Формальные методы в программной инженерии

Проверка моделей

Наталья Олеговна Гаранина

garanina@iis.nsk.su

Введение

Лекция 1

Надежность программ и систем

Критическое программное обеспечение






?

Корректность

Космос

- **Ariane 5 Flight 501** (4.06.1996)
  - Гибель аппарата на 40й сек после старта
- Программный модуль, *унаследованный* от «Ариан-4».
  - выравнивание инерционной платформы для оценки точности измерений ИСО,
  - После старта в «Ариан-5» не выполнял никаких функций
    - в «Ариан-4» работал 50 секунд
      - отличие траекторий полёта.
  - Вскоре после старта попытался просчитать значение исходя из горизонтальной скорости ракеты.
    - это значение было значительно больше, чем то, которое было у «Ариан-4»
      - возникла ошибка как на активной, так и на запасной ИСО.

Космос

- **Ariane 5 Flight 501** (4.06.1996)
  - Гибель аппарата на 40й сек после старта
- Исходя из требований реализации, *контекст ошибки должен сохраняться в ПЗУ* до отключения процессора
  - ИСО сохранила информацию об исключении
  - Бортовой компьютер на основании этих данных
    - дал команду соплам твердотопливного ускорителя и главному двигателю
      - полное отклонение сопел
      - ракета вышла на запредельную траекторию.
- На новой траектории ракета получила запредельную аэродинамическую нагрузку и начала разрушаться.
- Стартовые двигатели отделились от ракеты и это запустило её самоуничтожение.

Космос

- **Ariane 5 Flight 501** (4.06.1996)
  - Гибель аппарата на 40й сек после старта

Причины:

- Программный модуль был повторно использован в новой среде, где условия функционирования были другие.
  - ПО проверялось для «Ариан-4»
  - для новой ракеты требования не были пересмотрены.
  - ПО не проверялось ни на уровне оборудования, ни на уровне системной интеграции.
- Система выявила и распознала ошибку.
  - Спецификация механизма обработки ошибок не соответствовала ситуации.

Медицина

- **Therac-25** — аппарат для радиационной терапии
  - Погибло 5 человек.
    - Вместо слабого излучения включалось сильное.
    - Ошибка состязания (Race condition): результат зависит от порядка и/или времени выполнения действий.

Медицина

- **Therac-25** — аппарат для радиационной терапии
- В ПО найдены как минимум четыре ошибки, которые могли привести к переоблучению:
- Одна и та же переменная применялась как для анализа введённых чисел, так и для определения положения поворотного круга, отвечающего за тип излучения (электронное, рентген, либо его отсутствие).
  - Поэтому при быстром вводе *Therac-25* мог иметь дело с неправильным положением поворотного круга.
- Настройка положения отклоняющих магнитов, регулирующих мощность излучения, занимает около 8 секунд.
  - Если за это время параметры типа и мощности излучения были изменены, а курсор установлен на финальную позицию, то система не обнаруживала изменений.

Медицина

- **Therac-25** — аппарат для радиационной терапии
- Деление на величину излучения
  - отсутствие проверки деления на ноль
  - соответствующее увеличение величины облучения до максимально возможной.
- Установка однобайтовой булевской переменной в значение true производилось командой `x=x+1`.
  - С вероятностью 1/256 при нажатии кнопки «Set» программа могла пропустить информацию о некорректном положении диска.
- Потенциальные ошибки
  - в многозадачной ОС не было никакой синхронизации.

Медицина

- **Therac-25** — аппарат для радиационной терапии
- Были проигнорированы базовые практики программирования:
  - Написание документации и спецификаций к ПО.
  - Строгая верификация ПО с применением стандартов.
  - Проектирование должно быть достаточно простым и необходимо избегать опасных практик программирования.
    - В случае *Therac-25* в программном коде было много «трюкачества».
  - Способы ведения журнала работы и способов их получения должны быть заданы с самого начала разработки.

Медицина

- **Therac-25** — аппарат для радиационной терапии
- Были проигнорированы базовые практики программирования:
  - ПО должно быть подвергнуто детальному тестированию с использованием формального анализа, как на уровне модулей, так и на уровне всего комплекса в целом.
    - Одного системного тестирования недостаточно.
    - Необходимо проводить регрессионную верификацию на всех этапах.
  - Представление информации пользователю и сам интерфейс должны быть аккуратно спроектированы.
    - В частности, как сообщения об ошибках, так и документация и руководство пользователя.
  - Всё ПО было написано на ассемблере
    - язык высокого уровня.

