RoleModel: Towards a Formal Model of Dramatic Roles for Story Generation

Sherol Chen Adam M. Smith Arnav Jhala Noah Wardrip-Fruin Michael Mateas
Expressive Intelligence Studio
University of California Santa Cruz
Santa Cruz, CA USA
(831) 459-0111
{sherol, amsmith, jhala, nwf, michaelm} @soe.ucsc.edu

ABSTRACT

RoleModel is a novel story generator organized around explicit formal models of character roles. RoleModel expands the expressiveness of stories generated from arbitrary partial domain specification by using a formal model of roles within an abductive logic programming framework. Authorial goals in the system can be fully or partially specified as constraints in an abductive logic program. In particular, the RoleModel system focuses on representing and satisfying role constraints of the story characters. This paper discusses the basic architecture for the RoleModel approach, demonstrates example output from the system through three use-cases, discusses the authorial expressiveness enabled by a "stageless" abductive logic approach to story generation, and proposes the current and future directions.

Categories and Subject Descriptors

K.8.0 [**Personal Computing**]: Games

General Terms

Algorithms, Design

Keywords

Story Generation, Story Understanding

1. INTRODUCTION

This paper proposes a new approach for incorporating a formal model of character roles to generate stories and describes an implemented prototype of the core mechanisms. The major focus of this system is on reasoning about character roles to produce distinct and understandably contrasting variations of story.

Character roles and archetypes play an important part in storytelling by providing motivations for character actions and introducing clearly recognizable dramatic interactions. Expert storytellers exploit character roles and role changing situations to manipulate user's beliefs and expectations to bring about dramatic conflicts and resolutions. For example, in Kurosawa's *Rashomon*, several re-tellings of a dramatic situation are presented to the viewer. In each narration, from a different character's point-of-view, roles of participating actors (e.g. Victim, Aggressor) are manipulated to create coherent variations of the situation. Specific roles provide affordances for characters to undertake particular types of actions within the story. For example, in *Rashomon*, the woman's role of being either the aggressor or the victim provides the author with an option to create interesting variations on the aggressive episodes within the story. For intelligent storytelling systems, a rich formal model of roles enables authors to partially specify the domain and character constraints without sacrificing consistency of character behaviors with respect to their roles.

RoleModel is a story generator that explicitly models roles to generate meaningful variations of story situations. Due to the complexity involved in authoring complete and consistent formal domains that generate an authorially desired story space, we investigate the use of abductive logic programming to create models of possible story variations from a partially specified domain. Such a system provides authors with the ability to explore the space of possible variations given varying levels of story constraints.

In making roles a first class problem, our system takes advantage of the strong perception of affordances for roles, such as victim or hero, in story. With a dynamic constraint space designed around maintaining roles, there are three authorial use-cases that can be effectively implemented: (1) a tabula rasa generator, which takes few or no constraints and autonomously generates varied narratives from the background theory, (2) a partially constrained generator, with which the author can specify additional story constraints on top of the background theory, such as constraints on role fillers, character traits, and even the appearance of specific events within the story, without locking down a specific linear sequence of events, and (3) a highly constrained generator, with which an author can specify a linear story that the system generates variations and explanations on. In focusing on satisfying role constraints, the overall space of constraints can be viewed as properties of characters or properties of actions. Character properties include roles, traits, dynamic attributes, and sentiments towards actions, while action properties include a variety of contextual properties and causal constraints. The relationships among these constraints provide the background theory for the solver to use. For our prototype, generation involves asking the system to satisfy a list of additional story constraints (including no constraints) on top of the background theory. The system produces a collection of grounded predicates (an answer set), where each collection corresponds to a concrete story that satisfies the constraints given the background theory. In the prototype, actions are represented using the event calculus, supporting temporal inferences about actions.

2. PREVIOUS WORK
Approaches in story generation take on a few different strategies. Varieties of automatic story-generation include the character-goal driven approach, the story grammar approach, the author-goal based approach, and the audience-model driven approach. RoleModel is a generator towards the audience-model approach; however, functions based on models of authorial goals, role specifications, in particular. In implementation, RoleModel uses abductive logic reasoning as a preliminary implementation of role constraint satisfaction for story generation. This is similar to Mueller's goal-based approach for story understanding though model finding and planning which appropriately breaks down stories into models that can be reasoned upon by the background knowledge of story [7].

