Инструмент SPIN

Занятие №3

Интенсивные исследования в области model checking привели к тому, что разработанная к настоящему времени техника применения этого подхода для верификации намного проще, чем его теоретический базис.

Ю.Г. Карпов.
Model Checking. Верификация параллельных и распределенных программных систем

Александр Сергеевич Камкин

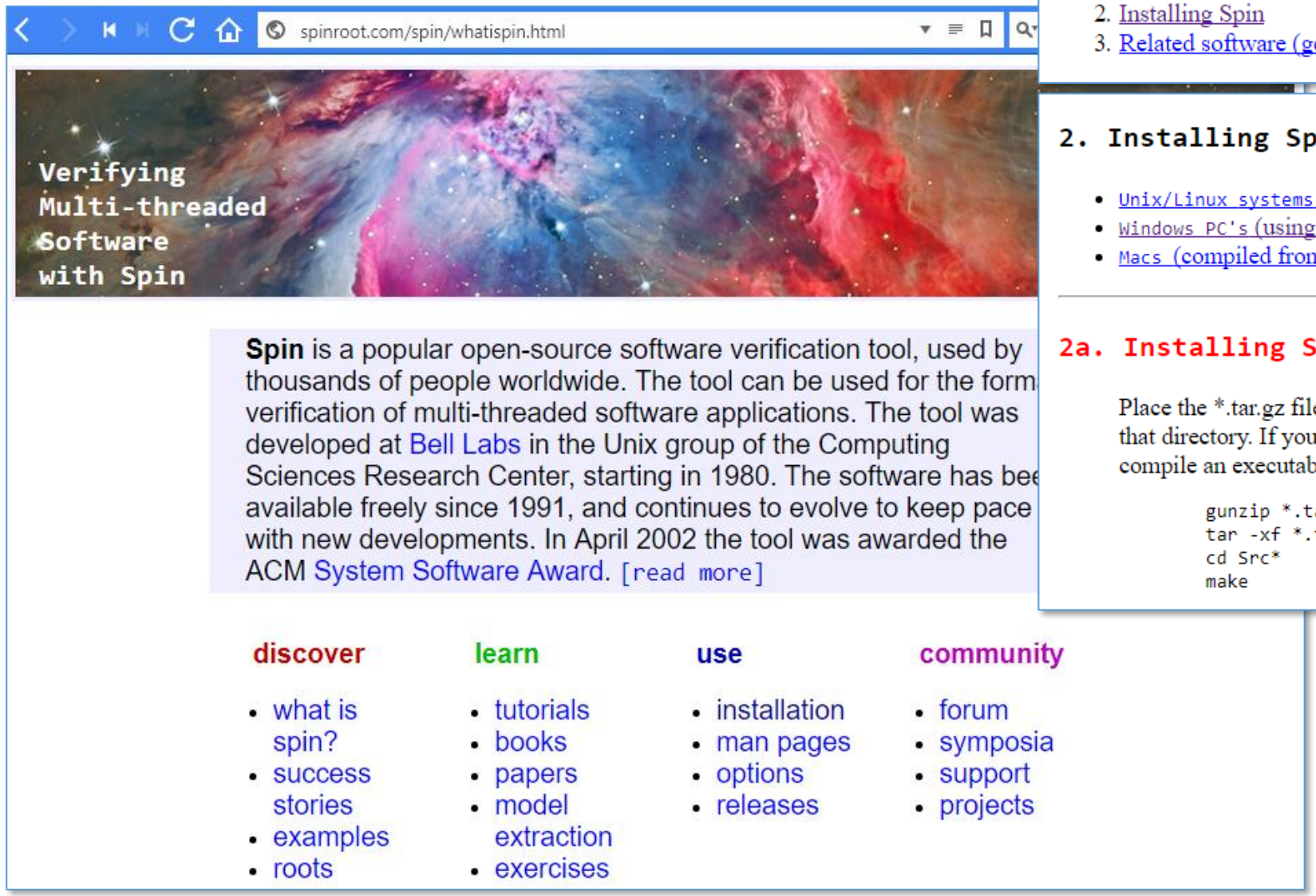
kamkin@ispras.ru

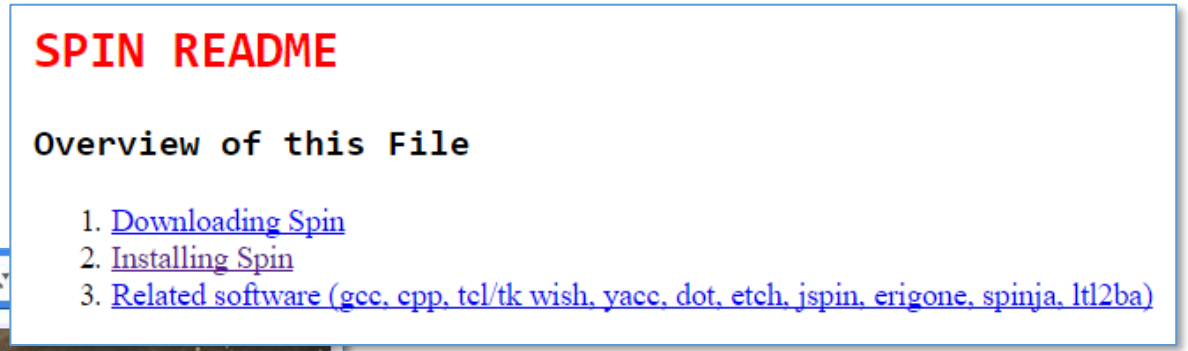
Язык PROMELA и инструмент SPIN

- **Джерард Хольцман (Gerard J. Holzmann)**
  - Computer Sciences Research Center, Bell Labs
  - SPIN (1980-ые гг., open source с 1991 г.)
- **PROMELA (Process Meta-Language)**
  - Моделирование параллельных систем
  - Спецификация требований (логика LTL)
- **SPIN (Simple PROMELA Interpreter)**
  - Рандомизированная симуляция моделей
  - Формальная проверка моделей (model checking)

