Generating Explorable Narrative Spaces with Answer Set Programming

Chinmaya Dabral csdabral@ncsu.edu
Chris Martens martens@csc.ncsu.edu
Computer Science Department
North Carolina State University
Raleigh, North Carolina

Abstract

Previous approaches to narrative generation have required a new planner implementation for each set of constraints deemed relevant to the narrative domain, each consisting of thousands of lines of code and supporting one primary mode of interaction: fully specifying a domain and problem, and receiving a plan as output. We present a lightweight, flexible narrative planner written with Answer Set Programming, designed specifically to support constraint-based narrative generation, show how it generalizes previous approaches, and show how it can be easily extended with notions of thematic plot schema such as "betrayal." Finally, we demonstrate how the ASP model can be explored through interactive question answering, where answers take the form of generated narratives. In the long term, we intend this work to support understanding of complex rule systems through interactive exploration.

Introduction

The need to understand complex rulesets pervade people's lives, including those governing board games and sports, end-user software policies, the law defined by government, product user manuals, and scientific theories. We posit that explorability has a critical role to play in making sense of such rule systems. Ideally, rulesets could be understood not as static documents, but as changing, interactive worlds, in which questions can be answered with specific examples, hypotheses tested and refined, and the consequences of pertinent scenarios can be explored. Our overarching goal is to realize this vision of explorable formal models, focusing on privacy policies and regulations as a timely case study.

Narrative generation plays a key role in realizing this vision. Humans have been shown time after time to make better sense of complex information through *stories*, where a progression of events over time with clear cause and effect relationships is depicted, than through unorganized collections of facts (Bruner 1991; Gerrig and Wenzel 2015). Therefore, narrative generation is a key competency that such a system must demonstrate. Existing approaches based on planning offer a promising path forward, due to their explicit world models that represent granular state change.

However, to incorporate the constraints of complex rule systems and to support the varied modes of interaction envisioned, we need a narrative generation system that is more flexible and extensible than those available today. Past approaches have required developing a new planner implementation for each set of constraints deemed relevant to the narrative domain (such as intention in IPOCL (Riedl and Young 2010) and conflict in Glaive (Ware and Young 2011)). Further, they support primarily one mode of interaction: given a planning domain (set of actions) and problem (initial and goal scenario), generate a plan to reach the goal from the initial state. The more general framework we propose supports multiple interaction modes, including partial domain specification (to be filled in by the system), querying narrative spaces (e.g. "How many stories are there where a certain event occurs?"), and other modes of interactive exploration and iterative refinement of narrative models.

To address these goals, we present a narrative planning engine implemented with Answer Set Programming (ASP), and show how to layer narrative constraints and formulate use-specific interactions expressed as ASP clauses. We use the expressiveness afforded by ASP constraints to implement notions of intention, conflict, and belief proposed as key aspects of character believability by previous work (Riedl and Young 2010; Ware and Young 2011; Wadsley and Ryan 2013; Eger and Martens 2017; Shirvani, Farrell, and Ware 2018). We then show how global narrative constraints, such as story grammars and plot structures, can be incorporated. Finally, we present a range of modes of interaction with the system by showing how to introduce scenario-specific constraints and ask a wide range of questions about the encoded narrative possibility space. The particular mode of interaction determines whether the system generates narrative, returns an answer to a specific query, or a combination of both.

Our key claim is that the proposed model generalizes prior narrative generation models and has the potential to enable several new, compelling modes of interaction with a narrative space. The contributions detailed in this paper to support this claim are: 1) our ASP implementation in Clingo (Gebser et al. 2008) of an intentional narrative planner, enabling the use of expressive narrative constraints; 2) a methodol-

ogy to encode arbitrary constraints that interact with the narrative possibility space, using the BRUTUS model of narrative themes like betrayal (Bringsjord and Ferrucci 1999) as a case study; and 3) example encodings of question-and-answer interactions, demonstrating the versatility of our approach. Our long-term aspiration is to use this constraint-based narrative generation system as the back-end for a user-facing explorable, interactive system that will deepen user understanding of complex rule systems.

