# Analyzing Computer Game Narratives

Clark Verbrugge and Peng Zhang*

School of Computer Science, McGill University
Montréal, Québec, Canada, H3A 2A7
{clump,pzhang13}@cs.mcgill.ca

**Abstract.** In many computer games narrative is a core component with the game centering on an unfolding, interactive storyline which both motivates and is driven by the game-play. Analyzing narratives to ensure good properties is thus important, but scalability remains a barrier to practical use. Here we develop a formal analysis system for interactive fiction narratives. Our approach is based on a relatively high-level game language, and borrows analysis techniques from compiler optimization to improve performance. We demonstrate our system on a variety of non-trivial narratives analyzing a basic reachability problem, the path to win the game. We are able to analyze narratives orders of magnitude larger than the previous state-of-the-art based on lower-level representations. This level of performance allows for verification of narrative properties at practical scales.

**Key words:** game narrative,verification,optimization,performance analysis

## 1 Introduction

Narrative is a central feature of many computer games, extending from text-based interactive fiction to current adventure, RPG, and mixed genres. For such games a number of narrative properties become important to an immersive game experience, including logical consistency, continuity of story elements [1], level of "tension" or atmosphere [2], as well as game-play issues such as ensuring player progress, and adequate coverage of potential player choices [3, 4]. The complexity of larger narratives, however, can make formal analysis of narratives difficult. For analysis of long game segments or existing, complete narratives the combinatorial explosion implied by a rich set of player choices and game content elements results in scaling issues, limiting conservative verification of narrative properties.

In this work we develop an efficient system for analyzing complex game narratives. Our approach is based on an exhaustive analysis of the narrative state-space, and thus applies to general reachability problems. To improve scalability, we extract high-level game information from game code by applying program analysis techniques more traditionally found in the compiler optimization domain. This information is used within an optimized search to reduce the branching factor and improve performance.

We illustrate our design by examining a fundamental example of reachability, the path to win the game. In this context we analyze a variety of non-trivial narratives

* Now at Microsoft: pezhang@microsoft.com.

derived from both commercial and amateur interactive fiction games. Using our optimized state search we are able to determine winnability within seconds to minutes on a modern machine. These results dramatically improve on previous, low-level work on formal winnability analysis, which has been limited to narrative inputs orders of magnitude smaller [5]. The ability to perform reachability analyses on narratives of much larger scales shows that formal verificaton of a wide variety of properties can be feasibly performed on non-trivial, industrial-size game narratives.

Contributions of our work include the design of a complete system for reachability analysis of computer game narratives, an optimized search based on a novel dataflow analysis, and non-trivial experimental investigation of basic winnability. In the sections that follow we give related work and contextual background for our research, followed by our system design and results.

## 2 Related Work

Our approach to game narratives aims at analysis/verification in order to improve design and thus player experience. A number of other works have also identified and formalized narrative properties that players find pleasing, disruptive, or which may improve story generation. Adams, for instance, proposed basic guidelines for industry developers in a series of online articles [3], where among many other properties the problem of narrative game-play being allowed past the point of winnability was presented; Verbrugge formalized this notion as "pointlessness." [4]. Barros and Musse proposed a model designed to ensure pace/tension [2], and Nelson et al. defined properties related to the spatial locality of actions, as well as motivational or logical continuity of events [1]. Logic-based approaches are popular in this context, and have been applied to verify the time-line of a non-linear story [6], and ensure logical correctness and continuity of events [7]. Petri net models have also been proposed and used to represent narratives for both story construction and analysis [4, 8–10]. Our work here builds on previous experience in winnability analysis, applying a generic SAT-based solver to a Petri net model of game narratives [5].

In this work we *post-facto* analyze narratives manually developed as text-based, interactive fiction adventures. Although text-based IF games are an older genre, they have had as a whole a strong influence on more modern interactive fiction, adventure, and RPG game design. An active community continues to exist as well, and a number of IF authoring kits are available as products on their own [11, 12] or in terms of extensions built on older commercial offerings [13]. To limit technical complexity in isolating narrative structure, we have based our design on the PNFG language for interactive fiction [5].

