CODEMIND
Into the Source

프로그램 분석과 검증

이론에서 상용도구로

신승철@코드마인드

2024. 12. 18

한국프로그래밍언어 역사워크샵

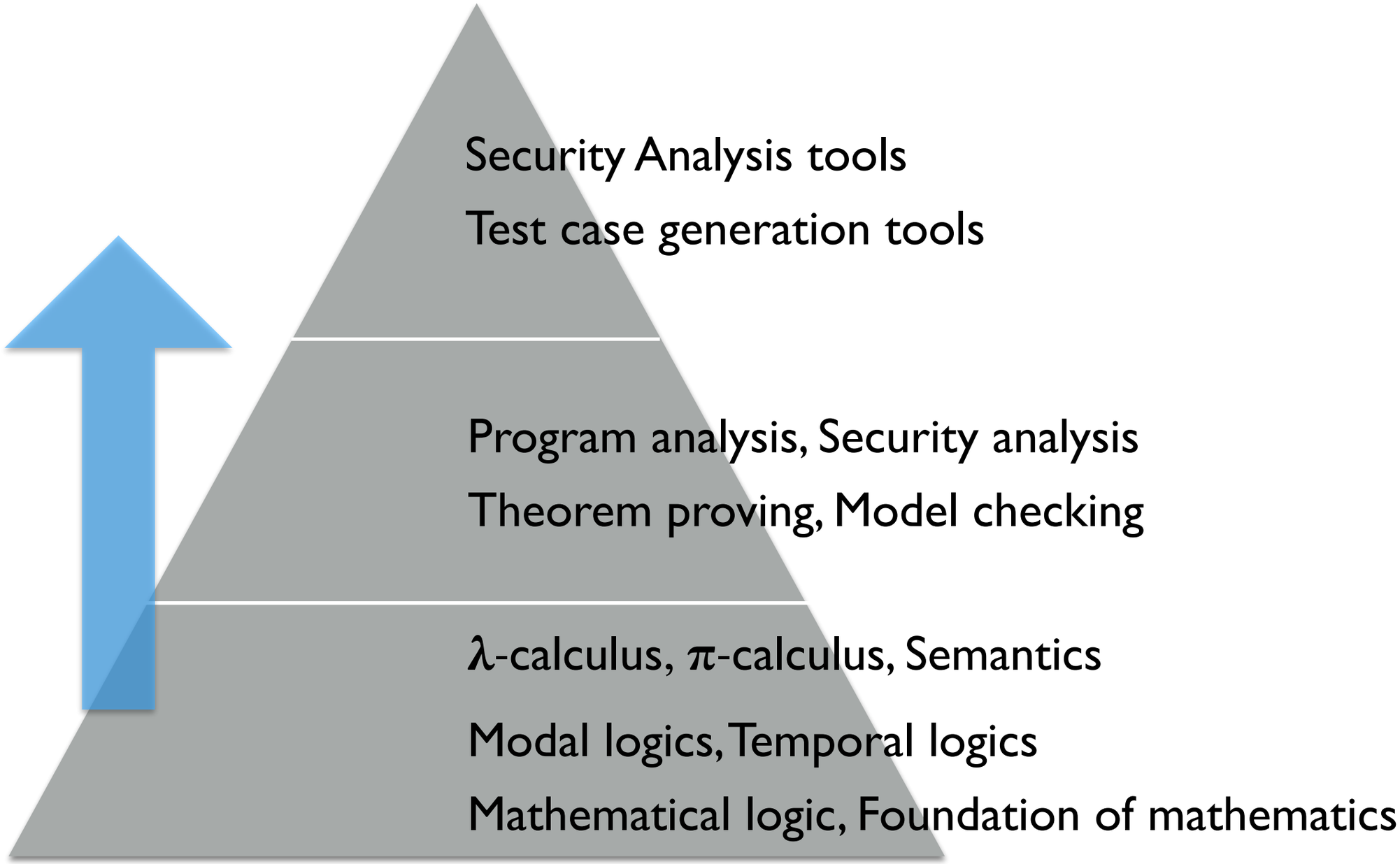
내용

- 개인적인 관심 주제
- 기초 이론에서 상용 도구로
- 상용 도구 개발
- 우리를 이끈 패러다임

주요 관심 주제

- 1980년대 ~ 1990년대
  - Functional languages, Semantics, Type systems
  - λ-calculus, π-calculus
- 1990년대 ~ 2000년대
  - Modal logics, Temporal logics, Model checking, Theorem proving
  - Mathematical logic, Foundation of mathematics
- 2000년대 ~ 2010년대
  - Software security analysis, Control language verification
- 2010년대 ~ 2020년대
  - Security Analysis tools, Test case generation tools

