НАДЁЖНОСТЬ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Лекция 2:
Моделирование программ

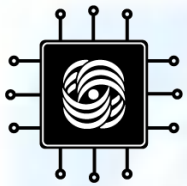
ВМиК МГУ им. М.В. Ломоносова,
Кафедра АСВК, Лаборатория Вычислительных Комплексов
Ассистент Волканов Д.Ю.

План лекции

- **Формальные методы проверки правильности программ:** место в жизненном цикле ПО, автоматизируемость
- **Верификация программ на моделях:**

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

Доказательств о 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

(области применения, общая схема, проверяемые свойства)

Области применения верификации на моделях

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

Примеры использования SPIN

- Верификация системы управл
  Роттердаме (Нидерланды)

- Верификация АТС

PathStar (Lucent)

- Верификация аэрокосми- ческих
  (DeepSpace 1, Cassini, Mars Explo
  итд.)

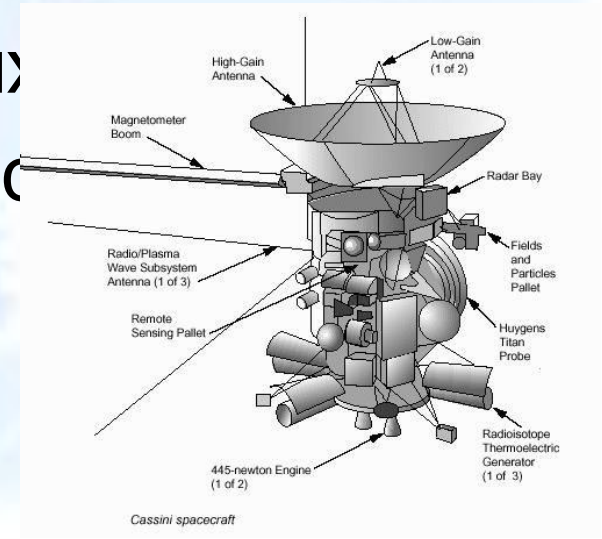
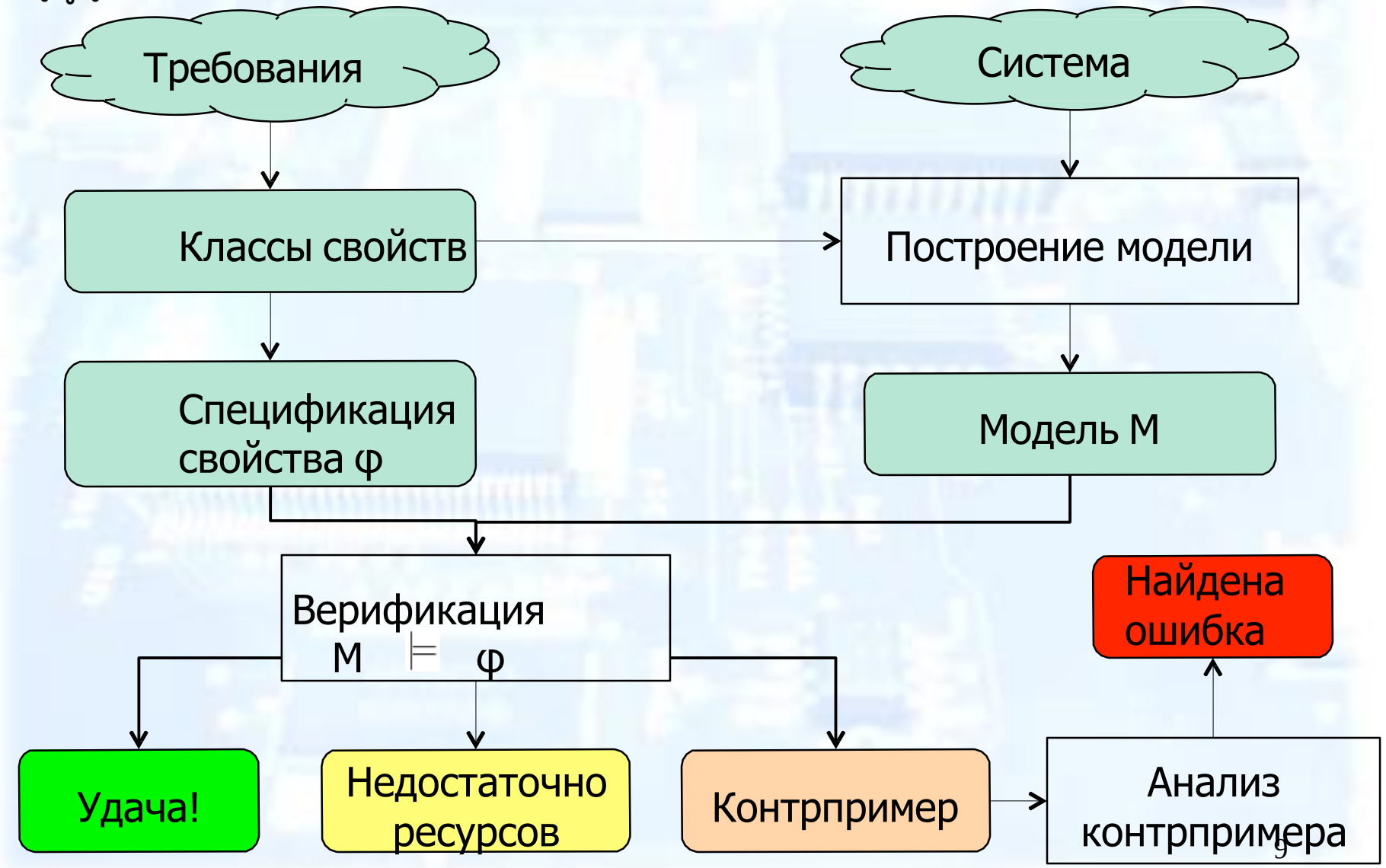
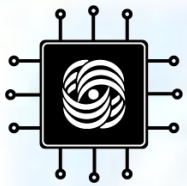