## 3 Narrative Model

Analysis of game narratives assumes of course a suitable representation. As mentioned in the previous section, we make use of a language for constructing interactive fiction (IF) games, *PNFG* [5]. The IF genre has the advantage of consisting of complex narratives, relatively easily extracted from the game architecture. The PNFG format is

preferred over more full-featured languages such as Inform [13] as a simplified, but complete IF language with a well-defined semantics and reduced syntax.

**IF Properties.** An interactive fiction narrative provides a minimal, typically text-based virtual environment. A player avatar is controlled through textual input forming game commands, and the current or resulting game state is reported by textual output. Game-play is turn-based, and usually involves exploration of an interconnected series of rooms, wherein the player (avatar) may examine, move, and otherwise manipulate game objects. Appropriate actions unlock or control narrative progress, moving the player from an initial state to either a winning or losing conclusion.

Basic game control flow is straightforward. After initialization, the game waits in an idle state for user input; user commands trigger game actions, which can result in either a game win, loss, or (more typically) a return to the idle state following any post-turn processing. The complexity of game state is an important property to analysis. In IF games critical game state consists of simple object properties, such as a room being lit or unlit, as well as the object containment hierarchy—the location of each object, including the player's inventory. Counters and dynamic object allocation can add further complexity, although games are usually finitely bounded, with a small limits on counters and a fixed maximum number of available game objects.

**PNFG.** The PNFG language provides a minimal model of IF game structure. Game objects and rooms are defined along with boolean state variables, and containment is internally represented by further boolean state ($x$ (not) in $y$ for all objects $x$ and locations $y$). User commands invoke code which can set and unset object states, move objects between locations, as well as branch conditionally on object state or location. Syntatic sugar is provided for a number of further common IF constructs, including finite counters, scoping of game commands; more language details can be found in [5].

## 4 Narrative Analysis

Our overall approach to narrative analysis is illustrated in Figure 1. PNFG game specifications are subject to initial, high-level analysis, which is then used in conjunction with the game's interpretable (compiled "NFG") form as part of an optimized search of the game state space. The subsections below describe the basic search behaviour and our optimization.

As an example of useful narrative analysis, and to give our study a specific analysis goal, we investigate the general problem of computing "winnability." This involves identifying the sequence of commands that bring a player from an initial state to a winning game completion. As well as verifying a fundamental game property (the game should be winnable), finding one or all "winning paths" is an instance of the more general problem of efficiently determining state reachability, and our techniques can be easily abstracted from the winnability goal and applied to other search problems.

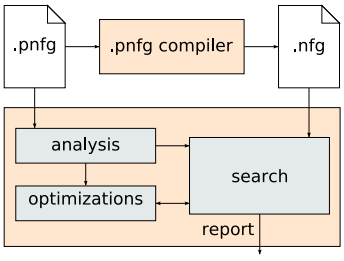



**Fig. 1.** *Overall system design.*

### 4.1 Basic search

The basic state-space search proceeds as a back-tracking, depth-first search of reachable game states, applying all possible game commands at each state. Game state is internally stored in terms of the state of the internal NFG representation, but is not directly accessed by the search system. Reachability and progress are instead determined by the ability of a game interpreter incorporated into the system to apply a given command, monitoring for win, lose or error conditions.

A number of generic optimizations improve performance of this naïve, brute-force approach. Cycle detection is an effective tool for most narratives. Many games, for example, tend to permit a variety of commands such as "look," "examine," or command sequences such as "take x, drop x" that do not overall modify game state, but nevertheless contribute to the branching factor and lengthen search paths. Our system thus maintains a stack of states during its DFS activity, and truncates searches containing states previously encountered in the current game search path. Caching is also used across search branches in order to avoid "dead-ends," or states which have no legal actions that can lead to a game win (i.e., all actions lead to either losing, errors, or dead-ends). These states are cached, and used to further prune the state space search.

Although these optimizations are effective, larger narratives have many commands and objects as well as potentially long solution depths, and for practical analysis a very small branching factor and/or good search heuristics are required. Our approach applies dataflow analysis techniques more typically used in compiler optimization to extract game information in order to further improve performance. Below we describe our primary techniques for optimizing the search process.

### 4.2 Pre/Post-condition Analysis

To reduce back-tracking in the search it helps to know ahead of time which commands may follow each other. Even if illegal combinations are quickly pruned during search the large branching factor of a naïve search limits scalability.