Related Work

The idea of using theorem proving technology to carry out narrative generation is not new. The BRUTUS system (Bringsjord and Ferrucci 1999) positions itself in this way: "we approach story generation through logic; in more specific terms, this means that we conceive of story generation as theorem proving" (Bringsjord and Ferrucci 1999). Mueller (2007) present a computational model of narrative that combines model finding and planning; however, their goal is story understanding rather than generation. A more recent line of work maps story domains into propositions in linear logic (Martens et al. 2013; 2014; Bosser et al. 2011), which allows for logical proofs to be directly mapped to stories; however, this approach depends on a specialized logic that is less readily adapted to meta-analysis and constraint within a standard logic programming language.

Outisde of the narrative domain, we are motivated by prior work in what we describe generally as systems that enable *explorable formal models*. Semantic modeling tools such as Rosette (Torlak and Bodik 2013), Alloy (Jackson 2012), PLT Redex (Klein et al. 2012), the K (Roşu and Şerbănuţă 2010) semantics language, the Spoofax language workbench (Kats and Visser 2010), Razor (Saghafi and Dougherty 2014), and miniKanren (Byrd 2009) provide a basis for authoring formal specifications and programs that manipulate them, usually centered on applications to programming language theory and design. These tools support interactive execution and model querying with the same versatility that we aim to provide for the narrative domain.

Our work is probably most closely related to the RoleModel system (Chen et al. 2010), whose efforts to develop a narrative generator in ASP via the Event Calculus inspire our own approach. However, RoleModel is explicitly *not* a planning-based approach, lacking mechanisms for modeling character intentions, causality, or alternative timelines. RoleModel also limits the constraints considered relevant to narrative generation to forbidding and requiring certain roles and actions.

An Answer Set Programming approach to Narrative Planning

The use of planning for narrative generation has a long history (Young 1999; Porteous, Cavazza, and Charles 2010; Riedl and Young 2010; Ware and Young 2011). In this approach, narratives are represented as causally-linked events, each of which changes the facts that are true in the world model. Two key attributes are considered important for the believability of a narrative: the *causal progression of plot* and *character believability*. To ensure believability of characters with cognitive processes, they must be *intentional agents*. In other words, each action a character takes should be in service to a goal (Riedl and Young 2010). A character's intentions are established as part of the narrative itself, and then drive their actions. We adopted this idea in our planner because it lets us explain a character's actions and generate examples which do not seem contrived.

Our software system consists of a general-purpose narrative planner supporting intention and conflict, implemented using Answer Set Programming (ASP). ASP is a declarative programming technique where programs are sets of logical expressions (i.e. *logic programs*), specified in terms of generative rules, facts, and constraints. A constraint solver then attempts to find stable models (answer sets) for the logic program (Gelfond and Lifschitz 1988). As opposed to other logic programming languages like Prolog, ASP provides a means of expressing disjunctive clauses through *choice rules*, which allow multiple possible worlds to be consistent with a given program. Specifically, we used Clingo (Gebser et al. 2008) as our implementation language.

Implementing a planner in ASP opens up the possibility of using the rich constraints provided by the system for sculpting the possibility space of generated narratives. The interplay between generative rules and constraints, coupled with relational programming, allows a model, once specified, to be used in several different ways. An ASP model merely specifies the relationships between different predicates. Additional constraints can be applied later to use some of the predicates as "inputs" to the program, while the solver finds consistent values for the rest.

