

스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발			
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

스마트 컨트랙트를 위한 정형기법

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

〈제목 차례〉

I. 목적	3
II. 용어 정의 및 약어	5
III. 스마트 컨트랙트 개발의 주요 요구사항	6
가. 스마트 컨트랙트의 핵심 특성	6
나. 대표적인 스마트 컨트랙트 보안약점 유형	8
다. 테스트 기반 검증의 한계와 정형 기법의 보완	12
IV. 스마트 컨트랙트를 위한 정형 기법 개념	14
가. 정형 기법의 원리	14
나. 상태 기반, 행위 기반, 논리 기반 모델링의 유형과 비교	14
다. 스마트 컨트랙트 모델의 주요 요소	16
V. 스마트 컨트랙트 정형 명세 방법	19
가. TLA+	19
나. Alloy	20
다. Coq / Isabelle / Lean (인터랙티브 정리 증명)	22
라. VeriSol	23
마. 심볼릭 실행 / SMT 기반 모델링	25
VI. 스마트 컨트랙트 정형 검증 기법	27
가. 모델 검증	27
나. 정리 증명	28
다. 추상 해석	29
라. 심볼릭 실행	30
마. 퍼징	31
VII. 스마트 컨트랙트 정형 기법 적용 사례	32
가. Ethereum 토큰 컨트랙트에 대한 정형 기법	32
나. DeFi 시스템에 대한 정형 검증 적용	33
다. SLA 기반 계약의 정형 기법	34
라. 대출 시스템에 대한 정형 기법	36
마. 스마트 컨트랙트 Security Policy Model (SPM)에 대한 정형 기법	37
VIII. 스마트 컨트랙트 정형 기법 도구 및 프레임워크	40
가. TLA Toolbox	40
나. KEVM	41
다. VeriSol	43
라. Mythril	44
마. Slither	46
바. 통합 전략	47

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

I. 목적

본 문서는 스마트 컨트랙트 개발 및 운영 환경에서 발생할 수 있는 다양한 오류와 보안 취약점을 사전에 탐지하고 제거하기 위해, 정형 기법 및 검증 기법을 효과적으로 적용하는 방법론을 체계적으로 정리하는 데 목적이 있다. 이 기술문서는 스마트 컨트랙트의 설계 단계부터 배포, 운영 전 과정에서 활용될 수 있는 정형 명세 작성, 모델 체킹 기법, 정리 증명, 심볼릭 실행 등 다양한 기법의 이론적 배경과 실제 적용 사례를 종합적으로 다루고 있다. 또한, 이를 통해 스마트 컨트랙트의 안전성과 신뢰성을 극대화하는 동시에, 개발자와 보안 전문가들이 보다 체계적이고 과학적인 접근법을 적용할 수 있도록 돋는 것을 주요 목표로 한다.

스마트 컨트랙트는 블록체인 기술의 핵심 구성 요소로, 계약의 이행을 자동화하고 제3자 없이도 신뢰성 있는 거래를 보장할 수 있는 혁신적인 도구이다. 전통적인 계약과 달리, 스마트 컨트랙트는 코드로 작성되어 배포된 후에는 변경이 불가능하며, 이로 인해 한 번 발생한 오류나 취약점이 심각한 경제적 손실로 이어질 수 있다. 이러한 특성은 스마트 컨트랙트가 금융, 공급망, 디지털 자산 관리 등 다양한 분야에서 핵심적인 역할을 수행하게 만드는 동시에, 매우 높은 수준의 보안과 검증이 필요함을 의미한다.

특히, 스마트 컨트랙트는 분산 환경에서 여러 노드에 의해 동시에 실행되기 때문에, 결정성과 불변성이 확보되어야 한다. 모든 참여자가 동일한 상태 전이를 경험해야만 전체 시스템의 합의가 유지될 수 있으며, 한 번 배포된 코드가 수정 불가능하다는 특성 때문에 초기 단계에서의 오류는 돌이킬 수 없는 결과를 초래할 수 있다. 이와 같은 이유로, 스마트 컨트랙트는 높은 신뢰성과 투명성을 요구하며, 이를 달성하기 위한 검증 과정은 단순한 테스트나 코드 리뷰를 넘어서는 정형적인 방법론을 필요로 한다.

더불어, 스마트 컨트랙트가 다양한 산업 분야에서 적용되면서 그 응용 범위도 급격하게 확장되고 있다. 예를 들어, 탈중앙 금융(DeFi) 분야에서는 대규모 자산이 스마트 컨트랙트에 의해 관리되며, 재진입 공격이나 정수 오버플로우와 같은 보안 취약점이 발생할 경우 심각한 손실로 이어질 수 있다. 또한, 공급망 관리나 디지털 자산 거래 등에서 스마트 컨트랙트는 투명성과 신뢰성을 보장하는 중요한 역할을 하지만, 한편으로는 이러한 시스템의 복잡성 때문에 잠재적인 오류와 공격 벡터가

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

증가하는 문제점도 안고 있다. 따라서, 스마트 컨트랙트의 중요성은 단순한 기술적 혁신을 넘어서, 실질적인 경제적, 사회적 영향을 미치는 만큼, 그 보안과 신뢰성 확보를 위한 철저한 검증 과정이 필수적이다.

스마트 컨트랙트는 단순한 소프트웨어와는 달리, 배포 후 수정이 어려운 특성을 지니고 있으며, 한번의 오류가 전체 시스템의 신뢰성을 크게 훼손할 수 있다. 따라서, 초기 설계 단계에서부터 시스템의 상태 전이, 계약 조건, 보안 취약점 등을 수학적 명세와 논리적 증명을 통해 철저하게 검증하는 정형 기법이 필수적으로 요구된다. 정형 기법은 자연어로 표현된 요구사항이나 설계 개념을 엄밀한 수학적 모델로 변환하여, 모든 가능한 실행 경로와 예외 상황을 체계적으로 분석할 수 있게 한다. 이는 전통적인 테스트 기반 검증 방식이 가지는 한계를 극복할 수 있는 강력한 도구로 평가된다.

정형 기법을 적용함으로써, 개발자는 스마트 컨트랙트의 모든 상태와 전이 과정을 명확하게 정의하고, 불변 조건과 핵심 계약 조건이 항상 유지됨을 증명할 수 있다. 예를 들어, “총 토큰 공급량은 항상 일정하다”거나 “잔액은 음수가 될 수 없다”는 조건을 수학적 명세로 표현하고, 이를 기반으로 자동 모델 체킹 기법이나 정리 증명 도구를 통해 모든 실행 경로에서 해당 조건이 위배되지 않음을 보장할 수 있다. 이러한 과정은 단순한 코드 리뷰나 단위 테스트로는 포착하기 어려운 미세한 오류나 예외 상황을 사전에 제거하는 데 결정적인 역할을 한다.

또한, 정형 기법은 스마트 컨트랙트와 같이 복잡하고 민감한 시스템의 보안 검증에 있어 투명성을 극대화할 수 있는 수단이다. 정형 명세를 통해 시스템의 동작을 수학적으로 증명하면, 외부 감사나 보안 평가 시 독립적인 검증 자료로 활용될 수 있으며, 이는 사용자와 투자자에게 높은 신뢰를 제공하는 데 크게 기여한다. 결국, 정형 기법은 스마트 컨트랙트의 설계 및 검증 단계에서 발생할 수 있는 모든 잠재적 오류와 보안 취약점을 사전에 탐지하고 제거할 수 있는 체계적인 방법론으로, 향후 스마트 컨트랙트의 안정적 운영을 위한 핵심 기술로 자리매김할 것이다.

본 문서는 향후 스마트 컨트랙트 보안 및 검증 분야의 연구 방향과 발전 가능성을 모색하는 데에도 기여하고자 한다. 기존의 단순 테스트 및 코드 리뷰 방식의 한계를 극복하고, 정형 기법을 통한 수학적 증명과 자동화된 검증 시스템이 어떻게 실질적인 보안 강화와 신뢰성 향상에 이바지할 수 있는지를 상세히 기술함으로써, 연구자 및 실무자 모두에게 유용한 참고 자료로 활용될 수 있도록 하는 것을 궁극적인 목적으로 한다.

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

할 수 있다.

II. 용어 정의 및 약어

1. SLA (Service Level Agreement) : 서비스 제공자와 사용자 간의 서비스 품질, 가용성, 성능 수준 등을 사전에 정의한 계약으로, 위반 시 자동 보상 또는 제재 조건이 포함됨
2. Smart Contract : 블록체인에서 사전 정의된 조건이 충족되면 자동으로 실행되는 코드로, 계약의 신뢰성과 자동화를 보장함
3. Formal Method (정형 기법) : 시스템의 요구사항이나 동작을 수학적 논리로 표현하고 분석하는 방법론으로, 오류 가능성은 사전에 차단함
4. Formal Specification (정형 명세) : 시스템의 구조, 동작, 제약 조건을 수학적 언어로 기술한 문서로, 검증의 기준이 되는 참조 명세임
5. Formal Verification (정형 검증) : 정형 명세에 따라 시스템이 올바르게 동작하는지 수학적으로 증명하는 절차로, 모델 검증, 정리 증명 등이 포함됨
6. Invariant (불변식) : 시스템이 어떤 상황에서도 항상 만족해야 하는 조건 또는 규칙
7. Oracle : 스마트 컨트랙트가 외부 현실 세계의 데이터를 가져오기 위한 데이터 공급자 역할을 하는 메커니즘
8. Runtime Verification : 스마트 컨트랙트가 실행된 이후, 실제 트랜잭션과 상태 변화를 실시간으로 감시하여 이상 동작을 감지하는 기법
9. Weakness (보안 약점) : 소프트웨어 내 포함되는 bug, flaw, mistake, error 등 모든 비정상 코드
10. Vulnerability (취약점) : 위협으로 악용되어 자산에 무단으로 접근할 수 있는 프로그램의 약점

스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발			
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

III. 스마트 컨트랙트 개발의 주요 요구사항

스마트 컨트랙트는 블록체인 상에서 자동으로 실행되는 프로그램으로, 높은 신뢰성과 안전성이 요구된다. 이를 위해 결정성, 불변성, 자동성, 보안성 등의 핵심 특성을 갖춘다. 이러한 특성들은 스마트 컨트랙트의 동작을 이론적으로 보장하는 토대로 작용하며, 실무에서 오류를 예방하는 데 매우 중요하다. 본 장에서는 각 특성의 이론적 의미와 실제 사례에서의 중요성을 살펴보고, 대표적인 취약점 유형들과 발생 조건, 사례, 방지 전략을 분석한다. 아울러 전통적인 테스트 기반 검증의 한계를 짚어보고 정형 검증이 이를 어떻게 보완하는지 논의한다.

가. 스마트 컨트랙트의 핵심 특성

1. 결정성

결정성이란 동일한 입력에 대해 항상 동일한 결과를 산출하는 성질로서, 분산 환경에서 모두가 같은 결과를 얻도록 하는 스마트 컨트랙트의 필수 조건이다. 블록체인 네트워크의 모든 노드가 스마트 컨트랙트 코드를 실행할 때, 결정성이 보장되어야만 네트워크 합의에 도달할 수 있다. 만약 한 노드는 어떤 트랜잭션 결과로 10을 얻고 다른 노드는 12를 얻는다면 합의가 깨져 분기가 발생할 것이다. 따라서 이론적으로 스마트 컨트랙트 언어와 런타임은 난수나 비결정적 함수를 제한하여 결정성을 강제한다. 결정성이 확보되면 계약의 실행 결과가 예측 가능해지며, 이는 사용자 신뢰성과 디버깅 용이성 측면에서 중요하다. 예를 들어 이더리움의 스마트 컨트랙트는 블록해시, 타임스탬프 등의 환경 변수도 제한된 범위 내에서만 사용 가능하게 하여 노드 간 실행결과 불일치를 막는다. 일반적으로 결정성 자체가 깨지는 경우는 드물지만, 과거 특정 컨트랙트가 미정의 동작을 이용하려 하다가 노드별 상이한 결과로 이어진 약점 사례들이 보고된 바 있다. 이를 방지하기 위해 개발자는 시간, 무작위성에 의존하지 않고, 외부 의존성이 있는 연산을 스마트 컨트랙트 내에서 수행하지 않아야 한다.

2. 불변성

불변성은 한 번 블록체인에 배포된 계약 코드를 변경할 수 없는 속성을 말한다. 스마트 컨트랙트 코드와 상태는 블록체인 원장에 영구 기록되며, 누구도 이를 임의로 수정할 수 없다. 이론적으로 불변성은 계약 실행의 무결성을 담보하며, 신

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

뢰 기반이 없는 환경에서도 코드가 곧 법률처럼 작동하도록 한다. 불변성 덕분에 배포된 계약은 외부 조작이나 검열 없이 약속된 대로 동작하므로, 참여자들은 코드에 대한 신뢰만으로 거래할 수 있다. 예컨대 탈중앙화 금융(DeFi)에서는 프로토콜 코드가 불변임을 전제로 수억 달러의 자산이 맡겨진다. 그러나 이 특성 때문에 오류 발생 시 치명적 결과를 초래하기도 한다. 반면, 2016년 발생한 DAO 해킹은 불변성의 양날의 검을 보여준다. DAO 스마트 컨트랙트에 치명적인 버그가 있었지만, 코드가 불변이라 개발팀이 즉각 수정할 수 없었다. 결국 공격자는 약 \$5,000만 상당의 Ether를 탈취했고, 이를 되돌리기 위해 이더리움 네트워크가 하드포크라는 극단적 조치를 취해야 했다. 또 다른 사례로 2017년 Parity 멀티시그 지갑 사고에서는, 라이브러리 컨트랙트의 버그로 인해 \$1.5억 상당 Ether가 동결되었는데, 불변성 때문에 해당 코드를 고쳐 자금 반환이 불가능하게 되었다. 이처럼 불변성은 신뢰 기반을 제거하는 장점과 함께 배포 전 철저한 검증 필요성이 요구된다.

3. 자동성

자동 실행은 스마트 컨트랙트가 사전에 정의된 규칙에 따라 제 3자의 개입 없이 자동으로 실행된다는 특성이다. 즉 트랜잭션에 의해 호출되면 조건문 검증과 상태 업데이트, 이벤트 발생 등의 일련의 동작을 계약이 스스로 수행되며, 계약 조건이 충족되면 자동으로 효력이 발생하도록 한다. 자율 실행 덕분에 중재자나 중앙 시스템 없이도 계약의 집행이 가능해져 운영 비용을 절감하고 투명성을 높인다. 예를 들어 어떤 지급 조건을 코딩해두면, 그 조건이 만족되었을 때 즉시 결제가 이행되고 블록체인에 기록된다. 사람이 개입했다면 실수나 지연이 있을 수 있지만, 코드 자동 실행은 엄밀하고 빠르다. 반면, 자동성의 특성으로 인한 사고 사례도 존재한다. DAO 사례에서 컨트랙트는 악의적인 재귀 호출에 자동으로 응답하여 계속 Ether를 보냈고, 이를 중단시킬 방법이 없었다. 또 2020년 DeFi 프로토콜에서 이자 계산을 자동화한 컨트랙트 버그로 올바르지 않은 계정에 과다 보상이 지급된 사례가 있다. 계약이 자동으로 실행되기에 한번 잘못 배포되면 오작동 역시 자동으로 계속되어 피해가 커질 수 있으므로, 사전 검증과 제어 장치가 중요하다.

4. 보안성

보안성은 스마트 컨트랙트의 가장 중요한 실용적 요구사항으로, 코드에 악용 가능한 취약점이 없고 자산이나 데이터가 안전하게 관리되는 것을 의미한다. 스마

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

트 컨트랙트는 신뢰 최소화를 목표로 하지만, 코드 자체에 대한 절대적인 신뢰를 필요로 한다. 따라서 코드 논리에 버그나 취약점이 있으면 공격자는 이를 악용하여 금전적 이익을 취하거나 시스템을 교란할 수 있다. 수많은 DApp과 DeFi 프로젝트에서 천문학적 가치의 암호화폐가 스마트 컨트랙트에 담기며, 보안성의 문제가 발생할 경우 막대한 재산 피해로 이어진다. 예를 들어, 2020~2022년 사이 여러 DeFi 플랫폼들이 스마트 컨트랙트 취약점으로 수백만 달러 규모의 해킹 피해를 입었다. 보안성을 확보하려면 개발 단계부터 공격 벡터들을 고려하고, 정적 분석, 동적 테스트, 외부 보안 감사, 정형 검증 등 다중적인 접근법으로 보안 약점을 제거해야 한다.

