時相論理を用いた物語のエンティティ状態検索

佐藤 浩輔 [1]　頼 展韜 [1]
[1] 株式会社バンダイナムコ研究所
{k18-sato,z-lai}@bandainamco-mirai.com

概要

近年大規模化しているゲームやアニメ、映画、漫画などといった物語を扱うコンテンツにおいて、物語およびその設定に関わる知識をどのように整備・運用するかが課題になっている。本研究では、物語の制作・監修を容易にするために動的に変化する物語世界内のエンティティの状態に関わる事実を検索する手法を提案する。具体的には① LLM によって物語テキストから情報抽出・構造化を行い、②時相論理を用いて検索を行う。提案手法を単純な構造を持つ物語に適用し、簡易的に評価を行った。

1　はじめに

近年、ゲームやアニメ、映画や漫画といった物語コンテンツ市場が世界的に拡大を続けている [1]。その中で作品の長期継続やスピンオフ、シェアード・ユニバースなどによって非常に大規模な作品世界が展開されるコンテンツも今日では珍しくない。

このようなコンテンツの越境化・大規模化に伴い物語に関わる膨大な情報をどのように管理・運用するかが課題になっており、設定や内容の確認を簡便に行うための技術が必要とされている [2]。

本研究では、時間を考慮して物語におけるエンティティの状態を検索する手法について提案する。

1.1　状態変化の記述としての物語

物語 (narrative) とは送り手から受け手への事象 (event) の報告である [3]。事象とは、行動や行為、偶発的な事象といった「状態の変化」を指す。

古典的には物語は物語内容 (story / fabula) と物語言説 (discourse / plot / sjuzět) から構成される [3]。前者が物語世界内の事象を時系列順に並べたもので、「何を」語るかに該当するのに対して後者は物語のある順序・方法での語り方すなわち「いかに」語るかに該当する。物語内容はひとつの物語に対してただひとつ想定されるのに対して物語言説は同一の物語内容であっても無数に存在し得る。

我々の関心は物語コンテンツの監修や検索にある。言い換えれば、物語世界内における事実、つまり物語内容の方に主要な興味がある。

Chatman(1978) は物語内容を伝達する陳述としてある時点の状態を記述する状態陳述 (stasis statement) と、状態の変化すなわちイベントを記述する経過陳述 (process statement) の 2 つがあるとした [4]。物語内容に着目した場合、物語は「物語世界内のエンティティの状態とその変化の記述」とみなすことができる。課題は、いかにこれら記述の集積から情報を抽出し、検索を可能にするかということになる。

1.1.1　物語の情報抽出

あるエンティティのある時点における状態を知るには、単純に考えれば、時系列順に並んだ記述から逐一状態とその変化を抽出していけばよい。

しかしながら、自然言語の文章においてはすべての情報が明示的に語られているわけではない。

発話における前提 (presupposition) とはある命題が発話される前にあらかじめ知られていなければならない命題のことである [5]。例として、「フランス国王は禿である」という命題は「フランス国王が存在する」という命題を前提している。人間にとって自明な情報はしばしば省略されるため、省略された前提を適宜推測して補う必要がある。一方で記述の粒度の問題もあり、周辺的・冗長な情報は捨象し、主要な情報のみを選択的に抽出する必要がある。

1.1.2　状態変化の表現

一般的な知識ベース、たとえば知識グラフは基本的に静的な知識体系を対象にしており、時間によって動的に変化する知識をうまく扱えない。

知識すなわち命題について、古典論理 (一階述語論理) の範囲では複数の時点を含んだり時点によって真理値の異なる命題を表現できない[1)]。

1) たとえば「彼はいつもおなかを空かせている」「買い続けていればいつかは宝くじが当たる」「明日は雨が降る」といった命題は表現できない。

そのような時や人、場合によって変わりうる様相 (modal) を扱うのが様相論理 (modal logic) である。様相論理のひとつである時相論理 (temporal logic) は時間に関する論理で、「いつでも真」「将来のある時点で真」といった複数の時点に関わる命題を扱える。この時相論理を活用した技術がモデル検査である。