Финансы

- NYSE, Knight Capital Group (20 лет на рынке, 11% всей торговли американскими акциями) — Авг 2012
  - был загружен обновленный софт, и ПО начало покупать и тут же продавать более ста наименований акций, искусственно завышая их цену при каждой сделке.
  - Ошибка была обнаружена через 45 мин, все акции были проданы по реальной цене.
  - Потери составили 440 млн долларов, примерно 1.5 выручки за 2й кв. – 10 млн долл в минуту.
  - В конце 2012 KCG была поглощена компанией, Getco.

Вооружение

- 25 фев 1991, Война в Заливе – Патриот пропустил атаку Скада
  - Первоначально Патриот был предназначен для мобильного базирования и перехвата ракет на *коротких* дистанциях.
  - Батарея стояла развернутой в ожидании атаки более 100 часов.
  - Софт накапливал ошибки, связанные с вычислениями и конвертированием плавающий-целый.
  - За 2 недели до инцидента ошибка потери точности была замечена в Израиле, 21.02 было направлено сообщение. Обновление ПО достигло Дахрана на день позже.
  - 25.02 накопленная ошибка достигла 0.34 сек, что достаточно для выхода Скада из зоны обнаружения и поражения Патриотом.
  - Результат атаки по базе – 28 погибших и 98 раненых

Компьютерные системы

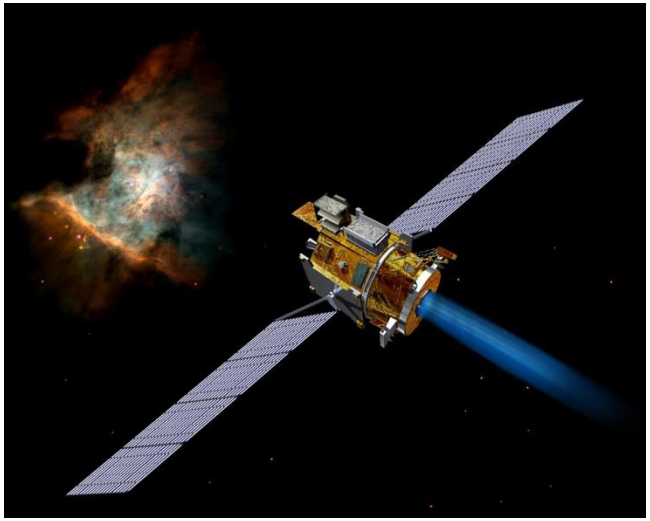
- Конкуренция за ресурсы:
  - В системах с высокой нагрузкой постоянный поток процессов с высоким приоритетом может не дать процессам с более низким приоритетом получить доступ к ресурсам никогда.
  - В MIT в **1973** был остановлен IBM 7094, были обнаружены процессы с низким приоритетом, которые были поставлены в очередь в **1967** (!) и так и не были запущены.

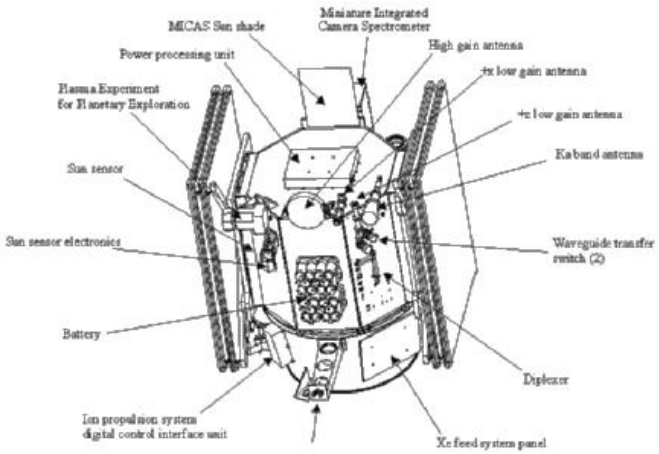
Верификация модулей Deep Space-1 (NASA)

Классические ошибки параллелизма: непредвиденный интерливинг.

