논리 기반 정형 명세 및 모델 검증

배경민

2022년 2월 10일

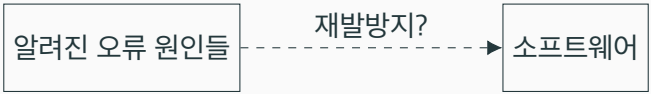
POSTECH 소프트웨어 검증 연구실

그룹3 연구목표: 소프트웨어재난 재발방지

소프트웨어

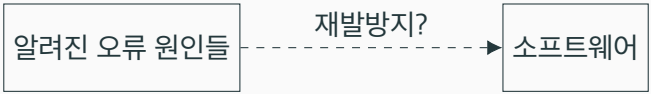
- 동일한 원인에 의해 발생하는 소프트웨어 재난의 재발을 방지

그룹3 연구목표: 소프트웨어재난 재발방지



- 동일한 원인에 의해 발생하는 소프트웨어 재난의 재발을 방지
- 연구내용1: 알려진 오류 원인들에 대한 재난오류 데이터베이스 구축

그룹3 연구목표: 소프트웨어재난 재발방지



- 동일한 원인에 의해 발생하는 소프트웨어 재난의 재발을 방지
- 연구내용1: 알려진 오류 원인들에 대한 재난오류 데이터베이스 구축
- 의문1: 코드기반? 오류 분석이 완료된 동일한 특정 소프트웨어 구현 재검증?

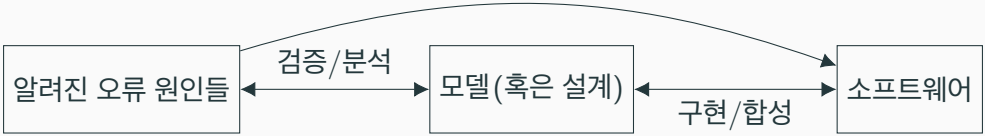
그룹3 연구목표: 소프트웨어재난 재발방지

- 동일한 원인에 의해 발생하는 소프트웨어 재난의 재발을 방지
- 연구내용1: 알려진 오류 원인들에 대한 재난오류 데이터베이스 구축
- 연구내용2: 모델(혹은 설계) 단계에서 소프트웨어 재난의 원인을 분석

그룹3 연구목표: 소프트웨어재난 재발방지

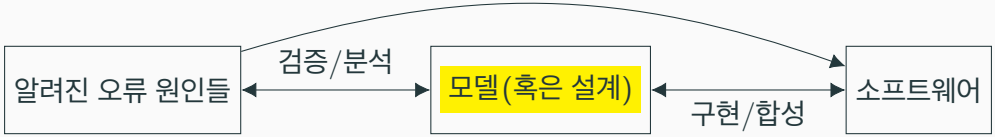
- 동일한 원인에 의해 발생하는 소프트웨어 재난의 재발을 방지
- 연구내용1: 알려진 오류 원인들에 대한 재난오류 데이터베이스 구축
- 연구내용2: 모델(혹은 설계) 단계에서 소프트웨어 재난의 원인을 분석
- 의문2: 모델이나 요구사항이 존재하지 않는 경우? 검증의 복잡성?

그룹3 연구목표: 소프트웨어재난 재발방지

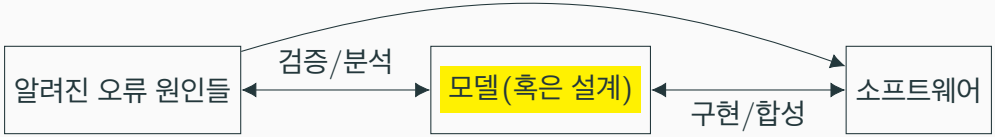


- 동일한 원인에 의해 발생하는 소프트웨어 재난의 재발을 방지
- 연구내용1: 알려진 오류 원인들에 대한 재난오류 데이터베이스 구축
- 연구내용2: 모델(혹은 설계) 단계에서 소프트웨어 재난의 원인을 분석
- 연구내용3: 모델 합성, 요구사항 추론, 오류패턴 기반 모델검증 등 신기술 개발

