Верификация программ на моделях

Лекция №2

Моделирование программ

Константин Савенков (лектор)

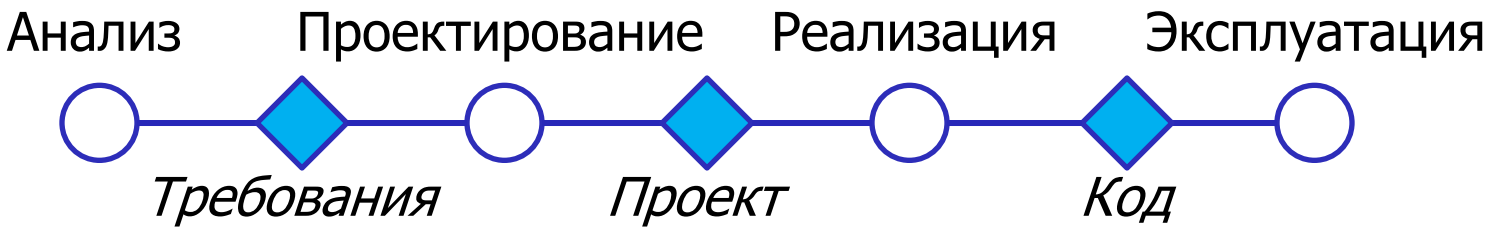
План лекции

- **Формальные методы проверки правильности программ:** место в жизненном цикле ПО, автоматизируемость
- **Верификация программ на моделях:** история развития, области применения, общая схема
- **Моделирование программ:** моделируемые свойства, состояние программы, процесс построения модели

Формальные методы проверки правильности программ

(место в жизненном цикле ПО, вопросы автоматизации)

Итоги предыдущей лекции



Валидация

Тестирование

Статический анализ кода

Доказательство теорем

Верификация на моделях

Динамическая верификация

Автоматизируемость методов верификации

- **Тестирование:** автоматическое и автоматизированное,
- **Доказательство теорем:** существенное участие человека,
- **Статический анализ:** полностью автоматический для заданной области и свойства,
- **Верификация на моделях:** участие человека при построении модели и при анализе контрпримеров,
- **«Комбинаторный взрыв».**

Верификация программ на моделях

(история развития, области применения, общая схема, проверяемые свойства)

История развития: 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)

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.

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.

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

Области применения верификации на моделях

- Сетевые и криптографические протоколы,
- Протоколы работы кэш-памяти,
- Интегральные схемы,
- Стандарты (напр. FutureBus+),
- Встроенные системы,
- Драйвера,
- Вообще программы на C.

Примеры использования SPIN

- Верификация системы управления дамбами в Роттердаме (Нидерланды)

- Верификация АТС PathStar (Lucent)
- Верификация аэрокосми-ческих систем (DeepSpace 1, Cassini, Mars Exploration Rovers, итд.)

