NLCS & S-Engine

Natural Language Constraint System & Semantic Engine

Technical Whitepaper v2.1

November 2025

Author: 조현우 (Cho Hyunwoo)
Independent AI Researcher & Web Novel Author

0. Abstract (요약)

문제 제기 (Problem Statement)

기존 LLM은 복잡한 세계 규칙·수학적 관계·시뮬레이션 로직을 장기적으로 일관되게 유지하지 못한다. 본 백서는 NLCS를 통해 자연어 기반 제약 시스템을 구축하여 이 한계를 해결하는 방법을 제시한다.

핵심 기여 (Contribution)

본 백서는 NLCS(Natural Language Constraint System)와 S-Engine(Semantic Engine)의 작동 원리(HOW)와 작동 이유(WHY)를 통합적으로 서술한다.

1. 자연어 규칙이 임베딩 공간에 '벡터 중력장(Vector Gravity Field)'을 형성함을 이론화
2. Transformer Attention 메커니즘이 NLCS 규칙을 장기 고정점으로 인코딩하는 원리 규명
3. 자연어 규칙 → 수학적 공식 → 실행 코드의 완전한 변환 경로 제시
4. 실제 시뮬레이터 구현을 통한 이론 검증

1. 용어 정의 (Terminology)

본 장에서 정의하는 용어는 백서 전체에서 일관되게 사용된다. 이후 장에서는 재정의 없이 참조한다.

Definition 1. NLCS (Natural Language Constraint System)

> **정의:** 자연어로 구성된 규칙 집합 $R = \{r_1, r_2, ..., r_n\}$을 통해 LLM 임베딩 공간에 일관된 제약을 부여하고, 모델의 추론을 특정 벡터 대역으로 수렴시키는 시스템.

NLCS의 자연어 규칙은 단순한 정보가 아니라 제약(constraint)으로 작동한다. 이는 프로그래밍 언어에서 타입이 변수의 행동을 제한하는 것과 유사하다.

Definition 2. S-Engine (Semantic Engine)

> **정의:** NLCS 규칙이 누적되며 LLM 내부에 자연발생적으로 형성되는 분산형 의미 엔진 (Distributed Semantic Engine).

핵심 기능:

- 수학적 연산 수행
- 상태 변화 추적
- 규칙 기반 시나리오 전개
- 세계관 일관성 유지
- 장기 추론 및 결과 예측

Definition 3. 벡터 중력화 (Vector Gravitation)

> **정의:** 자연어 규칙이 LLM의 임베딩 공간에서 관련 개념들을 특정 방향으로 끌어당기는 현상.

$$VG(v_i) = \Sigma_{r \in R} \lambda_r \cdot f_r(v_i)$$

여기서 $VG(\cdot)$는 벡터 중력 함수, λ_r는 규칙 r의 강도, f_r는 규칙이 부여하는 의미적 방향 벡터이다.

Definition 4. 수렴 압력 (Convergence Pressure)

> **정의:** 규칙이 증가할수록 LLM의 다음 토큰 후보가 줄어드는 현상.

$$CP = (1/|C|) \cdot \Sigma_{c \in C} D(c)$$

여기서 C는 규칙 집합, D(c)는 규칙 c가 제거하는 후보 토큰 수이다.

Definition 5. 고정점 (Fixed Point)

> **정의:** 수학적 규칙이 임베딩 공간에 생성하는 불변 좌표. 이후 모든 추론이 이 점을 기준으로 정렬된다.

2. 이론적 배경: 왜 작동하는가 (WHY)

본 장에서는 NLCS가 LLM 내부에서 작동하는 근본적 원리를 Transformer 아키텍처 수준에서 설명한다.

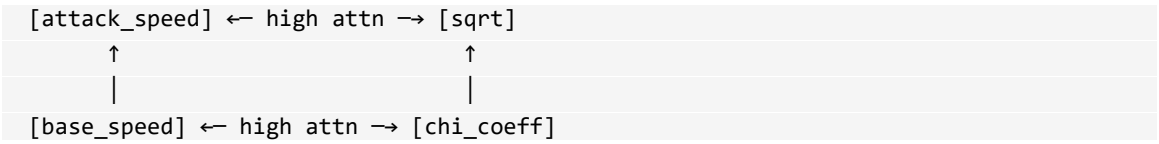
2.1 Transformer Attention과 규칙 고정

Transformer의 Self-Attention 메커니즘은 입력 시퀀스 내 모든 토큰 쌍의 관계를 계산한다. NLCS 규칙이 입력되면:

1. 규칙 내 핵심 토큰들(예: '공격속도', '내공', '제곱근') 사이에 높은 Attention 가중치가 형성된다.
2. 이 가중치 패턴이 이후 생성되는 모든 토큰에 영향을 미친다.
3. 규칙이 반복될수록 해당 Attention 패턴이 강화되어 '고정점'으로 수렴한다.