- Всегда ли верно $0 \le x \le 200$?

```
proc Inc = while true do if x < 200 then x := x + 1 fi od
proc Dec = while true do if x > 0 then x := x - 1 fi od
proc Reset = while true do if x = 200 then x := 0 fi od
```





Верификация модулей Deep Space-1 (NASA)

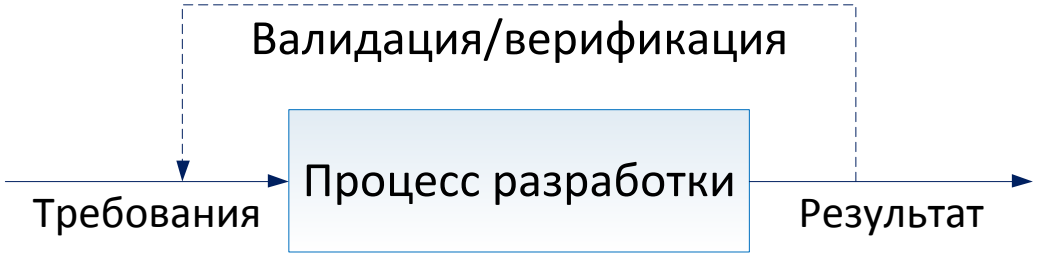
```
proc Inc = while true do if x < 200 then x := x + 1 fi od
proc Dec = while true do if x > 0 then x := x – 1 fi od
proc Reset = while true do if x = 200 then x := 0 fi od
```

- Всегда ли верно 0 ≤ x ≤ 200?

- x = 200
- `Dec`: 200 > 0?; `Reset`: 200 = 200?, `x:=0`; `Dec`: `x:= x-1` = -1.

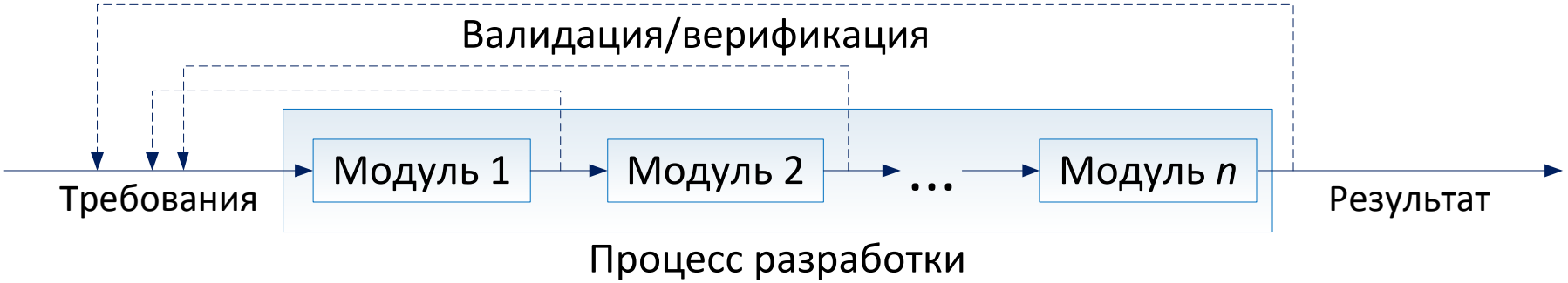
Валидация/верификация

- Надежность программных систем
  - Анализ требований заказчика.
  - Фазы разработки.
    - Прототип.
  - Валидация:
    - разработанный прототип удовлетворяет требованиям заказчика.
  - Верификация:
    - разработанный прототип удовлетворяет требованиям разработчика.



Валидация/верификация

- Валидация/верификация на лету
  - затратность проверки на ошибки только в конце разработки;
  - значительно уменьшает стоимость разработки.



Валидация/верификация

- Практика проверки соответствия результата требованиям
  - *экспертный анализ*
    - 80% всех проектов
    - полностью ручную деятельность, в которой прототип или его часть исследуется командой специалистов, которые не были вовлечены в процесс разработки.
  - *тестирование*
    - тесты генерируются вручную, и инструментальной поддержке уделяется мало внимания;
    - программные модули уже скомпонованы и исследуется вся система.

