

Верификация программ на моделях

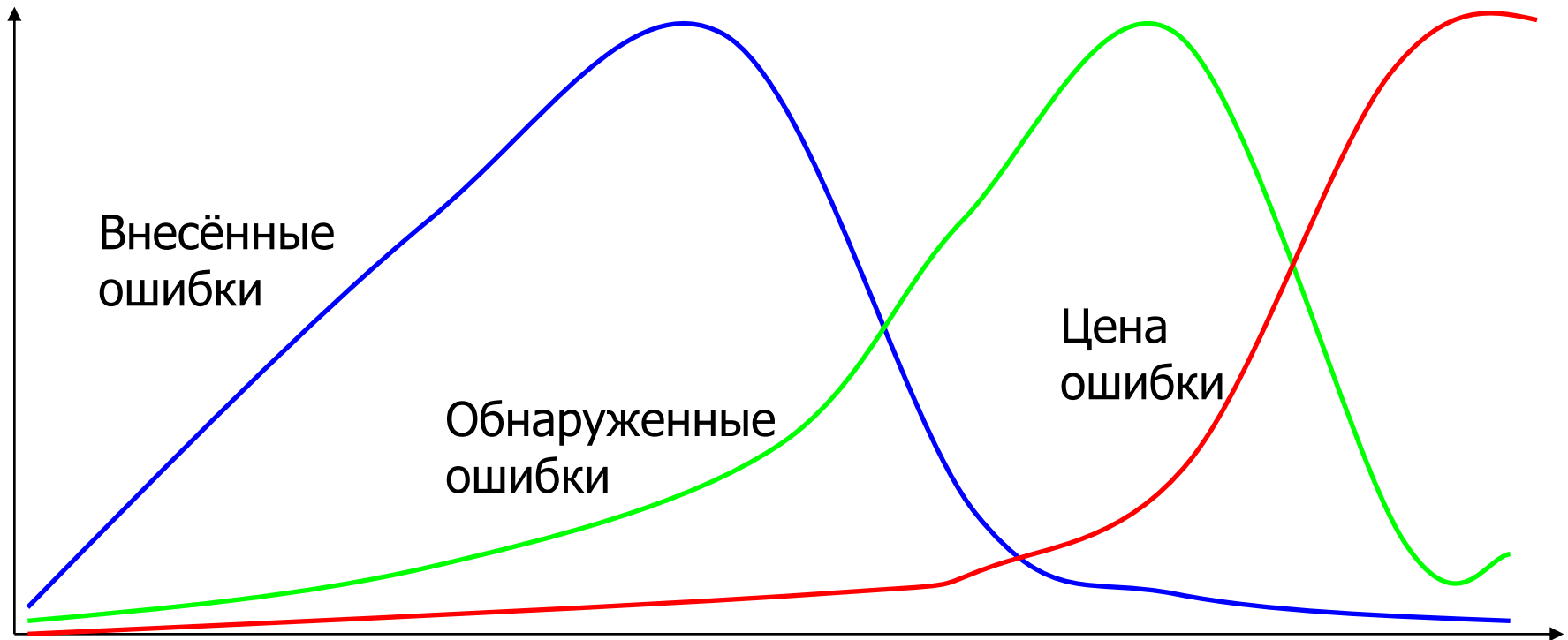
Константин Савенков (лектор)

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

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.



