Deductive Model Checking

동시성 시스템 설계의 모델검증을 통한 연역검증

배경민

2025년 8월 20일 (프로그래밍언어연구회 여름학교)

포항공과대학교 컴퓨터공학과

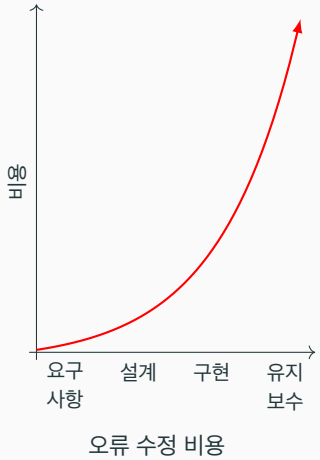
연구배경

소프트웨어 오류의 원인

- 구현 오류
  - (주로 개발자의 실수로) 코드 상에 존재하는 버그
- 설계 오류
  - 설계/알고리즘 수준의 오류

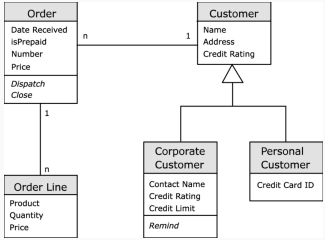
코드 수준 소프트웨어 분석

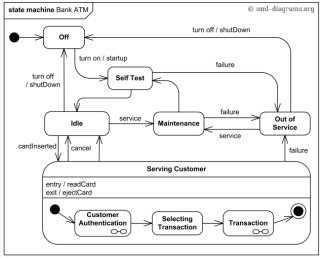
- 코드를 분석하여 구현/설계 오류 및 취약점 분석
  - 실행 가능한 산출물이 존재하여 직관적
- 다양한 종류의 코드 분석 기술 존재
  - systematic test, fuzzing, static analysis, ⋯
- 한계점
  - 구현 후에만 적용 가능
  - 실행 환경에 의존적 (예: 분산 시스템의 코드 수준 분석)

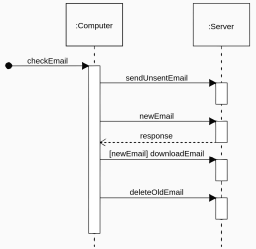


설계 수준 소프트웨어 분석

- 다양한 소프트웨어 구조 및 행위에 대한 설계 방법 존재







- 보통 소프트웨어 개발 단계에서 문서화 및 디자인 리뷰 과정에 사용됨
- 코드와 같이 설계 수준에서 실행 및 자동 분석이 가능한가?

필요 기술: 소프트웨어 설계에 대한 수학적 방법론

- 컴퓨터 시스템의 수학적 모델
- 이러한 모델을 분석할 수 있는 수학적 이론
- 이러한 이론에 근거한 분석/증명 기법

⇒ 공학에서의 일반적인 접근방법

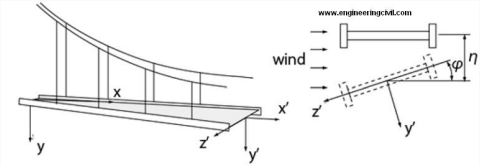


Figure 1. Theoretical model of suspension bridge.

Hirai's research on lateral torsional buckling of suspension bridge starts at the Equation 1.

$$EI\frac{d^4\eta}{dx^4} - 2H_\omega\frac{d^4\eta}{dx^4} - 2h_1\frac{d^2y}{dx^2} + \frac{d^2}{dx^2}(\mathrm{M}\varphi) - (S + (C_d))pb\varphi = 0$$

$$\mathrm{M}\frac{d^2\eta}{dx^2} - EC_\omega\frac{d^4\eta}{dx^4} - \left(GK + \frac{H_\omega b^2}{2}\right)\frac{d^2\eta}{dx^2} - bh_2\frac{d^2y}{dx^2} - S_1pb\varphi^2 = 0 \quad (1)$$

Where, η and φ mean main girder's buckling displacement in vertical and torsional

Formal Methods (정형기법)

