LISA: Lexically Intelligent Story Assistant

Rushit Sanghrajka, Daniel Hidalgo, Patrick P. Chen and Mubbasir Kapadia
Department of Computer Science
Rutgers University

Abstract

This paper serves as an introduction to building an assistive tool for story writers. Our tool, Lexically Intelligent Story Assistant (or LISA), aims to assist story writers by providing real-time feedback on lexical inconsistencies in the story. LISA analyzes the narrative and builds a knowledge base, using artificial intelligence to make inferences and point out errors in the narrative. Moreover, it also allows the user to interact with the system by querying the knowledge base in natural language form. This tool shows that it is possible to create a database for a narrative and use artificial intelligence to improve authoring of stories.

1 Introduction

Many word processors come with the ability to detect misspelled words and grammatical mistakes, but none come with the functionality to detect plot-holes and logical errors in narrative text. It is difficult to avoid contradictions when multiple authors collaborate on one story. It is also difficult to remember story specifics when the author resumes his work after a span of time. In such cases, it becomes a chore to go back and forth in the story looking for specifics about the narrative.

Extraction of story knowledge from narrative text can allow inferences at any state of the story as well as verification for new text to be consistent with the existing story world. Besides making story writing an easier task, this allows for groups of writers to maintain consistency as they contribute. Moreover, there are "problem-solving situations where humans suffer from cognitive overload, failing to effectively monitor all available information"(Laffey et al. 1988). This can be applied towards the domain of writing stories as well, and a tool that stores story knowledge can serve as a handy solution.

The primary challenge to finding a solution is extracting important facts from the natural language text. It also needs to record and maintain the veracity of facts even when the user modifies any part of the text. The third challenge involves inferring implicit facts about agent actions, knowledge and capabilities. The user should also be able to interact with the information by accessing and querying it in natural language form. Finally, the solution must reduce the amount of work required by the user while providing these capabilities.

The result of this research study is LISA, a *lexically intelligent storytelling assistant* application. LISA will allow the user to extract and verify story world knowledge while writing. LISA performs this task in real-time, while displaying errors with a description on the interface. LISA will also provide an interface for querying the story facts processed.

LISA exemplifies the bare-bones characteristics of a word processor with respect to analyzing narrative text. LISA allows the user to create consistent narratives with the aid of artificial intelligence methods. It also serves as a prototype for a flagging interface and engine, allowing for real-time AI-driven feedback during the process of story writing. Additionally, it also demonstrates a questioning interface and engine, allowing users to ask questions on story knowledge and receive AI-driven responses.

LISA allows the user to modify previous sentences which can change the entirety of the story knowledge base. Users can also resume or edit an existing story while maintaining the story knowledge base consistency. Additionally, the questioning interface serves as a debugging tool to verify the correctness of the story knowledge base. Moreover, LISA uses the information from the natural language story sentences and represents them as a tuple of their natural language parts of speech (Toutanova et al. 2003).

The main contribution of this paper is to introduce LISA, and describe the process involved in building an intelligent agent which can process natural language stories, make inferences about the information it processes, and interact with the user in real-time during the process of building stories. The interaction is both ways: it can provide feedback and point errors in the story, and the user can ask questions about the story.

2 Related Work

Extracting information from stories requires natural language parsing of sentences and coreference resolution. LISA uses the Stanford Core NLP toolkit (Manning et al. 2014), dependency parser (Chen and Manning 2014) and coreference resolution system (Lee et al. 2013).

The concept of breaking down sentences to elements has also been previously explored. (Carlson, Marcu, and

Okurowski 2003) use the concept of Elementary Discourse Units (EDUs) while building a discourse-tagged corpus in the Rhetorical Structure Theory framework. (Angeli, Premkumar, and Manning 2015) also break a sentence into "relation triples" by building three relevant clauses. Their approach provides more informative triplets. (Kubler, McDonald, and Nivre 2009) also perform dependency parsing using groupings. LISA also uses the concept of *tuples* to break down its sentences into triplets.