Выборочное тестирование

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

Выявление ошибок

частые

редкие

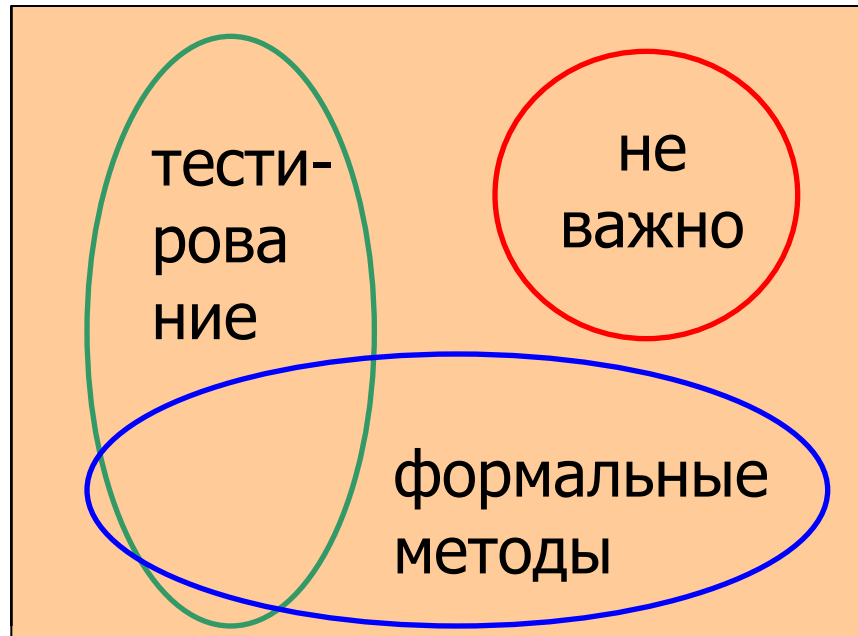
безвредные

тести-
рова
ние

не
важно

критические

формальные
методы



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: минусы

- **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

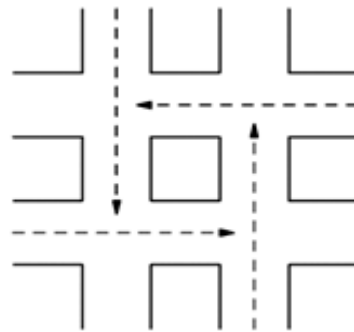
- Как правило, разработка критически важных систем регулируется каким-либо стандартом; Например, RTCA/DO-178B (авионика);
- Стандарты разрабатывались давно, поэтому основной способ верификации там – тестирование;
- Пример обоснования полноты тестового покрытия для критических систем – **MC/DC**
(<http://techreports.larc.nasa.gov/ltrs/PDF/2001/tm/NASA-2001-tm2108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

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.**

Реактивные программы

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

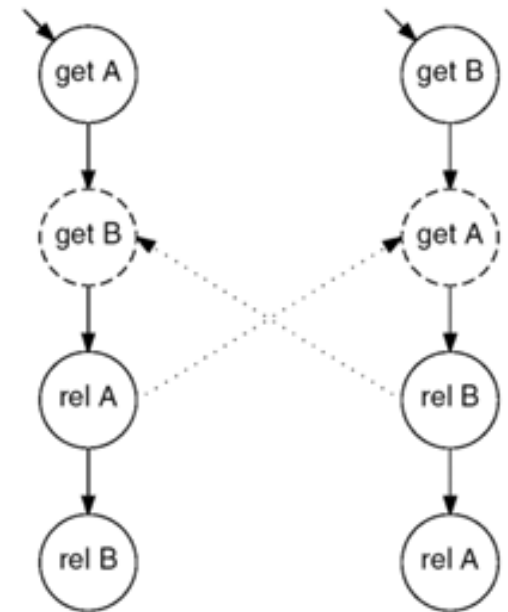


- Правила, реализованные в программах, должны быть универсальны

Системы с разделением ресурсов

Примеры:

- Дорожный трафик,
- Телефонные сети,
- Операционные системы,
- ...



Параллельные системы

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

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:

$$\frac{S_1 = a_1 \bullet S_2 \vee S_2 = a_2 \bullet S_3}{S_1 = a_1 \bullet a_2 \bullet S_3}, \frac{S_1 \bullet S_2 \vee S_2 \bullet S_3}{S_1 \bullet S_3}$$

