Chapter 7

Conclusion

This dissertation has described the use of linear logic for representing game mechanics and narrative systems, unifying the ideas of play and narrative through the waypoint of logical proof. In particular:

- Chapter 2 described how to model narratives in linear logic, bringing out the dual notions of *alternative* and *simultaneous* structure in nonlinear stories, each corresponding to derivability principles in linear logic.
- Chapter 3 showed how to use story worlds modeled in linear logic as *narrative generators* by mapping proof search onto narrative generation. We showed how proof terms generated by a linear logic programming language contain simultaneous narrative structure, and we provide two tools, a visualization and query tool *CelfToGraph*, and a playable story generator *Quiescent Theater*, for investigating that structure.
- Chapter 4 presented a new programming language based on linear logic programming that can be used to author complex interactive simulations. We introduce the concept of *stages*, which add the ability to program with quiescence, allowing the program to pass control of the state between different components.
- Chapter 5 presented five case studies of linear logic programming, showcasing its use for modeling standard video game idioms like combat and resource management, board games idioms (Settlers of Catan), emergent behavior, social story worlds, and participatory theater.
- Chapter 6 presented three candidate conceptual frameworks for reasoning about linear logic programs, including several example proofs carried out by hand. We gave a proof of decidability for a substantial fragment of the framework, building the initial pathways toward automated reasoning tools.

Recall the thesis statement from Chapter 1:

> **Thesis statement:** *Using linear logic to model interactive worlds enables rapid prototyping of experimental game designs and deeper understanding of narrative structure.*

We have supported this thesis in the following ways: Chapters 2 and 3 identify key aspects of narrative structure, alternatives and simultaneity, and show how linear logic proofs carry these structures intrinsically, providing *deeper understanding of narrative structure*. Chapters 4 and 5 shows how a concrete implementation of linear logic as a modeling language can be used to build interactive artifacts that support experimentation and iteration on the design of mechanics, supporting *rapid prototyping of experimental game designs*. Chapter 6 provides auxilliary support to the *rapid prototyping* part of the thesis statement, since simple invariants are often critical to program correctness. Having a framework of proof for properties of linear logic programs provides the basis for automatic tools that may help eliminate bugs.

Our findings constitute a foundation for unifying common formal aspects of digital games, analog games, participatory theater, and generative storytelling in a way that we believe furthers the prospects for creativity in all of these domains and their combinations.

7.1 Contributions

Because this thesis bridges two disciplines, logic/programming language theory and game/narrative design, we identify its core contributions in each of these two contexts.

7.1.1 Game and Narrative Design

For game and narrative design, our primary contribution has been that of establishing logical foundations for the field. We worked out the correspondence between proof and story in a way that enables programming with logical constructs to generate and analyze stories, and we showed how that same programming model could be used to describe game mechanics and interactive story worlds in a flexible and general way.

We believe that a logical foundation fills a gap in current game design tool theory. Current thought recognizes that people highly skilled in designing digital game mechanics need not also be highly skilled in programming in a general-purpose language. Thus, several tools have emerged as "end-user programming" languages for game designers—they tend to offer limited programming affordances (e.g. "attach a behavior to an object" or "link two passages of text") rather than general control constructs. But a common problem with these tools is that once the "end user" develops *mastery* with the limited affordances, they cannot grow their skill within the same framework: they need to break abstraction barriers and learn the more complex language underlying the tool.

A logical foundation for executable game descriptions addresses this problem: linear logic itself contains very few "features" to understand initially (effectively just $\otimes$ and $\multimap$), and Ceptre adds a few more (type declarations, stages, quiescence, and traces). But a *logic* as a formal structure is something that can be extended in a variety of ways to accommodate new domains and new representational concerns, while still being

grounded in fundamental principles like soundness and completeness to guide the design of extensions. Having those fundamental principles as an anchor for the tool design means that we can grow the language with a sound basis for determining when new features work seamlessly with old ones (for instance, generalizing linear logic to the first order case) and how to soundly incorporate more complex feature interactions (for instance, introducing the $\{-\}$ (lax) modality in Celf to allow programmer control of forward- and backward-chaining). The question of whether appropriate logical connectives will always be available for the most salient author-facing needs has yet to be fully investigated, but we believe there are several promising avenues of future work in this vein.

