

# Математические методы верификации схем и программ

Инструкция по использованию средства SPIN  
для верификации LTL-требований

Захаров В.А.  
Подымов В.В.

Все вопросы писать на почту

[valdus@yandex.ru](mailto:valdus@yandex.ru)

# Windows vs. Linux

Spin – мультиплатформенное средство

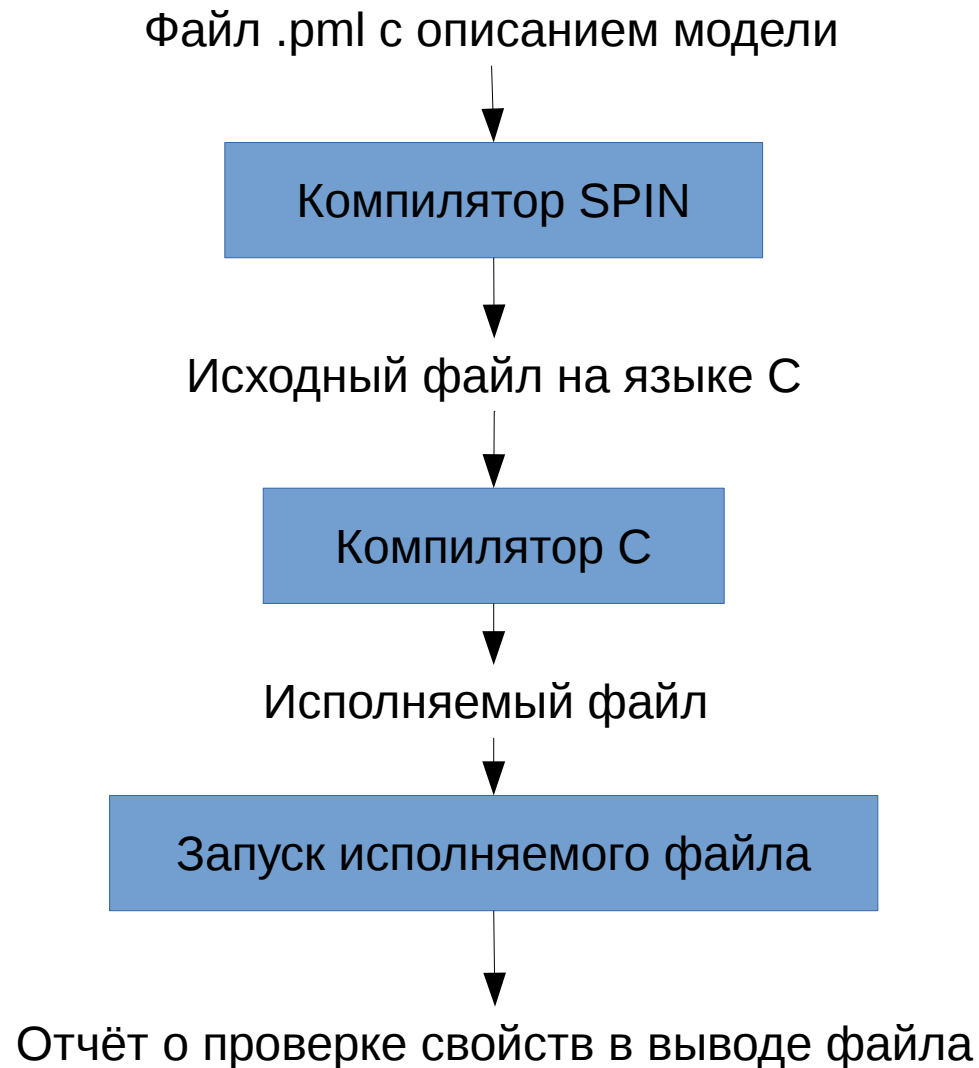
И в Windows, и в Linux доступны как консольная утилита, так и GUI Spin

Данная инструкция ориентирована на пользователей **Linux**

*Вроде бы* в Windows всё делается аналогично,  
однако детали запуска средства, само собой, могут отличаться

# Общая схема работы SPIN

Работа средства SPIN для проверки LTL-свойств в общем виде выглядит так:



# Консоль + Linux

Файл .pml с описанием модели

```
terminal> ls
ispin.tcl spin spin643_linux64 test.pml
terminal> cat test.pml
int a;
active proctype P() {
do
:: a < 3 -> a = a + 1;
:: a = 3 -> a = 0;
od
}
ltl f {[ ] (0 <= a & a <= 2)}
terminal>
```