기초 이론, 방법론, 상용도구

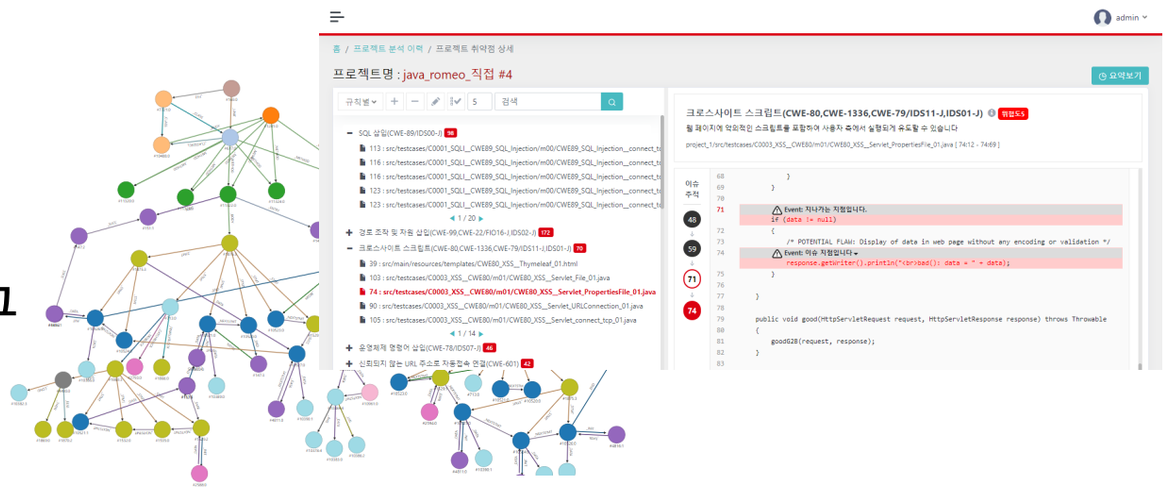


CODEMIND
Into the Source

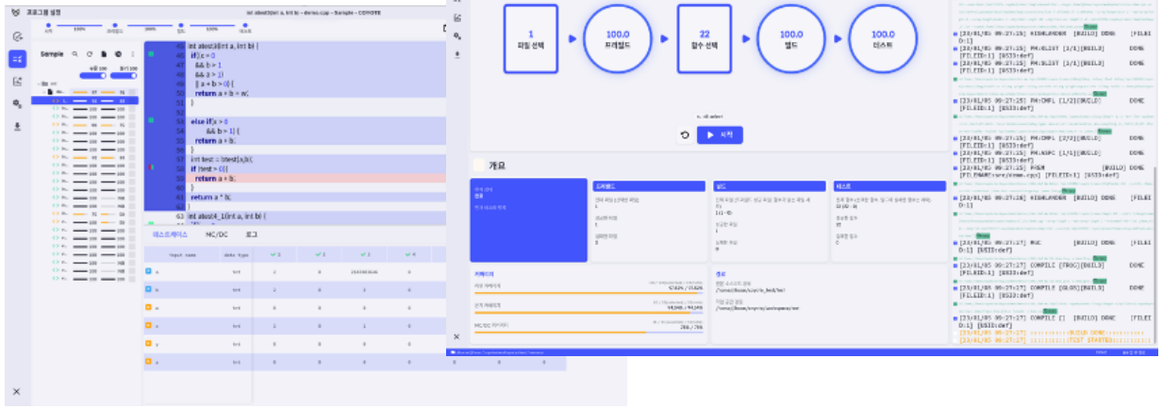
프로그래밍언어연구회 PL 1987 정보과학회

SW 분석·검증을 위한 상용 도구

- 정적 분석
  - 요약해석 + 질의 기반 분석 엔진
  - 그래프DB 기반 소스코드 보안진단 도구



- 동적 테스팅
  - 콘콜릭 실행 + 정리증명 기반 엔진
  - 테스트케이스 생성 도구



- 그 외에도
  - 악성코드 명령제어 흐름분석 엔진(국가보안기술연구소)
  - 차세대 엔진제어기 멀티코어 프로그램의 분산병렬화 분석 엔진(현대자동차)
  - COTS 바이너리를 위한 취약점 분석 엔진(미래과학아카데미)
  - 특수목적언어 정적분석 엔진(삼성SDS)

상용 도구 개발을 위한 기술

- 개발 언어
  - Ocaml, F#, Scala, Haskell (엔진)
  - Java, C# (시스템)
- 엔진 방법론
  - 요약해석 기반 정적분석, 값분석, 메모리분석, 오염분석
  - 질의 기반 보안 분석, 데이타흐름 분석, 타입 분석
  - 프로시저간 분석, 합성 분석
  - 콘콜릭 실행, 정리증명 (SMT 풀이)
  - 코드 생성, 코드 합성, 코드 삽입
  - 바이너리 분석, 값집합 분석
- 상용화 엔지니어링
- 백엔드 시스템과 프론트엔드 UX