Secondarily, the particular choice of linear logic for formalizing game rules, forcing a resource-oriented viewpoint, has enabled the following observation about game design authoring affordances: in order for rules to create harmonious interaction, they must effectively come in pairs. That is, if a rule *produces* some fact p (by containing p in its conclusion), some other rule should *consume* p, and vice versa. While this may seem like a simple observation for linear logic programs, tools such as Kodu and Twine do not always provide the means for an author to understand or create this duality. For instance, in Kodu, an entity can be programmed to `move towards` some other entity, and an entity can be programmed to do something whenever it `bumps` another entity. This action-sensor pair, `move towards` and `bump`, work crucially by their interaction with one another: an entity that moves toward another entity will eventually bump into it. But that relationship is not codified by the tool in the same way that those mechanics would be specified in linear logic. In languages like GameMaker, Stencyl, and Unity, behaviors may be attached to game entities, but there is no visible means of determining whether any of the game's progression depends meaningly on those behaviors; similarly, one may write code that case-analyzes certain behaviors that do not actually arise. Noticing the ubiquity of this pattern in linear logic leads us to make a recommendation for game creation tool designers: for every behavior that an author can add to their game, ensure that programming facilities exist both for introducing that behavior and depending on its outcome.

7.1.2 Programming Languages and Logic

In the area of logic-based programming language theory and design, we have contributed a large body of examples and studies on its use for the specific domains presented here, building an argument for the language design methodology in general.

By developing a new linear logic language designed around interactivity, we have also contributed to a better understanding of the logic, its use as a programming language, and forward chaining proof construction. In particular, we have clarified the notion that programming with *quiescence* enables a broad range of idioms that were unavailable before, and that it can ultimately be accounted for by the semantics already available in Celf (as shown in Chapter 4).

Finally, we also contribute results about the metatheory of linear logic that are broadly applicable for reasoning about concurrent state change, not just in games. We have es-

tablished that the checking of *generative invariants* is decidable for propositional linear logic programs and first-order linear logic programs with finite term domains (Chapter 6), paving the way for development of practical automated checking tools.

7.2 Future Work

We propose several avenues for future work, divided into the subjects to which we hope to make further contributions: narrative representation and modeling; generative methods; accessible game design tools; and facilities for computer-aided reasoning about game and narrative specifications.

7.2.1 Narrative Representation

The primary notion in narratives that we feel bears further investigation from a logical standpoint is *knowledge*: our current representation of narrative states presents all information about the story world in one monolithic body of state (the linear context), failing to distinguish between information that is localized to particular agents and knowledge that is communicated between them. Further, in the *discourse* aspects of narrative [You07], i.e. the narration of the story to a reader or viewer (recipient), the recipient's knowledge becomes relevant. In many films or television shows, each scene serves primarily to reveal part of the information state to the recipient, sometimes truthfully but sometimes deceptively or ambiguously (the "unreliable narrator"). Other work on interactive narrative has explored the modeling of player and character knowledge to create more flexible authorial imposition, as well [RY13]. Some facets of knowledge can be represented in linear logic—e.g. it is easy enough to create a `knows C M` predicate, where `M` is some pre-defined message type—but modeling knowledge of *logical* information where `M` instead represents some linear logic proposition and knowledge is reflected by logical provability seems necessary in order to capture story discourses involving mystery, deception, and surprise. Thus, we propose the use of epistemic modal logics [Ver02] (and their integration with linear logic [GBB+06]) as a candidate for representating character knowledge in narratives, and we would like to continue the programme of investigating proof theory for narrative representation via such a logic.