# Консоль + Linux

Файл .pml с описанием модели



Компилятор SPIN



Исходный файл на языке C

“-a” - создать файл “pan.c” с исходным кодом верификатора данной конкретной модели

По умолчанию создаётся верификатор в режиме проверки свойств живости

```
terminal> ./spin -a test.pml
ltl f: [] (((0<=a)&(a<=2)))
terminal> ls pan.c
pan.c
terminal>
```

Если уверены, что режима проверки свойств безопасности будет достаточно, то флаг “DSAFETY” переключает верификатор на этот режим

*(свойства безопасности  
проверяются быстрее с DSAFETY)*

```
terminal> ./spin -a -DSAFETY test.pml
ltl f: [] (((0<=a)&(a<=2)))
terminal> ls pan.c
pan.c
terminal>
```

# Консоль + Linux

Исходный файл на языке C



Компилятор C



Исполняемый файл

```
terminal> ls pan.c
pan.c
terminal> gcc -o pan pan.c
terminal> ls pan
pan
terminal> 
```

# Консоль + Linux

Исполняемый файл → **Запуск исполняемого файла** → Отчёт о проверке свойств в выводе файла

“-N <имя>” - проверить свойство, описанное как “ltl <имя>”

Без флага “-a” рассматриваются только **конечные** трассы

**Свойство не выполнено**

Проверяем свойство f

Про дополнительные опции читайте в документации

Статистика

```
terminal> ./pan -N f
warning: only one claim defined, -N ignored
warning: never claim + accept labels requires -a flag to fully verify
hint: this search is more efficient if pan.c is compiled -DSAFETY
pan:1: assertion violated  !((!(((0<=a)&(a<=2)))) (at depth 12)
pan: wrote test.pml.trail

(Spin Version 6.4.3 -- 16 December 2014)
Warning: Search not completed
        + Partial Order Reduction

Full statespace search for:
{ never claim           + (f)
{ assertion violations   + (if within scope of claim)
{ acceptance cycles     - (not selected)
{ invalid end states     - (disabled by never claim)

State-vector 28 byte, depth reached 12, errors: 1
    7 states, stored
    0 states, matched
    7 transitions (= stored+matched)
    0 atomic steps
hash conflicts:          0 (resolved)

Stats on memory usage (in Megabytes):
    0.000    equivalent memory usage for states (stored*(State-vector + overhead))
    0.291    actual memory usage for states
   128.000    memory used for hash table (-w24)
    0.534    memory used for DFS stack (-m10000)
   128.730    total actual memory usage

pan: elapsed time 0 seconds
terminal>
```

# Консоль + Linux



Попробуем проверить такое свойство

```
terminal> ls
ispin.tcl spin spin643_linux64 test.pml
terminal> cat test.pml
int a;
active proctype P() {
do
:: a < 3 -> a = a + 1;
:: a = 3 -> a = 0;
od
}
ltl f {![[] <> (a == 3)}
terminal> ./spin -a test.pml
ltl f: ! ([[] (<> ((a==3))))
terminal> gcc -o pan pan.c
```

Его можно опровергнуть только **бесконечной** трассой



# Консоль + Linux



```
terminal> ./pan
warning: never claim + accept labels requires -a flag to fully verify
hint: this search is more efficient if pan.c is compiled -DSAFETY

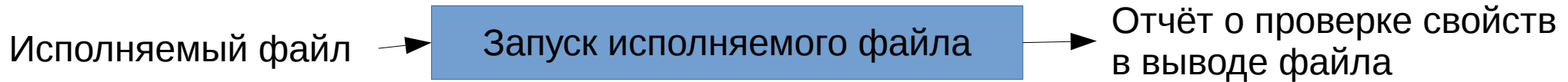
(Spin Version 6.4.3 -- 16 December 2014)
    + Partial Order Reduction

Full statespace search for:
    never claim                + (f)
    assertion violations      + (if within scope of claim)
    acceptance cycles        - (not selected)
    invalid end states        - (disabled by never claim)

State-vector 28 byte, depth reached 17, errors: 0
    10 states, stored
    7 states, matched
    17 transitions (= stored+matched)
    0 atomic steps
hash conflicts:                0 (resolved)
```