나. 대표적인 스마트 컨트랙트 보안약점 유형

1. 재진입 공격

1) 발생 조건

재진입 공격은 스마트 컨트랙트가 외부로 데이터를 전송하기 위해 다른 컨트랙트의 함수를 호출할 때, 외부 컨트랙트가 다시 원래 컨트랙트의 함수를 재귀적으로 호출(re-enter)할 수 있을 때 발생한다. 특히 상태를 변경하기 전에 외부 호출을 수행하면, 공격자가 그 틈을 노려 원래 함수가 끝나기 전에 다시 진입해 로직을 반복 실행시킬 수 있다.

2) 실제 사례

2016년 DAO 해킹이 고전적인 재진입 공격 사례다. DAO 컨트랙트의 withdraw() 함수는 잔고를 줄이기 전에 외부 송금을 수행했는데, 악의적인 공격자가 컨트랙트 함수에 재진입하여 잔고가 차감되기 전에 반복 출금함으로써 전체 자금을 탈취했다. 이후로도 2020년 Lendf.Me에서 약 \$2,500만 피해를 입은 사례가 있으며, 2021년 Cream Finance에서 \$1,800만, 2022년 Fei Protocol \$8,000만을 피해 입는 사례 등 재진입 공격으로 인한 피해 사례가 적지 않다. 이러한 사례들이 지속적으로 발생하는 이유는, 재진입 상황이 여러 함수와 컨트랙트를 걸쳐 예기치 않은 상호작용으로 나타나며, 경우의 수가 복잡해 사람이 직접 확인하거나 일반적인 테스트로 방식으로는 모든 시나리오를 놓치지 않고 검증하기 어렵기 때문이다.

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

3) 방지 전략

재진입 공격 방지 전략의 핵심은 외부 호출 전후로 상태 변경 순서를 올바르게 두는 것이다. 일반적으로 체크-이펙트-상호작용 패턴 (Checks-Effects-Interactions)을 따라, 외부에 Ether를 보내기 전에 컨트랙트 자신의 상태를 먼저 업데이트하여 재진입 여지를 없앤다. Solidity에서는 OpenZeppelin 라이브러리의 ReentrancyGuard와 같은 수단을 이용해 함수 진입을 1회로 제한할 수도 있다. 또한 가급적 저수준 호출로 Ether를 보내기보다 transfer나 send를 사용해 재진입 시도를 기본적으로 막는 것도 권장되었으나, 이후 call을 더 선호하는 추세에서는 반드시 재진입 락 등을 병행한다. 마지막으로 재진입 가능성이 있는 외부 호출을 최소화하고, 필요한 경우 풀 방식으로 외부에 토큰이나 Ether 인출을 맡기는 디자인도 고려된다.

2. 정수 오버 플로우, 언더 플로우

1) 발생 조건

정해진 비트 크기의 정수형 변수가 표현할 수 있는 범위를 넘는 값으로 연산을 수행할 때 오버플로우 또는 언더플로우가 발생한다. EVM의 과거 버전에서는 정수 오버/언더플로우 시 자동으로 값이 래핑되어 overflow의 경우 최소값으로, underflow의 경우 최대값으로 순환되었다. 공격자는 이러한 언어 특성을 악용해 의도치 않은 큰 값을 만들어내거나 조건식을 우회할 수 있다.

2) 실제 사례

2018년 발견된 BatchOverflow 보안 약점은 토큰 스마트 컨트랙트에서 연산을 할 때 값이 매우 큰 값이면 값이 극대화되어, 마음대로 토큰을 만들어낼 수 있었던 사례다. 이 버그로 다수의 ERC20 토큰에서 무제한 발행 사고가 발생했다. 또한 과거 일부 ICO 계약에서 안전한 수학 연산을 쓰지 않아 기부금 합산 과정에서 오버플로우가 발생, 목표 금액 검증이 무력화된 사례도 있다.

3) 방지 전략

가장 확실한 방법은 최신 Solidity 컴파일러를 사용하여 기본적으로 오버/언더 플로우 시 예외를 발생시키도록 하는 것이다. 구버전 호환이 필요하다면 OpenZeppelin의 SafeMath 라이브러리를 사용하여 모든 산술 연산에 대해 오버플

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

로우를 체크하고 처리한다. 더불어 연산 전에 값의 범위를 명시적으로 검증하는 방어적 프로그래밍도 병행한다. 정형적인 관점에서는 변수의 불변 조건을 수학적으로 모델링하여, 임의의 연산 후에도 그 범위가 유지됨을 증명함으로써 오버플로우 발생 여부를 검증할 수 있다.

3. 경쟁 상태 및 프론트 러닝

1) 발생 조건

경쟁 상태는 동일한 자원을 두고 여러 트랜잭션이나 연산이 동시에 실행되려 할 때 발생하는 문제로, 블록체인 맥락에서는 트랜잭션 순서 의존성 또는 프론트 러닝 형태로 주로 나타난다. 이더리움에서는 트랜잭션들이 순차적으로 처리되지만, 여러 사용자가 거의 동시에 함수를 호출하거나, 한 트랜잭션이 완료되기 전에 다른 트랜잭션이 그 상태를 예상하고 행동하면 논리적 경쟁이 벌어진다.

2) 실제 사례

프론트러닝은 대표적인 경쟁 상황으로, 공격자가 mempool에서 남의 트랜잭션 내용을 보고 가스 값을 더 높여 자기 거래를 먼저 포함시킴으로써 이득을 취하는 방식이다. 예를 들어 탈중앙 거래소에서 A가 토큰을 싸게 사려는 주문을 낸 것을 본 공격자 B가 즉시 동일 주문을 높은 수수료로 앞질러 실행, 가격을 올린 뒤 A에게 비싼 가격을 형성하게 만드는 일이 있다. 또 다른 형태로, 경매 계약에서 마감 직전에 남아 높은 입찰을 넣으려는 것을 눈치채고 자신의 거래를 먼저 넣어 경매를 조기 종료시켜버리는 경우도 있다. 경쟁 상태 버그는 멀티서명 지갑에서 동시 서명 요구 조건이 어긋난다거나, 이더리움 난이도 조정 같은 시스템 수준에서 복잡한 타이밍 이슈로 나타나기도 한다.

3) 방지 전략

우선 프론트러닝에 대비하여 commit-reveal를 활용할 수 있다. 사용자가 어떤 중요한 입력을 먼저 해시 키밋하고 나중에 공개하는 2단계 처리를 통해, 선행자가 내용을 알 수 없도록 한다. 또한 DEX 거래에서는 slippage 제한을 두어 가격이 급변하면 자동 취소되게 하여 프론트러닝 이익을 제한하기도 한다. 순서 의존적 로직을 작성할 때는 한 트랜잭션 내에서 원자적으로 처리하거나, 외부로부터 의존하지 않도록 한다. 이더리움 같은 공개 mempool 체계에서는 완전한 경쟁 상태 해

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

소가 불가능하므로, 민감한 작업에는 Flashbots와 같은 비공개 거래 시스템을 이용해 프론트러닝을 회피하는 기법도 실무에서 활용된다. 정형 기법으로는 여러 행위자의 상호작용을 모델로 만들어 가능한 시나리오를 모두 탐색함으로써, 경쟁 상황에서 시스템이 의도한 안전한 결과로 수렴하는지 검증할 수 있다.

4. Gas 한계 및 DoS 문제

1) 발생 조건

Gas 한계 취약점은 이더리움의 블록당 또는 트랜잭션당 가스 제한으로 인해, 특정 함수가 과도한 연산이나 루프를 포함하면 아예 실행되지 못하거나 의도치 않게 실패하는 문제를 말한다. 가스는 스마트 컨트랙트 실행 비용을 측정하는 단위로, 복잡한 연산일수록 더 많은 가스를 소모한다. 블록 가스 한도를 넘는 연산을 시도하면 트랜잭션은 항상 실패하므로, 공격자는 이를 악용해 서비스 거부 상황을 유발할 수 있다.

2) 실제 사례

어떤 스마트 컨트랙트가 동적 배열의 모든 원소에 반복적으로 지급을 수행한다고 가정하자, 공격자는 해당 배열을 비정상적으로 크게 만들거나, 각 지급받는 계정에 복잡한 fallback 함수를 설계하여, 지급 루프가 블록 가스 한도를 넘도록 유도할 수 있다. 그 결과 해당 함수는 실행될 수 없게 되어, 잔여 자금이 영구 동결되거나 특정 참여자가 영원히 인출을 못 하는 상황이 생긴다. 실제로 초창기 몇몇 토큰 계약에서 다수 사용자에게 한번에 Ether를 보내는 함수를 구현했다가, 너무 많은 사용자로 인해 gas 초과로 실행이 실패하고 누구도 출금을 못 하는 DoS 취약점이 발견되었다.

3) 방지 전략

가스 관련 취약점을 피하려면 연산 복잡도를 제어해야 한다. 무한 또는 비상한 루프, 너무 큰 배열 조작 등을 피하고, 필요한 경우 여러 트랜잭션에 작업을 분할하는 설계를 채택한다. 예를 들어 앞서 언급한 대량 지급의 경우 한 번에 모두 처리하는 대신, 각 사용자가 withdraw 함수를 호출해 자기 돈만 가져가도록 하거나, 반복 작업을 일정 개수씩 나눠 수행하는 방식을 쓴다. 또한 블록 가스 한도가 향후 상향조정되거나 EVM의 gas 비용이 변할 수 있음을 염두에 두고, 여유 있

스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발			
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

는 한도로 함수 설계를 해야 한다. 정형 분석 도구 중에는 가스 소모 추적을 통해 out-of-gas 시나리오를 탐지하는 기능을 제공하는 것도 있다. 이런 도구를 활용하면 계약이 어떤 입력에서 gas 한도를 넘을지 자동으로 점검할 수 있다. 한편 악의적인 DoS를 막기 위해, 외부 호출시 capped gas를 지정해 상대 컨트랙트가 과도한 연산을 수행하지 못하도록 하는 기법도 활용된다.

다. 테스트 기반 검증의 한계와 정형 기법의 보완

전통적으로 소프트웨어의 품질 확보를 위한 테스트 기법들이 보편적으로 사용되고 있으며, 스마트 컨트랙트 개발에서도 단위 테스트, 통합 테스트, 보안 테스트 등이 수행된다. 하지만 테스트 기반 방법에는 한계가 있다. 첫째, 테스트는 샘플링 기법을 활용하여 확인한다. 즉, 가능성 있는 입력과 시나리오 중 극히 일부만을 실행해볼 뿐이며 모든 경로를 커버하지 못한다. 테스트 케이스를 수백 개 작성해도, 프로그램 상태공간이 거대하다면 여전히 미검증 영역이 생길 수 밖에 없다. 개발자는 테스트를 실행하여 확인 하지만, 이는 그 시나리오에서만 올바름을 보인 것일 뿐 전체적인 버그의 부재를 의미하지 않는다. 스마트 컨트랙트의 경우 재진입 같은 복잡한 상호작용이 있는 오류는 수많은 인터리빙 중 특정한 순서로 발생하므로 임의 테스트로는 놓치기 쉽다. 둘째, 테스트는 명세의 완전성에 의존한다. 개발자가 미처 인지하지 못한 취약 시나리오는 테스트에도 반영되지 않는다. 즉, 테스트는 존재하는 버그를 발견할 순 있지만 버그가 없음을 증명하지는 못한다. 이로 인해 치명적 취약점이 배포 후에야 발견되는 경우가 있다. 셋째, 테스트는 실행 환경 조건을 모두 재현하기 어렵다. 블록체인의 병렬성, 타이밍, 가스 제한 등의 특수 상황은 로컬 테스트로 완벽히 모방되지 않으며, 악의적인 공격자는 일반적인 환경과 다른 극단 상황을 유도하기 때문에 테스트로 대비가 힘들다.

이러한 한계는 정형 기법을 활용하여 보완 할 수 있다. 정형 기법은 시스템의 동작을 수학적으로 모델링하고 논리적으로 검증함으로써 모든 가능한 경우에 대해 올바름을 보장하려는 접근이다. 첫째, 정형 검증 도구는 상태공간을 자동 탐색하거나 논리식 증명을 시도하여, 사람이 생각해내기 어려운 경로까지 검사하여, 테스트가 미치지 못하는 영역까지 확인 할 수 있다. 둘째, 정형 명세를 수학적으로 작성하면 기존의 자연어로 표현해 모호하게 표현되는 점을 없앨수 있다. 테스트는 종종 자연어와 같은 표기로 요구사항 문서를 작성하지만, 정형 기법을 활용하면 요구사항 자체를 논리식으로 명시하고 코드가 이를 항상 만족하는지 확인하여 모호함을 없앤다. 이러한 접근은 사람 실수나 주관을 배제하고, “이 프로그램

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

이 명세대로 동작하는가?“라는 명확한 질문에 대한 엄밀한 예/아니오 답을 준다.셋째, 정형 기법은 주어진 검증 범위 내에서 모든 상태와 경로를 다 살펴보기 때문에, 검증을 성공하면 해당 범위에서는 모든 옛지 케이스까지 문제 없음을 보장한다.

정형 기법도 전제 조건에 따른 한계가 있고 수행에 전문지식과 노력이 필요하지만, 스마트 컨트랙트처럼 변경 불가능한 코드에는 사후 보완이 어려우므로 배포 전 명세 및 검증 과정이 필수적이다. 실제로 마이크로소프트 리서치 등에서는 Solidity 코드를 자동으로 수학 모델로 변환해 검증하는 도구를 개발하여 플랫폼에 적용하였고, CertiK와 같은 보안 업체는 주요 프로젝트의 컨트랙트를 정형 검증하여 배포 전 취약점을 다수 발견해내고 있다. CertiK는 SushiSwap의 신규 컨트랙트를 검증해 풀이 고갈될 수 있는 버그를 사전에 차단했고, 그 결과 DeFi 프로젝트들이 수십 억 달러 규모 자산을 보다 안전하게 운영할 수 있었다. 테스트는 여전히 기본적으로 필요하지만 그것만으로는 부족하며, 수학적 기반의 명세 및 검증 방법인 정형 기법을 통해 스마트 컨트랙트의 신뢰성을 한층 높여준다.

스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발			
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

IV. 스마트 컨트랙트를 위한 정형 기법 개념

스마트 컨트랙트의 올바른 동작을 보장하려면 우선 시스템의 요구사항과 동작을 형식적으로 표현할 수 있어야 한다. 정형 기법은 자연어 대신 수학적 언어를 이용해 시스템을 추상화하는 기법이다.

가. 정형 기법의 원리

정형 기법은 수리논리학을 토대로 소프트웨어나 시스템의 스펙을 염밀하게 기술하는 방법론을 말한다. 여기에는 집합론, 논리, 그래프, 형태언어 등 다양한 수학적 도구가 활용된다. 시스템의 가능한 상태들과 상태 간 전이를 모두 포함하는 형상을 정의하고, 올바른 상태나 전이의 조건을 논리적으로 서술하는 것이다. 이를 통해 구현 코드보다 추상적이지만 정확한 의미론을 갖는다.

수학적으로 정형 명세는 다양한 형태를 취할 수 있다. 집합과 관계로 시스템 속성을 서술하는 집합론 기반 명세는 상태를 변수들의 집합으로 표현하고 불변 조건을 수학식으로 적는다. 시계열 논리를 사용해 시간에 따른 조건을 기술하는 논리 기반 명세는 “항상(\Box) 혹은 결국(\Diamond) P가 성립한다”와 같은 성질로 요구사항을 나타낸다. 간단한 예로 동작을 함수로 추상화하여 전후 조건을 명시하는 Hoare 논리는 논리 기반 검증 방식의 일종으로, 스마트 컨트랙트의 함수에 사전 조건과 사후 조건을 붙여 검증하는 방식으로 활용 할 수 있다. 다른 한편, 명세 자체를 실행 가능한 모델로 만드는 방법도 있다. 이는 상태 기계나 프로세스 연산으로 시스템 동작을 기술하는 것으로, 이후 모델 체커로 검증하거나, 동작 시뮬레이션을 통해 이상 여부를 찾는다. 이러한 여러 길이 있지만, 모두 엄격한 수학 체계를 사용함으로써 명확하고 검사 가능한 시스템 기술을 얻는다는 공통 목표를 갖는다.

정형 기법은 요구사항 분석 단계에서 모호함을 제거하고, 구현 전에 논리적 문제를 발견하는 데 유용하다. 또한 모델이 있으면 정형 검증 도구가 이를 입력으로 받아 자동 검증을 수행할 수 있다. 예를 들어 TLA+로 작성된 모델은 TLC 모델 체커의 입력이 되고, Alloy로 기술된 모델은 SAT 솔버가 모든 경우를 탐색해준다.