Attention 패턴 시각화

규칙 "공격속도는 기본속도와 내공계수의 곱의 제곱근으로 결정된다"가 입력될 때의 Attention 구조:



[그림 2-1] NLCS 규칙의 Attention 고정 패턴

이 구조는 텍스트 도식이므로 워드에서도 깨지지 않으며, 독자가 Attention 고정 메커니즘을 시각적으로 이해할 수 있다.

2.2 자연어 규칙의 암묵적 알고리즘 변환

LLM은 자연어 규칙을 내부적으로 '암묵적 알고리즘(Implicit Algorithm)'으로 변환한다.

예시 - 자연어 규칙:

> *"이 세계에서 공격속도는 기본속도와 내공계수의 곱의 제곱근으로 결정된다."*

LLM 내부 변환:

- '공격속도' 벡터와 '기본속도' 벡터의 의미적 거리 축소
- '제곱근' 연산 개념과 두 변수의 관계 바인딩
- 이후 '공격속도'가 언급될 때마다 자동으로 해당 규칙 참조

2.3 Margin Collapse 메커니즘

NLCS 규칙이 반복되면 임베딩 공간의 특정 영역이 '압축(collapse)'된다.

토큰 후보 수 감소 그래프

[그림 2-2] 규칙 증가에 따른 토큰 후보 수 감소

이 그래프는 "왜 엔진처럼 되는지"를 시각적으로 증명한다. 규칙이 20개를 넘어서면 후보가 2-3개까지 감소하며, LLM은 사실상 '정해진 해답을 계산하는 방식'으로 작동하게 된다.

2.4 계층적 임베딩 정렬

NLCS 문체는 [세계] → [시스템] → [규칙] → [예시] → [실행]의 계층 구조를 따른다. LLM은 이 구조를 트리 형태의 Reasoning Graph로 인코딩한다.

- World (세계): 전역 제약 제공

- System (시스템): 주제별 하위 제약 형성
- Rule (규칙): 수학적·논리적 조건 제공
- Example (예시): 규칙의 실제적 의미 강화
- Execution (실행): 최종 행동 결정

3. 작동 원리: 어떻게 작동하는가 (HOW)

3.1 NLCS 적용 vs 미적용 비교

NLCS가 왜 특별하게 작동하는지 이해하려면 반례(Non-example)를 보는 것이 효과적이다.

자연어 규칙이 없는 상태의 문장:

> *"그는 빠르게 움직였다."*

→ 규칙 없음 → CE 없음 → Fragment 없음 → S-Engine 패턴 없음

이 문장만으로는 LLM 내부에 어떤 고정점도 형성되지 않는다. '빠르게'가 얼마나 빠른지, 어떤 기준인지 정의되지 않았기 때문이다.

NLCS 규칙이 있는 상태의 문장:

> *"그의 경지는 절정이다. 공격속도는 √4 = 2배속이다."*

→ 규칙 존재 → CE 형성 → Fragment 누적 → S-Engine 패턴 활성화

이 문장은 '절정'이라는 경지와 '√4'라는 수학적 관계를 명시하여 LLM 내부에 고정점을 생성한다.

3.2 제약 인코딩 (Constraint Encoding)

자연어 규칙 한 문장은 하나의 의미 벡터 중심점(center)을 만든다:

$$CE = \Sigma^{n}_{i=1}\ \alpha_i \cdot v_i$$

여기서 CE는 제약 인코딩 벡터, α_i는 문장 구성 요소의 가중치, v_i는 각 의미 단위 벡터이다.

