Верификация программ на моделях

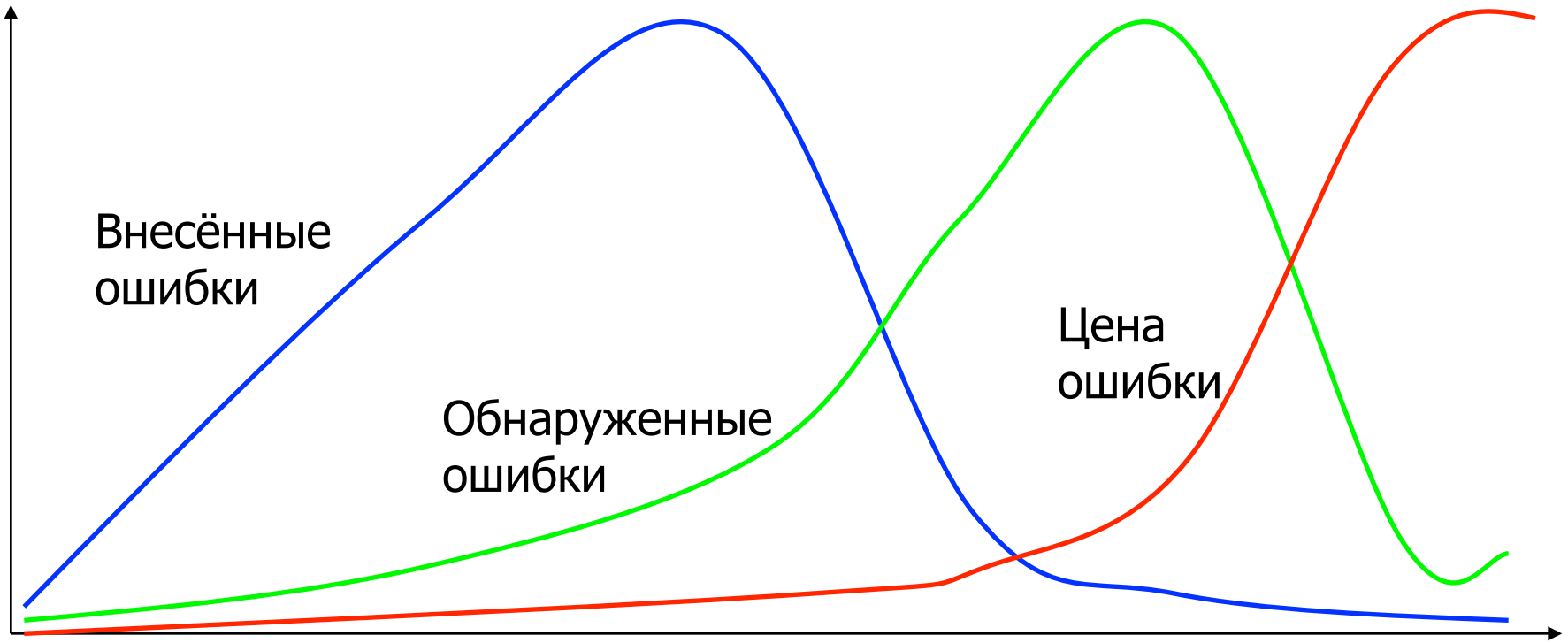
Константин Савенков (лектор)

План лекции

- Правильность программ
- Актуальность верификации
- Формальные методы проверки правильности
- Обзор курса
- Практикум

Разработка программы

Анализ	Проектиро-вание	Реали-зация	Тестирование		Эксплуа-тация
			Unit	System	



Правильность программ

- Нет требований – нет правильности
- Ошибка – несоответствие требованиям
- Ошибки:
  - в формулировке требований,

    building the wrong system,
  - в соблюдении требований,

    building the system wrong.

Правильность программ

- Валидация – исследование и обоснование того, что спецификация ПО и само ПО через реализованную в нём функциональность удовлетворяет требованиям пользователей,
- Верификация – исследование и обоснование того, что программа соответствует своей спецификации.

найти ошибки или доказать, что их нет

NB: ошибки формализации требований

План лекции

- Правильность программ
- **Актуальность верификации**
- Формальные методы проверки правильности
- Обзор курса
- Практикум

Цена ошибки

- В ряде приложений ошибки не критичны:
  - сводятся к лёгким моральным травмам,
  - возможность обнаружения сбоя и восстановления,
  - возможность быстрого исправления ошибки.

Цена ошибки

- Системы с повышенными требованиями к надёжности (Safety-critical)
- Ошибки приводят к:
  - Гибели или травмам людей,
  - Крупным финансовым потерям
  - Ущербу окружающей среде
  - Итд.

Цена ошибки: Ariane-5

- Июнь 1996 года, взрыв ракеты спустя 40 сек. после старта,
- Ущерб – $500млн (разработка – $7 млрд.),
- Причина – 64bit float -> 16bit int.

Цена ошибки: Patriot

- Февраль 1991 года, Patriot промахнулся мимо ракеты Scud,
- Ущерб – 28 убитых, >100 раненых,
- Причина – ошибка округления из-за 24bit fixed, Scud успел пролететь 500м.

Цена ошибки: Sleipner A

- август 1991 года, Северное море, платформа Sleipner A затонула после разрушения основания,
- Ущерб – $700 млн, землетрясение силой 3 балла,
- Причина – ошибка округления при моделировании платформы.

