UNIVERSITY OF CALIFORNIA
SANTA CRUZ

SKALD: EXPLORING STORY GENERATION AND INTERACTIVE STORYTELLING BY RECONSTRUCTING MINSTREL

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Brandon Tearse

December 2018

The Dissertation of Brandon Tearse
is approved:

Professor Noah Wardrip-Fruin, Chair

Professor Michael Mateas

William Ferguson

Lori Kletzer
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Skald: Exploring story generation and interactive storytelling by reconstructing Minstrel

by

Brandon Tearse

Within the realm of computational story generation sits Minstrel, a decades old system which was once used to explore the idea that, under the correct conditions, novel stories can be generated by taking an existing story and replacing some of its elements with similar ones found in a different story. This concept would eventually fall within the bounds of a strategy known as Case-Based Reasoning (CBR), in which problems are solved by recalling solutions to past problems (the cases), and mutating the recalled cases in order to create an appropriate solution to the current problem. This dissertation uses a rational reconstruction of Minstrel called Minstrel Remixed, a handful of upgraded variants of Minstrel Remixed, and a pair of similar but unrelated storytelling systems, to explore various characteristics of Minstrel-style storytelling systems.

In the first part of this dissertation I define the class of storytelling systems that are similar to Minstrel. This definition allows me to compare the features of these systems and discuss the various strengths and weaknesses of the variants. Furthermore, I briefly describe the rational reconstruction of Minstrel and then provide a detailed overview of the inner workings of the resulting system, Minstrel Remixed.

Once Minstrel Remixed was complete, I chose to upgrade it in order to explore the set of stories that it could produced and ways to alter or reconfigure the system with the goal of intentionally influencing the set of possible outputs. This

investigation resulted in two new storytelling systems called Conspiracy Forever and Problem Planets. The second portion of this dissertation discusses these systems as well as a number of discoveries about the strengths and weaknesses of Minstrel Style Storytelling Systems in general. More specifically, I discuss that, 1) a human reader's capacity for creating patterns out of an assortment of statements is incredibly useful and output should be crafted to use this potential, 2) Minstrel-Style Storytelling tends to be amnesiac and do a poor job of creating long stories that remain cohesive, and 3) the domain that a storytelling system is working from is incredibly important and must be well engineered. I continue by discussing the methods that I discovered for cleaning up and maintaining a domain and conclude with a section covering interviews with other storytelling system creators about the strengths and weaknesses of their systems in light of my findings about Minstrel Remixed.

In the final portion of this document I create a framework of six interrelated attributes of stories (length, coherence, creativity, complexity, contextuality, and consolidation,) and use this along with the learning discussed in the first two portions of the dissertation to discuss the strengths and weaknesses of this class of CBR systems when applied to both static story generation and interactive storytelling. I discuss the finding that these systems seem to have some amount of power and although they can be tweaked to produce for example, longer or more consolidated stories, these improvements always come along with a reduction in complexity, coherence, or one of the other attributes. Further discussion of the output power of this class of storytelling systems revolves around the primary limiting factor to their potential, namely the fact that they have no understanding of the symbols and patterns that they are manipulating. Finally, I introduce a number of strategies that I found to be fruitful for increasing the 'output power'

of the system and getting around the lack of commonsense reasoning, chiefly improving the domain and adding new subsystems.

To Kari Tearse for her love, support, and patience throughout this journey.

Acknowledgments

First and foremost I would like to thank Noah Wardrip-Fruin for patiently guiding my academic progress. His mentorship and encouragement have been key factors in this journey and without his persistent support this document would never have been completed. I would also like to thank Michael Mateas and Bill Ferguson, both of whom have played key roles in helping me chart a course for my dissertation research.

Additionally, I would like to acknowledge the fantastic environment created by the founders and members of the Expressive Intelligence Studio at UC Santa Cruz. Without the constant barrage of creative stimulation, my research would have been far less exciting and interesting.

Finally, I would like to thank Peter Mawhorter, my research partner for the entire construction of Minstrel Remixed, Skald, and Problem Planets. We had a good time doing a lot of great work and I couldn't have done what I did without Peter's contributions.

