# Математические методы верификации программ и схем

ЛЕКТОРЫ:

Владимир Анатольевич Захаров,
Владислав Васильевич Подымов

31 августа 2019 г.

# ПРОГРАММА КУРСА

1. Введение. Задача верификации аппаратуры и программного обеспечения

# ПРОГРАММА КУРСА

1. Введение. Задача верификации аппаратуры и программного обеспечения
2. Дедуктивный метод верификации программ

# ПРОГРАММА КУРСА

1. Введение. Задача верификации аппаратуры и программного обеспечения
2. Дедуктивный метод верификации программ
3. Дискретные модели параллельных программ и схем

# ПРОГРАММА КУРСА

1. Введение. Задача верификации аппаратуры и программного обеспечения
2. Дедуктивный метод верификации программ
3. Дискретные модели параллельных программ и схем
4. Темпоральные логики

# ПРОГРАММА КУРСА

1. Введение. Задача верификации аппаратуры и программного обеспечения
2. Дедуктивный метод верификации программ
3. Дискретные модели параллельных программ и схем
4. Темпоральные логики
5. Верификация моделей программ для темпоральной логики деревьев вычислений CTL: табличный и символьный подход

# ПРОГРАММА КУРСА

1. Введение. Задача верификации аппаратуры и программного обеспечения
2. Дедуктивный метод верификации программ
3. Дискретные модели параллельных программ и схем
4. Темпоральные логики
5. Верификация моделей программ для темпоральной логики деревьев вычислений CTL: табличный и символьный подход
6. Верификация моделей программ для темпоральной логики линейного времени LTL: теоретико-автоматный подход

# ПРОГРАММА КУРСА

7. Методы повышения эффективности верификации: абстракция, симуляция, редукция частичных порядков, ограниченная верификация

# ПРОГРАММА КУРСА

7. Методы повышения эффективности верификации: абстракция, симуляция, редукция частичных порядков, ограниченная верификация
8. Модели вычислительных систем реального времени

# ПРОГРАММА КУРСА

7. Методы повышения эффективности верификации: абстракция, симуляция, редукция частичных порядков, ограниченная верификация
8. Модели вычислительных систем реального времени
9. Верификация моделей программ реального времени

# ПРОГРАММА КУРСА

7. Методы повышения эффективности верификации: абстракция, симуляция, редукция частичных порядков, ограниченная верификация
8. Модели вычислительных систем реального времени
9. Верификация моделей программ реального времени
10. Методы статического анализа моделей программ

# ПРОГРАММА КУРСА

7. Методы повышения эффективности верификации: абстракция, симуляция, редукция частичных порядков, ограниченная верификация
8. Модели вычислительных систем реального времени
9. Верификация моделей программ реального времени
10. Методы статического анализа моделей программ
11. Методы уточнения абстракций по контрпримерам

# ПРОГРАММА КУРСА

Конспекты лекций будут размещаться по адресу

http://mk.cs.msu.ru

# Литература


- Э.М. Кларк, О. Грамберг, Д. Пелед. Верификация моделей программ: Model Checking. Изд-во МЦНМО, 2002.
- Ю.Г. Карпов. Model Checking: верификация параллельных и распределенных программных систем. Изд-во БХВ-Петербург, 2010.
- K. R. Apt, E.-R. Olderog. Verification of sequential and concurrent programs, Springer, 1997.
- B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, P. Schnoebelen. Systems and Software Verification: Model-Checking Techniques and Tools. Springer, 2001.
- **Baier C., Katoen J.-P. Principles of model checking, MIT Press, 2008.**

# Лекция 1.

## Задача верификации аппаратуры и программного обеспечения

1. Зачем нужна формальная верификация программ?
2. Основные подходы к задаче формальной верификации.
3. Принципы верификации моделей программ.
4. Исторические сведения.
5. Достижения методов формальной верификации программ.

# Зачем нужна формальная верификация программ?

Задачу спецификации и верификации программ принято рассматривать как некоторую вспомогательную деятельность, как неизбежный «довесок» к основному занятию программиста — разработке программного кода.

Однако в начале 80-х годов прошлого века разработчики программного обеспечения обнаружили, что свыше половины издержек, связанных с изготовление программного продукта или микросхемы, приходятся именно на этап верификации, отладки и устранения погрешностей, причем эта доля неуклонно возрастает.

Среди наиболее значимых причин, объясняющих это явление, можно выделить следующие.

# Зачем нужна формальная верификация программ?

## 1. Возрастание стоимости ущерба из-за пропущенной ошибки в программе

Аппаратные и программные системы широко используется во многих приложениях, где любой отказ считается недопустимым, таких как

- управление оружием,
- энергетика,
- электронная коммерция,
- телекоммуникация,
- управление транспортом,
- медицинская аппаратура,

и список этот довольно велик.