1.2 時相論理とモデル検査

形式手法のひとつであるモデル検査は、ソフトウェアやシステムなどといった状態遷移系をモデル化し、時相論理式で記述された特定の性質 (仕様) を系が満たすかを検証する技術である [6, 7][2]。

モデル検査で使われる時相論理に計算木論理 (Computational Tree Logic; CTL) がある。

1.2.1 計算木論理 (CTL)

CTL を使って状態遷移系のモデル検査を行う場合、系の状態とその遷移関係をクリプキ構造 (Kripke structure) として表現する。クリプキ構造 M は以下の 5 つ組によって定義される [6]。

$$M = (S, S_0, R, AP, L)$$

ただし S は状態の集合、$S_0 \subseteq S$ は初期状態の集合、$R \subseteq S \times S$ は状態間が直接遷移可能かを示す到達可能関係、AP は部分論理式を持たない原子命題 (atomic proposition) の集合、$L : S \to 2^{AP}$ は各状態において真である原子命題の集合を返す付値関数 (labeling function) である。

到達可能関係によって連続する状態の系列をパス (経路) と呼び、CTL では命題論理の演算子 $(\neg, \wedge, \vee, \to)$ に加えて、以下の演算子が使える[3]。

状態演算子　ある状態から始まるパスを対象とする演算子である。論理式を ϕ として、

Xϕ : 「次の (ne**X**t) 状態で ϕ が成り立つ」

Fϕ : 「この先いつか (**F**inally) ϕ が成り立つ」

Gϕ : 「ずっと (**G**lobally) ϕ が成り立つ」

ϕ_1**U**ϕ_2:「ϕ_2 が成り立つまで (**U**ntil) ϕ_1 が成り立つ」

の 4 種類がある[4]。

パス演算子　ある状態から始まるパスの集合を対象とする演算子である。論理式を ϕ として、

Aϕ : 「すべて (**A**ll) のパスで ϕ が成り立つ」

Eϕ : 「ϕ が成り立つパスが存在 (**E**xists) する」

の 2 種類がある。

CTL では、パス演算子・状態演算子を単独で使用することができず、必ず組み合わせて使わなければならない[5]。すなわち、 **AX**, **AF**, **AG**, **AU**, **EX**, **EF**, **EG**, **EU**, の 8 つが基本的な演算子となる。

モデル検査では系をクリプキ構造 M、仕様を CTL 論理式 ϕ で表現し、系が仕様を満たすかを M が ϕ を充足するか ($M \models \phi$ であるか) で判定する。

1.3 本研究のアプローチ

本研究では、物語世界をひとつの状態遷移系、物語テキストをその状態変化の記述とみなし、テキストから情報を抽出した上で事実を検索する手法を提案する。具体的には、① LLM(大規模言語モデル) を使って物語内の情報を抽出・構造化し、②時相論理を使ったクエリを用いて物語におけるエンティティに関わる事実を時間を考慮して検索する。

物語の構造化にあたっては、LLM を使うことで余分な情報を捨象しつつ文脈を考慮したうえで各時点における状態の記述と状態の変化を抽出する。

検索では、時相論理とモデル検査の技術を用いることで、特定の条件を満たす時点を列挙したり、時間に関する命題の真偽を調べることを可能にする。

2 関連研究

物語と知識グラフ　物語の構造化、特に知識グラフとの関わりで、山田ら (2024) はアイスランドのサガの知識グラフによる構造化を試みている [8]。LLM(ChatGPT) を用いて処理を行い、人物の親族関係について知識グラフと物語内のイベントについての知識グラフの構築を効率的に行う手法を提案している。また 2018 年より開かれている技術コンテストである「ナレッジグラフ推論チャレンジ」[6]では推理小説 (シャーロック・ホームズ) の短編を題材に、RDF により表現された知識グラフを用いて犯人を推論することが行われた [9]。

これらの研究は知識グラフにより物語の情報を包括的に記述し活用することを目的としている一方で、本研究では特に状態の変化に着目し、動的な情報に特化して検索することを目的としている。

