Инструмент SPIN (продолжение)

Занятие №4

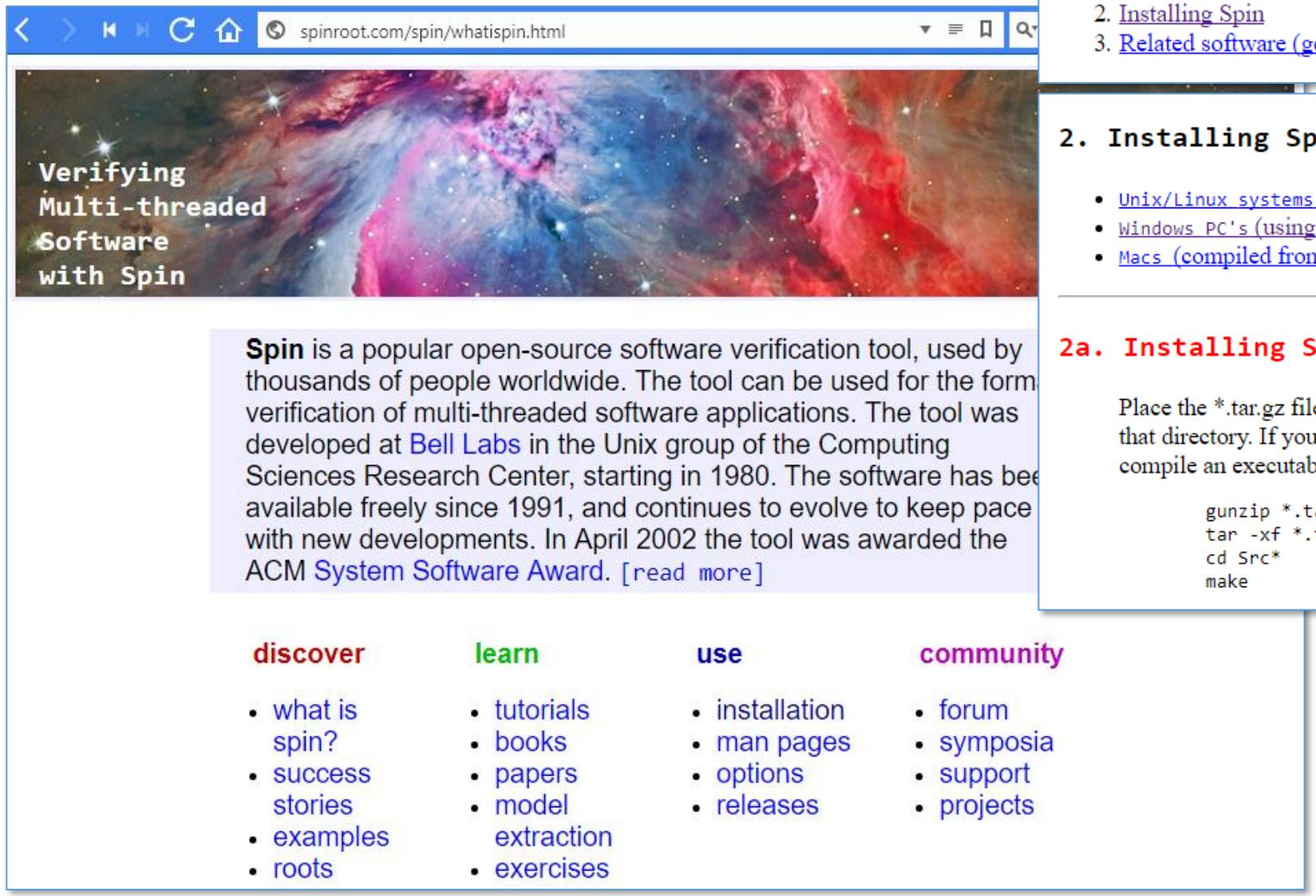
Интенсивные исследования в области model checking привели к тому, что разработанная к настоящему времени техника применения этого подхода для верификации намного проще, чем его теоретический базис.

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

Александр Сергеевич Камкин

kamkin@ispras.ru

Скоро будет поздно…



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

Атомарные действия

```
init:  /* инициализация мьютекса */
  count = 1
...
enter: /* захват мьютекса */
  do
  :: atomic { count > 0 -> count-- }; break
  :: else
  od
...
leave: /* освобождение мьютекса */
  count++
```

Разминка: критическая секция

Каналы передачи сообщений

chan *c*; *// неинициализированный канал*

chan *c* = [*N*] **of** {T_1, ..., T_n};

- c — имя канала
- N — емкость (размер буфера)
  - если $N = 0$, канал является *синхронным*
- $T_1 \times \cdots \times T_n$ — тип передаваемых сообщений
  - сообщения имеют вид $t_1, \ldots, t_n$