SW 검증 도구의 제품화

데이터 관리
사용자 관리
프로세스 관리
서버 시스템

사용자 인터페이스

분석 및 검증 엔진

CODEMIND
Into the Source

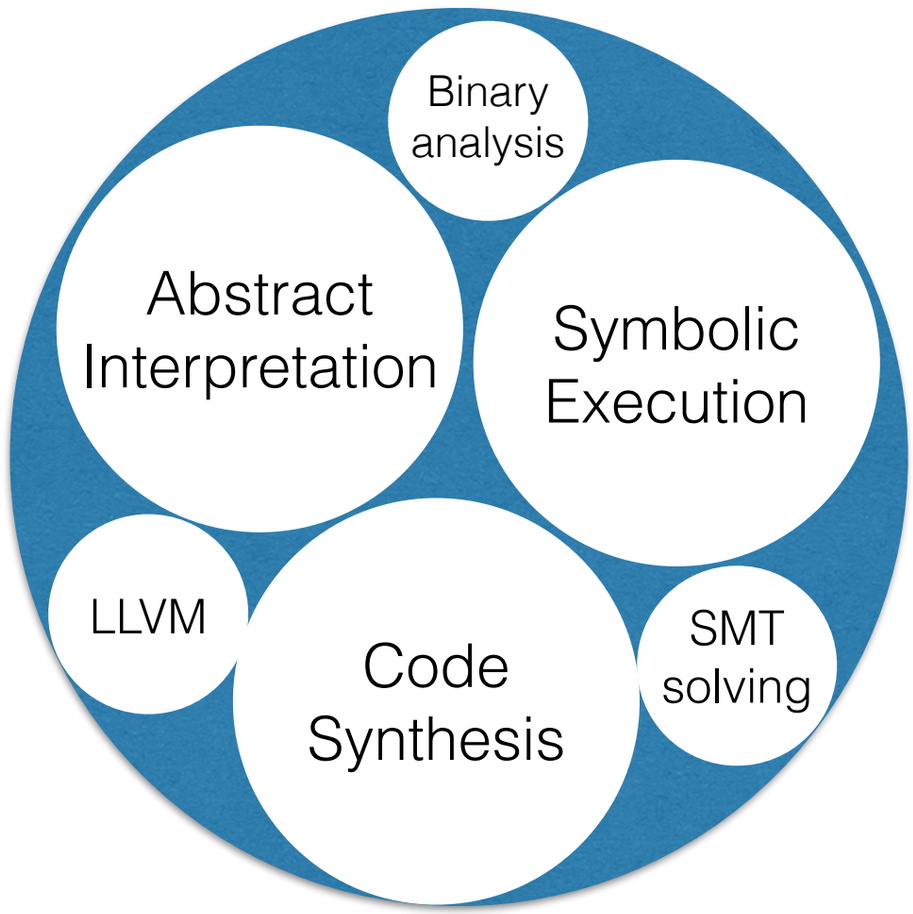
SW 검증 도구의 제품화

데이터 관리
사용자 관리
프로세스 관리
서버 시스템

사용자 인터페이스

분석 및 검증 엔진

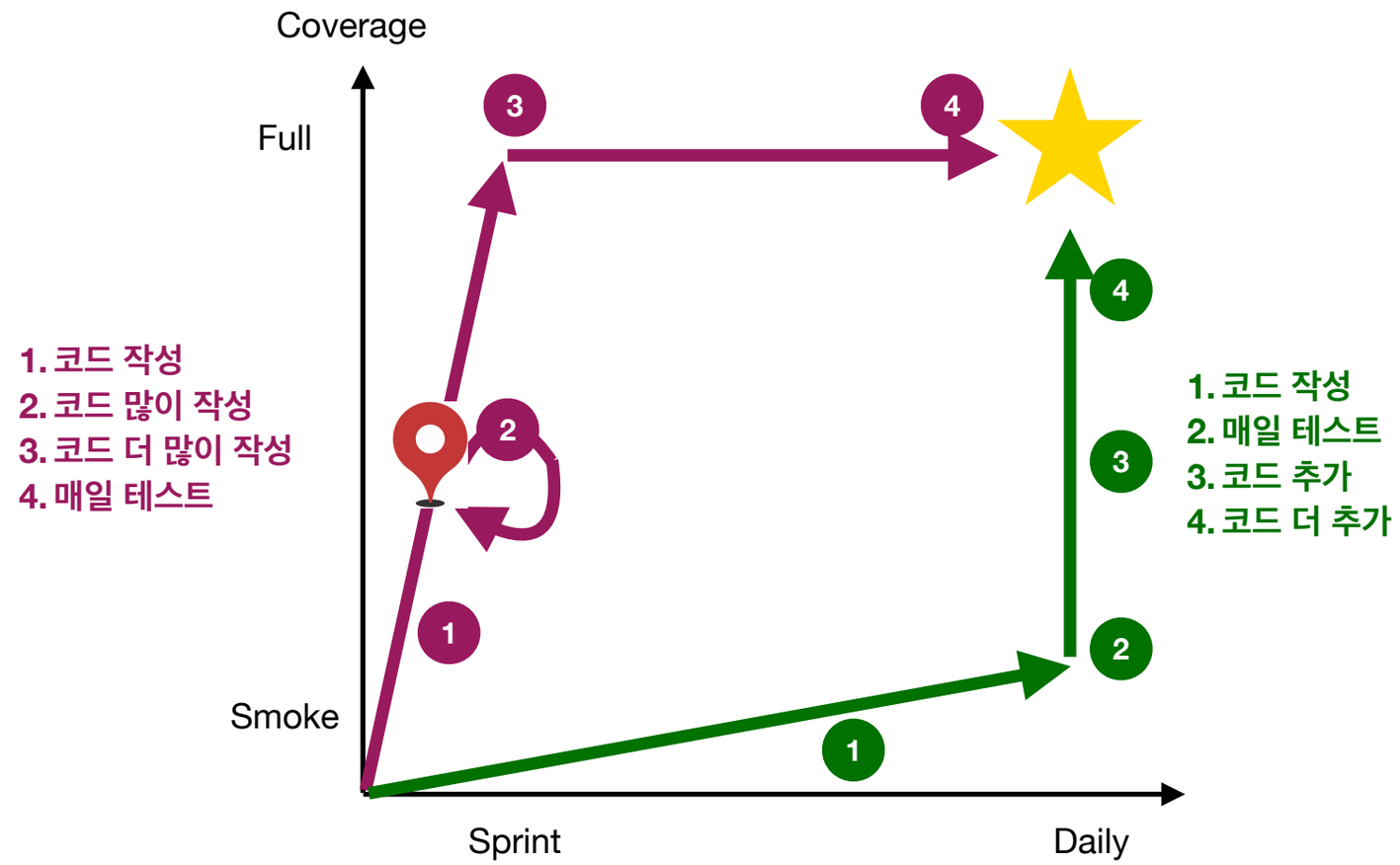
SW 검증 엔진의 제품화