Recent implementations of story generation include Perez y Perez's MEXICA:

> MEXICA is a computer model based on the engagement-reflection cognitive account of creative writing that produces stories about the Mexicas (the old inhabitants of what today is México city, also wrongly known as Aztecs). During the engagement-mode the system produces material driven by content and rhetorical constraints avoiding the use of explicit goal-states or story-structure information. During the reflection-mode the system breaks impasses generated during engagement, satisfies coherence requirements, and evaluates the novelty and interestingness of the story in progress. If the results of the evaluation are not satisfactory, MEXICA can modify the constraints that drive the production of material during engagement. In this way, the stories produced by the program are the result of the interaction between engagement and reflection.

RoleModel, similar to MEXICA, is a system designed to model the author [8]. More recently, the MEXICA-nn plot generator and automatic narrator creates variations on story based off of initial composition of story from the plot generator and the generated discourse from the narrator [6]. Minstrel is another system that takes an author modeling approach by satisfying constraints from authorial goals, which is similar to the approach that RoleModel takes [11].

As introduced, RoleModel's primarily function is to satisfy role constraints, which builds off of previous work in character believability. A final major body of previous work relevant to RoleModel is found in the area of ideological modeling.

2.1 Logical Representation of Story
Murray Shanahan [10] builds a prolog program to demonstrate the appropriateness of logical abduction for temporal analysis. He elaborates, "Temporal reasoning involves both *prediction* and *explanation.* Prediction is projection forwards from causes to effects whilst explanation is projection backwards from effects to causes. That is, prediction is reasoning from events to the properties and events they cause, whilst explanation is reasoning from properties and events to events that may have caused them." He concludes that prediction typically receives more attention than explanation [10]. Along those lines, RoleModel aims to thoroughly explore the explanation space, making prediction a secondary concern through the logical abduction like Shanahan describes.

More recently, Mueller uses model finding and planning to produce goal-based stories. As Mueller shows, stories are conveniently represented, understood, and inferenced through propositional satisfiability [7]. Through similar representation, RoleModel finds models from authorial constraints and background knowledge of roles and story to abduce valid explanations.

2.2 Character Believability
A side from the event calculus, integrity constraints, and abductive representation, the majority of RoleModel's operations is driven by its background knowledge of story. The contextual rules regarding character traits and action properties are built around representing and maintaining character roles. Maintaining role consistency is a form of character believability. Reidl and Young [6] describe character believability as, "the perceptions that story world characters are action according to their own beliefs, desires, and intentions." This sort of story explanation can powerfully evaluate viewer/reader comprehension [6] and engagement with represented stories, and, therefore, creating understandable and desirable story variations.

RoleModel, however, is not a character-goal driven story generator. Cavazza, Charles, and Mead created character-based storytelling using Hierarchical Task Networks formalized by AND/OR graphs [4]. Like RoleModel, their system maintains deterministic behaviors with varying interactions and sentiments between actors. They take this approach to avoid complex control problems of explicit plot representation [4]. RoleModel, however, is not concerned with the application of intelligent virtual agents, and far less concerned with generating behaviors, as it is with finding explanations (whether behavioral or contextual) to justify role assignments. This creates a less domain-specific story representation and requires a less complex model of character, while also avoiding major problems with explicit plot representation.

2.3 Ideological Modeling
In section 2.1, the prior work supports the use of logical representation of story to perform useful operations such as: understanding, explanation, and prediction [7][10]. In particular, the under-explored space of explanation is a direct application of logical abduction [10]. In section 2.2, the prior work determines that the audience engagement with story requires some perceptual believability in characters [6]. By creating a separate space for believability based story generation, complex control problems of explicit plot representation can be avoided [7].

Believability in RolModel is maintained, not by models of characters or intelligent virtual agents, but rather by models of ideological manipulation and representation. A well known early implementation of ideological modeling is Abelson's Goldwater Machine [1]. By using ideological models, RoleModel is able to maintain the integrity of authorial constraints despite how strict or contradictory they may be. Abelson demonstrates this through his Goldwater Machine by using rationalization mechanisms to deal with upsetting statements, "each of which represents a different way of denying the psychological responsibility of the actor for the action:" (1) re-assigning responsibility to another character, (2) by assuming that original action was accidental and unintended, (3) that original action sets into motion a more appropriate outcome [12]. Similar rationalization mechanisms are represented as contextual rules/constraints in RoleModel. Rationalization mechanisms, in RoleModel, are used to maintain role assignments of characters which, in addition to modeling ideologies, requires formal representation of character roles. Such mechanisms, in RoleModel, are used, for instance, to undo what would be, by default, perceived as a character becoming an aggressor, if the character must only be an innocent victim. Examples of this can be found in the Goldwater Machine's attempts to justify that "the United States can do no evil." Similar work has been done, such as Selmer Brinsjord's formal model of evil [2]. Subsequent systems that leverage ideology are the POLITICS [2] and Terminal Time [5].