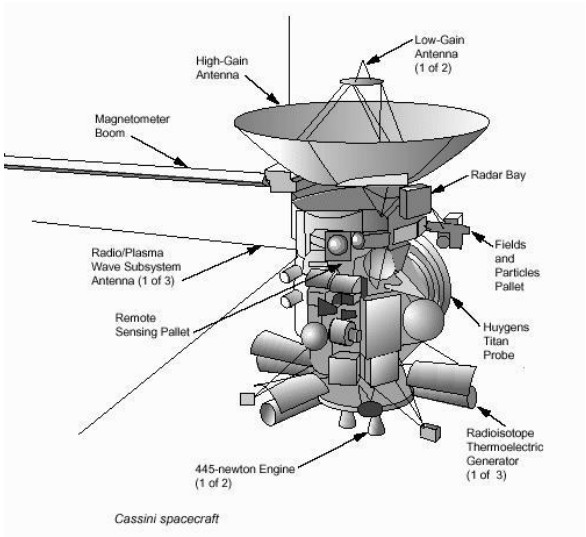
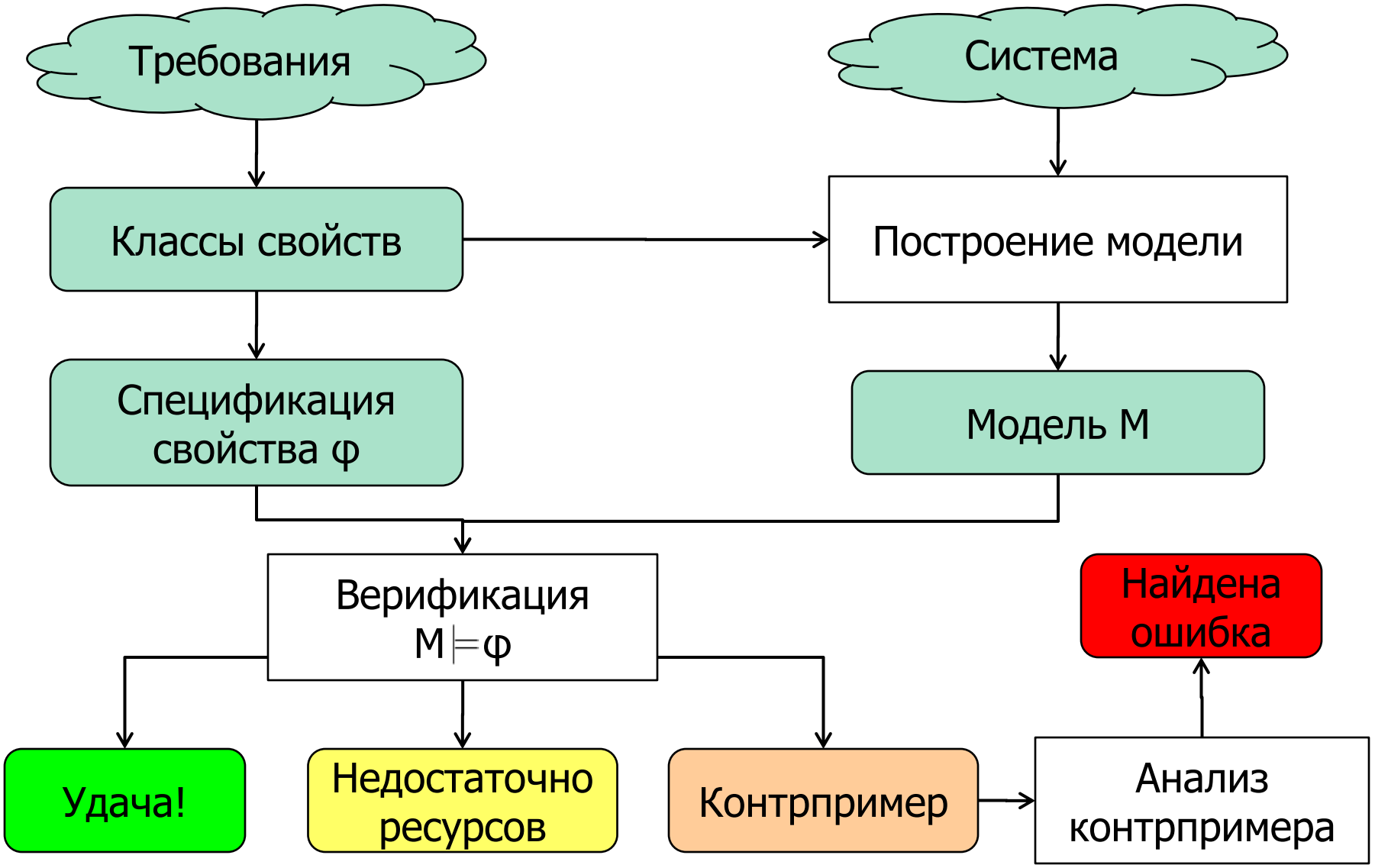


Схема верификации на модели



Примеры классов свойств

- Стандартные:
  - Отсутствие ошибок времени выполнения (RTE),
  - Отсутствие тупика (deadlock),
  - Отсутствие удушения (starvation),
  - Не срабатывают ассерты (assertions).
- Зависящие от приложения:
  - Инварианты системы,
  - Индикаторы прогресса,
  - Корректная завершаемость,
  - Причинно-следственный и темпоральный порядок на состояниях системы,
  - Требования справедливости.

Моделирование программ

(пример построения модели,
общая схема моделирования)

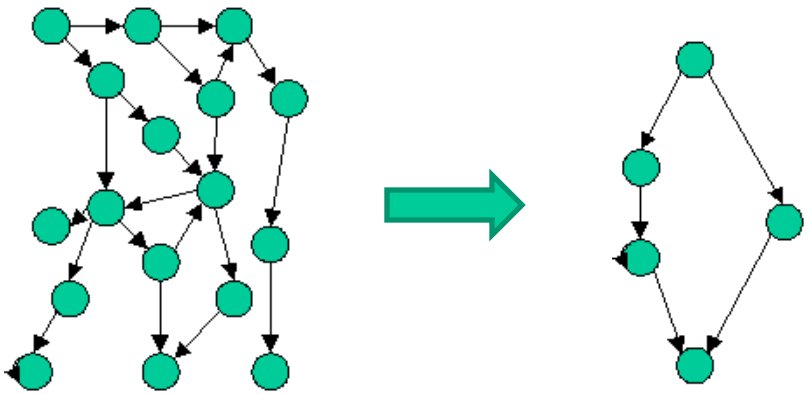
Зачем строить модели программ?