분석 및 검증 엔진

다양한 기술의 총체

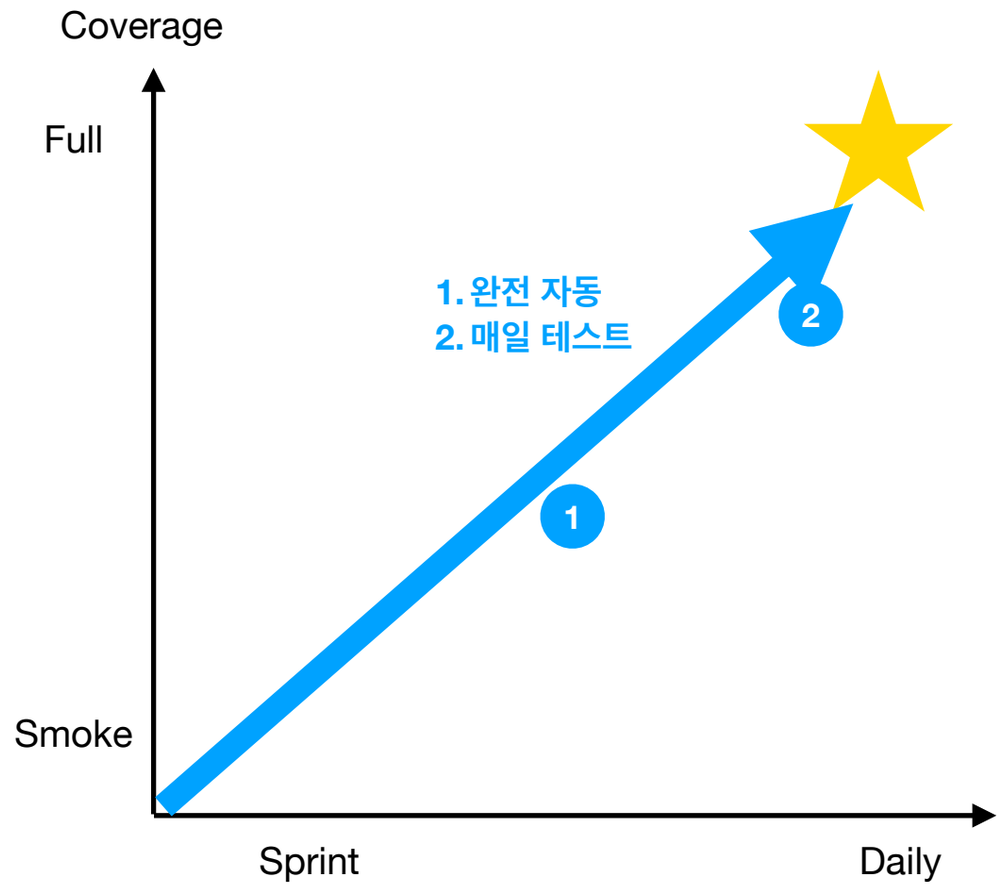
화이트박스 동적 테스팅 Unit Testing



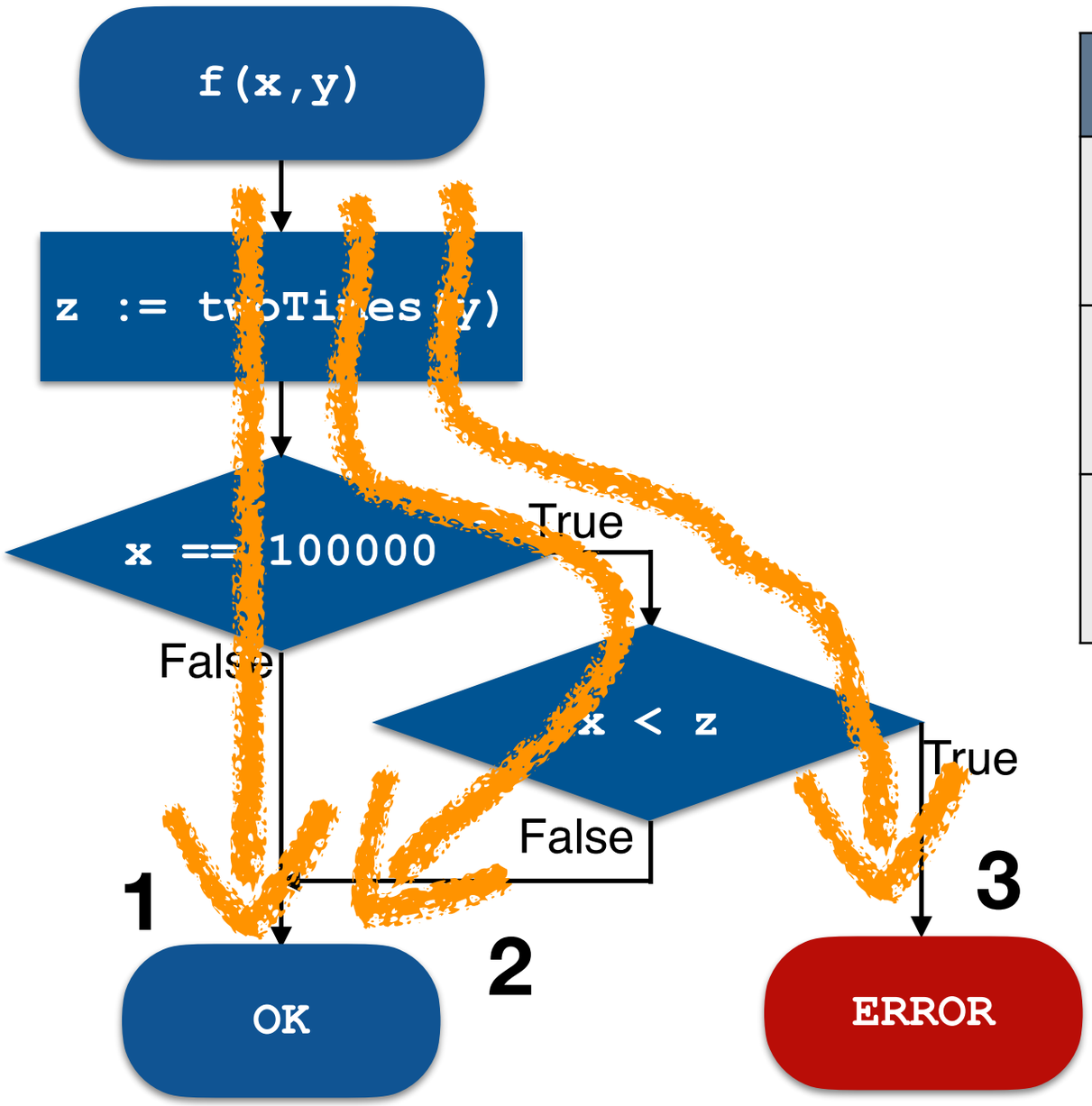
- 단위 테스트를 위한 수작업
  - 대상 함수 입력을 위한 테스트 케이스 작성
  - 대상 함수 호출을 위한 드라이버 코드 작성
  - 대상 함수가 호출하는 스텁 함수 작성
  - 복잡한 자료구조를 시뮬레이트하는 함수 작성