3. ROLEMODEL APPROACH

Figure 1 depicts the architecture of the RoleModel system. The system takes as input: constraints on the timeline, a set of character names, character traits associated with each name, required or forbidden role assignments for each character, and character actions. From this input, the system generates a pool of stories through further trait assignment, event selection, and role derivation that are consistent with the input constraints and the background theory.

RoleModel's ontology consists of:

- Character Traits and State Attributes
- Character Roles
- Observable Character Actions
- Contextual Details (properties of actions)
- Contextual Sentiments (desires and reactions)

Requiring or forbidding a role implies constraints on character actions, sentiments towards action, and other contextual details. As shown in Figure 1, RoleModel performs three major operations: trait assignment, event (or action/context) selection, and role derivation. Role constraints (or preconditions) are built on pre-established contextual rule systems, made up by background knowledge of stories. In order to satisfy authorial goals (represented by the input), the system justifies role assignments through assignment of actions and attribution of explanations for actions that are consistent with character roles and influenced by character traits. The three major operations are not performed sequentially as the diagram may suggest, but carried out simultaneously in a unified solving process based on logical abduction.

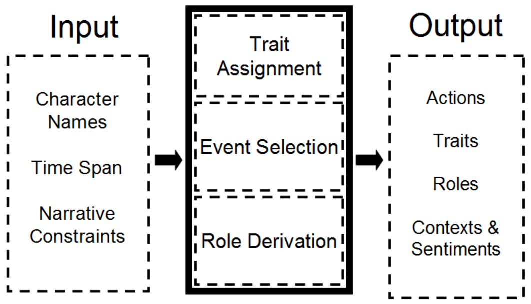

Figure 1. RoleModel System Architecture.

Based on given characters, role constraints, and goals, the RoleModel system aims to elaborate upon the given initial story assertions (or problem constraints) to establish or amplify character roles through addition of preconditions on actions that fulfill character role constraints. In changing the role assignments, the system manipulates background knowledge and elaborates upon the story without breaking the initial story conditions as specified by the author. While many story generators emphasize the means by which they maintain causal consistency between actions, we are interested in how dynamic role assignment, and the implications that follow from roles, can be used to reframe similar action sequences to have very different meanings.

3.1 Roles

Roles are abstract and universally identifiable labels that can be applied to characters and are built off of traits, sentiments, and actions. Example roles include: hero, fool, victim, aggressor, bystander, etc. RoleModel strategically focuses on role properties to show distinctive variations among generated stories and takes away focus from the causal construction of actions. A character's actions may implicitly designate role assignments; however, the background story knowledge must have methods to explain an undesired role regardless of circumstances within the story. The negation of a pre-existing role assignment can be made through certain character sentiments, character reactions to the incriminating act, motivational context, other contextual details, and additional actions. RoleModel intends to be able to refute alleged roles (when appropriate) so that we can satisfy author goals in all circumstances and illustrate the variational power of our approach. For instance, if a role-constraint forbids a character from being an aggressor, but the character commits a violent act, then there must be an operation that can explain away the prescribed role. By maintaining character action constraints, the generator makes compelling and yet understandably contrasting variations.

3.2 Characters

Characters may be associated with traits, roles, dynamic state-attributes, and contextual sentiments.

- **Traits** are characteristics of each character that can be used to contextually explain the motivation for character actions in a story and also determine new appropriate sentiments and actions among the other specified actions in the story. Examples of traits include: insane, strong, male, female, etc.
- A **role** can be required or forbidden for a character. If no role constraint is specified, the system may implicitly assign a role from the inputted and generated actions during story generation. If a required role is specified by the author, the system ensures that character actions are constrained by the given role, otherwise, contextually justified. Similarly, if a role is forbidden by the author, then the system will either prevent the defining actions of the role or contextual justify how a role is inappropriate for the character.
- **Character state-attributes** are necessary to maintain story coherence. For example, a person who has been "tied_up" (which implies being "restrained") has restricted freedom and is unable to do most other

physical actions until the restriction is undone. Dynamic state-attributes are further discussed for observable actions.