Кое-что посвежее (2010)

- Ошибки в ПО автомобилей: Toyota Prius (нарушен директивный интервал включения ABS), Chrysler (можно вытащить ключ, не переключившись в режим парковки)
- Ошибки в программе обработки заявлений доноров органов в UK (у 25 человек взяли не те органы)
- Ошибка в электронной системе налоговой службы США (не обслуживались 64-летние люди)
- Ошибка в антивирусе McAfee (системный файл Windows распознан как вредоносный и удалён, бесконечный ребут)
- Отключение Skype (вышла из строя одна из версий клиента, что привело к сбою всей P2P сети)
- Ошибка в апдейте NYSE Euronext (S&P упал на 10%)
- 88 критических ошибок в Android Froyo (доступ к личным данным)

McAfee
Proven Security™

facebook

at&t

Baxter

Повышенные требования к надежности

- Не только спутники, самолёты и АЭС!
- Системы, в которых
  - затруднено исправление ошибок (потребительская электроника)
  - велик масштаб использования (та же электроника, важные веб-сервисы)
  - высокая степень доверия человека (augmented reality)
- Критически-важных систем становится всё больше, они становятся более сложными и интероперабельными.

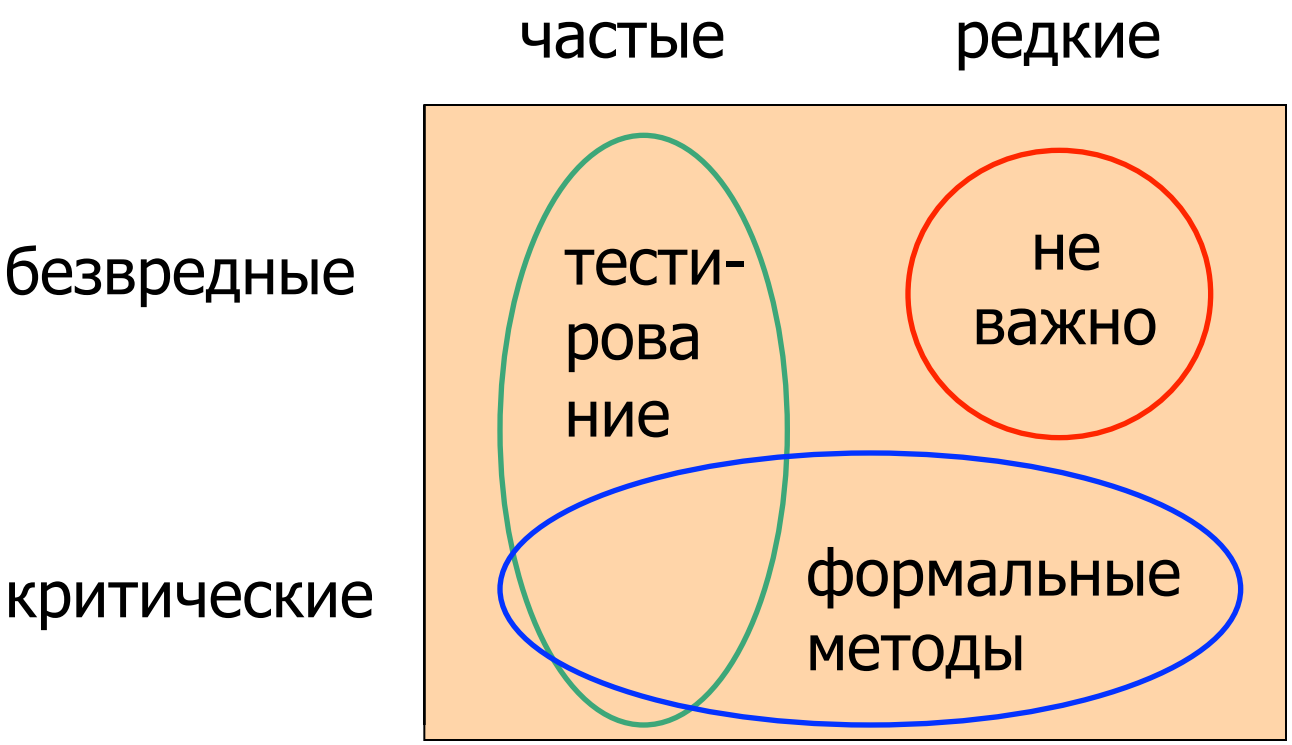
План лекции

- Правильность программ
- Актуальность верификации
- **Формальные методы проверки правильности**
- Обзор курса
- Практикум

Выборочное тестирование

«Тестирование может показать присутствие ошибок, но не может показать их отсутствия» (с) Дейкстра.

Выявление ошибок



Reminder

ВЕРИФИКАЦИЯ ПРОГРАММЫ
В ОБЩЕМ СЛУЧАЕ
АЛГОРИТМИЧЕСКИ НЕРАЗРЕШИМА

См., например, задачу останова программы, теорему Райса, etc

Формальные методы

«Использование математического аппарата, реализованного в языках, методах и средствах спецификации и верификации программ»

- Методы формальной спецификации
- Методы формальной верификации:
  - Доказательство теорем
  - Верификация на моделях
  - Кое-что ещё

Методы верификации

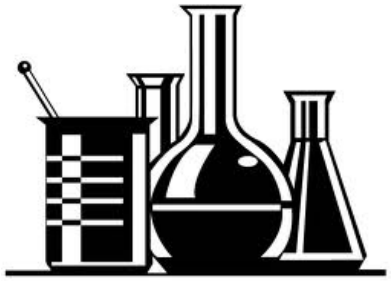
- «Полное» тестирование
- Имитационное моделирование
- Доказательство теорем
- Статический анализ
- Верификация на моделях
- Динамическая верификация

Тестирование

- Обоснование полноты тестового покрытия,
- Метод «чёрного ящика» (ЧЯ) -- полное покрытие входных данных,
- Метод «прозрачного ящика» (ПЯ) -- полное покрытие кода программы.