Preliminaries

ASP can be used to generate narrative scenarios from a possibility space of action sequences, as explored initially in the RoleModel system by Shen et al. (Chen et al. 2010). Possibility spaces are represented through *choice rules* of the form $\{\phi\}$, which indicate that the formula ϕ may, but is not required to, hold in each model. For example, the following program expresses the possibility for a user to do any, both, or neither of two post actions:

```
{happens(attack(princess, robot)); happens(
    attack(robot, princess))}.
```

A standard answer set solver will report these results as:

```
Answer: 1

Answer: 2
happens(attack(robot,princess))
Answer: 3
happens(attack(princess,robot))
Answer: 4
happens(attack(robot,princess)) happens(
    attack(princess,robot))
```

Logic variables (indicated syntactically as identifiers starting with a capital letter) and conditions (formulas to the right of the backwards implication symbol :−) can be used to quantify over finite sets and generate possible clauses for

each element. For example, the following rule generates a set of satisfying models in which `happens` may or may not hold for each action specified as "possible:"

```
{ happens(Action) } :- possible(Action).
```

The resulting generative space has size $2^{|Action|}$.

Event Calculus

In narratives, the state of the world changes over time. It is common to formalize state change with predicates whose truth depends on a temporal index; these time-varying predicates are called *fluents*. *Event calculus* (Kowalski and Sergot 1989) is a particular logical theory of fluents used to represent and reason about sequences of events. The state evolves through *actions*, which describe how fluents may change from one time step to the next. They may require preconditions to be satisfied before they can be executed. Actions are assumed to be instantaneous, and occur in discrete time steps.

A common formalization of event calculus axioms is as follows:

$$holds(T+1, F) \leftarrow initiates(T, F) \tag{1}$$

$$holds(T+1, F) \leftarrow holds(T, F) \wedge \neg terminates(T, F) \tag{2}$$

Part (1) states that if an action at time T $initiates$ (makes true) a fluent, then the fluent $holds$ (is true) at the next time step. Part (2) states that fluents have inertia, i.e., they continue to be true until $terminated$ by an action. This assumes default negation (if predicate P cannot be derived, then $\neg P$ is assumed to hold). This representation can be readily translated into ASP.

Event calculus for partially ordered events

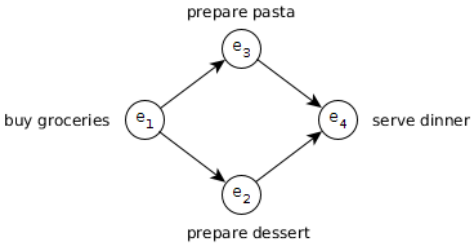

Figure 1: Consistent partial order

A narrative is a sequence of causally linked events (actions). Actions naturally form a partial order because of their prerequisites and effects. Some actions must be performed before others, while others may be mutually independent. For instance, in the sequence of events *buy groceries* $\rightarrow$ *prepare pasta* $\rightarrow$ *prepare dessert* $\rightarrow$ *serve dinner*, we can see that *buy groceries* must occur first, *serve dinner* must occur last, but the other two events can occur in either order in between them. We can depict this as a diamond graph (Figure 1), where e_2 and e_3 are *unordered*. This is a better representation than a linear graph, since it allows us to represent causality and to arbitrary reorder independent events.

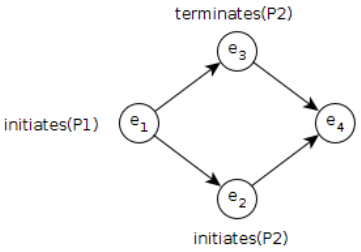

Figure 2: An example of an ambiguous partial order. The arrows represent the partial order relation *happens before*. This relation is transitive, but we have omitted the edge between e_1 and e_4, since it can be derived through transitivity. We will omit such edges throughout the paper for clarity. Nodes are labeled with the effects of each action.

The standard event calculus formulation works well for a totally ordered sequence of events, but breaks down if we have a partial order. Consider the four events shown as the nodes of a graph in Figure 2. Here we can be sure that $holds(e_4, P_1)$ is true, but we cannot say anything about $holds(e_4, P_2)$. In fact, (1) and (2) would incorrectly imply that $holds(e_4, P_2)$ is true.