- Finally, **Sentiments** are most significant in supporting a desired context and further described in section 2.4.1. Sentiments are "feelings" towards committed or anticipated actions.

3.3 Actions

For RoleModel, stories are driven by a sequence of actions. Each action is attributed to a character or some unbound character parameter. Actions performed by characters attribute state properties to the acting characters for two reasons: to maintain causal coherency (for instance, a man cannot do anything after he dies) and to satisfy role-preconditions (for instance, an aggressor must cause harm). From one state to its successor, actions modify temporal attributes of characters and determine conditions for role satisfaction. These attributes can be amplified, negated, or changed by contextual explanations, such as: sentiments, motivations, and reactions. For example, a man murders his best friend; however, the man was actually manipulated by a third sinister character. If his reaction to the act is the mourning of his best friend, he may have now been contextually unassigned the aggressor role, but regardless of context, a man still murders his best friend. In this case, kill or murder is the action committed.

3.4 Context

Context is critical to understanding the relationship between character traits, state-attributes, actions, and roles. Context requires its own category as it contains properties that are highly interrelated to the other conceptual components of RoleModel (roles, characters, and actions). Two types of contextual information are sentiments and properties of a committed action; however, traits and state-attributes are also aspects of contextual rules. Additionally, roles can be re-interpreted by contextual rules that are built from the properties described in the subsections below, in addition to traits and character state-attributes.

3.4.1 Sentiments

Sentiments represent motivational, observational, or reactionary beliefs of characters in RoleModel. These three types of sentiments distinguish responses prior, during, and after a given action and can be used to build a variety of models for desires and motivation for characters in particular circumstances. For instance, a reaction can turn into motivation for the subsequent act. An aggressive action for a victim could be explained as a reaction to an aggressor's action against the character. A regret sentiment can be attached to a victimized character after undertaking an aggressive action. Assertions made about a character's sentiment can negate, amplify, reduce, or change the implications of an act.

3.4.2 Properties of a Committed Act

Properties of a committed act determine changes made in the state-attributes of all characters affected, and are used to determine whether role constraints are satisfied. An integral property for roles such as victim and aggressor is the occurrence of harm. Properties such as *causes harm* require formal representation in order to accurately portray aggressor and victimhood. However, since roles are meant to control the flavors of variation, action properties must have negateable operations. Since harm determines aggressor and victimhood, harm must then be negatable (or undone), as specified in section 2.2. For example, consider a story in which a character that is harmed is forbidden to take the role of a victim by the author. In this case, the system explicitly elaborates the story by adding in actions that explain the uncharacteristic character behavior. Three examples of this could be: the character was tied up for protection from a greater harm, the character was an aggressor being restrained from causing a greater harm, or the ropes were weak and the action was instantly nullified. These would be examples of contextual properties of committed acts.

Character traits, character state-attributes, properties of actions, and sentiments are deeply related; therefore, a richer model of sentiments is necessary to support the construction or elaboration of stories with deep character roles.

4. PROTOTYPE IMPLEMENTATION

The scope of this prototype is to give a high-level understanding of how roles can influence the variation of a story or set of story constraints. For this system, roles, such as victim and aggressor, are simple functions of harm, and harm is simply a property of certain actions. The goal of building this system is to be able to discuss implications and future directions of modeling distinct background theory components (such as: role, motivation, happiness, and desire) to elaborate upon authorial preferences. Appropriately, the authoring of a story can be translated into logical statements. This implementation turns stories into satisfiability problems through LParse, taking a similar approach to Mueller's story understanding system [7] (though our background theory and motivation are very different).

4.1 Logical Abduction

Logical abduction is the diagnosis of "what must have been the case" to yield certain outcomes. The problems in story variation, elaboration, and explanation fit elegantly into this approach of model finding. RoleModel, most significantly, attempts to maintain role constraints. The evidence of a role-assignment is determined by its background knowledge of what it means to fill that role. Authorial goals are, then, easily represented as a partial model that yields satisfiable answer-sets, filling in the constraints with contextual and elaborative detail. The answer-sets represent generated stories that satisfy role constraints for our system. We allow the system to make assumptions about which traits a character has, which actions they take, and how they feel about them. Given these assumptions, the effects of the actions and the roles the characters take follows via deduction. Both abduction and deduction are computed simultaneously by our solving process. Constraints, determined by multiple types of authorial goals in the single framework, block certain kinds of assumptions from being made, shaping the space of stories that might result.

