Formal Method SPIN과 활용

=⊨⊑≠∀⊢⇒⊤∃¬=⊥p⇔¬=⊨⊑≠∀⊢⇒⊤∃¬=⊥p⇔⊣=⊨⊑≠∀⊢⇒⊤∃¬=⊥p⇔⊣=⊨⊑≠∀⊢⇒⊤∃¬=⊥p⇔⊣

고려대학교 정형기법 연구실
방기석
Kbang@formal.korea.ac.kr

[]<>(!f->g)

시작하기에 앞서

=⊨⊑≠∀⊢⇒⊤∃¬=⊥p⇔¬=⊨⊑≠∀⊢⇒⊤∃¬=⊥p⇔⊣=⊨⊑≠∀⊢⇒⊤∃¬=⊥p⇔⊣

- Gerard J. Holzmann
  - Bell Lab. → NASA
  - SPIN 개발
- Theo Ruys
  - Twente University
  - SPIN 튜토리얼 (ETAPS '02)

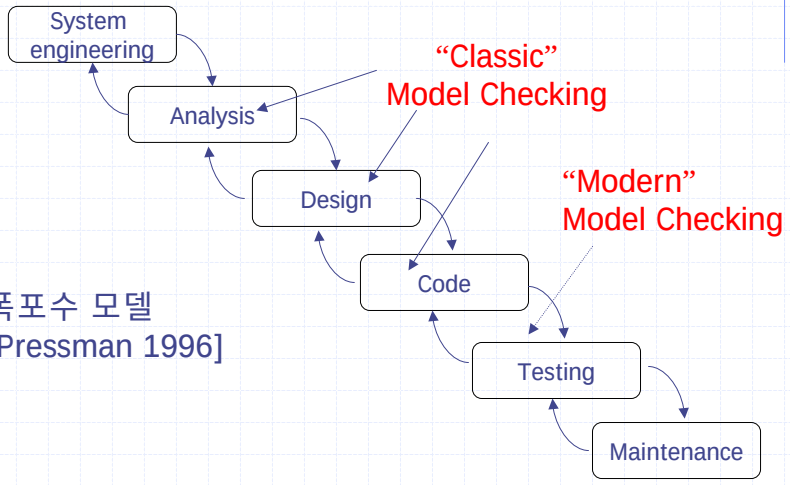
모델 체킹(Model Checking)?

=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔¬=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣

- [Clarke & Emerson 1981]:
  "Model checking이란 주어진 시스템의 유한상태 모델과 논리적 성질에 대해서, 그 특성이 모델에 대해 만족하는가를 보이는 자동화 기술이다.
- Model checking tools automatically verify whether $M \models \phi$ holds, where M is a model of a system and property ϕ is stated in some formal notation.
- 문제점 : 상태 폭발 문제가 발생
- Model Checkers
  - SPIN : awarded the prestigious System Software Award for 2001 by the ACM.
  - SMV, VIS,...

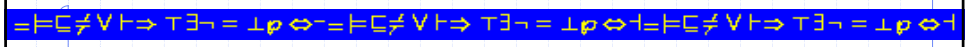
시스템의 개발 단계

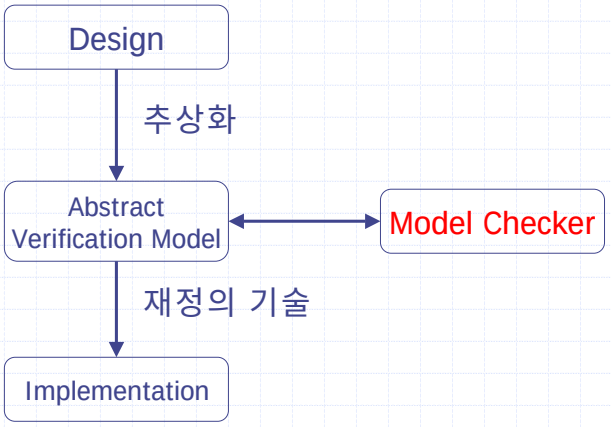
=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔¬=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣



