

ゲームシナリオのモデル検査 Model Checking for Game Scenarios

清木 昌

Masashi Seiki

mass@is.s.u-tokyo.ac.jp

東京大学大学院情報理工学系研究科

Graduate School of Information Science and Technology,
University of Tokyo

Abstract: As an attempt to reduce the production costs of games with interactive stories using computer science methodologies, this thesis proposes applying the model checking theory to game scenario verification. It enables systematic and automatic verification of consistency conditions such as reachability and freeness from infinite loops. The simplest model checking method enumerates all states explicitly. However, as is well known, the “state explosion problem” instantly occurs in this method. Thus, we apply symbolic model checking techniques based on BDDs (Binary Decision Diagrams), and CTL (Computation Tree Logic) is adopted to represent specifications. By representing the state space symbolically, we are able to deal with mass states and their relational operation as an logical formula and its logical operation. Nevertheless, the computation time is still large. To reduce the time, we focus on characteristics of game scenarios. The transition graph for a typical game scenario is divided into clusters more clearly than the ones for ordinary targets of model checking. The number of checked states is further reduced by the optimization with the live variable analysis. By using these methods, we checked two temporal properties of a game scenario of commercial PC game software, reachability and freeness from infinite loops, in about five minutes.

1 はじめに

ゲーム業界はソフトウェア産業の中で輸出が輸入を大きく上回っている、国際競争力が高い分野である。その中でもロールプレイングゲーム (RPG) と呼ばれる仮想世界の中で主人公とともに物語を疑似体験していく種類のゲームは幅広い年齢層に受け入れられ、全世界でシリーズ総計 5000 万本近くも出荷されているタイトルも生まれている。

近年は、三次元グラフィックスによるリアルな描写や DVD-ROM によるデータの大容量化などを代表とするハードウェア技術の進歩により、ゲームでの表現の幅が広がり、より刺激的なタイトルが続々と登場している。しかし、その陰でハードウェアの進歩にソフトウェアがなかなかついていけないという問題が生まれつつある。この問題は、3D 処理に起因するプログラムの複雑化や 3D モデルとそのモーションの作成によるコスト増大なども引き起こしているが、ここではそれには触れず、もう一つの問題であるデータの肥大化、特に物語を提示することを主とするゲームにおけるシナリオデータの大規模化に着目して議論を進めたい。

記憶媒体の大容量化によって広く複雑な仮想世界を実現することが可能となった。近年では文庫本数十冊分と言われるテキスト量のシナリオが 1 本のゲームの中に収まっていることも当然となっている。ゲームにおけるシナリオとはインタラクティブ性を持ち、状態と選択により分岐していく一種のプログラムである。しかし、そのシナリオを記述するスクリプト言語は開発期間の制限などから、一般的には最低限の条件分岐命令のみを持つシンプルな言語であることが多い。結果として、非常に大規模なシナリオをとともシンプルな言語で処理せねばならないケースが多くなり、そのデバッグは困難を極めている。

本研究では、RPG やアドベンチャーゲームなどと分類されるインタラクティブなストーリー展開を特徴とするコンピュータゲームのシナリオを対象として、デバッグ時の困難を低減するために、記号モデル検査理論を用いたゲームシナリオの自動検証を試みる。モデル検査は状態遷移系の自動的かつ網羅的な検証を可能とする理論である。それをゲームシナリオに適用することによって、ゲーム上重要な性質を検証することが可能となる。

本稿の後半では、実際に市販 PC ゲームのシナリオに対してモデル検査を行い、(1) 全てのイベントに到達可能であること (到達性)、(2) エンディングにたどり着けない状態に陥ることがないこと (「ハマリ」状態の有無)、の 2 つの性質を検証した。検証は計 5 分程度で終わり、実際に到達不可能なシナリオが存在することを発見し、また「ハマリ」状態が存在しないことを保証した。