The use of logical programming for analysis and fact-checking has been demonstrated previously. (Bhattacharya and Neamtiu 2011) demonstrate that a Prolog-based framework can be elegant and powerful for real-world integrated data, and can be used for development and empirical analysis tasks. Moreover, implementations of Prolog for other languages, such as GNU Prolog, enable Prolog to be used from other languages as well (Diaz and Codognet 2000). LISA uses the GNU Prolog system for processing and inferring, and provides fact-checking capabilities.

(Chen and Fahlman 2008) built the Scone knowledge base, a dynamic mental context network. (Ginsberg 1988) proposes ways to check knowledge bases for redundancy and inconsistency. (O'Leary 1998) discusses various knowledge base systems and how they're driven by AI technologies. (Laffey et al. 1988) describe the requirements and limitations of real-time knowledge base systems. (Ciampaglia et al. 2015) build information networks or knowledge graphs, check facts against them and report that it is more efficient and scalable than expert fact-checking.

There are also existing implementations for rule-based systems. (Hayes-Roth 1985) discusses that rule-based systems have many benefits, and automate problem-solving in many scenarios. (Suwa, Scott, and Shortliffe 1982) discuss the importance of rule-based systems being complete and consistent. They show the importance of tagging errors and preventing errors from being added to a knowledge base. LISA uses artificial intelligence in the form of a rule-based system, checking for errors as each error is an inference-based rule.

David Elson's thesis (Elson 2012) focuses on extracting a knowledge base from stories, and works with Scheherazade to understand the meaning of sentences. While Elson's work greatly contributes in the domain by providing valuable ways to build a story world, it relies largely on the user to verify the meanings it understands.

Melissa Roemmele (Roemmele 2016) uses neural networks to predict stories, and hence assist the user in writing stories by giving a possible scenario after the current sentence. Her application *Creative Help* is a story assisting tool similar to LISA, yet it differs in functionality by predicting stories rather than checking the existing story for inconsistencies. Josep Valls-Vargas et al. introduce a system *Vox* which performs information and character role extraction (Valls-Vargas, Zhu, and Ontanon 2016). This work is highly useful for extracting information from the story regarding characters and events, and would extend LISA to be used along with a character-centric system.

In Smith and Witten's *A Planning Mechanism for Generating Story Text* (SMITH and WITTEN 1991), a general structure was defined to hold all information on a story world. A connectivity relation called the "Cango rule" was used as a way of enforcing causal links.

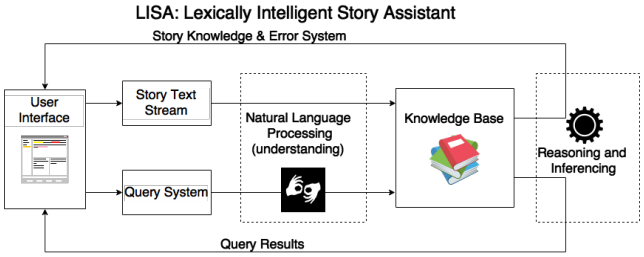

Figure 1: Information flow in LISA

Boyang Li's work in generating narrative intelligence and determining causal links from a crowd-sourced narrative is also an interesting approach to creating a script of a narrative (Li et al. 2012). Michael Mateas' work in interactive dramas such as *Facade* also serves as notable examples of using narrative intelligence in drama and stories (Mateas and Stern 2003; 2004). Shoulson et al. (Shoulson et al. 2013) discuss an event-centric planning approach to story creation for animation stories. Kapadia et al. (Kapadia et al. 2015) talks about authoring narratives through interactive behavior trees. One of our goals with LISA is to allow for story world building for text-based stories, and these works form an inspiration for LISA.

3 Framework Overview

Figure 1 shows the flow of the system as text stream is processed in real-time to give the user immediate feedback. LISA is the system that bridges user input to the back-end knowledge base, using natural language parsing and inferencing tools to provide further insight to the user.

Natural Language Processing. LISA allows the user to communicate in natural language, and for this purpose, natural language processing becomes a crucial element in the process of assisting story writers. All the information stored in the text is analyzed and extracted to consolidate entities and understand relationships through entities (Carlson, Marcu, and Okurowski 2003). Natural language processing also allows the user to ask questions in natural language, which are processed into a specific format used to store and query the knowledge base. The process is described in Section 4.