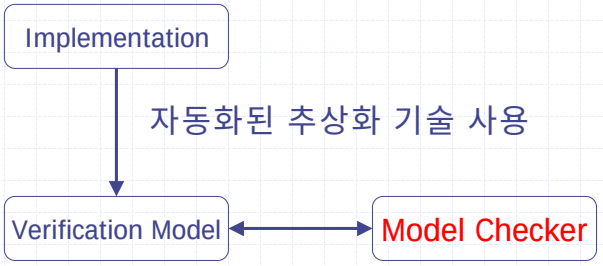
폭포수 모델
[Pressman 1996]

Classic Model Checking





Modern Model Checking



- 추상화는 두 종류의 모델 체킹에서 핵심

SPIN

=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔¬=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣

- SPIN의 소개
  - SPIN의 배경
- Promela Process들
- Promela Statement들
- Promela 통신
- Xspin
- SPIN의 활용분야

SPIN – 소개 (1)

=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔¬=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣

- SPIN = Simple Promela Interpreter
  - 데이터 통신 프로토콜과 같은 동시성 시스템의 논리적인 일관성을 분석하는 도구
  - PROMELA 라는 입력 언어를 이용해서 동시성 시스템을 명세
- Promela = PROcess/PROtocol MEta LAnguage
  - 동시성 프로세스의 동적 생성 허용
  - 메시지채널을 이용한 메시지의 동기적/비동기적 통신 모델링 가능
  - C언어와 비슷한 구조로 이해가 용이
  - 유한 상태 시스템의 명세를 위한 언어

SPIN – 소개 (2)

=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔¬=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣

- Version

1.0	1991.1	초기 버전 (Holzmann)
2.0	1995.1	Partial Order Reduction
3.0	1997.4	오토마타의 최소화
4.0	2002 이후	Ax : C코드로부터의 Automata 자동추출

- 성공 요인
  - 모델 체커
  - C 코드와 비슷한 구조
  - Xspin을 통한 그래픽 환경
  - 최적화된 알고리즘에 의한 검증 기술의 발전

Promela Model

=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔¬=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣

- 구성
  - 타입 선언
  - 채널 선언
  - 변수 선언
  - 프로세스 선언
  - [init process]
- 유한 상태 시스템 명세
  - 모든 데이터, 채널,프로세스들, 프로세스의 생성은 유한하다.

```
mtype = {MSG, ACK};

chan toS = [2] of {mtype, bit};
chan toR = [2] of {mtype, bit};

proctype Sender(chan in; chan out)
{
                        … Process body
}
proctype Receiver(chan in; chan out)
{
}

Init {
…   Creates Processes
}
```

Process (1)

=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔¬=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣

- Process type(proctype) 의 구성
  - 프로세스 이름
  - Parameter들의 목록
  - 지역 변수 선언
  - 본문

이름

Parameter목록

```
proctype Sender(chan in; chan out)
{
    bit sendbit, recvbit;
    do
    ::  out ! MSG, sendbit ->
        in ? ACK, recvbit;
        if
        ::  recvbit == sendbit
                -> sendbit = 1-
    sendbit
        ::  else -> skip
        fi
    od
}
```


Process(2)

=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔¬=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣

- 프로세스
  - “proctype” 키워드로 정의
  - 모든 프로세스들은 동시 수행
  - 다른 프로세스들간의 통신
    - 전역변수
    - 메시지 채널
  - **“run”** 에 의해서 수행
  - 수행에서의 임의의 시점에 생성
  - **“active”** 를 사용하면 자동으로 생성 가능

Hello World!