# Зачем нужна формальная верификация программ?

## Пример 1.

22 июля 1962 г. первая ступень ракеты Маринер-1 была взорвана через 293 секунды после старта. Антенна аппарата потеряла связь с наводящей системой на Земле, в результате управление взял на себя бортовой компьютер, программа которого содержала ошибку.

В отчете НАСА, отправленном в Конгресс в 1963, упоминается: «Пропуск дефиса в редактировании данных вынудил компьютер дать серию ненужных сигналов о коррекции курса, которые сбили корабль с курса и привели к его уничтожению.»

# Зачем нужна формальная верификация программ?

## Пример 2.

2 сентября 1988 г. на борт советской межпланетной станции "Фобос-1" была передана неверная команда. При обработке возникшей исключительной ситуации бортовой компьютер воспринял эту команду как команду на отключение системы стабилизации и ориентации.

Вследствие этого станция лишилась управления и затерялась в космосе.

# Зачем нужна формальная верификация программ?

## Пример 3.

4 июня 1996 г. европейская космическая ракета «Ариан-5» взорвалась менее чем через сорок секунд после запуска. Причиной послужила ошибка в программе компьютера, отвечавшего за расчеты движения ракеты. В ходе запуска возникла исключительная ситуация, когда большое 64-разрядное число с плавающей точкой было преобразовано в 16-разрядное целое число. Это преобразование не было защищено кодом для обработки исключительных ситуаций, что и привело к сбою компьютера. Та же самая ошибка привела к сбою дублирующего компьютера. В результате на бортовой компьютер ракеты были переданы неверные данные о высоте, что вызвало уничтожение ракеты.

Стоимость ущерба — 370 000 000 $.

# Зачем нужна формальная верификация программ?

## Пример 4.

В 1999 г. Один из субподрядчиков NASA поставил программное обеспечение в котором использовалась английская система мер, в то время как измерительное оборудование было откалибровано по метрической системе. Это послужило причиной аварии спутника, отправленного на Марс.

Стоимость ущерба составила 327 000 000 $.

# Зачем нужна формальная верификация программ?

## Пример 5.

15 января 2005 г. космический зонд "Гюйгенс" отделился от автоматической межпланетной станции "Кассини" , достигшей после 7-летнего полета орбиты Титана — спутника Сатурна, — и начал спуск в атмосферу спутника и передачу информации о поверхности Титана.

Ошибка в бортовой программе зонда привела к тому, что половина переданной информации была потеряна.

Затраты на разработку зонда составили 2 000 000 000 $.

# Зачем нужна формальная верификация программ?

## Пример 6.

Медицинский аппарат радиационной терапии Therac-25 период 1985-87 гг. подавал некоторым пациентам дозы облучания, превышающие предписанные врачами в 100 раз. Было обнаружено, что причиной послужила ошибка в программном обеспечении, управляющим аппаратом. Превышение дозы облучения послужило непосредственной причиной смерти по крайней мере трех пациентов.

Расследование показало, что при подаче некоторых последовательностей управляющих команд интенсивность облучения увеличивалась в несколько раз по сравнению с предписанной мощностью.

# Зачем нужна формальная верификация программ?

## Пример 7.

В 2000 г. в Панаме несколько пациентов получили передозировку облучения при радиотерапии. Было обнаружено, что управляющая программа задавала мощность излучения в зависимости от порядка, в котором поступали некоторые данные по параллелльным каналам (эффект «гонок»).

Передозировка послужила причиной смерти по меньшей мере пяти пациентов.

# Зачем нужна формальная верификация программ?

## Пример 8.

Программное обеспечение системы управления протвовоздушного ракетного комплекса MIM-104 Patriot содержала ошибку, в силу которой таймеры в разных компьютерах в составе этого комплекса имели тенденцию к рассинхронизации 0.3 сек на 100 часов непрерывной работы.

В результате 25 февраля 1991 г. такой комплекс не смог осуществить перехват иракской тактической ракеты, что привело к гибели 28 военнослужащих армии США.

# Зачем нужна формальная верификация программ?

## Пример 9.

23 марта 2003 г. система управления протвовоздушного ракетного комплекса MIM-104 Patriot ошибочно идентифицировала британский бомбардировщик Tornado как приближающуюся вражескую ракету и отправила команду на запуск ракеты.

В результате британский самолет был сбит; оба его пилота погибли.

# Зачем нужна формальная верификация программ?

## Пример 10.

Во время операции "Буря в пустыне" 24% потерь в живой силе произошло по причине "дружественного огня" .

Выводы многочисленных независимых комисий были однозначны — главная причиной послужили ошибки в программном обеспечении и аппаратуре автоматических систем управления оружием.