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

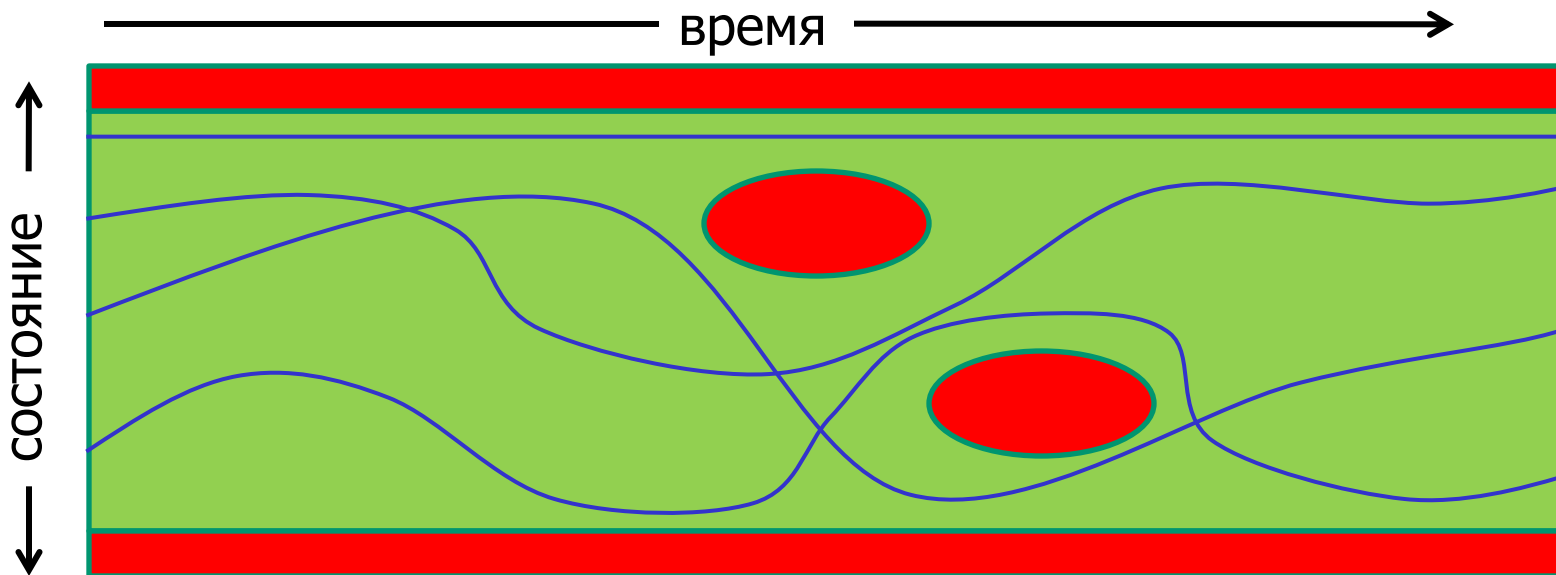
- Примеры инструментальных средств:
 - Isabelle/HOL, Coq, PVS, Vampire, SPASS, ACL2, Simplify, Microsoft Z
- Что почитать:
 - ATP было в курсе «Математическая логика» В.А.Захарова
 - Xavier Leroy, Mechanized semantics with applications to program proof and compiler verification, INRIA, France, 2009 (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

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.