=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔¬=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣

```
active proctype Hello() {
    printf("Hello process, my pid is: %d\n", _pid);
}

init {
    int lastpid;
    printf("init process, my pid is: %d\n", _pid);
    lastpid = run Hello();
    printf("last pid was: %d\n", lastpid);
}
```


Hello World! 결과

=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔¬=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣

```
/user/ksbang/spin/tutorial]spin hello.pr
        init process, my pid is: 1
        last pid was: 2
Hello process, my pid is: 0
                Hello process, my pid is: 2
3 processes created
```

변수와 타입(1)

=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔¬=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣

- 기본형
  - Bit [0...1]
  - Bool [0...1]
  - Byte [0...255]
  - Short [-2^{16}-1...2^{16}-1]
  - Int [-2^{32}-1...2^{32}-1]
- 배열
- Records(Structs)
- 타입오류는 수행시점에 검사가능
- 초기값은 0

[]<>(!f->g)

변수와 타입 (2)

=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔¬=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣

- 변수는 반드시 선언되어야 한다.
- 변수는 다음의 방법으로 값을 줄 수 있다.
  - Assignment
  - Argument passing
  - Message passing
- 변수는 기본적인 수식에 사용가능하다

```
int ii;
bit bb;
bb = 1;  /* assignment */
ii = 2;

short s=-1;
/* declaration + initialisation */

typedef foo {
  bit bb;
  int ii;
}

Foo f;
f.bb = 0;
f.ii = -2;

ii*s+27 == 23;
printf("value : %d", s*s);
```

Theory and Formal Methods lab.

Statements (1)

=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔¬=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣

- 프로세스는 statement들의 나열
- Statement는 다음의 두 가지 상태가 가능
  - 수행 가능
  - 수행 불가 대기 상태
- Assignment는 항상 수행 가능
- Expression도 statement – 0이 아닌 경우에만 수행 가능
  - 2 < 3 : 언제나 수행
  - X < 27 : X가 27보다 작을 때만
  - 3 + X : X가 –3이 아니면 수행 가능

Statements (2)

=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔¬=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣

- **skip**은 항상 수행가능
  - 아무 작업도 수행하지 않는다.
- **run**은 프로세스가 생성되었을 경우 수행가능
- **printf**는 수행가능하지만 검증시에는 수행하지 않음

```
Int x;
Proctype Aap()
{
   int y=1;
   skip;
   run Noot();          /* Noot 프로세스가 생성되면 수행 */
   x=2;
   x>2 && y == 1;       /* 다른 프로세스가 x를 2보다 크게 하면 수행 */
   skip;
}
```

Assert statement

=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔¬=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣

- 항시 수행가능한 특수 statement
- assert <expression>
  - expression을 수행하여 0이 되면 SPIN은 오류를 발생하고 수행을 멈춘다.
  - Promela 모델의 어떤 상태에서 특정한 property를 만족하는지를 검사하는 statement

```
Proctype Monitor(){
        assert(n <= 3);
}
```

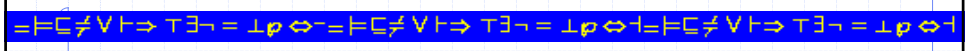

[]<>(!f->g)

Interleaving Semantics

=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔¬=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣

- Promela의 프로세스들은 동시에 수행
  - Concurrently execution
- 프로세스들은 비결정적으로 수행
  - Non-deterministically execution
- 프로세스들은 interleaving semantics를 따른다. -> 동시에 여러 문장 수행 불가
  - Rendez-vous communication 제외
- Atomic processes -> 각 문장이 다른 프로세스들의 간섭이 없이 수행

Demo – Mutual Exclusion(1)



```
bit  flag;          /* signal entering/leaving the critical section. */
byte mutex;         /* number of processes in the critical section.  */

proctype P(bit i) {
  flag != 1;
  flag  = 1;
  mutex++;
  printf("MSC: P(%d) has entered the critical section.\n", i);
  mutex--;
  flag  = 0;
}

proctype monitor() {
  assert(mutex != 2);
}

init {  atomic { run P(0); run P(1); run monitor(); } }
```

Assertion 오류
프로세스 두 개가 동시에 flag를 1로 바꿀 수 있다.

Demo–Mutual Exclusion(2)

```
bit  x, y;
byte mutex;

active proctype A() {
  x = 1;
  y == 0;
  mutex++;
  printf("MSC: A has entered the
    critical section.\n");
  mutex--;
  x = 0;
}

active proctype B() {
  y = 1;
  x == 0;
  mutex++;
  printf("MSC: B has entered the
    critical section.\n");
  mutex--;
  y = 0;
}

active proctype monitor() {
  assert(mutex != 2);
}
```

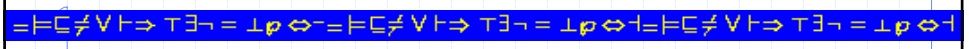
각 프로세스가 자신의 변수를 1로 바꾼 후 상대방이 다시 0으로 바꿀 때까지 기다린다.