We therefore extend the standard event calculus formulation to allow for partially ordered events by introducing a new predicate, $strong\_holds$, which we define as follows:

$$strong\_holds_{\mathcal{P}}(e, F) \iff \forall \mathcal{T} \in \boldsymbol{C}(\mathcal{P})\, holds_{\mathcal{T}}(e, F) \tag{3}$$

The predicate subscripts indicate the order over events in which the predicate holds, and $\boldsymbol{C}(\mathcal{P})$ is the set of all total orders consistent with the partial order $\mathcal{P}$. We say that total order $(A, <_{\mathcal{T}})$ is consistent with partial order $(A, <_{\mathcal{P}})$ iff:

$$\forall x, y \in A\, (x <_{\mathcal{P}} y \implies x <_{\mathcal{T}} y)$$

In other words, $strong\_holds(e, F)$ is only true if we can be sure that the fluent will hold regardless of how we "linearize" the partial order. Going back to figure 2, we can conclude that $strong\_holds(e_4, P_1)$ is true, but not $strong\_holds(e_4, P_2)$.

This, however, creates the issue that we can no longer rely on default negation. For instance, while $strong\_holds(e4, P_2)$ is not true, we also cannot say for sure that the fluent P_2 will not hold at e_4. So we introduce another predicate, $strong\_notholds$, which is defined in a similar manner:

$$strong\_notholds_{\mathcal{P}}(e, F) \iff \forall \mathcal{T} \in \boldsymbol{C}(\mathcal{P}) \neg holds_{\mathcal{T}}(e, F) \tag{4}$$

where all other terms are as defined in (3).[1] Then we can force the planner to generate sequences where the partial

[1] For implementation in Clingo, we need to translate this statement to an equivalent formalization expressible in Clingo's language of Horn clauses. We omit the expanded definition for brevity.

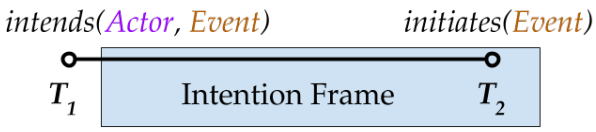

Figure 3: The definition of an intention frame, depicted schematically.

order matches causality by maximizing the number of unordered nodes in the graph,[2] as shown in Listing 1.

Listing 1: Showing causal links by maximizing the number of unordered nodes

```
unordered(X, Y) :-
    id(X), id(Y),
    not after(X, Y),
    not before(X, Y),
    X!=Y,
    concludes(C, X):
        terminus(C),
        concludes(C, Y).
#maximize
    {1@1,unordered(X,Y):unordered(X,Y)}.
```

Narrative Constraints

Intention

Character intentionality refers to the idea that each action a character takes in the generated narrative should be in service to a goal (Riedl and Young 2010). A character's intentions are established as part of the narrative itself, and then drive their actions. We adopted this idea in our planner because it lets us explain a character's actions and generate examples which do not seem contrived.

Figure 3 presents a schematic representation of a key part of our logical definition of intention. An intention created at time T_1 is *satisfied* at some later point T_2 iff T_2 initiates the intended event, and there is no earlier timepoint T' between T_1 and T_2 which satisfies it. We then define an intention frame by saying that any timepoint between when the intention is formed and satisfied, on which the satisfying event is causally dependent, is within the frame. Our ASP implementation is shown in Listing 2.

Listing 2: Definition of intention frames and intentionality of plans.

```
%% Defining satisfies(I, N)
satisfies(I2, intention(I1, X, A)) :-
    intends(I1, X, A), initiates(I2, _, X),
    after(I2, I1),
    not satisfies(I3, intention(I1, X, A)) :
        after(I3, I1), before(I3, I2).

% CONSTRAINT: All intended fluents must hold
% at some point after they're intended
:- intends(I1, X, A),
    not satisfies(_, intention(I1, X, A)).

% Defining frame(I, N)
frame(I3, intention(I1, X, A)) :-
    satisfies(I2, intention(I1, X, A)),
    after(I3, I1), before(I3, I2),
    actor(I3, A).

% CONSTRAINT: All events must be part of
% frames of all their actors
:- actor(I, A),
    not frame(I, intention(_, _, A)),
    not satisfies(I, intention(_, _, A)),
    A!=env.
```

[2]Clingo supports maximization of the cardinality of a given set through the `#maximize` directive.