- Свойства задают допустимые состояния и последовательности состояний (вычисления)

  т.н. *свойства линейного времени*

- Модель генерирует трассы, по которым можно проверить свойства,

- Абстракция от лишних состояний.



Состояние программы

- Состояние программы – совокупность значений объектов данных и счётчика управления,
- Как правильно оценить количество состояний?

Спецификация программы	**Исходный код на языке высокого уровня**	**Исполняемый код (ASM)**	**Программно-аппаратная система**	**Работающий компьютер**
Может вообще не быть состояний (stateless протоколы)	Данные – переменные языка, счетчик – номер выполняемого оператора	Данные – ячейки памяти в адресном пр-ве процессора и регистры, счётчик -- регистр команд процессора	Данные – ячейки памяти различных ЗУ, кэш, буфера выборки, состояние конвейеров, Счётчик – адрес последней команды	Атомы, электроны, спины …

- Понятие состояния **всегда** связано с некоторой моделью программы

Состояние программы
(на уровне языка C)

```
    int compare(int a, int b) {
0:          int res;
1:          if (a < b) {
2:                  res = -1;
3:          } else if (a >= b) {
4:                  res = 1;
            } else {
5:                  res = 0;
            }

6:          return res;
    }
```

Потенциально –

$7*2^{96}$

состояний

3 целочисленных переменных,
$|int| = 2^{32}$,
7 операторов языка.

Достижимые состояния

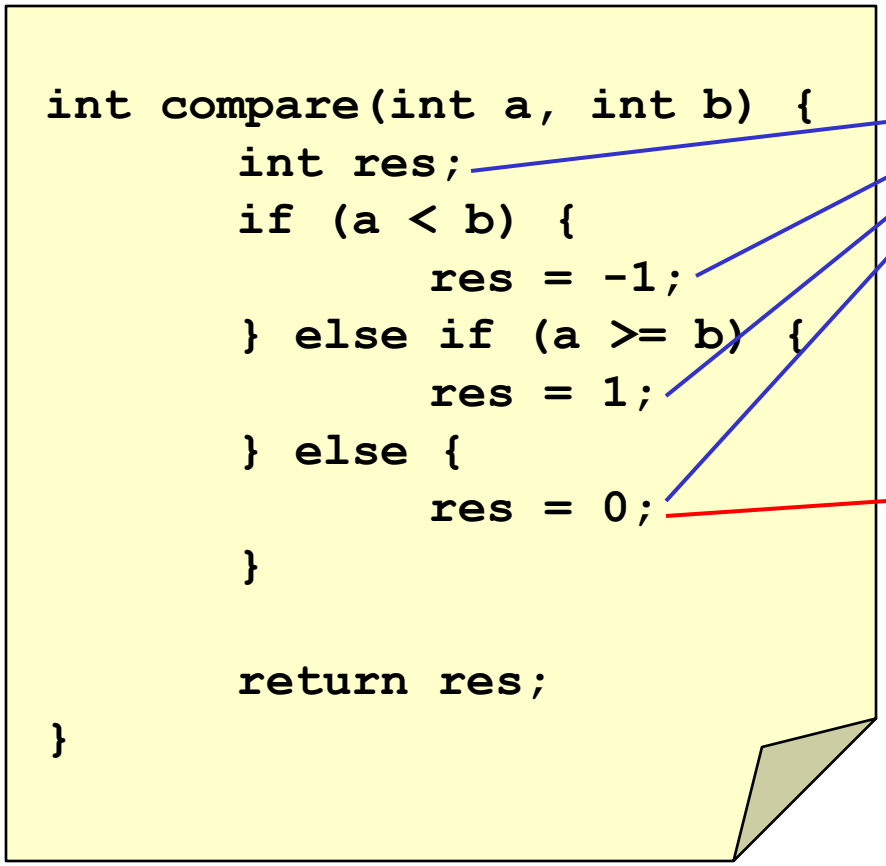
- Состояние называется **достижимым**, если существует вычисление программы, в котором оно присутствует

(формализация – позже)

- Достижимость состояния в программе в общем случае алгоритмически неразрешима.

