SMV

- Introduction -

남현우, 김성완

2017-04-03

정형기법이란?

- 수학과 논리학에 기반을 둔 방법으로 하드웨어, 소프트웨어 시스템을 명세하거나 검증하는 것

Automata

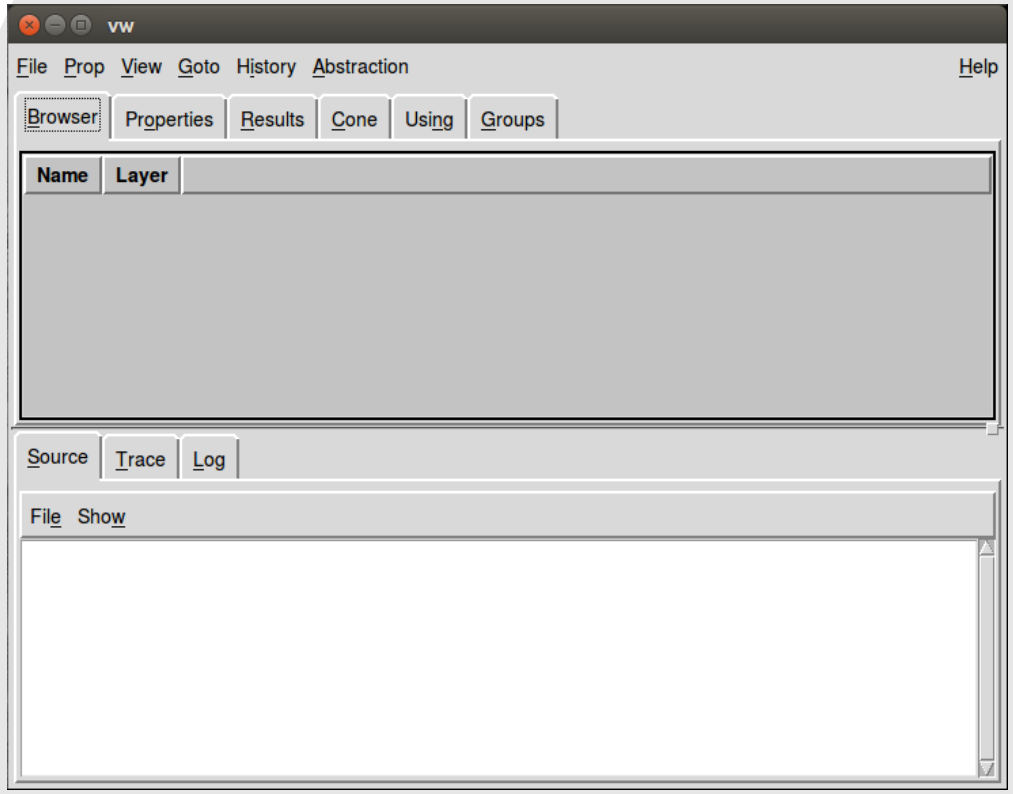
- 수학적으로 추상화된 기계의 모형을 표현
- 대상의 어떤 기능에 주목하여, 입력과 내부 출력 각 신호의 상호 관계를 수학모델로 옮기고, 이 모델을 수학적으로 고찰 ·결론을 유도한다.
- 오토마타는 컴퓨터 구조 설계와 컴파일러 설계, 파싱, 정형 모델의 정형 검증 등의 중요한 요소

Model checking tools 비교

Name	Modeling language	Properties language	example generation	GUI	Graphical Specification	Counter Example visualization	Platform / OS
Cadence SMV	Cadence SMV, SMV, Verilog	CTL, LTL	Yes	Yes	No	No	Windows and Unix related
SPIN	Promela	LTL	Yes	Yes	No	Yes	Windows and Unix related
UPPAAL	Timed automata, C subset	TCTL subset	Yes	Yes	Yes	Yes	Mac OS, Windows, Linux