User Interactions. Users can interact with LISA through the narratives that they type in the story text. Interaction with the application uses two systems. LISA's **query system** allows the user to interact with their own story by asking questions to it. It can then query the system's knowledge base for the result and provides the user with an answer based on the question. The **error system** flags possible errors in the user's story. Each time the user completes a sentence in the front-end, LISA's back-end determines whether the sentence causes any logical inconsistencies. As a brief overview, this algorithm considers the maximum self-consistent set of "known" facts and determines whether ex-

tending this set to include facts from the examined sentence would logically lead to the existence of an error. Section 5 provides detailed information on this. Figure 2 displays the various screens based on the user's interactions.

Knowledge Representation and Reasoning. LISA keeps track of information about the story and hence builds a knowledge base regarding the events that have taken place in the story. The knowledge base can store not only facts, but it also stores a set of inferencing rules which can be queried by the query system. Using the set of facts and rules in the knowledge base, LISA runs an inferencing algorithm in the back-end to produce second-order logic. Reasoning is also used in a variety of ways to automatically detect inconsistency. Section 5 describes the working of the knowledge base in detail.

4 Natural Language Processing

Natural Language Processing is an essential feature, and all information that is input to the application is processed before it gets stored or queried to LISA's knowledge base. LISA's natural language processing features enable users to write stories in natural form, hence eliminating the need for a new syntax to be developed for story-writing.

As depicted in figure 1, the user interface accepts inputs in two forms: the story text stream, as well as the query system. The user's story text is processed every time the user enters new text or edits an existing sentence, hence allowing for real time analysis and processing. The query system allows the user to ask questions and then processes the question into a *tuple* and queries the knowledge base.

4.1 Story Text Stream Processing

The story text stream is processed into a number of *tuples*. *Tuples* can be defined as a sequence of words that form a block of information for LISA's story knowledge base. LISA's natural language processor presently builds *tuples* out of dependency-parsed groups of sentences (Kubler, McDonald, and Nivre 2009; Chen and Manning 2014). The example in Fig 3 highlights the different *tuples* formed. The following dependency parsings form *tuples*:

1. Noun subjects (*nsub tags*) are paired with direct objects (*dobj tags*) when they have a common verb. The *tuples* formed through this grouping have three words in them, and form *action tuples*, since they describe an action taking place in the story.
   Morevoer, these *tuples* can be considered to be of the form *afforder-affordance-affordee*. The afforder and affordee are passed on to the list of entity objects for LISA to take into consideration.
2. Noun subjects (*nsub tags*) are paired with copula (*cop tags*) and connected with the common tag. The pairings are used to form two word *tuples*, and signify a state or an attribute to a story entity.
   These *tuples* form *object-state* pairs and get passed on to the story knowledge base. Moreover, the object in this *tuple* is passed on to LISA's entity objects for tracking them.
3. Co-ordinations (*cc tags*) and conjunctions (*conj tags*) are processed by LISA to understand and break down the sentence to form multiple *tuples* of the types mentioned above.
   These tags do not contribute by forming their own *tuples*, but allow for efficient processing of compound sentences so that multiple *action tuples* can be extracted from the sentence by LISA.
4. Phrasal verb particles (*compound:prt tags*) are paired with noun subjects (*nsub tags*. These pairings form three words, which describe an action, but have no affordee specifically. These tags are appended to form three word *action tuples*, with the action being a phrasal verb, and the affordee being blank.
   An example of a *tuple* generated from this kind of parsing is shown in Fig 3. There is no affordee, and *down* needs to be processed and passed into the knowledge base for LISA to make sense of the information. The afforder in this *tuple* is also passed into the user interface as an entity object.
5. Adjectival modifier (*amod tags*) are also processed by LISA, and are used to create tuples that simply act as adjectives to an object. They're processed into two-word *tuples* and passed into the story knowledge base.
   Each *tuple* of this type also has an object, which is passed into LISA's entity objects list.

LISA's current implementation allows for these dependency parsings to be processed, and also for more dependency parsings to be processed in the future, further strengthening LISA's natural language processing capabilities.