In the PNFG language, many of the operators that modify game state, either by changing boolean variables or by moving an object from one location to another, assume a specific input state. For example, the command to change a game object variable from false to true (`+object.var`) requires the variable be initially false, and similarly, the object move statement (`move x from y to z`) assumes the object `x` is currently in `y`. Note that successful execution also guarantees the resulting output state: in the first example `object.var` is certainly true if the statement completes without error, and in the second `x` will be in `z`.

These observations drive two basic analyses which we then combine in order to prune the branching factor. We first analyze each action in a *backward* direction, computing a conservative approximation of the minimal necessary conditions for the action to execute correctly. We then perform a very similar analysis in a *forward* direction, conservatively computing the output conditions. In order for one action to follow another then, output of the first must be compatible with the input requirements of the second.

In a formal sense these form symmetric dataflow problems. If we consider our forward, *post-condition* analysis, we associate with each object variable and $x, y$ object location state ($x$ in $y$) a value from the domain $\{$true,false, $\top\}$; *i.e.*, the (incomparable) elements true and false, along with a greatest element $\top$, representing a state which may be either true or false.

The dataflow technique propagates these pairings through all code paths starting from the action code entry (or exit) with all variables in unknown/inconsistent states ($\top$), modifying the associated domain values according to the game action. The statement `+object.var`, for example, will in a forward sense result in an output pairing `object.var:true`, and in a backward sense the pairing `object.var:false`. At conditionals information is duplicated along each branch, and at join points merged by setting `object.state:`$\top$ whenever `object.var` is not the same on both sides.

A short example of the application of post-condition analysis is shown in Figure 2. In this case post-condition analysis is reasonably effective, determining the location and the state of the candle, although not the state of the player. Pre-condition analysis is symmetric, performed in the reverse direction.

```
(you,take,candle) {                  candle-in-box:⊤, candle-in-you:⊤, candle.lit:⊤, you.hurt:⊤
  move candle from box to you;       candle-in-box:false, candle-in-you:true, candle.lit:⊤, you.hurt:⊤
  if (!candle.lit) {                 candle-in-box:false, candle-in-you:true, candle.lit:false, you.hurt:⊤
    +candle.lit;                     candle-in-box:false, candle-in-you:true, candle.lit:true, you.hurt:⊤
  } else {                           candle-in-box:false, candle-in-you:true, candle.lit:true, you.hurt:⊤
     +you.hurt;                      candle-in-box:false, candle-in-you:true, candle.lit:true, you.hurt:true
  }}                                 candle-in-box:false, candle-in-you:true, candle.lit:true, you.hurt:⊤
```

**Fig. 2.** An example of post-condition analysis: dataflow information (on the right) is propagated in a forward direction.

Upon completion of this analysis, each action has calculated pre and post-conditions. A variable with a post-condition of true is necessarily true on action exit, and a variable with a pre-condition of true must be true on input if the action is to execute correctly; symmetric for false of course. For each action pair compatibility can thus be easily tested: if the post-conditions of action $A$ include `object.var:`$a$ and pre-conditions of $B$ have `object.var:`$b$ for a given object variable, then action $B$ can follow action $A$ as long as $a \sqsubseteq b$ or $b \sqsubseteq a$. Actions available at each point in the state search are thus selected according to these restrictions.

## 5 Experimental Analysis

We have implemented our system and examined the effects of our optimized search on a variety of moderate size narratives, roughly the size of a commercial game chapter. In each case we measure the performance of the analysis system as it is applied to a basic winning path problem. Below we describe our benchmark suite, discuss our measurement strategy, and present experimental results and accompanying observations.

### 5.1 Benchmarks

Benchmarks are drawn from two main sources: the well-known commercial game, *Return to Zork*, where we have modeled the first two chapters in PNFG, and several new narratives developed directly in PNFG by undergraduate and graduate stu-

dents as part of a course assignment. The latter represent amateur efforts, but were required to respect some basic complexity measures, including lower bounds on number of objects, commands, solution complexity and length. Table 1 summarizes interesting, static properties of our benchmarks. Naïvely, the state graph to be searched will have a branching factor given by the number of commands (global and room-specific), and a depth at least of the solution depth. Assuming any object can be in any room and any room can be in any other room, the overall state space is bounded in size by $|\text{Rooms}|^{|\text{Objs}|} \times (|\text{Rooms}| - 1)^{|\text{Rooms}|} \times 2^{|\text{Vars}|}$.