수학적 방법론에 기반한 소프트웨어/하드웨어 개발 방법

- Formal specification (정형명세; 엄밀규격): 컴퓨터 시스템 설계에 대한 엄밀한 모델링
- Formal analysis (정형분석; 엄밀분석): 요구사항.성질을 명세 기반으로 검증/테스트
- 명세 기반 구현: 자동(코드 생성) 혹은 수동(설계 기반 구현)

대표적인 엄밀한 검증 방법

- 대화형 정리증명 (Interactive theorem proving)
  - 수학적 증명을 (증명 보조기를 활용하여) 사람이 직접 작성
  - 가장 높은 수준의 보장
  - 큰 비용/시간 소요
- 모델검증 (Model checking)
  - 모든 가능한 행위에 대한 알고리즘적 탐색을 통해 자동으로 검사
  - 자동적 기술로 (대화형 정리증명 대비) 적은 비용
  - 상태폭발문제: 필요한 계산자원이 SW의 복잡도에 따라 기하급수적으로 증가

모델검증의 태동 (1981)

Edmund Clarke

E. Allen Emerson

"*The task of proof construction is in general quite tedious and a good deal of ingenuity may be required to organize the proof in a manageable fashion. We argue that proof construction is unnecessary in the case of finite state concurrent systems and can be replaced by a model-theoretic approach which will mechanically determine if the system meets a specification expressed in propositional temporal logic.*"

- 유한상태 동시성 시스템에 대한 정리증명의 어려움을 해결하기 위한 기술로 제안됨
- 2007년도 ACM Turing Award: Edmund Clarke, E. Allen Emerson, Joseph Sifakis

---

Clarke, E. M., & Emerson, E. A. (1981, May). Design and synthesis of synchronization skeletons using branching time temporal logic. In Workshop on logic of programs (pp. 52-71). Springer.

모델검증 기술의 발전

- 1980 – 2000: 상태공간 탐색 기반
  - 유한상태(finite-state) 시스템의 검증에 초점
  - 효율적인 상태공간 탐색을 위한 자료구조 및 알고리즘
- 2000 – 2025: 자동추론(Automated reasoning) 기반
  - 무한상태를 포함한 적용범위의 확대: 프로그램 코드, 실시간 시스템, 사이버물리시스템, ⋯
  - 모델검증을 위한 자동정리증명(Automated theorem proving): SAT, SMT, Rewriting, ⋯
- 2025 – ?: Neuro-symbolic 기반?
  - 기계학습이나 거대언어모델 등의 적용 가능성에 대한 초기 연구

모델검증의 대표적인 응용분야

- 하드웨어 설계
  - Eectronic design automation (EDA)에서 주요 분석 기술로 널리 사용됨
  - 오류의 직접적인 파급효과가 크고 엄밀한 검증 기법의 적용이 소프트웨어 분야보다 용이함
- 임베디드 시스템의 모델 기반 개발 (Model-based development)
  - Simulink, AADL, Modelica 등 모델링 도구로 소프트웨어 개발, 분석 및 코드 자동 생성
  - 자동차, 항공기, 철도, 선박, 인공위성 등의 소프트웨어 개발에 널리 사용
- 프로토콜 및 분산시스템 검증
  - TLS를 포함하여 다양한 종류의 보안 프로토콜 검증
  - Amazon Web Service, Microsoft Azure 등 클라우드에서 사용되는 분산알고리즘 검증

연구개요

모델검증의 이론적 한계

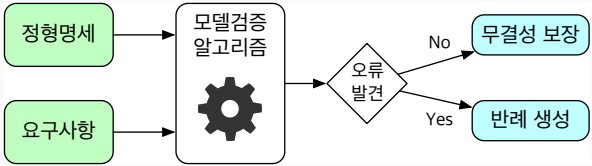
- 유한상태 시스템의 경우
  - 상태폭발문제
- 무한상태 시스템의 경우
  - 일반적으로 결정불가능(Undecidable)
- 해결책
  - 상태공간축소 및 요약 (Abstraction) 기술

모델검증 적용의 장애물 (혹은 모델검증에 대한 오해)