This answer set programming approach supports declarative modeling of background knowledge without having to commit to an imperative procedure. Story elements are easily translated and represented by symbols used to satisfy constraints. The solver is designed specifically to satisfy constraints, which translates well into interpreting and elaborating upon the story constraints input at generation time.

4.2 Background Theory

Our prototype models the background theory as highly interdependent components (roles, actions, traits, sentiments, etc.). Conceptually, these interdependencies are most easily divided into aspects of: roles, characters, actions, and context. These components are represented by predefined: traits, sentiments, dynamic attributes, roles, and action properties. Additionally, the background theory contains the axioms of the event calculus and a specification for what the system is allowed to abduce (the abducibles). The problem specification consists of: named characters and required/forbidden constraints on roles, traits, and actions (though, in the current prototype, action constraints don't let you specify time points of occurrence for simplicity).

The background theory in our prototype is motivated by the film *Rashomon* by director Akira Kurosawa. In Rashomon, the story of a murder and rape are retold from four different perspectives, with each perspective taking the same basic events (character actions) and using them to construct completely different interpretations, including different roles (who is the victim, who is the aggressor) motivations, etc. This subjective dependence of the meaning of events on observer perspective has been dubbed the “Rashomon Effect”. As modeling the Rashomon Effect for the purpose of creating a novel story generation framework is the initial motivation for RoleModel, we found it appropriate to base the initial background theory on the roles and characters found in the film *Rashomon*.

4.2.1 Roles

Roles are conceptualized as a top-level subdomain of our background theory, since it is the organizing concept for the generator. In our prototype, the roles consist of: aggressor, victim, and bystander.

By default roles constraints are satisfied if they meet the prescribed definition (or preconditions). These definitions are designed to be both straightforward, yet abstract enough to demonstrate significant variation. The role definitions are: aggressors must cause harm, victims must be harmed, and bystanders must perform no actions.

Actions, by definition, hold properties, and in this case, we define our roles as functions of harm; however, these default properties can be altered by context (described in 3.2.4). This allows the author to make intentional changes to the authorial constraints and produce meaningfully contrasting variations. For instance, if one character causes harm, but is not permitted to be an aggressor, then context is used to satisfy the role constraint.

4.2.2 Characters

Each character consists of a name and a set of traits and actions. For the prototype, characters can be assigned the traits: strong, trickster, or insane. These traits are used to help assign appropriate actions or derive appropriate explanations for actions (further discussed in 3.2.3 and 3.2.4 sections).

In addition to traits and actions, there are also state-attributes. The current possible attributes are: alive, manipulated, and restrained. Similar to traits, attributes determine valid actions afforded to a character and/or valid explanations to fulfill desired role constraints.

Character actions (or the properties of these actions), by default, determine role assignments. Roles, in and of themselves, are not parts of character, but are determined by actions of a character, in addition to contextual sentiments and details.

4.2.3 Actions

The actions available include: ties_up, tricks, kills, speaks_to, mourns, realizes, comforts. Inference rules constrain the subjects and objects of actions based on the character traits and state attributes.

- to **tie someone up** you must have relative strength over them (based on the strong/weak traits)
- only rational (non-insane) people can **trick** one another, and you have to be a trickster to do it
- you can’t **kill** dead people
- only insane people can **speak to themselves**
- you can only **mourn the dead** if they are not an aggressor
- **realizing** is a reflexive action only available to the currently manipulated
- **comforting** only makes sense with victims (or victims to be, as foreshadowing!)

The current traits are: strength, sanity, manipulation. Current state-attributes are: dead (or alive), restrained (or free), manipulated (or not). Traits are used to constrain the space of afforded actions, as are state-attributes of characters; although, state-attributes are a result of previous actions and can be changed by future actions, while traits do not change. In addition to altering the states of characters, actions also determine role constraint satisfaction. These properties of actions are, by default, implicitly applied and changed by contextual interpretations. Given that our current background theory is inspired by *Rashomon*, we focus on the roles of aggressor and victim, which are defined by actions that cause **harm**. Harm is a default property of the actions kills, ties_up, and tricks.