4.2 User Queries

LISA allows the user to query the knowledge base by asking questions in natural language. The system assumes that the user will ask a question involving one of the 8 interrogative words: who, what, when, where, why, how, did, or does. Using this set of keywords, a tuple can be formed using the dependency parsings from above, which is used later to index the knowledge base to retrieve information. These tuples are then queried to the knowledge base and results are displayed to the user in natural language by substituting X with the result. An example for the tuples generated is shown in Fig 3.

5 Story Centric Knowledge Representation and Reasoning

It is imperative that LISA's story-centric knowledge base excels in three important categories: run-time flexibility, logical capabilities, and case-by-case adjustability.

1. The knowledge base must be a flexible, growable set of facts that is constantly open to change, because the user can add to and delete from the story throughout run-time.
2. LISA's intelligence must be capable of making logical deductions from the facts it retains; otherwise, it wouldn't be more intelligent than a primitive bookkeeping device.
3. The machinery for the logical deductions must be adjustable for different contexts and genres: for example, a fantasy genre may find it normal to have talking animals

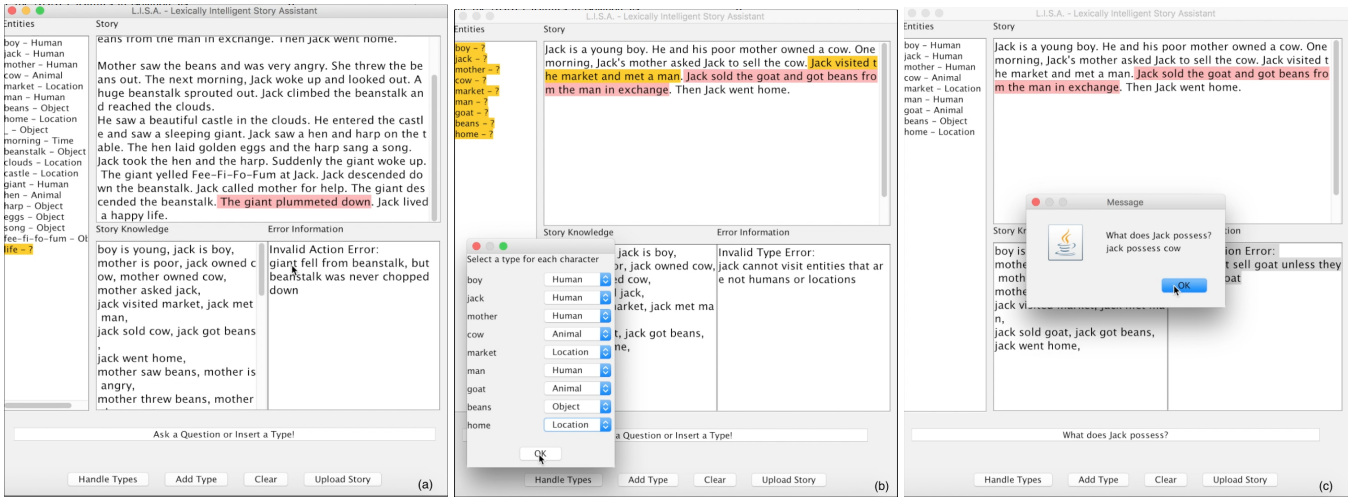

Figure 2: LISA in action with Jack and the Beanstalk as an example story– (a) Error Message Feature: The highlighted text says, "The giant plummeted down." The error says, "Invalid Action Error: giant fell from beanstalk, but beanstalk was never chopped down"; (b) The application with the Entity Type selection to add types to each entity.; (c) Querying example in action: Query: "What does jack possess?", Result: "jack possess cow"

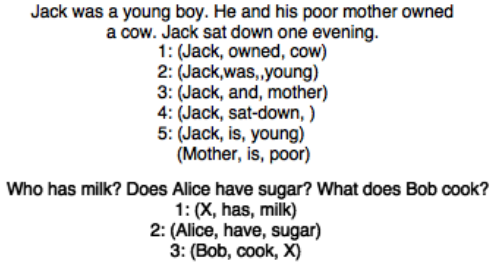

Figure 3: Tuples being generated from sentences and questions.

and moving trees, while a historical setting should not assume these events are possible unless the user specifies.