Chapter 1

Introduction

In my first class on Artificial Intelligence, one of the systems that was covered was Scott Turner's influential 1993 Minstrel system [Turner, 1993]. We learned that this graph-based story generator was designed to mimic a human author's creative process. As one of only a few early systems that attempt to model human creativity in the domain of storytelling, its unique approach is worthy of study and it is also often cited (e.g., [Gervas, 2009, Sharples, 1997, Wardrip-Fruin, 2009]). Minstrel is often included in lists of seminal story-generation systems alongside other early storytelling systems such as Meehan's Tale-Spin [Meehan, 1981] and Lebowitz's Universe [Lebowitz, 1984]. Based on a model of human creativity that centers on *imaginative recall*, Minstrel generates stories by modifying and splicing together fragments from a library of stories in order to fit its own needs. Despite its influence and important ideas, Minstrel, like many other early AI projects, is not accessible today and the only evidence that it ever existed and what it could do is found in Turner's Ph.D. dissertation and follow-up book [Turner, 1993, Turner, 1994].

While learning about Minstrel, I felt that it was a compelling storytelling system which produced output that was far more interesting than its contempo-

raries. My interest soon bloomed into a desire to rebuild Minstrel and after a short period of time exploring this possibility alone, I was joined on the project by Peter Mawhorter and eventually by Aaron Reed as well. Our main motivation to reconstruct Minstrel was based on the quality of the stories that it generated. We wanted to figure out how they were produced and how general this process was. For example, we wanted to explore how much of Minstrel's code would have to change to produce stories in a different genre. Reproduced here is an example story that Turner presented in his dissertation, which demonstrates the full capabilities of his system:

> **The Mistaken Knight**
>
> It was the spring of 1089, and a knight named Lancelot returned to Camelot from elsewhere. Lancelot was hot tempered. Once, Lancelot had lost a joust. Because he was hot tempered, Lancelot wanted to destroy his sword. Lancelot struck his sword. His sword was destroyed.
>
> One day, a lady of the court named Andrea wanted to have some berries. Andrea wanted to be near the woods. Andrea moved to the woods. Andrea was at the woods. Andrea had some berries because Andrea picked some berries. Lancelot's horse moved Lancelot to the woods. This unexpectedly caused him to be near Andrea. Because Lancelot was near Andrea, Lancelot loved Andrea. Some time later, Lancelot's horse moved Lancelot to the woods unintentionally, again causing him to be near Andrea. Lancelot knew that Andrea kissed with a knight named Frederick because Lancelot saw that Andrea kissed with Frederick. Lancelot believed that Andrea loved Frederick. Lancelot loved Andrea. Because Lancelot loved Andrea, Lancelot wanted to be the love of Andrea. But he could not because Andrea loved Frederick. Lancelot hated Frederick. Andrea loved Frederick. Because Lancelot was hot tempered, Lancelot wanted to kill Frederick. Lancelot wanted to be near Frederick. Lancelot moved to Frederick. Lancelot was near Frederick. Lancelot fought with Frederick. Frederick was dead.
>
> Andrea wanted to be near Frederick. Andrea moved to Frederick. Andrea was near Frederick. Andrea told Lancelot that Andrea was siblings with Frederick. Lancelot believed that Andrea was siblings

> with Frederick. Lancelot wanted to take back that he wanted to kill Frederick. But he could not because Frederick was dead. Lancelot hated himself. Lancelot became a hermit. Frederick was buried in the woods. Andrea became a nun.
>
> MORAL: Done in haste is done forever. [Turner, 1993]

This story has a coherent plot, reasonably dramatic action, and incorporates foreshadowing. The English is a bit stilted, but isn't full of errors, and the story has an explicit moral that it bears out. Even in the context of modern computer-generated stories, it stands out as both interesting and coherent.

1.1 Turner's Minstrel

A quick read of Turner's dissertation and book provides a number of tempting facts. Turner shares the above story a number of times in various states of construction along with seven other stories which are far shorter and less complex. He also shares a diagram showing 'a portion' of his object ontology which is a branching graph containing 18 nodes and 9 leaves. Along with an in-depth description of a lot of other machinery which I will describe in more detail in chapter 4, these documents describe a complex system, how it runs, and a compelling example output.

Viewed from the perspective of a pair of documents describing a story generator, Turner's description of Minstrel seems to lack a lot of detail: Minstrel builds stories based on a library of other stories but Turner never shares the content of his library or the entire object ontology. For quite some time I believed that Turner was being coy and only providing demonstrations which made Minstrel look good. Upon more in-depth reading, while searching for clues about Minstrel's missing story library and set of nouns, I discovered that my original assumption was wrong.

Minstrel is not actually a story generator, it is a demonstration of computational creativity. This demonstration does happen to generate a story but "The Mistaken Knight" is the only story that Minstrel was designed to build. This is due to the fact that Minstrel only includes the stories, templates, and logic required to piece together this singularly impressive construct. Turner did perform a handful of small experiments using Minstrel which resulted in a few shorter stories but none with the complexity of "The Mistaken Knight." Most importantly for my purposes, Turner's experiments demonstrated that Minstrel could be used to generate more stories as long as it was provided with more information to draw from.