4.2.4 Context

Context is composed of story elements that are neither characters, roles, nor actions. They are assertions in regards to characters or actions which redefine role assignments. Without context, all roles are assigned based on the default action preconditions. For each action, our system has two sentiments: desire and regret. In the implementation, only the action of ties_up has an assertable contextual property or detail, weak_ropes– this contextual property can undo the harm that being tied up has.

Context can be used to re-enforce role satisfaction, but for the current prototype, is only used to explain undesired role assignments. We call this nullifying of roles. Victim and aggressor can be nullified by two means each:

Victim: harm can be nullified for the victim if there is a contextual detail (such as weak ropes) attached to a harming event [**property of action**], or harm can be nullified if the action was done to oneself and the action was desired (intentional suicide) [**sentiment**].

Aggressor: aggression can be nullified by feeling regret for an action as it happens [**sentiment**] or by being temporarily manipulated [**dynamic state-attribute**].

4.2.5 Logical Foundations
The event calculus provides causal structure for the story generator. For the event calculus, there are timepoints zero through t_max, where t_max can be overridden on the command line. Required actions must happen at some point in time, and forbidden actions must never happen. The values of dynamic attributes are linked to event happenings via the event calculus axioms. Finally, how events initiate and terminate attribute states is delegated to the definition of actions. The event calculus axioms are independent of the background theory for the story domain. Currently our event calculus axioms allow only one action to occur at a given time point (actions can't occur simultaneously) though we plan to lift this limitation in the future.

Other logical foundations of RoleModel are abducibles and integrity constraints, which are primitives of abductive logical programming. Abducibles define the set of symbols over which the solver can make assumptions, while integrity constraints prevent conflicting assumptions.

Our set of abducibles is:

- for each trait, a character can have or not have that trait
- exactly one event happens at each time point (but only possible ones happen)
- exactly one sentiment may arise for each event that happens, the sentiment is that of the subject
- any number of contextual details may be specified about events that occur

Our major integrity constraints for roles are:

- if it is true that a role is required for a character, then that role cannot also not be true for that character
- if it is true that a role is forbidden for a character, then that role must also be not true for that character

In our system, there are also integrity constraints for traits, roles, and actions.

5. EXAMPLE
In this section, we provide two examples of how RoleModel generates stories given the background theory plus generation-time constraints. RoleModel uses context and the ability to vilify, unvilify, victimize, and unvictimize characters with additional actions, in order to satisfy role constraints.

5.1 Inputs and Outputs
In Figure 2, the author names 3 characters, chooses a span of 4 time points, and designates role and action constraints. Since the input gives an unordered list of actions, the system will build the required actions into the timeline. Incidentally, in finding a model that satisfies the authorial constraints, the randomly chosen model also determined that both Bob and alice are victims (which was not specified nor precluded by the author). Despite what happens to a character or what a character does, roles need to be both applicable and reversible (or undone). For the prototype implementation of RoleModel, negating aggression is accomplished by either showing regret or showing that the alleged aggressor was actually tricked into causing harm. In the example from Figure 2, despite desiring the act of murder, Bob was clearly tricked by Eve, satisfying the precondition of forbidden-aggressor by deferring blame to the "trickster," Eve.

```
% INPUT
person(Alice).
person(Bob).
person(Eve).

t(0..3).

forbidden_role(aggressor,Alice).
forbidden_role(aggressor,Bob).
required_role(aggressor,Eve).

required_action(Alice,comforts,Bob).
forbidden_action(eve,kills,Eve).
required_action(Bob,kills,Alice).

% OUTPUT
happens(Eve,tricks,Bob,0).
happens(Bob,ties_up,Alice,1).
happens(Alice,comforts,Bob,2).
happens(Bob,kills,Alice,3).

sentiment(Eve,tricks,Bob,0,desire).
sentiment(Bob,ties_up,Alice,1,desire).
sentiment(Alice,comforts,Bob,2,regret).
sentiment(Bob,kills,Alice,3,desire).

has_trait(trickster,Eve).
has_trait(trickster,Alice).

has_role(victim,Bob).
has_role(victim,Alice).
has_role(aggressor,Eve).
```