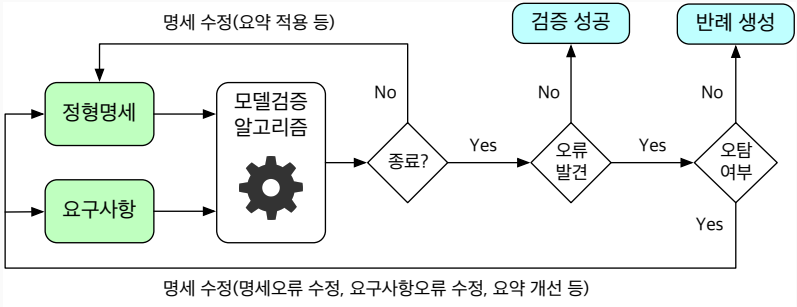
- 모델검증 기술의 장점으로 "자동화"가 많이 언급됨
  - a "push-button" technology
- 실제로 모델검증을 복잡한 시스템에 적용 시, 완전한 자동화는 달성하기 어려움
  - 명세 과정에서의 오류, 요약 기술의 적용 필요, …
- 방법론
  - 도메인에 특화되어 상태공간축소 및 요약/정제 내재화
  - "오류없음 증명"보다는 설계 수준에서의 "테스트"의 개념으로 많이 사용됨

모델검증: 이론 vs 실제

이론:



실제:



실제 적용에서의 모델검증의 한계

- 정형명세에 대한 진입장벽정형명세에 대한 진입장벽
- 모델검증 도구 언어의 표현력 문제모델검증 도구 언어의 표현력 문제
- 점진적인 검증 과정에 대한 고려가 부족함
  - 이전 모델검증 명령어의 결과에 대한 재사용을 지원하지 않음
- 일반적인 상황에 대한 무결성 검증에 제한적임
  - 무한히 많은 초기 상태(입력값, 프로세스의 수 등), 도달가능한 상태공간이 무한한 경우, …

대안: 모델검증 문제에 대화형 정리증명을 적용

- 긍정적인 면
  - 점진적인 검증 과정: 이전 증명 스크립트를 손쉽게 재사용 가능
  - 일반적인 상황에 대한 무결성 검증 가능
  - 효과적인 도구 및 라이브러리가 활발하게 개발되고 있음
- 부정적인 면
  - “모델링 언어”의 문법과 의미구조에 대한 명세가 필요 ⇒ 도메인에 특화된 라이브러리 개발
  - 단순한 문제의 확인에도 많은 작업이 필요 (모델검증기 1초 vs 증명보조기 1시간)
  - 설계 수준의 테스트 용도로 사용하기 어려움

연구목적

모델검증 및 대화형 정리증명의 장점을 결합하는 정형검증 기술 연구

- 점진적인 검증 및 이전 검증 결과의 재활용
- 일반적인 상황에 대한 무결성 검증 가능
- 모델검증의 상태공간 탐색에 대응되는 추론규칙 (혹은 tactic) 제공

- 설계 수준의 테스트 용도로 사용 가능: “실행가능”한 정형명세
- 일반적인 모델검증 문제를 위해 활용 가능한 명세 프레임워크 (혹은 라이브러리) 제공

접근방법: 동시성 시스템 설계의 정형명세

- 시스템 상태: 대수적 자료구조 (inductive data types)

$$\text{term } t$$

- 상태 변화: 재작성 규칙 (rewrite rules)

$$(\forall X)\ l \longrightarrow r \ \textbf{if}\ \psi \qquad \text{(패턴 } l\text{과 } r\text{, 조건 } \psi\text{, 변수 집합 } X\text{)}$$

- 예제: $(\forall x)\ f(x) \longrightarrow f(x+1)$ **if** $x < 10$
  - $f(5)$는 $?f(5+1)$로 변화함
- Rewriting logic 이론에 기반
  - 다양한 동시성 시스템 모델 및 프로그래밍 언어 의미구조 정의에 효과적임이 알려져 있음