Тестирование: плюсы

- Проверяется та программа, которая будет использоваться,
- Не требуется (знания) дополнительных инструментальных средств,
- Удобная локализация ошибки.



Тестирование: минусы

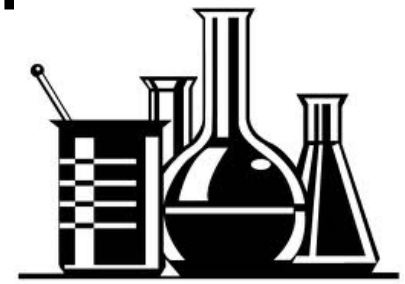
- Не всегда есть условия для тестирования системы,
- Проблема с воспроизводимостью тестов.

частичное решение –

имитационное моделирование

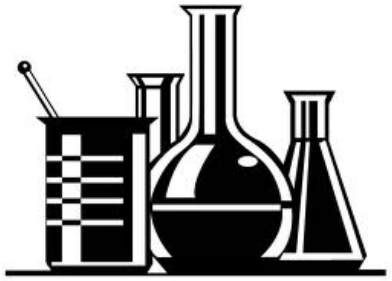
Тестирование: подводные камни

- **Полнота тестового покрытия:**
  - **ЧЯ**: для последовательных программ сложно перебрать все входные данные,
  - **ЧЯ**: для параллельных – очень сложно,
  - **ЧЯ**: для динамических структур данных, взаимодействия с окружением – **невозможно**.
  - **ПЯ**: большой размер покрытия,
  - **ПЯ**: часто **невозможно** построить 100% покрытие,
  - **ПЯ**: полное покрытие **не гарантирует** отсутствия ошибок.



Полное покрытие для черного ящика

- Поиск выигрышной стратегии в шашках:
  - 10^{14} тестов,
  - 18 лет,
  - постоянно работало от 50 до 200 десктопов.



Полное покрытие для прозрачного ящика

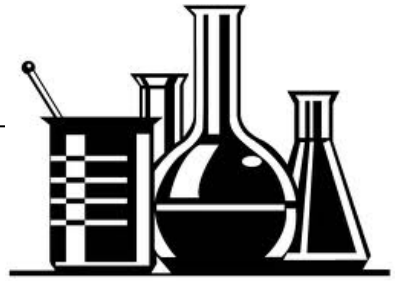
```
if (B1) {
   S1;
}
```

- Два теста

```
if (B1) {
   S1;
}
if (B2) {
   S2;
}
```

- Четыре теста

...exp...



Полное покрытие для прозрачного ящика

```
int x = 1;
if (x == 1) {
  std::cout << “Okay” << std::endl;
} else {
  std::cout << “Error” << std::endl;
}
```

-- полное покрытие кода невозможно

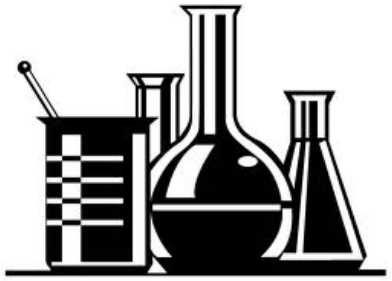
Полное тестовое покрытие – не панацея

```
int strlen(const char* p) {
    int len = 0;
    do {
        ++len;
    } while (*p++);
    return len;
}
```

«a», «bbb» -- полное покрытие кода, ошибка не найдена

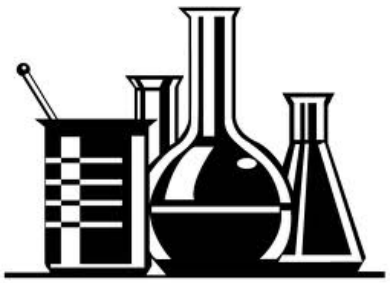
Полнота покрытия: итоги

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



Кое-что о тестировании

- Как правило, разработка критически важных систем регулируется каким-либо стандартом; Например, RTCA/DO-178B (авионика);
- Стандарты разрабатывались давно, поэтому основной способ верификации там – тестирование;
- Пример обоснования полноты тестового покрытия для критических систем – **MC/DC** (http://techreports.larc.nasa.gov/ltrs/PDF/2001/tm/NASA-2001-tm210876.pdf);
- В стандарт RTCA/DO-178C (2011г) включены model-driven development и верификация



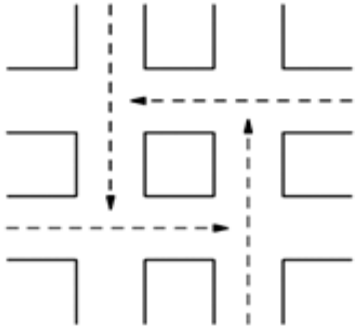
Дополнительные трудности:
Реактивные программы

- Традиционные программы:
  - Завершаются,
  - Описание «вход/выход»,
  - **Число состояний зависит от входных данных и переменных**;
- Реактивные программы:
  - Работают в бесконечном цикле,
  - Взаимодействуют с окружением,
  - Описание «стимул/реакция»,
  - (Не обязательно параллельные),
  - **Дополнительный источник сложности.**

Дополнительные трудности:

Параллельные программы

- Большое количество возможных вычислений,
- Неочевидные ошибки,
- Пример – системы с разделением ресурсов.
- Исключительная ситуация:

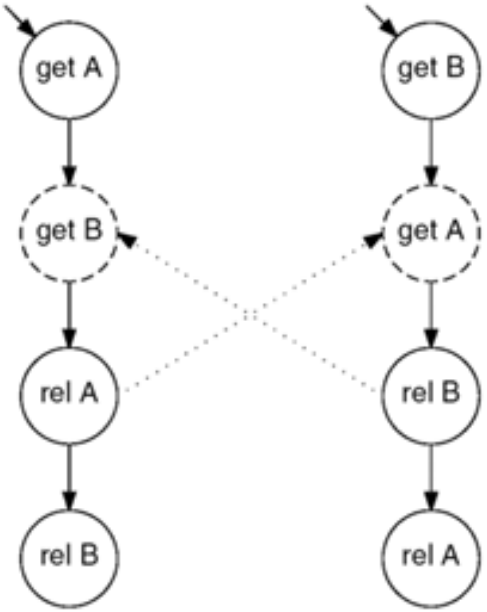


- Правила, реализованные в программах, должны быть универсальны

Системы с разделением ресурсов

Примеры:

- Дорожный траффик,
- Телефонные сети,
- Операционные системы,
- …



Дополнительные трудности:

Параллельные системы

- Новый источник ошибок – совместная работа проверенных компонентов,
- Невоспроизводимость тестов,
- Ограниченные возможности по наблюдению.

Методы верификации

- «Полное» тестирование
- Имитационное моделирование
- **Доказательство теорем**
- Статический анализ
- Верификация на моделях
- Динамическая верификация

Доказательство теорем

- Система и свойства – формулы
- Набор аксиом и правил вывода
- Строится доказательство свойства-теоремы
- *Качественный* анализ системы

Доказательство теорем

- Система:

$$A = c \bullet a \bullet B$$
$$B = b \bullet A$$

- Свойство: $a \bullet c ?$
- Правила вывода:

$$\frac{S_1 = a_1 \bullet S_2 \vee S_2 = a_2 \bullet S_3}{S_1 = a_1 \bullet a_2 \bullet S_3}, \quad \frac{s_1 \bullet s_2 \vee s_2 \bullet s_3}{s_1 \bullet s_3}$$

Доказательство теорем

- Достоинства:
  - работа с бесконечными пр-вами 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 (теорема неполноты Гёделя).

Доказательство теорем

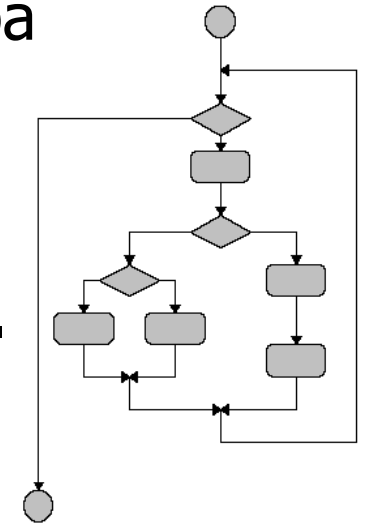
- Примеры инструментальных средств:
  - Isabelle/HOL, Coq, PVS, Vampire, SPASS, ACL2, Simplify, Microsoft Z
- Что почитать:
  - ATP было в курсе «Математическая логика» В.А.Захарова
  - Xavier Leroy, Mechanized semantics with applications to program proof and compiler verification, INRIA, France, 2009 (http://pauillac.inria.fr/~xleroy/courses/Marktoberdorf-2009/notes.pdf)
  - О PVS расскажут на 5-м курсе

Методы верификации

- «Полное» тестирование
- Имитационное моделирование
- Доказательство теорем
- **Статический анализ**
- Верификация на моделях
- Динамическая верификация