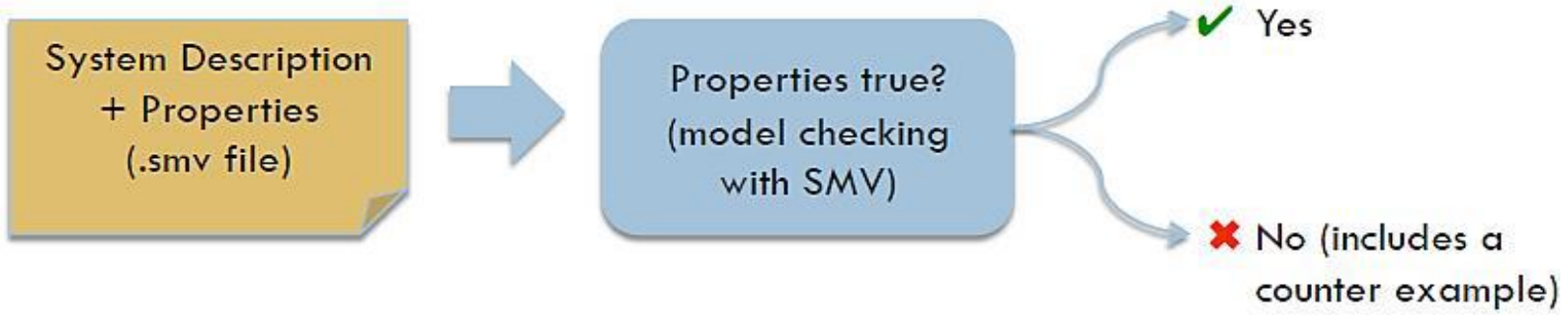
SMV

- SMV는 Symbolic Model Verifier의 약자로 유한 상태 시스템(finite state system)이 CTL(Computational tree logic)로 표현된 요구 명세를 만족하는지를 검증하는 정형 검증 도구



SMV

- SMV : 시스템 설명과 특성이 True인지 SMV로 확인



- Cadence SMV
  - Based on Ken McMillan's work at CMU
  - Can do compositional verification