접근방법: 상태 집합의 표현

- 패턴 논리식 (Pattern predicate)

$$p_{u;\phi}(y) \equiv (\exists X)\ y = u \wedge \phi$$

  - 패턴 u와 일치하고 조건 ϕ를 만족하는 모든 상태의 집합을 의미

- 예제: $p(y) \equiv (\exists z)\ y = f(g(z)) \wedge z > 1$
  - $f(g(2))$, $f(g(3))$, $f(g(4))$, $\cdots$

접근방법: 추론 규칙 (1)

- 패턴 논리식 변환 (Predicate transformer) 규칙

$$\frac{u\sigma = l\sigma}{(\exists X)\ y = u \wedge \phi \vdash\!\!\xrightarrow{(\forall X)\ l \longrightarrow r \textbf{ if } \psi} (\exists X)\ y = r\sigma \wedge (\phi \wedge \psi)\sigma}$$

  - σ: u와 l를 동일하게 만드는 변수 치환 (Unifier)

- 예제

$$(\exists z)\ y = f(g(z)) \wedge z > 1 \vdash\!\!\xrightarrow{(\forall x)\ f(x) \to f(x+1)\ \textbf{ if }\ x<10} (\exists x)\ y = f(g(z)+1) \wedge (z > 1 \wedge g(z) < 10)$$

- 모델검증의 상태공간 탐색에 대응되는 핵심 추론규칙 중 하나

접근방법: 추론 규칙 (2)

- 귀납 추론(Induction)을 통한 도달불가능(Unreachability)
  - 초기상태 패턴 p_{init}, 오류상태 패턴 p_{error}, 불변성(Invariant) 패턴 $p_1, \ldots, p_k$

$$\frac{\begin{array}{l} p_{init}\text{가 } p_1 \vee \ldots \vee p_k\text{에 포함됨} \\ \text{각 불변성 패턴 } p_i\text{의 '변환'이 } p_1 \vee \ldots \vee p_k\text{에 포함됨} \\ \text{오류상태 패턴 } p_{error}\text{는 각 불변성 패턴 } p_i\text{와 겹치지 않음} \end{array}}{p_{init}\text{에서 } p_{error}\text{로 도달불가능}}$$

- 모델검증에서 Safety 증명을 위한 기본 추론규칙에 해당

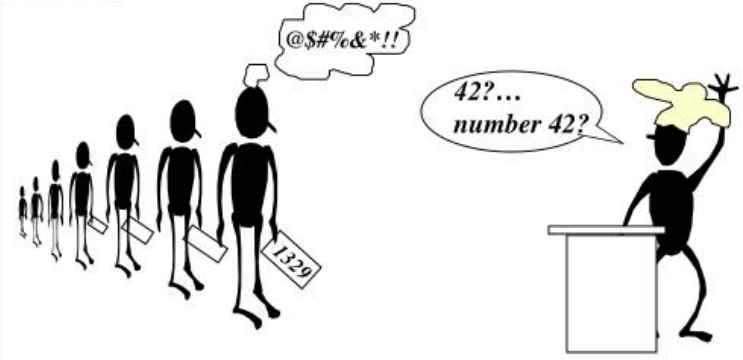
Deductive Model Checking 방법론: 도달불가능 검증

1. 입력
   - 초기상태 패턴 p_{init}, 오류상태 패턴 p_{error}
2. 불변성 패턴 $p_1, \cdots, p_k$ "발견"
   - 자동으로 찾는 연구는 현재 진행 중
3. 모델검증 기반 추론
   - 위 추론 규칙들을 적용. 추가 증명이 필요한 검증 조건 (Verification condition) 생성
4. 검증조건 증명
   - 증명보조기를 활용하여 증명

- DM-Check: 위 방법론을 구현하는 Deductive model checking 프로토타입 도구
  - Maude 모델검증기 및 NuIPT 정리증명기 기반 (Maude Team과의 공동연구로 개발 진행 중)

Deductive Model Checking 예제

Example: Lamport's Bakery Algorithm (1)



- 각 프로세스는 번호표를 뽑고 대기
- 가장 작은 번호를 가진 프로세스부터 순차적으로 자원에 접근

Example: Lamport's Bakery Algorithm (2)