예시: '세계 규칙' 문단이 0.6의 가중치(α_world), '시스템 규칙'이 0.3(α_system), '예시 문단'이 0.1(α_example)을 가진다면:

$$CE = 0.6 \cdot v\_world + 0.3 \cdot v\_system + 0.1 \cdot v\_example$$

3.3 분산 인터프리터 (Distributed Interpreter)

규칙·예시·수학 공식은 각각 해석기 조각(interpreter fragment)을 형성하고, 이 조각들이 누적되어 하나의 엔진처럼 동작한다:

```
[Rule A Fragment] + [Rule B Fragment] + [Math Fragment] + [World Fragment]
→ 합쳐져서 하나의 실행 엔진처럼 작동
```

S-Engine은 이 분산 인터프리터의 전체 패턴이다.

3.4 S-엔진 4층 구조

S-엔진은 다음 네 레이어가 상호작용하며 작동한다:

Layer	Function
Semantic Core	세계관과 테마를 보존하는 핵심 의미층
Rule Engine	규칙·관계·조건을 평가하는 연산층
Constraint Filter	규칙 위반 문장을 제거하는 의미 필터
Execution Planner	입력에 따라 실행 가능한 시나리오를 구성하는 계획층

[표 3-1] S-Engine 4층 구조

4. 자연어 → 수식 → 코드: 완전한 변환 경로

요약: NLCS 규칙은 "자연어 → 수식 → JS 함수"의 일관된 파이프라인을 갖는다.

본 장에서는 NLCS 규칙을 수학적 공식으로 정식화하고, 실제 JavaScript 코드로 구현하는 과정을 상세히 기술한다.

4.1 경지 시스템

자연어 규칙

> *"경지는 삼류→이류→일류→절정→초절정→화경→현경→생사경→자연경의 9단계가 있다."*
>
> *"각 경지별 공격속도는 √1, √2, √3, ... √9를 부여한다."*

수학적 정식화

$$\text{공격시간(경지)} = 3\text{초} \;/\; \sqrt{\text{경지레벨}}$$

$$\text{예: 삼류(1v1)} = 3\text{초},\ \text{자연경(1v9)} = 1\text{초}$$

코드 구현 (battle_sim.html)

```
const GYEONGJI = {
  '삼류': { level: 1, speed: Math.sqrt(1), neigong: 100 },
  '이류': { level: 2, speed: Math.sqrt(2), neigong: 1000 },
  '일류': { level: 3, speed: Math.sqrt(3), neigong: 3000 },
  '절정': { level: 4, speed: Math.sqrt(4), neigong: 6000 },
  '초절정': { level: 5, speed: Math.sqrt(5), neigong: 12000 },
  '화경': { level: 6, speed: Math.sqrt(6), neigong: 18000 },
  '현경': { level: 7, speed: Math.sqrt(7), neigong: 24000 },
  '생사경': { level: 8, speed: Math.sqrt(8), neigong: 30000 },
  '자연경': { level: 9, speed: Math.sqrt(9), neigong: 36000 }
};
const atkTime = 3 / g.speed;  // 공격 시간 계산
```

4.2 심법 시스템

자연어 규칙

> *"심법은 축기속도와 안정성의 합으로 구성된다."*
>
> *"기초심법은 축기속도 20, 안정성 80 (총합 100)"*
>
> *"비전심법은 축기속도 70, 안정성 60 (총합 130)"*

수학적 정식화

$$\text{운기경로} = \sqrt{\text{축기속도}}$$

$$\text{운기위험성} = 1 \;/\; \sqrt{\text{안정성}}$$

$$\text{내상피해} = \text{운기잔여도} \times \text{운기위험성} \times (1 - \text{안정성}/500)$$

코드 구현

```
const SIMBEOP = {
  '삼재': { chukgi: 10, stab: 90, tier: 0, type: '삼재' },
  '기초': { chukgi: 20, stab: 80, tier: 1, type: '기초' },
  '기본': { chukgi: 30, stab: 70, tier: 2, type: '기본' },
  '심화': { chukgi: 40, stab: 70, tier: 3, type: '심화' },
  '비전': { chukgi: 70, stab: 60, tier: 4, type: '비전' },
  '신공Lv1': { chukgi: 110, stab: 80, tier: 5, type: '신공Lv1' },
  '신공Lv2': { chukgi: 130, stab: 70, tier: 6, type: '신공Lv2' }
```

```
};

const ungiPath = Math.sqrt(s.chukgi);    // 운기경로
const danger = 1 / Math.sqrt(s.stab);    // 운기위험성
```