# Зачем нужна формальная верификация программ?

## Пример 11.

20 декабря 1995 г. в катастрофе самолета Боинг-757 (рейс Майами–Кали) погибли 159 человек.

Расследование показало, что причина катастрофы — ошибка в одном символе программной системы управления полетом.

Компания Боинг выплатила родственникам жертв этой авиакатастрофы свыше 300 000 000 $.

# Зачем нужна формальная верификация программ?

## Пример 12.

В 1998 г. в США на полгода был задержан ввод в строй новой системы обслуживания авиационного транспорта из-за непрекращающихся сбоев в работе программного обеспечения этой системы. По оценкам экспертов ущерб составил несколько сотен миллионов долларов.

# Зачем нужна формальная верификация программ?

## Пример 13.

Финансовая компания Knight Capital Group потеряла 440 000 000 $ за 45 минут в результате некорректной установки обновленной версии программного обеспечения на серверы, из-за чего в течение некоторого времени происходила попеременная работа старого и нового кода.

# Зачем нужна формальная верификация программ?

## Пример 14.

В 2009 г. японское подразделение швейцарского банка UBS чуть было не потратило 31 000 000 000 $ на покупку облигаций компании Capcom. Покупка была остановлена в последнюю минуту.

Расследование показало, что сотрудник компании отдал команду на покупку облигаций на сумму 310 000 $, но из-за системной ошибки программного обеспечения к заказу добавилось 5 нулей.

# Зачем нужна формальная верификация программ?

## Пример 15.

В 1994 г. при разработке процессоров Intel Pentium была пропущена ошибка во встроенной программе деления — несколько элементов вспомогательного массива не были инициализированы.

Ошибка была обнаружена только после того, как процессоры поступили в продажу; компания была вынуждена провести замену всех дефектных процессоров.

Ущерб оценивается в сотни миллионов долларов.

# Зачем нужна формальная верификация программ?

## Пример 16.

Статистические исследования показывают, что свыше 10% проектов по разработке микроэлектронных схем так и не доводятся до серийного изготовления, ввиду существенных ошибок, которые обнаруживаются в ходе эксплаутации опытных образцов. Стоимость разработки схемы и изготовления ее опытного образца может составлять несколько сотен тысяч долларов.

Исследования, проведенные в 2002 г. Национальным Институтом Стандартов и Технологий, показали, что потери экономики из-за ошибок в программном обеспечения оцениваются суммой 59.5 миллиардов долларов, причем этот ущерб можно сократить на треть за счет усовершенствования методов верификации программ.

# Зачем нужна формальная верификация программ?

## 2. Расширение области применения программного обеспечения

По мере того как возрастает участие программных систем в нашей жизни, растет и бремя ответственности за правильность их функционирования. Безопасность уже нельзя восстановить, просто отключив неправильно работающую систему: отключение устройства несет еще большую опасность. Мы зависим от правильности работы вычислительных устройств.

Поэтому еще более насущной задачей становится разработка методов, способствующих повышению нашей уверенности в правильности работы подобных систем.
Изменяется само понимание задачи верификации: программа может быть признана надежной не потому, что в ней не удалось обнаружить ошибок, а потому, что удалось убедительно **доказать отсутствие ошибок.**

# Зачем нужна формальная верификация программ?

## 3. Программирование усложняется

ОС Microsoft Windows 3.1 (1992 г.) содержит 3 млн строк кода.

ОС Microsoft Windows 98 (1998 г.) содержит 18 млн строк кода.

ОС Microsoft Windows XP (2002 г.) содержит 40 млн строк кода.

Сервисы Google (2015 г.) содержат более 2000 млн строк кода.

# Зачем нужна формальная верификация программ?

## 3. Программирование усложняется

Разработка программ (и уж тем более микроэлектронных схем) становится «каскадным» процессом, на разных этапах которого происходит уточнение модели программы или микроэлектронной схемы, оканчивающееся их реализацией. Ошибки, пропущенные на ранних этапах, невозможно устранить на последующих стадиях проектирования.

Поэтому необходим целый арсенал методов верификации, которые можно было бы применять не только к самим программам (схемам), но и также и к их моделям, представленных на разных уровнях абстракции.

# Зачем нужна формальная верификация программ?

## 4. Возрастание трудоемкости верификации

Вычислительная аппаратура становится дешевле, а производительность компьютеров возрастает.

Появляются новые возможности автоматического решения все более и более сложных задач.

Происходит увеличение объемов программного кода.

Вследствие этого, увеличивается вероятность появления ошибок, которые могли быть внесены в этот код как на этапе проектирования программы, так и в ходе реализации проекта.

# Зачем нужна формальная верификация программ?

## 4. Возрастание трудоемкости верификации

Современные технологии проектрования и разработки программ позволяют повысить производительность труда программистов, и она растет настолько же быстро, насколько увеличивается объем программного обеспечения.