Валидация/верификация

- Больше времени и усилий уходит на валидацию/верификацию, чем на разработку.
- В настоящее время методы валидации/верификации узкоспециализированные и основаны на спецификациях, выраженных на естественном языке.
- Использование современных методов и инструментов.
  - Подходы к верификации, базирующиеся на *формальных методах*.
    - Проектирование систем определено в терминах точных и недвусмысленных спецификаций, которые предоставляют базу для систематического анализа.

Методы верификации

- Статический анализ
- Симуляция
- Тестирование
- Формальная верификация
  - Дедуктивная верификация
  - Проверка моделей (верификация на моделях).

Симуляция

- Быстрая, первоначальная оценка качества результата
- Используется модель, описывающая возможное поведение системы.
  - Симулятор – программный инструмент
    - определяет поведение системы по отношению к некоторым сценариям.
    - реакция системы на сценарии, которые предоставлены
      - пользователем,
      - инструментом-генератором (случайных) сценариев.
- Симулировать все возможные сценарии непрактично (и часто невозможно).

Статический анализ

- Проверка формализованных правил корректного построения ПО;
- Поиск часто встречающихся дефектов по некоторым шаблонам;
- Хорошо автоматизируется;
- Способен обнаруживать только ограниченный набор типов ошибок;
- Проблема точности:
  - строгие методы анализа не допускают пропуска ошибок, но приводят к большому количеству ложных сообщений об ошибках
  - более лояльные методы выдают точный набор сообщений об ошибках, но могут пропустить ошибку.

Тестирование

- Используется реализация системы (программа/устройство).
- Тесты -- определенные значения входных данных
  - анализ реакции системы
- Рассмотрение только небольшого подмножества возможных примеров поведения системы
  - Неполнота
    - может показать только наличие ошибок, но не отсутствие.
- Дополнение формальной верификации, когда
  - когда корректную модель системы сложно построить;
  - когда части системы не могут быть формально промоделированы (например, физические устройства);
  - когда модель проприетарна.

Тестирование

- Анализ и оценка свойств программной системы делаются по результатам ее реальной работы (или моделей и прототипов).
- Необходимо иметь работающую систему, ее компоненты, прототипы
  - нельзя использовать на первых стадиях разработки
- можно контролировать характеристики работы системы в ее реальном окружении, которые иногда невозможно проанализировать с помощью других подходов.

Тестирование

- Дополнительная подготовка
  - разработка тестовой системы;
  - системы мониторинга, позволяющей контролировать характеристики поведения системы.
  - создание тестов,
    - готовятся заново для каждой проверяемой системы;
    - позволяет обнаружить множество дефектов в описании требований и проектных документах;
    - довольно трудоемкая задача, если необходимо получить адекватную оценку качества сложной системы;
- Очень широко используется на практике
  - не слишком надежные, но достаточно дешевые техники,
    - (нестрогое) вероятностное тестирование
      - тестовые данные генерируются случайным образом;
    - тестирование на основе простейших сценариев использования.

Формальная верификация

- Формальные модели требований, поведения ПО и его окружения.
- Анализ формальных моделей:
  - *дедуктивный анализ* (theorem proving),
  - *проверка моделей* (model checking),
  - *абстрактная интерпретация* (abstract interpretation)…
- Значительные усилия на построение формальных моделей.
  - специалисты по формальным методам (их мало и они дороги);
  - построение формальных моделей сложно автоматизировать;
  - анализ свойств в значительной мере может быть автоматизирован
    - требуется специфический набор навыков и знаний в разделах математической логики и алгебры.

Формальная верификация

- Активно используется в ряде областей, где последствия ошибки могут оказаться чрезвычайно дорогими;
- Способна обнаруживать сложные ошибки, практически не выявляемые с помощью экспертиз или тестирования;
- Формализация требований и проектных решений возможна только при их глубоком понимании
  - совместная работа специалистов по формальным методам и экспертов в предметной области;
- Инструменты, эффективно решающие ограниченные задачи верификации ПО из определенного класса для промышленных проектов, требующие для применения минимальных специальных навыков и знаний.

Формальная верификация

- Строгое *доказательство* того, что система работает корректно.
- Построение формальной (математической) модели исследуемой системы, которая отражает возможное поведение системы.
- Требования корректности записываются в виде формальной *спецификации требований*, которая отражает желаемое поведение системы.
- Точная и полная проверка согласованности возможного и желаемого поведения системы.
  - Доказательство корректности относительно формального представления.