Просмотрев все **конечные** трассы, верификатор не нашёл противоречий со свойством

# Консоль + Linux



При верификации смотрим  
и на бесконечные трассы

“acceptance cycle” =  
“свойство нарушено”

И здесь в подтверждение  
говорится “была ошибка”

```
terminal> ./pan -a
pan:1: acceptance cycle (at depth 0)
pan: wrote test.pml.trail

(Spin Version 6.4.3 -- 16 December 2014)
Warning: Search not completed
        + Partial Order Reduction

Full statespace search for:
      never claim                + (f)
      assertion violations        + (if within scope of claim)
      acceptance cycles          + (fairness disabled)
      invalid end states         - (disabled by never claim)

State-vector 28 byte, depth reached 15, errors: 1
      8 states, stored (9 visited)
      1 states, matched
      10 transitions (= visited+matched)
      0 atomic steps
hash conflicts:                  0 (resolved)
```

# Консоль + Linux

## Резюме

Для полной уверенности в результате верификации делаем так:

**spin -a <имя файла с моделью>**

(без флага -DSAFETY, чтобы иметь всевозможные трассы)

**gcc -o ran ran.c**

**./ran -a -N <имя свойства>**

(с флагом -a, чтобы учитывать бесконечные трассы при проверке)

# GUI + Linux

Есть как минимум две популярных графических оболочки для SPIN

## ispin

Это оболочка от разработчиков spin, она идёт в комплекте со средством

Оболочка основана на **tcl**

Она выглядит старомодно, но в ней есть всё, что нужно

О ней будут следующие слайды

## jspin

<http://spinroot.com/spin/Man/README.html#S3>

Это оболочка **не** от разработчиков spin

Оболочка основана на java

Она выглядит более современно, и *вроде бы* в ней тоже есть всё, что нужно

В ней всё делается примерно так же, как в ispin, так что о ней слайдов не будет

# ispin + Linux

Чтобы оболочка ispin работала, необходимо сделать так, чтобы терминал понимал команду “spin”:

- поставить spin в стандартные системные папки, или
- прописать в нужную переменную окружения путь к spin

Оболочка запускается исполняемым файлом **ispin.tcl**

```
terminal> ls
ispin.tcl  spin  spin643_linux64  test.pml  test.pml.trail
terminal> spin
If 'spin' is not a typo you can use command-not-found to lookup the package that contains it, like this:
cnf spin
terminal> export PATH=$PATH:$PWD
terminal> ispin.tcl
```

