Computer-assisted Authoring for Natural Language Story Scripts

Rushit Sanghrajka Disney Research
Wojciech Witoń Disney Research
Sasha Schriber Disney Research
Markus Gross Disney Research
Mubbasir Kapadia Disney Research, Rutgers University

Abstract

In order to assist scriptwriters during the process of story-writing, we have developed a system that can extract information from natural language stories, and allow for story-centric as well as character-centric reasoning. These inferencing capabilities are exposed to the user through intuitive querying systems, allowing the script writer to ask the system questions about story and character information. We introduce *knowledge bytes* as atoms of information and demonstrate that the system can parse text into a stream of knowledge bytes and use these mentioned reasoning capabilities through logical reasoning.

1 Introduction

Story-writing requires creative focus, as the writer needs to focus on making sure that their story is logical, and does not have any inconsistencies and plot-holes (Ryan 2009). Moreover, for stories that may take place in large pre-existing fictional story worlds, such as the *Harry Potter* world or *Star Wars* world, it becomes essential to maintain consistency with the existing laws of the world. For franchises like *Star Wars*, fact-checking is essential to ensure that there is no redundancy while introducing new characters or species of creatures which may have already had a minor appearance or reference in the past. Moreover, fictional universes like *Harry Potter* may have additional rules of their own, for example, the lack of use of electricity and technology in the Wizarding World (Rowling 2005).

Such book-keeping of story worlds often detracts story-writers from the creative story-writing process itself. Most computer-assisted writing softwares have features such as automatic spelling and grammar checking features. Applications such as Final Draft (Final Draft 1990) are widely used for screen-writing purposes for organization and efficiency in script and screenplay writing. However, commercially available options do not provide any reasoning or fact-checking capabilities on the story and characters.

Our goal is to provide an interactive intelligent system that can support screenwriters with feedback about the story by understanding the events and interactions in the narrative. The system should also allow them to interact with their story by asking questions, set up rules for their desired story world, and look at the belief-desire based conflicts being recognized by the system. Our current system supports scripts or screenplays, and is an emerging technology. Handling of unstructured stories like novels are exciting avenues of future exploration.

There are various challenges that exist to this problem. Natural language capabilities pose a challenge in information retrieval from complex stories: natural language understanding has a long way to go before being able to match the level of inferencing the human brain can make from reading a story. Another challenge lies in the ability to analyze, compare and sort information extracted from a screenplay or a script. The field of computational narratives has advanced greatly to represent narratives with the help of story graphs (Riedl and Young 2006), but these current data structures cannot be formed directly from a text-based story.

The central aspect to our proposed solution is the ability to extract meaningful information directly from the story itself, without any additional author supervision. We are able to understand characters, their beliefs and desires, interactions, and their relations with other characters, along with information about the narrative arc of the story. We introduce *knowledge bytes* to represent the information encapsulated in the script. Knowledge bytes can be defined as atomic structures which can represent a granular segment of information about the narrative. We also introduce a *cross-knowledge base reasoning approach* that is capable of reasoning across various character knowledge bases, the story knowledge, as well as knowledge about the story world. By building these different knowledge bases and sharing streams of knowledge bytes across them, the reasoning system can make inferences and respond to the user's queries in real-time during the process of storywriting.

2 Related Work

There are detailed studies that focus on extracting a knowledge base from stories, such as Scheherazade (Elson 2012) and ASM (Finlayson 2012), and even create

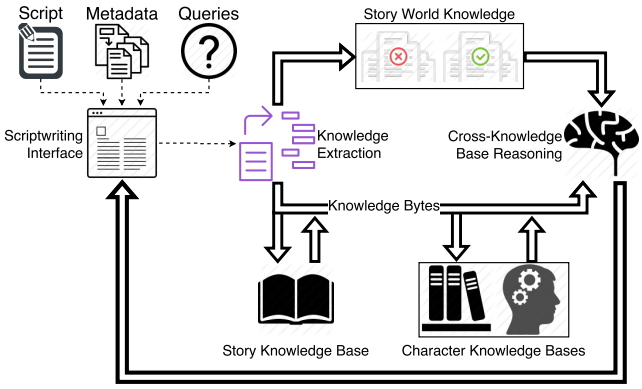

Figure 1: An overview of our framework.

fictional story worlds (Poulakos et al. 2015).

Boyang Li's work in generating narrative intelligence and determining causal links from a crowd-sourced narrative is an interesting approach to creating a script of a narrative (2012). Shoulson et al. (2013) discuss an event-centric planning approach to story creation for animation stories. There are also various planning based algorithms for story planning, which rely on a structure of story elements (Ware and Young 2011). Kapadia et al. focus on authoring of narratives with the help of Interactive Behavior Trees (2015).

Previous works have also focused on extracting information from scripts, focusing on various different approaches (Schank and Abelson 2013) and stories (Chaturvedi, Iyyer, and Daumé III 2017; Mateas and Stern 2003; Lehnert 1981; Goyal, Riloff, and Daumé III 2010; Valls-Vargas, Zhu, and Ontanon 2016). Sanghrajka et al. introduce LISA, a Lexically Intelligent Story Assistant, which uses logical inferencing for simple information extraction (2017).

Our work intends to expand upon the existing work by proposing a framework to extract event-centric and character-centric information, and perform reasoning with it in real-time. It serves to bridge the gap between these eloquent planning algorithms used for story planning, and other data structures focused on representing different aspects of narratives.