(неразрешима даже достижимость точки программы)

Число достижимых состояний



```
    int compare(int a, int b) {
0:          int res;
1:          if (a < b) {
2:                  res = -1;
3:          } else if (a >= b) {
4:                  res = 1;
            } else {
5:                  res = 0;
            }

6:          return res;
    }
```

$7*4*2^{64}$

переменная **res** может принимать только 4 значения

$6*3*2^{64}$

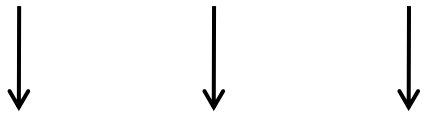
5-й оператор не достижим

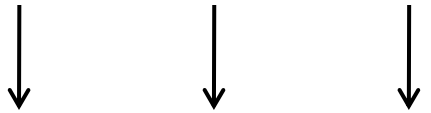
Пример построения модели программы

- Программа подсчёта количества слов в файле
- Проверяемое свойство – всегда ли программа закрывает открытый файл?
- См. следующий слайд.

Пример построения модели программы

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc,
             const char* argv[]) {
  FILE* f;
  int c, wordCnt = 0, inWord = 0;
  if (argc < 2) {
    printf("Use: %s filename\n”,
                            argv[0]);
    return 1;
  }
  f = fopen(argv[1], "r");
  if (f == NULL) {
    printf("Can't open file:
                        %s\n",
                        argv[1]);
    return 1;
  }
```

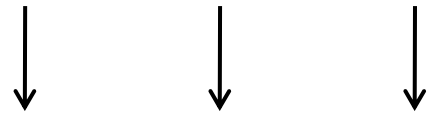


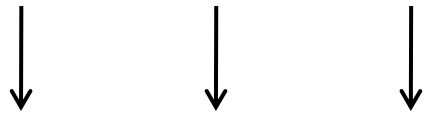


```
while ((c = fgetc(f)) != -1) {
    if (!isspace(c)) {
      if (!inWord) {
        ++wordCnt;
        inWord = 1;
      }
    } else {
          inWord = 0;
    }
  }
  printf("Word count: %d\n",
                        wordCnt);
  fclose(f);
  return 0;
}
```

Пример построения модели программы

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc,
             const char* argv[]) {
  FILE* f;
  int c, wordCnt = 0, inWord = 0;
  if (argc < 2) {
    printf("Use: %s filename\n”,
                              argv[0]);
    return 1;
  }
  f = fopen(argv[1], "r");
  if (f == NULL) {
    printf("Can't open file:
                         %s\n",
                         argv[1]);
    return 1;
  }
```

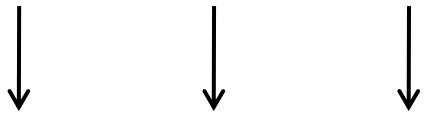


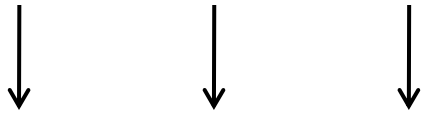


```
while ((c = fgetc(f)) != -1) {
    if (!isspace(c)) {
      if (!inWord) {
        ++wordCnt;
        inWord = 1;
      }
    } else {
          inWord = 0;
    }
  }
  printf("Word count: %d\n",
                         wordCnt);
  fclose(f);
  return 0;
}
```

Пример построения модели программы

```
#include <stdio.h>
#include <stdlib.h>
int main() {
  FILE* f;
  int c, wordCnt = 0, inWord = 0;
  enum {FALSE, TRUE} P1 = any();
  if (P1) {
    return 1;
  }
  f = fopen(“sample”, "r");
  if (f == NULL) {
    return 1;
  }
```

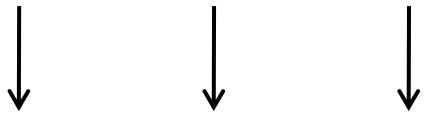


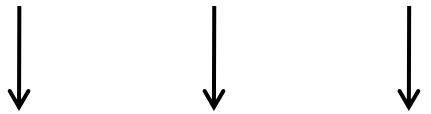


```
while ((c = fgetc(f)) != -1) {
    if (!isspace(c)) {
      if (!inWord) {
        ++wordCnt;
        inWord = 1;
      }
    } else {
          inWord = 0;
    }
  }
  fclose(f);
  return 0;
}
```

Пример построения модели программы

```
#include <stdio.h>
#include <stdlib.h>
int main() {
  FILE* f;
  int c, wordCnt = 0, inWord = 0;
  enum {FALSE, TRUE} P1 = any();
  if (P1) {
    return 1;
  }
  f = fopen(“sample”, "r");
  if (f == NULL) {
    return 1;
  }
```