Underspecified Intentions In prior work on which we base our intention model (Riedl and Young 2010), intentions are assigned to agents either as part of effects of certain actions (e.g. someone hurting a character might motivate them to exact revenge), or automatically as part of the initial conditions of the planning problem. The domain author needs to design actions that can cause intentions to arise. Since we want to be able to fully explore the possibility space consistent with a given policy, we would have to carefully design intention-causing actions so that all possible outcomes were covered, which would be a daunting task.

To solve this issue, in addition to supporting hand-authoring of intention assignment actions, we introduce a special agent, the *mastermind*, whose job is to *motivate* (assign intentions to) other agents in order to satisfy global planning constraints and goals. These intentions are selected from the space of all available fluents. This lets us avoid comprehensive hand-authoring of character intentions, making more actions reachable while retaining the ability to explain a character's decisions in every possible story.

Conflict

Conflict is a central concept of European-originating narrative theory: a story is generally considered interesting only if different characters (intentional agents, which may include nonhuman entities such as the environment or society) act towards conflicting goals (Ross 1993). The CPOCL narrative planner models conflict as a character's plans being thwarted by another character (Ware and Young 2011). In other words, a conflict occurs when two or more intentions are mutually incompatible. We incorporate this model into our planner, allowing us to model real-world scenarios with antagonistic actors in addition to fictional scenarios with greater narrative interest.

The CPOCL model induces a notion of alternate, or branching, timelines, in the sense that for a plan to be thwarted, that plan has to have been intended to take place by the thwarted agent. A given timestep can thus have multiple *conclusions*. We define a conclusion of a given timestep as any terminus that occurs after it. We then define a *split* in the timeline, indicating unexecuted actions, as occurring when a child node does not have all of the conclusions of its parent. Finally, a node where timeline splits should have a single parent, in order to avoid scenarios where a timeline

feeds back into another timeline. This definition is shown in Listing 3.

Listing 3: Definition of conflict creating split timelines.

```
terminus(C) :- id(C), not after(X, C): id(X)
    .
concludes(C, X)
    :- id(X), after(C, X), terminus(C).
concludes(C, C) :- terminus(C).

split(X, Y)
    :- edge(X, Y),
        concludes(C, X), not concludes(C, Y)
            .
:- split(X, Y),
    not { edge(P, Y): id(P) } = 1.
```

Plot Schema

BRUTUS (Bringsjord and Ferrucci 1999) is a narrative generation engine that encodes themes such as *betrayal* and generates stories that contain these themes. For example, BRUTUS uses a Prolog-like language to "mathematize" betrayal as shown in Listing 4.

Listing 4: BRUTUS plot schema definition for betrayal.

```
betrayal(Betrayor, Betrayed) :-
    goal(Evil, EvilPlan, Betrayor),
    includes(EvilPlan, BetrayorsLie),
    say(BetrayorsLie),
    includes(EvilPlan, Thwarting),
    thwart(Thwart),
    prevented_goal(Thwarting, BetrayedsGoal)
        ,
    supports(BetrayorsLie, BetrayedsGoal),
    goal(BetrayedsGoal, BetrayedsPlan,
      Betrayed),
    belief(Betrayed, BetrayorsLie).
```

The concept of themes of this nature, which we refer to as *plot schema*, dates back to pre-digital notions of narrative grammars, as in Propp's foundational *Morphology of the Folktale* (Propp 1928). Narrative grammars informed early computational approaches to narrative formalization, and they have an ongoing legacy in more recent work such as *plot units* (Lehnert 1981; Goyal, Riloff, and III 2013) and *story intention graphs* (Rishes et al. 2013). While planning-based approaches afford much richer world models, narrative grammars and plot schema remain useful as a way of codifying *global structure* to plots, allowing us to identify common plot structure between distinct story contexts (e.g. Disney's *The Lion King* as an adaptation of Shakespeare's *Hamlet*). This model is also useful if, say, we want to require generated narratives to carry particular themes or tropes, follow a particular narrative arc (pattern of rising and falling action), or subvert genre expectations for any of the above.