  - More expressive mode description language
    (.smv(2 types) , .v(Verilog))
  - Easy-to-use graphical user interface

CTL(Computation Tree Logic)

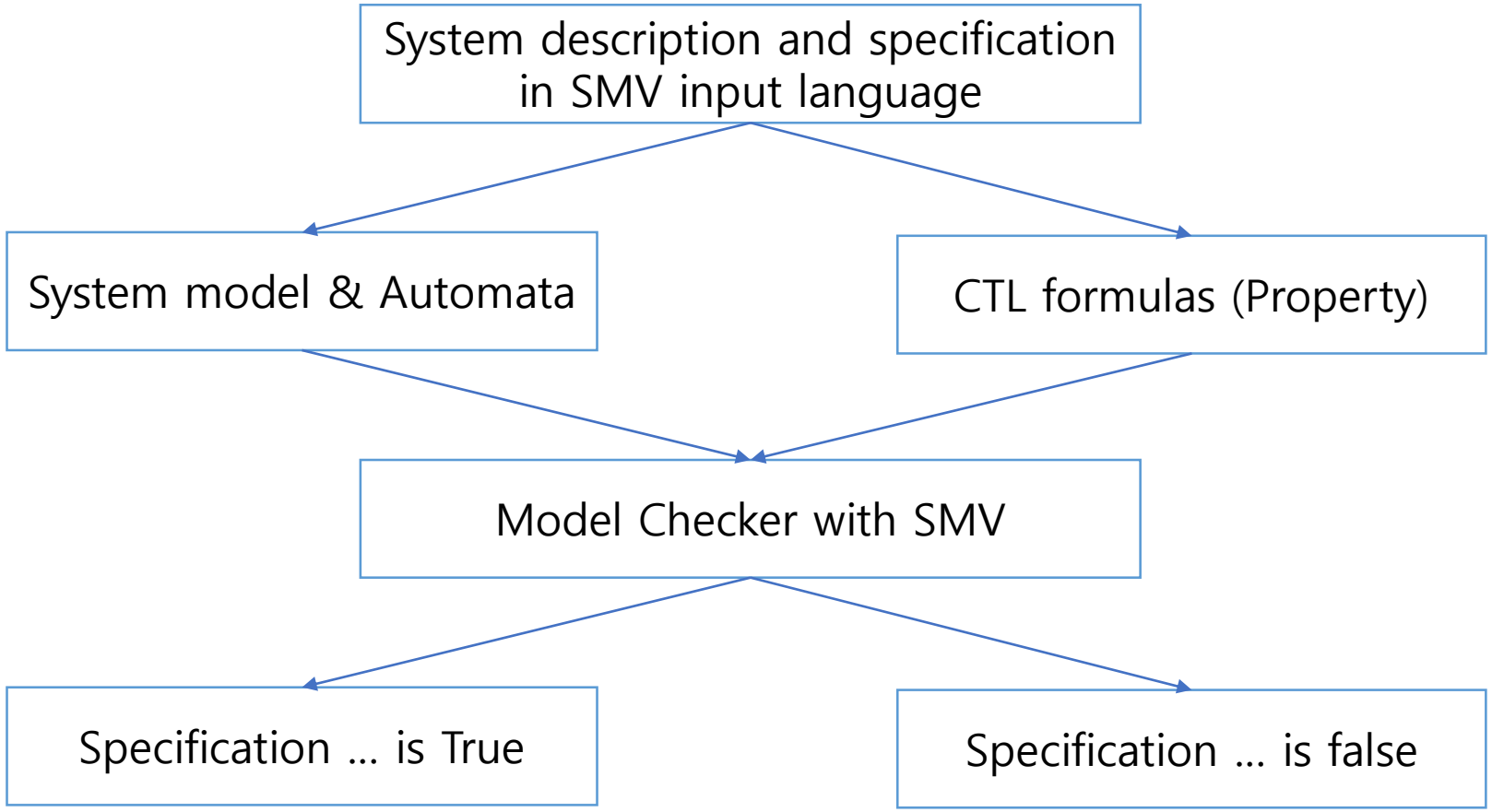
- CTL의 연산자들은 주어진 유한 모델(finite model)의 임의의 한 상태(state)가 주어진 논리식(formula)을 만족하는지 효율적으로 알아 볼 수 있다는 특성을 가짐.

