых состояний),
- Варди и Вольпер, 1986 – новая техника model checking (анализ конформности),
- Хольцман, 1981 – верификатор SPIN.
- Бриан, 1989 – Двоичные решающие диаграммы (BDD),
- МакМиллан, 1993 – верификатор SMV (символьная верификация, BDD),
- Хольцман, Пелед, 1994 – редукция частичных порядков,
- 1995 – редукция частичных порядков в SPIN.
- Кларк, 1992 – абстракция для уменьшения числа состояний модели,
- Эльсаиди, 1994 – семантическая минимизация,
- Пелед, 1996, Бир, 1998 – верификация модели «на лету»,
- Равви, 2000 – анализ достижимости с учётом спецификации,
- Эмерсон, Прасад, 1994 -- симметрия

Рост мощности model checking:

- 1992 год – 1020 состояний,
- 1994 год – 10120 состояний,
- 1998 год(?) – 10394 состояний

Лекция 2, Слайды 3-4

Примеры классов свойств:

- Стандартные
  - deadlocks (взаимная блокировка)
  - Race condition (состояние гонки)
- Специфичные для конкретного приложения
  - требования справедливости
  - корректная завершаемость
  - причинно-следственный и темпоральный порядок состояний системы

Схема верификации на модели: Лекция 2, слайд 3

Верификация при помощи Spin. Задание свойств состояний.

Свойства состояний

- asserts
    - локальные ассерты процессов
    - инварианты системы процессов
        - active proctype invariant() { assert(something)}
- метки терминальных состояний
    - задаём допустимые точки останова процесса
        - метка end -- система не может завершить работу без того, чтобы все активные процессы либо завершились, либо остановились в точках, помеченных метками end;

Верификация при помощи Spin. Задание свойств последовательностей состояний. Циклы бездействия. Ограничения справедливости.

Свойства последовательностей состояний

- метки прогресса (чтобы найти циклы бездействия)
- утверждения о невозможности (never claims)
    - например, определяются LTL-формулами
- трассовые ассерты

Циклы бездействия. Мы хотим найти потенциально бесконечные циклы, не выполняющие никакой полезной работы. Помечаем меткой *progress* полезные операторы. Если найдется цикл, работающий потенциально бесконечно и не проходящий по метке progress, верификатор выдаст ошибку.

Ограничения справедливости. Существует два основных варианта справедливости:

- слабая справедливость:
    - если оператор выполним бесконечно **долго**, то он в конце концов будет выполнен
- сильная справедливость:
    - если оператор выполним бесконечно **часто**, то он в конце концов будет выполнен.

Справедливость применима как к внутреннему, так и к внешнему недетерминизму. Использование сильной справедливости – существенно дороже слабой

Верификация при помощи Spin. Задание свойств последовательностей состояний. Утверждения о невозможности. Трассовые ассерты.

never claims (утверждения о невозможности):

- выполняются синхронно с моделью
- если достигнут конец, то – ошибка
- состоят из выражений и конструкция задания потока управления
- фактически, описывают распознающий автомат.

Конструкция never

- может быть как детерминированной, так и нет;
- содержит только выражения без побочных эффектов (соотв. булевым высказываниям на состояниях);
- используются для описания неправильного поведения системы;
- прерывается при блокировании:
    - блокируется => наблюдаемое поведение не соответствует описанному,

        - паузы в выполнении тела never должны быть явно заданы как бесконечные циклы;
- never нарушается, если:
    - достигнута закрывающая скобка,
    - завершена конструкция accept (допускающий цикл);
- бездействие может быть описано как конструкция never или её часть (для цикла бездействия есть тело never по умолчанию).

Видимость

- все конструкции never – глобальны;
- тем самым, в них можно ссылаться на
    - глобальные переменные,
    - каналы сообщений,
    - точки описания процессов (метки),
    - предопределённые глобальные переменные,
    - но не локальные переменные процессов;

Ассерты на трассы Используются для описания выполнения правильных и неправильных последовательностей операторов send и recieve. *Accepm notrace* - утверждает, что описанный шаблон поведения невозможен.

Верификация при помощи Spin. Принцип верификации нарушения свойств. Контрпримеры. Процесс верификации при помощи Spin. Использование LTL в Spin.