나. 상태 기반, 행위 기반, 논리 기반 모델링의 유형과 비교

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

정형 기법에는 여러 접근 방법이 존재하며, 그 중 크게 상태 기반(state-based), 행위 기반(behavior-based), 논리 기반(logic-based) 모델링으로 구분할 수 있다. 각각 시스템을 바라보는 관점이 다르며, 표현의 용이성과 검증 기법에서 장단점이 있다.

1. 상태 기반 모델링

시스템을 상태와 상태 간 전이로 표현하는 방법이다. 이 접근에서는 시스템의 상태를 나타내는 변수들과 그 값들의 조합 즉, 상태공간, 그리고 연산 또는 이벤트에 의한 상태 변화 규칙이 중심이 된다. 대표적으로 유한 상태 기계, 상태 다이어그램, Z, VDM, Event-B 같은 명세 언어가 상태 기반이다. 스마트 컨트랙트를 상태 기반으로 보면, 컨트랙트의 저장 변수들이 상태를 형성하고 함수 호출이 한 상태에서 다른 상태로 가는 전이 역할을 한다.

상태 기반 모델링은 시스템의 전체 구조와 자료 불변성을 드러내는 데 효과적이다. 각 상태에서 만족해야 할 조건들을 명시하기에 불변조건 검증이나 상태 도달성 검증에 유리하다. 또한 시각화가 용이한 장점 또한 있다. 반면, 상태의 개수가 많아지면 상태공간 폭발 문제가 있다. 시스템이 복잡할수록 상태 조합이 기하급수적으로 늘어나 모델 체크 등이 어려워질 수 있다. 또한 동시성 등 행위적인 측면을 표현하는 데는 추가 장치가 필요하다. 하지만 이를 보완하기 위해 타임 아웃이나 병행 상태를 나타내는 FSM with concurrency, Petri-net 등이 사용되기도 한다.

2. 행위 기반 모델링

시스템을 시간에 따라 일어나는 행위이나 사건의 흐름으로 기술하는 방법이다. 이는 보통 프로세스 간 상호작용이나 이벤트 순서에 초점을 맞춘다. 예를 들어 CSP(Communicating Sequential Processes), Pi-Calculus, 상태차트, 시퀀스 다이어그램 등이 행위 기반 접근이다. 스마트 컨트랙트를 행위 모델로 보면, 여러 사용자나 외부 계약과의 메시지 교환, 함수 호출 시퀀스, 병렬 실행 등이 주요 모델 요소다.

행위 기반 모델링은 동시성과 상호작용을 자연스럽게 표현한다. 여러 에이전트가 주고받는 메시지나 병렬 이벤트를 모형화하기에 적합하며, 시간적 제약이나 순서 의존 로직을 쉽게 서술할 수 있다. 이런 이유로 프로토콜 검증이나 통신 절차 분석에 강점을 보인다. 반면, 행위 모델은 상태보다 시나리오 중심이기에, 전체 상태를 포괄하는 불변성 표현이 어렵거나 간접적일 수 있다. 또한 데이터 값 자체보다는 이벤트 흐름에 집중하기에, 특정 변수 값의 정확성보다는 이벤트 간 인과 관계 검증에 치우칠 수 있다. 스마트 컨트랙트에서는 금융 로직처럼 금액 계

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

산 정확성도 중요한데, 행위 모델로는 이런 수치적 불변식을 드러내기엔 부적합할 수 있다. 따라서 복잡한 금전 보존 같은 성질은 상태 기반 모델과 조합해 다루는 경우가 있다. 행위 기반 접근은 순차/동시 동작 중심 모델링으로, 컨트랙트 간 메시지 순서, 사용자 액션 시나리오, 경쟁 상태 분석 등에 특히 효과적이다.

3. 논리 기반 모델링

시스템의 속성을 논리식이나 수학 공식으로 직접 서술하는 접근이다. 상태나 이벤트를 일일이 모델로 만들기보다, 시스템이 만족해야 할 요구사항을 논리 명제로 나타내는 것이 특징이다. 이러한 논리 기반 접근에는 명제/술어 논리, 임시 논리, 객체 제약 논리 등이 쓰인다. Alloy 같은 도구는 1차 논리와 집합 제약으로 모델을 기술하며, SAT 솔버로 충족 여부를 검사하는데, 이는 모델을 사실상 논리식의 집합으로 표현한 것이다.

논리 기반 모델링은 높은 수준의 추상화를 제공한다. 원하는 성질을 직접 기술하므로, 구현과 무관하게 달성해야 할 요건을 분명히 할 수 있다. 복잡한 시스템이라도 핵심 논리 제약을 간단하게 요약하여 표현 할 수 있다. 또한 수학적으로 정형화되므로 자동화 도구도 적용하기 쉽다. 반면, 논리만으로 시스템을 서술하면 구조적 맥락이 사라질 수 있다. 즉 구현이나 설계의 형태를 무시하고 결과 조건만 쓰기 때문에, 그 조건을 만족하는 구현이 여러 개일 수 있고 모델 자체로는 실행 순서를 알기 어렵다. 그래서 논리 명세를 만족한다고 해도 실제 코드가 어떻게 동작해야 하는지 직관이 부족할 수 있다. 또한 상태 변화 과정 검증에는 적합하지 않을 수 있다. 따라서 논리 기반은 보통 다른 모델과 조합되어 사용된다.

위 세 가지 모델링 패러다임은 상호 배타적이지 않고 보완적으로 사용된다. 예를 들어, 스마트 컨트랙트의 주요 상태와 전이는 상태 기반으로 모델링하고, 동시 사용자 행위는 프로세스 대수로 기술하며, 마지막으로 시스템이 만족해야 할 보안 속성은 논리 명제로 명시하는 식으로 혼합 가능하다. 중요한 것은 모델링 시 어떤 측면을 강조할지에 따라 도구와 방법을 선택하는 것이다. 데이터 무결성 검증이 목표라면 상태 기반, 프로토콜 합의 검증이면 행위 기반, 특정 보안 속성 증명이라면 논리 기반 접근이 적합할 수 있다.

4. 스마트 컨트랙트 모델의 주요 요소

스마트 컨트랙트를 정형 기법하려면, 계약의 특성을 잘 나타낼 수 있는 모델 요소들을 파악해야 한다. 일반적으로 주목되는 개념은 불변성, 트랜잭션 흐름, 상

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

태 전이이다.

1. 불변성

불변성은 시스템 실행 전반에 걸쳐 항상 참으로 유지되어야 하는 조건을 뜻한다. 스마트 컨트랙트 맥락에서 불변성은 흔히 자산 보존, 권한 관계, 논리 일관성 조건 등을 포함한다. 예를 들어 ERC-20 토큰 컨트랙트의 불변성으로 “총 토큰 발행량은 각 계정 잔고의 합과 같다”가 있다. 이를 수학적으로 표현하면, 모든 reachable state에 대해 $\text{totalSupply} = \text{sum(balances)}$ 라는 등식이 성립한다는 것이다. 또 다른 예로, 경매 컨트랙트에서는 “마감 후에는 더 이상 입찰을 받지 않는다”가 불변 조건이 될 수 있다. 불변성은 상태 기반 모델에서 safety 속성으로 분류된다. 모델 검증에서는 불변성이 깨지는지를 탐색함으로써, 논리적 오류를 발견 한다. 불변성을 수학적으로 다루려면 먼저 상태공간을 정의하고 불변 조건을 술어로 기술한다. 반례 상태를 찾으면 불변성 위반이 확인된다. 스마트 컨트랙트 개발에서는 핵심 불변성을 식별하고 이를 명세로 작성 함으로써, 코드 구현이 이를 보장하는지 검증할 수 있다. 불변성은 재진입 공격이나 오버플로우로 인한 이상값 등이 결국 불변성 위반으로 드러나기 때문에, 보안 약점 탐지의 기준으로도 활용 할 수 있다.

2. 트랜잭션 흐름

스마트 컨트랙트는 외부에서 트랜잭션을 통해 호출되며, 이로 인해 상태가 변화하고 다른 계약 호출이 일어나는 흐름이 생긴다. 이러한 트랜잭션 시퀀스나 상호작용의 과정을 모델링한 것이 트랜잭션 흐름이다. 한 개별 트랜잭션은 EVM에서 원자적으로 처리되지만, 다수의 트랜잭션에 걸친 시나리오를 통해 전체 시스템 동작이 시행된다. 정형 모델에서는 시스템 실행 개념으로 이를 다룰 수 있다. 즉, 상태 s_0 에서 시작하여 트랜잭션 t_1 적용 후 s_1 , 다시 t_2 적용 후 s_2 , ... 식으로 일련의 상태열 $s_0 \rightarrow s_1 \rightarrow \dots \rightarrow s_n$ 이 나온다. 이 시퀀스를 모델로 하여 올바른 흐름인지 검증할 수 있다. 예를 들어, “언제나 일정 시간 내에 특정 이벤트가 발생한다”는 liveness 속성은 개별 트랜잭션이 아닌 실행 경로 단위로 정의된다. 스마트 컨트랙트에서는 활성 속성으로 “모든 요청은 결국 처리된다” 등이 있을 수 있다. 트랜잭션 흐름을 모델링하는 방법으로는 유한 실행 경로를 나열하는 시퀀스 다이어그램, 또는 가능한 실행들의 집합을 정의하는 프로세스 알제브라 모델 등이 있다. 수학적으로는 실행을 상태와 전이 함수 δ 의 반복 적용으로 표현하며, 속성을 Path formula로 표현할 수 있다. TLA+에서는 “Eventually P”를 $\Diamond P$ 로, CTL 같은

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

논리에서는 $\forall \square P$ 같은 식으로 나타낸다. 트랜잭션 흐름 모델링의 목적은 상호작용 과정에서 문제가 생기지 않는지 확인하는 것이다. 이처럼 흐름을 모델링하면, 단일 상태가 아닌 상태 전환의 연쇄에서 나타나는 요구사항을 다룰 수 있다. 다만, 모든 가능 흐름이 아주 많을 수 있으므로, 모델 검증 시에는 일정 횟수 이내의 트랜잭션 시퀀스만 고려하거나 추상화를 사용하여 주요 흐름만 검토하기도 한다.

3. 상태 전이 모델

이는 상태 기반 모델링의 핵심으로, 시스템을 하나의 상태 머신으로 보는 관점이다. 스마트 컨트랙트에서 함수 호출은 조건에 따라 다른 동작을 수행하며, 때로는 컨트랙트가 여러 운영 모드를 갖기도 한다. 예를 들어 “Initialized”, “Active”, “Paused”, “Terminated” 같은 상태를 코드로 관리하는 경우가 있다. 이러한 컨트랙트는 전이 모델로 나타내기 좋다. 상태 전이 모델은 보통 (S, Act, δ) 로 정의된다. 여기서 S 는 상태들의 집합, Act 는 액션이나 행위들의 집합, $\delta: S \times Act \rightarrow S$ 는 전이 함수이다. 어떤 행동 Act 가 현재 상태에서 발생하면 δ 에 따라 다음 상태가 결정된다. 이 개념은 수학적으로 전이 시스템 또는 오토마トン으로 정식화된다. 상태 전이 모델의 장점은 시각적 검토와 자동 검증 모두에 적합하다는 점이다. 예를 들어 스마트 컨트랙트의 핵심 로직을 유한 상태 머신 디어그램으로 그리면, 개발자나 감사자가 쉽게 논리 흐름을 이해하고 결함을 토의할 수 있다. 또 이 디어그램을 Promela나 SMV 같은 언어로 옮기면 모델 검증 툴로 reachability, deadlock 등을 점검할 수 있다. 실제 연구에서 Solidity를 자동으로 FSM으로 추출해 검증하는 시도도 있었으며, Bamboo, Obsidian 같은 신규 스마트 컨트랙트 언어는 아예 상태 전이를 언어 수준에서 명시하게 만들어 안전성을 높였다. 상태 전이 모델은 불변성과 함께 쓰이기도 한다. 각 상태별로 만족해야 할 불변조건을 정의하고, 전이 함수가 불변조건을 항상 보존함을 증명하면 코드의 핵심 안전성이 확보된다. 더 나아가 전이 모델에 시간 제약이나 확률 등을 결합해, 시간적, 확률적 속성까지 분석하는 확장도 가능하다. 스마트 컨트랙트의 경우 일반적으로 비즈니스 로직이라 시간/확률 요소는 드물지만, 복권과 같은 시스템은 확률 모델링이 필요할 수 있다. 이런 것도 전이 시스템의 특별한 경우로 다룰 수 있다. 한편, 상태 전이 그래프는 코드 구현을 추론하는 데도 쓰인다. 정형 기법 도구 중 KEVM이나 BIP 등의 프레임워크는 실제 EVM 바이트코드를 state graph 형태로 나타내고, 모델 체킹이나 프로그램 변환에 활용한다.

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

V. 스마트 컨트랙트 정형 명세 방법

스마트 컨트랙트를 모델링하고 검증하기 위해 다양한 정형 기법과 도구들이 제안되고 있다. 본 장에서는 대표적인 정형 기법 및 명세 기법들 중 TLA+, Alloy, Coq/Isabelle/Lean과 같은 정형 명세 및 증명 도구, VeriSol과 같은 스마트 컨트랙트 전용 검증기, 그리고 심볼릭 실행/SMT 솔버 기반에 대한 설명 및 특징을 기술 한다. 각 기법의 수학적 기반, 검증 절차, 장점, 단점, 시나리오 등에 대해 설명 한다.

가. TLA+

1. 수학적 기반

TLA+는 Leslie Lamport가 개발한 정형 명세 언어로, 행위 논리에 기반하여 상태 변화 시스템을 기술한다. 특히 시간의 흐름에 따른 행위를 논리식(\square , \diamond 연산 등)으로 표현할 수 있으며, 집합과 함수 개념을 활용하여 상태를 서술한다. TLA+의 핵심 아이디어는 시스템을 초기 상태 Init과 상태 전이 관계 Next로 정의하고, 원하는 성질을 불변성이나 논리 속성으로 명세하는 것이다.

2. 검증 절차

TLA+는 모델 검증과 증명 두 가지를 지원하는데, 일반적으로 TLC 모델 체커를 통한 검증이 많이 쓰인다. 개발자는 TLA+로 시스템 명세를 작성하고, 검증 설정에서 검증 대상이 되는 Spec, Constant, 검증할 불변식 등을 지정한다. 그런 다음 TLC를 실행하면 상태 공간을 탐색하여 불변성 위반이나 사전 정의한 Safety, Liveness 속성 위반을 찾는다. 만약 문제가 발견되면 반례 실행을 출력하는데, 이는 어떤 순서의 사건으로 문제가 일어났는지 보여준다. 반대로 설정한 상태 공간 범위 내에서 버그가 없으면 검증 통과를 보고한다.

3. 장점

TLA+는 대규모 분산 시스템에 특화되어 개발되었기에, 스마트 컨트랙트의 경쟁 상호작용 검증에 강하다. 실제로 TLA+는 DAO 같은 재진입 버그를 모형화하여 찾아내는 데 효과적이라는 사례가 보고되었다. 한번 모델을 만들면 틀이 알아서 모든 케이스를 탐색하므로, 테스트로 잡기 어려운 버그까지 노출시킨다. 또한 명세 자체가 구현보다 추상화되어 있어, 개발 초기 설계 검증에 유용하다. TLA+는 수학 언어이지만 비교적 배우기 쉬운 편이며, 명세가 코드와 분리되어 있어 설계 문서로 활용할 수도 있다. Amazon 등 기업에서 TLA+로 설계 결함을 찾아낸 유명

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

사례들이 있다.

4. 단점

실제 코드로부터 모델을 추출하거나, 반대로 명세에서 코드를 생성하는 기능은 제한적이어서, 모델-구현을 잘 관리해야 한다. 즉, 명세가 제대로 구현에 반영되었는지 추가 검증이 필요하다. 또한 상태공간이 크면 모델 체커가 완전 탐색을 못 할 수 있어, 적절한 추상화와 변수 도메인 크기 제한과 같은 과정이 필요하다. 스마트 컨트랙트의 큰 값들을 계산할 때 비용이 많이드는 경우, TLA+ 모델에서는 종종 이를 작게 줄여서 검증하게 된다.