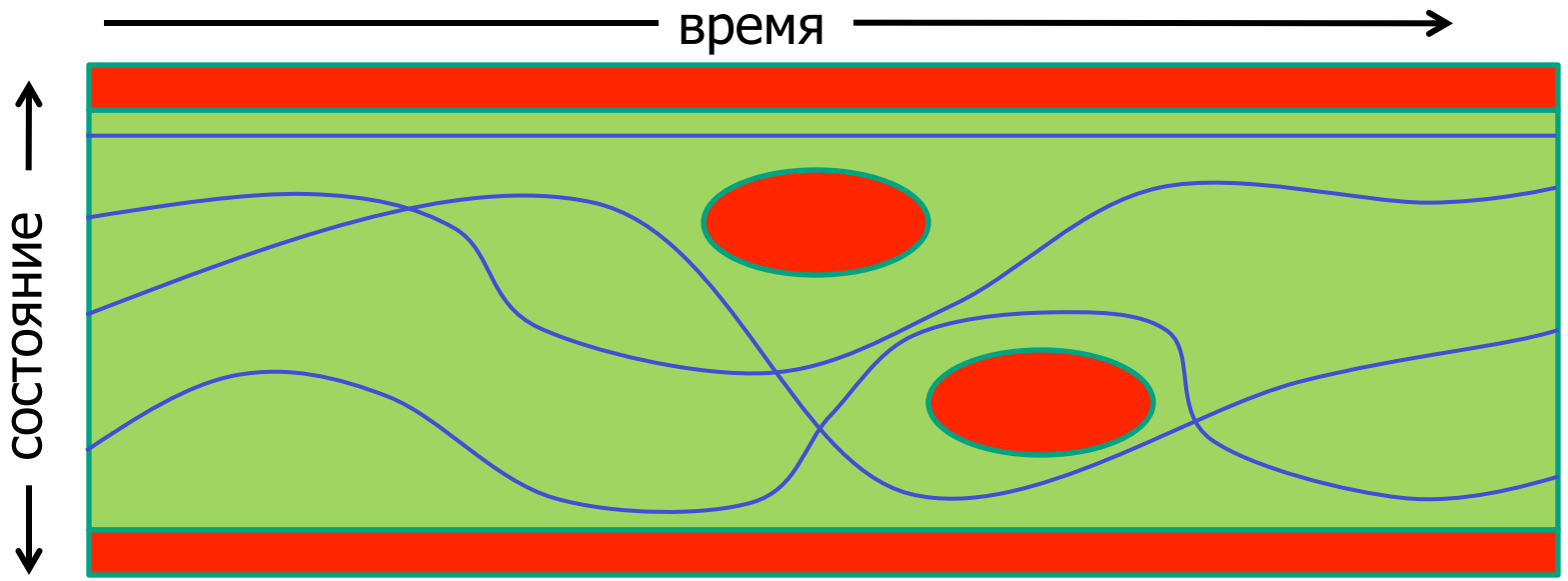
Статический анализ

- Более грубый и прагматичный подход,
- Анализ исходного текста программы без её выполнения,
- В общем случае задача неразрешима (сводится к анализу достижимости оператора программы),
- Поиск компромисса между потребностями и возможностями.



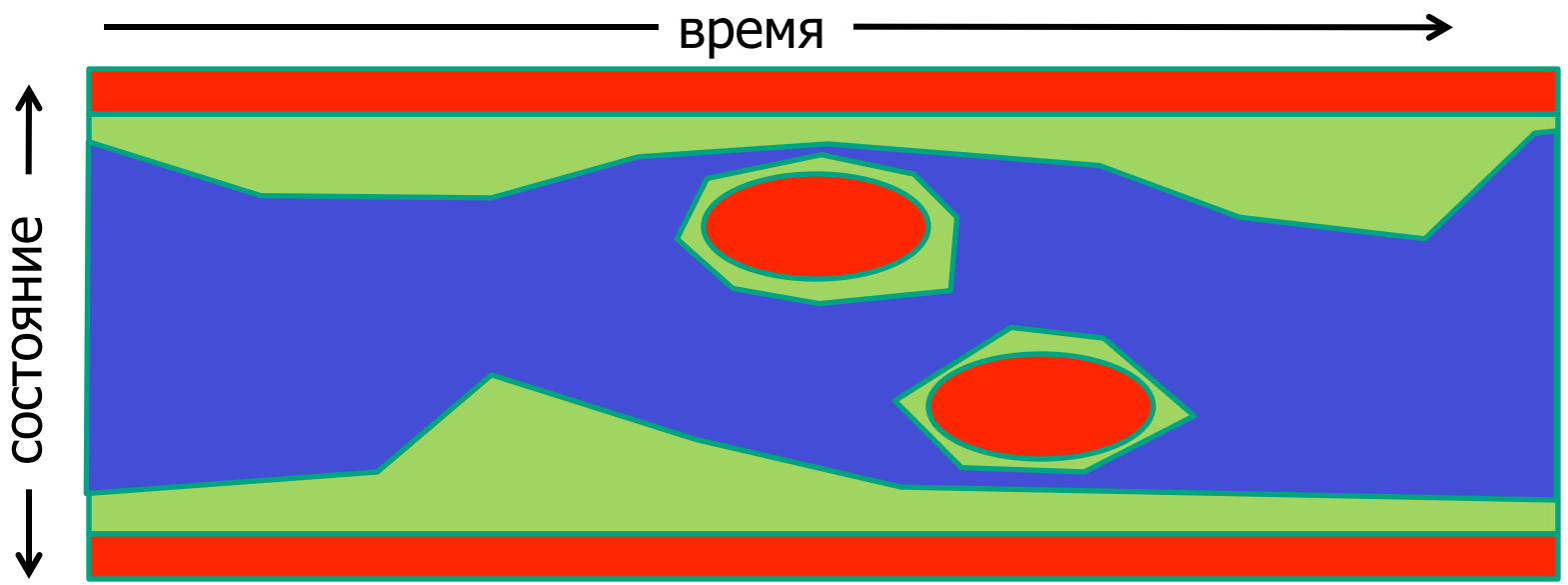
Статический анализ

- Абстрактная интерпретация: построение абстрактной семантики языка программирования и интерпретация текста программы в соответствии с этой семантикой,
- В случае тестирования: проверяем, что конкретные вычисления программы не приводят её в ошибочные состояния



Статический анализ

- **Абстрактная интерпретация:** построение абстрактной семантики языка программирования и интерпретация текста программы в соответствии с этой семантикой,
- Статический анализ: аппроксимируем «сверху» множество вычислений программы,
- Возможны ложные сообщения о нарушении свойств!