本稿では、まず 2 節で従来例とその問題点を示し、続いて 3 節でゲームシナリオにモデル検査の理論を用いる手法を述べる。そして 4 節で実際の PC ゲームのシナリオにモデル検査を適用した実験について記述し、最後に現状と課題を再び示して本稿のまとめとする。

2 従来の手法と問題点

2.1 従来のシナリオ記述手法

シナリオ記述言語 (以下、言語、ないしはその言語で書かれたプログラムをスクリプトと呼ぶ) に必要な機能を列挙すると以下ようになるであろう。

- 画面表示の更新
- 内部状態 (フラグ) の保持
- 内部状態の更新
- 状態を参照しての条件分岐
- ユーザ入力の内部状態への反映

そして、従来用いられているものはこれら最低限の機能のみを持ったテキストベースの言語であることが多い。以下に従来のスクリプトとしてありがちな例を上げる。

```
:
*label1501
IF flag(31) = 1 THEN GOTO label1502
IF flag(32) = 1 THEN GOTO label1503
GOTO label1504

*label1502
; 31 番フラグによって実行されるイベント
SHOWBG "background_01.gif"
SHOWIMG "character_01_01.gif"
PRINT "メッセージ出力 1"
WAIT
PRINT "メッセージ出力 2"
WAIT
GOTO label1504

*label1503
; 32 番フラグによって実行されるイベント
SHOWBG "background_02.gif"
SHOWIMG "character_01_02.gif"
PRINT "メッセージ出力 3"
WAIT
flag(33) = SELECT "選択肢 1", "選択肢 2", "選択肢 3"
:
*label1504
; 一般の処理
:
```

図 1: 従来のシナリオスクリプトのイメージ

これはアドベンチャーゲームと呼ばれるテキストとイラスト表示を主としたゲームを意識した例であるが、RPG などの他種のゲームでも (1) 現在の状態に応じて画面表示を行い (2) ユーザの入力により内部状態を変更する、を繰り返すという動作の本質は変わらない。

2.2 従来の手法の問題点

これでは究極の「スバゲッティプログラム」が容易に生まれることは想像に難くない。番号のみで管理された変数を使い、数千から数万のフラットなコードブロック群を条件分岐命令で気ままに移り渡るプログラムを作成/管理/デバッグすることを考えてみればよくわかるであろう。変数値の集合で表されるシナリオの内部状態を製作者が追いきれなくなり、納品直前に大量のシナリオ的なバグ (ストーリーの不整合・あるシーンから抜け出せない・エンディングまで到達できない等) が出ることになる。

今までこの状態で製品を製作できていたということは、こういった言語でもゲーム開発を行うには十分であるという考え方も可能であろう。しかし、それを支えるために多くのデバッグ人員を雇い、多大なマンパワーをかけて人海戦術でバグ出しすることによって対応している現状があることを理解する必要がある。

発売後に修正ができないコンシューマーゲーム機用のゲームソフトにとって、エンディングに到達できないなどのバグは時に製品回収につながる可能性がある。これらの致命的な障害を確実に発見できる手段を提供することの意義は大きい。

3 ゲームシナリオのモデル検査

3.1 モデル検査

モデル検査の技術は主に電気回路や通信プロトコルの分野で活用されてきた [Clarke 99]。明示的に各状態を数え上げていた当初は、計算機の性能の問題もあり数百から数千の状態数のものしか検証できなかったため、規模の小さい回路やプロトコルの検証に利用されていた。しかし、BDD (Binary Decision Diagrams) [Bryant 86] を用いた記号モデル検査 [J.R. Burch 90] や抽象モデル検査 [Clarke 94] などの理論の発展により、 10^{1300} もの状態数を持つ回路に対してもいくつかの性質を検証できるようになっている。

しかし、この技術をゲームのシナリオに適用しようとした例はほとんどない。1990 年に多人数で遊ぶゲームブックに対して通信プロトコルの検証ツールを適用しようとした論文 [柴崎 90] が残っているが、それもその後の展開はなかったようである。