Формальная верификация

Требуются:

- Модель системы
  - множество «состояний», которые хранят информацию о значениях переменных, программных счетчиках, операторах…
  - отношение переходов, которое описывает, как система переходит из одного «состояния» в другое.
- Метод спецификации для формального выражения требований.
- Правила доказательства, позволяющих определить, удовлетворяет ли модель сформулированным требованиям.

Формальная верификация

{ Предусловие } Программа { Постусловие }

$\{\varphi\}\ S\ \{\psi\}$

Аксиома для **skip**	$\{\ \varphi\ \}$ **skip** $\{\ \varphi\ \}$
Аксиома для присваивания	$\{\ \varphi[x=k]\ \}\ x := k\ \{\ \varphi\ \}$
Последовательная композиция	$\dfrac{\{\ \varphi\ \}\ S_1\ \{\ \chi\ \},\ \{\ \chi\ \}\ S_2\ \{\ \psi\ \}}{\{\ \varphi\ \}\ S_1\ ;\ S_2\ \{\ \psi\ \}}$
Альтернатива	$\dfrac{\{\ \varphi \wedge B\ \}\ S_1\ \{\ \psi\ \},\ \{\ \varphi \wedge \neg B\ \}\ S_2\ \{\ \psi\ \}}{\{\ \varphi\ \}$ **if** B **then** S_1 **else** $S_2\ \{\ \psi\ \}$
Итерация	$\dfrac{\{\ \varphi \wedge B\ \}\ S\ \{\ \varphi\ \}}{\{\ \varphi\ \}$ **while** B **do** $S\ \{\ \varphi \wedge \neg B\ \}$
Следствие	$\dfrac{\varphi \rightarrow \varphi`\ \{\ \varphi`\ \}\ S\ \{\ \psi`\ \}\ \psi` \rightarrow \psi}{\{\ \varphi\ \}\ S\ \{\ \psi\ \}}$
Параллелизм	$\dfrac{\{\ \varphi\ \}\ S_1\ \{\ \psi\ \},\ \{\ \varphi`\ \}\ S_2\ \{\ \psi'\ \}}{\{\ \varphi \wedge \varphi`\ \}\ S_1 \parallel S_2\ \{\ \psi \wedge \psi`\ \}}$

- Формальная система для частичной корректности (последовательных) программ

Формальная верификация

Что не так?

- Параллельные процессы могут взаимодействовать в любой момент их исполнения, а не только в начале вычисления.
  - Для анализ взаимодействия недостаточно знать свойства стартовых и финальных состояний.
  - Формулировка суждений о том, что происходит *во время* вычисления, свойства вычислений.
- Для параллельных систем вычисление обычно не завершается
  - корректность зависит от поведения системы во времени, а не только от результата вычисления
  - глобальные свойства параллельных программ часто не могут быть сформулированы в терминах отношений между входами и выходами.

Формальная верификация

- Обобщение классического метода для параллельных программ.
- Сложные правила вывода
  - взаимодействие между компонентами путем использования разделяемых переменных или путем (синхронного или асинхронного) обмена сообщениями.
- Очень большие и сложные доказательства для реальных систем.
  - Взаимодействие с пользователем при доказательстве и управление от него (в частности, в форме поиска подходящих инвариантов).
    - доказательство громоздко, неустойчиво к ошибкам, плохо читаемо.

Автоматическое доказательство теорем

- *Эффективно используется в областях, где доступны математические абстракции задач.*
- Спецификация и модель системы -- формулы некоторой логики (обычно вариант логики предикатов первого порядка).
- Верификация:
  - любое поведение реализации, удовлетворяющее ψ, является возможным поведением спецификации системы, и поэтому удовлетворяет φ.
  - доказательство формулы $\psi \rightarrow \varphi$.
    - программы автоматического доказательства теорем.
- *Проверка доказательств*
  - Пользователь проверить своё доказательство теоремы в программе для проверки доказательств.

Автоматическое доказательство теорем

- Алгоритмические компоненты программ доказательства теорем
  - методы применения правил вывода и получения следствий.
    - дедукция
    - резолюция
    - унификация и т.п.
- Работает с бесконечными множествами состояний
- Проверяет справедливость свойств с произвольными значениями параметров.