物語とモデル検査　モデル検査を物語に関わる分野に応用した例として、ビデオゲームへの応用があ

2) たとえば「望ましくないことがずっと起きない (e.g. デッドロックは生じない)」、「望ましいことがいつか必ず起きる (e.g. リクエストがあれば必ずレスポンスが返る)」、「どの時点でも特定の性質が実現する (e.g. いつでも応答が返る)」といった形で表現される

3) CTL の構文および意味論については付録 A にまとめた。

4) 状態演算子に **R**(**R**elease) や **W**(**W**eak until) が追加されることもあるが、これらは他の演算子から導出できる。

5) CTL の拡張である CTL*ではこの制約が外れ、パス演算子・状態演算子を単独で扱うことができる。

6) https://challenge.knowledge-graph.jp/

る [10, 11, 12, 13, 14, 15] 。これらはゲームシナリオにおけるフラグや変数の管理・スクリプトの動作・ゲーム仕様の検証としてモデル検査を用いており、監修のための検索に用いる本研究の観点とはスコープが異なる。またフラグや変数はスクリプトやタグで明示的に表現されていることが必要であり、自然言語で記述されたテキストからの自動的な情報抽出を試みる本研究とは異なる。

本研究の意義は、時相論理式とモデル検査の技術を事実の検索という形で物語というドメインに応用可能なこと、LLM を使うことで検索に適した形で自動的に物語テキストから情報を抽出・構造化ができることを示した点にある。

3 方法

LLM を用いてテキストから情報を抽出し構造化するフローおよび計算木論理 (CTL) を用いて検索を行うプログラムを実装した。抽出・構造化には OpenAI の GPT-4o を用い、検索プログラムの実装には Python(3.11) を用いた。

3.1 物語テキストからの情報抽出・構造化

3.1.1 前処理

自然言語で記述された物語を入力として、以下の処理を行った。

- 直接話法で書かれたセリフを行為に変換 (e.g.「おはよう」という発話は “挨拶した”に)
- 不要な副詞や擬音・修辞的な装飾の除去
- 各文がひとつの動詞を含む文になるように (複数の) 短文に変換

3.1.2 中間表現の抽出

各文をひとつのイベントとみなし、RDF ライクな表現でスキーマを作成させた。なお今回はできごとが時系列順に記述されていることを想定した。

各イベントには連番でイベント ID を付させた。

初めて登場するエンティティについては都度リソースを作成させた。また、目的語が行為の場合はそれもエンティティ化させた。

3.1.3 イベントと状態の抽出

中間表現からイベントと状態を抽出させた。

イベントの抽出については、各イベントに含まれる主語・述語・目的語をトリプルとして列挙させた。

状態の抽出については、著者らの過去の研究 [2] から、(1) 人や事物・所有権の移動、(2) 物理的な状態の変化 (生死・肉体・変身など)、(3) 社会的な状態の変化 (名称・称号・婚姻など) といった状態の変化が物語 (「グリム童話」) において重要であるということがわかっているため、それらを重点的に主語・属性 (述語)・値のトリプルとして抽出・列挙させた。述語や属性はなるべく汎用的な単語になるよう指示を与えた。また、発話の前提を含む、明示的でない状態についても記述させるよう指示した。

最後にイベントと状態それぞれについてイベント ID とトリプルのリストを出力させた。

出力後、状態に関して、所有を表す述語 `has` については主語と値を入れ替え、物を主語として `isOwnedBy` で表すトリプルに変換する処理を行った。

3.2 状態の検索

3.2.1 状態遷移系の表現

モデル検査を適用するために、状態の遷移をイベント ID のリスト、イベント間の遷移関係、初期状態のイベント ID から構成される状態遷移系として表現した。

3.2.2 基本的な検索処理

検索では、時相論理式と、論理式内の命題変数に対応するトリプルを入力とした。処理はまず、与えられたトリプルについてデータを検索し、各時点の真理値を取得する。データには状態が観測または変更された時点のみが格納されているため、値がない部分については変更がないと仮定し、イベント間の遷移関係を用いて各時点の状態を保存しながら原子命題が真となる時点の集合 (充足集合) を得る。その後文字列として与えられた時相論理式に対し、部分論理式ごとに再帰的に処理を行い集合演算を行うことで求める論理式の充足集合が得られる (処理の詳細については付録 A.3 を参照)。