완전 자동 단위 테스트

- 완전 자동
  - 테스트 케이스 자동 생성
  - 드라이버 코드 자동 생성
  - 스텁 함수 자동 생성
  - 테스트 커버리지 100% 달성



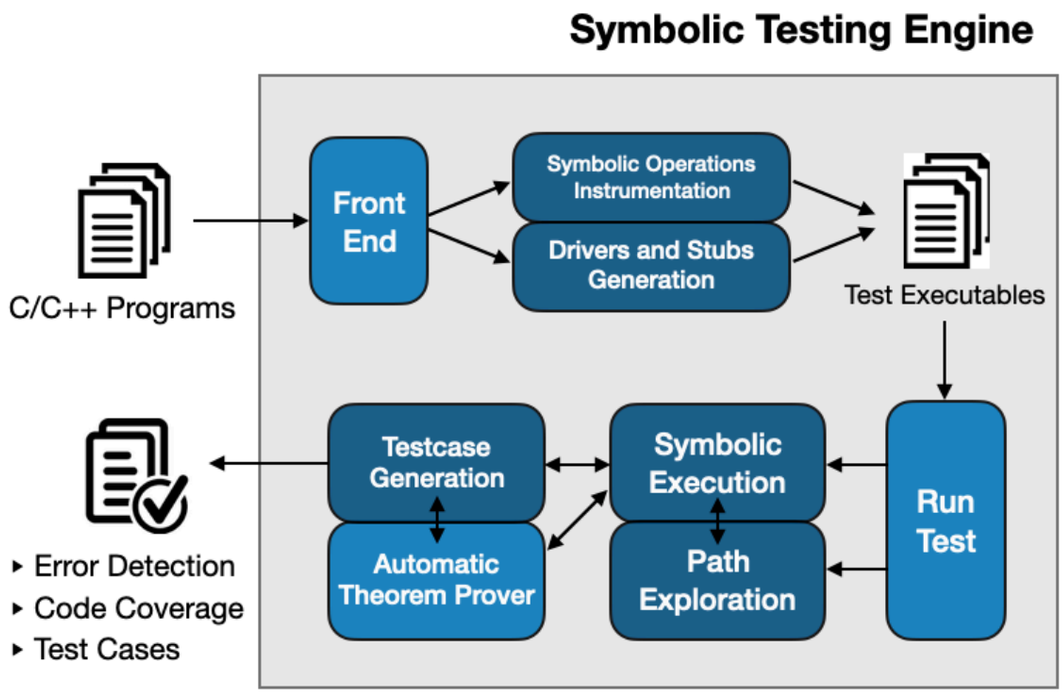
심볼릭 테스팅 Symbolic Testing



path	path formula	symbol
1	$x \neq 100000$	σ_1
2	$x = 100000 \wedge x \geq 2 \times y$	σ_2
3	$x = 100000 \wedge x < 2 \times y$	σ_3

심볼릭 테스팅

- 완전 자동 단위 테스트 도구
  - 콘콜릭 실행 + SMT 풀이 기반
  - 원클릭으로 코딩 없이 단위 테스트



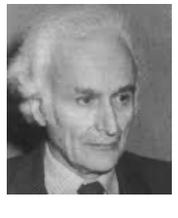
- 특징
  - LLVM 기반 콘콜릭 엔진
  - 소스코드 기반 드라이버/스텁 생성
  - 최적의 테스트케이스 생성
  - 머신러닝 기반 최적화
  - 명세 검증을 위한 테스트케이스 생성
  - 다양한 C/C++ 컴파일러를 위한 코드 변환
  - 설정 및 코드 추가 기능

논리와 수학 기반 컴퓨팅

- 초기의 생각
  - 프로그래밍을 수학적 행위로 규정하고 수리논리학으로 검증
  - Goldstine & von Neumann(1947), Turing(1949), Curry(1949), Böhm(1951)







- Herman Goldstine (1913 – 2004) & John von Neumann (1903 – 1957)
  - Planning and coding of problems for an electronic computing instrument, 1947
  - 전자 컴퓨팅 기기의 수학적, 논리적 측면을 제시 (flow diagram, assertion box)
- Alan Turing (1912 – 1954)
  - Checking a large routine, 1949
  - 명세와 구현을 구별하고 그 관계를 언급한 생각 (assertion, checker, proof)
- Haskell Curry (1900 – 1982)
  - On the composition of programs for automatic computing, 1949
  - 프로그램 모듈화, 타입 도입, 코드생성을 위한 수학적 접근(recursive code generation)
- Corrado Böhm (1923 – 2017)
  - Digital Computers:On encoding logical-mathematical formulas using the machine itself during program conception, 1951
  - 컴파일러, 프로그래밍 언어, 코드 생성, 보편 튜링기계와 동치인 추상기계 (three-address programmable computer)

논리적이고 수학적인 PL 연구

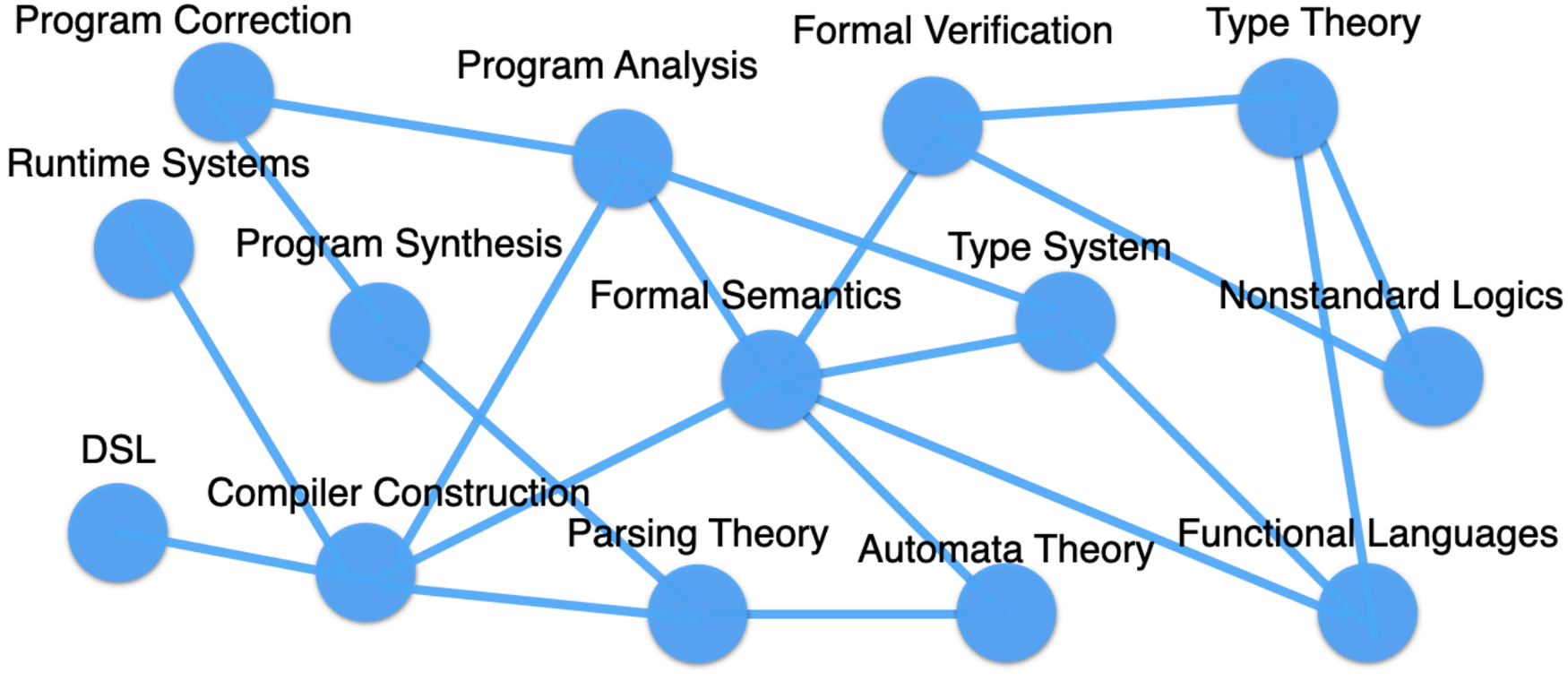
컴퓨터과학이 과학인가요?