Related to knowledge, but perhaps more feasible to explore in linear logic, is the idea of *narrative focalization*:[1] a generalization of "point of view" (first, second, third person) that describes what the narrative recipient learns throughout the story in relationship to what the characters learn. Three kinds of narrative focalization are considered standard:[2]zero, internal, and external. Zero focalization is the *omnicient* point of view where the recipient knows more than the character or characters, i.e. the entire narrative state; internal focalization is the point of view equated with a particular character's knowledge; and external focalization is a point of view that does not reveal character interiority, instead depicting only what the characters' actions would reveal to an objective

[1] Not to be confused with logical focalization.

[2] See `http://wikis.sub.uni-hamburg.de/lhn/index.php/Focalization` for an overview.

witness. We propose using (perhaps annotated) causally-structured traces as artifacts for representing a single narrative model that can be told with arbitrary focalization—the choice of which events to reveal during narration could arise from the combined specification of a narrative trace and a focalization scheme.

A third path of investigation is that of comparing plot structures in multiple narratives. Game-o-Matic [TBMB12] is a template for this idea in the domain of games: an abstract specification of mechanics as 2D movement and collision in Game-o-Matic can be given multiple graphical realizations and shown to model different narratives when interpreted by humans. Similarly, we would like to investigate narratives that have the same trace structure—or Twine games that have the same branching choice structure—and compare their interpreted content, answering the research question of how much structure (in the senses we have identified) informs overall meaning. In a sort of dual direction, we wonder whether narratives that have "the same" plot, such as *Into the Wild* and *Death of a Salesman*, would actually be revealed to have similar structure in linear logical terms, or whether it is instead mainly the interpretive themes that connect them.

We propose the domain of fables and myths as a starting point for the above investigation, perhaps in combination with the PLOTTO book of abstracted plot formulas [Coo28], for narratives that are usually small and abstract enough to formalize on relatively objective terms. The hypothesis to test in this case would be that a formalization of the plot abstraction in linear logic could be given different rendering parameters to generate muliple existing examples of that abstraction in popular culture.

7.2.2 Generative Methods

In a talk at the 2015 Game Developers Conference surveying the field of procedural content generation (or generative methods),[3] the speakers describe classes of techniques distinguished by whether they work *top-down* or *bottom-up*: whether content is generated by way of selecting rules that match certain constraints (top down) or by way of randomly selecting instantiations for variable parameters in the content definition. Backward- and forward-chaining proof construction are sometimes also called top-down and bottom-up (respectively), with very similar meanings to this distinction. While we have investigated generative methods for *narrative* generation using linear logic, we wonder how well the technique could extend to other forms of content generation.

To make the idea a bit more explicit, one idea in bottom-up content generation is that of *production grammars*, such as Lindenmayer systems [PH13] for generating line-based images. The production grammar technique is classified as bottom-up. Grammars and rewriting systems have a well-established correspondence, and the propositional forward-chaining fragment of linear logic has been shown to correspond to multiset rewriting, so at the very least we can express multiset-like production grammars as simple linear logic programs—this would mean using programs like the *generative signatures* described in Chapter 6 to *actually generate* the context sets they characterize.

[3] `http://www.gdcvault.com/play/1022134/Making-Things-Up-The-Power`

(Further work would need to be done to determine how to operationalize those signatures, since they do not always have clear operational semantics.)

If we allow for first-order term variables restricted to binary predicates, then we seem to have a viable grammatical interpretation of these programs as *graph grammars*, or ways of rewriting directional graphs (nodes and potentially-labeled edges). Each binary predicate can be said to model an edge (where the label is the predicate name). Graph grammars have been used quite fruitfully for generative methods, using a graph structure to model game levels, for instance, and generating levels based on graph production rules [Dor10]. The research question we pose is whether proof search on linear logic programs representing graph grammars could be used to similar, or more fruitful, ends.

Finally, a recent concern of generative methods researchers is that of *expressive range analysis*, or determining how the output of a content generator can vary along certain axes [SW10]. Our work on generative invariants, if successful in the future, would be a strong candidate as a framework for carrying out this analysis: generative invariants can be seen as a way of specifying a vector space to which all executions of a linear logic program must stay confined, a similar notion to Smith and Whitehead's characterization of generative variety.