3 Framework Overview

A framework overview for the proposed system is shown in Figure 1.

Knowledge Extraction Our system takes input from the scriptwriting interface and performs natural language understanding to create a stream of knowledge bytes – the atoms of information – that serves as input to story world knowledge, story knowledge and character knowledge bases.

Knowledge Bases The story knowledge base stores all the knowledge bytes that are present in the script. Character knowledge bases – one for each character in the story – store snapshots of the story in a form of knowledge bytes as a representation of what each character perceives of the world around them and allow for reasoning about the story from character's perspective. Story world knowledge stores rules and error definitions specified by the user.

Cross-Knowledge Base Reasoning The knowledge reasoning system is the core of the framework as it gathers information from story world knowledge, story knowledge and character knowledge bases in form of a stream of knowledge bytes. It performs logical inference based on a specific query from the user and creates a response/feedback to send back to the user.

4 Knowledge Extraction

The scriptwriting interface accepts a script in natural language, which consists of a main story plot, i.e., actions and dialogs (described in Section 4.2). It also accepts metadata, i.e., rules and error definitions (described in Section 5.4), and background information about characters, i.e., stories occuring before the actual story, to be used for characters' reasoning capabilities.

In order to demonstrate the features and capabilities of our system, we decided to use it on a short snippet of story based on the Disney movie *Tangled* (Greno and Howard 2010). An excerpt of the script is presented in the left-most column of Figure 2. Part of our example script involves two characters, Flynn and Patchy, escaping from authority for stealing a crown. Flynn ends up stealing the crown and escaping, and in a second scene, another character named Rapunzel encounters Flynn, and takes the crown away from him. In the second scene, Rapunzel and her mother converse regarding Rapunzel's wishes to watch the lights in the sky. We use these examples throughout this paper to demonstrate our framework.

4.1 Knowledge Bytes

We introduce the concept of a Knowledge Byte in order to be able to store information from the script. A Knowledge Byte can be defined as the smallest unit of information about a narrative. It can represent any kind of information present in a script, such as action, dialog, or questions. Moreover, the knowledge byte also has support to store the location, time point, and the coordinates where it takes place, in order to process knowledge for spatial reasoning, which could be used to support newer forms of storytelling.

The knowledge byte β can be defined as a tuple in the following form: $\beta := <t, l, \Pi>$. t stands for the time point in the narrative at which this knowledge byte was first produced. l denotes the coordinates and location information of the knowledge byte to allow for spatial reasoning and location-based reasoning of knowledge bytes. Π is the tuple of parser labels, which have been defined in Section 4.2.

The importance of breaking information down into knowledge bytes is that these knowledge bytes represent

Story World Knowledge:
If CHARACTER steals OBJECT, then CHARACTER owns the OBJECT.
If CHARACTER carries OBJECT, then CHARACTER owns OBJECT.
If CHARACTER wants OBJECT, then CHARACTER wants to own the OBJECT.
If CHARACTER takes OBJECT, then CHARACTER owns OBJECT.

Character Background:
Patchy:
Patchy currently owns the crown
Flynn:
Flynn is a happy-go-lucky goon. He cares about money.
Rapunzel:
Rapunzel has always lived in the castle, and never left the tower. Since a child, Rapunzel wants to watch the lights in the sky. She wants to visit the lanterns, and see them in person. Rapunzel feels that the lanterns are meant for her.
Mother:
Mother kidnapped Rapunzel when Rapunzel was a baby, and has taken care of Rapunzel since then. Mother uses hair of Rapunzel to stay young. That is why Mother wants Rapunzel to be in the castle forever. She wants Rapunzel to stay home no matter what.

Script:
Flynn: This way! The guards are catching on to us!
Patchy: Go slow! I cannot run this fast!
Flynn and Patchy enter the scene from a house. They are panicked and running from the city. In the background, the guards can be heard. They go towards the end of the street. They realize that they are cornered.
Patchy: We're stuck, now what?
Flynn: Help me climb up the wall. Patchy, you stand by this door here!
Flynn climbs up the wall with Patchy's help. While climbing, he steals the crown from Patchy.
Flynn: I have the crown now! See ya!
Patchy: No! Come back, Flynn! I want the crown! It belongs to me! Arrgh!
Flynn exits scene as guards enter and catch Patchy.
New scene in Rapunzel's tower
Flynn enters the tower. Rapunzel is hiding behind a curtain. Flynn walks towards the center, and Rapunzel hits him with a pan. Flynn falls on the floor, unconscious.
Rapunzel sees his bag and opens it. She sees the crown.
Rapunzel takes the crown and hides it away.
Mother enters.
Rapunzel: I want to see the lanterns! May I go?
Mother: Rapunzel, you will not see the lanterns. How many times do I have to tell you dear, you must stay home.
Rapunzel: I don't want to stay home!