| Benchmark | LoC | Rooms | Objs | Vars | Global Cmds | Room Cmds | Depth |
|---|---|---|---|---|---|---|---|
| ageorg15 | 1230 | 9 | 16 | 33 | 15 | 45 | 17 |
| dprykh | 1472 | 8 | 11 | 23 | 40 | 29 | 13 |
| hsafad | 387 | 10 | 14 | 18 | 2 | 39 | 12 |
| mcheva | 775 | 14 | 10 | 6 | 0 | 62 | 26 |
| RTZ Chap 1 | 583 | 11 | 20 | 10 | 5 | 42 | 11 |
| RTZ Chap 2 | 1113 | 22 | 37 | 16 | 4 | 118 | 19 |

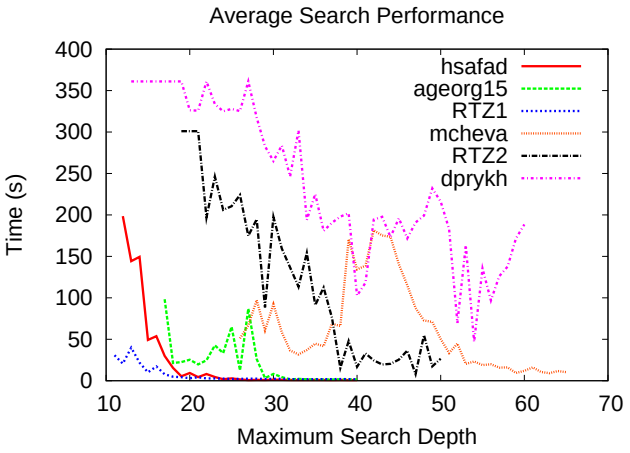
**Table 1.** Benchmarks and properties: lines of code, total number of rooms, objects, boolean object variables, commands (global and room-local), and minimum solution depth.

### 5.2 Measurements

We measure performance of winnability analysis on our benchmarks using our optimized analysis. Measurement of winnability analysis is complicated by several factors. An exhaustive test, reporting every winning path up to a given depth would, for instance, provide a deterministic workload. Complete enumeration of winning solutions, however, is not practical—the number of possible solutions grows very quickly as search depth increases, resulting in performance being quickly dominated more by reporting time than analysis. We avoid this problem at the cost of greater variance by measuring only the time to find the *first* winning solution. In order to avoid skew introduced by the order in which nodes are examined in the search, we also randomize the order of actions considered as each node is expanded in the search.

Timing results are shown in the graph on the left in Figure 3. For each benchmark and for a range of maximum search depths, a series of 10 analysis attempts are performed and the average time plotted. Note that while general trends are stable variance can be significant, as a fortunate or unfortunate node ordering during search can have a large impact on an individual experiment. All results were gathered on the same quad-core Xeon 2.3GHz machine with 16GB RAM, Debian 2.6.18, using Java HotSpot 1.5.0_14 and a 1.5GB heap, and were limited to 5 minutes per search (except for DPRYKH at 6 minutes). It is important to note that these results represent orders of magnitude improvement over previous work in this area using a generic, SAT-based solver, which was not able to complete given several *days* of execution time for RTZ CHAPTER 1, our smallest (in terms of solution depth) benchmark [5].

**Impact of optimization** Our optimized search process has varying performance, but has overall impressive efficiency, finding solutions within our time bounds for all of




Average Branching Factor

| | Naïve | Opt |
|---|---|---|
| AGEORG15 | 60 | 23 |
| DPRYKH | 69 | 44 |
| HSAFAD | 41 | 8 |
| MCHEVA | 62 | 5 |
| RTZ CH. 1 | 47 | 10 |
| RTZ CH. 2 | 122 | 10 |

**Fig. 3.** Search performance on benchmarks.

our benchmarks at most search depths. This is particularly evident at larger depths. The DPRYKH benchmark poses the most significant challenge for our analysis, mainly due to the high number of player commands that remain even in the optimized search cases, and hence large branching factor. This can also be seen in the table on the right of Figure 3, which shows the average branching factor for naïve and optimized searches. Optimization is able to reduce this dramatically for most benchmarks, but for DPRYKH the use of many global actions (see Table 1) limits the effectiveness.