To accomplish this, LISA's knowledge base implements GNU's library "Prolog for Java", with inferencing determined by a set of Prolog rules (Diaz and Codognet 2000). At this junction of the implementation, technical knowledge is required to modify the rules of the story world before runtime, although allowing the common user, with natural language input, to modify the rules during run-time is a promising prospect for future versions.

5.1 Story Knowledge and Universe Rules

The components to construct the story-centric knowledge base from the information extracted from the story input stream are modularized primarily by separating story action information (actions), entity trait information (traits), entity type information (types), and error information (errors), and modularized secondarily by distinguishing between explicit story knowledge (facts) and universe rules (rules).

Fact sets are maintained during application runtime (reflecting changes in story text), and contain explicit story knowledge from the story text stream. There is a one-to-one correspondence between statements extracted from the story and the union of the fact sets. LISA implements **action facts**, denoted by F_A, and **trait facts**, denoted by F_T. F_A denotes the set of tuples (o_1, a, o_2) such that the story explicitly states that entity o_1 has performed action a on entity o_2, with o_2 possibly empty. Examples include *(jack, visits, market)* or *(giant, woke-up,)*. F_T denotes the set of tuples (o, t) such that the story explicitly describes entity o with the trait t. An example is *(beanstalk, enormous)*.

A **type set**, maintained during application runtime via user input, is a specialized set containing entity type information for each entity. The type set is utilized as a third fact set, and is denoted by Y. Y denotes the set of tuples (o, y) such that the user has determined that entity o is of entity type y. An example is *(jack, human)*.

From here, let $F_S = F_A \cup F_T \cup Y$ denote the set of all facts from the story. Let $F_U \supset F_S$ be the space of all possible facts from all possible entities, actions, traits, and types in the story universe.

Rule functions, which are maintained before application runtime (reflecting changes in the assumptions of the story world), contain functions mapping possible facts from the story universe to the power set of all explicit story facts. Rule functions represent the universe rules that allow LISA to deduce implied information from explicit story information; thus, the function maps implied story knowledge to its set of requisite explicit story knowledge. For example, consider a rule r and an implicit fact f. We know that all facts $g \in F_S$ are true because those are facts explicitly generated from the story input stream. If $r(f) \in F_S$, however, then we also know that f is true, despite it not being explicitly stated. The rule functions utilized by LISA are listed below:

- Let $r_A : F_U \rightarrow \mathcal{P}(F_U)$ denote the rules for implied ac-

tions. That is, if $r_A(f) \in F_S$, then $f = (o_1, a, o_2)$ is an action that has occurred in the story. For example, if "X buys Y" is a fact from the story, we can conclude that "X owns Y", even if the statement is not explicitly written in the story.

- Let $r_E : F_U \rightarrow \mathcal{P}(F_U)$ denote rules for inconsistency checking. That is, if $r_E(e) \in F_S$, then there is some logical error e caused by some combination of facts in F_S. This allows LISA to determine when an inconsistency exists in the story-centric knowledge base and provide useful information for a user attempting to correct the inconsistency (see section 5.2).
- Let $r_T : F_U \rightarrow \mathcal{P}(F_U)$ denote rules for implied traits. That is, if $r_T(f) \in F_S$, then $f = (o, t)$ is a true fact; i.e., entity o must have trait t (in addition to its explicitly stated traits). For example, if "X is killed" is a fact from the story, we can conclude that "X is dead", even if the statement is not explicitly written in the story.

Thus, the set of facts that make up the concrete **story-centric knowledge base** is represented as $F_S \cup \{f : r_A(f) \in F_S\} \cup \{f : r_T(f) \in F_S\}$. An interesting note is that the knowledge base can be constructed for any time interval of the story by generating it anew from the unique F_S (i.e., extracted story information) of that time interval (e.g., a specific paragraph of the story).

5.2 Error Attribution

When the knowledge base is given the task of determining whether a specific sentence causes a logical inconsistency, it checks the given sentence with the previous information in the story. This is depicted in Algorithm 1.