조건문 - if-statement

```
if
:: choice1 -> stat1,1; stat1,2;...
:: choice2 -> stat2,1; stat2,2;...
:: ...
:: choicen -> statn,1; statn,2;...
fi
```

- 하나 이상의 선택이 가능하면 비결정적 수행
- 수행 가능한 선택이 없으면 block
- “->”는 “;”과 동일

if-statement(2)



- 선택될 수 있는 구문이 없는 경우 “else”를 이용하여 무조건 수행

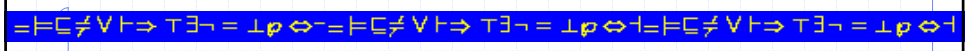
```
If
:: cond1 -> state1;
:: cond2 -> state2;
:: else -> skip;
fi
```

- 모든 선택구문을 임의로 선택하고 싶은 경우 “skip” 을 이용

```
If
:: skip -> state1;
:: skip -> state2;
:: skip -> state3;
fi
```

- “else”, “skip”은 무조건 수행구문

반복문 – do-statement



```
do
:: choice1 -> stat1,1; stat1,2;...
:: choice2 -> stat2,1; stat2,2;...
:: ...
:: choicen -> statn,1; statn,2;...
od
```

- 선택의 의미로는 if문과 동일
- 선택이 끝난 후 do문은 다시 선택구문의 처음으로부터 수행
- do문을 끝내고자 할 경우 “break”를 이용

통신 (1)

- 프로세스 간의 통신은 채널을 이용
  - 메시지 전달
  - rendez-vous 동기화(handshake)
- 채널을 선언하여 수행

Chan <name> = [<크기>] of {t_1, t_2, …, t_n}

채널이름

전송할 메시지 타입

채널로 보낼 메시지의 수

통신 (2)

=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔¬=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣

- 채널 = fifo 버퍼
- ! -> 메시지의 송신
  - Ch ! <expr$_1$>,<expr$_2$>,...<expr$_n$>;
  - 채널이 full이 아닌 경우에만 가능
- ? -> 메시지의 수신
  - Ch ? <var$_1$>,<var$_2$>,...<var$_n$>;
  - 채널이 empty가 아닌 경우에만 가능

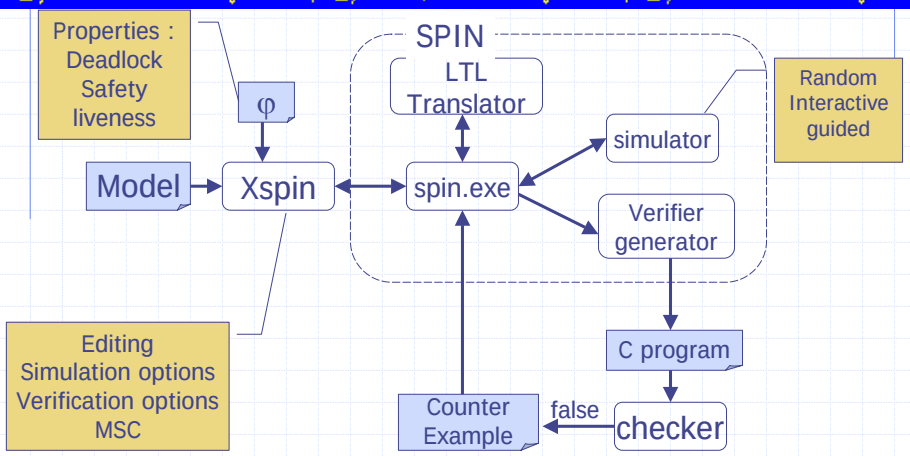
통신 (3)

=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔¬=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣

- Rendez-vous 통신
  - 버퍼의 크기가 0
  - Ch!는 ch?가 동시에 존재할 때만 수행가능
- Example

  Chan ch = [0] of {bit, byte}
  - P 는 ch!1, 3+7을 수행
  - Q 는 ch?1. x를 수행
  - 통신이 수행된 후 x의 값은? → x=10

XSpin



XSPIN에서는?

