Proof-Theoretic Study of Game Mechanics

Chris Martens
Carnegie Mellon University
Pittsburgh, PA
cmartens@cs.cmu.edu

ABSTRACT

We describe the use of proof-theoretic techniques for the analysis of games and the design of a programming language aimed at game design prototyping.

1. INTRODUCTION

My Ph.D. research focuses on the use of *proof-theoretic techniques* to create models of game mechanics that can be executed and analyzed.

Proof theory is a tool for the design of logics. For a given logic, one inspects the structure of proofs that one can form according to its rules and asks questions about global properties and structure. One may ask how proofs can be reduced while preserving certain properties and which proofs are considered equal. One may also view proofs as *programs*, and this discovery has led to greater understanding and informed design of programming languages. My work aims to connect these ideas specifically to *programming languages for game modeling*, viewing game rules themselves as rules embedded in a logic.

My goal is to study the computational processes involved in play of various kinds, especially focused on procedural generation and emergent narrative through multi-agent systems. To this end, my thesis project extends a proof-theoretic formal modeling system based on *linear logic programming* with as few additional language constructs as possible to express a wide range of gameplay idioms and interactions. Execution traces correspond to proofs, and their structure records which actions depend on the consequences of other actions, yielding not (necessarily) a linear sequence of program events but a partially-ordered DAG (directed acyclic graph) of such events. For this reason, the programming model can support specifications of *concurrent* actions among multiple game entities or agents. In the long run, such a language might be used as an intermediate language for a game-specific, user-facing game creation tool.

The research questions involved in this project include the following:

Proceedings of the 10th International Conference on the Foundations of Digital Games (FDG 2015), June 22-25, 2015, Pacific Grove, CA, USA.
ISBN 978-0-9913982-4-9.

1. Which game designs contain interesting concurrent structure? Can a nondeterministic concurrent specification straightforwardly be given meaning as a multiplayer game? How does the existence of multiple agents complicate the structure of branching narratives?

2. What feedback loops and other structural properties exist in procedural game mechanics, especially those involving resource management and open-world experimentation?

3. How can we classify, for the sake of eventually minimizing, *bugs* in emergent simulations? Can we model check or otherwise statically analyze the state space for certain game models?

4. What kinds of software tools based on these ideas can help game designers achieve useful digital prototypes?

2. BACKGROUND

Logic programming, also known as relational programming, allows an author (or game designer) to specify a collection of terms and predicates that maay relate them, then to write rules in the form of *logical implication* between predicates. A proof-theoretic logic programming language gives meaning to *program execution* as *proof search* in a given logic – each step the program takes as it runs corresponds to some valid and immediate inference step in the logic. In *linear logic programming*, inference steps may *exchange* some facts for others, rather than strictly increasing the body of known facts. For this reason, it is well-suited to model complex evolving states (as described through game rules) and study their dynamics on the basis of proof structure.

Consider as an example a game of social interaction such as Prom Week [3] wherein agents have internal state representing their sentiments toward other characters, and the player-controlled actions in the game both affect and are affected by those sentiments. One example might be a simplified rule for bullying described as:

```
bully (A, B):
    if: affection(A,B) < 0 and admiration(A,B) < -1
    then: decrement affection(B,A)
```

This rule schema may be instantiated for any pair of characters `A` and `B`, and the player may choose any pair for which the preconditions hold. Thus the story is formed by player choices between rules like this one that may be instantiated with any set of applicable characters. In proof search, that process corresponds to choosing a term to substitute into a

universal quantification and then eliminating the resulting implication by replacing its premises with its conclusion.

In traditional programming languages, modeling this sort of rule would involve crafting several data structures to model the relationships between characters and the evolution of those relationships, as well as complex control structures for checking the application of rules. In a logic programming language, all of that machinery comes for free. Such rules are easily modeled in linear logic and, using language extensions in my thesis, can scale to the multi-stage process of computing character AI behavior in systems such as Prom Week's.

The logic programming approach has a lot in common with planners such as STRIPS (note that the above is effectively a STRIPS rule), which have been used extensively in interactive story modeling. [6, 4] By contrast, linear logic programs have an execution semantics such that we can run the search mechanism as *itself* a game execution engine, rather than using a planner as intermediate process within the event loop of a running game.