The algorithm is summarized as follows: first, we consider the input to be the statements from the story F_S (processed as described in Section 4.1 and Section 5.1). Using predefined story universe rules K (e.g., if a human X performs a sell action on an entity Y, and X does not have possession of Y, then we can conclude there exists an error), on any input set of statements, we can determine the Boolean value of whether that set contains a logical error. By finding out which subsets of the story statements contain errors, we can identify where in the story an error originates from.

For LISA, these rules were programmed to return a helpful error message (e.g., "Jack sold cow, but Jack does not possess cow."), with the ability to substitute story information values into the error parameters (X = Jack, Y = cow).

Algorithm 1: error attribution algorithm

input : An indexed sequence of n story fact sets $(F_i)_{i \in [n]} : F_i \subset F_S$, as well as the knowledge base of story universe rules and story universe error rules $K = r_A \cup r_T \cup r_E$

output: A set of errors attributed to each story fact set $(E_i)_{i \in [n]}$.

```
F ← ∅;
for i ← 1 to n do
    F' ← F ∪ F_i;
    F* ← F';
    while F* changed do
        F* ← F* ∪ {x : (∃r ∈ K)(r(x) ∈ F*)};
    E_i ← {e : r_E(e) ∈ F*};
    if E_i = ∅ then
        F ← F';
```

6 User Interactions

LISA allows for writing new stories or continuing an existing story. An existing story can be imported using the Upload Story button on the bottom of the interface which prompts the user to select the text file which contains the story. The Text is copied into the Story text box. The Story text box is a text editor that detects when a new sentence is added or when an existing sentence is modified. Using a thread to track the position of the delimiters in the story, LISA ensures that each sentence is processed by the NLP keeping the story knowledge up to date.

Entities. LISA also displays all unique entities in the story and adds any new ones as they appear in the text. The entities are paired with a tag resembling its type. The type for each entity can be changed using the Handle types button on the bottom of the interface. The default type for entities is null which may create errors depending on the rules in the Story World domain. The entities are extracted from the tuples created by the NLP which end up being the afforders and affordees in each event.

Story Knowledge. LISA displays the story knowledge extracted for each sentence under the Story Knowledge label on the interface. Each sentence in the story corresponds to a line in the story knowledge that shows the extracted knowledge. If multiple tuples are extracted in a given sentence, they are separated by a comma in the same line.

Error Information. Any inconsistencies or errors in the knowledge base are highlighted in the Story text editor. LISA uses a color coded system to distinguish the different errors by highlighting the sentence in question. Mousing over to the sentence displays the reasoning in the Error Information text box. Once the error has been cleared, the highlight will be removed and the error information cleared.

Questioning and Types. LISA has a text field that is used to input questions or insert types. If a type is to be added, the Add Type button will add the type into the list of types. The list of types can be seen when handling types. If a question is asked, pressing enter will start processing the question. A new window will be displayed with the results.

7 Evaluation

Based on the design described in this paper, we developed a LISA prototype. The system is designed in Java, and is a swing application. The user interface and different features in LISA are shown in Fig 2. The first screen shows the various windows and information about the user interactions are described in Section 6. For this study, we tested LISA out with sections of a summary of *Jack and the Beanstalk*.

In order to be able to demonstrate LISA's functionali-

ties, we arbitrarily marked down a few possible errors that a writer may make while writing a story, such as making mistakes in insignificant details. In the above example, the writer could possibly make mistakes such as having a character give away something that they do not own, or doing an action which requires an object they do not own. After these possible errors were determined, we created inference rules and added them to the story-centric knowledge representation and reasoning system. Hence, if a writer did make any of the mentioned mistakes, the sentence would be flagged as an error, as demonstrated in Fig 2. The other window shows possible questions that a writer may ask the system along with LISA's responses.

