インタラクティブフィクションにおける LLM とニューロシンボリック手法を用いた到達可能性解析の検証

中島光人 [1]　上田亮 [1]　宮尾祐介 [1,2]
[1] 東京大学　[2] 国立情報学研究所大規模言語モデル研究開発センター
{rn86222-light0213, ryoryoueda, yusuke}@is.s.u-tokyo.ac.jp

概要

本研究では、インタラクティブフィクション（IF）におけるデッドエンド（DE）の検出に取り組んだ。IF の質を担保するには、プレイヤーの様々な入力に対応できるようにコマンドやパーサを定義する必要があり、コストがかかる。一方、想定しないコマンドをプレイヤーが入力した際の挙動を LLM に制御させようとする動きもみられるが、DE の検出が困難になってしまう。そこで、LLM に新たな行動を生成させるのではなく、固定されたスキーマに基づくモデルを初めにゲーム仕様から生成させ、シンボリックプランナーを組み合わせることで、そのゲーム世界において妥当な行動の網羅的実行可能性と DE の検出とを同時に達成することを試みる。

1　はじめに

インタラクティブフィクション（IF）は「コンピュータによりシミュレートされた世界に対し、プレイヤーがテキストを入力することで干渉し、その結果がテキストとして返される形式のゲーム」と定義される [1]。本稿では事前に何らかの明確なゴール条件が設定されている IF を扱うことにする。IF は実質的にその内部で状態遷移機械を持ち、プレイヤーの入力に基づいて状態が遷移することとなるが、Zork [2] に代表される古典的 IF では、十分なプレイ体験を提供するため、事前に多数のコマンドや工夫されたパーサを定義する必要があった。

近年では、大規模言語モデル（LLM）を用いて、想定外の入力に対する行動を動的に生成する手法が提案されている [3, 4, 5, 6]。しかしこの場合、生成された行動によってプレイヤーが二度とゴールに到達できないデッドエンド（DE）に陥る可能性があり、かつそれを検知することが困難である。LLM による行動生成は状態や遷移を動的に追加するため、世界観への適合性を保ったまま到達可能性を解析することは自明ではない。

我々は LLM による行動生成を用いた IF における DE 検出を主眼としたシステムを提案する。提案手法では、IF に共通する抽象的な操作やオブジェクトの属性を表現するためのゲーム非依存のスキーマを定義する。そして LLM に自然言語ベースのゲーム仕様を与えることで、ゲームを制御するためのモデルをそのスキーマに基づいて生成させる。また、到達可能性の判定はそのモデル上で動くシンボリックプランナーに委ねる。本設計により、(1) 開発者はゲームの説明や目標を与えるだけで IF エンジンを構築でき、(2) LLM による行動生成をモデル上の操作に制限することで、世界観を逸脱しない柔軟なインタラクションが可能となる。

2　関連研究

IF は、状態観測と行動選択の双方がテキストモダリティのみで行われる環境であり、自然言語処理と意思決定能力を測る重要なベンチマークとして機能してきた。 従来の研究では、TextWorld [7] のようにルールベースで手続き的にゲームを生成する手法や、Jericho [8] のように Zork [2] 等の人間が作成した既存の高品質な IF を強化学習環境として整備するアプローチが主流であった。これらの環境は、ゲームのルールや遷移がプログラムとして厳密に定義されている [2] ため、論理的な整合性やクリア可能性は保証されている。

一方で、近年の LLM の発展に伴い、あらかじめ定義されたルールに縛られず、LLM を用いて動的にストーリーや世界を生成する試みも活発に行われている。 例えば、Generative Agents [3] は、LLM を用いて複数のエージェントが相互作用する社会的なシミュレーションが可能であることを示した。また、AI Dungeon [4, 5] に代表される IF 生成システム

では、ユーザーの自由な入力に対して柔軟に応答できる高い没入感が実現されている。

しかし、LLM を世界シミュレーターとして利用する場合、その信頼性には課題が残る。 最近の研究では、テキストゲームの次状態予測においてオブジェクトの状態変化の一貫性を保つことや物理的な制約を遵守することに困難が伴うことが指摘されている [9]。 LLM は確率的に次トークンを予測する性質上、長期的な因果関係を維持することが難しく、ゲームとして成立し得ない状態や、DE 状態を予期せず生成してしまうリスクがある。