5. 적합한 시나리오

TLA+는 동시성과 상호호출이 복잡한 계약에서 잘 활용 할 수 있다. 예를 들어, DeFi 프로토콜처럼 여러 컨트랙트가 연계되어 동작하고, 다양한 시나리오를 모두 검토해야 할 때 적합하다. 또한 컨트랙트 레벨에서 전역 불변성을 검증하고 싶을 때 유용하다. TLA+는 시스템 아키텍처 검증에 가깝기 때문에, 구현 전에 설계를 검증하거나, 배포 전 통합테스트 격으로 모델화 하여 검증 하기에 유용하다. 실제 사례로, MakerDAO 팀이 TLA+로 새로운 시스템의 설계를 검증해 특정 시나리오에서 담보 자산이 제대로 경매 처리되지 않을 수 있음을 발견하고 수정했다는 보고가 있다.

나. Alloy

1. 수학적 기반

Alloy는 MIT Daniel Jackson이 개발한 경량 정형 명세로, 1차 논리와 관계 모델에 기반한다. Alloy에서는 시스템을 객체와 관계의 구조로 기술하고, 원하는 제약조건을 논리식으로 표한다. Alloy의 배경 이론은 관계 해석이며, 명세는 SAT 문제로 변환되어 분석된다. 즉, Alloy Analyzer는 주어진 논리 모델에 대한 반례 인스턴스를 SAT Solver를 통해 찾는다. 기본적으로 Alloy는 논리 기반 모델링의 대표적인 기법이다.

2. 검증 절차

Alloy 언어로 명세를 작성한다. 명세에는 sig으로 객체를 정의하고, fact로 항상 만족해야 할 제약, pred로 시나리오, assert로 검증할 명제 등을 포함한다. 그런 다음 Alloy Analyzer에서 해당 명세를 불러와 자동 분석을 실행한다. 두 가지 모드가 있는데, 하나는 인스턴스 생성으로, 제약을 만족하는 구체적인 사례를 찾아

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

시각적으로 보여준다. 다른 하나는 assert 검증으로, assert 명제가 불린값 False가 되는 인스턴스 즉, 명제가 성립하지 않는 사례를 SAT solver로 찾는다. 사용자는 검증 범위를 설정해야 하며, 이 범위 내에서 완전 탐색이 이뤄진다. 결과로 만약 assert에 대한 반례가 있으면, Alloy는 그 반례를 구체적인 구조로 제시한다. 반례가 없으면 “No counterexample found”로 표기되며, 해당 범위 안에서는 명제가 참임이 검증된 것이다.

3. 장점

가장 큰 장점은 사용 편의성과 자동화다. 개발자나 설계자가 비교적 쉽게 배울 수 있는 문법으로 추상 모델을 작성하면, 버튼 클릭만으로 SAT 기반 검증이 수행된다. 범위 내 완전탐색이라 작은 크기의 문제를 확실히 커버하며, 반례가 있다면 짧은 시간 내에 직관적인 형태로 보여주기 때문에 설계 단계 버그 잡기에 유용하다. 또한 Alloy 명세는 코드 수준 세부사항을 생략하고 시스템 구조와 제약에 집중하므로, 스마트 컨트랙트의 시스템 수준 속성을 실험적으로 검증하는 데 좋다.

4. 단점

Alloy는 분석 범위가 한정되어 있다. 사용자가 정한 객체 수, 숫자 범위 이내에서만 검증하므로, 엄밀히 말해 무한 상태 시스템의 완전한 증명은 아니다. 예를 들어 계정 수 3, 토큰 발행 100 이하로 설정해 불변식을 검증했다 해도, 실제 무한 범위에서 성립한다고 확신할 수는 없다. 또한 Alloy는 동적 행위 표현이 제한적이다. 주로 정적 구조 제약에 강하며, 시간에 따른 상태 변화보다는 관계적 불변에 초점이 있다. 스마트 컨트랙트의 트랜잭션 시퀀스를 Alloy로 표현하려면 시퀀스 번호를 도입하거나 해야 하는데, 이는 언어 설계상 부자연스러울 수 있다. 따라서 주로 실행 전후의 상태 관계나 “어떤 나쁜 상태가 존재 가능한가?”를 분석하는 데 쓰이고, 실제 코드의 실행흐름 검증은 어려울 수 있다.

5. 적합한 시나리오

스마트 컨트랙트 설계 초기에 개념 모델 검증에 적합하다. 새로운 DeFi 시스템을 설계하며 여러 역할과 자산 간 관계를 Alloy로 모델링해봤더니, 어떤 조건에서 자산이 보존되지 않는 반례가 발견되어 설계를 조정하는 식이다. 또는, 접근제어 규칙이 복잡한 경우 Alloy로 객체와 권한 관계를 모델링해 모순이나 약점을 찾아낼 수 있다. CertiK 등에서는 실제로 ERC20 표준의 예상 동작들을 Alloy와 유사한 논리 툴로 확인하기도 했다. Alloy는 코드 감사에 앞서 논리적 일관성을 점검하는 용도로, 비교적 빠르게 적용 가능하다. 다만, 안전한 범위 설정과 해석이 필

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

요하며, 모델과 실제 코드 구현의 차이를 늘 염두에 두어야 한다.

다. Coq / Isabelle / Lean (인터랙티브 증명)

1. 수학적 기반

Coq, Isabelle/HOL, Lean 등은 인터랙티브 증명기들로, 머신 체크 가능한 증명을 작성하는 데 사용된다. Coq은 type theory 기반의 언어 Gallina를 제공하고, Isabelle은 HOL 및 ZF 집합론 등 다양한 논리 선택을 지원하며, Lean은 Coq과 유사한 논리 핵심을 갖는다. 이들 도구는 프로그래밍 언어와 증명 작성 환경을 결합한 것으로 볼 수 있다. 개발자는 시스템을 함축적으로 모델링하고, 증명하고자 하는 theorem를 명시한 뒤, 대화형으로 증명 단계를 구축한다. 증명은 Lambda calculus 기반의 엄격한 논리 체계에서 진행되어, 최종적으로 Proof term를 얻게 된다.

2. 검증 절차

예를 들어 Coq의 경우, 사용자는 Coq 언어로 스마트 컨트랙트의 논리적 모델을 작성한다. 이를테면, 정수 모듈로 잔고 개념을 표현하고, 함수들을 재귀적으로 정의하거나 관계로 서술한다. 그리고 증명하고자 하는 명제들을 Lemma/Theorem으로 선언한다. 이후 proof 모드로 들어가 보조 정리 및 tactic을 사용해 증명을 진행한다. Coq의 전술은 auto, omega, lia 등 자동화 도구도 있지만, 복잡한 부분은 사용자가 논리 명령어로 직접 증명해야 하며, 필요하면 보조 정리를 추가로 증명해 사용한다. 이런 식으로 상호작용하며 모든 서브골을 해결하면 Coq이 Qed를 확인해준다. Isabelle이나 Lean도 유사한 상호작용 증명 방식을 가진다. 증명이 완료되면, 이는 해당 명제가 이론적으로 참임을 보장한다. 원하는 경우 Coq 모델로부터 OCaml 코드 추출도 가능한데, Isabelle은 Sledgehammer라는 자동화 툴로 외부 SMT solver의 도움을 받기도 한다.

3. 장점

인터랙티브 증명은 가장 높은 보증 수준을 제공한다. 한번 증명된 성질은 논리적으로 철회될 수 없으므로, 스마트 컨트랙트에 대한 완벽한 수학적 신뢰를 구축할 수 있다. 예컨대 이더리움의 EVM 자체를 Coq으로 모델링하여, 특정 컨트랙트 바이트코드가 어떤 안전 속성을 만족함을 증명하면, 이는 네트워크 어떤 상황에서도 성립함을 의미한다. 또한 매우 복잡한 성질도 사람이 논증할 수 있다면 증명 가능하다. Coq/Isabelle은 강력한 라이브러리와 모듈화를 지원해, 하나의 거대한 증명을 여러 부분으로 나눠서 증명 하는 것도 가능하다.

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

4. 단점

이러한 정리 증명은 매우 많은 시간과 노력을 요한다. 일반적으로 스마트 컨트랙트처럼 비교적 짧은 코드라도, 비트단위 정확성까지 증명하려면 구현 코드에 준하거나 그 이상의 분량의 증명 스크립트가 필요하다. 전문 지식도 요구되어, 개발자 혼자 하기 어렵다. 또한 증명 대상 자체의 오류를 배제할 수 없다. 잘못된 명세나 잘못 모델화된 정의를 증명하면 의미가 없는데, 이건 사람이 주의깊게 검토해야 한다. 증명을 완료한 뒤에도 실제 배포 코드와 증명된 모델이 일치하는지 확인이 필요하다. 이 부분은 도구 지원이 아직 제한적이라, 증명 기반 접근은 주로 스마트 컨트랙트 언어 설계나 가상머신 레벨에서 많이 시도되고, 개별 DApp에는 현실적으로 적용이 적은 편이다.

5. 적합한 시나리오

중앙은행 디지털화폐(CBDC)나 국가전략 자산이 걸린 스마트 컨트랙트, 또는 수학적으로 특정 알고리즘 구현을 증명해야 하는 경우 도입할 만하다. 실제 사례로 Tezos, Cardano 등의 블록체인에서 코어 프로토콜을 Coq/Isabelle로 검증하는 노력이 있었고, 이더리움 2.0의 일부 합의 알고리즘을 Coq 증명한 연구도 있다. 스마트 컨트랙트 수준에서는, 소규모로 Isabelle/HOL을 이용해 이더리움 컨트랙트를 모델링하고 특성 증명한 학술 보고도 있으나, 산업에선 찾아보기 드물다. 그러나 특정 금융기관이 자신들의 스마트 컨트랙트에 대해 최고 수준 검증을 원한다면, Coq 전문가를 고용해 주요 속성을 증명하는 방법도 가능할 것이다. Lean은 최근 GitHub Copilot 등과 연계해 증명 자동화를 시도하고 있어, 향후에는 더 접근성이 나아질 수도 있다. 요약하면, 정리 증명은 고비용-고신뢰 전략으로, 매우 중요한 계약이나 새로 만드는 스마트 계약 언어/VM에 적합하며, 일반적인 개발팀에는 부담이 크지만 그 가치 또한 최고로 인정받는 방식이다.

라. VeriSol

1. 수학적 기반

VeriSol은 마이크로소프트 리서치에서 개발한 Solidity 전용 정형 검증 도구로, Solidity 코드를 중간 검증 언어 Boogie로 변환한 뒤 SMT 기반 검증을 수행한다. VeriSol의 근간은 Hoare 논리적 검증과 SMT 솔버를 활용한 자동 추론이다. Solidity 프로그램을 Boogie라는 정형 명세 언어로 바꾸는데, Boogie는 명령형 프로그램의 전역 불변식, 루프 불변식, 사전/사후 조건 등을 표현할 수 있는 일종의 중간언어다. 이 Boogie 코드에 대해 검증 조건을 생성하고, SMT Solver로 풀어 충

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

족/위반 여부를 결정한다.

2. 검증 절차

개발자는 Solidity 스마트 컨트랙트에 대해 VeriSol을 실행한다. VeriSol은 소스 코드를 파싱하여 등가의 Boogie 프로그램으로 변환한다. 이 때 계약의 함수, 상태, require/assert 등이 Boogie의 절차적 명세로 매핑된다. 또한 사용자가 특정 속성을 검증하고 싶다면, Solidity 코드에 특수 주석 형태로 표시할 수 있다. 변환된 Boogie 프로그램은 Boogie 자체의 검증 엔진 또는 Z3 SMT solver를 통해 자동 검증된다. 결과적으로, 코드 내 삽입된 assert문이 항상 참인지, 또는 어떤 경로에서 거짓이 되는지를 판단한다. 만약 검증 실패시, Boogie가 반례 경로를 보여주는데, 이를 다시 Solidity 문맥으로 변환해 어느 함수의 어떤 조건에서 실패했는지 알려 준다. VeriSol은 현재 ERC 규격 검증 등 제한된 도메인에 특화되어 있었고, Azure Blockchain 환경의 특정 워크플로에 적용되었다는 보고가 있다.

3. 장점

개발자 친화적이다. 익숙한 Solidity 코드 위에서 동작하기 때문에 추가로 수학 모델을 배울 필요가 적다. 그냥 코드에 assert를 써넣고 돌리면, 자동으로 SMT 검증이 이뤄지는 형태다. 이처럼 자동화 수준이 높아 잠재적으로 CI 파이프라인에 넣어 수시로 실행할 수도 있다. 또 Boogie 언어가 강력하여, 정수 오버플로우, 배열 범위, 메모리 누수 같은 저수준 버그도 이론적으로 모두 모델링되어 체크 가능하다. MSR 발표에 따르면 VeriSol은 Ethereum 공용체인보다는 Azure같은 permissioned blockchain의 컨트랙트에 우선 적용되었는데, 그 맥락은 안전하게 스마트 컨트랙트를 배포하려는 산업 수요다.

4. 단점

모든 Solidity 코드를 다루지는 못하고, 특정 패턴 또는 크기에 한정된다. 복잡한 라이브러리나 어셈블리 코드, 난해한 메모리 연산 등을 지원이 미흡할 수 있다. 또한 스케일 문제로, 솔버가 복잡도에 따라 시간초과할 가능성이 있다. 자동화 도구다 보니, 결과를 신뢰하기 전에 도구의 soundness도 고려해야 한다. 그리고, interactive prover만큼 강력한 보장을 주는 것은 아니며, 찾지 못하면 통과하기에 버그를 놓칠 수 있다

5. 적합한 시나리오

스마트 컨트랙트 개발자들이 손쉽게 정형기법을 접할 수 있게 해준다. 특히 금융 관련 계약에서 금액의 보존, 제한 등의 조건을 검증하거나, 접근제어가 제대

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

로 적용됐는지 확인하는 등에 유용하다. Microsoft는 VeriSol을 자사 Azure 플랫폼의 스마트 계약 검증에 활용하여, 워크플로 엔진의 정책 위반을 잡아낸 사례를 소개했다. 다만, VeriSol은 현재 활발히 업데이트되고 있지 않고, Certora Prover 같은 상용 도구들이 점차 개선을 시도하면서 제공하고 있다. VeriSol은 Solidity를 직접 검증한다는 선구적인 접근으로 의의가 있으며, 향후 정형 검증 도구들이 나아갈 방향을 제시했다. 현업에서는 프로젝트에 맞게 이러한 SMT 기반 자동 검증기를 도입하거나, 오픈소스인 VeriSol을 확장하여 쓰는 전략이 가능하다.

마. 심볼릭 실행 / SMT 기반 모델링

1. 수학적 기반

심볼릭 실행(symbolic execution)은 프로그램의 입력을 구체값이 아닌 기호로 두고, 프로그램을 추상 해석하여 경로별 제약식을 만들어내는 기법이다. 이 제약식들은 보통 SAT/SMT 솔버로 풀어서, 충족 가능하면 구체적 반례 입력을 얻는다. 이는 논리 기반의 모델링과, 프로그램 경로 탐색의 조합이라 볼 수 있다. 스마트 컨트랙트 검증에서 심볼릭 실행은 EVM 바이트코드나 Solidity 소스에 적용되어, 잘 알려진 취약 패턴을 찾아내거나 사용자가 정의한 assert 위반을 찾는다. SMT 기반 모델링은 심볼릭 실행의 연장선으로, 아예 시스템을 수학 문제로 부호화하고 SAT/SMT solver에게 푸는 접근을 가리킨다.

2. 검증 절차

대표적인 예로 Mythril이라는 이더리움 보안 분석 도구를 보면, 컨트랙트 바이트코드를 받아 LASER 심볼릭 VM으로 실행한다. LASER VM은 EVM 명령어를 하나씩 따라가면서, 스택과 메모리를 심볼릭 변수로 취급하고 연산 결과를 SMT 식으로 누적한다. 분기 명령어를 만나면, 분기 조건을 분해하여 두 경로를 각각 탐색하며 path condition을 생성한다. 이렇게 해서 가능한 실행 경로를 폭넓게 탐색하고, 각 경로 끝에 도달할 때 미리 정의된 취약 패턴과 매칭되면 경고를 내거나, 또는 코드 내 assert 실패 조건을 검출한다. SMT 기반 접근은 이 과정에서 핵심이며, 실제로 Mythril은 Z3 솔버를 이용해 path condition + 취약조건이 SAT인지 체크한다. 만약 SAT, 즉 충족 가능한 경우 솔버가 변수값을 모델로 주는데, 이는 곧 공격 또는 버그를 실제로 실행하는 입력값이 된다. 이러한 검증은 대부분 자동으로 이뤄지고, 사람 개입은 필요 없으며, 도구가 정해진 깊이 또는 가스 한도까지 실행을 시도한다.