- CTL에서 사용되는 시간 연산자인 F, G, X, U, R 외에 '모든 실행 가능한 경로에 대해'를 의미하는 A, '어떤 실행가능한 경로에 대해'를 의미하는 E가 포함되어 사용됨

- 시간의 흐름에 따라 발생 가능한 경로를 트리로 표현

SMV 시스템 구조

- Model checking



Ubuntu에 SMV 설치방법

Step 1. SMV 파일을 다운로드한다.
Step 2. /usr/local/smv/ 폴더 안에 다운로드 받은 파일을 푼다.
Step 3. #gedit ~/.bashrc 에서 다음 내용 추가 후

```
declare -x SMV_DIR="/usr/local/smv"
declare -x PATH="$SMV_DIR/bin:$PATH"
declare -x MANPATH="$SMV_DIR/man:$MANPATH"
declare -x LD_LIBRARY_PATH="$SMV_DIR/lib:$LD_LIBRARY_PATH"
```

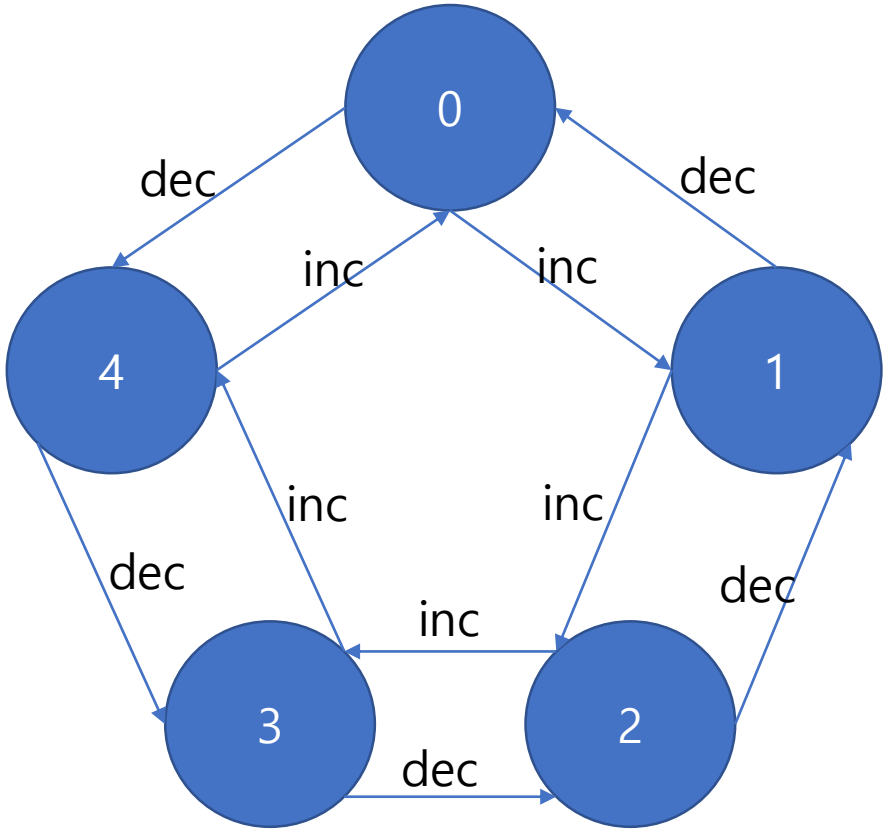
#source ~/.bashrc

Step 4. Package 설치
#apt-get install tk8.5
#apt-get install tix
#apt-get install xemacs21
#apt-get install build-essential

※ 참고) 윈도우용 설치파일 : http://www.kenmcmil.com/smv/smv_latest.exe