This begs the questions, how many unique stories could Minstrel generate and how hard would it be to increase that number? Given Turner's description of Minstrel's internal systems, these questions are difficult to answer. Also of interest was a second query: could Minstrel be used to create stories incrementally allowing a user (player?) to influence how they turned out? To answer these questions, we decided to undertake a rational reconstruction of Minstrel.

1.2 Rational Reconstruction

Rational reconstruction is the process of building a new version of a given system based on a description of that system. Graeme Ritchie describes the process of rebuilding Proteus as "an attempt to present a slightly cleaner, more general method, based on Davey's ideas and performing the same specific task as Proteus" [Ritchie, 1984]. In contrast, McDonald describes his reconstruction of Genaro as, "how Genaro would be rewritten today given the experience and theoretical developments of the last fifteen years" [McDonald, 1999]. Peinado and Gervás have a similar viewpoint stating that their reconstruction of Minstrel's knowledge rep-

resentation should "open a discussion for fair comparisons between [Minstrel's] achievements and the results of our up-to-date –technologically speaking– research projects" [Peinado and Gervás, 2006]. Rational Reconstruction is also discussed as a useful research tool in Partridge's books on the fundamentals of Artificial Intelligence [Partridge and Wilks, 1990, Partridge, 1991] and in efforts by both Musen [Musen et al., 1995] and Langley [Langley and Stromsten, 2000]. In all of these cases, the goal is to get access to a copy of an otherwise unobtainable system in order to explore some of its potential either by comparing it to other newer systems or by extending some portion of the reconstruction to see what it can do.

Reconstructing Minstrel was a challenge but it provided a new perspective on this acclaimed project and an opportunity to extend Turner's experiments and more fully understand how the system works as a story generator. My team found a number of places where the description of a system was unclear or contradictory, possibly indicating places where the system had been altered by Turner. Also, because we weren't working with the source code, our reconstruction revealed choices that Turner was likely unaware that he had made. While these findings were interesting, the real value of the reconstruction was having a relatively modern copy of Minstrel to experiment with.

1.3 Story Generation

Computer programs that generate compelling stories are difficult to create, as evidenced by some forty years of research on the subject. The earliest dedicated story generation system was Klein et al.'s 1971 Automatic Novel Writer [Klein et al., 1971, Klein et al., 1973], an extension of his earlier work on natural language processing and linguistics [Klein, 1965]. The most influential early system, however, was Meehan's Tale-Spin [Meehan, 1976], which came to be re-

garded as the foundation of story-generating systems. Tale-Spin used character-level planning to generate stories: each character in Tale-Spin would plan a series of actions to get out of some imposed predicament, and these actions became the story output. Tale-Spin inspired many other systems to use planning as their core assembly algorithm, but most later systems avoided purely character-driven stories.

After Tale-Spin, systems like Dehn's Author [Dehn, 1981] and Lebowitz's Universe [Lebowitz, 1984] introduced the concept of author-level reasoning. Dehn's work in particular used a method of memory organization (intended to mimic human memory) that Turner also used in Minstrel. Where Dehn focused on author-driven creation of story scenes, Lebowitz's system revolved around the properties of its characters, although the event creation mechanism functioned using author-level constraints (e.g., "now these two characters must fall in love") rather than through character-level planning as Tale-Spin utilized.

Building on these early systems, Turner's 1993 Minstrel [Turner, 1993] worked at the author level, but it used case-based reasoning (CBR) to fill in story fragments. CBR works by consulting a library of problem and solution pairs whenever a new problem is encountered. Where story generation is concerned, problems often look like fill-in-the-blank questions (e.g., "A _____ walks into a bar..." with possible solutions from the library yielding options: man, child, cow, rope, etc.) but the technique is not constrained to story generation and thus problems and solutions could take on any structured and searchable format. Our reconstruction operates almost identically to the original; full details of the architecture are described in Chapter 4.