- N개의 프로세스가 있는 상태 표현

$$\langle n, m, \{[d_1], \ldots, [d_N]\}\rangle$$

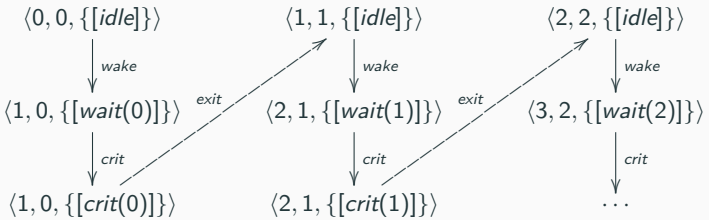
  - n: 번호표 발급기의 현재 번호
  - m: 현재 자원에 접근가능한 번호
  - $\{[d_1], \ldots, [d_N]\}$: 프로세스들의 중복 집합 (Multiset). $d_i \in \{\texttt{idle}, \texttt{wait}(k), \texttt{crit}(k)\}$

- 재작성 규칙

$$\begin{aligned}
\mathit{wake}: &\quad (\forall n, m, ps) \quad \langle n, m, \{[\mathit{idle}]\} + ps\rangle \longrightarrow \langle n+1, m, \{[\mathit{wait}(n)]\} + ps\rangle \\
\mathit{crit}: &\quad (\forall n, m, ps) \quad \langle n, m, \{[\mathit{wait}(m)]\} + ps\rangle \longrightarrow \langle n, m, \{[\mathit{crit}(m)]\} + ps\rangle \\
\mathit{exit}: &\quad (\forall n, m, ps) \quad \langle n, m, \{[\mathit{crit}(m)]\} + ps\rangle \longrightarrow \langle n, m+1, \{[\mathit{idle}]\} + ps\rangle
\end{aligned}$$

Example: Lamport's Bakery Algorithm (3)

- 프로세스 1개의 경우에 대한 상태 변화



- 무한상태 시스템: n, m, 프로세스 수

Example: Lamport's Bakery Algorithm (4)

- 초기패턴

$$(\exists n, ps)\ y = \langle n, n, ps \rangle \ \wedge\ allProcIdle(ps)$$

- 오류패턴

$$(\exists n, m, k, l, ps)\ y = \langle n, m, \{crit(k), crit(l)\} + ps \rangle$$

- 불변성패턴

$$(\exists n, ps)\ y = \langle n, n, ps \rangle \ \wedge\ allProcIdle(ps)$$

$$(\exists n, m, ps)\ y = \langle n, m, ps \rangle \ \wedge\ noProcCrit(ps) \ \wedge\ n > m \ \wedge\ (\forall k \in tickets(ps))\ m \leq k < n$$

$$(\exists n, m, ps)\ y = \langle n, m, \{crit(m)\} + ps \rangle \ \wedge\ noProcCrit(ps) \ \wedge\ n > m \ \wedge\ (\forall k \in tickets(ps))\ m < k < n$$

Example: Lamport's Bakery Algorithm (5)

- DM-Check에서 모델검증 추론규칙을 통하여 5개의 추가 검증 조건 생성
- 나머지 검증 조건들은 NuITP의 자동 추론 기능으로 증명됨
- NuITP를 통해서 추가 검증 조건들에 대한 검증 수행

맺는말

- Deductive model checking
  - 모델검증과 대화형 정리증명의 장점을 통합하는 새로운 검증 기술
  - 현재는 초기 연구 단계이며, DM-Check 프로토타입 도구 개발
- 진행 연구
  - Lean 증명보조기를 통한 Deductive model checking 진행
  - Lean에서의 rewriting logic 스타일의 명세 라이브러리 및 모델검증 “tactic” 연구
- 향후 연구
  - 도달불가능 성질 뿐만 아니라, 시제논리(LTL 등)로 검증 성질 확장
  - 적절한 패턴 논리식을 효과적으로 생성/합성/발견할 수 있는 기술 연구
  - 분산시스템 설계, 보안 프로토콜 등의 검증에 deductive model checking 적용

감사합니다!