Story Knowledge:
Patchy owns crown.
<patchy, own, crown, false>
Flynn cares about money.
<flynn, care, ∅, false, about, money, ∅, ∅>
Rapunzel lives in castle.
<rapunzel, live, ∅, false, in, castle, ∅, ∅>
Rapunzel did not leave tower.
<rapunzel, leave, castle, true>
Rapunzel wants to watch lights in sky.
<rapunzel, watch, lights, false>
Rapunzel wants to visit lanterns.
<rapunzel, visit, lanterns, false>
Rapunzel wants to see the lanterns.
<rapunzel, see, lanterns, false>
Lanterns are meant for Rapunzel.
<lanterns, be, meant, false, for, Rapunzel, ∅, ∅>
Mother kidnaps Rapunzel.
<mother, kidnap, rapunzel, false>
Mother uses hair.
<mother, use, hair, false>
Mother wants Rapunzel to be in castle.
<rapunzel, be, ∅, false, in, castle, ∅, ∅>
Mother wants Rapunzel to stay home.
<rapunzel, stay, ∅, false, at, home, ∅, ∅>
Patchy cannot run fast.
<patchy, run, ∅, true, ∅, ∅, ∅, patchy >
Flynn enters scene from house.
<flynn, enter, scene, false, from, house, ∅, ∅>
Patchy enters scene from house.
<patchy, enter, scene, false, from, house, ∅, ∅>
Flynn runs from city.
<flynn, run, ∅, false, from, city, ∅, ∅>
Patchy runs from city.
<patchy, run, ∅, false, from, city, ∅, ∅>
Flynn goes towards street.
<flynn, go, ∅, false, towards, street, ∅, ∅>
Patchy goes towards street.
<patchy, go, ∅, false, towards, street, ∅, ∅>
Flynn is cornered.
<flynn, be, cornered, false>
Patchy is cornered.
<patchy, be, cornered, false>
Flynn is stuck.
<flynn, be, stuck, false>
Patchy is stuck.
<patchy, be, stuck, false>
Flynn climbs wall with help.
<flynn, climb, wall, false, with, help, ∅, ∅>
Flynn steals crown from Patchy.
<flynn, steal, crown, false, from, patchy, ∅, ∅>
Flynn has crown.
<flynn, have, crown, false>
Patchy wants to have the crown.
<patchy, have, crown, false, ∅, ∅, ∅, patchy>
Flynn exits scene.
<flynn, exit, scene, false>
Guards enter scene.
<guards, enter, scene, false>
Guards catch Patchy.
<guards, catch, patchy, false>
Flynn enters tower.
<flynn, enter, tower, false>
Rapunzel hides behind curtain.
<rapunzel, hide, ∅, false, behind, curtain, ∅, ∅>
Flynn walks toward center.
<flynn, walk, ∅, false, towards, cente, ∅, ∅>
Rapunzel hits Flynn with pan.
<rapunzel, hit, flynn, false, with, pan, ∅, ∅>
Flynn falls on floor.
<flynn, fall, ∅, false, on, floor, ∅, ∅>
Rapunzel sees bag.
<rapunzel, see, bag, false>
Rapunzel opens bag.
<rapunzel, open, bag, false>
Rapunzel sees crown.
<rapunzel, see, crown, false>
Rapunzel takes crown.
<rapunzel, take, crown, false>
Rapunzel hides crown away.
<rapunzel, hide, crown, false>
Mother enters.
<mother, enter, ∅, false>
Rapunzel wants to see lanterns.
<rapunzel, see, lanterns, false, ∅, ∅, ∅, rapunzel>
Rapunzel will not see lanterns.
<rapunzel, see, lanterns, true, ∅, ∅, ∅, mother>
Rapunzel stays at home.
<rapunzel, stay, ∅, false, at, home, ∅, mother>
Rapunzel does not want to stay at home.
<rapunzel, stay, ∅, true, at, home, ∅, rapunzel>

Character Knowledge Bases:
Patchy:
BELIEF: Patchy owns crown.
BELIEF: Patchy cannot run fast.
BELIEF: Patchy enters scene from house.
BELIEF: Patchy runs from the city.
BELIEF: Patchy goes towards street.
BELIEF: Patchy is cornered.
BELIEF: Patchy is stuck.
BELIEF: Flynn steals crown from Patchy.
DESIRE: Patchy wants the crown.
BELIEF: Crown belongs to Patchy.
BELIEF: Guards catch Patchy.
Flynn:
BELIEF: Flynn cares about money.
BELIEF: Flynn enters scene from house.
BELIEF: Flynn runs from the city.
BELIEF: Flynn goes towards street.
BELIEF: Flynn is cornered.
BELIEF: Flynn climbs wall with help.
BELIEF: Flynn steals crown from Patchy.
BELIEF: Flynn has crown.
BELIEF: Flynn exits scene.
BELIEF: Flynn enters tower.
BELIEF: Flynn walks toward center.
BELIEF: Rapunzel hits Flynn with pan.
BELIEF: Flynn falls on floor.
Rapunzel:
BELIEF: Rapunzel lives in castle.
BELIEF: Rapunzel did not leave tower.
DESIRE: Rapunzel wants to watch lights in sky.
DESIRE: Rapunzel wants to visit lanterns.
DESIRE: Rapunzel wants to see the lanterns in person.
BELIEF: Lanterns mean for Rapunzel.
BELIEF: Rapunzel hides behind curtain.
BELIEF: Rapunzel hits Flynn with pan.
BELIEF: Rapunzel sees bag.
BELIEF: Rapunzel opens bag.
BELIEF: Rapunzel sees crown.
BELIEF: Rapunzel takes crown.
BELIEF: Rapunzel hides crown away.
DESIRE: Rapunzel wants to see lanterns.
DESIRE: Rapunzel does not want to stay home.
Mother:
BELIEF: Mother kidnaps Rapunzel.
BELIEF: Mother uses hair.
DESIRE: Mother wants Rapunzel to be in castle.
DESIRE: Mother wants Rapunzel to stay home.
BELIEF: Mother enters.
BELIEF: Rapunzel will not see lanterns.
BELIEF: Rapunzel stays home.