Example – a module 5 counter

- .smv 파일 작성



A module 5 counter

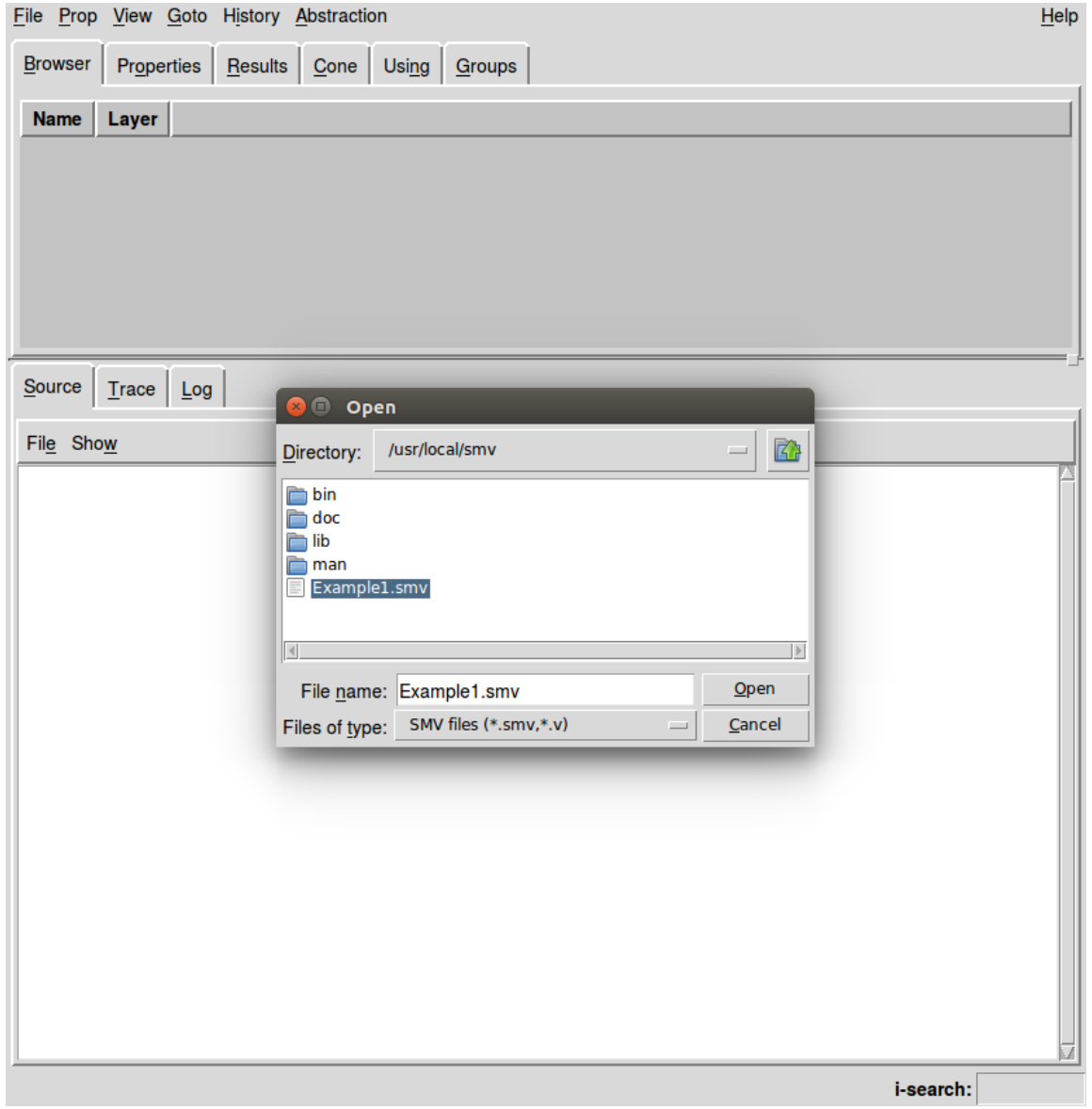
```
module main(inc, dec){
 input inc, dec : boolean;
 state : {0,1,2,3,4};
 default;

 in switch(state){
 0:
   if(inc) next(state) :=1;
   else if(dec) next(state) :=4;
 1:
   if(inc) next(state) :=2;
   else if(dec) next(state) :=0;
 2:
   if(inc) next(state) :=3;
   else if(dec) next(state) :=1;
 3:
   if(inc) next(state) :=4;
   else if(dec) next(state) :=2;
 4:
   if(inc) next(state) :=0;
   else if(dec) next(state) :=3;
};
test1 : assert G ((state = 0&dec) -> !(state=1));
test2 : assert F ((state = 4&inc) -> (state=0));
test3 : assert F ((state = 0&dec) -> (state=4));
test4 : assert G ((state = 0&dec) -> X(state=1));
}
```