7.2.3 Accessible Game Design Tools

One of the major limitations of this work is that we have developed Ceptre as a core calculus rather than as a deliberately end-user-facing tool, due to our priorities defined by the scope of this thesis. In the future, we would like to develop a front-end development tool for Ceptre that would allow for visual specification of initial states and rules, and corresponding visualization of intermediate states as the program runs. This problem itself poses various research challenges, since the generality of the language means there will be no uniformly appropriate way of representing states visually. As a way forward, we propose using sensing and acting predicates hooked up to inter-process communication to provide an API for programmers to build their own visualization schemas. We would like to provide a few common schemas we imagine Ceptre being used for, such as a web-based interactive fiction parser and renderer, and a grid-based tile renderer and key press processing event loop, using visual rule-based languages such as Kodu, StageCast Creator, [4] and PuzzleScript as inspiration for the design of such a tool in these limited domains.

In the long run, we imagine that a language based on linear logic (not necessarily Ceptre) could be built into a *mixed initiative design* tool [SWM10] that can reason about game designs and provide feedback for a human novice designer on their own designs, using logical proofs as formal models of *design justification*. One proposal for a way forward with such a system would be to build a library of *patterns*, such as the mechanic patterns described at the following site: `http://www.jorisdormans.nl/machinations/wiki/index.php?title=Pattern_Library` Ideally, we could model these

[4] `http://www.stagecast.com/`

patterns in such a way that their recombination could be characterized via logical provability, providing a formal basis for compositional pattern-based design.

A recombination tool for game design patterns has been described in prior work [Orw00], which was used fruitfully in a board-game-like subspace of game designs—we would like to investigate its use for mechanics better suited to linear logical investigation, like emergent story worlds, quests, and resource management strategy.

7.2.4 Reasoning Tools

Finally, we posit that there is much work left to do in terms of building reasoning tools for game prototyping. Game designers frequently express the need for such tools, including hobbyist interactive fiction authors, as evidenced by Andrew Plotkin's development of the *PlotEx* tool for modeling and querying interactive fiction puzzle mechanics.[5] One of the major barriers to the success of simulationist interactive media is the lack of ability to predict and control for unexpected behavior from rules that are too general (or not general enough), and being able to state design intents that are automatically checked would be a major step forward.

Our most-developed avenue of future work to meet this goal is to continue developing the theory of equality for traces that arise from generative signatures such that the admissibility of their reconfiguration witnessing the preservation of an invariant across a rule is decidable. We conjecture that a suitable representation with fewer dependencies to reason about than concurrent traces will be needed to move this idea forward.

Further afield, we note that *answer set programming* (ASP) is a technique that has been investigated quite fruitfully to generate games via authors expressing constraints [SM11], where those constraints are not unlike the kinds of properties we are interested in verifying for hand-authored games. This approach lends itself to a kind of *correct-by-construction* mode of game prototyping. However, the process of *grounding* logic programs in ASP by instantiating each rule with every possible combination of terms makes the specifications less practical as executable artifacts, for which linear logic programs seem better suited. We speculate that a hybrid technique to game specification could be devised wherein linear logic programs could be checked *against* an ASP specification defining its correctness criteria. Along similar lines, we speculate that there is some duality between the constructive logic program *proof* representing a play trace and ASP's *model* representing the same, and we would like to explore those connections more explicitly.

7.3 Final Remarks

We have presented the use of linear logic to model interactive worlds across a broad spectrum of use cases. Linear logic itself has proved a useful tool for this domain, but we suggest that the broader take-away of this work is that the *methodology* of proof theory

[5] `http://eblong.com/zarf/plotex/`

can be quite useful in application to creative tasks. Logic, with all its flaws for capturing the reality of human reasoning, serves as an important common language for translating between human intuitions and executable artifacts. We present this thesis as a first step in establishing that proof-theoretic methodology, with its prescription for logic design and its application to programming languages and knowledge representation, can be a torchlight guiding the way through the twisty passages of understanding computer-assisted creativity.