Since Minstrel was written, several other story generation approaches and associated concepts have emerged. Some systems have continued to use planning-based

approaches (e.g., [Riedl, 2004] and [Riedl and Young, 2006]), while others have further developed concepts such as agent-based algorithms [Theune et al., 2003], rule-systems [Sgouros, 1999], or even genetic algorithms [Bui et al., 2010]. Other research which is directly applicable to this dissertation are the many efforts to better measure what a given storytelling system can produce. Smith discusses a technique to measure the potential space for content generation which is quite similar to the Possibility Story Space and Story Likelihood Map that I discuss in chapter 9 [Smith, 2012]. Additionally, numerous efforts aim to explore how different techniques affect *authorial levarage*. In the same way that a tool's leverage amplifies the work of anyone using that tool, the *authorial leverage* provided by a story generation system amplifies the work of an author, enabling them to make more or better stories. Although there are many examples of this, some interesting and relevant articles examine the use of language [Walker et al., 2013], drama management [Chen et al., 2009], and experience management [Thue et al., 2013] while attempting to maximize *authorial leverage*.

One line of work similar to Minstrel's case-based approach is Zhu et al.'s 2010 Riu system [Zhu and Ontanón, 2010], which uses a computational analogy algorithm to fill in parts of a story. A few other systems have also used CBR, including Pérez y Pérez's 1999 Mexica [Pérez y Pérez, 1999], Fairclough & Cunningham's 2003 system [Fairclough and Cunningham, 2003] and Gervas et al.'s 2005 Proto-Propp [Gervas et al., 2005]. These systems all derive world knowledge from a library of stories rather than encoding it explicitly, while also being able to use near-matches to assist in recalling solutions to novel problems.

Of these CBR-based systems, Riu's computational analogy algorithm is most similar to Minstrel's imaginative recall approach. Like Minstrel, Riu operates at the level of an author, modifying stories by filling in scenes sequentially. Although

computational analogy and case-based approaches have a lot in common (notably both rely on a story library), computational analogy uses a very specific matching and adaptation algorithm that differs from Minstrel's TRAMs. The other three notable CBR-based story generators function differently: Mexica uses a Tale-Spin-like approach, telling a story based on character-level simulation, while both Fairclough & Cunningham's system and ProtoPropp are based on Vladimir Propp's work on story grammars [Propp, 1968]. Propp was a Russian narratologist who proposed a grammar-based model of Russian folk tales. His model is attractive because it's computationally tractable, but it is often applied outside of its indended domain (folk tales) and thus may be a less effective tool for some cases.

1.4 Computational Creativity

Turner's original thesis claimed that Minstrel was a computer model of human creativity during story construction [Turner, 1993]. Creativity, however, especially computational creativity, is an extremely difficult topic to define. Most academics are willing to agree that creativity can be computationally modeled, that the output must be suitably unique and have some amount of quality or value in order for it to be creative, and that what exactly creativity is and how it is measured is still a very open question. In our previous work we have tried to measure how creative Minstrel is using notions of variability and quantity of output [Tearse et al., 2011b]. These efforts were fairly coarse, measuring other values as a stand-in for creativity. Ultimately, I could not find a good way to measure creativity; but in chapter 7 I do discuss an evaluation of creativity that is mostly based on the novelty of stories.

Margaret Boden's idea of p-creativity is one starting point for a more formal

definition of creativity. The output of a system is p-creative if it is unique with regards to the input to and previous output of that system [Boden, 2004]. This idea coupled with the constraint that the output must be consistent with the context in order to qualify as creative (i.e., producing unique gibberish is not creative) is the basis of a number of creativity frameworks (e.g., [Boden, 2004, Ritchie, 2001]). Although Turner's original definition [Turner, 1994] agrees with much of the thinking in the field and we have used variability and quality of output to measure the creativity of the system, other more complex metrics exist (e.g., [Wiggins, 2006, Colton et al., 2011]).

1.5 Interactive Storytelling

The capabilities of computers have dramatically improved in the decades since Minstrel was designed. While academics have used these improvements to create more powerful general generators that can perform story generation more quickly and using techniques other than CBR, the games industry has been incrementally improving solutions for a related goal: bringing interactive storytelling elements from tabletop games to electronic ones. Interactive storytelling is what happens when some form of communication between two separate entities causes a story to be created. One of the most clear examples of this is a group of humans using a codified simulation such as Dungeons and Dragons [Gygax and Arneson, 1974]. Using a game to facilitate interactive storytelling between people is common but humans need not be the only participants in these activities. The Choose Your Own Adventure (CYOA) franchise of books exemplified by Edward Packard's, "The Mystery of Chimney Rock," is a wonderful example of a person collaborating with a book to create a story [Packard, 1980].

Most examples of a person participating in interactive storytelling nowadays

take place on a computer and use a video game as one of the parties. There are a wide variety of different video games that include storytelling as an important aspect of the game. An exhaustive list is unnecessary for the purposes of this dissertation so I will discuss only the Adventure and Role-Playing Game (RPG) genres.