本研究で扱う DE の検出は、現在の状態からゴール状態への到達可能性を検証する問題であり、本質的には長期的なプランニング能力が要求される。しかし、LLM 単体でのプランニング能力については懐疑的な報告がなされている。 Valmeekam らは、LLM に対して古典的なプランニング問題を解かせる包括的な実験を行い、LLM は単純なタスクであっても、自律的に有効な計画を生成する能力が著しく低い（成功率が〜12%程度）ことを明らかにした [10]。 彼らは、LLM が「もっともらしい計画」を生成することはできても、前提条件のチェックや厳密な因果関係の追跡において頻繁に失敗することを示している。したがって、LLM に行動生成を委ねる本研究の IF システムにおいて、LLM 自身にその行動が DE を引き起こすか否かを判断させることは信頼性の観点から困難である。

一方、LLM の持つ柔軟な言語処理能力と、論理的な厳密さを補完する手法として、ニューロシンボリック AI のアプローチが注目されている。 特に、LLM を自然言語と形式言語の翻訳機として利用し、推論や計画作成自体は外部のシンボリックソルバーに委ねる手法が有効である。Liu らは、LLM+P と呼ばれるフレームワークを提案した [11]。これは、自然言語で記述された問題を LLM が PDDL（Planning Domain Definition Language）に変換し、古典的なプランナーで解を求めた後、再び LLM が自然言語に翻訳してユーザーに提示するものである。この手法により、LLM 単体では解けない複雑な計画問題に対しても、最適解を導出できることが示されている。

本研究は、ニューロシンボリック手法の枠組みを「ゲーム環境そのものの構築と検証」に応用する点に新規性がある。 具体的には、本提案手法は LLM+P のようなアプローチを発展させ、ゲーム固有のテキスト情報をスキーマに基づいてモデル化し、プランナーを用いて「プレイヤーが詰んでいないか（DE か否か）」をシステム側が常時監視するものである。 これにより、LLM による柔軟な行動生成という利点を活かしつつ、ニューロシンボリックな検証によってゲームとしての成立性（到達可能性）を保証するシステムを実現する。

3　手法

本章では、IF における DE 検出のための提案手法を説明する。本手法は、オペレータやオブジェクトの属性に関するゲーム非依存のスキーマを定義し、LLM によってゲーム固有情報で具体的にモデル化した後、ゲームの状態についてシンボリックプランニングによりゴール到達可能性を判定するものである。図 1 はシステム全体の概要図であり、ゲーム仕様からモデルを生成し、それを用いてデッドエンド検出可能なゲームプレイを行う流れを表している。灰色背景の吹き出しは具体例である。

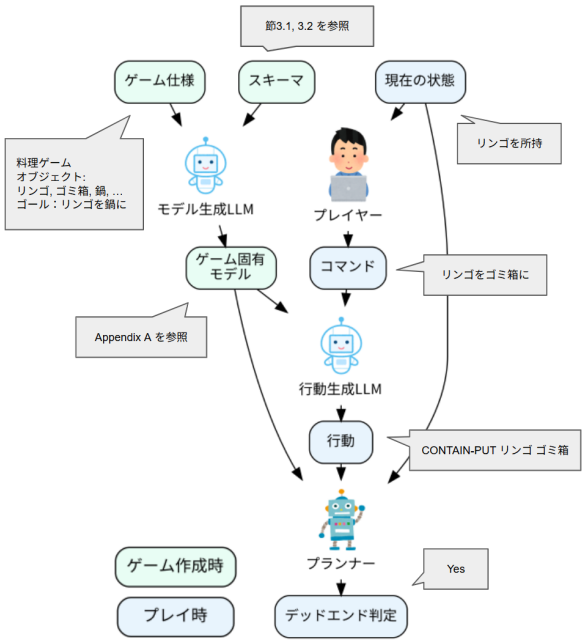

図 1　システム全体概要図

3.1　システム概要

本システムは、ゲーム仕様、スキーマ、LLM、プランナーの 4 要素を主軸として構成される。まず、ゲーム仕様として、オブジェクト、場所、初期状態、ゴール条件を与えると、LLM がオブジェクト属性や制約を推論し、スキーマに則った具体的なモデルを作る。これを PDDL 形式に変換し、シンボリックプランナーによって現在状態からゴールへのプラン