Автоматическое доказательство теорем

- Ограниченность автоматического применения методов для поиска доказательства заданной теоремы, если доказательство существует.
  - Отсутствие оптимальной стратегии, которая говорит, как искать доказательство.
  - Стратегии, которые используются программами доказательства теорем, базируются на алгоритмах обхода в ширину и в глубину.
- Проблема доказательства теорем экспоненциально сложна
  - Для интерактивных программ доказательства теорем эта сложность значительно уменьшается.
- Процесс верификации с использованием программ автоматического доказательства теорем является медленным и трудоемким, требует довольно высокой квалификации пользователей.

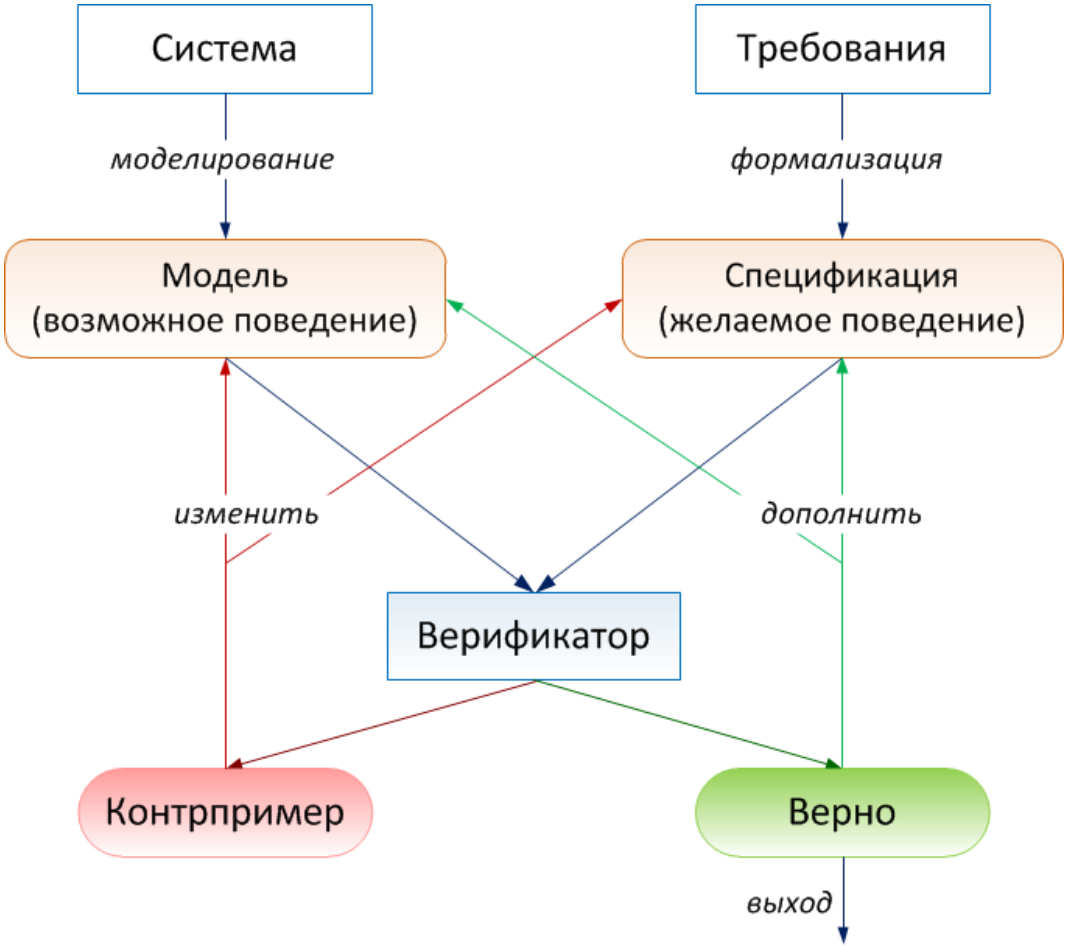
Темпоральная логика

- Корректность реактивных систем
  - поведение систем в течение времени.
- Коммуникационный протокол

- Свойство «если процесс *P* посылает сообщение, он не пошлет следующее сообщение до тех пор, пока не примет подтверждения».
  - **G** [$snd_P(m) \rightarrow \neg snd_P(nxt(m))$ **U** $rcv_P(ack)$].
- Если сообщение *m* послано процессом *P*, то этот процесс не передаст следующее сообщение *nxt*(*m*), пока не получит подтверждение.
- **U** (*until*) и **G** (*globally*) -- темпоральные операторы, относятся к последовательностям состояний.