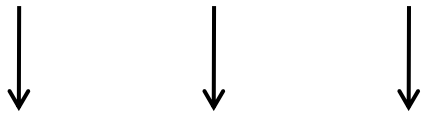


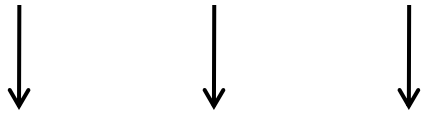


```
while ((c = fgetc(f)) != -1) {
    if (!isspace(c)) {
      if (!inWord) {
        ++wordCnt;
        inWord = 1;
      }
    } else {
          inWord = 0;
    }
  }
  fclose(f);
  return 0;
}
```

Пример построения модели программы

```
int main() {
  enum { FCLOSED, FOPEN, FERROR} fileState;
  enum { V0, V1 } inWord = V0;
  enum {FALSE, TRUE} P1 = any(),
                     P2, P3;
  if (P1) {
    return 1;
  }
  if(any()) {
    fileState = FOPEN;
  } else {
    fileState = FERROR;
  }
  if (fileState == FERROR) {
    return 1;
  }
```

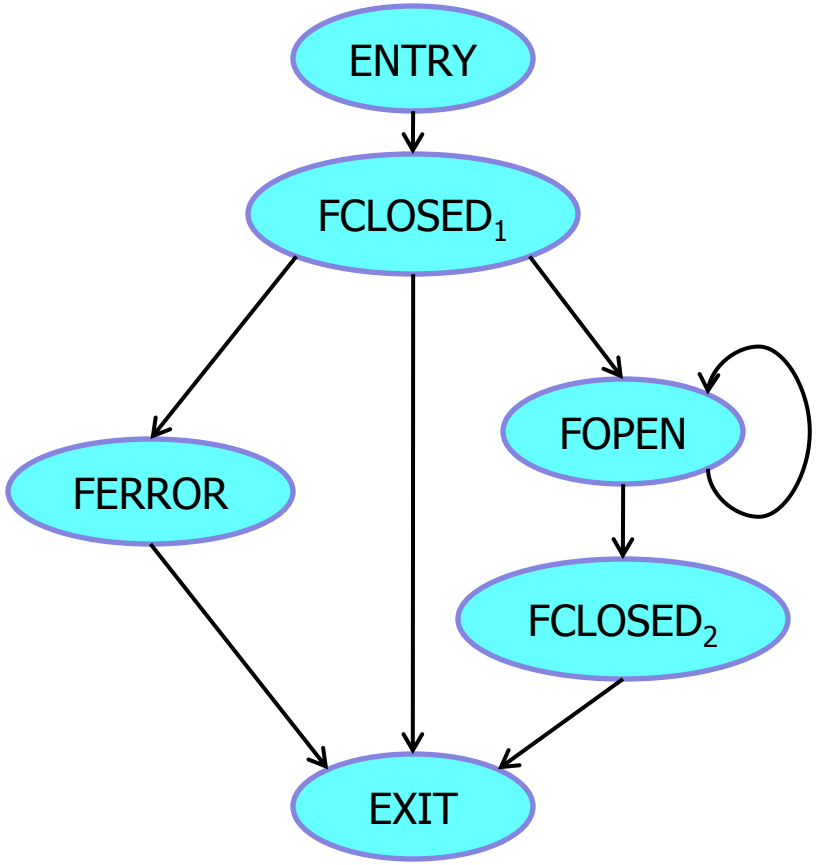




```
while (P2 = any()) {
    if (P3 = any()) {
      if (!inWord) {
        inWord = V1;
      }
    } else {
        inWord = V0;
    }
  }
  fileState = FCLOSED;
  return 0;
}
```

Пример построения модели программы

```
int main() {
  enum { FCLOSED, FOPEN, FERROR} fileState;
  if(any()) {
    fileState = FERROR;
  } else if (any()) {
    fileState = FOPEN;
    while (any());
    fileState = FCLOSED;
  }
  return 0;
}
```