Queries:
1. Query: Who owns the crown?
Script: Rapunzel
2. Query: Patchy, who owns the crown?
Patchy: Flynn
3. Query: Who all want to own the crown?
Script: Patchy, Flynn want to own the crown: desire contradiction
4. Query: Rapunzel, where do you live?
Rapunzel: Castle
5. Query: Flynn, who hits you?
Flynn: Rapunzel
6. Query: What does Patchy want?
Script: Crown
7. Query: Who does Mother kidnap?
Script: Rapunzel
8. Query: Who all want Rapunzel to stay home?
Rapunzel does not want to stay home, Mother wants Rapunzel to stay home: desire contradiction
9. Query: Who all think that Flynn owns the crown?
Flynn thinks Rapunzel owns the crown, Patchy thinks Flynn owns the crown: belief contradiction
Rapunzel thinks Rapunzel owns the crown, Patchy thinks Flynn owns the crown: belief contradiction

Figure 2: An excerpt of our script along with the Knowledge Bases is shown here. In the left-most column, the Story World Knowledge, along with character background and the script text is input. Story knowledge and the character knowledge bases are also shown, with the parser result tuples shown in the story knowledge. The sentences in the character knowledge systems are flagged to show which facts get processed as beliefs or desires. Various example queries are also shown in the end.

information in the forms of beliefs or desires, and can be stored across multiple characters' knowledge bases. This linking of knowledge bytes across various characters' knowledge bases allows for interesting inferences, which is discussed in Section 6. Moreover, the introduction of knowledge bytes as a data structure for handling information about stories allows for many useful operations. These knowledge bytes can be sorted and oriented based on different parameters. They can be aligned spatially with respect to locations as well as temporally, depending on the amount of information the user chooses to provide to the system through the input. Alternatively, the knowledge bytes can also be clustered based on the characters' knowledge, with some knowledge bytes being shared across multiple knowledge bases.

4.2 Parsing System

Knowledge bytes are extracted with the help of Natural Language Understanding on the script, consisting of actions and dialogs, using a Stanford CoreNLP framework (Manning et al. 2014). We introduce a parser result tuple $\Pi := < s, r, o, n, rm, ro, rq, sp >$, where the fields are defined as:

s : Subject (a noun)
r : Relation (a verb)
o : Object (often a noun)
n : Negation Flag (boolean)
rm : Relational Modifier (e.g. "from", "to")
ro : Relational Object (often a noun)
rq : Relational Question (e.g. "who", "where")
sp : Character speaking in a dialog

We mark empty fields as $\varnothing$, and can support a shorter representation where the rm, ro, rq and sp fields are assumed to be empty. Additionally, in a tuple, empty fields towards the end of the tuple can be discarded. Knowledge bytes are shown in Fig 2.

Co-reference resolution We assume that users input script text in segments consisting of one "thought", i.e., a set of logically connected sentences. A first step of parsing any segment involves applying co-reference resolution – it is focused on assigning the real names of actors/objects/places to personal pronouns ("he", "they", "it", etc.) based on previously analyzed sentences. An example input "Rapunzel wants to watch the lights in the sky. She wants to visit the lanterns, and see them in person." would be then translated into "Rapunzel wants to watch the lights in the sky. Rapunzel wants to visit the lanterns, and see the lanterns in person.". We use a neural-network approach for co-reference resolution.

Actions After applying the co-reference resolution, the text is then split into individual sentences, which are later tokenized – single words (tokens) are extracted. Each token is usually related to the others which is resembled in tree-like constituency and dependency graphs (Chen and Manning 2014). We used the latter ones in a form of Enhanced++ Dependencies using Enhanced English Universal Dependencies (Schuster and Manning 2016).

We extract subjects, relations and objects – *relation triples*. We created our own pipeline, extracting fields in parse result tuple Π as follows: s – forms *subj* dependency with r; r – usually a root of dependency tree; o – forms *obj* dependency with r; n – "true" iff r, o or ro has any *neg* dependency; rm – usually precedes ro and forms *case* dependency with it (e.g. "go **to** someone"); ro – usually forms *nmod* dependency with r (e.g. "go to **someone**"); rq – described in Section 6.1.

There are some exceptions to these rules. The most common case is while using a verb "to be", as it can have different meanings depending on the context – an auxiliary verb (*aux*), e.g., in continuous tenses, a passive auxiliary verb (*auxpass*), e.g., in passive voice, or a copula (*cop*), used mainly for describing s.

It is worth mentioning that o either may not be set or can be a verb, e.g., when r and o are connected by open clausal complement (*xcomp*). For sentences with more than one subject, relation or object several triples can be created, each consisting of one s, one r and one/no o. Some examples are presented in Table 1.

Each paragraph and each sentence in the paragraph is indexed, which is used to assign a proper time point t to knowledge byte β. All actions generated from one sentence have the same t and are believed to occur simultaneously.

Dialogs While analyzing dialogs we also fill the *sp* field. During co-reference resolution any usage of personal pronoun with lemma "I" is matched to the name of the character.