Gerard J. Holzmann

Установка Spin

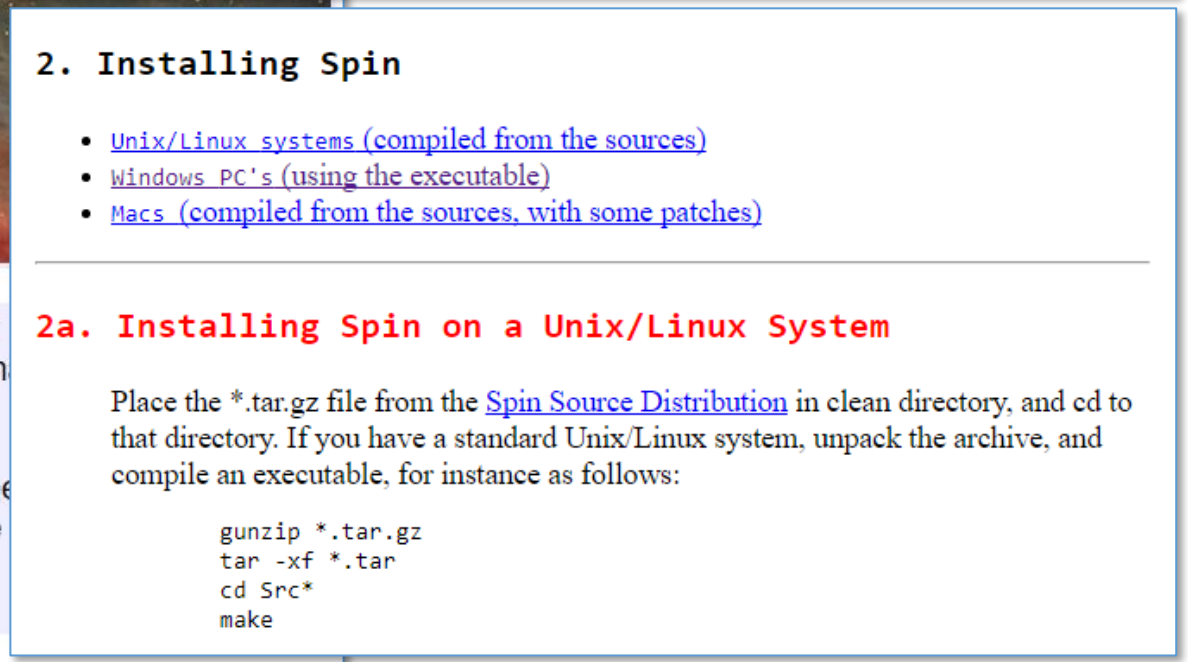




SPIN README

Overview of this File

1. Downloading Spin
2. Installing Spin
3. Related software (gcc, cpp, tcl/tk wish, yacc, dot, etch, jspin, erigone, spinja, ltl2ba)



2. Installing Spin

- Unix/Linux systems (compiled from the sources)
- Windows PC's (using the executable)
- Macs (compiled from the sources, with some patches)

2a. Installing Spin on a Unix/Linux System

Place the *.tar.gz file from the Spin Source Distribution in clean directory, and cd to that directory. If you have a standard Unix/Linux system, unpack the archive, and compile an executable, for instance as follows:

```
gunzip *.tar.gz
tar -xf *.tar
cd Src*
make
```

http://spinroot.com

Installation

Installing Spin

Unix/Linux

Windows

Пример: Hello World!

```
active proctype main() {
   /* комментарии пишутся так же,
      как в языке программирования C */
   /* функция печати аналогична
      функции printf в C */
   printf("Hello World: %d\n", _pid)
}
```

Пример: алгоритм Петерсона (while)

```
while true do
   NCS_i: /* некритическая секция */
   SET_i: flag_i := 1; turn := i;
   TST_i: while (flag_1-i = 1) ∧ (turn = i) do
             skip /* ожидание */
          end;
   CRS_i: /* критическая секция */
   RST_i: flag_i := 0
end
```

Пример: алгоритм Петерсона (PROMELA)

```
bit flag[2] = 0;
int turn = 0;

active [2] proctype P() {
  int i = _pid;
  assert(0 <= i && i <= 1);
  NCS: skip; /* некритическая секция */
  SET: flag[i] = 1; turn = i;
  TST: !(flag[1 - i] == 1 && turn == i); /* ожидание */
  CRS: skip; /* критическая секция */
  RST: flag[i] = 0;
  goto NCS
}
```

Пример: спецификация свойств

```
ltl safety {
  []!(P[0]@CRS && P[1]@CRS)
}

ltl liveness {
  [](P[0]@SET -> <>P[0]@CRS)
}
```

Симуляция и верификация

- **Рандомизированная симуляция модели**
  - `spin [-p] [-usteps] file-name`
- **Генерация, компиляция и запуск программы проверки**
  - `spin -a file-name`
  - `gcc –o pan pan.c`
  - `./pan`
- **Воспроизведение контрпримера**
  - `spin -t file-name`

Опции программы проверки (pan)

Опция	Описание
`-a`	включает поиск допускающих циклов (нужно для проверки свойств живости)
`-f`	включает предположение о справедливости планировщика
`-i`	включает поиск кратчайшего контрпримера
`-m`*`глубина`*	устанавливает максимальную глубину поиска в пространстве состояний модели
`-N` *`клауза`*	задает имя проверяемой клаузы (**never**- или **ltl**-блока)