We show how our planner can gracefully account for plot schema like this by encoding the betrayal example into the notions of intention and conflict supported by our planner (presented here in natural language for readability):

- The Betrayed has intention I_1.

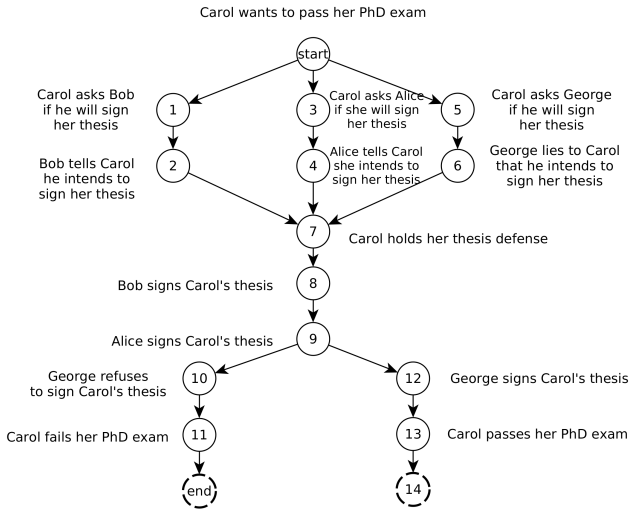

Figure 4: A betrayal narrative generated by our system.

- The Betrayer expresses intention I_2, but has intention I_3.
- Step S_2, which would satisfy the expressed intention I_2, lies inside the intention frame of the Betrayed's intention I_1.
- Step S_2 occurs in an alternate timeline, in which I_1 is also satisfied.
- Step S_3 occurs on the actual timeline and satisfies I_3.

Figure 4 shows a betrayal narrative generated by our system using this encoding, using an example domain inspired by BRUTUS. The Betrayed (Carol) needs 3 signatures on her thesis to pass her PhD exam (I_1). The Betrayer (George) promises to sign her thesis (I_2), but instead has the intention to see her fail (I_3). The main timeline extends from the $start$ node to the end node, and satisfies the Betrayer's intention. The Betrayed's intention is satisfied in an alternate timeline (extending from $start$ to node 14), and depends on the Betrayer's promise.

Example Interaction

Our central claim is that our ASP formulation unifies and generalizes prior work on plan-based narrative generation, supporting several modes of interaction through slight modifications on input constraints. This section validates the final claim by demonstrating an example domain and interaction.

Consider a domain where a dragon needs to be slayed. There are various ways to achieve that goal, but the dragon has a specific weakness. The goal of the user is to find that weakness.

Our hero may ask one or more of: Ice Wizard, Air Bender, or Fire Demon to accompany them.
They may take one or more of: Earth Stone, Fire Lantern, or Ice Diamond with them.
They may travel through one or more of: Fire Pit, Air Gardens, or Metal Caves.

To find the pattern, the user first asks the system to generate a few narratives where the dragon is successfully slayed.

Listing 5: Generating successful narratives

```
:- not strong_holds(end, dragon_slayed).
```

The user sees the following two narratives:
Narrative 1:

- The hero asks the Ice Wizard to accompany them
- The hero travels through the Fire Pit
- The hero acquires the Earth Stone

Narrative 2:

- The hero asks the Fire Demon to accompany them
- The hero travels through the Air Gardens
- The hero acquires the Earth Stone

The user observes two common elements: earth stone, and fire. The user wants to check whether a successful narrative can be generated without using either.