Однако верификация программ рискует оставаться экстенсивным трудом, требующим большого числа квалифицированных работников, занимающихся исключительно поиском ошибок в программном коде. Но даже такой способ не позволяет убедиться в надежности программного продукта.

Нужны совсем другие средства, при помощи которых можно было бы не только обнаруживать ошибки, но и убеждаться в их отсутствии.

# Основные подходы к задаче верификации

Простейшие методы проверки правильности программ и микроэлектронных схем — имитационное моделирование и тестирование .

**Имитационному моделированию** подвергается абстрактная схема или прототип, **тестирование** применяется непосредственно к самому продукту.
В случае электронных схем, например, имитационному моделированию подвергается проект разрабатываемой схемы, тогда как тестированию подвергается сама микросхема. В обоих случаях в этих методах определенные сигналы обычно вводятся в заданных точках схемы, а в других точках снимают соответствующие показания.

Эти методы могут быть весьма экономичны для выявления многих ошибок. Но проверить **ВСЕ** возможные взаимодействия и обозреть **ВСЕ** мыслимые «сценарии», применяя лишь моделирование и тестирование, вряд ли удастся.

# Основные подходы к задаче верификации

Формальные методы верификации предназначены для **доказательства** того, что в проектируемой системе **НЕТ** ошибок определенного вида.

Основные подходы к формальной проверке правильности сложных управляющих систем (программ и проектов микроэлектронной аппаратуры):

1. **проверка эквивалентности** (equivalence checking),
2. **дедуктивный анализ** (proof-theoretic approach),
3. **верификация моделей программ** (model checking),
4. **оперативная верификация** (runtime verification),

# Основные подходы к задаче верификации

## Проверка эквивалентности

Каждая управляющая система определяется двумя составляющими:

- управляющая схема $\Sigma$ (синтаксическая характеристика),
- вычисляемая функция $F_\Sigma$ (функциональная характеристика).

Задача проверки эквивалентности: для заданной пары схем $\Sigma_1$ и $\Sigma_2$ проверить выполнимость соотношения

$$F_{\Sigma_1} = F_{\Sigma_2} \ .$$

# Проверка эквивалентности

Система на кристалле при проектировании представляется на разных уровнях абстракции. Маршрут проектирования Synopsis включает в себя три основных уровня:

1. Системный уровень:
   - Этап 1: создание алгоритмической модели и ее верификация;
   - Этап 2: создание модели уровня транзакций (модели макроархитектуры) и ее верификация;
   - Этап 3: создание т.н. "золотой модели" системы на кристалле как технического задания для проектирования схемы.

# Проверка эквивалентности

2. Логический , или вентильный , уровень, состоящий из пяти этапов:

- Этап 1: создание синтезируемой RTL (Register-Transfer Level)-модели системы на кристалле на одном из языков описания аппаратуры (Verilog, VHDL);
- Этап 2: логическая верификация RTL-модели посредством моделирования;
- Этап 3: физическая верификация посредством прототипирования;
- Этап 4: синтез в базисе вентильных схем и автоматическая генерация тестовых структур для контроля годности;
- Этап 5 логическая и формальная верификация списка цепей.

3. Топологический уровень

# Проверка эквивалентности

Таким образом, на протяжении проектирования СБИС последовательно строятся 5 описаний системы:

- Алгоритмическое описание системы (на языке C/C++);
- Макроархитектурное описание системы (на языке System-C);
- Микроархитектурное (RTL) описание системы (на языках VHDL или Verilog);
- Описание списка цепей (на языках VHDL или Verilog и в форматах IEEE PDEF, DEF, LEF);
- Описание топологии СБИС (в формате GDSII).

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

# Проверка эквивалентности

Возникает следующая задача верификации.

$Log2Top$ : Логическая схема $\Sigma$

$\Downarrow$

Топологическая реализация схемы $T$ ,

$Mod$ : Топологическая реализация схемы $T$

$\Downarrow$

Улучшенная топологическая реализация схемы $T'$

$Top2Log$ : Топологическая реализация схемы $T'$

$\Downarrow$

Логическая схема $\Sigma'$ .

Задача: проверить выполнимость соотношения $F_{\Sigma} = F_{\Sigma'}$ .

Для логических схем эта задача сводится к проблеме выполнимости булевых формул SAT.

# Дедуктивный анализ

Термин дедуктивный анализ подразумевает применение аксиом и правил вывода для доказательства правильности функционирования системы.

На ранних этапах исследований по дедуктивному анализу основное внимание уделялось обеспечению правильности работы критически важных систем, и такие доказательства строились исключительно вручную.