This work may impact several games-related bodies of study. First, the need for general game description languages capable of describing digital games has been well articulated [1]. Linear logic could be a good candidate due to its generality; nothing in it is specialized to any specific kind of game logic, but it is flexible enough to express idioms from several. The implications of a general enough executable game description language include the ability to compare the core systems within different games side-by-side for the sake of analysis, as well as rapid prototyping and easy experimentation with mechanics for game designers.

My work aims to unify certain computational tools employed in formal methods and programming languages research with the creation and study of games. In this work, I think of games as systems like any other with the distinguishing property that program evolution may be influenced by more than one process (at least one computational agent and one human), and in this sense I do not really distinguish games from arbitrary concurrent programs except at an informal level.

3. METHOD AND CURRENT STATUS

My thesis project consists of three main components:

1. Using an existing linear logic programming language, Celf, as a narrative generation engine by describing a story world as a linear logic program [2]. This work shows how linear logic can be use to depict interactions between subsets of agents, and the resulting proof corresponding to a story contains concurrent structure embodying independent courses of narrative events. This structure can be visualized to reveal narrative flow and feedback loops between groups of narrative actions, enabling some answers to research questions 1 and 2 from Section 1.

2. Designing and implementing the language Ceptre, a language based on a subset of Celf and extended with constructs for interaction and control flow.[1] Along the way I have developed numerous examples of game mechanic design in this language, showing how to extend the construct used in narrative descriptions to also depict basic movement and puzzle logics. The development of a progrmaming language allows one to easily mock up a game simply by describing its rules, rather than by first requiring the designer to describe a mapping from internal game state to visual or physical components. The ability to mock up a set of rules in an executable language allows for answers to questions 2 and 3 by way of playtesting and post-playtest proof term analysis.

3. Designing a meta-language for describing Ceptre program invariants and an algorithm for automatically checking them. This part of the project aims to address research questions 2 and 3 by allowing a game designer to state design intents as formal, checkable program invariants.

I have currently completed part (1) of my thesis and have made considerable progress on parts (2) and (3): I have written a draft paper on the design of Ceptre and implementation is in progress.[2]

I am designing a meta-language for describing contract-like properties of linear logic program stages (pre-conditions, post-conditions, and invariants) that aims to let a designer express her intent in a way that can be programatically checked.

4. FUTURE PLANS

In the immediate future (i.e. for my thesis) I intend to complete the proof-of-concept implementation of Ceptre and the proofs for the invariant language. In the longer term, I hope to continue investigating logical formalisms based on proof theory (including and outside of linear logic) as tools for expressing playful rule systems. For example, one related project I would like to pursue is a mixed-initiative game design tool wherein a human and computational agent take turns proposing rules and exploring their consequences. This project would involve as a substantial component the formalization of a large library of game mechanics in such a way that a user could combine them and edit them off-the-shelf.

I would also like to more deeply investigate the implications of concurrent structure in simulations and narrative timelines, particularly as they pertain to narrative focalization (an agent's point-of-view). Justus and Young's work [5] on rearranging narratives as long as what is observable to a player stays consistent could be a form of proof equivalence within an epistemic modal logic, for instance.

5. REFERENCES

[1] M. Ebner, J. Levine, S. M. Lucas, T. Schaul, T. Thompson, and J. Togelius. Towards a video game description language. *Artificial and Computational Intelligence in Games*, 6:85–100, 2013.

[2] C. Martens, J. F. Ferreira, and A.-G. Bosser. Generative story worlds as linear logic programs. In *Intelligent Narrative Technologies*, 2014.

[1] Draft available at `http://www.cs.cmu.edu/~cmartens/ceptre.pdf`

[2] See `http://www.github.com/chrisamaphone/interactive-lp`

[3] J. McCoy, M. Treanor, B. Samuel, M. Mateas, and N. Wardrip-Fruin. Prom week: social physics as gameplay. In *Proceedings of the 6th International Conference on Foundations of Digital Games*, pages 319–321. ACM, 2011.

[4] J. Porteous, M. Cavazza, and F. Charles. Applying planning to interactive storytelling: Narrative control using state constraints. *ACM Trans. Intell. Syst. Technol.*, 1(2):10:1–10:21, Dec. 2010.

[5] J. Robertson and R. M. Young. Modelling character knowledge in plan-based interactive narrative to extend accomodative mediation. In *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2013.

[6] R. M. Young. Notes on the use of plan structures in the creation of interactive plot. In *Narrative Intelligence: Papers from the AAAI Fall Symposium*. AAAI Press, 1999.