**Inherent winnability** A striking, and initially counter-intuitive feature of all our data sets is the way search times tend to decrease as maximum search depth is increased. Since the state space is exponential in the search depth, our expectation was that increasing depth would uniformly increase search time. The data in Figure 3 show this is not the case, and we may even conclude that merely increasing search depth is a feasible search optimization strategy, if minimal solutions are not required.

This behaviour can be explained in relation to a few main factors. In a general sense finding a *minimum* length solution is a harder problem than just finding *any* solution. Minimum length solutions tend to be few, and the number of solution permutations grows very quickly if lattitude is given to contain extra sub-optimal actions or orderings. This may increase the relative solution density as depth grows, and patterns in solution densities can cause periodicity in the search performance as well, partially explaining the less monotonic search performance of MCHEVA and AGEORG15. For smaller narratives, such as RTZ CHAPTER 1, our optimized search may also have reduced the state space to the point where it could be feasibly exhausted. Although this is unlikely in general, and certainly not true for larger games, we note that most IF games are both finite and designed to be eventually won. Narrative games are often time-intensive and have relatively low replayability—good game design avoids the need for the player to save and reload earlier states, and thus a player who plays the game without repetition and without losing should always be able to eventually win.

## 6 Conclusions and Future Work

Narrative analysis has many potential applications. Here we have investigated and shown the practical feasibility of a basic reachability problem, but pre/post-condition analysis is relatively independent of goal and our design could be easily extended to apply to a variety of interesting narrative verification questions. Properties more generically depending on game paths or state relations, such as "pointlessness" [4], or constraints on spatial locality [1] are search-related problems, and could be examined using our techniques. Further application is found in determining the path to more immediate game goals (such as how to open the next door), where search speed improvements are essential for developing online and interactive but generic and automated game hint systems.

**Acknowledgements** This research was supported by the Natural Sciences and Engineering Research Council of Canada.

## References

1. Nelson, M.J., Mateas, M., Roberts, D.L., Jr., C.L.I.: Declarative optimization-based drama management in interactive fiction. IEEE Computer Graphics and Appl. **26**(3) (2006) 32–41
2. Barros, L.M., Musse, S.R.: Towards consistency in interactive storytelling: Tension arcs and dead-ends. Comput. Entertain. **6**(3) (2008) 1–17
3. Adams, E.: The designer's notebook: Bad game designer, no twinkie! parts I–VI. `http://www.gamasutra.com` (1998–2005)
4. Verbrugge, C.: A structure for modern computer narratives. In: CG'2002: International Conference on Computers and Games. Volume 2883 of LNCS. (July 2002) 308–325
5. Pickett, C.J.F., Verbrugge, C., Martineau, F.: (P)NFG: A language and runtime system for structured computer narratives. In: GameOn'NA: Proceedings of the 1st Annual North American Game-On Conference, EUROSIS (August 2005) 23–32
6. Burg, J., Boyle, A., Lang, S.D.: Using constraint logic programming to analyze the chronology in "A rose for Emily". Computers and the Humanities **34**(4) (December 2000) 377–392
7. Lindley, C.A., Eladhari, M.: Causal normalisation: A methodology for coherent story logic design in computer role-playing games. In: CG'2002: International Conference on Computers and Games. Volume 2883 of LNCS. (July 2002) 292–307
8. Natkin, S., Vega, L.: A Petri net model for computer games analysis. International Journal of Intelligent Games & Simulation **3**(1) (March 2004) 37–44
9. Brom, C., Sisler, V., Holan, T.: Story manager in 'Europe 2045' uses Petri nets. In: International Conference on Virtual Storytelling. Volume 4871 of LNCS., Springer (2007) 38–50
10. Araújo, M., Roque, L.: Modeling games with Petri nets. In: DiGRA Conference, London, Brunel University (September 2009)
11. Tessman, K.: The Hugo Book—Hugo: An Interactive Fiction Design System. 1st edn. The General Coffee Company Film Productions, Toronto, Canada (2004)
12. Wild, C.: ADRIFT: Adventure Development & Runner—Interactive Fiction Toolkit, version 4.0 manual. (2003) `http://www.adrift.org.uk`.
13. Nelson, G., Seebach, P., Firth, R., Plotkin, A., Short, E.: Inform. `http://www.inform-fiction.org/` (1993)