A demonstration video was recorded and can be accessed here: `https://goo.gl/PkmflE`. The demonstration video shows the examples and errors for parts of *Jack and the Beanstalk*. Additionally, a small-scale user study task was designed to test similar features for LISA. The users were given a demonstration of LISA's features and asked to perform a simple story task on a text editor and then using LISA. The results for the user study were inconclusive; there was no time difference between the attempts involving LISA and involving a simple text editor. Due to the small scale of the task, further research and user study is required to complete accurate evaluation on LISA and its benefits over a normal text editor.

8 Conclusion

Limitations and Complications. While LISA acts as a tool introducing a new approach to storytelling, LISA also opens doors for a lot of questions. The core of LISA's efficiency lies in the rules that it checks its facts against. These rules, if broken, cause errors, which are then highlighted and displayed to the user. In the current implementation, the rules have been added manually by the researchers, and were tailored to the task performed in the user study. LISA's error checking capabilities are only as powerful as the information stored in the story-centric knowledge reasoning system. In order for LISA to apply for complex real-world storytelling, the system must be able to automatically generate rules and enforce them on stories.

LISA heavily relies on natural language processing. Therefore, its accuracy also is a factor of the accuracy of the natural language parser. In the current implementation of LISA, the parsing of the tuples only takes certain nodes of the parse tree into consideration. This forms a limitation: the complexity of the sentences in the stories determine the accuracy of the knowledge base in the current implementation. Future work in LISA will involve exploring other information extraction annotators such as Stanford's OpenIE (Angeli, Premkumar, and Manning 2015), and optimizing the accuracy and completeness of information extracted from stories.

Another complication stems from choosing the right platform for the knowledge reasoning and inferencing system. The current implementation of LISA uses Prolog for inferencing. Because Prolog doesn't take order into consideration by default, LISA's current story knowledge base is time agnostic. Possible workarounds to this can be identifying story by assigning identity tags, and using them in the prolog facts. Prolog also uses symbolic reasoning over probabilistic reasoning, which limits the reasoning capabilities. This currently forms a limitation but can also be further explored by adding confidence as a factor. The system currently has no way to record uncertainty about events.

The user-friendliness of the application also forms a bottleneck for the error information provided by the system. This is a trade-off between providing the user with as much error information possible and how to fix it, while making the application user-friendly simultaneously. LISA uses some amount of input from the user through the entities window, but choosing to limit user clarification leads to uncertainty, where LISA cannot accurately infer knowledge without further clarification from the user. Presently, LISA highlights the sentence with the error, and then when the user moves the cursor to the highlighted sentence, it describes the error in the error window. This error display mechanism might not be sufficient for more complicated errors that may span across more than a sentence.

Future Work. The knowledge inferencing system needs more research in order to make it more exhaustive and robust. Future work to improve LISA's error-detection capabilities could focus on connecting to a database online which provides common knowledge regarding the usage and meaning of words used in stories. There are existing datasets, such as Wordnet (Miller 1995) and Verbnet (Schuler 2005), that provide a lexical database. Moreover, natural language parsing systems have also been built with inbuilt integration to Wordnet (Lee et al. 2011). Integration of these lexical databases can assist in detecting more lexical errors and assisting the authoring of lexically correct narratives.

Another possible approach is to use LISA with another knowledge based system. Moreover, various knowledge based systems can be analyzed to look for optimal systems for LISA. The knowledge base could be event-centric or character-centric. Character-centric knowledge bases could be tackled using the Scone knowledge base system (Chen and Fahlman 2008). Adding a character-centric knowledge system would allow for asking questions to different characters in the story and could potentially be used to analyze different character's answers to the same questions based on their knowledge.

Future work in knowledge representation and reasoning for stories also needs to expand on more than encoding just actions from stories. Stories can contain complex structures, and it may be interesting to be able to recognize irony, sarcasm and other metaphorical information and extract it. It could also be possible to check for errors and loopholes in the metaphors used by the writers, or any contradictions in the meta-information inferred from the forms of speech used.

While LISA deals with text-based stories, it's knowledge base could also be extended to other forms of media. Future work in LISA could include collaborations with other libraries such as the CANVAS library (Kapadia et al. 2016) as a plug in to be able to understand different affordances and provide feedback in animation format as well.

References

Angeli, G.; Premkumar, M. J.; and Manning, C. D. 2015. Leveraging linguistic structure for open domain information extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL 2015)*.

