Ключевые понятия по курсу

«Математические методы верификации программ и схем»

*Лектор: Владимир Анатольевич Захаров*

*Составитель: avasite*

*Здесь описан основной смысл понятий и упоминание некоторых понятий. Это* ***не*** *полноценный теормин.*

*В этом документе освящены только лекции 1-6.*

Оглавление

[1. Лекция 1. Введение. 2](#_Toc472009257)

[2. Лекция 2. Общие принципы дедуктивной верификации программ. Операционная семантика императивных программ. Формальная постановка задачи верификации программ. Логика Хоара: правила вывода и свойства. Автоматизация проверки правильности программ. 2](#_Toc472009258)

[3. Лекция 3. Моделирование схем. Системы переходов - модели Крипке. Представление систем переходов формулами логики предикатов первого порядка. Синхронные и асинхронные схемы. Степень детализации представления. Трансляция описаний программ и схем в модели Крипке. 3](#_Toc472009259)

[4. Лекция 4. Темпоральная логика деревьев вычислений CTL. Синтаксис и семантика CTL. Примеры спецификаций моделей в терминах формул CTL. Темпоральная логика линейного времени PLTL. Синтаксис и семантика PLTL. Свойства живости и безопасности. Ограничения справедливости. Задача верификации моделей (model-checking). 3](#_Toc472009260)

[5. Лекция 5. Табличный алгоритм верификации моделей для CTL. Обоснование корректности и сложности табличного алгоритма верификации моделей. Проблема “комбинаторного взрыва”. Символьные средства описания моделей. Двоичные разрешающие диаграммы (BDD). Алгоритм редукции BDD к каноническому виду (OBDD). Выполнение операций над OBDD: унарные и бинарные булевы операции, квантификация, проверка выполнимости, подсчет числа единиц. Общие представления о сложности в классе OBDD. 5](#_Toc472009261)

[6. Лекция 6. Представления неподвижной точки в CTL. Алгоритм символьной верификации моделей в CTL. 6](#_Toc472009262)

1. Лекция 1. Введение.

Формальные методы верификации

1. Проверка эквивалентности

*Основной принцип*: функциональность более высокого уровня абстракции не должна нарушаться при переходе на более низкий уровень.

1. Дедуктивный анализ (проверка выполнения спецификаций для отношения, заданного формальной программой (теоретически тут модель - бесконечна))
2. Верификация моделей программ (проверка выполнимости спецификации на конечной модели (посредством обхода всех состояний))
3. Лекция 2. Общие принципы дедуктивной верификации программ. Операционная семантика императивных программ. Формальная постановка задачи верификации программ. Логика Хоара: правила вывода и свойства. Автоматизация проверки правильности программ.

Два способа описания семантик компьютерных программ:

1. *Денотационный* – функция описывается как композиция других функций (функциональное программирование)
2. *Операционный* – описывается отношение перехода из одного состояния в другое (императивное программирование)

Операционная семантика программ:

1. *Состояние вычисления* – это пара *состояния управления* (программа) и *оценки переменных* (отображение множества переменных на термы).
2. *Отношение переходов* – это бинарное отношение перехода из одного состояния вычисления в другое (нужно раскрыть присваивания, сравнения, циклы, последовательные программы)

*Частичное вычисление* – программы пи для оценки переменных тета - соответствующая последовательность состояний вычислений.

*Вычисление* – максимальное частичное вычисление (нельзя продолжить), если оно конечно, то будет *результат вычисления*.

*Частично-корректная программа* – если для любых входных данных удовлетворяющих *предусловию* фи, результат вычисления удовлетворяет *постусловию* пси.

Задача верификации заключается в проверке частичной корректности.

*Тотальная корректность* – если программа обязательно останавливается при любой интерпретации и удовлетворяет постусловию.

*Триплет Хоара* – это тройка: предусловие {программа} постусловие.

Выполнимость триплетов Хоара в интерпретациях, логика Хоара (правила вывода для императивной программы) (ass, cons, comp, if, while) – для этих правил есть *теорема Корректности*.

Вывод в логике Хоара – это корневое дерево, листья – формулы логики предикатов (на них разваливается программа при применении логики Хоара).

*Успешный вывод* в интерпретации I, если дерево вывода конечно и все листья – это истинные предикаты в интерпретации I.

***Полнота правил вывода Хоара***:

1. Нету системы правил вывода, позволяющей построить для любой интерпретации I и для каждого триплета успешный вывод и доказать его успешность.

Следует из теоремы Гёделя о неполноте всякой рекурсивно-перечислимой системы аксиом арифметики натуральных чисел.

Некоторый пример: теорему Ферма фиг докажешь.