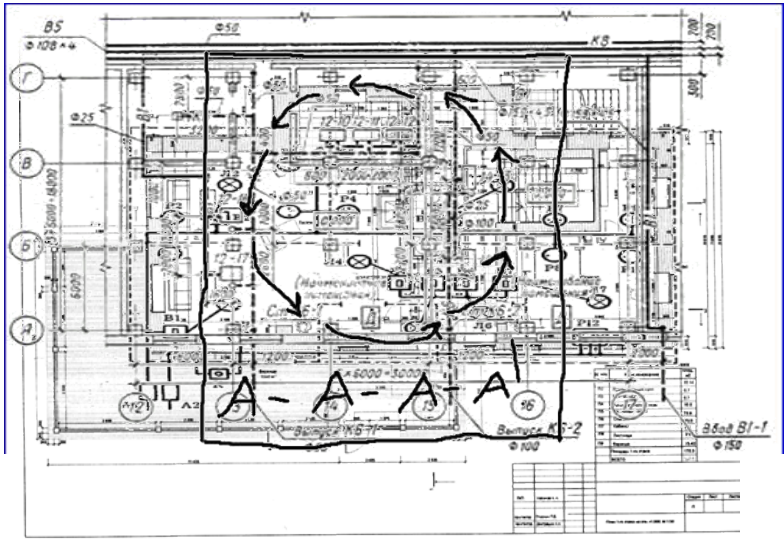


Моделирование

- Модель – упрощённое описание реальности, выполненное с определённой целью.
- Например, карта – модель местности.

Моделирование

- Модель – упрощённое описание реальности, выполненное **с определённой целью**.
- С одним объектом может быть связано несколько моделей.
- Каждая модель отражает свой аспект реальности
- Например, схемы зданий:
  - поэтажный план
  - электропроводка
  - водоснабжение
  - план эвакуации



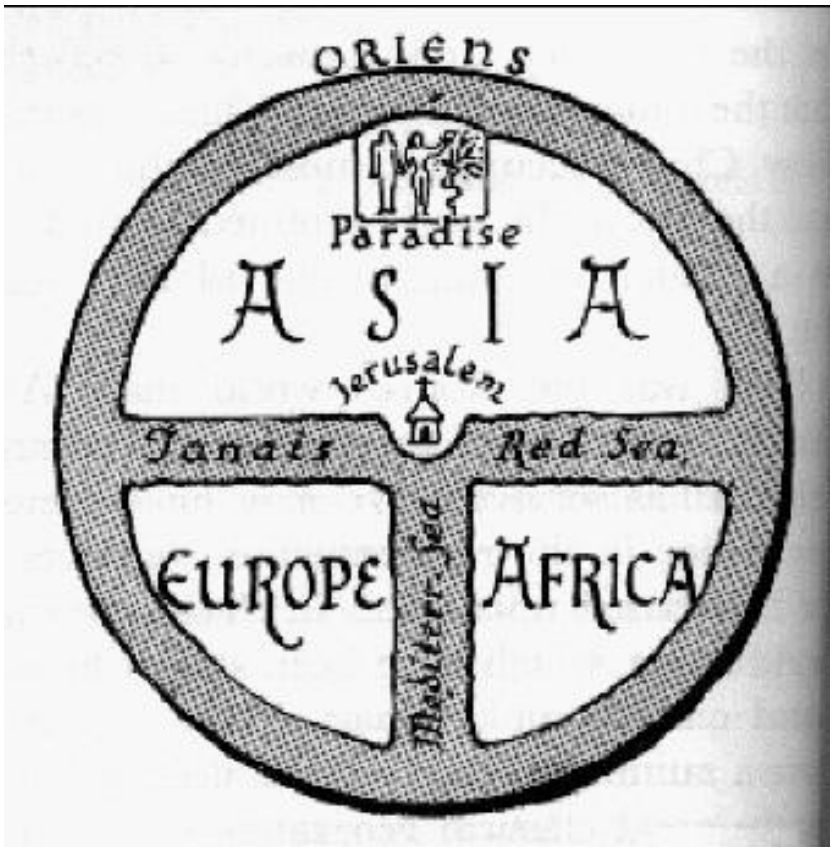
Моделирование

- Модель должна быть:
  - **Простой,**
    *быть проще, чем **реальность***
  - **Корректной,**
    *не расходиться с **реальностью***
  - **Адекватной.**
    *соответствовать **решаемой задаче***

Ни одно из этих качеств нельзя проверить на основе только описания модели

Моделирование

- Пример сильной абстракции:

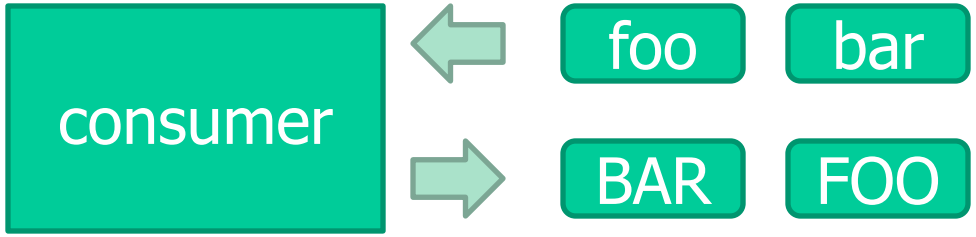


Построение модели

- Формализуемые требования (постановка задачи моделирования)
- Выбор языка моделирования
- Абстракция системы с учётом требований