3.2.3 追加の検索クエリ

該当する状態の集合を返すクエリの他、追加で以下のようなクエリを想定し、実装した。

- 物語全体における真偽を判定するクエリ：
  e.g.「オオカミは最終的に死ぬか」
- 該当するする時点のイベントを返すクエリ：
  e.g.「飲み込まれている間に、何が起きたか」

3.3 評価

グリム童話「赤ずきん」を用いて、評価を行った。著名な童話で内容の確認がしやすく物語内の時系列と記述の順序が一致しているためである。入力テキ

ストは青空文庫で公開されている「赤ずきん」(矢崎源九郎訳) [7] を用いた。

ルビを削除した本文に対して処理を行い、抽出されたデータについて評価を行った。また、3.2.3 節で挙げたクエリを実行して検索を行った。

4 結果

イベントの抽出 19 件のイベント、のべ 27 件のトリプルが抽出された (表 1)。物語の主要なイベントは抽出されているといえる。ただしオオカミがおばあさんに化けたり、赤ずきんを飲み込んだ後に眠るという記述が欠落している。

状態の抽出 19 件のトリプルが抽出された (表 2)。発話の前提については、オオカミや赤ずきんについては場所に関する状態 (in) が抽出できているが、おばあさんに関しては抽出できていない。またおばあさんが病気であったこと、物語の最後で回復したことが状態に含まれておらず、物語の細部までは抽出されていないといえる。

表 1 抽出されたイベント

ID	トリプル
1	(ex:Grandmother, make, ex:RedHood)
2	(ex:Mother, ask, ex:Task)
3	(ex:LittleRedRidingHood, promise, ex:Task)
4	(ex:LittleRedRidingHood, enter, ex:Forest)
5	(ex:Wolf, appear, null)
5	(ex:Wolf, ask, ex:Destination)
6	(ex:LittleRedRidingHood, answer, ex:Destination)
7	(ex:Wolf, suggest, ex:Activity)
8	(ex:LittleRedRidingHood, gather, ex:Flowers)
9	(ex:Wolf, go, ex:GrandmotherHouse)
9	(ex:Wolf, swallow, ex:Grandmother)
10	(ex:LittleRedRidingHood, finish, ex:Activity) (ex:LittleRedRidingHood, go, ex:GrandmotherHouse)
11	(ex:LittleRedRidingHood, enter, ex:GrandmotherHouse) (ex:LittleRedRidingHood, greet, ex:Grandmother)
12	(ex:Wolf, swallow, ex:LittleRedRidingHood)
13	(ex:Hunter, find, ex:Wolf) (ex:Hunter, cut, ex:WolfStomach)
14	(ex:Hunter, save, ex:LittleRedRidingHood) (ex:Hunter, save, ex:Grandmother)
15	(ex:LittleRedRidingHood, fill, ex:WolfStomach)
16	(ex:Wolf, wake, null) (ex:Wolf, die, null)
17	(ex:Hunter, take, ex:WolfFur)
18	(ex:Grandmother, enjoy, ex:Goods) (ex:Grandmother, recover, null)
19	(ex:LittleRedRidingHood, decide, ex:Resolution)

表 2 抽出された状態

ID	トリプル
1	(ex:LittleRedRidingHood, has, ex:RedHood)
2	(ex:LittleRedRidingHood, has, ex:Goods)
3	(ex:LittleRedRidingHood, promise, ex:Task)
4	(ex:LittleRedRidingHood, in, ex:Forest)
5	(ex:Wolf, in, ex:Forest)
8	(ex:LittleRedRidingHood, has, ex:Flowers)
9	(ex:Wolf, in, ex:GrandmotherHouse)
9	(ex:Grandmother, isSwallowed, true)
10	(ex:LittleRedRidingHood, in, ex:GrandmotherHouse)
12	(ex:LittleRedRidingHood, isSwallowed, true)
13	(ex:Hunter, in, ex:Forest)
14	(ex:LittleRedRidingHood, isSwallowed, false)
14	(ex:Grandmother, isSwallowed, false)
15	(ex:WolfStomach, has, ex:Stones)
16	(ex:Wolf, isAlive, false)
17	(ex:Hunter, has, ex:WolfFur)
18	(ex:Grandmother, has, ex:Goods)
19	(ex:LittleRedRidingHood, hasResolution, ex:Resolution)