Example
– a module 5 counter

- 파일 불러오기

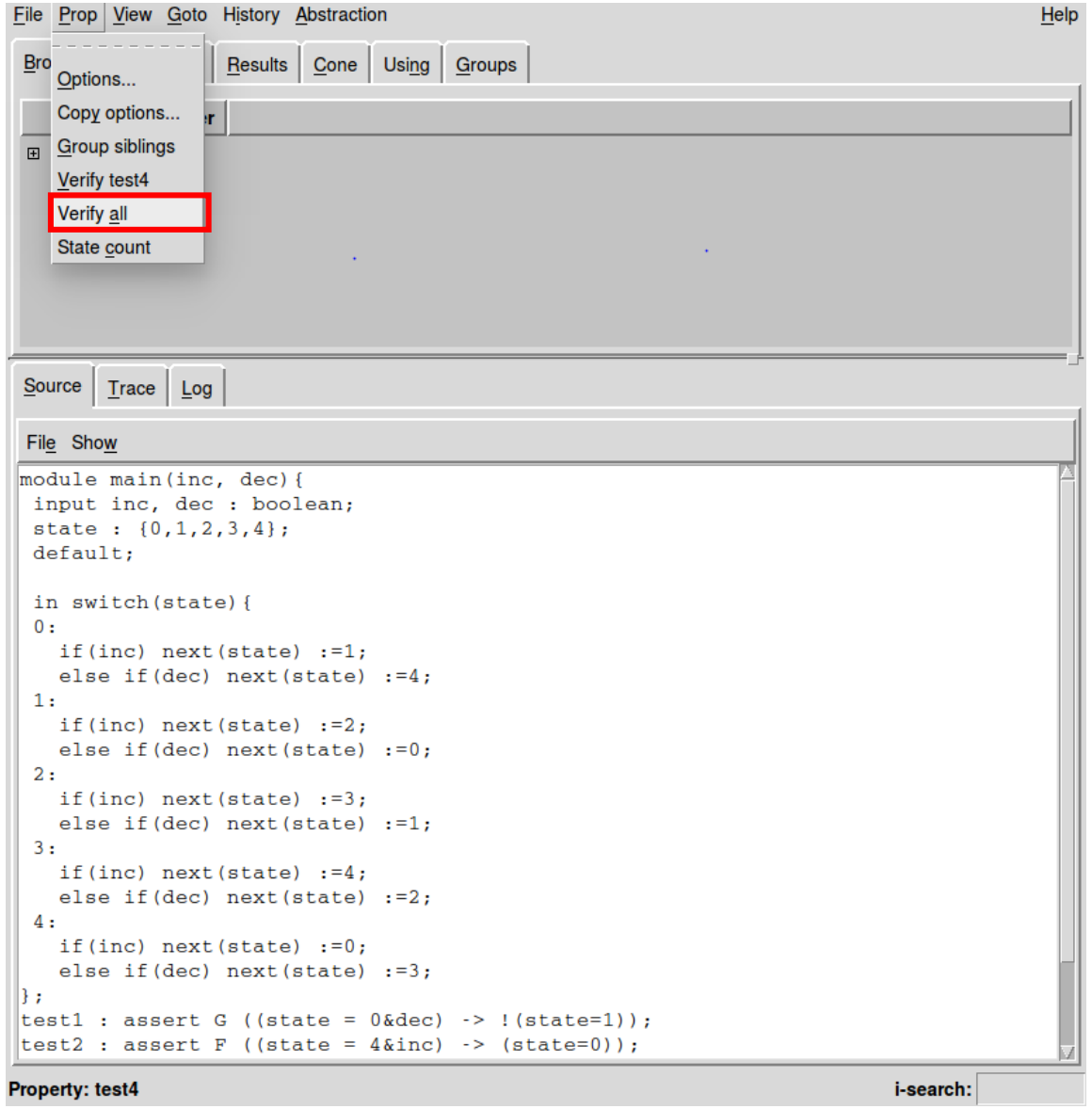
  File -> Open -> ~.smv 파일을 찾은 후 연다.