Bhattacharya, P., and Neamtiu, I. 2011. A prolog-based framework for search, integration and empirical analysis on software evolution data. In *Proceedings of the 3rd International Workshop on Search-Driven Development: Users, Infrastructure, Tools, and Evaluation*, SUITE '11, 29–32. New York, NY, USA: ACM.

Carlson, L.; Marcu, D.; and Okurowski, M. E. 2003. Building a discourse-tagged corpus in the framework of rhetorical structure theory. In *Current and new directions in discourse and dialogue*. Springer. 85–112.

Chen, W., and Fahlman, S. E. 2008. Modeling mental contexts and their interactions. In *AAAI Fall Symposium: Biologically Inspired Cognitive Architectures*, volume 8, 04.

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

Ciampaglia, G. L.; Shiralkar, P.; Rocha, L. M.; Bollen, J.; Menczer, F.; and Flammini, A. 2015. Correction: Computational fact checking from knowledge networks. *PLOS ONE PLoS ONE* 10(10).

Diaz, D., and Codognet, P. 2000. The gnu prolog system and its implementation.

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

Ginsberg, A. 1988. Knowledge-base reduction: A new approach to checking knowledge bases for inconsistency and redundancy. In *AAAI*, volume 88, 21–26.

Hayes-Roth, F. 1985. Rule-based systems. *Commun. ACM* 28(9):921–932.

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.

Kapadia, M.; Frey, S.; Shoulson, A.; Sumner, R. W.; and Gross, M. 2016. Canvas: Computer-assisted narrative animation synthesis. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '16, 199–209. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association.

Kubler, S.; McDonald, R.; and Nivre, J. 2009. *Dependency parsing*. Morgan and Claypool Publishers. Synthesis Lectures on Human Language Technologies.

Laffey, T. J.; Cox, P. A.; Schmidt, J. L.; Kao, S. M.; and Readk, J. Y. 1988. Real-time knowledge-based systems. *AI magazine* 9(1):27.

Lee, H.; Peirsman, Y.; Chang, A.; Chambers, N.; Surdeanu, M.; and Jurafsky, D. 2011. Stanford's multi-pass sieve coreference resolution system at the conll-2011 shared task. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*, 28–34. Association for Computational Linguistics.

Lee, H.; Chang, A.; Peirsman, Y.; Chambers, N.; Surdeanu, M.; and Jurafsky, D. 2013. Deterministic coreference resolution based on entity-centric, precision-ranked rules. *Computational Linguistics* 39(4):885–916.

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.; Inc, P.; Bethard, S. J.; and Mcclosky, D. 2014. The stanford corenlp natural language processing toolkit. In *In ACL, System Demonstrations*.

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

Mateas, M., and Stern, A. 2004. A behavior language: Joint action and behavioral idioms. In Prendinger, H., and Ishizuka, M., eds., *Life-Like Characters*, Cognitive Technologies. Springer Berlin Heidelberg. 135–161.

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

O'Leary, D. E. 1998. Using ai in knowledge management: Knowledge bases and ontologies. *IEEE Intelligent Systems and Their Applications* 13(3):34–39.

Roemmele, M. 2016. Writing Stories with Help from Recurrent Neural Networks. In *AAAI Conference on Artificial Intelligence; Thirtieth AAAI Conference on Artificial Intelligence*, 4311 – 4312. Phoenix, AZ: AAAI Press.

Schuler, K. K. 2005. *Verbnet: A Broad-coverage, Comprehensive Verb Lexicon*. Ph.D. Dissertation, Philadelphia, PA, USA. AAI3179808.

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.

SMITH, T. C., and WITTEN, I. H. 1991. A planning mechanism for generating story text. *Literary and Linguistic Computing* 6(2):119–126.

Suwa, M.; Scott, A. C.; and Shortliffe, E. H. 1982. An approach to verifying completeness and consistency in a rule-based expert system. *Ai Magazine* 3(4):16.

Toutanova, K.; Klein, D.; Manning, C. D.; and Singer, Y. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, 173–180. Association for Computational Linguistics.

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* PP(99):1–1.