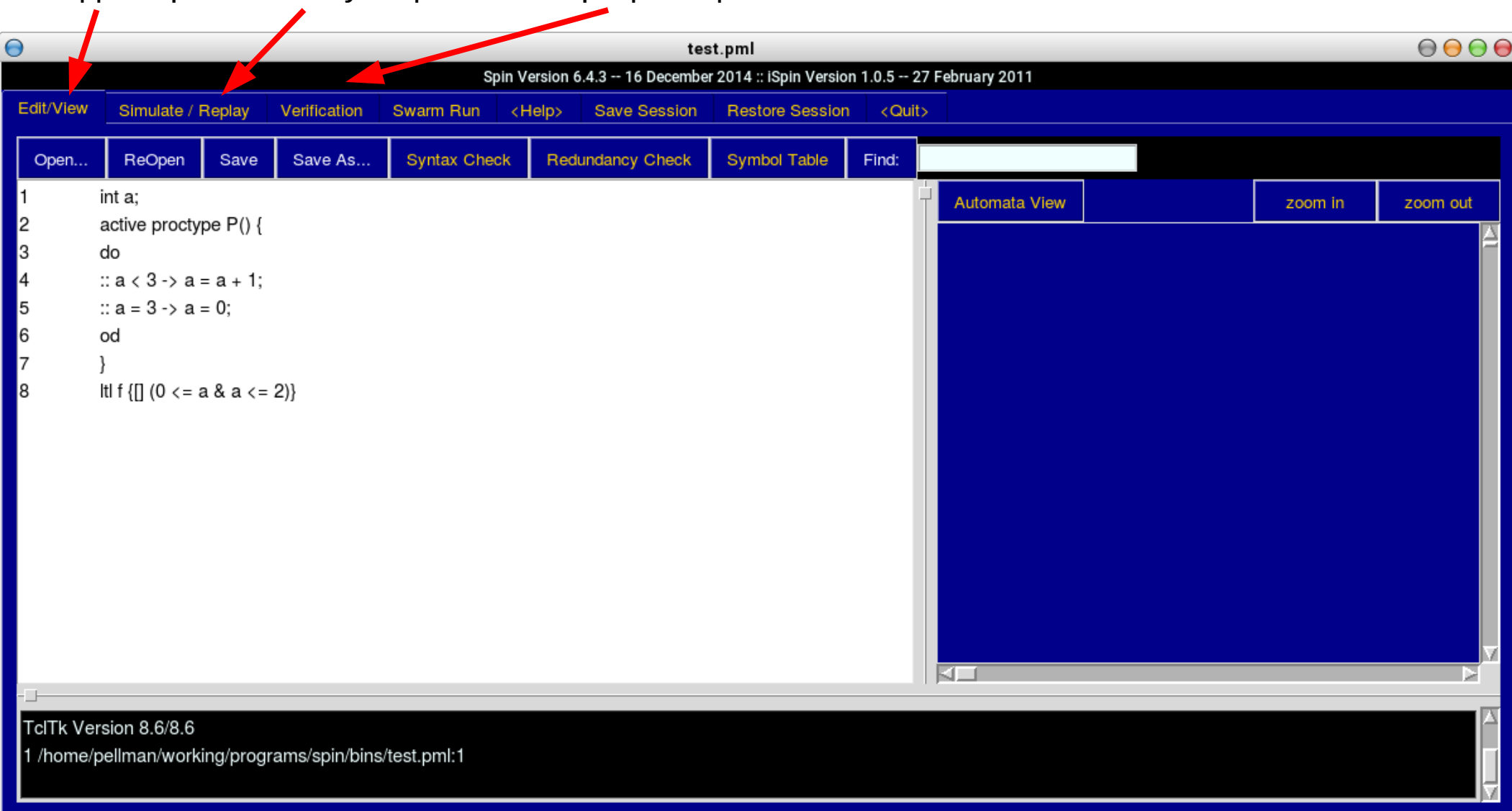
# ispin

Всё довольно user-friendly

Редактор

Симуляция

Верификация



# ispin

Режим свойств безопасности

Проверить ltl-свойство

Запустить проверку

Рекомендованный режим свойств живости

Имя свойства

Отчёт

The screenshot shows the Ispin software interface with the following components and annotations:

- Menu Bar:** Edit/View, Simulate / Replay, Verification, Swarm Run, <Help>, Save Session, Restore Session, <Quit>.
- Safety Section:**
  - ☐ safety
  - ☒ + invalid endstates (deadlock)
  - ☒ + assertion violations
  - ☐ + xr/xs assertions
- Liveness Section:**
  - ☐ non-progress cycles
  - ☒ acceptance cycles
  - ☐ enforce weak fairness constraint
- Storage Mode Section:**
  - ☒ exhaustive
    - ☐ + minimized automata (slow)
    - ☐ + collapse compression
  - ☐ hash-compact
  - ☐ bitstate/supertrace
- Never Claims Section:**
  - ☐ do not use a never claim on ltl property
  - ☒ use claim
- Search Mode Section:**
  - ☒ depth-first search
    - ☒ + partial order reduction
    - ☐ + bounded context switching
    - with bound: 0
    - ☐ + iterative search for short trail
  - ☐ breadth-first search
    - ☒ + partial order reduction
  - ☒ report unreachable code
- Buttons:** Run, Stop
- claim name (opt):**
- Save Result in:**
- Buttons:** Show Error Trapping Options, Show Advanced Parameter Settings
- Code Editor:**

```
1 int a;  
2 active proctype P() {  
3   do  
4     :: a < 3 -> a = a + 1;  
5     :: a = 3 -> a = 0;  
6   od  
7 }  
8 ltl f {} (0 <= a & a <= 2)}
```
- Output Window:**

```
Stats on memory usage (in Megabytes):  
0.000 equivalent memory usage for states (stored*(State-vector + overhead))  
0.291 actual memory usage for states  
128.000 memory used for hash table (-w24)  
0.534 memory used for DFS stack (-m10000)  
128.730 total actual memory usage  
  
pan: elapsed time 0 seconds  
To replay the error-trail, goto Simulate/Replay and select "Run"
```