の有無を判定する。プランが存在しない場合、その状態を DE と判定する。LLM の役割を知識抽出とモデル上の行動生成に限定し、論理的な到達可能性判定をプランナーに委ねることで、LLM の推論誤りが DE 判定に与える影響を抑制する。

3.2 スキーマ

スキーマは、IF に共通する基本的な物理・論理操作を抽象化したものであり、オブジェクト属性、抽象オペレータ、基本述語から構成される。各オブジェクトには 9 種類の静的ブール属性を割り当てる（表 1）。これらには整合性制約があり、システムが自動的に検証する。また、10 種類の抽象オペレータを定義し、それぞれに基本的な前提条件を与える（表 2）。状態表現には、位置関係、存在、所有、生存状態などを表す基本述語と、属性に対応する静的述語を用いる。なお、どの範囲の DE を検出することができるかはスキーマをどのように定義するかに依存するが、今回のスキーマにおいて検出に失敗する DE の例は 4.1 節で簡単に触れている。

表 1 オブジェクト属性一覧

属性	説明
is_unique	唯一のインスタンスが存在
is_destructible	破壊可能
is_restorable	破壊後に復元可能
is_consumable	消費可能
is_transferable	移動・所持可能
is_container	他のオブジェクトを格納可能
is_alive	生命体である
is_location	場所として機能
is_light_source	光を発する

表 2 抽象オペレータ一覧

オペレータ	説明
MOVE	オブジェクトを移動
TAKE	オブジェクトを拾う
DROP	オブジェクトを置く
CONSUME	オブジェクトを消費
DESTROY	オブジェクトを破壊
RESTORE	破壊されたオブジェクトを復元
DAMAGE	生命体にダメージ
CONTAIN-PUT	オブジェクトを入れる
CONTAIN-GET	オブジェクトを取り出す
PLAYER-MOVE	プレイヤーが移動

3.3 スキーマからのモデル化

モデル化には GPT-4o を用いる。LLM は、(1) 各オブジェクトへの属性割り当て、(2) 抽象オペレータへのゲーム固有制約の付与、(3) ゴール達成に必須なオブジェクトの識別、を行う。LLM は DE 判定を直接行わず、その出力は JSON 形式で構造化され、プランナー入力の生成に用いられる。オペレータ制約は、特定オブジェクト間での操作禁止や、追加前提条件を表す。モデルの例は Appendix A に示す。なお、本研究では行わないが、このモデルは後から人手でも修正可能であり、ブラックボックス化されないという点に注意する必要がある。

3.4 到達可能性判定

モデルから生成した PDDL ドメインと、到達可能性の判定対象である状態とゴール状態から生成される問題ファイルを Fast Downward（LAMA-first 設定）[12, 13] を入力してゴール到達可能性を判定する。プランが見つかれば到達可能、見つからなければ DE と判定する。これにより、モデルによって表現された物理的・論理的に可能な状態遷移に基づく DE 検出が可能となる。

4 実験結果と議論

4.1 デッドエンド検出

3 節で述べた DE 検出システムを用いて、4 つのベンチマークゲームとして料理（材料収集・調理）、戦闘（敵討伐・宝獲得）、迷宮（鍵・暗闇）、錬金術（材料収集・調合）を用いる。各ゲームの詳しい説明は省くが、料理の場合においてどのようなモデルが実際に生成されるかの例を Appendix A に示す。

まず、これらのゲームに関する予備実験として、LLM によって自動生成されたモデルがどれほど適切かを評価するため、様々な状態をテストケースとして DE 検出システムに与え、DE 判定が正しく行えるかを調査する。各ゲームごとに約 30 件のテストケースを用意しており、半数近くが DE 状態になっている。またベースラインとして、プランナーなしで LLM に直接判定させたものやランダムなものに対しても実験を行い、これらとの結果を比較する。全ゲームで集計した正解率を表 3 に示す。この結果から、PDDL 化した際に正しく DE 検出できるかどうかという観点において、LLM によるモデル

表 3　DE 検出に関する正解率

手法	料理	戦闘	迷宮	錬金術	平均
提案手法（LLM が生成したモデル + プランナー）	0.931	**1.000**	**1.000**	0.972	0.976
手動で作成したモデル + プランナー	**0.966**	**1.000**	**1.000**	0.972	**0.985**
Chain-of-Thought 推論で LLM が直接判定	0.655	0.800	0.839	**1.000**	0.824
LLM が直接判定（Chain-of-Thought なし）	0.690	0.533	0.613	0.861	0.674
ランダム（$p = 0.5$）	0.655	0.533	0.581	0.556	0.581