Схема верификации на модели



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

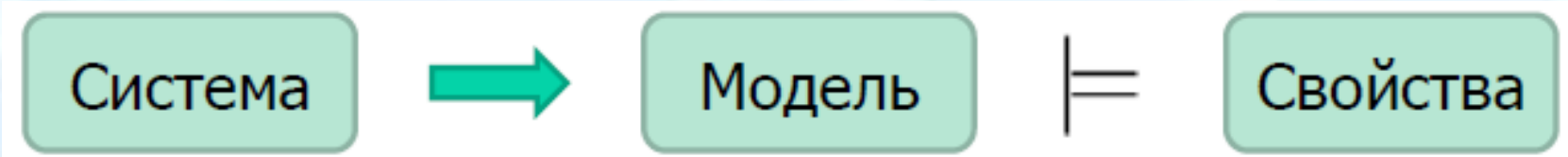
(пример построения модели,
общая схема моделирования)

Зачем строить модели программ?

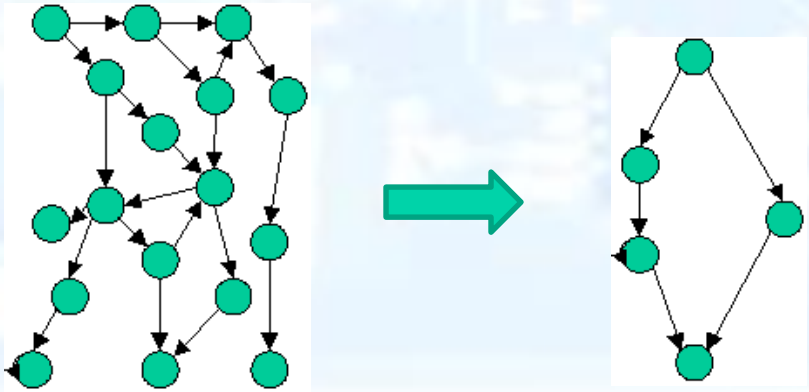
- Свойства задают допустимые состояния и последовательности состояний (вычисления)

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 …

Состояние программы (на уровне языка С)

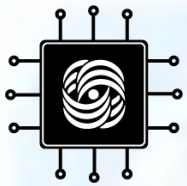
```
    int compare(int a, int b) {
0:      int res;
1:      if (a < b) {
2:          res = -1;
3:      } else if (a >= b) {
4:          res = 1;
        } else {
5:          res = 0;
        }

6:      return res;
7:  }
```

Потенциально –

$8*2^{96}$

состояний

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



Состояние программы (на уровне языка C)

```
    void foo(int a) {
0:      int  x= 5;
1:      if   (a < x) {
2:      x =  x-a;
3:      }
4:  }

    void bar(int b) {
0:        int x;
1:        x = b;
2:  }
```

Потенциально –

$15*2^{128}$

состояний

4 целочисленных переменных, $|int| = 2^{32}$,
2 потока управления, в foo – 5 операторов, в bar – 3.