3. 장점

스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발			
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

심볼릭 실행/SMT 접근은 현재 스마트 컨트랙트 보안 업계에서 가장 실용적인 정형 기법으로 평가된다. 이유로는 완전 자동화가 가능해 개발 파이프라인에 쉽게 통합되고, 실제 코드 수준에서 동작하기 때문에 결과의 현실성이 높으며, 미리 정의된 많은 취약점 패턴들에 대해 높은 커버리지를 보이기 때문이다. Mythril, Oyente 등 도구들이 DAO 해킹 이후 속속 등장하여 재진입, 오버플로우 등을 자동 탐지했고, 현재도 DeFi 프로젝트들에서 CI 도구로 쓰이고 있다. 또한 심볼릭 실행은 경로 탐색을 체계적으로 하기 때문에, 무작위 fuzzing보다 적은 케이스로도 깊은 버그를 찾을 수 있다.

4. 단점

상태 폭발 문제가 있다. 분기와 loop가 많으면 경로 수가 기하급수적으로 늘어나, 심볼릭 엔진이 모든 경로를 살필 수 없게 된다. 이를 완화하려고 경로 제한, 근사 추론, 특정 패턴 우선 분석 등을 하지만, 여전히 scaling 이슈는 존재한다. 그래서 Mythril 같은 도구도 종종 몇가지 경로를 탐색한 후 중단 메시지를 낸다. 이는 곧 미처리된 경로에 버그가 숨어 있을 가능성을 의미한다. 또한 SMT 솔버의 한계로, 매우 복잡한 산술이나 논리식은 풀지 못하거나 시간 초과될 수 있다. 특히 이더리움의 비트조작 연산, 암호학적인 부분은 심볼릭으로는 사실상 해내기 어렵다. 이런 부분은 추측이나 단순화하여 넘어가기에, false positive나 false negative가 생길 수 있다. 그리고 자동 도구는 사람이 정의한 패턴 이상은 못 찾으므로, 새로운 유형의 논리 버그는 탐지 불가하다.

5. 적합한 시나리오

심볼릭 실행/SMT 기반 분석은 일반적인 스마트 컨트랙트 보안 검증에 사용될 수 있다. 개발 단계에서 사전에 코드 취약점을 잡는 용도로, 혹은 보안 감사자가 수동 코드리뷰와 병행해 도구를 돌려보는 식으로 쓸 수 있다. Mythril, Slither이 대표적이고, asset 크기나 민감도가 큰 DeFi나 NFT 프로젝트일수록 이런 도구 사용이 권장된다. 특히, 표준 취약점 검사를 자동화하기 때문에, 개발자는 사람의 실수로 놓칠 수 있는 부분을 기계가 커버해준다. 그러나, 앞서 언급했듯 높은 보증이 필요한 경우에는 심볼릭 실행만으로 불충분할 수 있으므로, 이때는 모델 검증이나 정리증명을 병행해야 한다. 현실적으로, 대부분 프로젝트는 출시 전 Mythril 같은 툴을 돌려보고, 큰 문제 없으면 넘어가는 식이다. 중요한 건 도구의 결과를 사람이 해석하고, 필요하면 수동 추가분석으로 false positive를 거르거나, false negative 가능성을 추정하는 작업이다.

스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발			
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

VI. 스마트 컨트랙트 정형 검증 기법

정형 명세가 시스템을 수학적으로 표현하는 것이라면, 정형 검증은 그렇게 만들어진 모델이나 실제 코드를 대상으로 논리적 검토를 수행하여 올바름을 보장하는 과정이다. 정형 검증에는 여러 가지 접근법이 있으며, 스마트 컨트랙트에 적용되는 주요 기법으로 모델 검증(Model Checking), 정리 증명(Theorem Proving), 추상 해석(Abstract Interpretation), 심볼릭 실행(Symbolic Execution), 퍼징(Fuzzing) 등을 들 수 있다. 이번 장에서는 각 검증 기법의 원리와 메커니즘, 적용 범위, 자동화 정도, 한계 등을 분석하고, 서로 비교한다.

가. 모델 검증

1. 메커니즘

모델 검증은 시스템의 상태공간을 자동 탐색하여, 주어진 속성을 만족하는지 확인하는 기법이다. 보통 대상 시스템을 유한 상태 모델로 추상화하고, 탐색 알고리즘을 통해 모든 도달 가능한 상태를 살펴본다. 탐색 도중 검증하고자 한 속성이 위배되는 상태를 발견하면, 그 경로를 반례로 제시한다. 혹은 모든 상태에서 속성이 유지되면 검증 성공을 선언한다. 모델 검증은 완전 자동화되어서, 사용자가 명세를 준비하면 나머지는 도구가 처리한다.

2. 적용 범위

모델 검증은 유한하거나 적당히 추상화된 시스템에서 효과적이다. 스마트 컨트랙트의 경우, 컨트랙트 자체는 상태공간이 거대할 수 있어 바로 적용하기는 힘들지만, 핵심 로직을 추상화한 모델에는 적용 가능하다. 예를 들어 3명 이하 사용자, 100이하 토큰 같은 제한을 둔 모델에 대해 총합 불변성을 검증하는 식이다. 또한 동시성, 순서 검증에 강하므로, 재진입이나 교착상태, 특정 시퀀스 의존 버그 등을 찾는 데 쓰인다. Ethereum에서 병렬 트랜잭션은 없지만, 다중 컨트랙트 간 호출 순서 이슈 등은 모델체커로 탐지 가능하다. 다만 모델 검증 결과가 실제 무한상태에도 일반화되려면, 별도 증명이나 범위 확장 시험이 필요하다.

3. 자동화

한 번 모델을 만들면, 검증은 버튼 클릭이나 커맨드 실행으로 이뤄진다. 완전 탐색이 핵심이므로, 사람이 경로별로 뭘 할 필요가 없다. 앞서 언급한 TLA+ TLC, SPIN 등이 모두 이런 자동 모델 체크 도구다. TLA+는 spec과 불변식만 주면 TLC가 에러트레이스를 내놓는다. SPIN도 Promela 모델과 LTL 속성을 주면, 만족/불만

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

족 여부를 알려준다. 단, 사용자는 모델 추상화에 신경써야 해서, 이 과정은 수동이다. 예를 들어 무한 배열을 유한 작게 자르거나, 연속 시간 대신 이산 step으로 바꾸는 등의 작업이 필요할 수 있다.

4. 한계

모델 검증의 최대 단점은 상태 폭발이다. 상태 수가 기하급수적으로 늘어나면, 탐색이 실질적으로 불가능해진다. 시간 복잡도도 커서, 조금만 모델이 복잡해져도 검증이 오래 걸릴 수 있다. 또한 모델링 오차 위험이 있다. 구현 코드와 모델이 달라서, 모델에서는 맞았지만 실제 코드는 문제있는 경우가 생길 수 있다. 이건 정형 기법 전반의 이슈지만, 모델 체크도 자유도 높은 추상화 때문에 종종 겪는다. 그리고, 속성이 증명되었다 해도, 제한된 범위 내라는 점이 완전성을 해칠 수 있다. 무한 상태 논리를 직접 다루는 CTL나 SMT-BMC(Bounded Model Checking) 같은 기법으로 일반화하려는 노력도 있으나, 현실적으로 무한은 증명 쪽에 맡기고 모델체커는 한정검증을 한다.

나. 정리 증명

1. 메커니즘

정리 증명은 수학적 증명을 통해 시스템 속성이 항상 성립함을 입증하는 기법이다. 인터랙티브 정리 증명기가 대표적으로 사용되며, 사람과 도구가 협력하여 불변식이나 전역 속성을 논리적으로 유도한다. 증명이 완료되면, 이는 무한한 상태공간에 대해서도 해당 정리가 참임을 의미한다. 즉, 완벽한 검증을 제공한다. 정리 증명은 모델 체크와 달리 탐색이 아니라 추론 방법이다. 공리와 가정으로부터 목표 명제를 증명규칙에 따라 유도해 나가는 과정이다.

2. 적용 범위

이론적으로 범위 제한이 없다. Turing-complete한 프로그램의 속성도 증명 가능하다. 스마트 컨트랙트의 임의의 성질을 증명할 수 있다. 하지만 실무 적용은 주로 핵심 알고리즘이나 수학적 정합성 검증에 한정된다. 예컨대, 복잡한 금융 계산 컨트랙트에서 이자 계산 공식을 증명하거나, 랜덤 비콘 생성기가 공정함을 증명하는 식이다. 또는 새로운 스마트 컨트랙트 언어의 타입시스템이 안전함을 증명하기도 한다. 일상적인 컨트랙트 코드 전부를 증명하는 것은 비용 상 어려우므로, 가장 중요한 부분에 집중하는 경향이 있다.

3.자동화

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

완전히 자동화되기 어렵다. 자동 정리 증명이라는 분야도 있지만, 일반적으로 프로그램 검증에는 인력의 투입이 필요하다. Coq의 SMT 자동화 tactic이나 Isabelle의 Sledgehammer처럼 보조도구는 있으나, 수동 보조 없이는 대부분 현실 문제를 풀지 못한다. 즉, 인터랙티브를 전제로 한다. 이는 사람이 수동으로 증명해야 하므로 시간도 오래 걸리고, 전문성도 요구된다. 부분적으로 검증 조건을 자동 생성해주는 도구도 있어 개발자 부담을 줄이고자 하지만, 여전히 남은 증명 의무가 생긴다.

4. 한계

가장 큰 한계는 비용이다. 시간 비용, 인력 비용 모두 크다. 일주일이면 끝낼 코딩을 증명하려면 몇 달이 걸릴 수 있다. 또, 증명한 속성이 유의미한지도 문제다. 또한 스스로가 버그가 없는지 확신해야 한다. 사람이 하는 작업이라 실수 가능성이 있고, 증명 도중 잘못된 보조정리를 세우면 잘 못 된 증명이 될 수 있다. 다만 최종적으로 증명 검증기는 proof term을 체크하므로, 논리적 오류는 없게 된다. 또 다른 현실적 문제는 코드와 증명의 연계다. 증명은 추상 모델이나 사양에 대해 이뤄지는데, 실제 구현 코드가 개선되면 증명도 업데이트가 필요하다. 개발 사이클 내내 증명을 유지하는 건 번거롭기 때문에, 애자일 개발에는 안 맞는다. 그래서 코드가 확정된 후 중요한 부분만 일회성으로 증명하는 사례가 많다.

다. 추상 해석

1. 메커니즘

추상 해석은 프로그램의 가능한 모든 실행을 수학적으로 추상화된 의미로 해석하여, 일반적 성질을 자동 추출하거나 오류를 검출하는 정적 분석 기법이다. 쉽게 말해, 프로그램을 실제 실행하지 않고, 각 명령이 상태에 미치는 영향을 수식으로 계산하여, 도달 가능한 값들의 범위를 근사적으로 파악한다. 이는 컴파일러 최적화나 정적 분석 도구의 이론적 기반으로 활용된다.

2. 적용 범위

추상 해석은 정적 분석 도구 형태로 주로 접할 수 있다. 스마트 컨트랙트의 경우, Slither, Securify 등 일부 도구들이 데이터 흐름 분석이나 패턴 매칭에 이 개념을 쓴다. 예를 들어 Slither는 CFG(Control Flow Graph)를 만들어 변수 정의-사용체인을 따라가며, 특정 함수 인자가 msg.sender에서 비롯되어 not onlyOwner에 들어가는지 탐지한다. 이는 추상 해석으로 볼 수 있다. 또한 언어 기반으로, Solidity 코드를 SSA 폼으로 변환해 부정가능한 조건이 있는지, 테드코드인지, 특

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

정 패턴에 부합하는지 등을 계산한다.

3. 자동화

자동화되며, 일반적으로 컴파일 타임에 수행된다. 개발자가 신경쓸 건 많지 않고, 도구가 경고/정보를 제공한다. OpenZeppelin의 static analyzer나, Ethereum 자체의 오버플로우 검사기도 추상 해석 개념에 가깝다. 일단 작성된 분석기는 많은 코드를 빠르게 분석할 수 있어 규모 확장성이 좋다.

4. 한계

추상 해석은 근본적으로 근사이므로, 오탐이 흔하다. 모든 가능 경로를 커버하려다 보니, 실제로는 일어나지 않을 상황도 고려해 경고를 낼 수 있다. 반대로, 가정에 따라 놓치는 케이스도 가능하다. 예컨대 간단히 Interval 분석을 하면 $x \in [0,100]$ 라 해서 안심했는데, 사실 경계조건 버그로 101 될 수도 있는데 근사 분석이 못 잡는 경우다. 이건 분석자의 trade-off 설정에 따라 다르다. 또한 정교한 추상 해석은 구현이 까다롭고, 각 언어 특성을 모두 포괄하기 힘들다.

라. 심볼릭 실행

1. 메커니즘

심볼릭 실행은 프로그램의 경로를 따라가며 입력을 기호로 두고 수식화한 후, 각 경로 조건을 논리 문제로 풀어보는 방식이다. 따라서 버그가 존재하면 구체적 입력을 찾아주는 점이 특징이다. 이는 곧 모델 검증의 완전탐색과 테스트의 구체 사례 장점을 결합한 접근으로 볼 수 있다.

2. 적용 범위

심볼릭 실행은 일반 프로그램에도 쓰이지만, 스마트 컨트랙트에 특히 적합한데, 왜냐하면 컨트랙트는 외부 의존이 비교적 적고, 상태 공간이 그래도 전형적 프로그램보다 작다고 볼 수 있기 때문이다. 실제로 Oyente, Mythril 등은 수만 개의 메인넷 컨트랙트를 심볼릭 실행으로 분석해 많은 취약점을 자동발견했다. 다만 경로가 너무 많거나, 반복 구조가 크면 한계에 부딪힌다.

3. 자동화

원클릭으로 실행 할 수 있다. Mythril은 명령 한 번으로 수분 내 결과를 주고, Manticore는 사용자 스크립트로 특정 상황만 심볼릭 실행해볼 수도 있다. 자동화 측면에선 모델체커와 유사하지만, 완전탐색은 보장 못하므로, 테스트 자동화의 연

스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발			
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

장 정도로 보면 된다.

4. 한계

경로 폭발, 제약 복잡도 외에도, 모델링 문제도 있다. 일반적으로 단일 컨트랙트 분석에선 환경을 단순화하지만, multi-contract 경우 일일이 고려해줘야 한다. 이게 부족하면 false negative 위험이 있다. 또한 어떤 경로는 solver가 풀지 못할 수 있고, 그 경우 그 경로의 버그는 못 찾는다.

마. 퍼징

1. 메커니즘

퍼징은 프로그램에 무작위 또는 전략적으로 다양한 입력을 제공하고 실행하여, 예외나 취약점이 발생하는지 모니터링하는 동적 검증 기법이다. 스마트 컨트랙트 퍼징 도구로는 Echidna가 유명하며, 컨트랙트 함수 호출을 랜덤 조합으로 수행하면서 assert 위반이나 Ether 도난 등의 실패 조건을 체크한다.

2. 적용 범위

퍼징은 실제 EVM 상에서 코드를 돌리므로, 솔리디티 구현 그 자체를 검증한다. 개발자는 컨트랙트의 property를 assert문 등으로 코드에 넣어두고, Echidna가 그것을 깨뜨리는 임의 호출 시퀀스를 찾도록 한다. 특정 버그는 퍼징만으로 찾기 어려울 수 있지만, 의외의 상태 조합에서 실행 오류를 찾는 데 효과적이다. Ethereum 테스트넷에서 대규모 퍼징을 돌리면, 가스 한계 상황 등도 자연스럽게 탐구된다.

3. 자동화

퍼저가 알아서 입력을 생성하고 실행한다. 개발자는 실패 조건만 명시하면 된다. 커버리지 기반 기법으로 효율을 높이는 등 연구도 활발하다. 다만 무한정 실행할 수 없으니, 수 분~수 시간 정도 돌리고 나서 결과를 수집한다.

4. 한계

Randomness에 의존한다. 특정 버그 트리거 입력을 못 찾아내면 그냥 통과해 버린다. 심볼릭 실행처럼 논리적 보장을 못 하니, “못 찾았으니 없겠지”라고 확신 하긴 어렵다. 또한 블록체인 퍼징은 노드 동기화 등이 필요해 세팅이 까다롭고 느릴 수 있다. 예를 들어, Echidna는 local EVM vm을 사용하지만, 복잡한 상호작용은 느려질 수 있다. 그리고 assert로 명시 안 한 보안 문제를 알아채지 못한다.