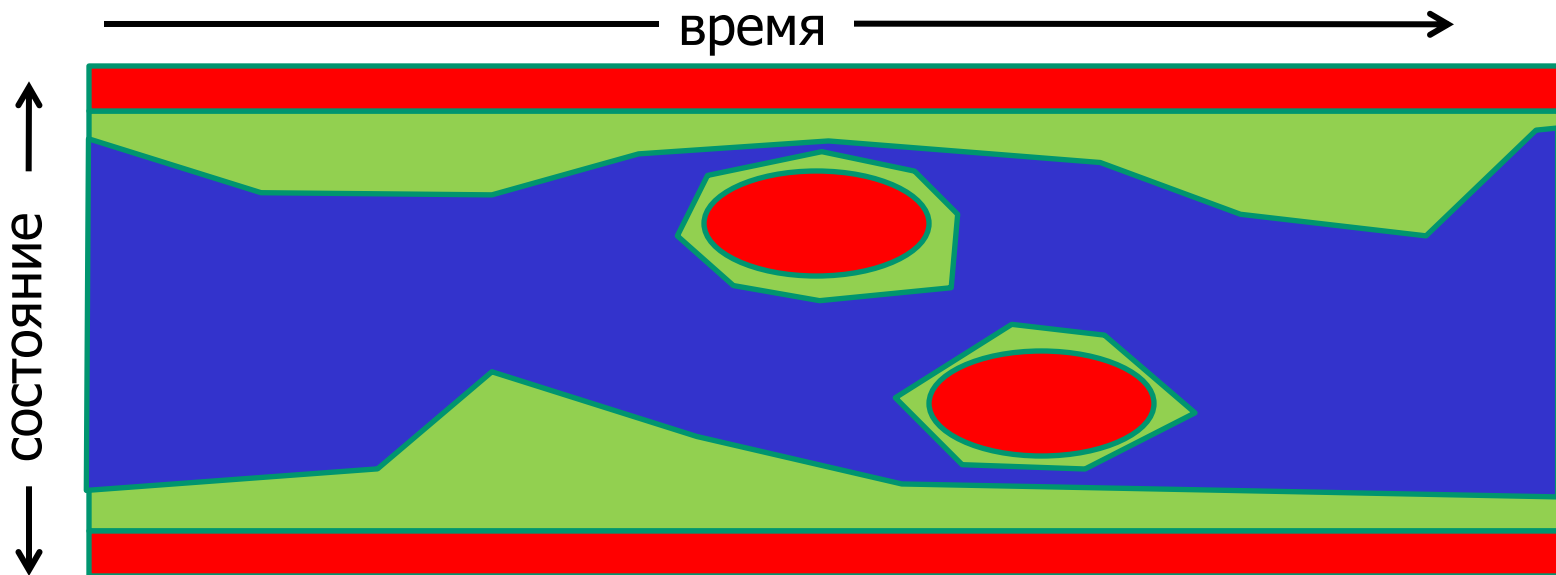
Статический анализ

- 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;  
}
```

$$\text{dom}(m) = \text{Int} + \{ \omega \}$$
$$\text{NI} = \{ \omega \}$$
$$I = \text{Int}$$
$$v: \text{Expr} \rightarrow \{ \text{NI}, I \}$$

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 }>  
}
```

$\text{dom}(m) = \text{Int} + \{ \omega \}$

$\text{NI} = \{ \omega \}$

$I = \text{Int}$

$v: \text{Expr} \rightarrow \{ \text{NI}, I \}$

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.

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.

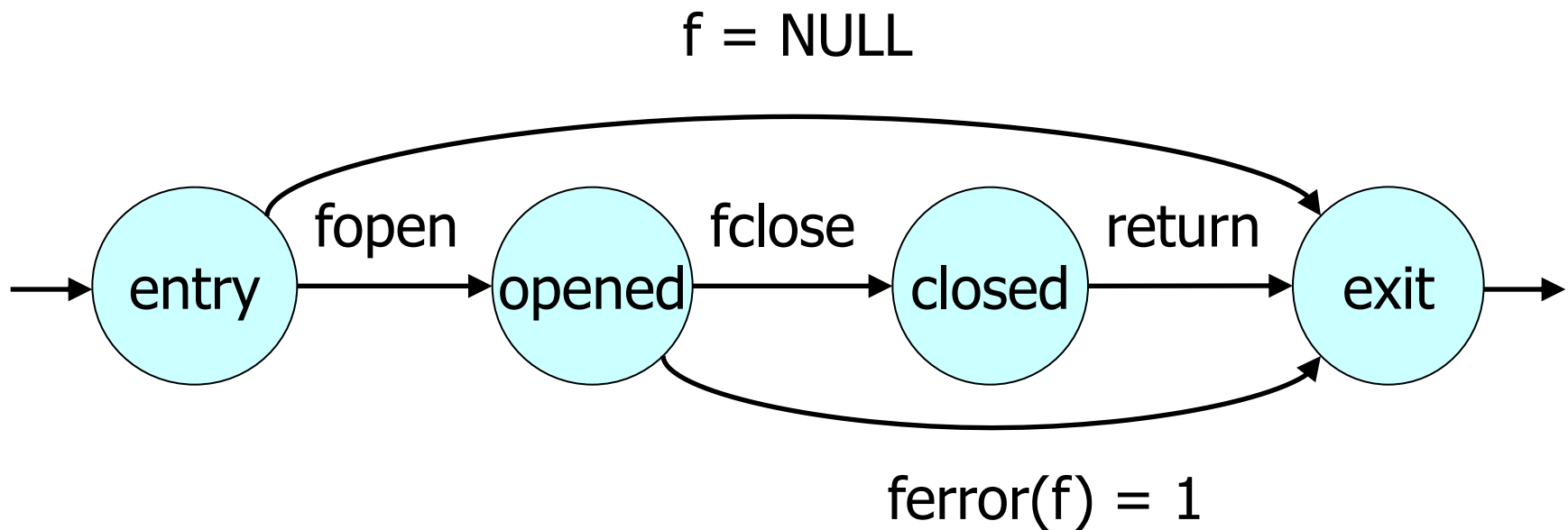
Верификация программ на моделях: пример

```
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

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

Литература

- Сайт курса:
<http://savenkov.lvk.cs.msu.su/mc.html>
- Кларк, Грумберг, Пелед. Верификация моделей программ: Model checking, МЦНМО, 2002.
- Holzmann. The Spin Model Checker: Primer and Reference Manual, Addison Wesley, 2003.
- Peled. Software Reliability Methods, Springer, 2001.
- Сайт Spin: <http://www.spinroot.com>.

Практикум

- Ауд. 758, Linux, SPIN;
- 6 задач, по мере прохождения курса;
- Экзамен – устный;
- 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

- 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;
- Если решение задачи присылается по почте, оценка за него является окончательной.

Спасибо за внимание!