세부연구목표: 효과적인 모델링 및 검증

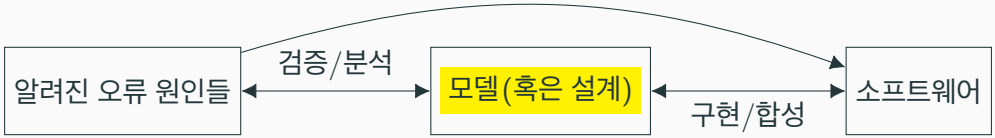


세부연구목표: 효과적인 모델링 및 검증



- 정형명세: 위 목적에 부합하는 효과적인 모델링/명세 기법
  - 고려사항: 검증 대상 소프트웨어, 분석 대상 오류, 모델 합성 용이성, …

세부연구목표: 효과적인 모델링 및 검증



- 정형명세: 위 목적에 부합하는 효과적인 모델링/명세 기법
  - 고려사항: 검증 대상 소프트웨어, 분석 대상 오류, 모델 합성 용이성, …
- 모델검증: 주어진 모델의 요구사항을 효과적으로 검증하는 기법
  - 고려사항: 모델검증의 상태폭발(state-space explosion) 문제

모델검증이란?

모델 검증 (Model Checking)

- 시스템의 오류를 자동으로 찾는 기술
- 시스템의 모든 가능한 상태를 확인하여 "오류 없음" 증명 가능
- 소프트웨어/하드웨어 디자인, 프로토콜 디자인, 소스 코드, ⋯

모델 검증 단계

1. 시스템 명세 (system specification)
   - 모델링 언어 (Promela, Simulink, Verilog, ⋯)
   - 프로그래밍 언어 (C, Java, Haskell, ⋯)
2. 검증 성질 명세 (property specification)
   - functional correctness, safety, liveness, fault tolerance, ⋯
3. 모델 검증 도구
   - SPIN, CBMC, CPAchecker, NuSMV, PAT, TLA+, ⋯

모델 검증 예제: 시스템 명세

- Dekker의 알고리즘
  - 오직 하나의 스레드만 critical section에 접근 가능
  - 일반적인 테스팅을 통해서는 검증이 어려움

```
1  c1 = 1;
2  while (c2 == 1) {
3    if (turn == 2) {
4      c1 = 0;
5      while (turn == 2) { /* wait */ }
6      c1 = 1;
7    }
8  }
9  ... /* critical section */
10 turn = 2;
11 c1 = 0;
12 ...
```

```
1  c2 = 1;
2  while (c1 == 1) {
3    if (turn == 1) {
4      c2 = 0;
5      while (turn == 1) { /* wait */ }
6      c2 = 1;
7    }
8  }
9  ... /* critical section */
10 turn = 1;
11 c2 = 0;
12 ...
```

모델 검증 예제: 검증 성질 명세

- Mutual exclusion

  여러 스레드가 동시에 critical section에 접근할 수 없다.

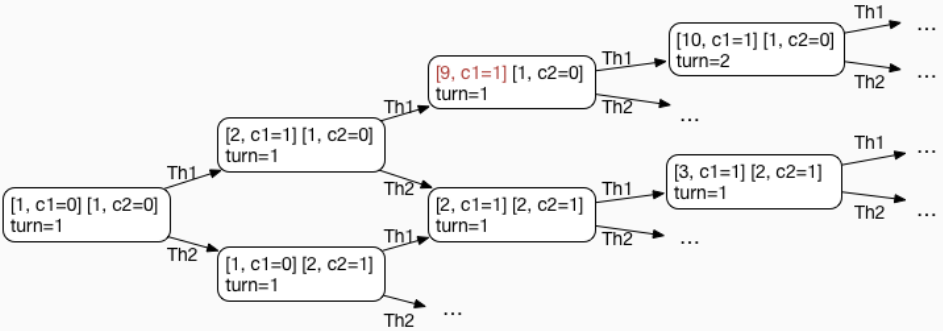
- No starvation

  각 스레드는 언젠가는 critical section에 접근할 수 있다.

