

Promela의 Rewriting 기반 의미구조



손병호 (POSTECH)



5th Summer Workshop in Ulsan, Korea
2023. 07. 05.(Wed) ~ 2023. 07. 07.(Fri)

SPIN 모델체커

- Distributed / Concurrent 시스템의 정형검증 툴
- 가장 널리 쓰이는 모델체커의 하나
- 2002 ACM Software System Award 수상
 - Unix, Tex, Java, LLVM, Coq, CompCert, Sel4
 - 모델체커 중에서는 유일

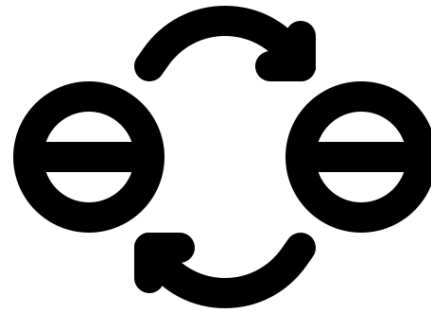


Promela란?

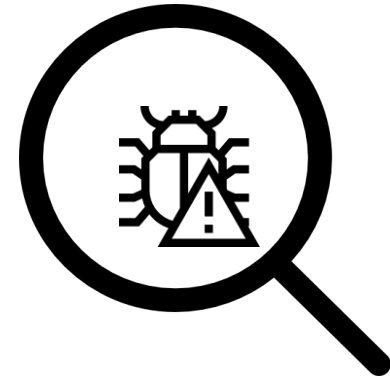
SPIN 모델체커



모델링 언어
(**Promela**)



모델
(유한상태머신)



모델체킹
(LTL)

연구의 방향

1

Promela 언어의
의미구조 정형명세

2

정형명세를 기반으로
효율적인 모델체킹
테크닉 개발

3

SPIN으로 검증하던
기존의 Promela
모델에 적용

연구의 필요성

- Promela 언어의 정형명세 미비
- SPIN 모델체커의 제약
 - 검증성질은 LTL 성질에 한정 (c.f. CTL, PCTL, etc)
 - 모델체킹의 성능향상 테크닉의 제약 (e.g. Partial Order Reduction, etc)
- 모델체킹 커뮤니티 내에서 협업의 어려움
 - 인터페이스의 필요성
 - 특정 모델체킹 알고리즘은 특정 모델체커에서만 활용 가능
 - 서로 다른 모델링 언어로 작성된 모델 간 비교의 어려움

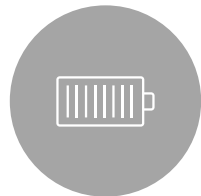
연구의 목표

“검증 친화적인 의미구조”

Promela 언어의 특징



Concurrent Process



Synchronous /
Asynchronous Channel



Guard



Atomicity



Goto



Nondeterminism

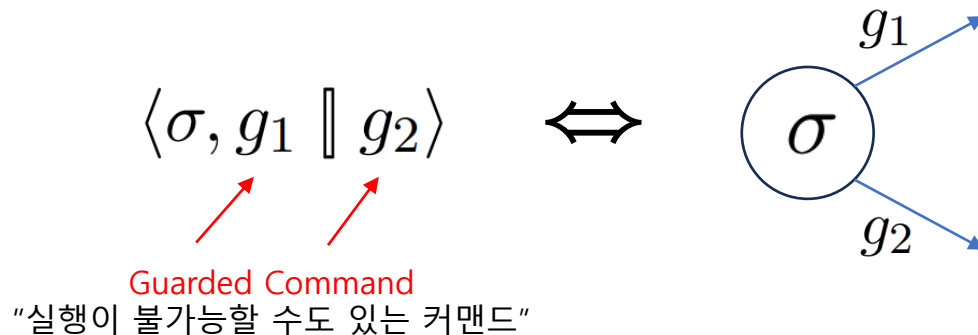


Nondeterminism

Nondeterministic Choice :