Примеры объявлений каналов

```
// неинициализированные каналы
chan a, b, c;

// массив неинициализированных каналов
chan d[10];

// синхронный канал
chan e = [0] of {byte};

// канал, по которому передаются каналы
chan f = [16] of {chan};

// канал, по которому передаются составные сообщения
chan g = [1024] of {mtype, short};
```

Проверка состояния канала

- **len**(c) – число сообщений в буфере канала *c*
- **empty**(c) ≡ (**len**(c) == 0)
- **nempty**(c) ≡ (**len**(c) > 0)
- **full**(c) ≡ (**len**(c) == N)
- **nfull**(c) ≡ (**len**(c) < N)

Передача и прием сообщений

- `c!m` запись сообщения m в канал c
- `c?p` если первое сообщение c соответствует p, оно считывается в p и удаляется из c
- `c?<p>` сообщение считывается, но не удаляется

Сообщения и паттерны: $\texttt{item}_1\texttt{,...,item}_n$

- В сообщениях: каждый `item` – выражение
- В паттернах: значение, переменная или `_`

Что делают операторы?

```
c!1
c!ack,p
c?_
c?x
c?5
```

```
c?eval(x)
c?ack,p
c?<ack,_>
c?<_,_>
c?<_>
```

Разминка: пинг-понг процессов

Пример: Alternating Bit Protocol (ABP)

- Процесс P передает сообщения процессу Q
- Каждое сообщение содержит данные, контрольную сумму и 1-битный порядковый номер (SEQ)
- Процесс P шлет сообщение до тех пор, пока не получит подтверждения (ACK) от Q с таким же номером SEQ
- Когда это происходит, процесс P инвертирует SEQ и начинает передавать следующее сообщение
- Когда процесс Q получает сообщение, он шлет $ACK(SEQ)$ до тех пор, пока не получит сообщение со следующим номером SEQ

Моделирование состояния

```
chan m = [1] of {
  byte, // номер сообщения (SEQ): 0 или 1
  bool, // признак ошибки (CRC не моделируется)
  byte  // передаваемые данные
};

chan a = [1] of {
  byte, // номер подтверждения (SEQ): 0 или 1
  bool  // признак ошибки
};

byte sendSEQ = 0;       // текущий номер сообщения
byte sendACK = INVALID; // текущий номер подтверждения

byte sendMSG = 0;       // переданное сообщение
byte recvMSG = INVALID; // полученное сообщение
```

Моделирование действий

```
active proctype P() {
  byte ack; bool err;
  do
  :: a?ack,err ->
     if
     :: !err -> if
                :: (ack == sendSEQ) ->
                       atomic {
                         sendSEQ = 1 - sendSEQ; // следующий SEQ
                         sendMSG = (sendMSG + 1) % BUFFER_SIZE
                       }
                :: else // игнорируем: ошибка некорректный номер
                fi
     :: else // игнорируем: ошибка передачи
     fi
  :: nfull(m) -> m!sendSEQ,0,sendMSG // повторная передача
  od
}
```

Моделирование ошибок передачи

```
active proctype msgMedia() {
  byte seq, msg;
  do
  :: true -> atomic {
               m?seq,_,msg;
               printf("Loss of SEQ=%d, MSG=%d\n", seq, msg)
             }
  :: true -> atomic {
               m?seq,_,msg; m!seq,1,INVALID;
               printf("Corruption of SEQ=%d, MSG=%d\n", seq, msg)
             }
  :: true -> atomic {
               m?<seq,0,msg>; empty(m);
               printf("Transmission of SEQ=%d, MSG=%d\n", seq, msg)
             }
  od
}
```

Спецификация свойств

- Что такое надежная передача?

```
ltl property { LINK_ALIVE -> DATA_TRANSMITTED }
```

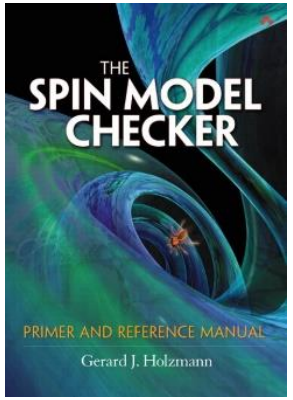
- `LINK_ALIVE`
  - Ваши варианты…
- `DATA_TRANSMITTED`
  - Ваши варианты…

```
> ./pan -a -f
```

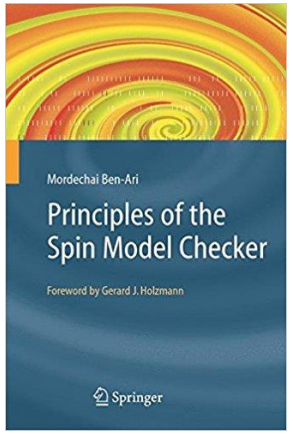
Options for checking **liveness**:

- `-a` – acceptance cycles
- `-f` – fairness (weak)

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.