Figure 2. Sample Input/Output.

```
% INPUT (Narrative Constraints Only)

forbidden_role(aggressor,Bob).
required_role(aggressor,Alice).
required_role(aggressor,Eve).

required_action(Alice,comforts,Bob).
forbidden_action(Eve,kills,Eve).
required_action(Bob,kills,Alice).

% OUTPUT
happens(Alice,tricks,Bob,0).
happens(Eve,ties_up,Alice,1).
happens(Alice,comforts,Bob,2).
happens(Bob,kills,Alice,3).

sentiment(Alice,comforts,Bob,2,desire).
sentiment(Bob,kills,Alice,3,regret).

has_trait(insane,Eve).
has_trait(trickster,Eve).
has_trait(trickster,Bob).
has_trait(trickster,Alice).

has_role(victim,Bob).
has_role(victim,Alice).
has_role(aggressor,Alice).
has_role(aggressor,Eve).
```

Figure 3. Input/Output with altered Narrative Constraint.

Figure 3 shows the same set of constraints (the person and timeline assertions have been elided), except Alice must also be an aggressor now. In both stories, Alice comforts Bob and Bob kills Alice. For the second variation, Alice must appear to be an aggressor, while Bob is still not allowed to have the aggressor role assigned. Having been tricked and the sentiment of "regretting" are used in this instance to negate Bob's aggressive action of killing, while in Figure 2, Bob was un-vilified because he was "tricked" or manipulated by another character. Regardless, both required actions still take place in both generated variations.

5.2 Exploring the Background Theory

Another example in using RoleModel is to give the minimal authorial constraints and produce stories that explore the background theory. This allows the story generator to create variations from a blank slate (no generation-time constraints). An example of this is shown in Figure 4 where larry tricks then ties up moe, then ties up curly, moe then realizes that he's been tricked, and tricks larry back.

```
% INPUT (No Narrative Constraints)
% OUTPUT
context(larry,ties_up,curly,2,weak_ropes).

happens(larry,tricks,moe,0).
happens(larry,ties_up,moe,1).
happens(larry,ties_up,curly,2).
happens(moe,realizes,moe,3).
happens(moe,tricks,larry,4).

has_role(aggressor,larry).
has_role(bystander,curly).
has_role(victim,larry).
has_role(victim,moe).

has_trait(strong,larry).
has_trait(trickster,larry).
has_trait(trickster,moe).

sentiment(larry,ties_up,moe,1,desire).
sentiment(moe,realizes,moe,3,desire).
sentiment(moe,tricks,larry,4,regret).
```

Figure 4. Sample Story with no Narrative Constraints.

6. DISCUSSION

Role Model's ability to generate stories under arbitrary generation-time constraints allows a single system to support multiple generation use cases, ranging from tabula rasa generation from the underlying logical model (classical story generation) through to generating variations of a highly specified skeleton story with specific role, event and sequencing constraints. The different use cases correspond to authorial preferences on degree of story variability, story quality, authorial effort, and authorial control.

We consider three use cases: tabula rasa generation, constrained generation, and generation of variations based on a partial story. Regarding the last use case, one of the goals of RoleModel is to support linear story authors who are not familiar with story generation technologies in creating and exploring variations on authored linear stories. To accomplish this, a story generator must be able to determine the *most significant* story aspects which can be varied without crossing outside the implicit generative space established by a partial linear story specification. This is one of the reasons for focusing on a theory of roles as a central representation within the generator; role variation is a highly salient narrative feature, but can respect a highly constrained linear story specification.

The following use-cases progress from maximum system control to maximum author control.

Tabula rasa generation. Tabula rasa generation utilizes a background theory as a generative model, generating stories with few authorial specifications. The RoleModel approach performs tabula rasa generation by finding assignments of abducibles that satisfy the background theory with no additional narrative constraints, or minimal narrative constraints, such as the number of characters and a timespan for the narrative. In terms of the examples in the previous section, this would correspond to having no inputs (narrative constraints) as in Figure 4.

Constrained generation. In constrained generation, the author can specify many constraints on role fillers, character traits, and even the appearance of specific events within the story, without locking down a specific linear sequence of events. Figure 2 and 3 in section 4 demonstrate this use-case through telling a story about a particular characters with various unknowns.

Generation based on a partial story. As the number of constraints grows, the author is able to specify a linear story that the system generates variations on. This supports authors who prefer to think in terms of fully specified linear stories. They can start with such a story, and then begin removing constraints, such as by replacing constants with variables. They can then iteratively explore the generative spaces defined as they incrementally add and remove constraints to their linear story. This involves elaborating on a critical event sequence among partially developed characters.