- Свойство *выполняется* на модели, если оно выполняется на всех трассах.
- Свойство *нарушается* на модели, если нарушается хотя бы на одной из трасс.
- **Принцип верификации нарушения свойств** - проще проверять нарушение свойства, чем выполнение свойства.
    - Достаточно найти один контрпример
- Нарушение свойства описывается при помощи конструкции never – автомата, распознающего неправильное поведение
    - Автоматы Бюхи
- Свойства на последовательностях состояний удобно описывать при помощи темпоральной логики
    - Логика LTL

- <u>Связь LTL с автоматами Бюхи</u>
    - При помощи автомата Бюхи удобно проверять допустимость трасс.
    - При помощи темпоральных формул удобно описывать свойства правильности.
    - Для всякой LTL-формулы f существует автомат Бюхи, который допускает те и только те прогоны, которым соответствуют трассы, на которых выполнима f
- <u>Переход от LTL к автоматам</u>
    - f выполняется на всех вычислениях <=>
    - !f не выполняется ни на одном вычислении <=>
    - автомат never {!f} не допускает ни одного прогона, полученного в результате синхронного выполнения с системой

Использование LTL в Spin

- SPIN как раз и занимается тем, что преобразует LTL-формулы в автомат Бюхи, описываемый конструкцией never.

- Если во время верификации нашлась трасса, принадлежащая языку автомата Бюхи (т.е, конструкция never выполнилась, и мы дошли до её конца), SPIN выдаст **контрпример**, содержащий эту трассу.

Система Spin и язык Promela

Система Spin. Процесс моделирования и верификации при помощи системы Spin. Конечность моделей на Promela. Асинхронное выполнение моделей. Недетерминированный поток управления. Понятие выполнимости оператора.

Верификация программы на модели

- Мы хотим задавать, как система устроена и как она должна быть устроена
- Таким образом, нужно две нотации:
  - чтобы описать поведение (устройство системы)
  - чтобы описать требования (свойства правильности)
- Верификатор:
  - проверяет, что устройство системы удовлетворяет свойствам правильности
  - выбранная нотация гарантирует разрешимость проверки правильности любого свойства любой модели
- Все держится на трех китах
  - *SPIN* – Simple Promela Interpreter
    - верификация
    - моделирование
  - *Promela* – Process Meta Language - описание поведения модели
    - недетерминированный язык с охраняемыми командами
    - задача языка – разрешить описывать такие модели, которые могут быть верифицированы
  - *LTL* – Linear Temporal Logic - описание свойств

Конечность моделей на Promela

- У моделей – конечное число состояний (потенциально бесконечные элементы моделей в Promela ограничены)
  - гарантирует разрешимость верификации
- Почему число состояний конечно?
  - Число активных процессов конечно (от 0 до 255)
  - У каждого процесса – ограниченное количество операторов
  - Диапазоны типов данных ограничены
  - Размер всех каналов сообщений ограничен

Асинхронное выполнение моделей

- нет глобальных часов
- по умолчанию синхронизация отсутствует

Недетерминированный поток управления

- абстракция от деталей реализации
- Два уровня недетерминизма
  - внешний (выбор процесса)

        - процессы выполняются параллельно и асинхронно (между двумя последовательными операторами одного процесса может быть сколь угодно длинная пауза)
        - произвольная диспетчеризация процессов
        - выполнение операторов разных процессов происходит в произвольном порядке (основные операторы выполняются атомарно)
- внутренний (выбор действия в процессе).
    - в теле одного процесса также допускается недетерминированное ветвление

Понятие выполнимости оператора

- с любым оператором связаны понятия предусловия и эффекта
- оператор выполняется (производя эффект), только если предусловие истинно, в противном случае он заблокирован
    - Пример 1: *q?m*; если канал *q* не пуст, читаем из него сообщение, иначе ждём
    - Пример 2: *(x > y)* -> *y++*; процесс будет заблокирован до тех пор, пока *x* не станет больше *y*. После этого *y* увеличится на единицу.

Язык Promela. Основные компоненты модели на языке Promela. Процессы, локальные и глобальные объекты данных, каналы сообщений.