- John McCarthy (1927 – 2011)
  - A basis for a mathematical theory of computation, 1961
  - Towards a mathematical science of computation, 1962
- Peter J. Landin (1930 – 2009)
  - The mechanical evaluation of expressions, 1964
- Robert W. Floyd (1936 – 2001)
  - Assigning meanings to programs, 1967
- Christopher Strachey (1916 – 1975)
  - Towards a formal semantics, 1966
  - Fundamental concepts in programming languages, 1967
- Tony Hoare (1934 –)
  - An axiomatic basis for computer programming, 1969

PL 연구 패러다임

지난 백 년 동안 물리학과 해석학의 관계가 맺은 결실 만큼이나 많은 성과를
다음 세기에는 컴퓨터 과학과 수리논리학의 관계에서 얻어질 것이다.
이를 위해서는 수학적 우아함과 응용력 모두에 관심을 기울여야 한다.

John McCarthy(1961)

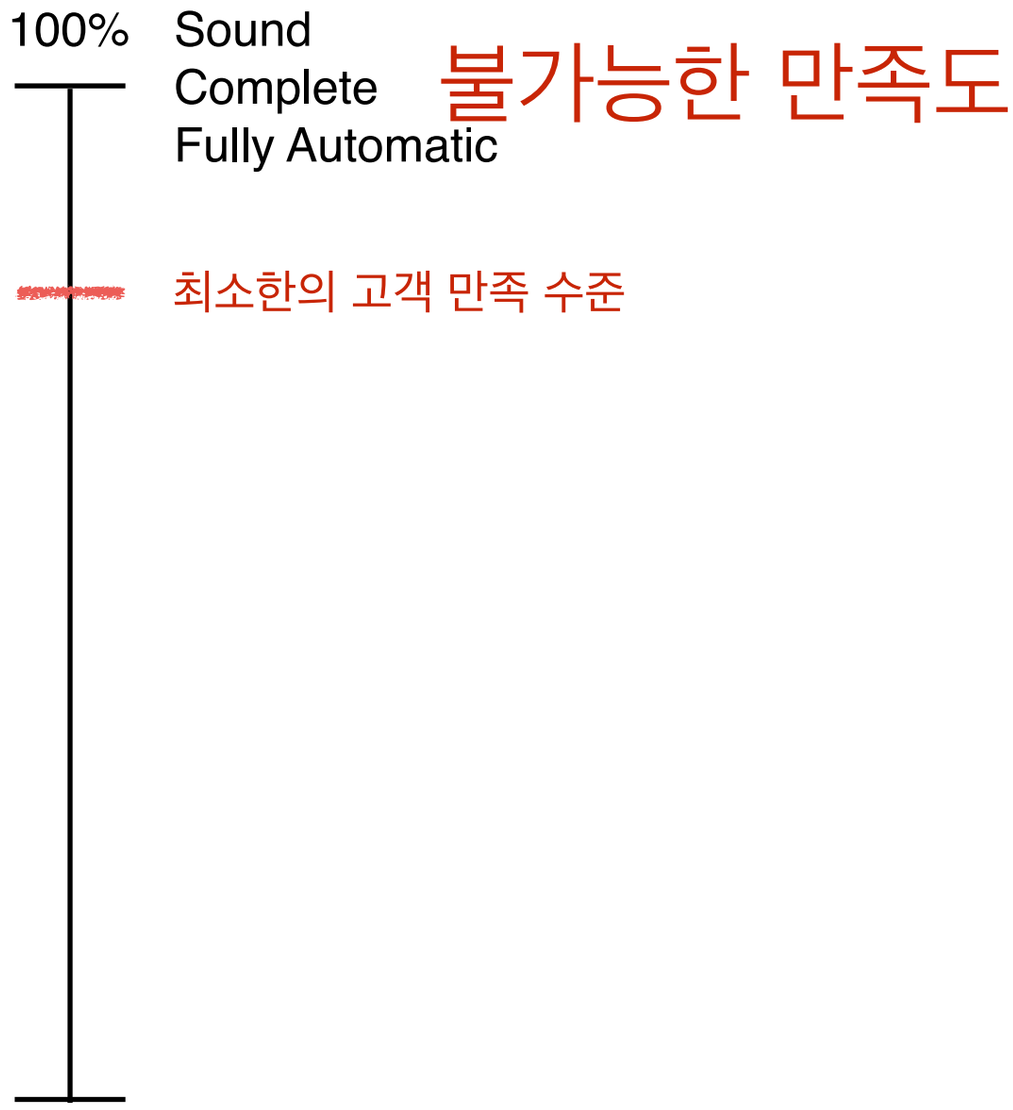


난 반댈세

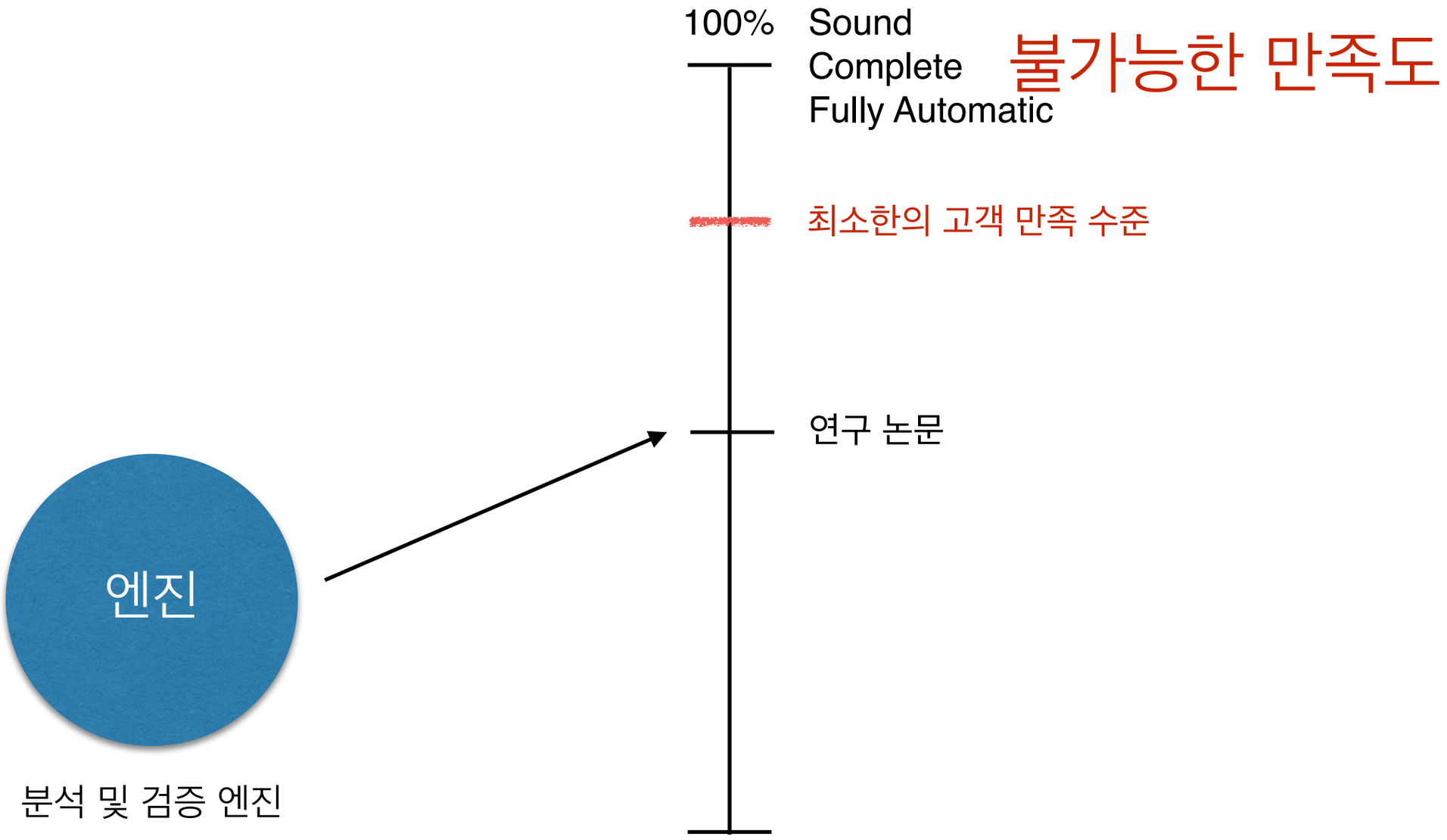
- 프로그램은 검증될 수 없다.
  - 프로그램은 외부적인 컴퓨팅 환경에 영향을 받는 동적 개체임
  - 정형기법 자체도 근본적인 한계를 가짐
- Henri Rice
  - Classes of Recursively Enumerable Sets and Their Decision Problems, 1953
  - 프로그램 분석과 검증의 이론적인 한계를 증명
- Richard DeMillo, Richard Lipton, Alan Perlis
  - Social Processes and Proofs of Theorems and Programs, 1979
  - 프로그램 검증도 수학 증명과 같이 사회적 검증 과정이 필요함을 주장
- James Fetzer
  - Program Verification: The Very Idea, 1988
  - 프로그램 검증이 수학 검증과 다르게 논리적, 물리적 한계로 비현실적임을 주장

분석 검증의 실용화

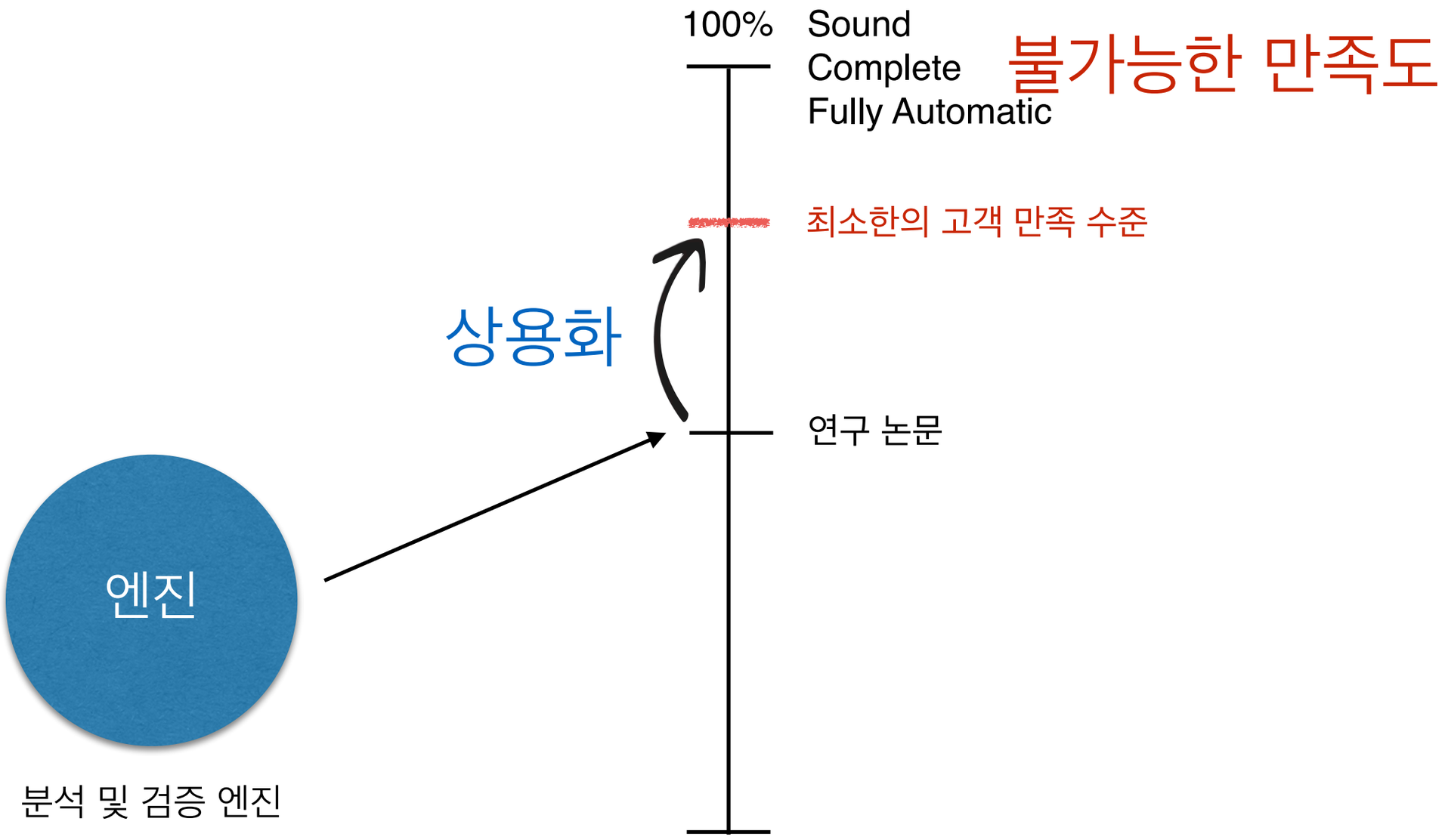
분석 검증의 실용화



분석 검증의 실용화



분석 검증의 실용화



상용화를 위한 엔지니어링

- 드라이버 작성
  - 가상 함수, 함수 포인터, 널 포인터 분석
  - 루프 횟수 최적화
  - 재귀 데이터 초기화
  - void* 타입 추론