表 3 検索の結果

クエリ	結果
①「赤ずきんが森に入っておばあさんの家に着くまで」	4, 5, 6, 7, 8, 9
②「オオカミは最終的に死ぬか」	true
③「赤ずきんが飲みこまれている間に何が起きたか」	(ex:Hunter, find, ex:Wolf), (ex:Hunter, cut, ex:WolfStomach), (ex:Hunter, save, ex:LittleRedRidingHood), (ex:Hunter, save, ex:Grandmother)

エンティティの状態検索 クエリを実行した結果を示した (表 3)。該当する状態を示すクエリ (①)、物語全体に関する真偽を判定するクエリ (②)、ただしデータに含まれていない情報については正しい結果が返らないことに留意が必要である。

5 考察

本研究では物語コンテンツの制作・監修作業の簡便化に向け、LLM を用いて物語テキストから情報を抽出・構造化した上で、時間を考慮してエンティティに関する事実を検索できる手法を提案した。

現実の物語テキスト (「赤ずきん」) に対して本手法を適用したところ、検索に関しては論理ベースの手法であるため情報が正しく抽出されている限りにおいては十分機能すると考えられる。一方で、情報抽出については物語の中核的な内容については問題なく抽出できるものの、発話の前提や詳細など周辺的な部分については欠落があった。LLM やプロンプトによる改善や、抽象化と詳細とのバランスをどのようにとるかが課題であるといえる。

その他今後の課題としては。より複雑な物語への適用可能性がある。一般に、物語テキスト内の事象は起こった順番で提示されるとは限らない。より複雑な物語に対しては物語世界内で発生した順序で事象を配列しなおすといった処理が必要となろう。また、今回は小規模な物語を扱ったが、より規模の大きい物語についてうまく動作するかを検証する必要がある。CTL は分岐やループを扱えるため、ゲームのシナリオのような分岐を含む物語についても今後適用可能性を検討できるだろう。

本研究の手法をより広範に活用するためには、知識グラフやオントロジーとの連携についても考える必要がある。静的な知識を扱う機構と動的な知識を扱う本手法とを組み合わせることでより強力な検索機能が実現できると考えられるが、強力な表現力を持つ推論機構は必ずしも効率的な形で実装できるとは限らない [16]。コンテンツ産業への応用に向け、今後さらなる研究が必要である。

7) https://www.aozora.gr.jp/cards/001091/files/59835_72466.html

参考文献

[1] 一般社団法人日本経済団体連合会. Entertainment contents ∞ 2023 参考資料集. Technical report, 2023.

[2] 佐藤浩輔, 頼展韜. 知識グラフ構築に向けた物語文の構造分析. 言語処理学会 第 30 回年次大会 発表論文集, 2024.

[3] G. Prince. **A dictionary of narratology**. the University of Nebraska Press, 1987. (ジェラルド・プリンス (著) 遠藤健一 (訳) (2015). 物語論辞典. 松柏社).

[4] S. Chatman. **Story and discourse: Narrative structure in fiction and film**. Cornell university press, 1978.

[5] 谷中瞳. ことばの意味を計算するしくみ : 計算言語学と自然言語処理の基礎. 講談社, 2024.

[6] E. M. Clarke, O. Grumberg, D. Kroening, D. A. Peled, and H. Veith. **Model checking (2nd ed.)**. MIT Press, 2018.

[7] M. Huth and M. Ryan. **Logic in computer science: Modelling and reasoning about systems**. Cambridge University Press, 2004.