Статический анализ: пример

Проверка инициализированности переменной:

```
int min(int* arr, int n) {
  int m;
  if (n > 0) {
    m = arr[0];
  }
  int i = 0;
  while (i < n) {
    if (m > arr[i]) {
      m = arr[i];
    }
    i++;
  }
  return m;
}
```

dom(m) = Int + { ω }

NI = { ω }

I = Int

v: Expr -> {NI, I}

Статический анализ: пример

Проверка инициализированности переменной:

```
int min(int* arr, int n) {
  int m;          <v = { NI }>
  if (n > 0)      <v = { NI }> {
    m = arr[0]; <v = { I }>
  }  <v = { NI, I }> (!)
  int i = 0; <v = { NI, I }>
  while (i < n) <v = { NI, I }> {
    if (m > arr[i]) <v = { NI, I }> {
      m = arr[i]; <v = { I }>
    }
    i++; <v = { NI, I }>
  }
  return m; <v = { NI, I }>
}
```

dom(m) = Int + { ω }

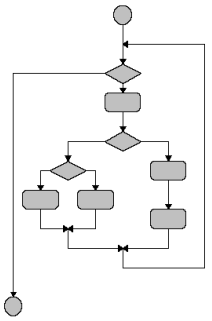
NI = { ω }

I = Int

v: Expr -> {NI, I}

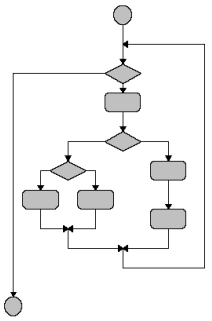
Статический анализ

- Достоинства
  - Высокая скорость работы,
  - Если ответ дан, ему можно верить.
- Недостатки
  - Узкая область применения (оптимизация в компиляторах, анализ похожести кода, анализ безопасности итп.),
  - Ручная настройка при изменении проверяемых свойств



Статический анализ

- Примеры средств:
  - ASTREE (http://www.astree.ens.fr),
- Что почитать:
  - Reps, T., Program analysis via graph reachability. *Information and Software Technology 40*, 11-12 (November/December 1998), pp. 701-726.
  - P. Cousot & R. Cousot
    A gentle introduction to formal verification of computer systems by abstract interpretation, *Logics and Languages for Reliability and Security*, 2010, NATO Science Series III: Computer and Systems Sciences, IOS Press, 29 pages.

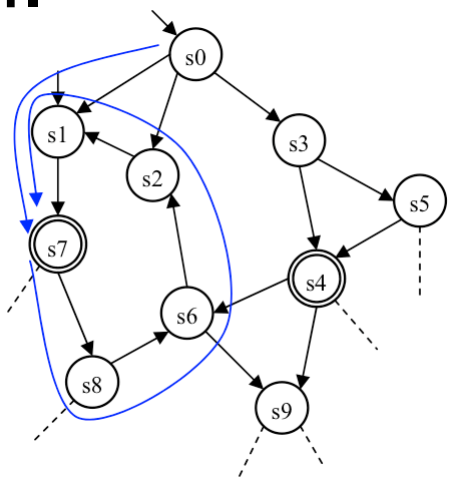


Методы верификации

- «Полное» тестирование
- Имитационное моделирование
- Доказательство теорем
- Статический анализ
- **Верификация на моделях**
- Динамическая верификация

Верификация программ на моделях (model checking)

- Проверка свойства на конечной модели программы,
- Свойства – в терминах значения предикатов в состоянии программы и последовательности значений.
- Исчерпывающий поиск по пространству состояний.



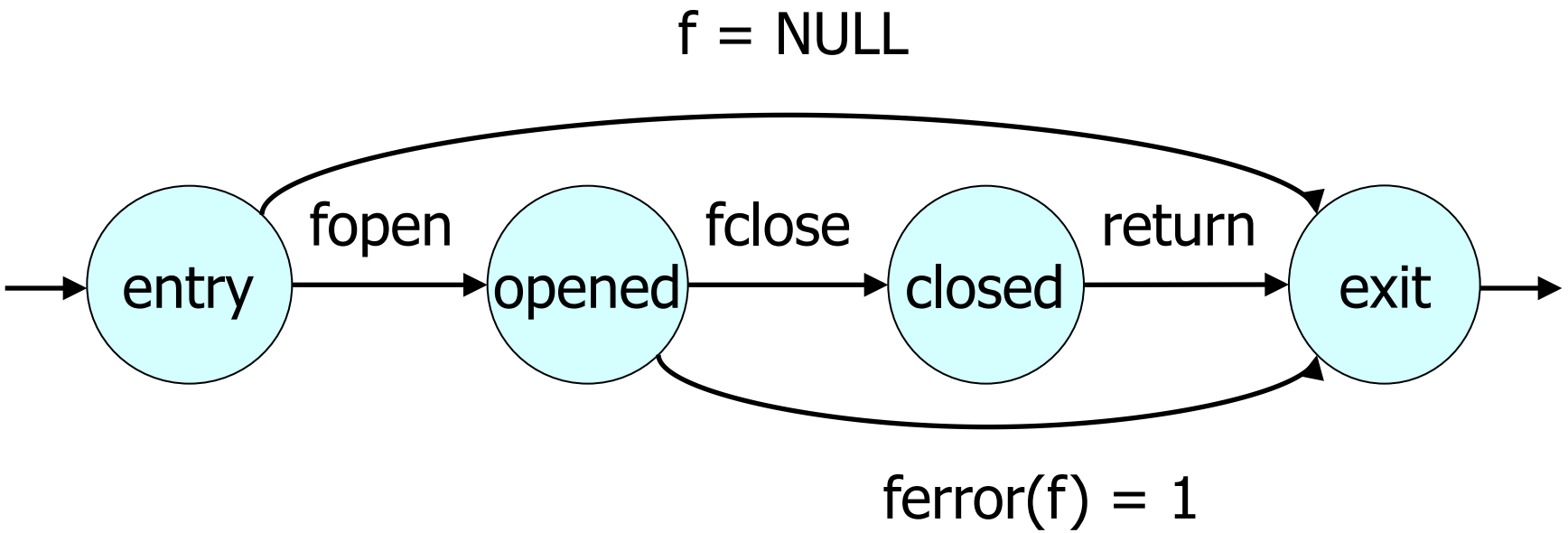
Верификация программ на моделях: пример

```
int count_lines(const char* filename) {
    int c, count = 0;
    FILE* f = fopen(filename, "r");
    if (f != NULL) {
        c = fgetc(f);
        while (c != EOF) {
            if (c == '\n') {
                ++count;
            }
            c = fgetc(f);
        }
        if (ferror(f)) {
            return -1;
        }
        fclose(f);
        return count;
    } else {
        return -1;
    }
}
```