- 시제 논리 언어 (temporal logic)로 엄밀히 표현 가능

모델 검증 예제: 모델 검증 알고리즘

- 가능한 모든 실행 시나리오를 검사하여 얻어지는 그래프



- 시스템 모델이 주어진 성질을 만족하는지 검사
  - 예: "mutual exclusion" 이 위배되는 상태에 도달 가능한가?

모델 검증 기법의 장점

- 자동화
  - 시스템 및 성질 명세 후 자동 실행
- 복잡한 성질 검사 가능
  - 동시성 오류, 실시간 요구조건 등
- 오류 재현 용이
  - 오류 발견 시 반례 생성
- 무결성 증명 가능
  - 시스템/성질 명세 수준에서 “오류 없음” 증명

모델 검증 기법의 장점

- 자동화
  - 시스템 및 성질 명세 후 자동 실행
- 복잡한 성질 검사 가능
  - 동시성 오류, 실시간 요구조건 등
- 오류 재현 용이
  - 오류 발견 시 반례 생성
- 무결성 증명 가능
  - 시스템/성질 명세 수준에서 “오류 없음” 증명

⇒ 다양한 산업체에서 활발하게 활용

- 안전필수 SW, 하드웨어 설계, 분산 시스템, 운영체제 설계, …

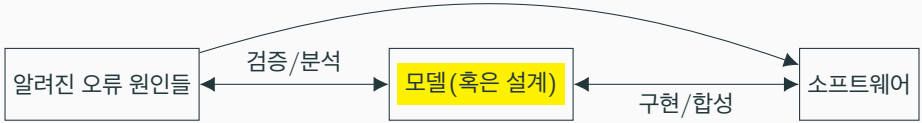
모델 검증 기법 적용의 장애물

- 상태 폭발 문제 (state space explosion)
  - 가능한 상태의 숫자가 기하급수적으로 증가

모델 검증 기법 적용의 장애물

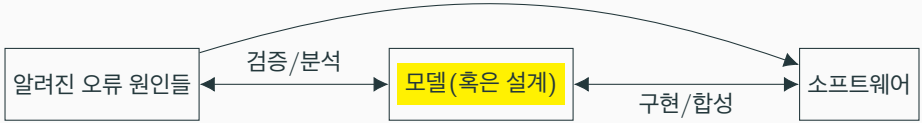
- 상태 폭발 문제 (state space explosion)
  - 가능한 상태의 숫자가 기하급수적으로 증가
- 모델 검증 도구 언어의 표현력 문제
  - 해당 도구의 언어(e.g, CBMC: C)에서 대상 시스템의 중요한 성질이 표현 불가능한 경우

Representational Gap: 소프트웨어재난 재발방지 연구의 경우



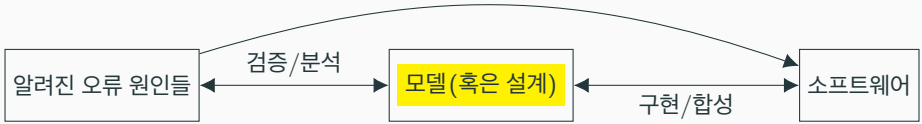
- 어떠한 모델링 언어로 모델을 작성/합성하여야 할 것인가?
- C? Java? Boolean circuit? Petri nets? π-calculus? $\cdots$

Representational Gap: 소프트웨어재난 재발방지 연구의 경우



- 어떠한 모델링 언어로 모델을 작성/합성하여야 할 것인가?
- ~~C? Java? Boolean circuit? Petri nets? π-calculus? ...~~
- 대상 성질/오류 및 시스템의 특성에 따라서 상이한 모델링 언어(들) 필요