Further, by employing abductive logic programming, we have shown how to concisely resolve complex constraint structures in a story generator without having to commit to a particular procedure with distinct stages of generation.

7. FUTURE WORK

The current RoleModel implementation provides a proof of concept that a generator organized around formal models of roles embedded in an abductive logic framework can successfully generate stories subject to a variety of generation-time constraints. Future work will focus on: expanding the selection of possible roles, developing models of motivation and stronger models for context, and creating a larger dictionary of reusable actions, traits, and sentiments.

Currently, RoleModel only has roles of victim, aggressor, and bystander available. In order to demonstrate more expansive variation, other roles will be introduced and formally represented. Victim and aggressor fundamentally depend on harm. In creating new roles for the system, additional properties of actions will need to be identified and represented. In addition to creating more roles, this will also further abstract upon actions and action properties, as well as deepen the contextual rule set.

Contextual rules operate on traits, properties of actions, sentiments, and state-attributes to satisfy role constraints. Conceptually, these models can be organized in a variety of ways to suit the system. For RoleModel, the conceptual components are not individually defined in the code, which is also a result of using declarative answer-set logical programming vs. more imperative approaches. As these aspects of the formal model are further defined, the organization of the rules will be more easily universally applied to a variety of story spaces. Additional models for motivation are also necessary to develop a more sophisticated contextual rule system that can derive appropriate sentiments for characters.

In order to be able to show more universality of the RoleModel approach, more actions or a sufficient ontology of actions must be developed. The actions defined for RoleModel were taken from the *Rashomon*, and gave some sense of variation for victim and aggressor; however, to be able to satisfy future role definitions, new actions and properties of actions must be identified. With a better library of actions, there can be more varieties of story and a better understanding of the relationship between actions, causal implications of actions, and properties associated with actions.

By bringing roles to the foreground of RoleModel we hope to focus attention on how role-specific perspectives help create interesting narrative variations within a given domain and determine narrative meaning, in contrast to the more traditional emphasis on action causality.

8. REFERENCES

[1] Abelson, R. P. and Carroll, J. D. Computer Simulation of Individual Belief Systems. American Behavioral Scientist, p. 24. May 1965.

[2] Bringsjord, S., Khemalani, S., Arkoudas, K., McEvoy, C., Destefano, M., and Daigle, M. Advanced Synthetic Characters, Evil, and E*. Game-On, Vol 6. p. 31-39, 2005.

[3] Carbonell, J.G. POLITICS: Automated ideological reasoning. Cognitive Science: A Multidisciplinary Journal, Vol 2. Psychology Press. 1978.

[4] Cavazza, M. and Charles, F. and Mead, S. Agents' Interaction in Virtual Storytelling. Intelligent Virtual Agents, p. 156-170. 2001.

[5] Mateas, M. and Domike, S. and Vanouse, P. Terminal time: An ideologically-biased history machine. AISB Quarterly, Special Issue on Creativity in the Arts and Sciences, Vol 102. 1999.

[6] Montfort, N. and Perez y Perez, R.P. Integrating a Plot Generator and an Automatic Narrator to Create and Tell Stories. On Computational Creativity. 2008.

[7] Mueller, E. T. Understanding goal-based stories through model finding and planning. Intelligent Narrative Technologies: from the AAAI Fall Symposium, p. 95–101. Menlo Park, CA: AAAI Press. 2007.

[8] Perez y Perez, R. and Sharples, M. Three computer-based models of storytelling: BRUTUS, MINSTREL and MEXICA. Knowledge-Based Systems, Volume 11. 2004.

[9] Reidl, M. O. and Young, R. M. An Objective Character Believability Evaluation Procedure for Multi-agent Story Generation Systems. Intelligent Virtual Agents, p. 287-291. 2003.

[10] Shanahan, M. Prediction is deduction but explanation is abduction. Proceedings IJCAI, Vol 87. 1989

[11] Turner, S.R. MINSTREL: A Computer Model of Creativity and Storytelling. University of California at Los Angeles Los Angeles, CA, USA. 1993.

[12] Wardrip-Fruin, N. Expressive Processing: Digital Fictions, Computer Games, and Software Studies. MIT Press. 2009