Проверка моделей (model checking)

- Схема проверки моделей



Проверка моделей

- Исчерпывающий обзор множества всех состояний модели системы:
  - для каждого состояния системы проверяется, удовлетворяет ли оно требуемому свойству
  - завершается в силу конечности модели
- *Анализ достижимости*.
  - Может ли система попасть в состояние, в котором вычисление не может продолжаться (*блокировка*).
    - Определить все достижимые состояния и
    - Проверить есть ли среди них состояние блокировки.
  - Доказательство инвариантных свойств, которые выполняются в течение всего вычисления.
    - отсутствие блокировок, наличие прогресса и т.п.
  - Недостаточно, например, для коммуникационных протоколов,
    - Свойство: если сообщение послано, то оно обязательно когда-нибудь будет получено.

Подходы в проверке моделей

Различие в описании желаемого поведения.

1. *Логический или разнородный подход*.
2. *Поведенческий или однородный подход*.

- *Логический или разнородный подход*.
  - Спецификация свойств (требований) в подходящей логике (темпоральной или модальной).
  - Модель системы
    - Конечная модель Крипке (автомат и т.п.)
      - состояния -- значения переменных и управляющие позиции,
      - переходы – смена состояний системы.
  - Система считается корректной по отношению к требованиям, если заданное множество начальных состояний выполняет эти требования.

Подходы в проверке моделей

- *Поведенческий или однородный подход*.
  - Желаемое и возможное поведение задаются в одной и той же нотации (конечный автомат, сеть Петри и т.п.)
  - Критерий корректности -- отношения эквивалентности (или предпорядки).
    - Симуляция
    - Бисимуляция
    - Пошаговая
    - Вход-выход
    - Включение языков и т.п..
  - Система считается корректной, если желаемое и возможное поведение эквивалентны (или упорядочены) по отношению к исследуемой эквивалентности (или предпорядку).

Достоинства проверки моделей

- Это *общий подход* с приложениями к верификации аппаратуры, программ, коммуникационных протоколов, мультиагентных систем, встраиваемых систем и т. д.
- Полнота проверки требований
- Подход поддерживает *частичную* верификацию.
  - проверка наиболее важных свойств, игнорируя проверку менее важных, но вычислительно дорогих требований.
- Встраивание проверки моделей в процесс проектирования не требует больше времени, чем симуляция и тестирование.
- Программы для проверки моделей не требует высокой степени взаимодействия с пользователем.
- *Надежный математический фундамент*: моделирование, семантика, логика и теория автоматов, структуры данных, алгоритмы на графах и т.п.

Ограничения проверки моделей

- Плохо применима к приложениям обработки данных
  - бесконечные пространства состояний.
- Верификация *модели* системы, а не самой системы.
  - Нет гарантии, что финальная реализация обладает теми же свойствами.
- Построение «правильной» модели системы и «правильная» формулировка требований требует соответствующей квалификации.
- Недостаточная верификация верификаторов.
- Ограниченность обобщений результатов проверки
  - Верифицированный протокол для трех процессов необязательно корректен для четырёх.
  - Существуют методы обобщения.

Проверка моделей

- Абсолютно гарантированная корректность систем невозможна.
  - Проверяется ровно то, что сформулировано.
- Но проверка моделей существенно повышает уровень доверия к системам.
- Основная проблема проверки моделей -- *комбинаторный взрыв.*
  - параллельные и распределенные системы с большим числом состояний индивидуальных компонент.

Проверка моделей vs доказательство теорем

- Процесс полностью автоматический и быстрый **1 : 0**
- Программы имеют понятный интерфейс, язык и легко используются **1 : 0**
- Применение к реактивным системам **1 : 0**
- Применимо для приложений обработки данных **0 : 1**
- Максимальный уровень точности и надежности доказательства в случае успеха. **0 : 1**
- Генерация контрпримеров для анализа и отладки. **1 : 0**
- Применение произвольных значений параметров системы. **0 : 1**

- Интеграция методов для получения эффекта от объединения достоинств подходов.