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:  }
```

$8*4*2^{64}$

переменная `res` может принимать только 4 значения

$7*3*2^{64}$

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

```
#include <stdio.h>
#include <stdlib.h>
int main() {
  FILE* f;
  int inWord = 0;
  enum {FALSE, TRUE} P1 = any(),
                     P2, P3;
  if (P1) {
    return 1;
  }
  f = fopen("sample", "r");
  if (f == NULL) {
    return 1;
  }
```

```
while (P2 = any()) {
    if (P3 = any()) {
      if(inWord)
        inWord = 1;
      }
    } else {
          inWord = 0;
    }
  }
  fclose(f);
  return 0;
}
```

Пример построения модели программы

```
#include <stdio.h>
#include <stdlib.h>
int main() {
  FILE* f;
  int inWord = 0;
  enum {FALSE, TRUE} P1 = any(),
                     P2, P3;
  if (P1) {
    return 1;
  }
  f = fopen("sample", "r");
  if (f == NULL) {
    return 1;
  }
```

```
while (P2 = any()) {
    if (P3 = any()) {
      if(inWord)
        inWord = 1;
      }
    } else {
            inWord = 0;
    }
  }
  fclose(f);
  return 0;
}
```

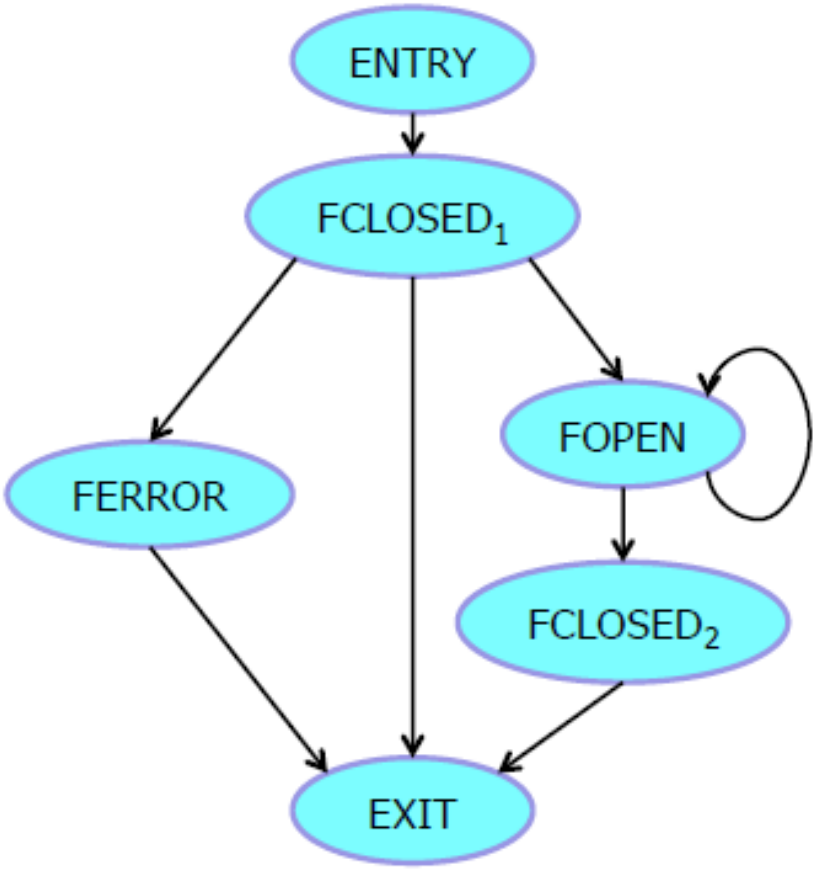
Пример построения модели программы

