반응형 시스템을 위한 LLM 기반 상태머신 생성 기법 및 성능평가

최승빈[1], 김요엘[2], 최윤자[2†]
경북대학교 소프트웨어재난연구센터[1], 경북대학교 컴퓨터학부[2]
csb8226@naver.com, kimyoel2305@gmail.com, yuchoi76@knu.ac.kr

LLM-Based State Machine Generation Technique for Reactive Systems and Its Performance Evaluation

Seungbin Choi[1], Yoel Kim[2], Yunja Choi[2†]
Software Disaster Research Center, Kyungpook National University[1]
School of Computer Science & Engr., Kyungpook National University[2]

요 약

본 연구에서는 C 코드를 입력받아 LLM을 통해 상태머신을 자동 생성하는 기법(LLM_State_Generate)을 제안하고, 생성된 상태머신을 CBMC 형식 검증을 통해 확인한 뒤 능동학습 기법과 성능을 비교 평가하였다. Simulink 예제 45개 프로그램을 대상으로 실험한 결과, 제안한 방식인 LLM_State_Generate는 단일 생성 기준 프로그램 단위 46.7%의 성공률을 보인 반면 반복적 정제를 포함한 기존의 정형적 능동학습을 통한 상태머신 생성 기법은 93.3%의 성공률을 달성했다. 본 연구는 LLM 기반 접근법의 한계와 개선 방향을 제시한다.

1. 서 론

상태머신(State Machine)은 시스템의 상태와 전이를 정의하는 수학적 모델로, 높은 신뢰성이 요구되는 임베디드 시스템, 반응형 시스템, 제어 시스템 개발에 필수적으로 사용되나, 이러한 상태머신을 수작업으로 작성하는 것은 어렵기 때문에 상태머신 없이 개발되는 경우가 많다. 그래서 개발된 소스코드로부터 상태머신을 학습하는 연구가 진행되었다[1, 2].

상태머신을 정형적 능동학습(Active Learning)을 통해 생성하는 기법[1]은 형식적 방법을 기반으로 하여 높은 정확성을 보장한다. 그러나 능동학습은 엄밀한 모델 정제 과정으로 인하여 상태머신을 생성하는 데 많은 비용이 소요된다. 반면, LLM을 활용한 자연어 기반 접근은 능동학습 기법 대비 빠르게 모델을 생성할 수 있다. 최근 LLM은 테스트 코드 생성, 스펙 생성, 자동 오류 수정 등 다양한 곳에서 활용되고 있다[3,4]. 그러나 상태머신 자동 생성에 적용된 사례는 없으며 본 논문이 최초 시도이다. 다만 LLM 기반 접근은 상태 전이를 누락하거나 잘못된 조건을 만들어내는 등 논리적 정확성과 안정성 측면에서 한계가 존재할 수 있다.

이에 본 연구에서는 C 코드와 프롬프트를 LLM에게 입력해 상태머신을 자동 생성하는 기법(LLM_State_Generate)을 제안한다.

2. LLM 기반 상태머신 생성

2.1. 기법 개요 및 흐름

본 연구에서 제안하는 기법은 C 소스 코드로부터 상태머신의 자동 생성이다. 핵심은 LLM의 패턴 인식을 활용해 입력 대상인 C 코드와 관심 변수를 제공해 상태머신으로 변환하는 것이다.

상태머신 시각화 예시는 그림 1과 같다.