Listing 6: Narrowing down

```
:- not strong_notholds(end, acquired(
    earth_stone)).
:- type(I, fire), id(I).
```

The system then generates the following narrative, which means indeed, it is possible:

- The hero asks the Air Bender to accompany them
- The hero acquires the Ice diamond
- The hero travels through the Metal Caves

The user notices that the sequence always contains 3 distinct element types. At this point, the user would like to know what an unsuccessful narrative looks like.

Listing 7: Unsuccessful narrative

```
:- not strong_notholds(end, dragon_slayed).
```

- The hero asks the Air Bender to accompany them
- The hero acquires the Ice diamond
- The hero travels through the Air Gardens
- The hero asks the Air Bender to accompany them
- The hero travels through the Metal Caves
- The hero acquires the Earth Stone
- The hero travels through the Air Gardens

At this point, it seems that the occurrence of the same element more than once prevents the dragon from being slayed, and we require 3 distinct elements. To confirm this, the user inputs:

Listing 8: Unsuccessful narrative

```
:- 2 { type(I, T): id(I) }, type(T).
```

Discussion

The entire CPOCL planner implementation is 160 lines of ASP code, a small enough size to support independent review and reuse in other systems. All code will be made available and linked in the paper upon publication.

There are some limitations of our work that need to be addressed. First, while we have a notion of causality and ordering of events, the narratives do not specify how this ordering relates to time in the real world in days and hours. This can be relevant when encoding constraints that mention specific time intervals. One way to implement this is to introduce a fluent `days_elapsed(N)`, which is less than ideal, because it will need to be included in every action definition.

Second, we did not focus much on optimization, and state space explosion can create performance issues. For narratives longer than 20 steps, grounding can result in a 1GB+ file, with a memory usage of 8GB+. We will focus on optimization in the next phase of our work.

Third, when generating all possible answer sets, many of the narratives generated are very similar and differ only in inconsequential details, like graph node identifiers. The user can usually apply additional constraints to narrow down to the desired narrative, but this is less than ideal.

While Clingo proved to be a good choice for representing our system, we did face some hurdles during implementation. A lack of support for typed predicate arguments means that typos in predicate names and arguments often go unnoticed, leading to hours of time wasted debugging. This issue could be alleviated by an extension to the language allowing explicit typing and static type checking.

Conclusion and Future Work

This work unifies several previous narrative models within a relational programming framework that can be adapted to a number of use cases. We presented a flexible, CPOCL-style narrative planner written in pure ASP, showing how prior work on intention and conflict could be represented elegantly. We then demonstrated the versatility of this approach by incorporating thematic constraints via *plot schema* and showing how a rich interaction could be supported with small changes to the input.

Our long-term vision with this project is to develop explorable formal models that users can interact with to develop an understanding of complex rule systems through play. While we have demonstrated that our system can gracefully represent concepts in preceding narrative generation systems and support several modes of interaction, we require development of a user interface for translating between natural language and computable ASP queries. We currently have an interface to parse, filter, and visualize the output of our planner, but a more usable interface will need to be devised for users to be able to add constraints.

References

Bosser, A.-G.; Courtieu, P.; Forest, J.; and Cavazza, M. 2011. Structural analysis of narratives with the coq proof assistant. In *International Conference on Interactive Theorem Proving*, 55–70. Springer.

Bringsjord, S., and Ferrucci, D. 1999. *Artificial intelligence and literary creativity: Inside the mind of BRUTUS, a storytelling machine*. Psychology Press.

Bruner, J. 1991. The narrative construction of reality. *Critical inquiry* 18(1):1–21.

Byrd, W. E. 2009. *Relational Programming in miniKanren: Techniques, Applications, and Implementations*. Ph.D. Dissertation, Indiana University.

Chen, S.; Smith, A. M.; Jhala, A.; Wardrip-Fruin, N.; and Mateas, M. 2010. Rolemodel: towards a formal model of dramatic roles for story generation. In *Proceedings of the Intelligent Narrative Technologies III Workshop*, 8. ACM.