Моделирование программ

процесс consumer:




```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define MSG_LEN 100

struct msgbuf {
  long mtype;
  char mtext[MSG_LEN];
};

int main(int argc,
              const char* argv[]) {
  key_t key;
  int qid, i;
  struct msgbuf msg;
  ssize_t size;
  key = ftok("./mc_lec2_ex3", 1);
  if ((qid = msgget(key,
        IPC_CREAT | 0666)) == -1) {
    perror("Error creating
                  message queue");
  }
  while (1) {
    size = msgrcv(qid, &msg,
                   MSG_LEN, 1, 0);
    for (i = 0; i < size; ++i) {
      msg.mtext[i] =
             toupper(msg.mtext[i]);
    }
    /* send it back */
    if (size > 0) {
      msgsnd(qid, &msg, size, 0);
    }
  }
}
```

Моделирование программ

процесс producer:


```
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define MSG_LEN 100
struct msgbuf {
  long mtype;
  char mtext[MSG_LEN + 1];
};

int main(int argc,
             const char* argv[]) {
  key_t key;
  int qid;
  struct msgbuf msg;
  ssize_t size;
  msg.mtype = 1;
  key =
        ftok("./mc_lec2_ex3", 1);
  if ((qid = msgget(key, IPC_CREAT
                  | 0666)) == -1) {
msgget_error:
    perror("Error creating
                  message queue");
  }
  while (1) {
     printf("Your message: ");
     fgets(msg.mtext, MSG_LEN, stdin);
     if (msgsnd(qid, &msg,
       strlen(msg.mtext), 0) == -1) {
       perror("Error sending");
       continue;
    }
    size = msgrcv(qid, &msg, MSG_LEN, 1, 0);
    if (size > 0) {
      printf("Reply is: %s\n", msg.mtext);
    } else {
      printf("No reply\n");
    }
  }
}
```

Свойства правильности процесса producer:

- msgsnd всегда вызывается только после msgget;
- если поток управления попадает в msgget_error, то после этого не вызываются функции msgsnd и msgrcv;
- ни в одном вычислении не встречаются msgrcv без msgsnd между ними.

Свойство №1

процесс producer:

```
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define MSG_LEN 100
struct msgbuf {
  long mtype;
  char mtext[MSG_LEN + 1];
};

int main(int argc,
             const char* argv[]) {
  key_t key;
  int qid;
  struct msgbuf msg;
  ssize_t size;
  msg.mtype = 1;
  key =
        ftok("./mc_lec2_ex3", 1);
  if ((qid = msgget(key, IPC_CREAT
                  | 0666)) == -1) {
msgget_error:
    perror("Error creating
                  message queue");
  }
  while (1) {
      printf("Your message: ");
      fgets(msg.mtext, MSG_LEN, stdin);
      if (msgsnd(qid, &msg,
        strlen(msg.mtext), 0) == -1) {
        perror("Error sending");
        continue;
    }
    size = msgrcv(qid, &msg, MSG_LEN, 1, 0);
    if (size > 0) {
      printf("Reply is: %s\n", msg.mtext);
    } else {
      printf("No reply\n");
    }
  }
}
```

Свойство №1

процесс producer:

```
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define MSG_LEN 100
struct msgbuf {
  long mtype;
  char mtext[MSG_LEN + 1];
};

int main(int argc,
             const char* argv[]) {
  key_t key;
  int qid;
  struct msgbuf msg;
  ssize_t size;
  msg.mtype = 1;
  key =
        ftok("./mc_lec2_ex3", 1);
  if ((qid = msgget(key, IPC_CREAT
                  | 0666)) == -1) {
msgget_error:
    perror("Error creating
                  message queue");
  }
  while (1) {
      printf("Your message: ");
      fgets(msg.mtext, MSG_LEN, stdin);
      if (msgsnd(qid, &msg,
        strlen(msg.mtext), 0) == -1) {
        perror("Error sending");
        continue;
    }
    size = msgrcv(qid, &msg, MSG_LEN, 1, 0);
    if (size > 0) {
      printf("Reply is: %s\n", msg.mtext);
    } else {
      printf("No reply\n");
    }
  }
}
```

Свойство №1

процесс producer:

```
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stubs.h>

int main(int argc,
         const char* argv[]) {

  pass();
  pass();
  if (msgget() == -1) {

msgget_error:
    pass();

  }
  while (1) {
    pass();
    pass();
    if (msgsnd() == -1) {

      pass();
      continue;
    }
    pass();
    if (any()) {
      pass();
    } else {
      pass();
    }
  }
}
```

Свойства правильности процесса producer:

- msgsnd всегда вызывается только после msgget;
- если поток управления попадает в msgget_error, то после этого не вызываются функции msgsnd и msgrcv;
- ни в одном вычислении не встречаются msgrcv без msgsnd между ними.