- Promela 모델링
- Simulation
  - Random
  - Interactive
  - Guided
- Verification
- 그 외
  - Slicing
  - Automata
  - LTL Property

LTL Properties (1)

=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔¬=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣

- 모델 체킹은 유한 상태 모델이 어떠한 특성을 만족하는지를 검사하는 것이다.
- SPIN은 기본적으로 다음의 특성을 검증
  - Deadlock(invalid endstates)
  - Assertions
  - Unreachable code
  - LTL formulae
  - Liveness
    - Non-progress cycles
    - Acceptance cycles

[]<>(!f->g)

LTL Properties (2)

=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔¬=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣

- Safety
  - "나쁜 일은 일어나지 않는다."
  - Invariant

  "x는 항상 5보다 작다."
  - Deadlock free
  - SPIN : "bad" 상태로 전이되는 경로를 추적
- Liveness
  - "올바른 동작을 언젠가 수행해야 한다."
  - Termination

  "시스템은 언젠가 종료할 것이다."
  - Response

  "만일 X가 발생하면 언젠가 Y가 발생할 것이다."
  - SPIN : "good"가 발생하지 않는 loop를 찾는다.

Theory and Formal Methods lab.

LTL Properties (3)

=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔¬=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣

- LTL (Linear Temporal Logic)
  - [] P Always P (언제나 P는 참이다.)
  - <> P Eventually P (언젠가 P가 참이 된다.)
  - P U Q P는 Q가 참이 될 때까지 참이다.

- 예제
  - Invariance [](p)
  - Response [](p) -> (<>(q)))
  - Precedence [](p) -> ((q) U (r)))

[]<>(!f->g)

SPIN의 활용

=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔¬=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣

- 통신 프로토콜의 설계 및 검증
- 운영체제의 IPC/스케줄링 검증
- PLC 프로그램의 검증
- 시스템 자원 활용도 검사
- 정형 명세도구로부터의 변환 후 검증
  - MOCES(Statechart → SPIN)
  - SCR (Software Cost Reduction tool)
  - MSC (Message Sequence Charts)

Papers

=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔¬=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣

- 통신 프로토콜/시스템의 검증
  - "The Agreement Problem Protocol Verification Environment"
    - J. Pascoe, R. J. Loader (The University of Reading) and V. S. Sunderam (Emory University, Atlanta)
  - "A Spin-based model checker for telecommunications protocols"
    - Vivek K. Shanbhag, and K. Gopinath, Indian Institute of Science, Bangalore, India.
  - "Verification experiments on the MASCARA protocol"
    - Guoping Jia, and Susanne Graf, Verimag, Grenoble, France.

[]<>(!f->g)

Papers

=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔¬=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔⊣

- 운영체제의 IPC/스케줄링 검증
  - "*The Model Checker Spin*"
    - IEEE Trans. on Software Engineering, Vol. 23, No. 5, May 1997, pp. 279-295.
- 시스템 자원 활용도 검사
  - "Bottleneck Analysis of a Gigabit Network Interface Card: A Formal Verification Approach"
    - H. Jin, K. Bang, C. Yoo, J. Choi
- PLC 프로그램의 검증
  - "Verification and Optimization of a PLC Control Schedule"
    - E. Brinksma and A. Mader, Twente University

Theory and Formal Methods lab.

SPIN에 관련된 문서들

=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔¬=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔┤=⊨⊏≠∀⊢⇒⊤∃¬=⊥p⇔┤

- SPIN Homepage : http://spinroot.com
  - SPIN Manual
  - Xspin의 사용법
  - 예제들
  - SPIN workshop 논문들
- Gerard J. Holzmann 홈페이지
  http://spinroot.com/gerard/
  http://eis.jpl.nasa.gov/~gh/