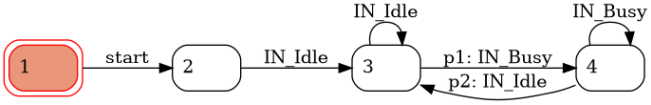

그림 1. 상태머신 예시

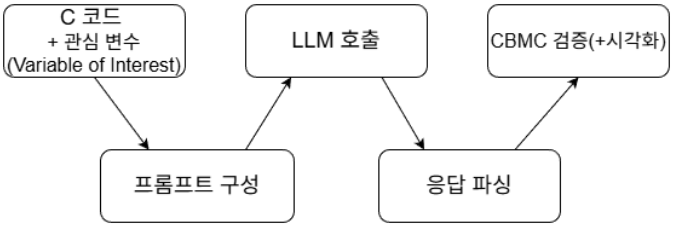

그림 2. LLM_State_Generate의 전체 기법 흐름

본 기법은 먼저 그림 2와 같이 C 코드와 관심 변수를 입력으로 받아 사전 구성한 프롬프트와 함께 LLM에 전달해 상태머신 전이 리스트의 생성을 요청한다. LLM의 응답을 받은 후에는 LLM이 생성한 상태머신의 전이 리스트의 형식 검증 후 파싱하며, 파싱한 전이 리스트를 이미지로 변환하여 사용자가 상태머신 생성이 잘 이루어졌는지 육안으로 확인하기 용이하도록 한다. 마지막으로 생성한 전이 리스트를 CBMC(C Bounded Model Checker)[5]로 모델 검증을 실시하여 상태머신이 C 코드를 올바르게 반영하고 있는지 확인한다.

2.2. LLM 기반 생성 방법

본 연구의 실험 데이터셋인 C 코드들은 변수형과 변수 선언부, 함수선언, 반응형 시스템의 주요 제어논리 등으로 구성되어 있다.

본 기법에서 사용된 프롬프트의 핵심 내용은 다음과 같다.

1. 상태머신 생성 작업을 한다고 프롬프트 도입부에 명시한다.
2. 생성 요구사항(연속적인 상태 ID 사용 규정, 초기 상태는 1로 고정, 고립된 상태 없음)과 전이 형식 규칙을 알려준다.
3. 전이 형식은 [현재상태, '가드: 동작', 다음상태]로 정의되며 조건식은 동일 상태에서 다중 전이가 있을 때만 포함하여 조건식 오류를 최소화하며 상태머신의 일관성을 보장할 수 있도록 한다.

해당 프롬프트로 생성되는 그림 1 예시에 대한 상태머신 전이 리스트는 다음과 같다: "[[1, 'start', 2], [2, 'IN_Idle', 3], [3, 'IN_Idle', 3], [3, '![InputTask == 0]: IN_Busy', 4], [4, 'IN_Busy', 4], [4, '![TaskTime > Counter]: IN_Idle', 3]]"

LLM 모델은 GPT-5를 사용했으며, GPT-5는 API 응답 설정이 Reasoning Effort와 Verbosity로 구성되며, 각각 기본값인 medium으로 설정하여 LLM의 기본 성능을 확인하고자 하며, 각 케이스 별 수행은 1회 수행한다.

2.3. 모델 생성 성공여부 평가 방법

모델 생성 성공 여부는 CBMC[5]를 사용하여 확인한다. CBMC는 C 프로그램의 형식 검증 도구로, 프로그램이 특정 명세(assertion)를 만족하는지 자동으로 확인한다. 일반 테스트가 몇 가지 입력만 검증하는 반면, 형식 검증은 모든 실행 가능한 경로를 수학적으로 분석한다.

검증 방식은 다음과 같다. LLM이 생성한 상태머신 전이 리스트 $[[s, 'g: a', s'], ...]$를 기반으로 assertion으로 변환하여(예: $assert(!(state == s \&\& g) || (next\_state == s'))$) 원본 C 코드에 삽입한다. CBMC를 이용하여 C 코드가 전이 리스트 기반으로 생성된 assertion 들을 모두 만족하는지 검증하며, 모든 assertion들이 만족되면 생성된 상태머신이 원본 코드와 동등한 상태 전이 구조를 가졌다고 볼 수 있다.

3. 실험 설계

3.1. 데이터셋

본 연구에서는 LLM 상태머신 생성 기법의 성능을 객관적인 확인을 위해 Simulink C 코드 변환 예제[1]를 실험 대상으로 선정하였다. 해당 데이터셋은 단순한 구조부터 복잡한 상태 전이까지 폭넓은 복잡도 분포가 존재하며, 자동차 제어, 산업 로봇 등 실제 산업에서 사용되는 제어 로직을 포함하므로 성능 평가에 적합한 표준 벤치마크이다[1].

3.2. 평가 방법

3.2.1. 성공률 평가 기준

LLM_State_Generate와 능동학습 기반 생성 기법의 성능 비교를 위해 두 기법의 상태머신 생성 성공률을 두 가지 기준으로 평가하였다.

첫 번째는 예제 단위 성공으로 45개 예제 프로그램 각각에 대해 예제별로 생성된 assertion이 전부 통과하면 성공으로 분류한다.