Базовые типы данных

Тип данных	Множество значений	Битовая длина
bit	$\{0, 1\}$	1
bool	$\{\mathbf{false}, \mathbf{true}\}$	1
byte	$[0, 255]$	8
short	$[-2^{15}, 2^{15} - 1]$	16
int	$[-2^{31}, 2^{31} - 1]$	32
unsigned: n	$[0, 2^n - 1]$	$n \leq 32$

Декларация переменных и массивов

```
bit x, y;                   // 1 бит, значение 0
bool turn = true;           // 1 бит, значение 1
unsigned v:5;               // 5 бит, значение 0
unsigned w:3 = 5;           // 3 бита, значение 5

byte a[12];                 // a[0]==...==a[11]==0
short b[2] = {-1, 1}; // b[0]==-1, b[1]==1
int c, d[3] = 1;          // c==0, d[0]==d[1]==d[2]==1
```

Тип `mtype` (message type)

```
// константы для типов сообщений
mtype = { req, ack, nack, err };

// несколько mtype
mtype = { a1, ..., an };
mtype = { b1, ..., bm };

// эквивалентны одному
mtype = { a1, ..., an, b1, ..., bm };
```

Структурные типы

```
// определение структурного типа Packet
typedef Packet {
  mtype type;     // тип пакета
  short source;   // адрес отправления
  short target;   // адрес назначения
  byte  data[4]   // передаваемые данные
};

// декларация переменной структурного типа
Packet p;

// использование переменной структурного типа
p.type = req;
p.target = 1;
```

Операции: приоритет и ассоциативность

Операции	Ассоциативность	Комментарий
`()  []  .`	слева направо	Скобки, индексирование массива, доступ к полю структуры
`!  ~  ++  --`	справа налево	Отрицание, побитовое отрицание, инкремент, декремент
`*  /  %`	слева направо	Умножение, деление, остаток от деления
`+  -`	слева направо	Сложение, вычитание
`<<  >>`	слева направо	Сдвиг влево, сдвиг вправо
`<  <=  >  >=`	слева направо	Сравнение на меньше / больше
`==  !=`	слева направо	Сравнение на равенство / неравенство
`&`	слева направо	Побитовое И
`^`	слева направо	Побитовое исключающее ИЛИ
`|`	слева направо	Побитовое ИЛИ
`&&`	слева направо	Логическое И (конъюнкция)
`||`	слева направо	Логическое ИЛИ (дизъюнкция)
`->  :`	справа налево	Условная операция
`=`	справа налево	Присваивание

Процессные типы и процессы

```
// определение процессного типа
proctype P(int arg1, int arg2) { ... }

// автоматическое создание процессов
active proctype Q() { ... }
active [10] proctype R() { ... }

// процесс инициализации (точка входа)
init { run P(1, 2) }
```

Оператор выбора `if - fi`

```
if
:: guard_1 -> action_1
...
:: guard_n -> action_n
fi
```

- Guard – любой оператор, **`else`** или **`timeout`**
- Action – последовательность операторов
- Действие разрешено, если защита исполнима

Оператор повторения do – od

```
do
:: guard_1 -> action_1
...
:: guard_n -> action_n
od
```

- Guard – любой оператор, **`else`** или **`timeout`**
- Action – последовательность операторов
- Выход из цикла: **`break`** или **`goto`**

Разминка: алгоритм Евклида

```
x := a;
y := b;
while x ≠ y do
 if x > y then
  x := x - y
 else
  y := y - x
 end
end;
c := x
```

Синтаксический сахар: `for`

Цикл do-od	Цикл for

`T i = a;`
`do`
`:: i <= b -> P;`
`i++`
`:: else -> break`
`od`
	`T i;`
	`for (i : a..b) {`
	`P`
	`}`

Синтаксический сахар: inline

```
// определение макроса
inline M(arg_1, ..., arg_n) {
  ... // фрагмент кода,
      // использующий arg_i
}

// использование макроса
M(expr_1, ..., expr_n)
```