1. Нету системы правил вывода, позволяющей построить для любой интерпретации I и для каждого триплета успешный вывод, но без доказательства его успешности.

Невозможно так как могут не найтись такие пред- и пост- условия для правила CONS, которые выразят нужные соотношения между переменными.

1. Существуют такие интерпретации, что для любого триплета можно построить успешный вывод.

Да, так как достаточно, чтобы для любого цикла существовал такой терм, который для любой начальной оценки переменных примет значение n+1 <=> цикл в вычислении совершает лишь n и итераций.

Стратегия вывода в логике Хоара:

1. *Слабейшее предусловие* (для заданной программы и постусловия).

Может быть вычислено автоматически для всех, кроме WHILE-GEN – для него нужен инвариант, который автоматически – трудно придумать.

1. Лекция 3. Моделирование схем. Системы переходов - модели Крипке. Представление систем переходов формулами логики предикатов первого порядка. Синхронные и асинхронные схемы. Степень детализации представления. Трансляция описаний программ и схем в модели Крипке.

*Вычисление* – бесконечная последовательность состояний, каждое из которых получено из предыдущего посредством некоторого перехода.

*Модель Крипке* – размеченный граф переходов. Каждое состояние помечается набором свойств, истинных в этом состоянии.

Состояния бывают *достижимые*, *тупиковые*.

*Отношение переходов*.

Синхронные системы можно выразить через конъюнкцию (и) возможных переходов.

Асинхронные системы через дизъюнкцию (или) возможных переходов.

1. Лекция 4. Темпоральная логика деревьев вычислений CTL. Синтаксис и семантика CTL. Примеры спецификаций моделей в терминах формул CTL. Темпоральная логика линейного времени PLTL. Синтаксис и семантика PLTL. Свойства живости и безопасности. Ограничения справедливости. Задача верификации моделей (model-checking).

*Событие* – состояние системы.

*Трасса* – последовательность событий (вычисление системы).

*Свойство* – множество трасс.

Модель Крипке является *уточнением* другой модели Крипке, если множество её трасс является подмножеством множества трасс другой модели Крипке.

***Свойства безопасности***: отсутствие безопасности – если что-то плохое случилось, то этого уже не исправить.

***Свойства живости***: чтобы ни случилось вначале, потом всегда можно достичь своей цели.

Утверждения:

1. Если свойство P – это и свойство живости, и свойство безопасности, то P – это все возможные трассы.
2. Для любого свойства P существуют такие свойства живости и безопасности, что их пересечение есть свойство P.

***Ограничения справедливости***: ограничения, необходимые для соблюдения принципа чередования (чтобы не залипать на одной компоненте системы).

*Слабая справедливость*: если путь почти всегда проходит через состояния, в которых может быть выполнено некоторое действие, то оно выполняется бесконечно часто. (почти всегда означает «с какого-то момента – всегда»)

*Сильная справедливость*: если путь бесконечно часто проходит через состояния, в которых может быть выполнено некоторое действие, то оно выполняется бесконечно часто.

Темпоральные логики предназначены для описания свойств вычислений реагирующих систем, т.е. для задания множества допустимых трасс.

CTL\* - описывают свойства деревьев вычислений (развёртка модели Крипке из некоторого начального состояния).

1. A – для всех путей
2. E – для некоторого пути
3. X – следующий момент времени этого пути
4. F – когда-то в будущем этого пути
5. G – в каждом состоянии в будущем этого пути
6. P1 U P2 – P1 соблюдается до некоторого состояния, а на следующем будет P2
7. P1 R P2 – P2 выполняется до состояния, где выполняется P1, но это может не наступить

Формулы CTL\* - это формулы состояний.

Базовые: or, not, X, U, E.

CTL – логика деревьев вычислений (AX, AF, AG, AU, AR, EX, EF, EG, EU, ER).

В логике ветвящегося времени темпоральные операторы находятся непосредственно под действием кванторов по тем путям, которые исходят из заданного состояния.

Базовые: или, не, EX, EG, EU.

LTL – линейная темпоральная логика – формулы вида Af, где f – формула пути, в которой все формулы состояния – атомарные высказывания.

В логике линейного времени операторы предназначены для описания событий на протяжении единственного пути вычисления.

CTL\*, CTL, LTL – имеют разные выразительные мощности.