[8] 山田慎太郎, 小川潤, 大向一輝. Chatgpt を用いた人名・地名エンティティの自動抽出〜生成 ai を活用した中世アイスランド・サガの知識グラフ構築にむけて〜. じんもんこん 2024 論文集, pp. 329–334, 2024.

[9] 古崎晃司, 江上周作, 松下京群, 鵜飼孝典, 川村隆浩. 説明生成のための知識グラフ構築ガイドラインの考察 -ナレッジグラフ推論チャレンジを例にして-. 人工知能学会全国大会論文集, pp. 2E6–GS–3–02 1–4, 2022.

[10] I. Hasegawa and T. Yokogawa. Automatic verification for node-based visual script notation using model checking. In **Formal Methods and Software Engineering**, Lecture notes in computer science, pp. 52–68. Springer International Publishing, 2019.

[11] 木間塚達, 野田夏子. モデル検査によるゲームシナリオの 不具合発見ツールの提案. 研究報告ソフトウェア工学（SE）, Vol. 2017-SE-195, No. 13, pp. 1–9, 2017.

[12] 長久勝. 「モデル検査」のススメ （ゲームシナリオ進行編）. In **CEDEC2013**, 2013.

[13] C. J. F. Pickett. Formal verification of computer narratives. Technical report, McGill University School of Computer Science Game Research at McGill, 2005.

[14] N. Rempulsky, A. Prigent, P. Estraillier, V. Courboulay, and M. P. Da Silva. Adaptive storytelling based on model-checking approaches. **International Journal of Intelligent Games & Simulation**, 2009.

[15] 清木昌. ゲームシナリオのモデル検査. 情報処理学会研究報告ゲーム情報学（GI）, Vol. 2004, No. 28(2003-GI-011), pp. 51–56, 2004.

[16] 兼岩憲. OWL の推論とその計算量. コンピュータ ソフトウェア, Vol. 22, No. 4, pp. 26–34, 2005.

[17] チェシャ猫. モデル検査器をつくる〜Go で実装して学ぶ形式手法〜. Dodgson Labs, 2024.

A　CTL とモデル検査

A.1　CTL の構文

CTL 式を ϕ としたとき、CTL の構文は BNF 記法を用いて以下のように書ける[8]:

$$\phi ::= \top|\bot|p|(\neg\phi)|(\phi \wedge \phi)|(\phi \vee \phi)|(\phi \rightarrow \phi)|$$
$$\mathbf{AX}\phi|\mathbf{EX}\phi|\mathbf{AF}\phi|\mathbf{EF}\phi|\mathbf{AG}\phi|\mathbf{EG}\phi|$$
$$\mathbf{A}[\phi\mathbf{U}\phi]|\mathbf{E}[\phi\mathbf{U}\phi]$$

ただし $\top$ は恒真, $\bot$ は矛盾, p は原子論理式を表す.

A.2　CTL の意味論

$M = (S, S_0, R, AP, L)$ をクリプキ構造, ϕ を論理式, ある状態 $s \in S$ について以下のように定義する.

$M, s \vDash \top$ かつ $M, s \nvDash \bot$

$M, s \vDash p$ iff $p \in L(s)$

$M, s \vDash \neg\phi$ iff $M, s \nvDash \phi$

$M, s \vDash \phi_1 \wedge \phi_2$ iff $M, s \vDash \phi_1$ かつ $M, s \vDash \phi_2$

$M, s \vDash \phi_1 \vee \phi_2$ iff $M, s \vDash \phi_1$ または $M, s \vDash \phi_2$

$M, s \vDash \phi_1 \rightarrow \phi_2$ iff $M, s \nvDash \phi_1$ または $M, s \vDash \phi_2$

$M, s \vDash \mathbf{EX}\phi$ iff $M, s_1 \vDash \phi$ である $s_1 \in S$ (ただし sRs_1) が存在する

$M, s \vDash \mathbf{EG}\phi$ iff $s_1 = s$ として, そのパス上のすべての状態 s_i において, $M, s_i \vDash \phi$ が成り立つようなパスが存在する