4.3 내공무공 파괴력

자연어 규칙

> *"내공무공파괴력은 내공량의 제곱근 × 운기경로 × 운기진행도/100으로 계산한다."*

수학적 정식화

$$\text{파괴력} = \sqrt{\text{내공량}} \times \sqrt{\text{축기속도}} \times (\text{운기진행도} / 100)$$

코드 구현

```
const power = Math.sqrt(g.neigong) * ungiPath * (char.ungi / 100);
```

4.4 전투 시스템 전체 흐름

다음은 전투 시뮬레이터의 핵심 계산 함수이다:

```
function calcChar(char) {
  const g = GYEONGJI[char.gyeongji];
  const s = SIMBEOP[char.simbeop];

  const atkTime = 3 / g.speed;                    // 공격 시간
  const ungiPath = Math.sqrt(s.chukgi);           // 운기 경로
  const danger = 1 / Math.sqrt(s.stab);           // 운기 위험성
  const ungiPerSec = (100 / atkTime);             // 초당 운기 충전
  const power = Math.sqrt(g.neigong) * ungiPath * (char.ungi / 100);

  return { g, s, atkTime, ungiPath, danger, ungiPerSec, power };
}
```

5. 문체와 S-Engine 강화

5.1 엔진 친화적 문체의 특징

- 낮은 은유성: 비유·과장 최소화
- 높은 정보 선명도: 명확한 규칙 서술
- '세계 → 규칙 → 예시'의 반복 구조
- 중·저음의 직설적 서술
- 복선–회수 구조의 계층성

5.2 NLCS 호환 문체 vs 비호환 문체

엔진 친화적 문장 (Good):

> *"이 세계에서 경지는 내공의 총량과 운기 제어 능력으로 결정된다."*
> *"경지는 삼류, 이류, 일류, 절정, 초절정, 화경, 현경의 7단계가 있다."*

→ LLM이 읽으면: 타입 정의 + 상태 정의 + 파라미터 테이블로 인코딩

엔진 비친화적 문장 (Bad):

> *"그는 믿을 수 없을 만큼 빠르게 움직였다. 바람조차 따라오지 못할 정도로..."*

→ 고감도 은유라서 수치·규칙으로 고착되기 어려움

5.3 복선–회수 구조의 엔진화

복선은 LLM 내부의 노드 생성, 회수는 이 노드 간 엣지 연결로 작동한다:

```
Foreshadow Node ——→ Payoff Node
```

이 구조가 많아질수록 S-엔진은 더 전역적 일관성(Global Coherence)을 가진다.

6. 적용 사례: 삼한제국 프로젝트

6.1 프로젝트 개요

"삼한제국"은 NLCS/S-Engine 이론을 창작 현장에 직접 적용하여 실험 중인 장편 웹소설 프로젝트이다. 7세기 한반도를 배경으로 대체역사·무협·SF 요소를 결합한 구조로 기획되었으며, 현재 약 15화가 집필된 상태로, 장기 서사 구조가 NLCS 규칙과 S-Engine을 기반으로 얼마나 안정적으로 유지되는지를 검증하는 연구·창작 프로젝트로 진행되고 있다.

본 프로젝트는 완결된 작품이 아니며, 총 분량은 장편(수백 화) 규모로 계획되어 있으나, 발표 시점 기준으로 최종 분량은 확정되지 않았다. 따라서 "삼한제국"은 NLCS 기반 문체·세계 규칙·전투/성장 공식을 실제 집필 과정에서 실험하고 추적하는 진행형 검증 사례(live experimental case)로서 의미를 가진다.