Со временем появились инструментальные средства (пруверы), позволяющие применять систематический перебор с целью поиска различных направлений построения доказательства.

# Дедуктивный анализ

Принципы дедуктивного анализа программ таковы:

1. Программа (схема) $\pi$ вычисляют отношение $R_\pi$ между данными входе и на выходе программы (схемы);

# Дедуктивный анализ

Принципы дедуктивного анализа программ таковы:

1. Программа (схема) $\pi$ вычисляют отношение $R_\pi$ между данными входе и на выходе программы (схемы);
2. Текст программы (описание схемы) — это формальное описание отношения $R_\pi$ на языке программирования;

# Дедуктивный анализ

Принципы дедуктивного анализа программ таковы:

1. Программа (схема) $\pi$ вычисляют отношение $R_\pi$ между данными входе и на выходе программы (схемы);
2. Текст программы (описание схемы) — это формальное описание отношения $R_\pi$ на языке программирования;
3. Требование (спецификация) правильности $\Phi$ программы — это описание того отношения, которое должна реализовывать программа, но на другом языке (или в другой форме);

# Дедуктивный анализ

Принципы дедуктивного анализа программ таковы:

1. Программа (схема) $\pi$ вычисляют отношение $R_\pi$ между данными входе и на выходе программы (схемы);
2. Текст программы (описание схемы) — это формальное описание отношения $R_\pi$ на языке программирования;
3. Требование (спецификация) правильности $\Phi$ программы — это описание того отношения, которое должна реализовывать программа, но на другом языке (или в другой форме);
4. Проверка правильности программы $\pi$ относительно спецификации $\Phi$ — это доказательство того, что $R_\pi \Rightarrow \Phi$ .

# Дедуктивный анализ

Принципы дедуктивного анализа программ таковы:

1. Программа (схема) $\pi$ вычисляют отношение $R_\pi$ между данными входе и на выходе программы (схемы);
2. Текст программы (описание схемы) — это формальное описание отношения $R_\pi$ на языке программирования;
3. Требование (спецификация) правильности $\Phi$ программы — это описание того отношения, которое должна реализовывать программа, но на другом языке (или в другой форме);
4. Проверка правильности программы $\pi$ относительно спецификации $\Phi$ — это доказательство того, что $R_\pi \Rightarrow \Phi$ .

Обычно спецификация задается предусловием $\varphi$ , описывающим множество данных, которые могут поступать на вход программы, и постусловием $\psi$ , описывающим отношение между входными и выходными данными.

# Дедуктивный анализ

Дедуктивный анализ оказал значительное влияние на подходы к разработке программного обеспечения (например, способствовал введению понятия инварианта ).

Тем не менее, он занимает много времени и может быть осуществлен только экспертами, обладающими знаниями в области логического вывода и имеющими немалый практический опыт. Доказательство правильности отдельного протокола или электронной схемы может занять целые дни или даже месяцы.

Поэтому дедуктивный анализ используется главным образом для проверки программных систем, наиболее чувствительных к дефектам и сбоям, когда можно выделить необходимые ресурсы, для того чтобы гарантировать правильность функционирования системы.

# Дедуктивный анализ

Важно осознавать также, что не существует алгоритма, способного распознать, завершается ли работа произвольной вычислительной программы (написанной на одном из языков программирования, наподобие Си или Паскаля). Это ограничивает и возможности автоматической верификации. В частности, правильность завершения работы программы в общем случае нельзя проверить автоматически. По этой причине большинство систем построения доказательств не могут быть полностью автоматизированы.

Преимущество методов дедуктивного анализа состоит в том, что они могут быть применены к системам с бесконечным числом состояний. Эту задачу можно до определенных пределов автоматизировать. Однако даже в том случае, когда проверяемое свойство действительно выполняется, нельзя наложить никаких ограничений на промежуток времени и объем памяти, необходимый для поиска доказательства.

# Верификация моделей программ

Метод верификации моделей (или проверки выполнимости спецификации на модели, или model checking) применяется для верификации систем с конечным числом состояний. Обычно процедура верификации заключается в исчерпывающем обходе пространства состояний системы. При наличии достаточных ресурсов эта процедура всегда завершается и может быть реализована достаточно эффективным алгоритмом.

Хотя ограничение, связанное с конечным числом состояний системы, очень существенно, метод верификации моделей применим ко многим важным классам вычислительных систем: контроллеры, драйверы, коммуникационные протоколы.

В некоторых случаях системы с бесконечным числом состояний могут быть проверены этим методом в сочетании с разнообразными правилами абстракции и индукции.

# Верификация моделей программ

Поскольку метод проверки на модели может применяться чисто автоматически, он предпочтительнее дедуктивного анализа в тех случаях, когда может быть использован.