Всегда ли функция закрывает открытый файл ?

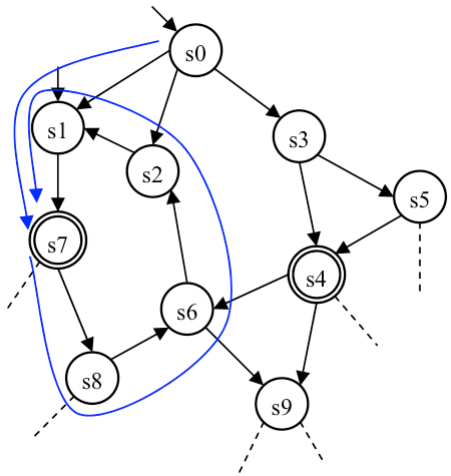
Верификация программ на моделях: пример

- Модель функции count_lines:



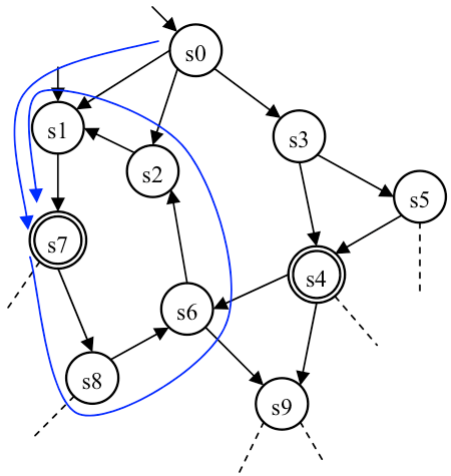
Процесс верификации программ на моделях

- Моделирование
  - Построить адекватную и корректную модель,
  - Избежать «лишних» состояний;
- Спецификация свойств
  - Темпоральная логика,
  - Полнота свойств;
- Верификация
  - Построение контрпримера,
  - Анализ контрпримера.



Верификация программ на моделях

- Достоинства
  - Хорошо автоматизируем,
  - Если модель конечна, корректна и адекватна проверяемому свойству, то даётся точный ответ,
  - Выявляет редкие ошибки.
- Недостатки
  - Работает только для конечных моделей.



Динамическая верификация

- Иногда на этапе разработки системы невозможно гарантировать её правильную работу в ходе эксплуатации:
  - Динамическое изменение конфигурации системы

    пример: компьютерная сеть
  - Неполное описание компонентов системы

    пример: мультивендорная система
  - Работа в сложном окружении

    пример: взаимодействие с Интернет

Динамическая верификация

- Т.е. невозможно составить достаточно детальное конечное описание системы
- Решение: в систему добавляется компонент, выполняющий
  - **мониторинг** поведения системы,
  - **анализ** наблюдаемого поведения,
  - **реакцию** на обнаруженные нарушения спецификации

Динамическая верификация

- Проверяется правильность не ***описания программы***, а её ***наблюдаемого поведения***
- Примеры:
  - Система контроля поведения приложений в ОС
  - Система обнаружения атак
  - Встроенная система контроля

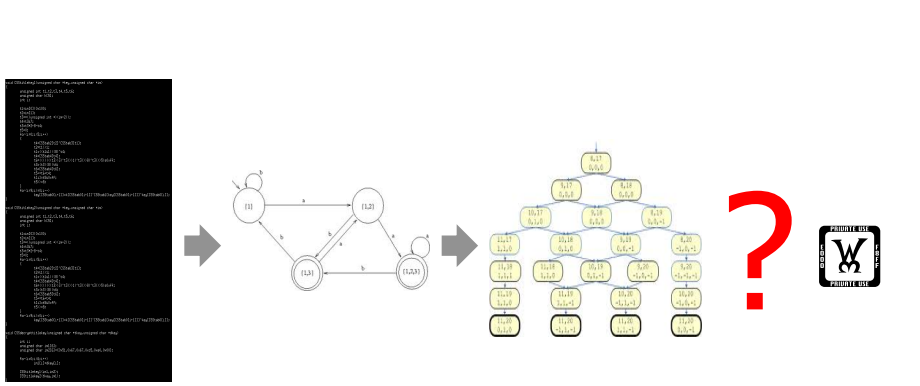
История развития[1]: фундамент

- Флойд, 1967 – assertions, гипотеза о доказуемости корректности программы,
- Хоар, 1969 – пред- и пост-условия, триплеты Хоара (P | S | Q), логический вывод,
- Бойер, Мур, 1971 – первый автоматический прувер,
- Дейкстра, 1975 – Guarded Command Languages,
- Хоар, 1978 – взаимодействующие последовательные процессы (CSP),
- Милнер, 1980 – Calculus of Communicating Systems (CCS)