Методы верификации

Синтетические методы

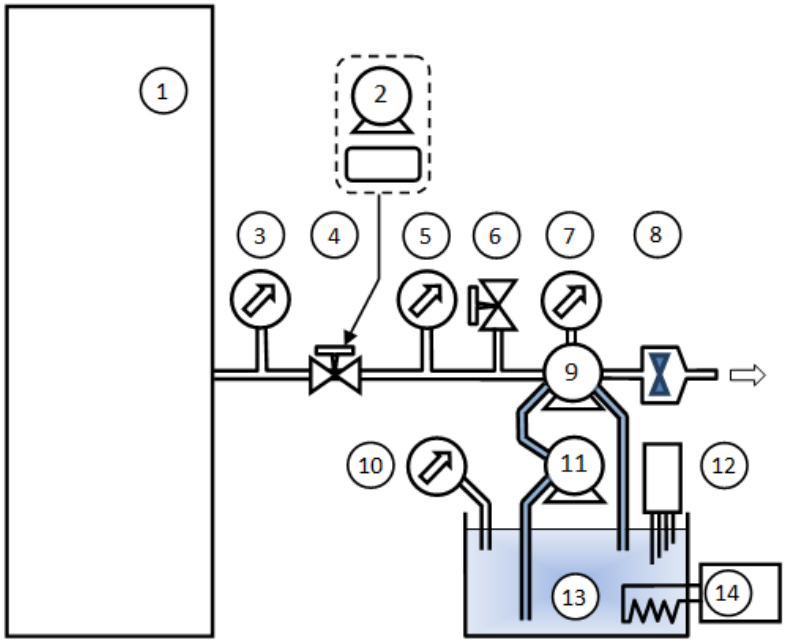
- Комбинирование нескольких перечисленных выше видов верификации.
  - динамические методы, использующие элементы формальных
    - *тестирование на основе моделей* (model-based testing, model driven testing)
    - *мониторинг формальных свойств* (runtime verification, passive testing)
  - инструменты построения тестов используют
    - формализацию некоторых свойств ПО,
    - статический анализ кода.
- Общая идея — попытаться сочетать преимущества основных подходов к верификации, смягчая их недостатки.

Реактивные системы

- Непрерывное взаимодействием с окружением
  - принимают значения входов из своего окружения и реагируют на них.
- Операционные системы
- Коммуникационные протоколы
  - криптографические, блокчейн, связи, и т.д.
- Системы управления
  - самолетами, автомобилями, заводами и пароходами, технологическими процессами и т.п.

Реактивные системы

- Подсистема вакуумирования БСВТ:



1 – труба телескопа,

2 – пневмоустройство,

3 – датчик давления в трубе телескопа,

4 – клапан подключения вак.насоса к трубе телескопа,

5 – датчик давления в патрубке вак.насоса,

6 – отсечной клапан соединения вак.насоса с атмосферой (сапун),

7 – датчик температуры воды в рубашке охлаждения вак.насоса,

8 – вентилятор, 9 – вакуумный насос, 10 – датчик температуры воды в системе климат-контроля, 11 – насос охлаждения, 12 – датчики уровня воды в системе климат-контроля, 13 – система климат-контроля, 14 – нагреватель воды в системе климат-контроля.

Реактивные системы и верификация

- Методология построения корректно работающей реактивной системы

1. Анализа и спецификация требований.
2. Концептуальное проектирование:
   - абстрактная спецификация проекта:
     - проверка на непротиворечивость и соответствие требованиям:
       - Статический анализ, симуляция, формальная верификация
         - модель абстрактной спецификации проекта может быть исчерпывающим образом проверена.
3. Построение системы, реализующей абстрактную спецификацию.
   - Тестирование.

План лекций

1. Постановка задачи. Надежность программ и систем. Логический язык спецификаций. Понятие корректности программ.
2. Проверка моделей. Моделирование систем. Проверка моделей для временных логик.
    1. Структура Крипке. Логическое представление систем.
    2. Примеры параллельных и взаимодействующих систем.
    3. Спецификация программных систем с помощью временных логик CTL и LTL.
    4. Алгоритмы проверки на модели формул временных логик CTL и LTL.
3. Основная проблема проверки моделей: методы решения.
    1. Символьная проверка моделей.
    2. Использование особенностей моделей: редукция, композиция, абстракция, симметрия.