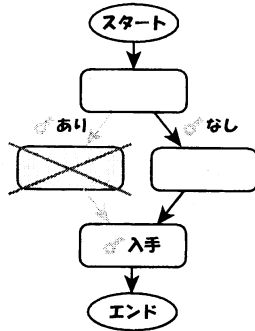


図 2: 到達不能ブロック

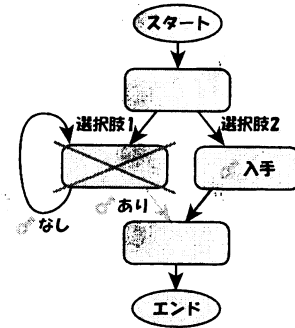


図 3: 「ハマリ」状態

3.2 ゲームシナリオへのモデル検査理論の適用

ゲームシナリオを状態遷移モデルと解釈して取り扱うことによってモデル検査のアルゴリズムの適用でき、ある種の典型的なバグは自動的に発見することが可能である。例を以下に上げる。

実行不能ブロックの存在 図 2 の状態遷移図に×印で示されているような、初期状態からどうやっても到達することのできないブロックを発見する。何らかのバグであることが想定できる。

「ハマリ」状態の存在 図 3 の状態遷移図で×印となっているような、そこからどんな遷移をしたとしてもエンディングまでたどり着けない状態を発見する。俗にハマリ状態と呼ばれており、出荷段階では絶対に存在してはならない状態である。

ストーリーの依存関係の矛盾 ストーリー展開上、あるシーンが起こった後には別のシーンが発生しないとイケないという関係があった場合、それに矛盾する実行経路の存在を発見する。

各種 invariant の矛盾 あるフラグとあるフラグは同時に on になることはない、などといった性質がシナリオ上で常に成り立っているかを検証する。

3.3 CTL による仕様記述

検証したい性質の記述には時相論理の一つである CTL (Computation Tree Logic) を用いることが適当と考えている。CTL は後に説明する「ハマリ」状態の検出 $s_0 \models \mathbf{AG} \mathbf{EF}(\mathbf{PC} = \text{end})$ を記述でき、なおかつ計算量が実用的な範囲で収まるからである。典型的かつ致命的なバグである「ハマリ」状態を検出することは、ゲームシナリオという応用範囲においては非常に重要となる。

CTL は時相論理の一種で通常の論理式の演算子 (and, or, not) の他に図 4 に示す $\mathbf{AG}$, $\mathbf{AF}$, $\mathbf{EG}$, $\mathbf{EF}$

と図には示されていないが $\mathbf{AX}$, $\mathbf{EX}$, $\mathbf{AU}$, $\mathbf{EU}$, $\mathbf{AR}$, $\mathbf{ER}$ の計 10 の時相演算子^{*1}を持つ。

いくつかの時相演算子を以下に説明する。

AG これから先どんな経路をとっても、常に、論理式が成立することを表す。

AF これから先どんな経路をとっても、いずれ、論理式が成立することを表す。

EG ある経路をとると、常に、論理式が成立することを表す。

EF ある経路をとると、いずれ、論理式が成立することを表す。

これらを用いると、さきほどの検出したい性質は次のようにして記述することができる。なお、現在実行しているブロック (CFG 上での 1 ノード) は PC という変数によって表し、初期状態を s_0 で表すことにする。

到達不能ブロックの検出 コンテンツを含む全てのブロック x について、

$$s_0 \models \mathbf{EF}(\mathbf{PC} = x)$$

が不成立のものがいないかを確認。

「ハマリ」状態の検出 エンディングのブロックを end とおいたとき、

$$s_0 \models \mathbf{AG} \mathbf{EF}(\mathbf{PC} = \text{end})$$

が不成立ではないかを確認。

ストーリーの依存関係の矛盾 X というブロックが実行されたら Y というブロックが実行されないといけない場合、

$$s_0 \models \mathbf{AG}((\mathbf{PC} = X) \rightarrow \mathbf{AF}(\mathbf{PC} = Y))$$