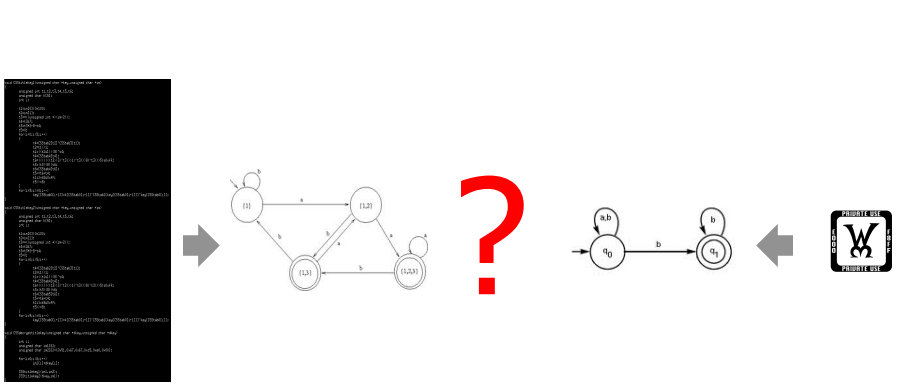
[1]Vahe Poladian, Software Technology Maturation Study: Model Checking Techniques and Tools, 2001.

История развития: развитие методов

- 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, 1989 – верификатор SPIN.



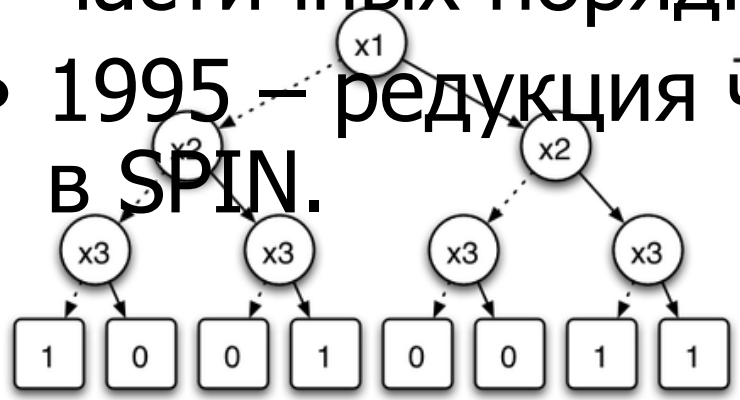
1981



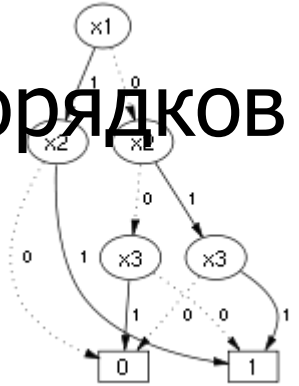
1986

История развития: проблема «комбинаторного взрыва»

- 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.



x1	x2	x3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



История развития: дальнейшее развитие

- 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

План лекции

- Правильность программ
- Актуальность верификации
- Формальные методы проверки правильности
- **Обзор курса**
- **Практикум**

Программа курса

- Практические знания:
  - Оценка числа состояний программы, построение модели программы на Promela, спецификация свойств программы, верификация при помощи Spin, анализ результатов верификации
- Теоретическая основа:
  - Математическая модель программы (состояние, вычисление, система переходов граф программы)
  - Построение модели (абстракция) программы, вопросы её корректности

Материалы

Сайт курса: http://savenkov.lvk.cs.msu.su/mc.html

Материалы по SPIN:

- Holzmann. The Spin Model Checker: Primer and Reference Manual, Addison Wesley, 2003.
- Сайт Spin: http://www.spinroot.com.

Дополнительная литература:

- Кларк, Грумберг, Пелед. Верификация моделей программ: Model checking, МЦНМО, 2002.
- Peled. Software Reliability Methods, Springer, 2001.
- Миронов А.М. Верификация программ методом Model Checking. http://intsys.msu.ru/staff/mironov/modelchk.pdf
- Миронов А.М. Теория процессов http://intsys.msu.ru/staff/mironov/processes.pdf

Практикум

- Первое задание – 20.02;
- Ауд. 758, Linux, SPIN;
- 5 задач, по мере прохождения курса;
- Экзамен – ~~устный;~~***посмотрим***
- E-mail: *model-checking@lvk.cs.msu.su*,
- **Подписаться: отправить ФИО и номер группы** на *model-checking@lvk.cs.msu.su*,
- Информация и задачи – по почте.

Оценка за практикум и курс

- Курс ориентирован на получение **знаний** и **навыков**,
- Навыки не получится развить перед экзаменом и продемонстрировать за 20 минут
- Оценка за курс:
  - Оценка за практикум (0..3 балла),
  - Оценка за экзамен (0..3 балла),
  - Оценка за «летучки» (-1..1 балл).
- Для АСВК оценка за практикум идёт в диплом.

Оценка за практикум и курс

- 5 задач: 50+100+50+80+100 = 380 очков
  - [0,230) – 0 баллов (неуд по практикуму, недопуск на основной экзамен),
  - [230,300) – 1 балл,
  - [300,380) – 2 балла,
  - **380** – 3 балла (кандидат на «автомат»);
- На занятиях можно сдавать сколько угодно раз, вплоть до получения максимального балла;
- Если решение задачи присылается по почте, оценка за него является окончательной;
- Сдача задания после дедлайна:
  - в течение двух недель,
  - только очная (на семинаре),
  - оценка умножается на 0.6.

Спасибо за внимание!