Устройство модели. Существует три типа объектов:

- процессы
- глобальные и локальные объекты данных
- каналы сообщений

Процессы

- Поведение процесса задаётся в объявлении типа процесса (proctype)
- Экземпляр процесса – инстанциация proctype
- Два вида инстанцирования процессов:
    - В начальном состоянии системы - *префикс active*
    - В произвольном достижимом состоянии системы - *оператор run*

Локальные и глобальные объекты данных. Два уровня видимости:

- глобальный (данные видны всем активным процессам)
- локальный (данные видны только одному процессу)
    - есть особенность: локальная переменная, объявленная в теле процесса, видна во всем процессе (нет понятия *область видимости переменной* внутри процесса)

Каналы сообщений

- каналы бывают двух типов:
    - буферизованные (асинхронные)
    - небуферизованные (синхронные, рандеву)
- пример объявления канала: chan x = [10] of {int, short, bit};
    - *10* - максимальное число сообщений в канале. 0 определяет канал рандеву.
    - *{int, short, bit}* - структура пересылаемых сообщений

Язык Promela. Механизмы взаимодействия процессов в языке Promela. Глобальные переменные, каналы сообщений, явная синхронизация.

см. предыдущий вопрос

Явная синхронизация:

```
active proctype A() provided (toggle == true){
  L: cnt++;
  printf("A: cnt=%d\n", cnt);
  toggle = false;
  goto L
}
```

```
active proctype B() provided (toggle == false){
  L: cnt--;
  printf("B: cnt=%d\n", cnt);
  toggle = true;
  goto L
}
```

Процесс выполняется, только если значение provided clause равно true. По умолчанию значение равно true

Язык Promela. Основные операторы языка Promela. Операторы-выражения, присваивания.

Основные операторы языка Promela

- задают элементарные преобразования состояний,
- размечают дуги в системе переходов соответствующего процесса,
- их немного – всего 6 типов:
  - выражение
  - присваивание
  - печать
  - проверка свойства безопасности (assert)
  - отправка сообщения
  - прием сообщения
- оператор может быть:
  - выполнимым: *может* быть выполнен,
  - заблокированным: (пока что) *не может* быть выполнен,
  - выполнимость может зависеть от глобального состояния.

Оператор-выражение

- выполним только если выражение не равно 0 (истинно):
  - 2 < 3 – выполним всегда,
  - x < 27 – выполним только если значение x < 27,
  - 3 + x – выполним только если x != -3.

Оператор-присваивание

- всегда безусловно выполним, меняет значение только одной переменной, расположенной слева от “=”.

Язык Promela. Основные операторы языка Promela. Отладочная печать, операторы skip, true, run, assert.

Основные операторы языка Promela

- присваивание: x++, x--, x = x + 1, x = run P();
- выражение: (x), (1), run P(), skip, true, else, timeout;
- печать: printf(“x = %d\n”, x);
- ассерт: assert(1+1 == 2), assert(false);
- отправка сообщения: q!m;
- приём сообщения: q?m;

Отладочная печать

- оператор печати *printf*, всегда безусловно выполним, на состояние не влияет

Псевдо-операторы

- skip: всегда выполним, без эффекта, эквивалент выражения (1),
- true: всегда выполним, без эффекта, эквивалент выражения (1),
- run: 0 если при создании процесса превышен лимит, pid в противном случае.

Оператор assert

- всегда выполнимо, не влияет на состояние системы,
- Spin сообщает об ошибке, если значение выражения равно 0 (false),
- используется для проверки свойств безопасности.

Язык Promela. Чередование (интерливинг) операторов. Внешний и внутренний недетерминизм. Управление выполнимостью операторов.

Чередование (интерливинг) операторов

- процессы выполняются параллельно и асинхронно, *между двумя последовательными операторами одного процесса может быть сколь угодно длинная пауза*,
- произвольная диспетчеризация процессов,
- выполнение операторов разных процессов происходит в произвольном порядке, *основные операторы выполняются атомарно*,
- в теле одного процесса также допускается недетерминированное ветвление

Внешний и внутренний недетерминизм

- два уровня недетерминизма:
  - внешний (выбор процесса),
  - внутренний (выбор действия в процессе).

Управление выполнимостью операторов Основной инструмент управления выполнимостью операторов в Promela – выражения (expressions). (a + b) -> c++; Так же существуют управляющие конструкции if..fi и do..od

Язык Promela. Задание потока управления последовательного процесса. Управляющие конструкции if, do. Организация внутреннего недетерминизма.

Задание потока управления последовательного процесса 5 способов задать поток управления:

- последовательная композиция(“;”), метки, goto,
- структуризация (макросы и inline),
- атомарные последовательности (atomic, d_step),
- недетерминированный выбор и итерации (if..fi, do..od),
- escape-последовательности ({...}unless{...}).