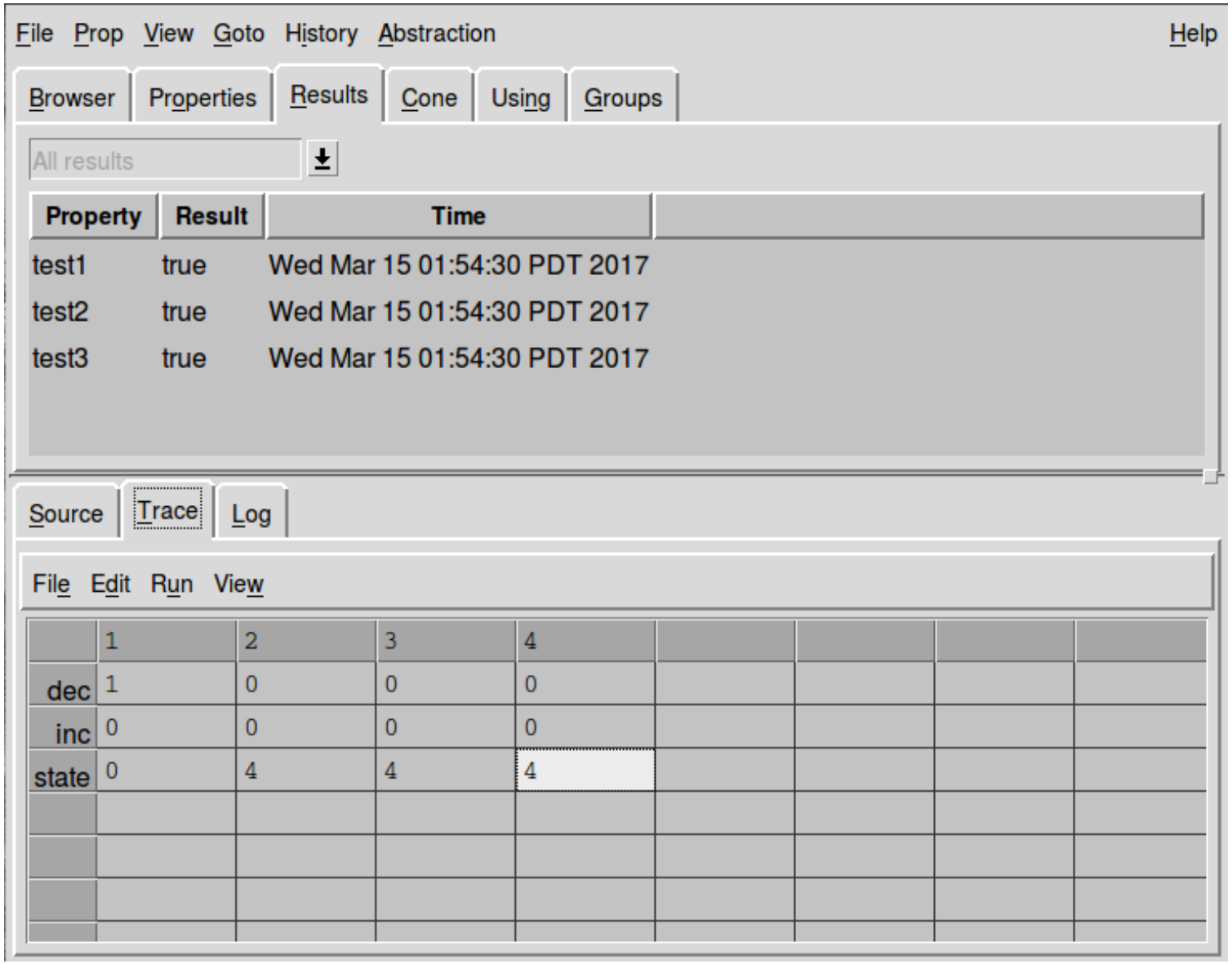


Example
– a module 5 counter

- 파일 실행

  Prop -> Verify all




File Prop View Goto History Abstraction Help
Browser Properties Results Cone Using Groups
All results
Property Result Time
test1 true Wed Mar 15 01:54:30 PDT 2017
test2 true Wed Mar 15 01:54:30 PDT 2017
test3 true Wed Mar 15 01:54:30 PDT 2017
Source Trace Log
File Edit Run View
1 2 3 4
dec 1 0 0 0
inc 0 0 0 0
state 0 4 4 4

참고. 연구 사례

그림 7 SMV를 이용한 검증 결과

원자력 발전소 내장형 Protection 시스템의 설계에 대한 정형기법의 적용+

원자력 발전 내장형 Protection 시스템은 원자력 발전의 과부하를 검출하여 원자로의 가동을 중단시키는 대표적인 Safety-critical 시스템이다. 이러한 시스템은 그 위험성으로 인해 안전성과 신뢰성이 절대적으로 요구되는 시스템이다. 따라서 원자력 발전 내장형 Protection 시스템은 그 위험성을 가만하여 설계 단계부터 V&V가 철저하게 이루어져야 한다. 이러한 내장형 시스템을 설계하기 위해 다양한 V&V 방법이 제시되고 있으며 특히 선진국에서는 정형기법을 이용한 설계가 연구되고 있다. 본 연구에서는 원자력 발전소 소프트웨어의 가이드라인에 맞추어 다양한 측면에서의 정형기법을 이용한 원자력 발전 내장형 Protection 시스템의 설계에 대한 기법을 소개한다.