が不成立ではないかを確認。

^{*1} 実際には not などを組み合わせることにより $\mathbf{EX}$, $\mathbf{EU}$, $\mathbf{EG}$ の 3 つの時相演算子で表現可能である

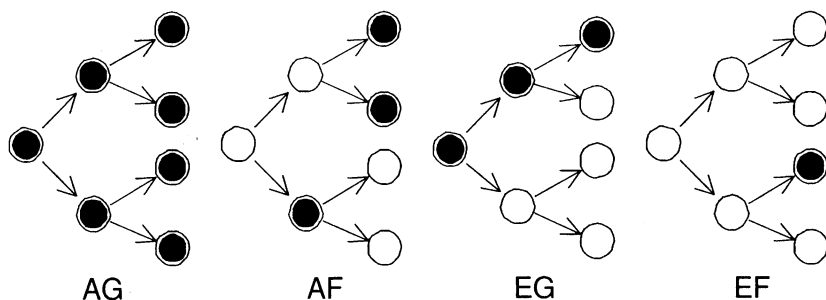


図 4: CTL の演算子

各種 invariant $s_0 \models \text{AG}(\text{常時成立してほしい式})$

4 実験

ここで、モデル検査理論をゲームシナリオに適用可能かを調べるために、実際に市販されているゲームのシナリオに対して検証を行った。

4.1 検証対象

種類 PC 向けのデジタルノベルタイプのアドベンチャーゲーム。選択肢の数や分岐の複雑さなどは少ないほうに分類される。

サイズ シナリオスクリプトの総容量で 6.6MBytes。うち、台詞などの純粋なテキストデータは 2.2MBytes。行数は約 20 万行。

ゲーム内の選択肢数 130 個

4.2 実験手法

実験の手順は以下の通り。

- 1) シナリオスクリプトを parse し、Control Flow Graph (CFG) を作成する。
- 2) CFG を静的に解析し、自明な部分の簡約を行う。
- 3) CFG に対して live variable 解析を行い、死んでいる変数の状態を追わないように CFG を書き換える。
- 4) 全てのブロックに到達可能かを検証する。
- 5) 求めた到達可能集合を用いつつ、「ハマリ」状態が存在しないことを検証する。

4.2.1 CFG の簡約

CFG の簡約の詳細は以下の通りである。

- 1) CFG に $A \rightarrow B$ という遷移があり、A から出て行く辺が他になく、B に入る辺も他にない場合、A と B を統合する。
- 2) 何も変数を触っていないブロック A に関して
 - (a) $B \rightarrow A$ という遷移があり、A に入る辺が

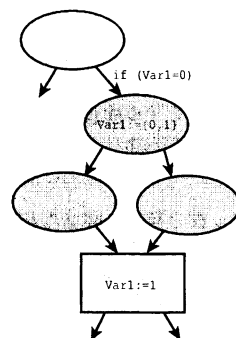


図 5: live variable 解析による最適化

他にない場合、A を B を統合し、A から出て行く辺を全て B につなぎ変える。

- (b) $A \rightarrow B$ という遷移があり、A から出る辺が他にない場合、A を B を統合し、A に入る辺を全て B につなぎ変える。

CFG の簡約によって PC に関する条件が縮退してしまうために、到達性検査では到達不能なノードがあっても検出できない可能性が生まれるが、「ハマリ」状態の検出は健全かつ完全のままである。

4.2.2 live variable 解析

生の CFG をそのまま解析に用いると死んでいる変数の状態まで BDD 内に保存されてしまうため、非効率となる。そこで、CFG の live variable 解析を行い、ある変数について生きているノードから死んでいるノードへの遷移があった場合に、死んでいるノードで必ず don't care 値を代入するように CFG を書き換える (図 5)。