- LLVM & Clang 수정
  - 소스코드와 LLVM 매칭
  - 타입정보, 가상메소드 테이블 정보 유지

- 머신러닝 기반 최적화
  - 자원 할당 (Concolic Execution vs SMT Solving)
  - 경로 탐색 전략, 테스트케이스 생성 개수, 병렬처리 개수
  - 데이터 구조 초기화의 길이와 깊이, 루프 최대 횟수, 객체값 크기

- 벤치마크 기반 최적화
  - 2백만개 이상 프로젝트, 144억 라인
  - 프로그램 특성에 대한 통계
  - 각 프로그램의 특성 분석에 따른 대응

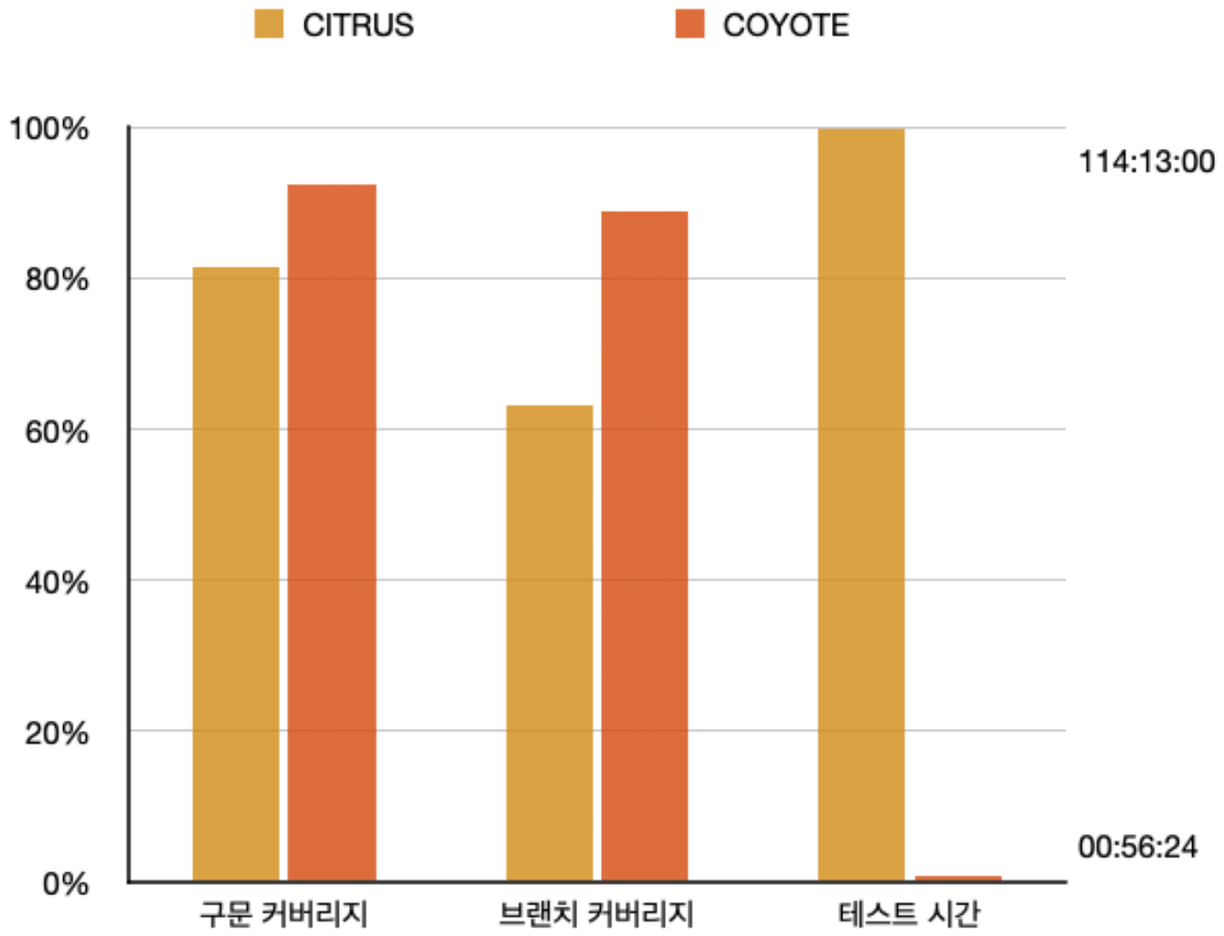
- SMT solver 최적화
  - 반복 패턴 경로조건 요약
  - 비선형 연산 기호식 근사값 구하기

실용적인 결과

총 8 개 프로젝트 145,141 LOC

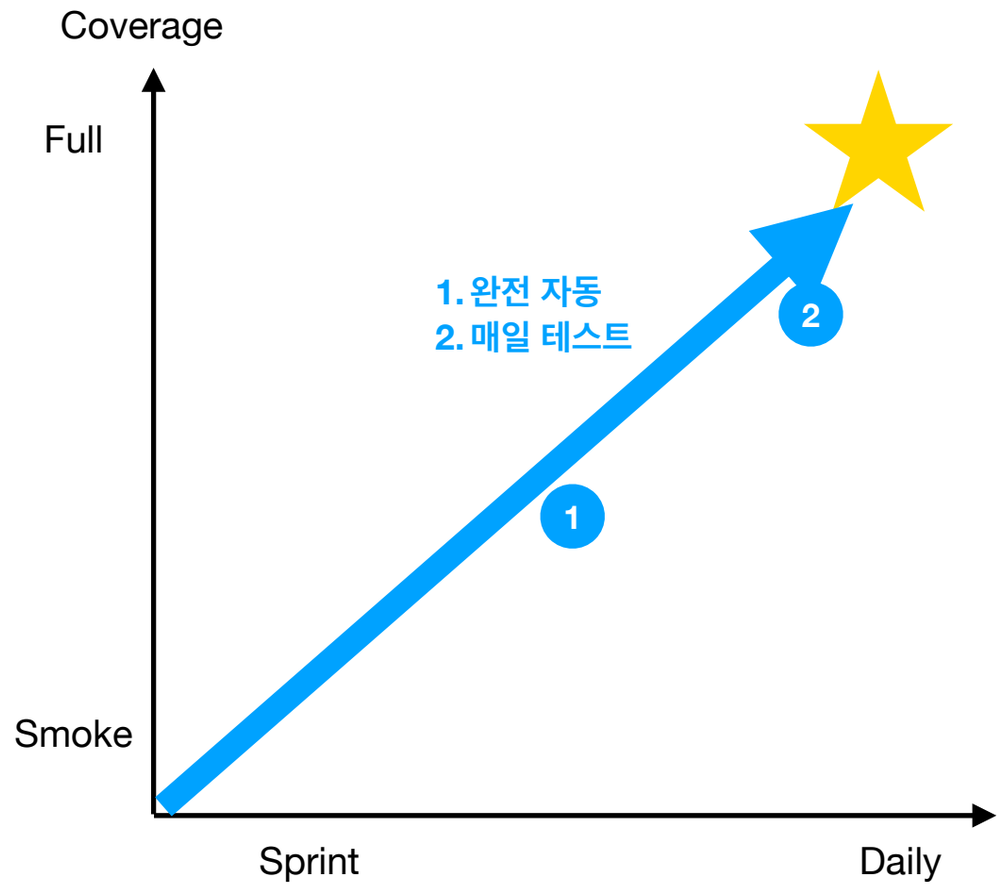
	구문 커버리지	브랜치 커버리지	테스트 시간
CITRUS*	81.39%	63.06%	114:13:00
COYOTE	92.34%	88.85%	00:56:24