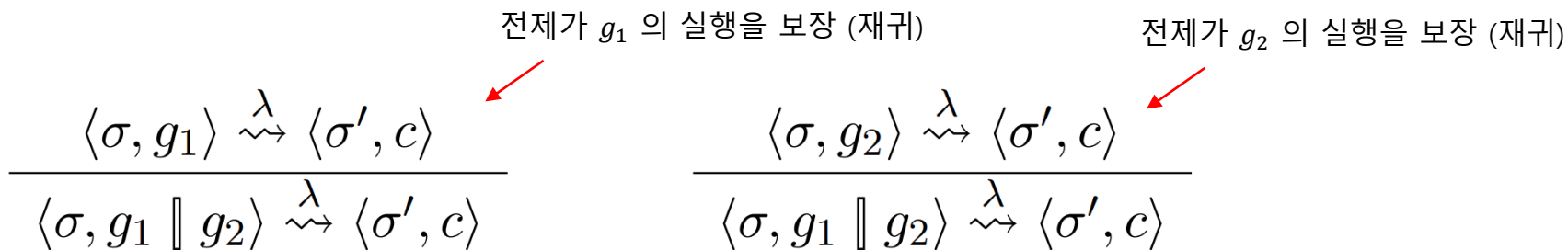
(규칙) g_1 또는 g_2 중

“실행가능한 것 중 아무거나” 실행



전통적인 의미구조 :

“선택과 실행이 한번에”



문제 :

수학적으로는 OK, 계산적으로는 비효율적



Nondeterminism

전통적인 의미구조 :

"선택과 실행이 한번에"

$$\frac{\langle \sigma, g_1 \rangle \xrightarrow{\lambda} \langle \sigma', c \rangle}{\langle \sigma, g_1 \parallel g_2 \rangle \xrightarrow{\lambda} \langle \sigma', c \rangle}$$

전제가 g_1 의 실행을 보장 (재귀)

$$\frac{\langle \sigma, g_2 \rangle \xrightarrow{\lambda} \langle \sigma', c \rangle}{\langle \sigma, g_1 \parallel g_2 \rangle \xrightarrow{\lambda} \langle \sigma', c \rangle}$$

전제가 g_2 의 실행을 보장 (재귀)

재귀를 없애면?

"선택만"

$$\langle \sigma, g_1 \parallel g_2 \rangle \xrightarrow{\lambda} \langle \sigma, g_1 \rangle$$

$$\langle \sigma, g_1 \parallel g_2 \rangle \xrightarrow{\lambda} \langle \sigma, g_2 \rangle$$

문제 :

$$\langle \sigma, g_1 \parallel g_2 \rangle \xrightarrow{\lambda} \langle \sigma, g_1 \rangle \xrightarrow{\lambda} \mathbf{fail}$$

"유효하지 않은 실행 "

과제:

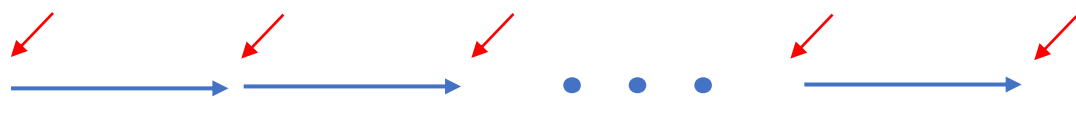
" 유효한 실행만을 효율적 으로 탐색할 수 있는 의미구조 "



Atomicity

Process 0 `acquire lock;`

Process 1  n개

Process 2  n개

2개 프로세스의 경우: $n * n / 2$

k개 프로세스의 경우: $n^k / k!$

과제: 상태공간폭발 문제 최소화!

접근

- Rewriting 기반의 의미구조
 - operational semantics 기반
 - 동시성 프로그래밍 언어를 모듈러하게 정의하는 데에 특화
 - C, Java, Ethereum VM 등의 사례 (K Framework)
- 상태공간폭발 문제
 - Partial Order Reduction
 - SMT 솔버 활용
 - Rewriting 기반 테크닉