Confidence resolution While analyzing actions and dialogs we infer how confident a character is about some facts by checking for usage of one of the confidence words presented in Table 2, where each confidence word is assigned a value in a range from 0, i.e., improbable action, to 1, i.e., surety that an action happened (values are arbitrarily set and future work would focus on studying discourse to test accuracy). For example, the sentence "Patchy thinks that Flynn owns the crown" would result in creating a knowledge byte containing a belief "Flynn owns the crown" with a confidence of 0.6.

Desire resolution We distinguish between beliefs and desires. The latter are recognized by looking for words such as "want", "wish", "need" in a provided sentence, either in a script or a dialog. As a result, sentences "Rapunzel visits lanterns." and "Rapunzel wants to visit lanterns" would create similar knowledge bytes with the former resolved as Rapunzel's belief and the latter as Rapunzel's desire to visit lanterns. For desires, the confidence is assumed to be 1 because characters always are sure about their desires for simplicity.

5 Knowledge Bases

5.1 Knowledge Facts

The knowledge bases store information from the knowledge bytes β in the form of logical reasoning facts, known as Knowledge Facts, which are described below.

Belief facts ψ: These facts store the most important information from the knowledge byte, and store it as a belief. They take two forms: `belief(Id, s, r, o, n)`, or `belief(Id, s, r, o, n, rm, ro, rq).`

Desire facts δ: These facts store the same information as the belief facts, but they store the information as a desire. They also have two forms: `desire(Id, s, r, o, n)`, or `desire(Id, s, r, o, n, rm, ro, rq).`

Location facts λ: These facts store locations or scene information, and reference to the knowledge byte ID. They are of the form `location(Id, Location).`

Coordinates facts ω: The coordinates facts can store information about the spatial coordinates of the knowledge byte taking place in the system. They are of the form `coordinates(Id, X, Y, Z).`

Confidence facts χ: The confidence facts store the confidence level as a floating value from 0 to 1. They have the form `confidence(Id, Confidence).`

Time facts τ: Time facts allow us to build reasoning systems with temporal reasoning capabilities. Time facts take the form `timeof(Id, Time).`

5.2 Story Knowledge

The Story Knowledge Σ stores all the knowledge bytes that are present in the script, along with references to the various characters as well. Every knowledge byte from the script is fed to the knowledge base for the story knowledge, and this allows for reasoning on the information stored in this knowledge base as well.

5.3 Character Knowledge

We implement Character Knowledge bases Υ as a means for the application to form a knowledge base for

Table 1: Examples of Parsed Knowledge Bytes

Type	Sentence	Knowledge Byte
Continuous tense/no object	Rapunzel is hiding behind a curtain.	<rapunzel,hide,∅,false,behind,curtain>
Passive voice	The lanterns are meant for Rapunzel.	<lanterns,be,meant,false,for,rapunzel>
Description/two subjects	Flynn and Patchy are panicked.	<flynn,be,panicked,false>, <patchy,be,panicked,false>
Object as verb	Rapunzel enjoys watching the lights.	<rapunzel,enjoy,watch,false,∅,lights>
Negation in dialog	Mother: "Rapunzel, you will not see the lanterns."	<rapunzel, see, lanterns, true, ∅, ∅, ∅, mother>

Table 2: Confidence Words

Verb	sure	confident	know	state	say	think	feel	suppose	believe	assume	presume	expect
Confidence	1.0	1.0	0.8	0.6	0.6	0.6	0.4	0.2	0.2	0.2	0.2	0.1

the information possessed by each character. This allows the reasoning system to let each character work on their own set of beliefs and desires in the story world. The benefit of having separate character knowledge bases for each character is that it allows scriptwriters to ask questions to each character and gauge the difference in their responses based on the information that the character system possess. These knowledge bases are created transparently while the user is writing the story. This facilitates an interactive script writing process and does not disrupt creative flow of the writer.

Moreover, each character knowledge base stores knowledge bytes with some wrapper information which describes a relation between the character knowledge base and the knowledge byte. The relation-specific wrapper contains information about the character's confidence about the knowledge, the time point that the character learns about the information, and flags that denote whether the knowledge byte is a belief or a desire. Feedback from the character's reasoning system is also stored to provide back to the user interface.

5.4 Story World Knowledge

Story World Knowledge Ω stores rules and error definitions provided by scriptwriters in natural language. The former ones enable automatic inferencing of information about the story and characters, and the latter ones can be used to ensure consistency of the story. Rules and errors also use the concept of *types*, where the type fact `type(flynn, goon)` assigns a type "goon" to Flynn, so rules for the goon type can apply to Flynn as well (Sanghrajka et al. 2017).

Rules and errors, which the user types in natural language, are translated into the structure required by knowledge reasoning system described in Section 6. During parsing we make use of TokensRegex framework (Chang and Manning 2014) to create regular expressions over (a sequence of) tokens. We can check for properties of single tokens extracted by TokenizerAnnotator – lemma, named entity and a part of speech. The main patterns used for regular expressions are Type pattern, Inference pattern and Error pattern, shown in Table 3. Sentences with subject and/or object in all UPPERCASE make general rules for every subject and/or object, otherwise we create rules for specific subject and/or object. We can combine patterns, use coreferences, resolve desires and negations, include information about location and a time of an action. Example sentences with created rules are shown in Table 4.