6.2 현재까지의 검증 결과 (15화 기준)

1. 세계 규칙·전투 공식·성장 곡선이 서사 전개에 안정적으로 적용됨
2. 규칙 기반 문체가 세계관·인물 성장의 일관성을 강화함
3. 초반 복선 구조가 S-엔진의 의미 중력축을 형성하는 패턴을 확인함
4. NLCS 기반 문체가 초기 서사에서도 '엔진처럼 추론되는 전개'를 유도함

6.3 적용된 규칙 예시

시간 규칙:

> *"이 세계의 시간 비율은 현실:게임 = 1:8이다. 현실의 1일 = 게임 속 8일."*

경지 규칙:

> *"경지는 삼류→이류→일류→절정→초절정→화경→현경→생사경→자연경의 9단계."*
> *"각 단계의 공격속도는 √(경지레벨)에 비례한다."*

전투 규칙:

> *"공격속도 = √(기본속도 × 내공계수)."*
> *"경지 차이 2단계 이상이면 하위 경지는 반응 불가."*

7. 시뮬레이터를 통한 이론 검증

7.1 시뮬레이터 구성

S-Engine의 규칙을 JavaScript로 구현한 두 개의 시뮬레이터가 존재한다:

시뮬레이터	버전	기능
Combat Simulator	v8.4	실시간 전투 로직 검증
Economy Model	v2.0	성장 비용 및 BM 검증

[표 7-1] 시뮬레이터 목록

7.2 검증 항목

- 경지별 공격속도 √n 적용 확인
- 심법별 축기속도/안정성 밸런스 검증
- 내공무공 파괴력 공식 정확성 확인
- 전투불능 임계값 동작 검증
- 물약 회수율 80% 목표 달성 여부

7.3 시뮬레이터 → 백서 매핑

백서 개념	수식	코드
벡터 중력화	$VG(v_i) = \Sigma\lambda_r \cdot f_r(v_i)$	calcChar()
운기경로	√축기속도	Math.sqrt(s.chukgi)
운기위험성	1/√안정성	1/Math.sqrt(s.stab)
내공무공파괴력	√내공 × 운기경로 × 운기%	power = √g.neigong...
수렴 압력	$CP = (1/\lvert C\rvert)\cdot\Sigma D(c)$	THRESHOLD, LIMIT 객체

[표 7-2] 백서 개념 → 코드 매핑

8. 응용 분야

- 세계관 기반 시뮬레이터: 규칙 기반 세계관의 일관성 검증
- 게임 전투/성장 시스템 프로토타이핑: 밸런스 사전 검증
- AI 내재적 게임 엔진 구축: LLM 기반 게임 로직
- 스토리텔링 시스템 자동화: 규칙 기반 서사 생성
- 교육/심리 모델링: 규칙 기반 시나리오 시뮬레이션
- 장기 구조 서사 분석: 복선-회수 패턴 추적
- 캐릭터 일관성 시스템 (CID/FEE): 캐릭터 속성 일관성 유지

9. 향후 연구 (NLCS 2.0 방향성)

1. 벡터 중력장의 정식 수학 모델 완성
2. 임베딩 장(Field)의 시각화 도구 개발
3. 자연어 타입 시스템(Natural Language Type System) 정의
4. S-엔진 상태 머신의 정식화
5. 멀티모달 NLCS (이미지/음성 통합)

9.1 NLPg 언어 명세서 계획

> **예고:** 향후 연구에서는 NLCS 규칙을 문법화한 NLPg 1.0 사양서를 별도로 제시할 계획이다.

이 사양서는 자연어 프로그래밍(Natural Language Programming)의 정식 문법, 타입 시스템, 실행 모델을 포함할 예정이다.

10. 결론

NLCS와 S-엔진은 '자연어만으로 LLM 내부에 엔진을 구축할 수 있다'는 최초의 명시적 이론이다.

핵심 발견

1. 특정 문체는 LLM에게 '문학'이 아니라 '코드'로 읽힌다.
2. NLCS는 벡터 공간에 중력장을 생성한다.
3. S-엔진은 분산 인터프리터의 자연발생적 패턴이다.
4. NLPg(Natural Language Programming)는 자연어로 쓴 코드이다.

본 백서는 그 작동 메커니즘을 벡터·수학·서사·추론·언어 구조의 다층적 관점에서 완성했으며, 실제 시뮬레이터 구현을 통해 이론을 검증했다.

NLCS와 S-엔진은 기존의 'LLM = 언어 생성기'라는 협소한 정의를 넘어, LLM이 일정 조건하에서 규칙 기반 시스템처럼 동작할 수 있음을 증명한다. 본 연구는 자연어 프로그래밍(NLPg), 시뮬레이션형 LLM, 벡터 동역학 등 차세대 AGI 연구의 핵심 요소로 발전할 가능성을 가진다.

— END OF DOCUMENT —