表 4　LLM プレイヤーのゴール達成率

DE 検出	料理	戦闘	迷路	錬金術	全体
あり	**0.92**	0.93	**1.00**	**0.94**	**0.948**
なし	0.88	**0.99**	**1.00**	0.86	0.932

は手動で作成したものと同等の性能を示すことがわかる。なお、手動で作成したモデルでも一部で誤りが発生しているが、これは今回設計したスキーマの限界を示すものである。そのテストケースは CONTAIN-GET オペレータに関するもので、具体的には、「中の物を回収可能なコンテナ A とそうでないコンテナ B があり、A が B の中に入っているとき、A の内容物を取り出すことはできないので、A にゲームクリアに必須なアイテムがある場合は DE となる」という、入れ子のコンテナによる間接的なブロッキングを含むものである。

4.2　LLM プレイヤーのゴール達成率

節 4.1 にて、LLM によって生成されたモデルが、手動で作成したものに匹敵する性能を示すことが確認できた。続いて、実際にこの DE 検出がゲームのプレイ体験をどのように変化させるかについて実験を行う。具体的には、DE 検出システムを組み込んだゲームエンジンを用い、LLM にプレイヤーとして実際にプレイさせて、ゲームをクリアできるかどうかを実験する（図 1 を参照）。その際、もし DE 状態に入るアクションの実行を試みた際には、自動的にそれをキャンセルし、その旨を LLM プレイヤーに伝え引き続きプレイさせるようにする。またベースラインとして、DE 状態に入るアクションを許容した場合（つまり DE 状態に入ってしまっても LLM プレイヤーは気づかない）でも実験する。節 4.1 で用いた 4 つのサンプルゲームに対し、100 回ずつ LLM（GPT-4o-mini）にプレイさせたときのゴール達成率を表 4 に示した。なおそれぞれのゲームの初期状態からゴール状態までの最短ステップ数の 2 倍を最大のステップ数とし、それ以上かかる場合を失敗とみなしている。また、LLM は人間がプレイするときと同様に行動をテキストとして出力するものとし、それを抽象オペレータを使ったコマンドに変換するために別途 LLM（GPT-4o-mini）を用いている。料理、錬金術、そして全体においては DE 検出ありがなしの場合を上回ったが、戦闘では逆の結果になった。統計学的有意水準を $p = 0.05$ として、全体の結果に関してフィッシャーの正確確率検定を行ったところ、$p = 0.457$ となり、統計的に有意な差は認められなかった。したがって、DE 検出が上手くいったとしても、プレイヤーがどれだけゴールに到着しやすいかに及ぼす影響は統計的には明らかでなかった。

この原因としては、そもそも今回扱っている 4 つのゲームが単純すぎるために DE 状態に到達しにくいことが挙げられる。実際、戦闘については最短で 4 ステップでゴール状態にたどり着くことができ、一方錬金術は 20 ステップ以上かかるので、複雑なゲームにおいては DE 検出が役立つという可能性がある。また別の原因としては、Fast Down-ward プランナーを LAMA-first 設定で動かしたために到達化可能性判定が不完全で、DE の誤検出が生じてしまったことなどが考えられる。

5　限界と結論

本稿では、IF のためのスキーマを定義し、具体的な IF の仕様を LLM に与えることで、ゲームの挙動を制御するモデルを生成でき、かつそれが DE 検出に用いることが可能であることを実験的に示した。LLM によって生成されたモデルは、DE の検出という観点から見て十分な精度であることが確認できたが、実際のゲームのプレイ体験の向上（有限ステップ数でのクリア率）といった観点からは有意な結果は得られなかった。DE の検出精度は、LLM 自身の精度以外にも、スキーマをどのように定義しておくかに大きく依存する。今後はより汎用的なスキーマの定義や、より広範囲なドメインのゲームでの実験、また異なるモデルの LLM の場合における精度の改善等に取り組む必要がある。

謝辞

本研究は、文部科学省補助事業「生成 AI モデルの透明性・信頼性の確保に向けた研究開発拠点形成」の支援を受けたものです。

参考文献

[1] Nick Montfort. **Twisty little passages: An approach to interactive fiction**. Mit Press, 2005.