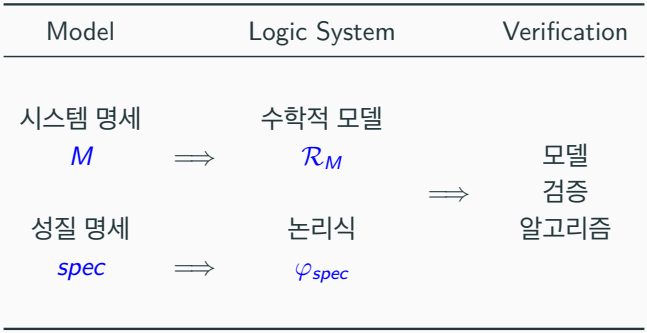
Representational Gap: 소프트웨어재난 재발방지 연구의 경우



- 어떠한 모델링 언어로 모델을 작성/합성하여야 할 것인가?
- ~~C? Java? Boolean circuit? Petri nets? π-calculus? $\cdots$~~
- 대상 성질/오류 및 시스템의 특성에 따라서 상이한 모델링 언어(들) 필요
- 정형 명세에서의 low representational gap을 달성하려면?

논리 기반 모델 검증

접근방법: 논리 기반 모델 검증



- 원하는 모델링 언어 및 모델링 결과를 사용 가능
- 논리 시스템의 다양한 알고리즘 및 최적화 기법 적용 가능
- 플러그인의 형태로 위 과정을 디자인 도구와 결합 가능

예: 다양한 기반 논리 시스템 및 모델검증 도구들

- Boolean logic
  - CBMC, NuSMV, ⋯
- Satisfiability modulo theories (SMT)
  - nuXmv, MCMT, ⋯
- Rewriting logic
  - Maude, KEVM, RV-Predict, CafeOBJ, ⋯
- Temporal logic of actions
  - TLA+

연구 방향

1. 대상 시스템에 최적화된 모델링/정형명세 기법 연구
    - low representational gap 달성

연구 방향

1. 대상 시스템에 최적화된 모델링/정형명세 기법 연구
   - low representational gap 달성
2. 해당 모델링 언어의 의미구조 정형명세 연구
   - rewriting logic, SMT 등으로 operational semantics 정의

연구 방향

1. 대상 시스템에 최적화된 모델링/정형명세 기법 연구
   - low representational gap 달성
2. 해당 모델링 언어의 의미구조 정형명세 연구
   - rewriting logic, SMT 등으로 operational semantics 정의
3. 의미구조 수준에서 효과적인 알고리즘/최적화 기법 연구
   - encoding, abstraction, search heuristics/strategies, ⋯,

진행 연구: Signal Temporal Logic 모델 검증 (1)

- 하이브리드 시스템 (Hybrid system): 소프트웨어 모델 + 물리모델

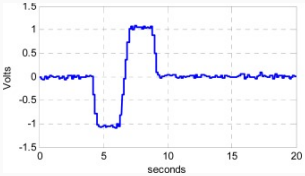


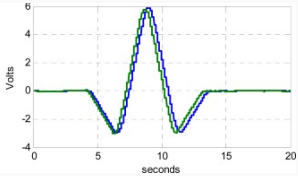
진행 연구: Signal Temporal Logic 모델 검증 (1)

- 하이브리드 시스템 (Hybrid system): 소프트웨어 모델 + 물리모델



- 하이브리드 시스템의 상태: 연속적인 시그널로 표현



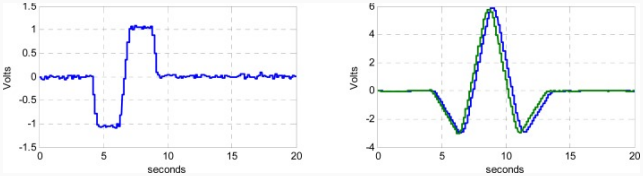


진행 연구: Signal Temporal Logic 모델 검증 (1)

- 하이브리드 시스템 (Hybrid system): 소프트웨어 모델 + 물리모델



- 하이브리드 시스템의 상태: 연속적인 시그널로 표현



- Signal Temporal Logic (STL): 연속적인 시그널의 성질을 표현

진행 연구: Signal Temporal Logic 모델 검증 (2)