$M, s \vDash \mathbf{E}[\phi_1\mathbf{U}\phi_2]$ iff $s_1 = s$ から始まるパス $s_1, s_2, s_3...$ のパス上において, $M, s_i \vDash \phi_2$ が成り立つような $i \geq 1$ が存在し, すべての $j = 1, 2, ..., i-1$ において $M, s_j \vDash \phi_1$ が成り立つ

それ以外の演算子は上述の演算子の組み合わせから導出できる. 論理式 ϕ, ψ が意味論的に等価であることを $\phi \equiv \psi$ と表すとき,

$\mathbf{AF}\phi \equiv \neg\mathbf{EG}\neg\phi$

$\mathbf{AX}\phi \equiv \neg\mathbf{EX}\neg\phi$

$\mathbf{EF}\phi \equiv \mathbf{E}[\top\mathbf{U}\phi]$

$\mathbf{AG}\phi \equiv \neg\mathbf{EF}\neg\phi$

$\mathbf{A}[\phi_1\mathbf{U}\phi_2] \equiv \neg(\mathbf{E}[\neg\phi_1\mathbf{U}(\neg\phi_1 \wedge \neg\phi_2)] \vee \mathbf{EG}\neg\phi_2)$

が成り立つ.

8)　本節および次節の記述は文献 [7] を参考にした。

A.3　モデル検査のアルゴリズム

クリプキ構造 $M = (S, S_0, R, AP, L)$ について、CTL 論理式 ϕ ある状態 $s \in S$ で成り立つことを $M, s \vDash \phi$ と表すことにする。$M, s \vDash \phi$ であるすべての s の集合を充足集合 (satisfying set) と呼ぶ。

CTL 論理式 ϕ について, $M, s \vDash \phi$ を満たすすべての s の集合を $\mathsf{SAT}(\phi)$ と表すとすると、与えられた CTL 論理式 ϕ に対する $\mathsf{SAT}(\phi)$ は、以下の手続きを再帰的に適用することで得られる。

1. ϕ が $\top$ なら S を、$\bot$ なら空集合を返す
2. $\phi \in AP$ なら $\{s \in S | \phi \in L(s)\}$ を返す
3. $\neg\phi$ なら $S \setminus \mathsf{SAT}(\phi)$ を返す
4. $\phi_1 \wedge \phi_2$ なら $\mathsf{SAT}(\phi_1) \cap \mathsf{SAT}(\phi_2)$ を返す
5. $\phi_1 \vee \phi_2$ なら $\mathsf{SAT}(\phi_1) \cup \mathsf{SAT}(\phi_2)$ を返す
6. $\phi_1 \rightarrow \phi_2$ なら $(S \setminus \mathsf{SAT}(\phi_1)) \cup \mathsf{SAT}(\phi_2)$ を返す
7. $\mathbf{EX}\phi$ なら $\mathsf{SAT}(\phi)$ に含まれる状態のひとつ前の状態の集合を返す
8. $\mathbf{EG}\phi$ なら終端からはじめて各状態で ϕ が成立するかチェックし、ϕ が成立すれば結果にその状態を追加し、そのひとつ前の状態をキューに追加し、ϕ が成立しなければそのパスの検査は終了する、という手続きを停止するまで続け、その結果の集合を返す[9]
9. $\mathbf{E}[\phi_1\mathbf{U}\phi_2]$ なら、$\mathsf{SAT}(\phi_2)$ に含まれる各状態から始め、そのひとつ前の状態で ϕ_1 が成立するかをチェックし、成立すれば結果にその状態を追加してさらにそのひとつ前の状態をキューに追加する、という手続きを停止するまで続け、その結果の集合を返す
10. それ以外の CTL 演算子なら上記の演算子の組み合わせに変換して計算した結果を返す (A.2 を参照)

検査のアルゴリズムについての詳細は文献 [6, 7] 実装については [17] などを参照のこと。

9)　厳密には無限の繰り返しがあり得るため強連結成分を用いた手法が使われるが、今回は無限の繰り返しはないと仮定し簡便な方法をとった。