Eger, M., and Martens, C. 2017. Character beliefs in story generation. In *Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Gebser, M.; Kaminski, R.; Kaufmann, B.; Ostrowski, M.; Schaub, T.; and Thiele, S. 2008. A user's guide to gringo, clasp, clingo, and iclingo.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *ICLP/SLP*, volume 88, 1070–1080.

Gerrig, R. J., and Wenzel, W. G. 2015. The role of inferences in narrative experiences. *Inferences during reading* 362.

Goyal, A.; Riloff, E.; and III, H. D. 2013. A computational model for plot units. *Computational Intelligence* 29(3):466–488.

Jackson, D. 2012. *Software Abstractions: logic, language, and analysis*. MIT press.

Kats, L. C., and Visser, E. 2010. The spoofax language workbench: Rules for declarative specification of languages and ides. In *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '10, 444–463. New York, NY, USA: ACM.

Klein, C.; Clements, J.; Dimoulas, C.; Eastlund, C.; Felleisen, M.; Flatt, M.; McCarthy, J. A.; Rafkind, J.; Tobin-Hochstadt, S.; and Findler, R. B. 2012. Run your research: on the effectiveness of lightweight mechanization. *ACM SIGPLAN Notices* 47(1):285–296.

Kowalski, R., and Sergot, M. 1989. A logic-based calculus of events. In *Foundations of knowledge base management*. Springer. 23–55.

Lehnert, W. G. 1981. Plot units and narrative summarization. *Cognitive science* 5(4):293–331.

Martens, C.; Bosser, A.-G.; Ferreira, J. F.; and Cavazza, M. 2013. Linear logic programming for narrative generation. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, 427–432. Springer.

Martens, C.; Ferreira, J. F.; Bosser, A.-G.; and Cavazza, M. 2014. Generative story worlds as linear logic programs. In *Seventh Intelligent Narrative Technologies Workshop*.

Mueller, E. T. 2007. Understanding goal-based stories through model finding and planning. In *AAAI Fall Symposium: Intelligent Narrative Technologies*, 95–102.

Porteous, J.; Cavazza, M.; and Charles, F. 2010. Applying planning to interactive storytelling: Narrative control using state constraints. *ACM Transactions on Intelligent Systems and Technology (TIST)* 1(2):1–21.

Propp, V. 1928. *Morphology of the Folktale*, volume 1. University of Texas Press.

Riedl, M. O., and Young, R. M. 2010. Narrative planning: Balancing plot and character. *Journal of Artificial Intelligence Research* 39:217–268.

Rishes, E.; Lukin, S. M.; Elson, D. K.; and Walker, M. A. 2013. Generating different story tellings from semantic representations of narrative. In *International Conference on Interactive Digital Storytelling*, 192–204. Springer.

Ross, E. I. 1993. *Write Now*. Barnes & Noble Publishing.

Roşu, G., and Şerbănuţă, T. F. 2010. An overview of the K semantic framework. *Journal of Logic and Algebraic Programming* 79(6):397–434.

Saghafi, S., and Dougherty, D. J. 2014. Razor: Provenance and exploration in model-finding. In *PAAR@ IJCAR*, 76–93. Citeseer.

Shirvani, A.; Farrell, R.; and Ware, S. G. 2018. Combining intentionality and belief: Revisiting believable character plans. In *Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Torlak, E., and Bodik, R. 2013. Growing solver-aided languages with rosette. In *Proceedings of the 2013 ACM international symposium on New ideas, new paradigms, and reflections on programming & software*, 135–152. ACM.

Wadsley, T., and Ryan, M. 2013. A belief-desire-intention model for narrative generation. In *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Ware, S. G., and Young, R. M. 2011. Cpocl: A narrative planner supporting conflict. In *Seventh artificial intelligence and interactive digital entertainment conference*.

Young, R. M. 1999. Notes on the use of plan structures in the creation of interactive plot. In *AAAI fall symposium on narrative intelligence*, 164–167.