4.2.3 記号モデル検査

BDD ライブラリとして BuDDy を用いつつ、Java にて記号モデル検査部分を作成した。到達性検証のアルゴリズムは標準的な到達可能集合を求め

るアルゴリズムを用いた。一方、

$$s_0 \models \text{AGEF}(\text{PC} = \text{end})$$

による「ハマリ」状態の検証に関しては、**AGEF**の計算速度を上げるために、あらかじめ求めた到達可能集合で探索空間を押さえる最適化テクニックを用いた [Clarke 99].

4.3 実験結果

実験は CPU: Pentium III 1.13GHz-M, RAM: 640MBytes の WindowsXP マシン上で行った。

4.3.1 CFG の生成

分析した結果のうち、前節のアルゴリズムをそれ以上簡約できなくなるまで摘要した CFG を Graphviz[Koutsofios 96] にて可視化したものを図 6 に示す。

CFG のノード数は、簡約前は 2503 個、簡約後は 165 個であった。

また、Control Flow に影響を与える変数の数は 76 個であった。それを取りうる値の個数で分類したのが以下の表である。今回調査したスクリプトでは変数への代入は整数型の即値代入しか存在していない。また、配列に対する動的なインデックスでのアクセスもなかった。

表 1: 取りうる値の数で分類した変数の個数

取りうる値	変数の数
2	69
3	3
4	2
5	2

この状態遷移系のとりうる状態数を初期状態からの到達性を考慮せずに計算すると、CFG 上での位置と各変数の値域の積となるので

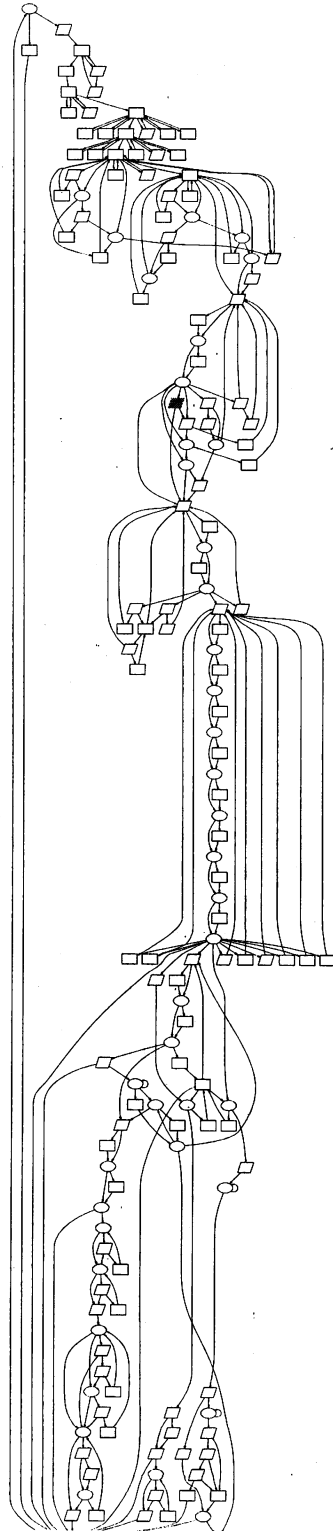
$$165 \times 2^{69} \times 3^3 \times 4^2 \times 5^2 \doteq 10^{27}$$

となる。

4.3.2 モデル検査

到達性検証は 237 秒で、また、それにプラス 45 秒で「ハマリ」状態検証が終了した。

その結果、図 6 で色が違うノードが到達不能ノードとして検出された。右下のノードはいわゆる sink ノードであり、CFG のエラー状態を示すものである。到達不能であることは当然である。一方、中央左のノードは対応するスクリプトファイルを見ても明らかにコンテンツのある部分であり、ゲーム製作者の意図していないシナリオ上の不具合を発見したと言える。