Специальные выражения и переменные

- else – true, если ни один оператор процесса не выполним,
- timeout – true, если ни один оператор модели не выполним,
- _ – переменная, доступная только по записи, значение не сохраняет,
- _pid – минимальный доступный pid,
- _nr_pr – число активных процессов.

Язык Promela. Каналы сообщений. Операторы отправки и приёма сообщений. Тип mtype. синхронная и асинхронная передача сообщений.

Каналы сообщений

- сообщения передаются через каналы (очереди/буфера ограниченного объёма)
- каналы бывают двух типов:
  - буферизованные (асинхронные),
  - небуферизованные (синхронные, рандеву);

chan x = [10] of {int, short, bit};

Операторы отправки и приема сообщений Отправка: ch!expr1,...exprn

- значения expri соответствуют типам в объявлении канала;
- выполнимо, если заданный канал не полон;

Приём: ch?const1 или var1,...constn или varn

- значения vari становятся равны соотв. значениям полей сообщения;
- значения consti ограничивают допустимые значения полей;
- выполнимо, если заданный канал не пуст и первое сообщение в канале соответствует всем константным значениям в операторе приёма сообщения;

Объявление mtype

- способ определить символьные константы (до 255)

Объявление mtype: mtype = {foo, bar}; mtype = {ack, msg, err, interrupt};

Объявление переменных типа mtype: mtype a; mtype b = foo;

Синхронная и асинхронная передача сообщений

- Асинхронная передача
  - асинхронные сообщения буферизуются для последующего приёма, пока канал не полон,
  - отправитель блокируется, когда канал полон,
  - получатель блокируется, когда канал пуст.
- Синхронная передача
  - ёмкость канала равна 0 - chan ch = [0] of {mtype};
  - передача сообщений методом “рандеву”,
  - не хранит сообщения,
  - отправитель блокируется в ожидании получателя, и наоборот,
  - отправка и приём выполняются атомарно.

Язык Promela. Каналы сообщений. Вспомогательные операции с каналами сообщений.

Другие операции с каналами

- len(q) – возвращает число сообщений в канале,
- empty(q) – возвращает true, если q пуст,
- full(q) – возвращает true, если q полон,
- nempty(q) – вместо !empty(q) (в целях оптимизации),
- nfull(q) – вместо !full(q) (в целях оптимизации).
- q?[n,m,p]
  - булевое выражение без побочных эффектов,
  - равно true только когда q?n,m,p выполнимо, однако не влияет на значения n,m,p и не меняет собержимое канала q;
- q?<n,m,p>
  - выполнимо тогда же, когда и q?n,m,p; влияет на значения n,m,p так же, как q?n,m,p, однако не меняет содержимое q;
- q?n(m,p)
  - вариант записи оператора приёма сообщения (т.е. q?n,m,p),
  - может использоваться для отделения переменной от констант
- q!!n,m,p – аналогично q!n,m,p, но сообщение n,m,p помещается в канал q сразу за первым сообщением, меньшим n,m,p;
- q??n,m,p – аналогично q?n,m,p, но из канала может быть выбрано любое сообщение (не обязательно первое).

Имя канала может быть локальным или глобальным, но канал сам по себе – всегда глобальный объект

Язык Promela. Основные типы данных. Область видимости данных.

Основные типы данных

Тип	Диапазон	Пример объявления
bit	0..1	bit turn = 1;
bool	false..true	bool flag = true;
byte	0..255	byte cnt;
chan	1..255	chan q;
mtype	1..255	mtype msg;
pid	1..255	pid p;
short	$-2^{15}..2^{15}-1$	short s = 100;

int $-2^{31}..2^{31}-1$ int x = 1;
unsigned $0..2^{n}-1$ unsigned u : 3; // 3 бита [?]

- по умолчанию все объекты (и локальные и глобальные) инициализируются нулём;
- все переменные должны быть объявлены до первого использования;
- переменная может быть объявлена где угодно.

'Одномерные массивы'

byte a[27]; bit flags[4] = 1;

- все элементы массива инициализируются одним значением,
- индексы нумеруются с 0;

'Пользовательские типы данных'

typedef record { short f1; byte f2 = 4; } record rr; rr.f1 = 5;

Только два уровня видимости

- глобальный (данные видны всем активным процессам),
- локальный (данные видны только одному процессу)
    - подобластей (напр. для блоков) нет,
    - локальная переменная видна везде в теле процесса;