5.5 Knowledge Base Construction

For every knowledge byte, we extract $\psi, \lambda, \omega, \chi$ and τ, and send it to the system's story knowledge Σ. Then we look at the s, o, ro variables to check if any of them refer to a character name. For all the characters referenced, we extract either the ψ or the δ depending on the parsing system's information, and then add it to each referenced character's knowledge base Υ_C (where C is the character) along with the λ, ω, χ and τ.

6 Cross-Knowledge Base Reasoning

6.1 Knowledge Query Extraction

As soon as a story is written and knowledge bases are created, a scriptwriter can type questions in natural language to get information stored in either story or character knowledge bases. We analyze questions in similar way as actions, filling *rq* field of parser result tuple Π with relational question words, such as interrogative pronouns "who", "what", and pro-adverbs "when", "where" etc. A question κ can be defined as a set of incomplete knowledge bytes, where the *rq* field contains a question word, and there are one or more fields that contain question marks. An example question "Patchy, who owns the crown?" queries Patchy's knowledge base with a knowledge byte. Equations 1 through 3 show the query and knowledge byte. Another example question "Who all have contradictory desires regarding owning the crown?" creates a query with multiple knowledge bytes, shown in equations 4 through 8.

$$\kappa_1 = \{\beta_1\} \tag{1}$$

$$\beta_1 =< t_1, l_1, \Pi_1 > \tag{2}$$

$$\Pi_1 =< \mathbf{?}, own, crown, false, \varnothing, \varnothing, who, \varnothing > \tag{3}$$

$$\kappa_2 = \{\beta_2, \beta_3\} \tag{4}$$

$$\beta_2 =< t_2, l_2, \Pi_2 > \tag{5}$$

$$\beta_3 =< t_3, l_3, \Pi_3 > \tag{6}$$

$$\Pi_2 =< \mathbf{?}, own, crown, \mathbf{?}, \varnothing, \varnothing, who, \varnothing > \tag{7}$$

$$\Pi_3 =< \mathbf{?}, own, crown, \mathbf{?}, \varnothing, \varnothing, who, \varnothing > \tag{8}$$

Table 3: Regular Expressions for Story World Knowledge Patterns

Pattern	Regular Expression
Type	`(?$subj [tag:/NN.*/]+) [lemma:/be/] [tag:/DT.*/]+ [!tag:/NN.*/]* (?$type [tag:/NN.*/]+)`
Inference	`/if/ (?$cond [!lemma:/then|,/]+) /then|,/+ (?$res []+)`
Error	`/show/ [tag:/DT.*/]* /error/ (?$err []+) /if/ (?$cond []+)`

Table 4: Examples for Story World Knowledge Patterns

Sentence	Rule
Flynn is a happy-go-lucky goon.	`type(flynn, goon).`
If PERSON enters scene at time T1 and PERSON exits scene at time T2, then PERSON is onstage from T1 to T2.	`type(X, onstage, T1, T2) :- belief(Id1, PERSON, enters, scene, false), timeof(Id1, T1), belief(Id2, PERSON, exits, scene, false), timeof(Id2, T2).`
If PERSON steals OBJECT, then PERSON owns OBJECT.	`belief(Id1, PERSON, own, OBJECT, false) :- ↪ belief(Id2, PERSON, steal, OBJECT, false).`
Show error "Cannot steal something they already own" if PERSON has OBJECT and PERSON steals it.	`error(Id1, Id2, 'Stealing something owned') :- ↪ belief(Id1, PERSON, have, OBJECT, false), ↪ belief(Id2, PERSON, steal, OBJECT, false).`

6.2 Reasoning Approach

When a query is received in the knowledge reasoning system, the reasoning system performs two actions. The reasoning system first uses logical reasoning in order to generate a set of knowledge bytes which are responses to the query provided. For implementation, we use the GNU Prolog for Java as our logical knowledge reasoning system (Diaz and Codognet 2000). After the resultant knowledge bytes are constructed from the logical reasoning system, we then perform cross-knowledge base reasoning to understand possible relations for each pair of resultant knowledge bytes. We then generate the result. The resultant function $\alpha(\kappa)$ takes in a query κ and produces a resultant set of completed knowledge bytes $\{\beta_1^*, ...\beta_N^*\}$. R(i,j) gives the relationship between a pair of knowledge bytes β_i^* and β_j^*in the resultant set.

Logical Reasoning. All the character knowledge bases and the story knowledge base have their own logical reasoning environment. The facts which contain the missing values in the βs from the κ are queried to the logical environment, and the resultant β^* is constructed. In our example, Patchy's knowledge base believes that Flynn owns the crown. The resultant β^* contains the completed information, as shown in equations 9 and 10.

$$\beta_1^* =< t_1, l_1, \Pi_1^* > \quad (9)$$

$$\Pi_1^* =< flynn, own, crown, false, \varnothing, \varnothing, \varnothing, \varnothing > \quad (10)$$

Determination of Relationships. Reasoning across knowledge bases involves comparison of knowledge bytes in the various knowledge bases in order to discern whether they have a similarity or a contradiction. Looking for a possible connection between knowledge bytes is the core for the system's ability to infer across knowledge bases. Using WordNet (Miller 1995), we extract synonyms and antonyms for the relations used in knowledge bytes, and compare the knowledge bytes to form relations between them.