Тем не менее, всегда останутся некоторые жизненно важные приложения, для полной верификации которых необходимо проводить доказательство теорем.

В связи с этим можно отметить новое перспективное направление, предусматривающее такую интеграцию дедуктивного анализа и метода проверки на модели, чтобы фрагменты системы, имеющие конечное число состояний, проверялись автоматически.

# Процесс верификации моделей программ

## Принципы model checking

1. Для заданной вычислительной системы (программы или проекта микроэлектронной схемы) построить модель $M$ — дискретную структуру, которая

# Процесс верификации моделей программ

## Принципы model checking

1. Для заданной вычислительной системы (программы или проекта микроэлектронной схемы) построить модель $M$ — дискретную структуру, которая
   - может рассматриваться как интерпретация для некоторой формальной логики, и

# Процесс верификации моделей программ

## Принципы model checking

1. Для заданной вычислительной системы (программы или проекта микроэлектронной схемы) построить модель $M$ — дискретную структуру, которая
   - может рассматриваться как интерпретация для некоторой формальной логики, и
   - описывает (на некотором уровне абстракции) поведение вычислительной системы.

# Процесс верификации моделей программ

## Принципы model checking

1. Для заданной вычислительной системы (программы или проекта микроэлектронной схемы) построить модель $M$ — дискретную структуру, которая
   - может рассматриваться как интерпретация для некоторой формальной логики, и
   - описывает (на некотором уровне абстракции) поведение вычислительной системы.
2. Для технических требований, предъявляемых к системе, сформулировать эти требования на формальном логическом языке — задать спецификацию $\varphi$ .

# Процесс верификации моделей программ

## Принципы model checking

1. Для заданной вычислительной системы (программы или проекта микроэлектронной схемы) построить модель $M$ — дискретную структуру, которая
   - может рассматриваться как интерпретация для некоторой формальной логики, и
   - описывает (на некотором уровне абстракции) поведение вычислительной системы.
2. Для технических требований, предъявляемых к системе, сформулировать эти требования на формальном логическом языке — задать спецификацию $\varphi$ .
3. Проверить выполнимость спецификации $\varphi$ на модели $M$ :

$$M \models \varphi$$

# Процесс верификации моделей программ

Применение метода model checking состоит из нескольких этапов.

# Процесс верификации моделей программ

Применение метода model checking состоит из нескольких этапов.

## Моделирование

Первая задача заключается в приведении проектируемой системы к такому формальному виду, который был бы приемлем для инструментальных средств верификации моделей программ.

Часто это просто задача компиляции. В других случаях, при ограничениях по времени и объему памяти, моделирование может потребовать абстракции, чтобы избавиться от несущественных деталей, не относящихся к делу.

# Процесс верификации моделей программ

## Спецификация

Перед проведением верификации нужно сформулировать свойства, которыми должна обладать проектируемая системы.

Обычно спецификации задаются на языке формальной логики. Для аппаратуры и программного обеспечения, как правило, применяют темпоральную логику , позволяющую описывать, как поведение системы протекает во времени.

Важным вопросом спецификации является полнота . Метод проверки на модели дает возможность убедиться, что модель проектируемой системы соответствует заданной спецификации, однако определить, охватывает ли заданная спецификация все свойства, которыми должна обладать система, невозможно.

# Процесс верификации моделей программ

## Верификация

В идеальном случае верификация проводится полностью автоматически. Для этого используются стандартные алгоритмы алгебры и дискретной математики, к числу которых относятся алгоритмы

- проверки выполнимости булевых формул;
- проверки достижимости заданных вершин в конечном графе;
- проверки достижимости сильно связных компонент в ориентированном графе;
- построения схем (диаграмм), реализующих булевы функции;
- решения систем линейных неравенств.

# Процесс верификации моделей программ

## Уточнение модели

На практике верификация часто требует содействия человека. Одной из сторон деятельности человека является анализ результатов верификации. Если результаты проверки отрицательные, то пользователю нередко предоставляют трассу, содержащую ошибку. Она строится в качестве контрпримера для проверяемого свойства и может помочь проектировщику проследить, где возникает ошибка. В этом случае анализ ошибочной трассы может повлечь за собой модификацию системы и повторное применение алгоритма проверки на модели.

Существуют методы, которые позволяют автоматически уточнять модель проверяемой программы в том случае, когда предложенный контрпример (трасса предположительно ошибочного вычисления) не реализуется в программе.

# Темпоральная логика и модели

Темпоральные логики — полезное средство спецификации параллельных систем, поскольку они позволяют описывать порядок событий во времени без привлечения параметра времени в явном виде.

Темпоральные логики обычно классифицируются в соответствии с тем, является ли структура времени

- линейной или ветвящейся ;
- дискретной или непрерывной .