Свойство №2

процесс producer:

```
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define MSG_LEN 100
struct msgbuf {
  long mtype;
  char mtext[MSG_LEN + 1];
};

int main(int argc,
             const char* argv[]) {
  key_t key;
  int qid;
  struct msgbuf msg;
  ssize_t size;
  msg.mtype = 1;
  key =
        ftok("./mc_lec2_ex3", 1);
  if ((qid = msgget(key, IPC_CREAT
                  | 0666)) == -1) {
msgget_error:
    perror("Error creating
                  message queue");
  }
  while (1) {
      printf("Your message: ");
      fgets(msg.mtext, MSG_LEN, stdin);
      if (msgsnd(qid, &msg,
        strlen(msg.mtext), 0) == -1) {
        perror("Error sending");
        continue;
    }
    size = msgrcv(qid, &msg, MSG_LEN, 1, 0);
    if (size > 0) {
      printf("Reply is: %s\n", msg.mtext);
    } else {
      printf("No reply\n");
    }
  }
}
```

Свойство №2

процесс producer:

```
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define MSG_LEN 100
struct msgbuf {
  long mtype;
  char mtext[MSG_LEN + 1];
};

int main(int argc,
             const char* argv[]) {
  key_t key;
  int qid;
  struct msgbuf msg;
  ssize_t size;
  msg.mtype = 1;
  key =
        ftok("./mc_lec2_ex3", 1);
  if ((qid = msgget(key, IPC_CREAT
                  | 0666)) == -1) {
msgget_error:
    perror("Error creating
                  message queue");
  }
  while (1) {
      printf("Your message: ");
      fgets(msg.mtext, MSG_LEN, stdin);
      if (msgsnd(qid, &msg,
        strlen(msg.mtext), 0) == -1) {
        perror("Error sending");
        continue;
    }
    size = msgrcv(qid, &msg, MSG_LEN, 1, 0);
    if (size > 0) {
      printf("Reply is: %s\n", msg.mtext);
    } else {
      printf("No reply\n");
    }
  }
}
```

Свойство №2

процесс producer:

```
#include <stub.h>

int main(int argc,
         const char* argv[]) {

  pass();
  pass();
  if (msgget() == -1) {

msgget_error:
    pass();

  }
  while (1) {
    pass();
    pass();
    if (msgsnd() == -1) {

      pass();
      continue;
    }
    msgrcv();
    if (any()) {
      pass();
    } else {
      pass();
    }
  }
}
```

не выполняется!

подходит и для проверки свойства №3

Свойства правильности процесса producer:

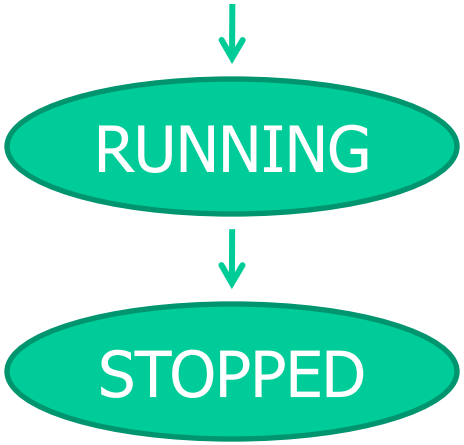
- msgsnd всегда вызывается только после msgget;
- если поток управления попадает в msgget_error, то после этого не вызываются функции msgsnd и msgrcv;
- ни в одном вычислении не встречаются msgrcv без msgsnd между ними.

Корректная, но практически бесполезная модель

```
int main() {

enum {RUNNING, STOPPED} state;

state = RUNNING;
state = STOPPED;
}
```



Построение модели

- Задачи:
  1. Постановка задачи моделирования

     не все требования к программе формализуемы
  2. Формализация описания программы

     описание программы предназначено для удобства разработки и компиляции, но не для проверки её свойств
  3. Абстракция

     уменьшение пространства состояний программы

Спасибо за внимание!
Вопросы?

Литература

- Сайт курса: http://savenkov.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.