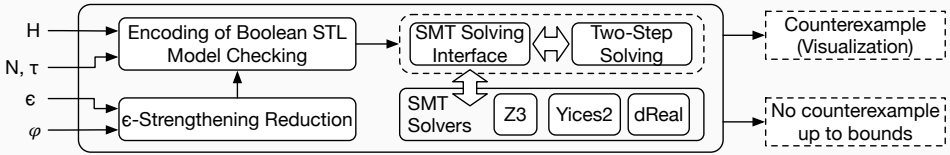
- 과거: 주로 오류탐색(falsification) 또는 테스트 목적으로 활용됨
  - STL 모델검증 기술은 거의 개발되지 않았음

진행 연구: Signal Temporal Logic 모델 검증 (2)

- 과거: 주로 오류탐색(falsification) 또는 테스트 목적으로 활용됨
  - STL 모델검증 기술은 거의 개발되지 않았음
- STL 모델검증 알고리즘 개발
  - 주어진 한계까지 오류를 완전히 탐색하는(refutation-complete) 이론 [POPL'19]
  - 알고리즘의 성능 향상을 위해 불필요한 계산을 줄이는 방법 [ASE'21]
  - 기반논리시스템: SMT + Ordinary differential equations

진행 연구: Signal Temporal Logic 모델 검증 (2)

- 과거: 주로 오류탐색(falsification) 또는 테스트 목적으로 활용됨
  - STL 모델검증 기술은 거의 개발되지 않았음
- STL 모델검증 알고리즘 개발
  - 주어진 한계까지 오류를 완전히 탐색하는(refutation-complete) 이론 [POPL'19]
  - 알고리즘의 성능 향상을 위해 불필요한 계산을 줄이는 방법 [ASE'21]
  - 기반논리시스템: SMT + Ordinary differential equations
- STLmc: 실용적인 STL 모델검증 도구 개발 [in submission]
  - 시스템 행위의 작은 차이에 견고한 모델 검증 (robust model checking)

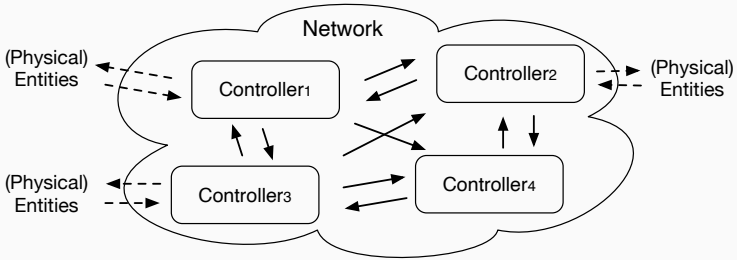


진행 연구: 분산 시스템 AADL 모델 검증 (1)

- AADL (Architecture Analysis and Design Language)
  - (항공분야) 임베디드 시스템의 표준 모델링 언어
  - Airbus, Boeing, Rockwell-Collins, Ford, Lockheed Martin, Raytheon, Toyota, ⋯

진행 연구: 분산 시스템 AADL 모델 검증 (1)

- AADL (Architecture Analysis and Design Language)
  - (항공분야) 임베디드 시스템의 표준 모델링 언어
  - Airbus, Boeing, Rockwell-Collins, Ford, Lockheed Martin, Raytheon, Toyota, ⋯
- Virtually synchronous 분산 시스템
  - Synchronous design: 각각의 컴포넌트가 주기적으로 동시에 실행
  - Distributed implementation: 분산 네트워크 환경에 구현



진행 연구: 분산 시스템 AADL 모델 검증 (1)

- AADL (Architecture Analysis and Design Language)
  - (항공분야) 임베디드 시스템의 표준 모델링 언어
  - Airbus, Boeing, Rockwell-Collins, Ford, Lockheed Martin, Raytheon, Toyota, ⋯
- Virtually synchronous 분산 시스템
  - Synchronous design: 각각의 컴포넌트가 주기적으로 동시에 실행
  - Distributed implementation: 분산 네트워크 환경에 구현
- AADL로 설계된 Virtually synchronous 시스템의 모델검증
  - 실시간 제약조건 고려: network delays, execution times, clock skews. ⋯
  - 물리 환경과의 상호작용 고려