Не существует CTL, эквивалентной LTL формуле A(FG stable)

Не существует LTL, эквивалентной СЕД формуле AG (EF restart)

Дизъюнкция A(FG stable) or AG (EF restart) – является CTL\*

1. Лекция 5. Табличный алгоритм верификации моделей для CTL. Обоснование корректности и сложности табличного алгоритма верификации моделей. Проблема “комбинаторного взрыва”. Символьные средства описания моделей. Двоичные разрешающие диаграммы (BDD). Алгоритм редукции BDD к каноническому виду (OBDD). Выполнение операций над OBDD: унарные и бинарные булевы операции, квантификация, проверка выполнимости, подсчет числа единиц. Общие представления о сложности в классе OBDD.

***Табличные алгоритмы*** – алгоритмы верификации моделей, работающие с явным представлением моделей. Итеративно заполняют таблицу выполнимости подформул заданной формулы CTL во всех состояниях модели.

Для (EG фи) нужно выделять сильно-связные компоненты (нетривиальные, или с 1 вершиной с петлёй), у которых выполнено условие фи. Для остальных EX, EG, или, не – разметка очевидна.

Был приведён табличный алгоритм проверки выполнимости произвольной CTL формулы фи модели М за время O(|фи|\*(|S|+|R|))

Model checking для CTL в ограничениях справедливости. Ограничения справедливости – это множество подмножеств состояний модели. Пересечение с каждым из этих подмножеств inf(путь) должно быть не пусто, чтобы путь был справедливым.

Сильно-связная компонента справедлива относительно ограничений справедливости <=> для каждого из подмножеств ограничений найдётся состояние, принадлежащее этой компоненте.

Для проверки с учётом справедливости, необходимо каждое свойство проверять вместе с (EG true) – с учётом справедливости. (т.е. выкидываются компоненты, которые не подходят для справедливого EG true)

Приведённый табличный алгоритм с учётом справедливости имеет сложность O(|фи|\*(|S|+|R|)\*|F|), где F – ограничения справедливости.

Табличный алгоритм:

1. Простой
2. Время выполнения пропорционально числу состояний – это плохо для больших моделей
3. Затратный по памяти, нужно хранить все состояния модели

***Символьные алгоритмы*** – основаны на средствах символьного описания множеств двоичных векторов (а не массивов перечислений).

*Двоичные разрешающие диаграммы (binary decision diagrams - BDD)* – структуры представления размеченных систем переходов с конечным числом состояний. – корневой ориентированный ациклический граф, вершины которого разбиты на 2 класса: терминальные и нетерминальные вершины.

*OBDD – ordered binary decision diagrams* – каноническая форма представления булевых функций. (компактнее КНФ и ДНФ и с ними можно эффективно работать).

*Двоичное разрешающее дерево* – это корневое ориентированное дерево, вершины которого разбиты на 2 класса: *терминальные вершины* и *нетерминальные вершины*.

Построение разрешающих диаграмм:

1. В каждом пути из корня в терминальную вершину переменные должны следовать в одном и том же порядке.
	1. Достаточно упорядочить переменные (выбор оптимального порядка – NP-полная задача)

(пример: ROBDD функции, вычисляющей n-й бит произведения двоичных чисел будет расти экспоненциально вместе с количеством переменных, независимо от упорядочивания)

1. В диаграмме не должно быть изоморфных поддеревьев или избыточных вершин
	1. Удаление повторных вхождений терминалов
	2. Удаление повторных вхождений нетерминалов
	3. Удаление избыточных проверок

ROBDD – редукционная упорядоченная двоичная разрешающая диаграмма – BDD, к которой не применимы 3 правила, описанные выше.

Выполнимость может быть установлена путём проверки эквивалентности исследуемой ROBDD и вырожденной ROBDD, состоящей из единственной терминальной вершины, помеченной 0. - ???

Совершение операций над ROBDD: основано на разложении шеннона и динамическом программировании (используется кеш-результатов).

<жутковатое построение OBDD по модели Крипке>

1. Лекция 6. Представления неподвижной точки в CTL. Алгоритм символьной верификации моделей в CTL.

Для представления неподвижной точки используется решётка на множестве состояний модели Крипке. (решётка строится как множество всех подмножеств состояний модели Крипке). *Преобразователь* (отображение из решётки в решётку) может быть монотонным и непрерывным по объединению и пересечению. Предикат (элемент решётки, т.е. некоторое подмножество множества состояний) является ***неподвижной точкой*** преобразователя, если преобразователь переводит его в его же.

У монотонного преобразователя всегда есть наименьшая и наибольшая неподвижная точка.

Леммы:

1. Если S – конечное множество состояний, а преобразователь монотонен, то он также непрерывен по объединению и пересечению.
2. <ещё несколько лемм>

Нижняя и верхняя неподвижные точки могут быть найдены посредством нужного количества рекурсивных применений преобразователя к false и к true.

QBF – квантифицированные булевы формулы, задаваемые над пропозициональными переменными.

Оценка значений истинности.