두 번째는 전체 실험에서 생성된 모든 assertion 중 검증을 통과한 비율을 측정한다. 케이스 단위로는 실패했더라도 부분적으로 정확한 전이가 있을 수 있으므로, 기법의 잠재적 성능을 평가하는 지표가 된다.

4. 실험 결과

4.1. 전체 성능 비교 (RQ1)

본 연구에서는 45개 예제에 대해 두 기법의 상태머신 생성 성공률을 측정하였다(표 1). 실험 결과, 능동학습 기반 생성 기법은 예제 단위 93.3%, 전체 assertion은 97.5% 성공률을 기록하여 형식적 기법 기반 도구의 높은 신뢰성을 입증하였다. 반면 LLM_State_Generate는 예제 단위에서 46.7%의 성공률을 보여 단순 자연어 기반 접근만으로는 구조적 정확성을 보장하기 어려운 것으로 보이나 전체 assertion에 대한 CBMC 검증 성공률을 보면 LLM의 성공률이 78.1%까지 올라가는 것으로 나타났다. 해당 결과를 통해 LLM 생성 성능이 결코 낮지 않다는 점을 알 수 있다. 그리고 능동학습 기반 생성 방법은 시간 제한(Time-out)을 1시간으로 잡고 진행되었으나, LLM_State_Generate는 전체 예제 중 가장 오래 걸린 케이스가 200초를 넘지 않았다. 이를 통해 생성 속도가 능동학습 기법 대비 상당히 빠르다는 것을 확인할 수 있었다.

표 1. 기법별 성공률 비교

기법	예제 단위	전체 assertion
능동학습 기반 [1]	42/45(93.3%)	384/394(97.5%)
LLM_State_Generate	21/45(46.7%)	349/447(78.1%)

5. 논의 및 결론

본 연구에서는 LLM 기반 상태머신 생성 기법의 성능을 능동학습 기반 생성 기법과 비교 평가하고, 복잡도가 성능에 미치는 영향을 정량적으로 분석하였다. 실험 결과, LLM은 프로그램 단위에서 46.7%의 성공률을 기록하여 93.3%의 성공률을 보인 능동학습 기반 생성 기법에 비해 성능 격차가 확인되었으며, 특히 상태 수와 전이 수가 증가할수록 실패율이 뚜렷하게 증가함을 확인하였다. 다만, assertion 단위로 평가했을 때는 LLM은 성공률 78.1%, 능동학습은 97.5%로 성능 격차가 상대적으로 완화되어 단일 생성 방식으로도 일정 수준의 검증 조건을 추출할 수 있음을 보여준다.

본 연구는 제한된 데이터셋을 사용했으나 단일 생성만으로 예제 단위 46.7%(assertion 단위 78.1%)의 성공률을 달성하였으며 LLM 기반 코드 생성의 정량적 한계를 규명했다는 점에서 의의가 있다. 향후 연구를 통하여, LLM 기반 반복 정제의 효율성과 정확도 향상 정도를 고찰하고, LLM 기반 상태머신 생성이 기존 정형적 방법의 실용적 대안으로 활용될 수 있을지 확인할 계획이다.

Acknowledgement

이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(RS-2021-NR060080).

참고문헌

[1] Jeppu, N. Y., Melham, T., & Kroening, D., "Enhancing active model learning with equivalence checking using simulation relations," Formal Methods in System Design, vol. 61, no. 2–3, pp. 164–197, 2022.

[2] A. Grosche, B. Igel, and O. Spinczyk, "Transformation-and Pattern-based State Machine Mining from Embedded C Code," in Proceedings of the 15th International Conference on Software Technologies (ICSOFT), 2020, pp. 43-54.

[3] Le-Cong, T., Manh, D. T., Inoue, S., and Ghosh, S., "SpecGen: Automated Generation of Formal Program Specifications Assisted by Large Language Models," in Proceedings of the 47th International Conference on Software Engineering (ICSE 2025), 2025.

[4] Fan, Z., Gao, X., Mirchev, M., Marinescu, A., and Tipson, Y., "Automated Repair of Programs from Large Language Models," in Proceedings of the 45th International Conference on Software Engineering (ICSE 2023), pp. 947–958, 2023.

[5] Clarke, E., Kroening, D., and Lerda, F., "A tool for checking ANSI-C programs," in Tools and Algorithms for the Construction and Analysis of Systems (TACAS), LNCS 2988, pp. 168-176, Springer, 2004.