진행 연구: 분산 시스템 AADL 모델 검증 (2)

- 아키텍처 설계 수준에 적용가능한 효과적 검증 방법론 연구
  - Synchronous design 검증 수행 $\Longrightarrow$ 분산 시스템 모델의 올바름 보장
  - PALS, TTA, LTTA 등 다양한 synchronizer 기반 기법 존재

진행 연구: 분산 시스템 AADL 모델 검증 (2)

- 아키텍처 설계 수준에 적용가능한 효과적 검증 방법론 연구
  - Synchronous design 검증 수행 $\Longrightarrow$ 분산 시스템 모델의 올바름 보장
  - PALS, TTA, LTTA 등 다양한 synchronizer 기반 기법 존재
  - PALS와 TTA를 일반화하여 개선한 설계 및 검증 방법론 [EMSOFT'21]

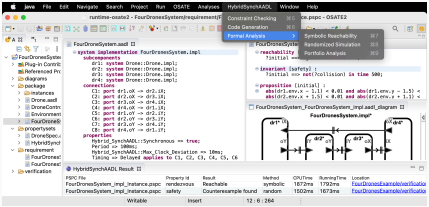
진행 연구: 분산 시스템 AADL 모델 검증 (2)

- 아키텍처 설계 수준에 적용가능한 효과적 검증 방법론 연구
  - Synchronous design 검증 수행 $\Longrightarrow$ 분산 시스템 모델의 올바름 보장
  - PALS, TTA, LTTA 등 다양한 synchronizer 기반 기법 존재
  - PALS와 TTA를 일반화하여 개선한 설계 및 검증 방법론 [EMSOFT'21]
- Virtually Synchronous AADL 모델의 논리기반 모델 검증 연구
  - 기반 논리 시스템: rewriting logic + SMT
  - AADL 언어의 synchronous subset 및 해당 의미구조 정의

진행 연구: 분산 시스템 AADL 모델 검증 (2)

- 아키텍처 설계 수준에 적용가능한 효과적 검증 방법론 연구
  - Synchronous design 검증 수행 $\Longrightarrow$ 분산 시스템 모델의 올바름 보장
  - PALS, TTA, LTTA 등 다양한 synchronizer 기반 기법 존재
  - PALS와 TTA를 일반화하여 개선한 설계 및 검증 방법론 [EMSOFT'21]

- Virtually Synchronous AADL 모델의 논리기반 모델 검증 연구
  - 기반 논리 시스템: rewriting logic + SMT
  - AADL 언어의 synchronous subset 및 해당 의미구조 정의
  - HybridSynchAADL 도구 개발 [CAV'21]



1차년도 연구 내용 및 도전 과제

1차년도 연구

- 목적: 모델생성 및 검증의 공통플랫폼 구축

1차년도 연구

- 목적: 모델생성 및 검증의 공통플랫폼 구축
- 관심 도메인의 소프트웨어에 대한 정형명세 수행

1차년도 연구

- 목적: 모델생성 및 검증의 공통플랫폼 구축
- 관심 도메인의 소프트웨어에 대한 정형명세 수행
  - 차량 전장용 운영체제 표준인 OSEK/VDX 사용

1차년도 연구

- 목적: 모델생성 및 검증의 공통플랫폼 구축
- 관심 도메인의 소프트웨어에 대한 정형명세 수행
  - 차량 전장용 운영체제 표준인 OSEK/VDX 사용
  - Low representational gap을 지향하는 정형명세

1차년도 연구

- 목적: 모델생성 및 검증의 공통플랫폼 구축
- 관심 도메인의 소프트웨어에 대한 정형명세 수행
  - 차량 전장용 운영체제 표준인 OSEK/VDX 사용
  - Low representational gap을 지향하는 정형명세: 모듈 수준 vs. 코드 수준 ?