*ICST 2022, CITRUS: Automated Unit Testing Tool for Real-world C++ Programs



완전 자동 단위 테스트

- 완전 자동
  - 테스트 케이스 자동 생성
  - 드라이버 코드 자동 생성
  - 스텁 함수 자동 생성
  - 테스트 커버리지 100% 달성

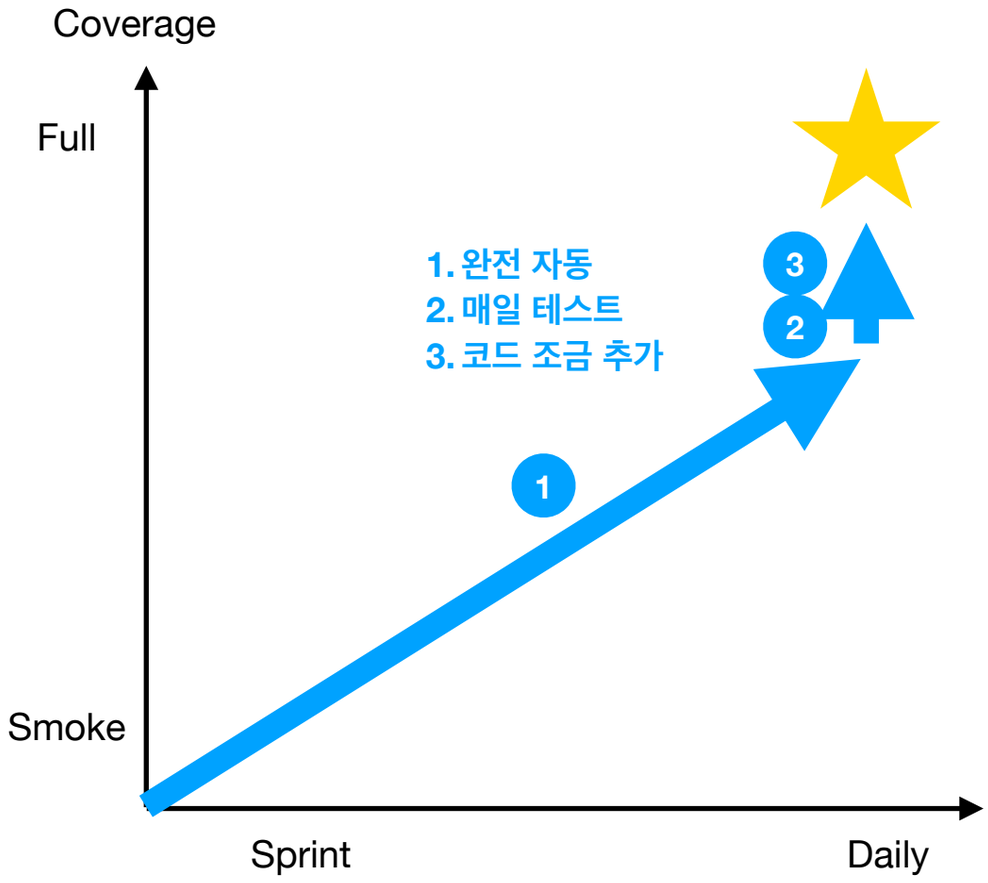


완전 자동 단위 테스트

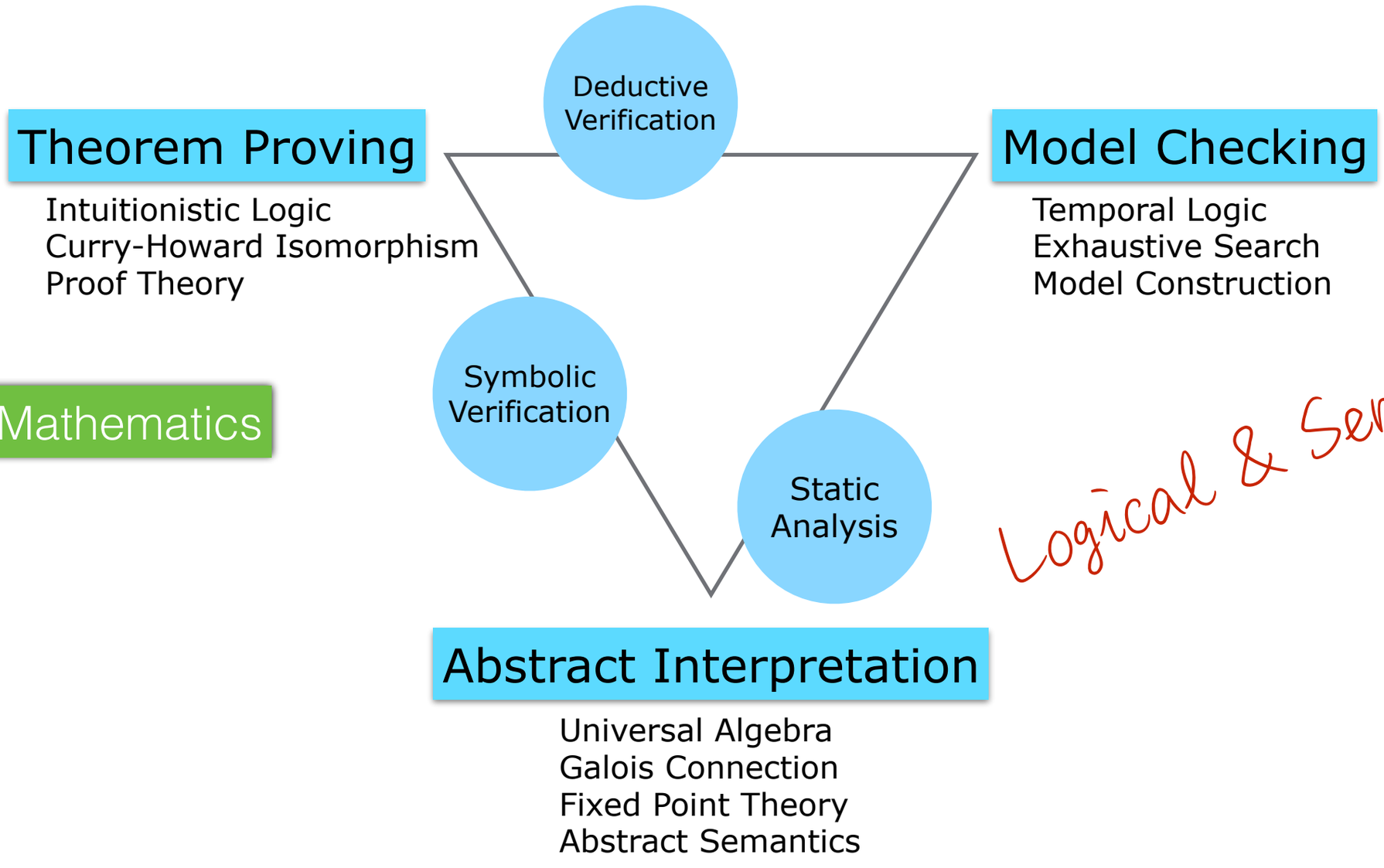
COYOTE
Fully Automated Testing Tool

- 테스트 케이스 자동 생성
- 드라이버 코드 자동 생성
- 스텁 함수 자동 생성
- 테스트 커버리지 90% 이상 달성

- 자동차, 국방 분야 적용 가능



논리·수학 기반 분석·검증



Formalized Mathematics

정리

- 논리와 시맨틱스를 기반하는 PL 연구
- SW 분석 검증 기술의 상용화
- PL 연구의 패러다임
  - 논리 기반 엄밀한 연구 vs 경험적이고 통계적인 AI 기반 연구
  - 기술의 대중화: 한계의 이해가 아니라 한계의 제거
  - 이성과 경험의 융합: 화이트박스 기술 + 블랙박스 기술

20여 년 전