Смысл всякой формулы темпоральной логики определяется по отношению к размеченному графу переходов. В силу причин исторического характера структуры такого вида называются моделями Крипке .

# Верификация выполнения

Оперативная верификация занимает промежуточное положение между тестированием и верификацией моделей программ.

Проверке подвергается не сама программа, а трассы ее вычислений (сценарии выполнения, наблюдаемое поведение), которые порождаются в результате тестовых экспериментов или реального исполнения программы.

Требование правильного поведения задается в виде формальной спецификации (например, в виде регулярного выражения).

Трасса вычисления представляется в виде формальной модели (например, как слово в некотором алфавите)

При оперативной верификации проверяется соответствие трассы вычисления заданной формальной спецификации (например, принадлежность слова языку, являющемуся значением регулярного выражения).

Оперативная верификация может проводится как в оперативном, так и в автономном режиме.

# Исторические сведения

Модальные логики (Аристотель, 4 век д.н.э.)

# Исторические сведения

Модальные логики (Аристотель, 4 век д.н.э.)

Аксиоматизация модальных логик (C.I. Lewis, 1910)

# Исторические сведения

Модальные логики (Аристотель, 4 век д.н.э.)

Аксиоматизация модальных логик (C.I. Lewis, 1910)

Темпоральные логики (A.N. Prior, 1957)

# Исторические сведения

Модальные логики (Аристотель, 4 век д.н.э.)

Аксиоматизация модальных логик (C.I. Lewis, 1910)

Темпоральные логики (A.N. Prior, 1957)

Семантика возможных миров (S. Kripke, 1959)

# Исторические сведения

Модальные логики (Аристотель, 4 век д.н.э.)

Аксиоматизация модальных логик (C.I. Lewis, 1910)

Темпоральные логики (A.N. Prior, 1957)

Семантика возможных миров (S. Kripke, 1959)

Применение дедуктивных методов для доказательства правильности программ (C.A. Hoare, M. Floyd 1968)

# Исторические сведения

Модальные логики (Аристотель, 4 век д.н.э.)

Аксиоматизация модальных логик (C.I. Lewis, 1910)

Темпоральные логики (A.N. Prior, 1957)

Семантика возможных миров (S. Kripke, 1959)

Применение дедуктивных методов для доказательства правильности программ (C.A. Hoare, M. Floyd 1968)

Динамические логики (V. Pratt, 1976)

# Исторические сведения

Модальные логики (Аристотель, 4 век д.н.э.)

Аксиоматизация модальных логик (C.I. Lewis, 1910)

Темпоральные логики (A.N. Prior, 1957)

Семантика возможных миров (S. Kripke, 1959)

Применение дедуктивных методов для доказательства правильности программ (C.A. Hoare, M. Floyd 1968)

Динамические логики (V. Pratt, 1976)

Применение темпоральных логик
для анализа параллельных программ (A. Pnueli, 1977)

# Исторические сведения

Табличный алгоритм model checking
для темпоральной логики деревьев вычислений CTL
(E. Clarke, E. Emerson, J. Sifakis, 1981)

# Исторические сведения

Табличный алгоритм model checking
для темпоральной логики деревьев вычислений CTL
(E. Clarke, E. Emerson, J. Sifakis, 1981)

Сведение задачи model checking к задаче проверке пустоты автоматов Бюхи
(Systla A.P., Vardi M.Y., Wolper P., 1985)

# Исторические сведения

Табличный алгоритм model checking
для темпоральной логики деревьев вычислений CTL
(E. Clarke, E. Emerson, J. Sifakis, 1981)

Сведение задачи model checking к задаче проверке пустоты автоматов Бюхи
(Systla A.P., Vardi M.Y., Wolper P., 1985)

Символьный алгоритм model checking,
система верификации SMV
(K. McMillan, 1991)

# Исторические сведения

Табличный алгоритм model checking
для темпоральной логики деревьев вычислений CTL
(E. Clarke, E. Emerson, J. Sifakis, 1981)

Сведение задачи model checking к задаче проверке пустоты автоматов Бюхи
(Systla A.P., Vardi M.Y., Wolper P., 1985)

Символьный алгоритм model checking,
система верификации SMV
(K. McMillan, 1991)

Система верификации SPIN
(Holzmann G.J., 1980-90)

# Исторические сведения

Табличный алгоритм model checking
для темпоральной логики деревьев вычислений CTL
(E. Clarke, E. Emerson, J. Sifakis, 1981)

Сведение задачи model checking к задаче проверке пустоты автоматов Бюхи
(Systla A.P., Vardi M.Y., Wolper P., 1985)

Символьный алгоритм model checking,
система верификации SMV
(K. McMillan, 1991)

Система верификации SPIN
(Holzmann G.J., 1980-90)