Time points and confidence measures are also taken into account to analyze relationships between knowledge bytes. By default, we decided to assign a 0.8 weight to confidence and 0.2 weight to time, in order to make the impact of confidence stronger than time. In some cases, for some pairings of knowledge bytes, there is the ability to specify custom values for the weights for confidence and time, in cases where one may require different level of impact of the difference in time or confidence of the two knowledge bytes on the possibility that the two knowledge bytes are indeed related. Let the time and confidence values for a knowledge byte be denoted by T_x and C_x respectively. We assume that N is the final time point of the script.

A relational factor θ' is calculated based on the lexical comparison of two knowledge bytes, which is a measure of how related they are. θ' varies from -1 to $+1$, with -1 denoting contradictory relation, a $+1$ denoting similarity between the knowledge bytes, and a θ' closer to 0 representing that the knowledge bytes may be unrelated. The algorithm to calculate the relational factor is shown in Algorithm 1. Once we find that the two knowledge bytes are related, then we compare their links, to obtain the following types of relationships:

Input : A pair of Knowledge Bytes β_1^* and β_2^*;
Output: Relational Factor θ';
if $\Pi_1 = \Pi_2$ *or* $(r_1 \simeq r_2, s_1 = s_2, o_1 = o_2)$ **then**
  $\theta \leftarrow 1$;
else if r_1 *and* r_2 *are antonyms,* $s_1 = s_2, o_1 = o_2$ **then**
  $\theta \leftarrow -1$;
else if $r_1 \simeq r_2, o_1 = o_2, s_1 \neq s_2$ **then**
  $\theta \leftarrow -0.5$;
else if $r_1 \simeq r_2, s_1 = s_2, o_1 \neq o_2$ **then**
  $\theta \leftarrow 0.5$;
else
  $\theta \leftarrow 0$, which implies they are likely to be unrelated
$\theta' = \frac{|C_2 - C_1| C_W + \frac{|T_2 - T_1|}{T_N} T_W}{\theta}$
return θ'.

Algorithm 1: Relational factor calculation

- If β_1^* and β_2^* are both desires and $0 < \theta' < 1$, we consider this to be a similarity in desires between

two character knowledge bases Υ_1 and Υ_2.

- If β_1^* and β_2^* are both desires and $-1 < \theta' < 0$, we consider this to be a contradiction or a competition in desires between two character knowledge bases Υ_1 and Υ_2.
- If β_1^* and β_2^* are both beliefs and $0 < \theta' < 1$, we consider this to be a similarity in beliefs between two character knowledge bases Υ_1 and Υ_2.
- If β_1^* and β_2^* are both beliefs and $-1 < \theta' < 0$, we consider this to be a contradiction in beliefs between two character knowledge bases Υ_1 and Υ_2.
- If β_1^* and β_2^* are a belief and a desire and $\theta' \neq 0$, we consider this to be a misconception between two character knowledge bases Υ_1 and Υ_2.

In our example, for the query κ_2 in Section 6.1, a resultant response would seeing contradictions in desires regarding the ownership of the crown among Flynn and Patchy. Equations 11 through 14 show the resultant knowledge bytes β_2^* and β_3^*, and Equation 15 shows the relationship R(2,3) flagged as "contradictory desires".

$$\beta_2^* =< t_2, l_2, \Pi_2^* > \text{from Flynn's } \Upsilon \text{ as desire} \quad (11)$$

$$\beta_3^* =< t_3, l_3, \Pi_3^* > \text{from Patchy's } \Upsilon \text{ as desire} \quad (12)$$

$$\Pi_2^* =< flynn, own, crown, false, \varnothing, \varnothing, \varnothing, \varnothing > \quad (13)$$

$$\Pi_3^* =< patchy, own, crown, false, \varnothing, \varnothing, \varnothing, \varnothing > \quad (14)$$

$$\text{R(2,3)= "contradictory desires"} \quad (15)$$

7 Application

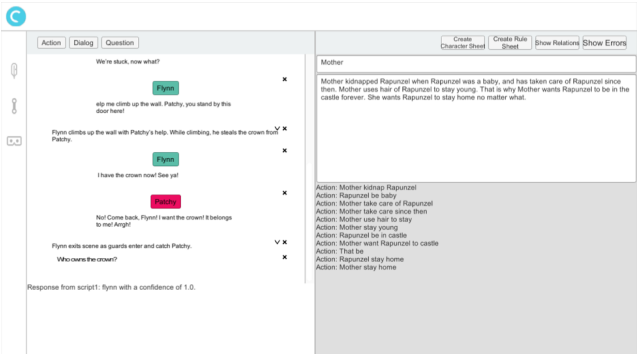

Figure 3: An image of the prototype we developed. The left window contains the script. The question asked by the user is "Who owns the crown?", and the script responds with "Flynn with a confidence of 1.0". On the right side, the user has entered information about the mother to the character base for the mother. The parsed information can be seen in the bottom right window.

Our proposed framework contributes to emerging technology for assisting story-writers real-time. A scriptwriter would start by adding the story world knowledge to the system, which defines the rules for the story world. The scriptwriter would then proceed to write his story, using the system as a guide for reference. Additionally, the system ensures that the narrative is within the bounds defined by the story world. The automatic extraction and construction of the knowledge bases enables the possibility to interact with the story and characters during story creation.