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

VII. 스마트 컨트랙트 정형 기법 적용 사례

정형 기법과 검증기법들은 이론적으로 타당할 뿐 아니라, 여러 실전 사례에서 가치가 입증되고 있다. 이번 장에서는 대표적인 스마트 컨트랙트 사례들에 정형 기법/검증을 적용한 경우를 살펴본다. Ethereum 기반 토큰 컨트랙트, 탈중앙 금융 시스템, 그리고 서비스 수준 계약, 스마트 컨트랙트를 활용한 대출 시스템, 학술적 관점 등의 분야에서, 어떠한 정형 기법이 쓰였고 어떤 결과를 얻었는지 소개한다. 각 사례는 적용된 구체적 방법, 검증 절차, 발견된 문제와 해결, 그리고 실무적인 활용을 설명한다.

가. Ethereum 토큰 컨트랙트에 대한 정형 기법

1. 개요

ERC-20 토큰은 이더리움에서 가장 널리 사용되는 스마트 컨트랙트 표준으로, 수만 개 이상의 토큰이 이 표준을 구현하고 있다. 토큰 컨트랙트의 주요 기능은 잔고 관리와 전송으로 단순해 보이지만, 방대한 가치가 걸려있기에 작은 논리 버그도 큰 문제를 야기할 수 있다. 2017~2018년 경, 여러 토큰 컨트랙트에서 오버플로우, 잘못된 초기화 등의 버그로 혼란이 발생하자, CertiK 등의 보안업체들은 ERC-20 토큰에 대한 정형 검증을 시도하였다.

2. 적용 기법

CertiK는 자체 개발한 정형 검증 프레임워크를 사용했는데, 이는 심볼릭 모델 체킹 기법에 속한다. 구체적으로, CertiK 엔지니어들은 토큰 컨트랙트 코드를 수학적 모델로 변환하고, ERC-20 표준의 요구사항을 템플릿 불변식으로 명세하여, 모델 체커로 검증하였다. 예를 들면, “transfer 후에도 총 공급량은 변함없다”, “승인한 토큰만 인출된다”, “이중 지불 불가” 등의 속성을 일반화된 논리 형태로 정의했다. 이 명세들은 CertiK의 라이브러리에 누적되어 있어, 매번 일일이 쓰지 않고도 각 토큰 계약에 적용할 수 있는 재사용 가능한 형태였다.

3. 검증 절차

우선 분석 대상 토큰 컨트랙트를 바이트코드나 소스 레벨에서 파싱해, CertiK의 수학 모델로 변환했다. 이 모델은 해당 컨트랙트가 취할 모든 행동과 상태 변수를 포함한다. 그런 다음 미리 준비된 ERC-20 속성 템플릿을 이 특정 모델에 주입하여, 상태 공간 탐색을 수행하였다. CertiK는 이 때 BMC(Bounded Model

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

Checking)와 심볼릭 실행을 혼합한 엔진을 사용한 것으로 알려져 있다. 검증 엔진은 “모든 함수 호출 시나리오에서 속성이 성립하는가”를 자동으로 살폈고, 만약 위반되는 경우 counterexample을 제공했다. CertiK 보고서에 따르면, 2022년까지 416개의 토큰 컨트랙트에서 이러한 정형 검증을 수행했고, 수많은 프로젝트에서 기본 기능의 안전성을 수학적으로 확인받았다.

4. 결과

대부분의 검증에서 ERC-20 표준 속성이 충족됨이 확인되었으나, 일부 컨트랙트에서는 사소하지만 표준과 어긋나는 동작이 발견되었다. 예를 들어, 어떤 토큰 구현은 특수한 예외 케이스에서 transfer 함수가 false를 리턴하지 않고 revert해 버리는 비표준 동작이 있었는데, 검증을 통해 발견되어 수정되었다. 또 몇몇 토큰은 총 발행량 상한이 동적으로 변할 수 있는 설계를 가졌는데, 검증결과 특정 시나리오에서 총합 불변성이 깨질 위험이 있음을 경고하여, 개발자가 초기값 설정을 바꾸는 계기가 되었다. CertiK는 이러한 검증 결과를 감사 보고서에 포함시켜 신뢰성을 강조했으며, 후에 이 방식을 일반 스마트 컨트랙트 감사에도 확대 적용했다.

나. DeFi 시스템에 대한 정형 검증 적용

1. 개요

DeFi 프로토콜들은 복잡한 경제 논리와 다중 스마트 컨트랙트로 구성되어 있어, 보안 취약점이 발생할 경우 대규모 자산 피해로 직결된다. 2021년 이전까지 Uniswap, Compound 등에서 잇따른 해킹사건이 발생한 후, 많은 DeFi 프로젝트들이 정형 검증을 도입하기 시작했다. 여기서는 SushiSwap의 새로운 유동성 풀 엔진인 Trident에 대한 정형 검증 사례를 살펴본다. SushiSwap은 2022년 Trident 출시 전, Certora사와 협력하여 프로토콜에 정형 검증을 적용했다.

2. 적용 기법

Certora사의 Certora Prover라는 도구가 사용되었다. Certora Prover는 SMT solver 기반의 맞춤형 정형 검증 엔진으로, 스마트 컨트랙트 코드를 직접 분석하지만, 개발자가 규칙(rule)이라는 형태로 검증 속성을 작성해야 한다. 이는 일종의 Hoare 논리 스타일의 불변식 명세라고 볼 수 있다. SushiSwap 개발팀과 Certora 연구원들은 Trident 프로토콜에서 지켜야 할 여러 가지 비즈니스 불변성을 도출하였다. 예를 들어, “풀에서 유동성을 인출하면 남은 총액은 이전 총액 - 인출량과 정확히 일치해야 한다”, “수수료 계산으로 인해 일시적으로 생길 수 있는 잔액 오

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

차는 일정 한계 이하이어야 한다”, “공격자가 임의의 순서로 함수 호출을 시도해도 자금이 생성되지 않는다” 등이었다. 그런 다음 이러한 요구사항을 Certora Prover의 규칙 언어로 명세화했다.

3. 검증 절차

Certora Prover는 솔리디티 코드를 입력으로 받아 경로 탐색과 SMT 솔버로 규칙 검증을 시도한다. 개발팀은 Trident의 핵심 컨트랙트들을 대상으로 규칙을 작성하고, Certora 서버에서 연산을 수행했다. 이 과정은 완전 자동은 아니어서, 초기에는 몇몇 규칙이 false positive 경고를 내면 이를 규칙 세분화나 환경 가정 추가로 수정하는 식의 반복 과정이 있었다. 예를 들어, 외부 가격 피드가 임의로 변한다고 가정하면 풀 불변식이 깨진다는 경고가 나왔으나, 실제로 가격 피드는 신뢰된 것으로 제한된다는 가정을 규칙에 넣어 해결했다. 이렇게 규칙 세트가 안정화된 후, Certora Prover는 “규칙 X, Y, Z 모두 검증 통과”를 보고했다. 특히 한 규칙에서, 특정 순서의 연속 호출이 없다면 절대 풀에서 자산이 소실되지 않음을 증명한 것이 핵심 성과였다.

4. 결과

검증 과정에서 SushiSwap 팀은 심각한 버그 하나를 사전에 발견할 수 있었다. Certora Prover가 제시한 한 반례 시나리오에서, 유동성 풀에 A, B 두 토큰이 있는데, 특정 희귀한 순서로 함수를 호출하면 풀의 내부 회계에 오류가 생겨 일부 자산이 이상하게 계산되는 케이스를 찾아냈다. 이는 코드 리뷰 단계에서 간과된 논리 경로였는데, 자동 검증이 짚어준 것이다. 팀은 즉시 코드를 수정해 이 문제를 제거했고, 이후 공개된 Trident 코드에는 해당 취약점이 존재하지 않았다. Certora에 따르면, 이 버그는 exploitation 시 약 수백만 달러 손실을 유발할 수 있는 심각한 것이라고 한다. 추가로, 여러 작은 최적화 사항도 발견되었다. 예를 들어 불필요한 조건문 분기가 명세상 언제나 false임이 증명되어 제거되어 가스비를 줄였거나, 상수 설정이 여유 범위가 부족함이 검증되어 넉넉하게 재조정된 경우 등이 있었다. 최종 결과로, SushiSwap Trident는 Certora Prover 검증을 통한 속성 목록과 함께 공개되어, 투자자들에게 신뢰를 주었다.

다. SLA 기반 계약의 정형 기법

1. 개요

스마트 컨트랙트는 금융 이외에도 물류, 클라우드, IoT 등 다양한 분야에서 서비스 수준 계약(SLA) 이행에 활용되고 있다. 예를 들어, 클라우드 서비스 업체와

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

고객 간에 “월 가동시간 99.9% 미만이면 환불” 같은 SLA 조항을 스마트 컨트랙트로 자동화할 수 있다. 이러한 스마트 법률 계약 분야에서도 정형 기법이 시도되고 있다. 예로, 보험 SLA 계약의 사례를 보면, 한 보험사가 항공편 지연 보험을 스마트 컨트랙트로 구현한다고 할 때, 계약 내용은 “항공편이 2시간 이상 지연되면 자동으로 배상금 X를 지불”이라는 SLA 조항이다. 이 계약의 정확성과 악용 방지 여부를 정형 모델로 검증해볼 수 있다.

2. 적용 기법

이 사례에서는 상태 전이 시스템 모델링과 모델 검증을 활용했다. 보험 스마트 컨트랙트를 TLA+ 같은 언어로 명세화하고, 중요한 속성을 검증하는 접근이다. 먼저, 계약의 상태를 정의한다: $\text{State} = \{\text{Scheduled}, \text{Delayed}, \text{Paid}\}$ 같은 식으로 항공편 상태를 모델링하고, Paid는 배상 완료 상태라고 두자. 상태 전이 함수는 Oracle로부터 지연 통보를 받으면 Scheduled \rightarrow Delayed로 가고, 컨트랙트가 지불 트랜잭션을 실행하면 Delayed \rightarrow Paid로 전이된다고 정의한다. 그 외 시간 제한, 보험가입자 정보 등 부가 요소도 있다. 중요한 속성은 “지연이 발생하면 결국 보상이 지불된다”(활성 속성)와 “이중 지급은 없다”(안전 속성) 등이다. 이를 TLA+의 임시 논리로 표현하면, $\Box(\text{DelayReported} \rightarrow \Diamond\text{Paid})$ and $\Box(\text{Paid} \rightarrow \Box\neg\text{PaidAgain})$ 등의 식으로 된다. 여기서 PaidAgain은 한 번 Paid된 후 다시 Paid되는 상황을 의미하는 상태 플래그다. 이러한 명세를 써놓고, TLA+ TLC로 검증한다.

3. 검증 절차

모델 검증기(TLC)가 보험 계약의 모든 가능한 이벤트 순서를 탐색한다. Oracle이 거짓으로 지연 정보를 보내는 경우, 지연 없이도 Paid로 가는 시나리오가 탐색될 것이다. 만약 스마트 컨트랙트에 그런 경우를 막는 검증 (Oracle 신뢰 등)이 없다면, “지연 통보 없이 Paid됨”이라는 불변성 위반 경로를 발견할 수 있다. 실제 코드에서는 Oracle 서명을 검사하여 이 경우를 막아야 한다. 또한 두 번 지연 통보가 들어오는 경우 등도 모델에 포함된다. 검증기는 “Paid 상태에서 또 Paid로 전이”하는 케이스가 가능한지 체크하고, 만약 Oracle이 중복 통보를 할 수 있는데 컨트랙트가 방비를 안 했으면 PaidAgain이 true 되는 경로를 보여줄 것이다. 그러면 설계를 수정해야 한다. 개발자는 이 결과를 보고, 중복 통보 시 두 번째는 무시하도록 상태 분기 추가 등 설계를 변경한다. 다시 모델 검증을 돌려, 이제 “이중 지급 없음” 불변성이 만족됨을 확인한다.

4. 결과

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

정형 기법을 통해, 몇 가지 잠재 버그를 미리 제거 할 수 있다. 예를 들어, Oracle의 잘못된 행동이 있더라도 고객에게 부당 이익이 안 가도록 컨트랙트를 설계할 수 있고, 또한 SLA 조항의 모호함도 발견될 수 있다. “2시간 지연“의 기준을 명세하면서, 만약 출발이 취소된 경우 처리를 어떻게 할지 등 추가 케이스가 드러났다. 모델에서는 항공편 상태에 “Canceled“를 추가하고, Canceled일 때 지불은 안 한다고 결정, 모델과 실제 계약서 모두에 반영했다. 이로써 법률적 분쟁 여지를 줄일 수 있다.

라. 대출 시스템에 대한 정형 기법

1. 개요

스마트 컨트랙트를 활용한 대출 관리 시스템은 자산 흐름과 권리 이전을 자동화함으로써 금융 거래의 신뢰성과 투명성을 높인다. 이러한 시스템에서는 대출 약정, 상환, 연체, 담보 몰수 등 다양한 상태 변화를 정확하고 오류 없이 처리해야 하며, 실세계에서 발생할 수 있는 모든 경로를 안전하게 다루는 것이 필수적이다. 본 사례에서는 대출 관리용 스마트 컨트랙트를 대상으로 TLA+를 활용하여 정형 명세와 검증을 수행한 연구가 존재하고, 활발히 해당 분야에 대한 탐구가 이루어지고 있다. 대출 시스템은 신용 대출과 담보 대출을 모두 지원하며, 다양한 대출 시나리오를 포괄하는 복합 상태 모델을 구축하고, 이를 통해 시스템의 안전성과 진행성 보장을 수학적으로 검증하였다.

2. 적용 기법

본 대출 관리 시스템에서는 TLA+를 활용하여 스마트 컨트랙트의 모든 상태와 동작을 명확하게 모델링하였다. 대출 생성, 상환, 조기 상환, 연체 발생, 디폴트 확정, 담보 몰수, 대출 종료 등의 전 과정을 체계적으로 기술하였으며, 각각의 상태 전이에 필요한 조건과 결과를 수학적으로 정의하였다. 신용 대출과 담보 대출의 구분은 명확히 이루어졌으며, 신용 대출은 연체 발생 시 정해진 최대 허용 기간을 초과하면 디폴트로 전환되고, 담보 대출은 연체 발생 후 정해진 조건 충족 시 담보를 몰수하는 흐름으로 설계되었다. 이자율 계산, 조기 상환 수수료 부과, 연체 이자 누적 등 복잡한 금융 연산 역시 세밀하게 모델링되었으며, 실시간 변화에 따른 정합성을 유지하도록 명세되었다.

검증 대상 속성으로는 대출 잔액이 절대 음수가 되어서는 안 된다는 조건, 상환 금액이 약정된 대출 금액과 최대 이자 및 수수료 한도를 초과할 수 없다는 조건, 담보는 연체 조건을 충족할 때만 소유권이 이전되어야 한다는 조건과 같이 시스템의 안전성을 보장하는 항목들이 포함되었다. 또한 모든 대출은 일정 시간 내

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

에 상환 완료 또는 디폴트 종료 상태에 도달해야 한다는 진행성 조건도 설정하였다.

이러한 명세를 기반으로, 모델 검증 도구인 TLC를 이용해 가능한 모든 상태 전이 경로를 탐색하고, 시스템이 정의된 안전성과 진행성 속성을 일관되게 만족하는지 확인하였다. 초기 상태에서는 대출이 생성되지 않은 상태에서 출발하여, 다양한 시나리오를 통해 대출 생성, 상환, 연체, 조기 상환, 담보 몰수, 종료까지의 흐름을 자동으로 검증하였다.

3. 검증 절차

TLA+로 작성된 명세를 기반으로 모델 검증기인 TLC를 사용하여 대출 시스템의 모든 가능한 상태 전이를 자동으로 탐색하였다. 초기 상태에서는 대출이 생성되지 않은 상태에서 시작하여, 대출 생성, 상환, 조기 상환, 연체 발생, 담보 몰수, 대출 종료 등 다양한 경로를 모두 고려하였다. 각 단계에서 정의된 Safety 속성과 Liveness 속성이 항상 유지되는지 검증하였다. 특히 조기 상환 시 조기 상환 수수료가 올바르게 적용되는지, 연체 발생 시 연체 이자가 누적 적용되며 최대 연체 기간 초과 시 디폴트로 전이되는지, 디폴트 후 담보 소유권이 정확히 이전되는지 등 복합적인 조건을 염밀히 검증하였다.