一方、「ハマリ」状態に関しては、検証により存在しないことが保証された。

なお、この合計計算時間約5分というスピードは、上記手順で述べた高速化のためのテクニックを全て適用した結果の数値であり、どれか一つでもかけると実用的な時間では計算が終了しなくなることには注意が必要である。

5 今後の課題

5.1 スケーラビリティの確保

今回分析に用いたゲームタイトルはアドベンチャーゲームやRPGと呼ばれるジャンルのゲームの中ではかなり単純なシナリオ構造をしている部類である。ユーザからの行動入力を毎回受け付けるようなゲームの場合は、CFGのエッジがもっと密になり、ノードの絶対数も大幅に増えることは確実にある。

しかしながら、今回のCFGのグラフを見る限りにおいては、シナリオがもっと大規模化していった際にも、CFGの幅は一定のサイズで収まりながら、縦に長くなっていくのみなのではないかと予想される。幅が狭いことをうまく生かすアルゴリズムを考えることにより、スケーラビリティを確保できると考えられる。

6 まとめ

本稿では、ゲームソフトにおけるシナリオ記述用の言語の現状とその問題点を説明し、その解決策としてモデル検査理論に基づいた自動検証機能を提案した。また、実際に実験として、市販PCゲームのシナリオスクリプトを検証し、20万行ものスクリプトを5分という実用的な時間で検証した。これは、今回のターゲットが比較的単純なシナリオ構造を持つものであったためでもあるが、他の多くのインタラクティブなストーリーを持つゲームでも適用可能であると考えられる。ゲームシナリオの自動検証をモデル検査理論の実用に直接結びつく応用のひとつとして確立していきたい。

将来的にもっと切実な需要として、ネットワークゲームのスクリプトの検証もあると考えられる。複数のプレイヤーが同時に仮想世界に対して行動を起こし、それをまとめた上で結果を返さねばならないネットワークゲームにおいては、シナリオ記述言語は並列プログラムとなる。本稿で説明したような性質をもつシナリオスクリプトがさらに並列になった場合、人力でのデバッグに限界があることは明らかであろう。さらに、ネットワークゲームにおいてはライブ性を持たせるために短期間のサイクルでのシナリオの更新作業が必要となり、デバッグにかける

れる時間はますます短くなる傾向にある。

ゲーム機の性能向上は目覚ましい勢いで続いている。そして、ネットワークという新しい要素も否応なしに浸透しつつある。自動検証に限らず、ソフトウェア技術による開発のサポートを行うという発想はこれから重要になることは間違いない。アカデミックな情報科学的知見によるアプローチをゲーム開発に導入していくことは諸外国に押されがちな日本の開発力を保持し、より高めていくためには必須となってくるであろう。

参考文献

- [Bryant 86] Bryant, R. E.: Graph-Based Algorithms for Boolean Function Manipulation, IEEE Trans. Comput., Vol. C-35, No. 8, pp. 677-691 (1986)
- [Clarke 94] Clarke, E. M., Grumberg, O., and Long, D. E.: Model Checking and Abstraction, ACM Transactions on Programming Languages and Systems, Vol. 16, No. 5, pp. 1512-1542 (1994)
- [Clarke 99] Clarke, E. M., Grumberg, O., and Peled, D. A.: Model Checking, The MIT Press, Cambridge, Massachusetts (1999)
- [J.R. Burch 90] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang.: Symbolic Model Checking: 10^{20} States and Beyond, in Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science, pp. 1-33, Washington, D.C. (1990), IEEE Computer Society Press
- [Koutsosifos 96] Koutsosifos, E. and North, S. C.: Drawing graphs with dot, Murray Hill, NJ (1996)
- [柴崎 90] 柴崎 雅史: 多人数ゲームシナリオの記述と検証法, 情報処理学会研究報告マルチメディア通信と分散処理, Vol. 47, No. 5 (1990)