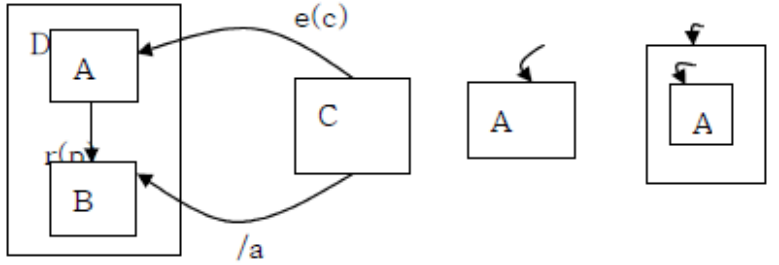


그림 3 STATECHART

그림 4 default state

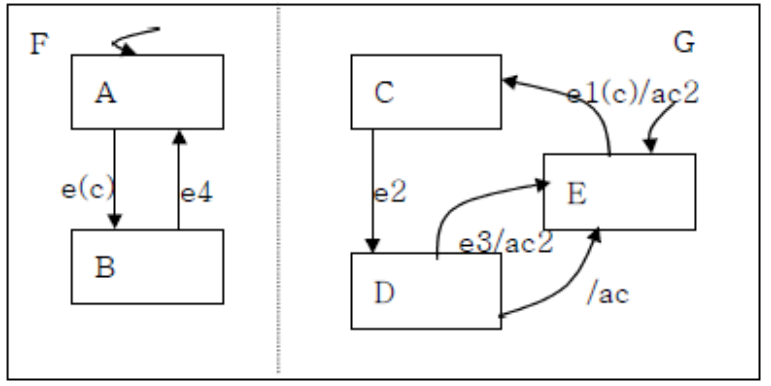


그림 5 병렬적인 STATECHART의 예

참고. 연구 사례

SMV를 이용한 접근통제모델의 정형적 설계방법 연구[1)]

황대연[○*] 강인혜[**] 강필용[***] 이완석[***] 최진영[*]
[*]고려대학교 컴퓨터학과
{dyhwang[○], choi}@formal.korea.ac.kr

요 약

컴퓨터 시스템에 대한 보안의 필요성이 계속적으로 증대되고 있으며 이에 다양한 보안시스템들이 개발되고 있다. 이러한 보안 시스템들이 높은 등급의 평가를 받기위해서는 정형적 방법론을 사용하여 명세 및 검증을 해야 한다. 본 논문에서는 정형 검증의 한 방법론인 모델 체킹을 이용하여 접근통제모델을 설계하고 검증하는 방법을 제안하고자 한다.

4.3 보안 기준

다음은 접근통제모델이 만족해야할 보안 기준들이다.

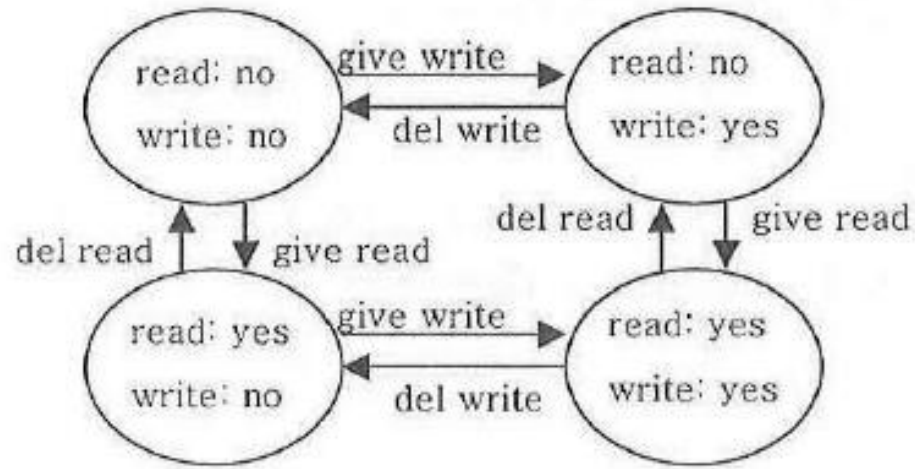
보안 기준 1 : 항상 모든 높은 등급의 주체는 낮은 등급의 객체에 대해 쓰기 권한을 갖지 못한다.

보안 기준 2 : 항상 모든 낮은 등급의 주체는 높은 등급의 객체에 대한 어떠한 권한도 갖지 못한다.

이 보안 기준들을 CTL을 이용하여 SMV에 넣을 속성으로 만들면 각각 다음과 같다.

속성 1: SPEC AG !(obj[2].sub[subject].p[2] = 1
& sub[subject] = high)

속성 2: SPEC AG !((obj[1].sub[subject].p[1] = 1
| obj[1].sub[subject].p[2] = 1)
& sub[subject] = low)



[그림 2] 객체의 접근통제목록 상태도

	1	2	3	
action	4	3	-	
grantPerOk	1	0	-	
obj[2].sub[1].p[2]	0	0	0	
obj[2].sub[2].p[2]	0	1	1	
object	2	1	-	
per	2	1	-	
prohibitPerOk	0	0	-	
sub[1]	high	high	high	
sub[2]	low	low	high	
subject	1	2	2	
tarsubject	2	1	-	

[그림 3] 속성 1에 대한 반례

감사합니다.