```
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;
}
```

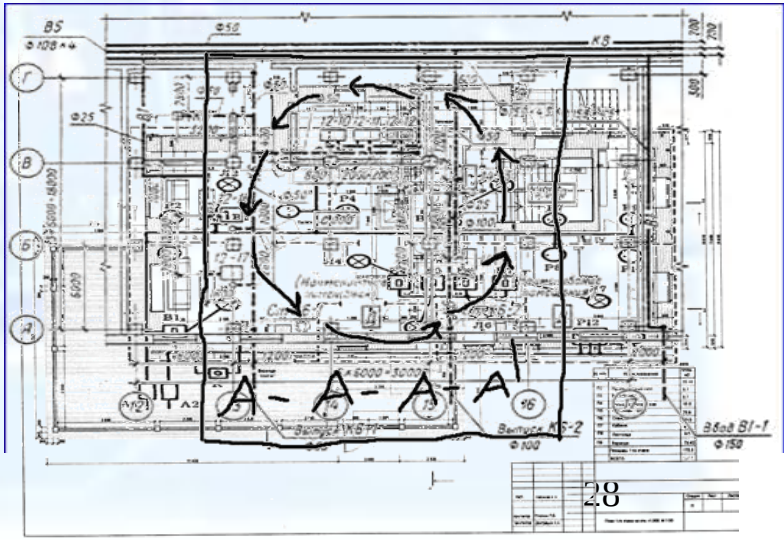


Моделирование

- 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**

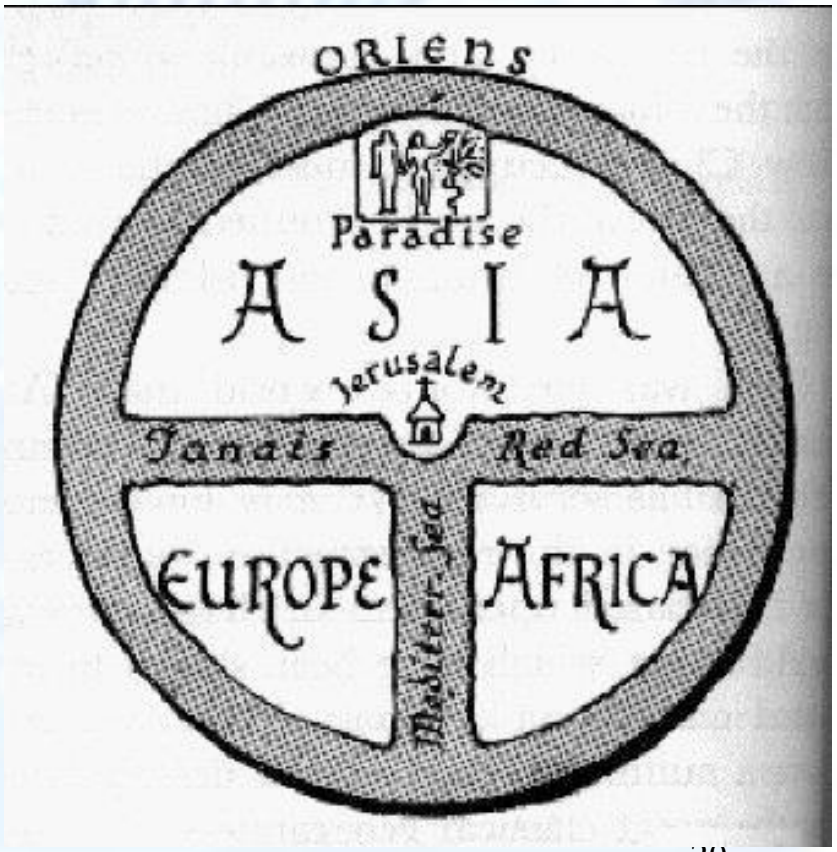
  - **Адекватной.**

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

 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:

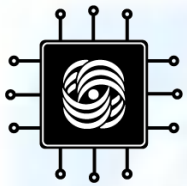
- msgsnd всегда вызывается только после msgget;
- если поток управления попадает в msgget_error, то после этого не вызываются функции msgsnd и msgrcv;
- ни в одном вычислении не встречаются msgrcv без msgsnd между ними.

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:

не выполняется!

```
#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();
    }
  }
}
```

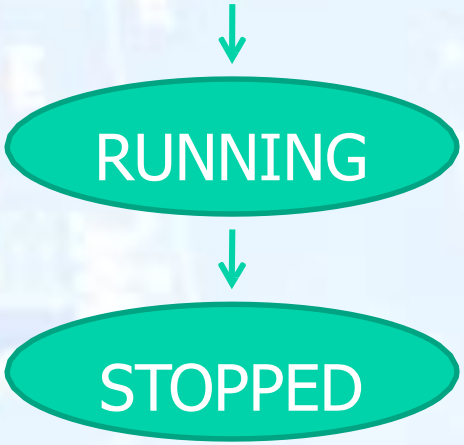
подходит и для проверки свойства №3

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!