[2] P. David Lebling, Marc S. Blank, and Timothy A. Anderson. Zork: A computerized fantasy simulation game. **Computer**, Vol. 12, No. 4, pp. 51–59, 1979.

[3] Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior, 2023.

[4] Nick Walton. Ai dungeon 2: Creating infinitely generated text adventures with deep learning. https://pcc4318.wordpress.com/2019/11/21/ai-dungeon-2-creating-infinitely-generated-text-adventures-with-deep-learning-language-models/, 2019. Accessed: 2026-01-05.

[5] Latitude. AI Dungeon. https://play.aidungeon.io/, 2019. Online; accessed 2026-01-05.

[6] Eric Zhou, Shreyas Basavatia, Moontashir Siam, Zexin Chen, and Mark O. Riedl. Story2game: Generating (almost) everything in an interactive fiction game, 2025.

[7] Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew J. Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. Textworld: A learning environment for text-based games. **CoRR**, Vol. abs/1806.11532, , 2018.

[8] Matthew J. Hausknecht, Prithviraj Ammanabrolu, Marc-Alexandre Côté, and Xingdi Yuan. Interactive fiction games: A colossal adventure. **CoRR**, Vol. abs/1909.05398, , 2019.

[9] Ruoyao Wang, Graham Todd, Eric Yuan, Ziang Xiao, Marc-Alexandre Côté, and Peter Jansen. Bytesized32: A corpus and challenge task for generating task-specific world models expressed as text games, 2023.

[10] Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. On the planning abilities of large language models : A critical investigation, 2023.

[11] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. Llm+p: Empowering large language models with optimal planning proficiency, 2023.

[12] M. Helmert. The fast downward planning system. **Journal of Artificial Intelligence Research**, Vol. 26, p. 191–246, July 2006.

[13] S. Richter and M. Westphal. The lama planner: Guiding cost-based anytime planning with landmarks. **Journal of Artificial Intelligence Research**, Vol. 39, p. 127–177, September 2010.

A　料理ゲームにおけるモデル例

ここでは、料理ゲームにおいて実際に LLM が生成したモデルの例を示す。ABSTRACT-CONTAIN-GET は表 2 における CONTAIN-GET に対応する。

```
{
  "objects": [
    {"name": "tomato", "attributes": {"is_unique": true, "is_destructible": true, "is_restorable":
        false, "is_consumable": true, "is_transferable": true, "is_container": false, "is_alive": false
        , "is_location": false}},
    {"name": "onion", "attributes": {"is_unique": false, "is_destructible": true, "is_restorable":
        false, "is_consumable": true, "is_transferable": true, "is_container": false, "is_alive": false
        , "is_location": false}},
    {"name": "knife", "attributes": {"is_unique": true, "is_destructible": true, "is_restorable": false
        , "is_consumable": false, "is_transferable": true, "is_container": false, "is_alive": false, "
        is_location": false}},
    {"name": "pot", "attributes": {"is_unique": true, "is_destructible": true, "is_restorable": false,
         "is_consumable": false, "is_transferable": true, "is_container": true, "is_alive": false, "
        is_location": false}},
    {"name": "trash_bin", "attributes": {"is_unique": false, "is_destructible": false, "is_restorable":
         false, "is_consumable": false, "is_transferable": false, "is_container": true, "is_alive":
        false, "is_location": false}},
    {"name": "fridge", "attributes": {"is_unique": false, "is_destructible": false, "is_restorable":
        false, "is_consumable": false, "is_transferable": false, "is_container": true, "is_alive":
        false, "is_location": false}},
    {"name": "recipe_book", "attributes": {"is_unique": false, "is_destructible": true, "is_restorable
        ": false, "is_consumable": false, "is_transferable": true, "is_container": false, "is_alive":
        false, "is_location": false}},
    {"name": "spare_pot", "attributes": {"is_unique": false, "is_destructible": false, "is_restorable":
         false, "is_consumable": false, "is_transferable": true, "is_container": true, "is_alive":
        false, "is_location": false}}
  ],
  "constraints": [
    {
      "operator": "ABSTRACT-CONTAIN-GET",
      "condition": "trash_bin からは物を取り出せない",
      "blocked_pairs": [{"subject": "*", "target": "trash_bin"}]
    }
  ],
  "goal_critical_objects": ["tomato", "onion", "pot"]
}
```