We performed internal experiments with our system with scripts of moderate complexity. An excerpt of an example script has been shown in Figure 2, and various examples of different features have been discussed throughout this paper. Additionally, an image of our prototype system developed is shown in Figure 3. Our system is able to autonomously construct the various knowledge bases from the script, and allows authors to specify the rules for the world in which the story occurs. The character knowledge bases have information specific to the character, and this encapsulation of information can be clearly observed in the different responses. Referring to the queries in Figure 2, the question "Who owns the crown?" is directed to the story in question 1 but to the character of Patchy in question 2. A difference in the responses can be observed, due to Patchy's character knowledge base having information known to his character. We were also able to perform cross-knowledge base reasoning, where we asked the systems various questions and received expected responses. For example, in the script shown, questions 3, 8, and 9 are questions which involve looking for relations across multiple knowledge bytes, which can be seen in the results obtained.

8 Conclusion

In proposing a new structure for representation and reasoning in narratives with the help of knowledge bytes, knowledge bases, rules and errors definition, we set up ground for various directions in natural language understanding and linguistic reasoning, assisted storywriting, and computational narratives.

Our application demonstrated that the introduced concepts work well for an example story, but we are aware that the implementation still needs some improvements before becoming a solution ready for deployment. The reasoning capabilities of our system are highly dependent on the capabilities of the mechanism analyzing rules to be able to completely reason about the story world knowledge. While this can be quite specific for story worlds like *Star Wars* or *Toy Story*, there is still a need for adding world knowledge rules. Automatic rule generation is an avenue that needs to be explored and a challenge that needs to be solved to make the creation of story world knowledge less cumbersome.

The future goals include analyzing more complicated (real) movie scripts and creating advanced rules and error definitions. We have implemented the theoretical concepts proposed in this work and are building towards a large-scale deployment of the tool and intend to perform detailed user studies in the future.

References

Chang, A. X., and Manning, C. D. 2014. TokensRegex: Defining cascaded regular expressions over tokens. Technical Report CSTR 2014-02, Department of Computer Science, Stanford University.

Chaturvedi, S.; Iyyer, M.; and Daumé III, H. 2017. Unsupervised learning of evolving relationships between literary characters. In *AAAI*, 3159–3165.

Chen, D., and Manning, C. 2014. A fast and accurate dependency parser using neural networks. 740–750.

Diaz, D., and Codognet, P. 2000. The gnu prolog system and its implementation. In *Proceedings of the 2000 ACM Symposium on Applied Computing - Volume 2*, SAC '00, 728–732. New York, NY, USA: ACM.

Elson, D. 2012. *Modelling Narrative Discourse*. Ph.D. Dissertation, Columbia University.

1990. Final draft. `http://www.http://finaldraft.com`.

Finlayson, M. M. A. 2012. *Learning narrative structure from annotated folktales*. Ph.D. Dissertation, Massachusetts Institute of Technology.

Goyal, A.; Riloff, E.; and Daumé III, H. 2010. Automatically producing plot unit representations for narrative text. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, 77–86. Association for Computational Linguistics.

Greno, N., and Howard, B. 2010. *Tangled*. Moore, M. and Levi, Z.: Walt Disney Animation Studios.

Kapadia, M.; Falk, J.; Zünd, F.; Marti, M.; Sumner, R. W.; and Gross, M. 2015. Computer-assisted authoring of interactive narratives. In *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games*, i3D '15, 85–92. New York, NY, USA: ACM.

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

Li, B.; Lee-Urban, S.; Appling, D. S.; and Riedl, M. O. 2012. Crowdsourcing narrative intelligence. *Advances in Cognitive Systems* 2(1).

Manning, C. D.; Surdeanu, M.; Bauer, J.; Finkel, J.; Bethard, S. J.; and McClosky, D. 2014. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, 55–60.

Mateas, M., and Stern, A. 2003. Façade: An experiment in building a fully-realized interactive drama. In *Game developers conference*, volume 2.

Miller, G. A. 1995. Wordnet: A lexical database for english. *COMMUNICATIONS OF THE ACM* 38:39–41.

Poulakos, S.; Kapadia, M.; Schüpfer, A.; Zünd, F.; Sumner, R. W.; and Gross, M. 2015. Towards an accessible interface for story world building. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*.

Riedl, M. O., and Young, R. M. 2006. From linear story generation to branching story graphs. *IEEE Computer Graphics and Applications* 26(3):23–31.

Rowling, J. K. 2005. Harry Potter and the Goblet of Fire.

Ryan, M.-L. 2009. Cheap plot tricks, plot holes, and narrative design. *Narrative* (1):56.

Sanghrajka, R.; Hidalgo, D.; Chen, P. P.; and Kapadia, M. 2017. Lisa: Lexically intelligent story assistant. *Proceedings of the 13th Artificial Intelligence and Interactive Digital Entertainment Conference*.

Schank, R. C., and Abelson, R. P. 2013. *Scripts, plans, goals, and understanding: An inquiry into human knowledge structures*. Psychology Press.

Schuster, S., and Manning, C. D. 2016. Enhanced english universal dependencies: An improved representation for natural language understanding tasks. In *LREC*.

Shoulson, A.; Gilbert, M. L.; Kapadia, M.; and Badler, N. I. 2013. An event-centric planning approach for dynamic real-time narrative. In *Proceedings of Motion on Games*, 121–130. ACM.

Valls-Vargas, J.; Zhu, J.; and Ontanon, S. 2016. Error analysis in an automated narrative information extraction pipeline. *IEEE Transactions on Computational Intelligence and AI in Games*.

Ware, S. G., and Young, R. M. 2011. Cpocl: A narrative planner supporting conflict. In *AIIDE*.