Средства спецификации требований

- **Утверждения**
  - **`assert`**(*`expression`*)
- **Never-клаузы**
  - *Представляют автоматы Бюхи*
- **LTL-клаузы**
  - **`ltl`** *`optional-name { formula }`*

Проверка свойства $\mathbf{G}p$ ($\mathbf{never}\ \mathbf{F}\neg p$)

```
never {
   do
   :: !p -> break
   :: else
   od
}
```

Формулы LTL

Операция	Комментарий
[] или always	темпоральный оператор **G** (Globally)
<> или eventually	темпоральный оператор **F** (in the Future)
!	отрицание
U	темпоральный оператор **U** (Until)
V	темпоральный оператор двойственный к **U**: $\varphi \, \mathrm{V} \, \psi \equiv ! \, (! \, \varphi \, \mathrm{U} \, ! \, \psi)$
&&	конъюнкция
||	дизъюнкция
–>	импликация
<–>	эквивалентность

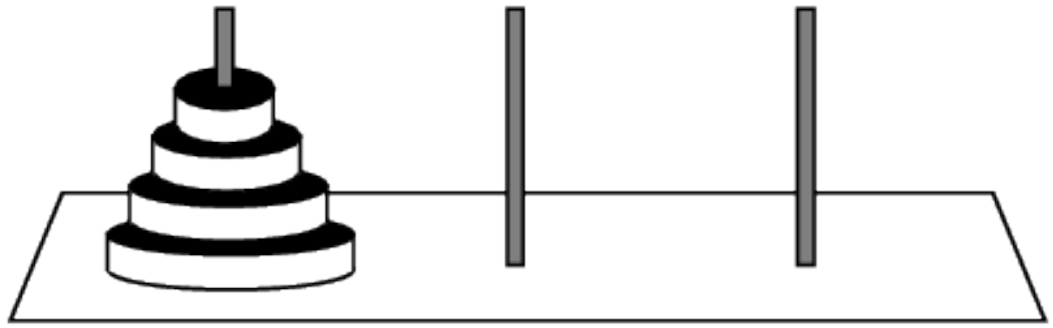
Решение головоломок с помощью SPIN

Найти **последовательность шагов** для достижения некоторой **цели**

$$\neg \mathbf{F}\{goal\} \text{ или } \neg \{rules\}\mathbf{U}\{goal\}$$

$$\downarrow$$

$$\text{Контрпример} = \text{Решение}$$



http://mathworld.wolfram.com

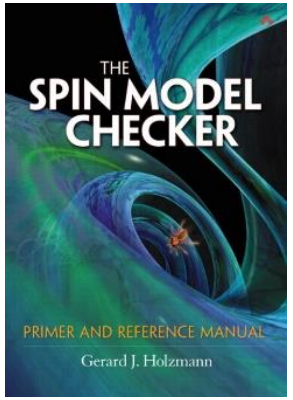
Пример: волк, коза и капуста

- Крестьянину нужно перевезти через реку волка, козу и капусту
- В лодке может поместиться крестьянин, а с ним либо волк, либо коза, либо капуста
- Если оставить волка с козой на берегу, волк съест козу
- Если оставить козу с капустой, коза съест капусту
- Только когда крестьянин рядом, коза и капуста в безопасности

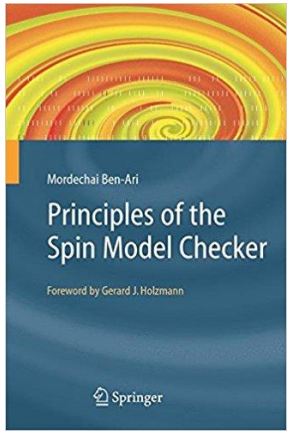
https://illuminations.nctm.org

Как крестьянину перевезти свой груз?

Что почитать: PROMELA и SPIN



Gerard J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, 2003. 608 p.



Mordechai Ben-Ari.
Principles of the Spin Model Checker.
Springer, 2008. 220 p.