4. 결과

모델 검증 결과, 대출 시스템은 정의된 모든 Safety 및 Liveness 속성을 만족함이 확인되었다. 대출 잔액이 음수로 떨어지는 경우나, 상환 금액이 약정 금액을 초과하는 경우, 담보 소유권이 부정확하게 이전되는 경우 등의 오류가 발생하지 않음을 수학적으로 입증하였다. 또한, 모든 대출은 정상 종료되거나 디폴트 처리로 귀결되어, 무한 진행이나 교착 상태 없이 시스템이 종료됨을 확인하였다. 해당 연구를 통해, 복잡한 금융 로직을 가지는 스마트 컨트랙트 시스템도 정형 기법을 통해 높은 수준의 신뢰성과 안전성을 확보할 수 있음을 입증하였다. 특히 대출 관리와 같은 고위험 금융 응용 분야에서는 정형 명세와 검증의 중요성이 실질적으로 증명된 사례라 할 수 있다.

마. 스마트 컨트랙트 Security Policy Model (SPM)에 대한 정형 기법

1. 개요

스마트 컨트랙트는 다양한 자산 관리, 권리 이양, 상태 변화 등의 민감한 작업을 자동화하는 시스템으로, 설계 단계부터 명확한 보안 정책을 수립하고 이를 수학적으로 검증하는 것이 필수적이다. 특히 탈중앙화된 환경에서는 중앙 통제가 불

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

가능하기 때문에, 시스템 내부 규칙만으로 안전성과 신뢰성을 보장해야 한다. 스마트 컨트랙트에 요구되는 핵심 보안 속성들을 별도의 SPM으로 정형화하고, 이를 TLA+를 이용해 명세하고 검증할 수 있다. 이 접근 방식은 단순히 기능적 요구사항을 충족시키는 것을 넘어서, 시스템이 모든 실행 경로에서 보안 속성을 일관되게 유지하는지를 수학적으로 명세 할 수 있다.

2. 적용 기법

먼저 스마트 컨트랙트가 수행하는 기능과 예상되는 위험 시나리오를 분석하여, 핵심 보안 요구사항을 도출한다. 여기에는 자산 보존성, 권한 검증, 재진입 공격 방지, 거래 원자성 등이 포함될 수 있다. 자산 보존성은 시스템의 자산 총량이 모든 트랜잭션 이후에도 변하지 않아야 하는 것을 의미하고, 권한 검증은 민감한 연산이 사전에 인증된 사용자에 의해 수행되어야 함을 요구한다. 재진입 공격 방지는 외부 호출 중 내부 상태의 불안정성이나 이중 호출을 방지하기 위한 설계이며, 거래 원자성은 트랜잭션 실행이 부분 완료 없이 전부 성공 또는 전부 실패해야 한다는 조건을 담고 있다. 이러한 보안 요구사항을 기반으로 스마트 컨트랙트의 상태 및 동작을 수학적으로 모델링하고, TLA+를 사용하여 정형 명세를 작성하는 접근이 가능하다. 각 보안 정책은 상태 전이 과정에서 항상 만족되어야 할 불변성과 Safety, Liveness 속성으로 표현 할 수 있다.

3. 검증 절차

정의된 명세를 기반으로, 모델 검증 도구인 TLC를 이용하여 모든 가능한 상태 전이 경로를 자동 탐색하고 보안 속성의 유지 여부를 확인할 수 있다. 초기 상태는 정해진 자산 분포와 사용자 권한 상태로 설정하며, 이후 다양한 트랜잭션 시나리오, 외부 호출, 실패 및 복구 과정을 포함하여 시뮬레이션을 수행한다. 검증 과정에서는 승인되지 않은 사용자가 민감 연산을 시도하는 경우, 외부 호출 중 재진입 공격이 발생하는 경우, 트랜잭션 중 오류로 인해 자산이나 권한 상태가 불안정해지는 경우 등을 중점적으로 검토할 수 있다. 모든 상태 전이 경로에서 자산 보존성, 권한 검증, 재진입 방지, 거래 원자성 조건이 일관되게 유지되는지를 점검하는 방식으로 진행된다.

4. 결과

이러한 검증 과정을 통해 스마트 컨트랙트가 설계한 보안 요구사항을 일관되게 만족하는지 여부를 조기에 확인할 수 있을 것으로 기대된다. 특히 승인되지 않은 사용자의 민감 연산 접근 차단, 재진입 공격으로 인한 상태 불안정성 제거, 트랜잭션 실패 시 일관성 없는 상태 방지 등의 보안 측면에서 신뢰성을 높일 수 있

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

다. 또한 코드 리뷰나 수동 테스트만으로는 발견하기 어려운 복합 경로상의 잠재적 취약점까지 조기에 식별하고 대응할 수 있는 이점이 존재한다. 설계 단계에서부터 보안 요구사항을 수학적으로 검증하는 체계를 갖춤으로써, 고 신뢰와 고 안정성을 요구하는 스마트 컨트랙트의 신뢰성을 한층 더 강화 할 수 있다.

스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발			
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

VIII. 스마트 컨트랙트 정형 기법 도구 및 프레임워크

다양한 정형 기법들을 실무에 적용하려면 적절한 도구나 프레임워크의 도움이 필수적이다. 이번 장에서는 스마트 컨트랙트 분야에서 널리 쓰이거나 주목할 만한 정형 기법에 관련된 도구들에 대해 기술한다. TLA+ Toolbox, KEVM, VeriSol, Mythril, Slither 등의 도구를 다루며, 각 도구의 내부 구조와 기능, 장단점, 실무 활용법 등에 대해 기술한다. 또한 이러한 도구들을 상호 보완적으로 사용하는 전략과, 개발 파이프라인에 통합하는 방안을 제안한다.

가. TLA Toolbox

1. 개요

TLA+ Toolbox는 TLA+ 언어를 사용한 정형 명세 작업을 지원하는 통합 개발 환경이다. Leslie Lamport가 주도 개발한 Toolbox는 사양 편집기, 구문 검사기, 모델 구성 창, TLC 실행기, 오류 추적 뷰어 등을 포함한다. TLA+ Toolbox를 통해 사용자는 TLA+ 명세를 작성하고, 몇 번의 클릭만으로 모델 검증을 수행할 수 있다.

2. 내부 구조 및 기능

Toolbox는 Eclipse 기반 GUI로 구현되어 있다. 주요 구성요소는 첫 째로, Spec Editor, 구문 하이라이트와 에러 검출 등을 제공하는 편집기, 두 번 째로, Model Configurator, 여기서 상태공간 제한, 검증하고픈 불변성/속성, 탐색 깊이 등을 설정한다. 세 번째로 TLC Model Checker Interface, 설정된 모델을 TLC 엔진에 전달하고, 진행 상황을 모니터링하며, 완료 후 결과를 표시한다. 네 번째로, Error Trace Viewer, 만약 오류가 발견되면, 그 실행 경로를 단계별로 보여주는 창이 열린다. 각 단계의 변수 값과 상태를 볼 수 있어 디버깅에 용이하다. 마지막으로, TLAPS Proof Manager, 정리 증명용 보조 도구도 있으나, Toolbox 내에서 주로 사용되는 건 모델 체크 부분이다.

3. 입력 방식

사용자는 .tla 확장자의 텍스트 파일로 명세를 작성하며, .cfg 파일로 검증 환경을 지정한다. Toolbox에서는 이를 GUI 폼으로 입력하도록 도와준다. 예를 들어, 모델 페이지에서 “Constant parameter N = 3” 등으로 설정하면 .cfg에 자동 반영 된다. 불변성은 명세 내 선언한 이름을 체크박스로 선택하여 활성화할 수 있다. 그러면 TLC가 그 불변성을 검사하게 된다. 이러한 GUI 조작은 정형 기법에 익숙

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

하지 않은 개발자도 쉽게 설정하도록 돋는다.

4. 자동화

TLA+ Toolbox는 수동 명세 작성 이후의 검증 과정을 모두 자동화하지만, 개발 파이프라인에 통합되어 지속적 검증은 직접 스크립트화를 해야 한다. UI 중심이다 보니, 대량 반복 실행은 CLI 모드의 TLC를 쓰거나 해야 한다. 그럼에도 다수 사용자들이 GUI 환경에서 설계 검증하는 패턴으로 쓰고, CI 통합은 비교적 드물다.

5. 장단점 및 실무 활용

Toolbox의 장점은 완성도 높은 사용자 경험이다. 산업 엔지니어들이 사용하기 쉽게 만들어져, 아마존 등에서 실제 활용되었다. GUI를 통해 복잡한 수식도 편히 쓰고, 결과를 해석하기 좋다. 또 다중 모델 관리가 쉬워, 한 명세에 대해 여러 시나리오 모델을 저장/검증할 수 있다. 단점으로, IDE 특유의 무거움과, 큰 모델 검증 시 메모리/성능 한계가 있다. 터미널에서 TLC를 돌리면 로그만 보이지만 더 가볍게 큰 탐색을 할 수 있는데, Toolbox는 Eclipse 기반이라 과부하 시 멈춤 현상 등이 보고된다. 실무에서는, 주로 설계 리뷰 단계에서 Toolbox를 사용한다. 예컨대 컨트랙트 작성 전, 프로토콜 디자이너가 TLA+로 시나리오를 만들어 Toolbox로 검증한 후 개발자에게 명세와 결과를 전달한다. 또는 개발자 자신이 컨트랙트 간 복잡한 인터랙션을 이해하려고 TLA+를 써볼 수 있다.

나. KEVM

1. 개요

KEVM은 K 프레임워크로 정의한 Ethereum Virtual Machine(EVM)의 형식적 의미론이다. K Framework는 일종의 메타-언어로 프로그래밍 언어의 형식 정의를 작성하면, 실행기, 모델체커, 검증기 등을 자동 생성해주는 도구다. KEVM은 K로 기술된 EVM 바이트코드의 작동 규칙 모음으로, 이를 통해 EVM 프로그램을 정확히 실행하거나 검증할 수 있다.

2. 내부 구조 및 기능

KEVM은 크게 두 부분으로 이뤄진다. 첫 번째, EVM Semantics in K, 이더리움 엘로우페이퍼의 내용을 K 언어로 구현한 것이다. K의 문법 규칙으로 바이트코드 명령어들의 효과를 rewrite rule로 나열한다. rule <opcode ADD> H:Int, T:IntStack => (H +Int T) ... 식으로 EVM 스택 연산 규칙이 들어있다. 두 번째, K Tools, K 프레임워크 툴체인은 이 K 정의를 받아들여, 인터프리터와 단위 테스트 실행기,

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

증명 지원기를 생성한다. KEVM팀은 이를 활용해 EVM의 공식 testsuite를 전부 통과시켜, KEVM의 정확성을 검증했다. 또한 K에는 reachability logic prover라는 검증기가 있어, 사용자가 EVM 바이트코드에 대해 속성을 주면 그것이 항상 성립함을 증명하거나 반례를 찾는다.

3. 입력 방식

KEVM을 활용하려면, 우선 Solidity 코드를 바이트코드로 컴파일해야 한다. 그런 다음, 사용자는 증명하고자 하는 요구사항을 K의 형식으로 작성한다. 예를 들어, “함수 f를 호출한 후 상태 변수 x는 늘 증가한다“를 증명하고 싶다면, K의 규칙 형식으로 pre/post 상태를 패턴 매칭하여 requires/ensures 절을 적는다. K 증명 구문은 익숙지 않은 개발자에게 어려울 수 있어, 보통 이 작업은 K 전문가나 플로우를 아는 사람들이 맡는다. 입력은 .k 파일로 EVM 상태를 기술하는데, KEVM은 이걸 internal representation으로 처리한다. 입력 없이 KEVM을 단순 interpreter 모드로 돌리면, 실제 EVM과 동일하게 프로그램을 실행시켜 최종 상태를 얻을 수도 있다. 예컨대 “바이트코드 X를 입력 Y로 실행“을 시킬 수 있어, 시뮬레이터처럼 쓰는 것도 가능하다.

4. 자동화

K 프레임워크의 강력함은 실행기+검증기 일체화에 있다. KEVM은 공식 이더리움 테스트를 모두 패스했으므로, K로 정의한 규칙이 실제 EVM과 동등함을 의미한다. 따라서 이 규칙을 기반으로 증명된 속성은 곧 실제 EVM에서 증명된 것과 같다. 이는 SMT 등으로 EVM을 근사모델로 만든 것보다 신뢰도가 높다. 반면, K 자체에 진입장벽이 있고, 증명 과정이 자동이라기보단 수작업 구성이다.

5. 장단점 및 실무 활용

정확성이 장점이다. KEVM 기반으로 하면 EVM 모델링 오류 걱정이 없다. 또한 표현력도 하나의 장점이다. 임의의 EVM 속성을 증명 가능하고, 다른 EIP도 K로 미리 모델링해 영향평가를 할 수 있다. 반면, 사용 난이도 높다는 단점이 있다. 솔리디티 개발자가 바로 쓰긴 힘들고, Runtime Verification 같은 전문 업체의 컨설팅이 필요했다. 또, 성능의 단점이 부각된다. K기반 검증은 무겁고 복잡해서, 간단한 인바리언트 증명도 오래 걸릴 수 있다. 그래서 KEVM은 주로 핵심 프로토콜이나 VM 레벨 검증에 쓰였고, 개별 프로젝트에는 많이 안 쓰였다. 하지만 Cardano같은 프로젝트에서는 Plutus 검증에 K를 활용했고, Ethereum 2.0의 일부 Beacon Chain spec도 K로 모델링 검토했다. 실무에서는, 플랫폼 차원에서 도입하는 게 좋다. 예컨대 이더리움 재단이 KEVM을 공식화하여, 컨트랙트 보안 점검서

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

비스로 제공한다면 가치가 클 것이다. 아직 그 단계는 아니지만, KEVM은 중요한 인프라로 발전 중이다.

다. VeriSol

1. 개요

Microsoft Research에서 개발한 스마트 컨트랙트 형식 검증 도구로, Solidity로 작성된 스마트 컨트랙트의 논리적 정확성과 안전성을 검증하는 데 사용된다. 이를 위해 VeriSol은 Solidity 코드를 중간 표현으로 변환한 뒤, Boogie라는 검증 언어를 기반으로 정형 기법을 활용해 스마트 컨트랙트가 사양에 부합하는지를 확인한다. Ethereum 스마트 컨트랙트의 보안 결함 및 오류를 사전에 탐지할 수 있는 강력한 정적 분석 도구이다.

2. 내부 구조 및 기능

VeriSol의 내부 구조는 크게 네 가지 주요 구성 요소로 나뉜다. 첫 번째는 Solidity to Boogie 변환기로, 이는 Solidity 코드를 Boogie 중간 언어로 자동 변환하여 검증 가능한 형태로 만드는 역할을 한다. 두 번째는 사양 명세 처리기로, 스마트 컨트랙트 코드에 포함된 명시적 명세를 Boogie 명세로 해석하여 검증의 기준점을 설정한다. 세 번째는 Boogie 검증 엔진이며, 변환된 중간 언어를 바탕으로 논리 모델을 생성하고 SMT Solver를 통해 사양 위반 가능성을 분석한다. 마지막은 결과 해석기 및 리포터로, 검증 결과를 사용자에게 명확하게 전달하고 문제가 발생한 위치나 조건을 설명하는 역할을 수행한다. 이처럼 VeriSol은 Solidity 코드와 Boogie 언어 간의 매핑을 기반으로 하여 정형 분석을 수행하는 구조를 갖는다.

3. 입력 방식

VeriSol은 기본적으로 사용자가 작성한 Solidity 스마트 컨트랙트를 입력으로 받는다. 사용자는 컨트랙트 내부에 assert, require, invariant, modifies 등의 명령어를 활용하여 명시적으로 검증하고자 하는 조건을 코드에 삽입할 수 있다. 또한 특정 함수에 대한 사전 조건이나 사후 조건 또는 불변 조건을 명시적으로 지정할 수도 있다. 명시된 조건이 없는 경우, VeriSol은 기본적인 상태 safety 속성과 함수 실행의 논리적 일관성 등을 기준으로 분석을 수행한다. 명세와 소스 코드를 함께 포함한 디렉터리를 VeriSol 명령어에 전달하면, 내부적으로 자동 변환 및 분석이 진행된다.