Самонастраивающийся алгоритм model checking,
(Counter-example guided abstraction refinement, CEGAR)
Системы верификации Blast, Slam
(T. Ball, S. Rajamani, 2000)

# Исторические сведения

Модель вычислительных систем реального времени:
временные автоматы
(R. Alur, D.L. Dill, 1990)

# Исторические сведения

Модель вычислительных систем реального времени: временные автоматы
(R. Alur, D.L. Dill, 1990)

Алгоритм верификации временных автоматов
(R. Alur, D.L. Dill, 1996)

# Исторические сведения

Модель вычислительных систем реального времени:
временные автоматы
(R. Alur, D.L. Dill, 1990)

Алгоритм верификации временных автоматов
(R. Alur, D.L. Dill, 1996)

Система верификации UPPAAL
(K. Larsen, 1995)

# Достижения методов формальной верификации программ

Один из первых примеров успешного практического применения формальных методов верификации программ является обнаружение в 1993 г. ошибок в протоколе когеррентности кеш-памяти FutureBus+, который был принят в качестве IEEE стандарта и определяет архитектуру шины для высокопроизводительных компьютеров.

Проверка правильности протокола проводилась при помощи системы верификации моделей программм SMV.

# Достижения методов формальной верификации программ

Другие примеры успешного применения средств формальной верификации включают

1. Проект компании Immos Ltd (1988 г.) создания микропроцессоров для транспьютеров.
   Язык спецификации Z.
   Язык программирования Occam.
   Метод верификации — доказательство правильности в логике Хоара.
   **Эффект:** обнаружены ошибки в процедурах округления и вычисления остатков в блоке арифметики с плавающей точкой; разработка завершилась на 3 месяца быстрее, чем у альтернативной группы разработчиков, не использовавших методы формальной верификации.
   Оксфордский университет и Immos Ltd были удостоены Queen's Award for Technological Achievement.

# Достижения методов формальной верификации программ

2. Проект компаний GEC Alsthom, MATRA Transport, RATP (оператор парижского общественного транспорта) по компьютеризации системы управления парижским метро (RER) (1988 г.).
   Язык спецификации B.
   Язык программирования Modula-2.
   Метод верификации — доказательство правильности в логике Хоара.
   **Эффект:** использование формальных методов верификации позволило избежать тестирования отдельных модулей системы и ограничиться глобальным тестированием. Впоследствии по той же методике проводилась полная автоматизация одной из линий парижского метро.

# Достижения методов формальной верификации программ

3. Проект National Westminster Bank и компании Platform Seven по созданию электронной платежной системы с применением smart-cards (8-битовый микропроцессор, 256 байт RAM и несколько килобайт ROM) (1992 г.).
   Язык спецификации Z.
   Метод верификации — доказательство (вручную) свойств безопасности.
   **Эффект:** обнаружены и исправлены уязвимости; получено доказательство выполнимости требований стандарта безопасности (200 стр.).

# Достижения методов формальной верификации программ

4. Проект компании Airbus по автоматизации проектирования программного обеспечения и электронного оборудования для авиации и наземного транспорта (1982- наст. время г.).
   Язык спецификации Esterel.
   Язык программирования C, Verilog.
   Метод верификации — model checking, автоматическая генерация кода по спецификациям.
   **Эффект:** сокращение числа ошибок в коде, 70% кода генерируются автоматически, повышение оперативности внесения изменений в проект.

# Достижения методов формальной верификации программ

5. Проект системы автоматического управления подвижным барьером для защиты Роттердама от наводнений (1992-наст. время г.).
   Язык спецификации Z, Promela.
   Язык программирования C.
   Метод верификации — model checking с использованием системы верификации SPIN.
   **Эффект:** обнаружение ошибок в спецификациях и коде.

# Достижения методов формальной верификации программ

6. Полная верификация микроядра linux (seL4) группой программистов из фирмы «NICTA» (Австралия). (2009-наст. время г.).
   Язык спецификации Hascel.
   Язык программирования C, Assembler
   Метод верификации — Isabelle/HOL (интерактивный, полуавтоматический прувер).
   **Эффект:** Доказательство отсутствия ошибок определенного вида в коде микроядра. "We can predict precisely how the kernel will behave in every possible situation."

# Достижения методов формальной верификации программ

В 2003 г. А. Хоар инициировал международный проект создания верифицирующего компилятора как Grand Challenge for Computer Science. В 2006 начато создание и пополнение Verified Software Repository.

Подразделения формальных методов верификации имеются в таких компаниях как Microsoft, Intel, Cisco, IBM, Cadence, Synopsys, Mentor Graphics.

По оценкам экспертов задачами разработки и применения формальных методов верификации в мире занимаются свыше 4000 исследователей.

КОНЕЦ ЛЕКЦИИ 1.