1차년도 연구

- 목적: 모델생성 및 검증의 공통플랫폼 구축
- 관심 도메인의 소프트웨어에 대한 정형명세 수행
  - 차량 전장용 운영체제 표준인 OSEK/VDX 사용
  - Low representational gap을 지향하는 정형명세: 모듈 수준 vs. 코드 수준 ?
- 관련된 모델검증의 표현력 및 상태폭발문제 해결 기법 탐색

1차년도 연구

- 목적: 모델생성 및 검증의 공통플랫폼 구축
- 관심 도메인의 소프트웨어에 대한 정형명세 수행
  - 차량 전장용 운영체제 표준인 OSEK/VDX 사용
  - Low representational gap을 지향하는 정형명세: 모듈 수준 vs. 코드 수준 ?
- 관련된 모델검증의 표현력 및 상태폭발문제 해결 기법 탐색
  - 정형명세 수준에서 추상화 / 알고리즘 수준의 요약 기법들

1차년도 연구

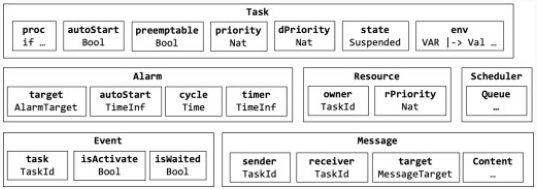
- 목적: 모델생성 및 검증의 공통플랫폼 구축
- 관심 도메인의 소프트웨어에 대한 정형명세 수행
  - 차량 전장용 운영체제 표준인 OSEK/VDX 사용
  - Low representational gap을 지향하는 정형명세: 모듈 수준 vs. 코드 수준 ?
- 관련된 모델검증의 표현력 및 상태폭발문제 해결 기법 탐색
  - 정형명세 수준에서 추상화 / 알고리즘 수준의 요약 기법들
  - 상태공간 축소 기법들

진행 연구: 차량 전장용 운영체제 API 정형명세 (1)

- Low representational gap을 지향하는 정형명세 및 모델검증
  - cf. 기존의 SPIN, CBMC 등 특정한 모델검증 도구를 활용하는 다양한 연구들

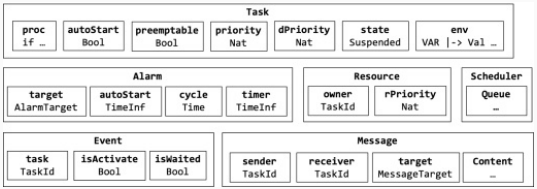
진행 연구: 차량 전장용 운영체제 API 정형명세 (1)

- Low representational gap을 지향하는 정형명세 및 모델검증
  - cf. 기존의 SPIN, CBMC 등 특정한 모델검증 도구를 활용하는 다양한 연구들
- 모듈수준: OSEK/VDX OS 규격의 핵심 부분에 대한 객체지향 명세



진행 연구: 차량 전장용 운영체제 API 정형명세 (1)

- Low representational gap을 지향하는 정형명세 및 모델검증
  - cf. 기존의 SPIN, CBMC 등 특정한 모델검증 도구를 활용하는 다양한 연구들
- 모듈수준: OSEK/VDX OS 규격의 핵심 부분에 대한 객체지향 명세



- 코드수준: C-like 명령형 언어의 K-style 의미구조 정의
- 기반 논리 시스템: Rewriting logic

진행 연구: 차량 전장용 운영체제 API 정형명세 (2)

- 실제(에 가까운 소규모) 응용프로그램 검증 수행
  - Winlift: 차량전장용 창문제어 소프트웨어

진행 연구: 차량 전장용 운영체제 API 정형명세 (2)

- 실제(에 가까운 소규모) 응용프로그램 검증 수행
  - Winlift: 차량전장용 창문제어 소프트웨어
- 상태공간축소 기법
  - "수동"으로 정형명세를 수정하여 모델 수준 단순화
  - Partial order reduction 및 time abstraction 적용

진행 연구: 차량 전장용 운영체제 API 정형명세 (2)