4. 자동화

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

VeriSol은 전반적인 정형 분석 절차를 자동화하여 사용자 개입을 최소화한다. Solidity 코드를 Boogie로 변환하고, 지정된 명세에 따라 SMT Solver를 활용한 정형 검증이 자동으로 이루어진다. 사용자는 단순히 CLI를 통해 .sol 파일을 입력하거나 GitHub 프로젝트를 지정하면, VeriSol이 전체 컨트랙트를 분석하고 사양 위반 여부를 보고서 형식으로 출력한다. 특히 반복되는 검증 과정에서도 자동으로 조건이 보강되거나, 반례가 자동 도출되므로 스마트 컨트랙트 개발자의 디버깅 및 리팩토링 효율이 크게 향상된다. 이러한 자동화는 CI/CD 파이프라인에 VeriSol을 통합함으로써 실시간 스마트 컨트랙트 품질 검증 환경을 구현하는 데에도 효과적이다.

5. 장단점 및 실무 활용

VeriSol은 주로 엔터프라이즈 전용 컨소시엄 체인 환경을 대상으로 개발되었기 때문에, 공개 이더리움 네트워크의 모든 종속성을 완벽하게 지원하기에는 한계가 있었다. 프로젝트가 Azure Blockchain을 기반으로 하거나, Microsoft의 OpenZeppelin과 유사한 표준 라이브러리만 사용하는 경우라면 VeriSol을 무리 없이 활용할 수 있었지만, 그렇지 않은 경우에는 지원되지 않는 구문이 발생할 가능성이 있었다. 또한 VeriSol은 2019년 이후 활발한 업데이트가 이루어지지 않아, 최신 Solidity 문법을 완전히 지원하지 못할 수 있다.

실제 협업에서는 VeriSol 자체보다는, VeriSol의 아이디어를 바탕으로 개발된 다른 도구나 서비스를 활용하는 경우가 더 많다. 예를 들어, 개발팀이 별도의 비용 없이 직접 정형 검증을 시도하고자 한다면 VeriSol을 사용할 수 있지만, 공식 문서와 기술 지원이 제한적이라는 점을 고려해야 한다. 이와 유사한 오픈소스 도구로는 'Solv-verify'가 있으며, Boogie 변환 방식을 기반으로 VeriSol의 영향을 받아 개발되었다.

라. Mythril

1. 개요

Mythril은 스마트 컨트랙트 보안 분석을 위해 개발된 오픈소스 도구로, 이더리움 기반의 Solidity 스마트 컨트랙트를 대상으로 동작한다. 이 도구는 EVM 바이트 코드를 분석하여 취약점을 탐지하며, 특히 symbolic execution, taint analysis, control-flow checking 등의 기법을 활용하여 정적 및 동적 분석을 수행한다. Mythril은 CLI 또는 JSON-RPC를 통해 사용 가능하며, 다양한 보안 약점을 빠르게 탐지할 수 있어 보안 전문가와 개발자 모두에게 유용하다.

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

2. 내부 구조 및 기능

Mythril의 구조는 크게 세 가지 주요 구성 요소로 이루어진다. 첫 번째는 Solidity 또는 바이트코드 입력을 EVM 명령어 레벨로 해석하는 파서와 디코더이다. 이 단계에서 EVM 코드를 분석 가능한 내부 표현으로 변환한다. 두 번째는 symbolic execution 엔진으로, 다양한 실행 경로를 시뮬레이션하고 조건부 분기나 반복문 등을 고려해 취약점 가능성을 추적한다. 세 번째는 탐지된 실행 경로에서 보안 위협 패턴을 식별하는 취약점 탐지 모듈이다. 또한 Mythril은 Ethereum에서 발생한 실제 해킹 사례의 패턴을 정제한 시그니처 기반 분석 기능도 포함하고 있어, 알려진 공격 유형에 대해서는 빠른 진단이 가능하다.

3. 입력 방식

Mythril은 Solidity 소스코드, EVM 바이트코드, JSON ABI 파일을 모두 입력으로 지원한다. 일반적으로는 Truffle 또는 Hardhat 프로젝트 폴더 내에서 Solidity 파일을 대상으로 Mythril을 실행하거나, 배포된 스마트 컨트랙트 주소를 입력하여 실제 메인넷 상의 코드에 대해서도 분석을 수행할 수 있다. 입력 형식에 따라 내부적으로 디코딩 및 변환 과정을 거쳐 symbolic execution이 가능한 형태로 변환되며, 추가적으로 분석 수준과 옵션을 커스터마이징할 수 있는 플래그를 제공한다.

4. 자동화

Mythril은 CI/CD 파이프라인에 쉽게 통합될 수 있도록 설계되어 있으며, 명령 줄 도구로 자동 실행이 가능하다. 특히 GitHub Actions, Jenkins, GitLab CI 등과 연동하여 스마트 컨트랙트 배포 전에 보안 분석을 자동화할 수 있으며, Docker 이미지도 제공되어 배포 환경에서도 손쉽게 활용 가능하다. 분석 결과는 JSON, 텍스트, 또는 그래픽 형태로 출력되며, 실행 가능한 취약점 경로를 시각적으로 확인할 수 있는 call graph나 control flow graph도 함께 제공된다. Mythril은 MythX라는 상용 클라우드 서비스와도 연동되어 분석 자동화를 강화할 수 있다.

5. 장단점 및 실무 활용

Mythril은 현재 오픈소스 스마트컨트랙트 분석 도구 중 가장 널리 쓰이는 축에 든다. 개발 단계에서는 주로 개발자 개인이 수시로 실행해보고 이슈 수정에 쓴다. 감사 단계에선, 감사팀이 Slither와 Mythril을 함께 돌려 보고서를 작성한다. 중요 한건, Mythril 결과를 맹신하기보다, 수동검토와 함께 사용해야 한다는 것. Mythril이 “경고 없음”이어도, 코드가 완벽하단 뜻은 아니므로, 미검출 영역은 따로 살펴야 한다. 그래도 Mythril은 많은 common bug를 잡아내 팀의 부담을 줄여준다. 또한 이더스캔 등에서 Mythril을 돌려 확인한 결과를 게시하기도 하며, bug bounty

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

참여자들도 Mythril로 퍼징해 취약점 찾는다.

마. Slither

1. 개요

Slither는 Trail of Bits에서 개발한 정적 분석 기반 스마트 컨트랙트 보안 도구로, Solidity 언어로 작성된 스마트 컨트랙트의 구조 및 보안 취약점을 분석하는 데 최적화되어 있다. Python으로 구현되어 있으며, 속도와 정밀도 모두 우수하여 보안 분석은 물론 리팩토링 및 구조 개선 도구로도 자주 활용된다. Slither는 AST(Abstract Syntax Tree), SSA(Static Single Assignment) 기반의 코드 해석을 통해 함수 호출, 제어 흐름, 상태 변경 패턴 등을 분석하며, 다양한 내장 분석기를 통해 수십 가지 이상의 보안 이슈를 자동으로 탐지한다.

2. 내부 구조 및 기능

Slither의 내부 구조는 분석 모듈과 탐지기로 구성되어 있다. Solidity 소스코드를 심볼릭 실행을 하지 않고, Slither 자체 분석기에서 AST와 CFG로 변환한 뒤, 각 분석 모듈에서 함수 간 상호작용, 상태 변수 접근, 호출 체인 등을 추적한다. 이후 취약점 탐지기들은 이 정보를 기반으로 보안 이슈를 자동 탐색하며, 대표적으로 재진입 가능성, delegatecall 오용, 무한 루프, 접근 제어 문제, 변수 가시성 누락 등이 탐지된다. Slither는 분석된 결과를 구조적으로 정리하고, 의심되는 부분에 대한 경고 메시지와 코드 위치까지 함께 제공하여 수정 및 리팩토링을 용이하게 만든다.

3. 입력 방식

Slither는 주로 .sol 형식의 Solidity 소스코드를 직접 입력받아 분석을 수행한다. Truffle, Hardhat 등 기존 프로젝트 디렉터리를 그대로 활용할 수 있으며, 의존성 있는 컨트랙트도 함께 불러와 분석할 수 있다. 사용자는 명령줄에서 slither <file.sol> 형태로 간단히 실행할 수 있으며, 분석기 설정 파일이나 옵션 플래그를 활용해 분석 범위나 결과 출력 방식을 상세히 제어할 수 있다.

4. 자동화

Slither는 빠른 분석 속도와 CLI 기반의 유연한 실행 방식 덕분에 자동화 환경에서도 매우 유용하다. 분석을 위한 스크립트를 구성하여 CI/CD 파이프라인에 통합할 수 있으며, GitHub Actions, GitLab CI 환경에서 코드 푸시 또는 PR 이벤트 발생 시 자동으로 보안 점검을 수행하도록 설정할 수 있다. 결과는 JSON,

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

Markdown, Text 형식으로 출력 가능하며, 이를 기반으로 자동 리포트 생성도 지원한다. 특히 Slither는 Python API를 제공하므로, 사용자 맞춤형 분석 로직을 구현하거나 커스터마이징된 취약점 탐지기를 설계할 수 있다는 점에서 매우 확장성이 높다.

5. 장단점 및 실무 활용

보안 감사 기관에서는 Slither를 기본 툴로 사용한다. Slither를 실행하고, 나온 이슈 중 심각한 건 보고서에 포함하거나, 참고로 쓰며 수동검사로 확증한다. 개발 팀에서는 CI에 넣거나, 수시로 돌려 잠재 문제나 미사용 코드를 잡는다. Slither printers도 인기가 있는데, 큰 프로젝트에서 함수 호출 관계나 상속 구조를 자동 다이어그램으로 얻어 문서화에 쓴다. Slither API는 최적화에도 쓰여, gas 소비 큰 부분을 찾아준다.

바. 통합 전략

TLA+는 설계 단계 명세 검증에, KEVM은 구현 단계 바이트코드 검증에 쓰인다. Toolbox는 사용이 쉬운 대신 코드 레벨 구체성은 낮고, KEVM은 정확하나 다루기 어렵다. 전자는 프로토콜 수준 검증, 후자는 EVM 스펙 검증에 강한 특성이 있다. 또한, VeriSol은 개발자의 의도를 명시적으로 증명해주는 도구이고, Mythril/Slither는 알려진 나쁜 패턴을 잡아주는 도구다. VeriSol은 “X는 절대 일어나지 않아야함” 같은 요구를 직접 검증하지만, 미처 생각 못한 문제는 발견 못한다. Mythril/Slither는 일반적 취약점을 잘 찾지만, 특정 프로젝트 요구사항 만족 여부는 모른다. 이러한 차이로 인해 복합 활용 전략이 필요하다.

1. 설계 단계

시스템 아키텍쳐를 TLA+ Toolbox를 활용해 스마트 컨트랙트의 핵심 로직을 수학적으로 모델링한다. 이 과정을 통해 일반적인 코드 기반 구현 이전에 동시성 문제, 상태 간 경합(Race), 잘못된 상태 전이 등 논리적 결함을 사전에 탐지하고 제거할 수 있다.

보안 아키텍쳐는 TLA+ 명세와 주요 보안 자산에 대한 불변 조건(Invantics)을 정의한다. 예를 들어, “총 토큰 공급량은 절대 감소하지 않는다”, “권한이 없는 사용자는 인출 권한이 없다”와 같은 불변성 속성을 활용 한다.

2. 개발 단계

개발자는 Slither를 활용하여 커밋 단계에서 실시간으로 잘못된 코드 패턴, 무

스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발			
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

시된 반환 값, 잘못된 함수 가시성, 재사용 취약점 등을 탐지한다. 또한 Unit test 외에 Echidna 퍼징으로 설정된 속성을 기준으로 예상치 못한 입력 값이나 경계 상황에서도 스마트 컨트랙트가 정해진 규칙을 위반하지 않음을 검증한다.

3. 사전 감사 단계

팀 내 검증 담당자가 Mythril을 돌려 알려진 취약점을 추가로 점검 (재진입 등), 그리고 VeriSol 또는 Certora Prover와 같은 **명세 기반 모델 검증 도구 (Spec-driven Verification Tool)**를 활용해, 앞서 TLA+로 정의된 불변성이나 설계 규칙을 Solidity 코드에 직접 적용한다. 이때 KEVM을 활용할 수도 있는데, 주요 함수에 대한 사후조건을 KEVM reachability로 증명할 수 있다.

4. 외부 감사 단계

외부 감사자는 Slither/Mythril 결과를 참고하되, 새롭게 수동 분석도 하고, 필요한 경우 K, Coq 전문가들을 투입해 복잡한 부분을 추가 검증을 가질 수 있다. 기존 도구들이 다루지 못하는 신규 메커니즘은 TLA+로 Safety/Invariant 검토하거나, K 프레임워크 기반 KEVM을 활용해 저수준 명세의 실행가능성을 수학적으로 입증한다.

5. 배포 후 모니터링

Perseus 모니터링 툴과 같은 시스템을 활용하면, 컨트랙트의 런타임 이벤트 흐름을 추적하고, 실시간으로 사전 정의된 속성이 위반되었을 경우 즉시 알람을 받을 수 있다. 또한, 실제 사용자 트랜잭션을 기반으로 한 후속 퍼징 테스트 및 Regression Analysis를 통해, 이전에 없던 입력 조합이나 공격 시나리오에 대한 회귀 테스트를 수행해야 한다. 또한, 오라클로부터 수집한 외부 데이터를 바탕으로 상태 추적 및 예외 감지도 병행되며, 이 모든 정보는 후속 버전에 중요한 데이터로 작용된다.

6. IDE 통합 측면

현재 Remix나 VSCode Solidity plugin 등이 Slither, Mythril 연동을 지원하고 있다. 예를 들어, Solidity Visual Auditor은 코드 저장 시 Slither를 돌려 인라인 문제 표시해준다. CI에서는 GitHub Actions로 Mythril, Slither를 함께 돌리는 workflow가 존재한다. 정형 명세는 아직 개발 파이프라인에 완전히 녹아있지 않지만, 향후에는 Solidity에 사양을 주석으로 넣으면 (VeriSol 스타일) CI에서 솔버로 확인하는 식으로 발전할 수 있다.

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

참고문헌

- [1] Leslie Lamport, Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers, Addison-Wesley, 2002.
- [2] Daniel Jackson, Software Abstractions: Logic, Language, and Analysis, MIT Press, 2012.
- [3] Microsoft Research, “VeriSol: A Formal Verification Framework for Solidity Smart Contracts,” 2019. [Online]. Available: <https://www.microsoft.com/en-us/research/project/verisol/>
- [4] Certora Ltd., “Certora Prover: Formal Verification of Smart Contracts,” 2021. [Online]. Available: <https://www.certora.com/>
- [5] Runtime Verification, “Runtime Monitoring for Blockchain Smart Contracts,” [Online]. Available: <https://runtimeverification.com/>
- [6] M. Bhargavan, C. Fournet, et al., “Formal Verification of Smart Contracts: Short Paper,” Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security (PLAS), pp. 91–96, 2016.
- [7] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, “Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts,” IEEE Symposium on Security and Privacy (SP), 2016.
- [8] OpenZeppelin, “OpenZeppelin Contracts – Secure Smart Contract Library,” [Online]. Available: <https://openzeppelin.com/contracts/>
- [9] CertiK, “Formally Verifying OpenZeppelin’s ERC-20 Implementation,” 2022. [Online]. Available: <https://www.certik.com/resources/blog/7EELzmUpEOE7yhow8LpA3A-formally-verifying-openzeppelins-erc-20-implementation>
- [10] SushiSwap & Certora, “Exploiting an Invariant Break: How We Found a Pool Draining Bug in SushiSwap’s Trident,” 2022. [Online]. Available: <https://medium.com/certora/exploiting-an-invariant-break-how-we-found-a-pool-draining-bug-in-sushiswaps-trident-585bd98a4d4f>
- [11] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” 2008. [Online]. Available: <https://bitcoin.org/bitcoin.pdf>
- [12] R. E. Bryant, “Graph-Based Algorithms for Boolean Function Manipulation,” IEEE

	스마트 컨트랙트의 개발-배포-실행의 전주기적 취약점 및 신뢰성 오류 개선 기술개발		
소속	고려대학교	연구 책임자	인호
센터	블록체인 연구센터		
제목	스마트 컨트랙트를 위한 정형기법		

Transactions on Computers, vol. C-35, no. 8, 1986.

- [13] MythX, “Security Analysis for Smart Contracts,” [Online]. Available: <https://mythx.io/>
- [14] G. Wood, “Ethereum: A Secure Decentralised Generalised Transaction Ledger,” Yellow Paper, 2014. [Online]. Available: <https://ethereum.github.io/yellowpaper/paper.pdf>
- [15] Yoon, Seongho, and Jin-Young Choi. “Formal Specification and Verification of Smart Contract-Based Loan Management System Using TLA+.” IEEE Access (2025).