- 실제(에 가까운 소규모) 응용프로그램 검증 수행
  - Winlift: 차량전장용 창문제어 소프트웨어
- 상태공간축소 기법
  - "수동"으로 정형명세를 수정하여 모델 수준 단순화
  - Partial order reduction 및 time abstraction 적용
- 14개의 성질에 대하여 모델검증 수행 (new: 상태공간축소기법 적용)

Test Cases	Old		New		Test Cases	Old		New	
	#State(k)	Time(s)	#State(k)	Time(s)		#State(k)	Time(s)	#State(k)	Time(s)
MotorUp	95.9	67.9	16.5	11.8	CtrlLocked	T/O	T/O	962.5	1287.0
MotorDown	96.1	68.0	16.7	11.9	CtrlUp	T/O	T/O	74.3	52.7
LockTrue	227.8	175.0	33.2	24.5	CtrlDown	T/O	T/O	75.7	53.6
CtrlLock	76.7	53.1	14.9	10.3	CtrlRev	581.0	565.1	107.4	76.7
CtrlOpen	38.7	26.1	8.9	6.6	CtrlStall	364.3	302.1	53.1	38.4

도전과제: 2차년도

- Configurable 명세 수준 추상화
  - 정형명세의 재작성없이 추상화 수준을 자유롭게 선택할 수 없을까?

도전과제: 2차년도

- Configurable 명세 수준 추상화
  - 정형명세의 재작성없이 추상화 수준을 자유롭게 선택할 수 없을까?
  - 아키텍처 수준, 컴포넌트 수준, 코드 수준, ⋯

도전과제: 2차년도

- Configurable 명세 수준 추상화
  - 정형명세의 재작성없이 추상화 수준을 자유롭게 선택할 수 없을까?
  - 아키텍처 수준, 컴포넌트 수준, 코드 수준, ⋯
  - cf. 데이터 요약(data abstraction)

도전과제: 2차년도

- Configurable 명세 수준 추상화
  - 정형명세의 재작성없이 추상화 수준을 자유롭게 선택할 수 없을까?
  - 아키텍처 수준, 컴포넌트 수준, 코드 수준, ⋯
  - cf. 데이터 요약(data abstraction), 점진적 모델링 및 검증(Alloy), ⋯

도전과제: 2차년도

- Configurable 명세 수준 추상화
  - 정형명세의 재작성없이 추상화 수준을 자유롭게 선택할 수 없을까?
  - 아키텍처 수준, 컴포넌트 수준, 코드 수준, ⋯
  - cf. 데이터 요약(data abstraction), 점진적 모델링 및 검증(Alloy), ⋯
- 기존 알려진 오류들을 활용하여 명세 수준 추상화 자동화
  - 매우 많은 수의 명세 수준 추상화 가능성

도전과제: 2차년도

- Configurable 명세 수준 추상화
  - 정형명세의 재작성없이 추상화 수준을 자유롭게 선택할 수 없을까?
  - 아키텍처 수준, 컴포넌트 수준, 코드 수준, ⋯
  - cf. 데이터 요약(data abstraction), 점진적 모델링 및 검증(Alloy), ⋯
- 기존 알려진 오류들을 활용하여 명세 수준 추상화 자동화
  - 매우 많은 수의 명세 수준 추상화 가능성
  - 오류와 관련성이 적은 부분(컴포넌트, 함수, 코드 등)을 우선적으로 자동으로 추상화

도전과제: 2차년도

- Configurable 명세 수준 추상화
  - 정형명세의 재작성없이 추상화 수준을 자유롭게 선택할 수 없을까?
  - 아키텍처 수준, 컴포넌트 수준, 코드 수준, ⋯
  - cf. 데이터 요약(data abstraction), 점진적 모델링 및 검증(Alloy), ⋯
- 기존 알려진 오류들을 활용하여 명세 수준 추상화 자동화
  - 매우 많은 수의 명세 수준 추상화 가능성
  - 오류와 관련성이 적은 부분(컴포넌트, 함수, 코드 등)을 우선적으로 자동으로 추상화
  - cf. 알고리즘 수준의 요약 자동화 (CEGAR 등)

Thank you!