Adventure games and RPGs both generally provide a static linear storyline which players progress through. Whether the player is allowed to explore the storyline out of order or not, they generally are only allowed to move from one section to another by completing a challenge of some sort. Games such as The Secret of Monkey Island and Zork [LucasArts, 1990, Infocom, 1977] are wonderful experiences but include a smaller degree of interactive storytelling than others since they make no attempt to give players control over the direction of the narrative. In contrast, games such as Dragon Age, Fallout 4, and Grand Theft Auto V [BioWare, 2009, Studios, 2015, RockstarGames, 2013] do tend to provide at least a minimal amount of agency to players when it comes to choosing the path for the narrative. A favorite technique for doing this without pre-generating an exponentially branching set of narrative paths is to use what Greg Costikyan terms a 'Beads on a String' model [Costikyan, 2007]. The defining characteristic of a game using such a model is that narrative choice is only locally relevant and all branching narrative options in a given section of the game eventually lead to the same conclusion prior to the next section of the game.

A few excellent examples of interactive storytelling do exist which focus much more on working with the player to create a narrative rather than forcing players through a single premade narrative path. A wonderful example that comes from academic sources is Facade [Mateas and Stern, 2003], a game about a friend visiting a couple for dinner, where the player can seemingly ask any question or make

any remark and the non-player characters do a remarkable job of making their responses fit nicely within the confines provided by the story. The only electronic example that I can find in the games industry is Heavy Rain [QuanticDream, 2010] which sits somewhere between an Adventure game and an RPG. Heavy Rain features a fairly long branching static storyline which does a good job of supporting a diverse range of player actions and outcomes throughout any playthrough. In essence, Heavy Rain is an electronic version of the old CYOA novels. Both Heavy Rain and CYOA novels rely on authors pre-generating all possible narratives. This is a daunting task but the end result can be a compelling experience. Similar to the games which navigate a branching pre-made story are the games which have some rule system for piecing a story together from a collection of fragments. Great examples of this are the board games, Tales of the Arabian Nights and Betrayal at House On The Hill [Goldberg, 1985, Glassco, 2004] which both include a huge book full of story fragments and a rule system both for traversing from one to another.

1.6 Research Contributions

The contributions described in the following chapters are all derived from our reconstruction of Minstrel. I and the rest of the Minstrel Remixed team produced a story generation system that was as true to Turner's original as we could get: this is discussed in detail in chapter 4. We also built some addons to Minstrel Remixed which let us study how this style of storytelling can be applied to interactive storytelling (More on this in chapters 6 and 8).

The original Minstrel was able to generate one excellent story (The Mistaken Knight) through careful crafting of the domain so that it interacted well with Minstrel's combinatorial machinery. Our work indicates that Minstrel is far less

capable as a general story generator than the demonstration story implies; without a carefully tailored story library, the quality of its output decreases drastically. The original Minstrel was intended more as a model of computational creativity than as a general storytelling system, so issues of authorability and story quality across the entire generative space were not a primary focus of that research. In this dissertation, however, we take story generation and the issues of authorial control as the primary lens through which we look at Minstrel.

To make the imaginative recall algorithm more robust we've developed alternate implementations of Turner's fragment transformation, selection, and boredom systems which work better without a tailored library, and which spread the material available in the story library over a larger number of output stories. Our work also illuminated places where the original system's documentation was unclear or contradictory as well as many idiosyncrasies of the underlying representation schema that limit what kinds of stories can be represented. When possible, we have tried to implement something as close as possible to the original system and created modularized additions when adding new features so the core can still be experimented with. All of these improvements are discussed in chapters 4 and 5 and analyzed in chapter 7.

In addition to the rational reconstruction of Minstrel, we also explored ways that our storytelling system's story library and configuration affect the various qualities of its output. We noticed very early on that the coherence of our output stories is inversely related to creativity, length, and complexity. The system seems to have an "output power" which could be reallocated between attributes but not increased. Detailed in chapters 4 and 5 are the new additions that we added to the reconstructed system to fix perceived flaws or enable new behaviors. Finally, I discuss how we attempted to update our system using concepts

borrowed from interactive storytelling experiences. This effort ultimately failed since our upgraded version of Minstrel has a high likelihood of breaking either the user's suspension of disbelief or immersion. While this failure is frustrating, one of the major contributions of this document is an analysis of the reasons for these challenges which is discussed in chapters 7 and 8.

Chapter 2

Foundations

To facilitate the later rational reconstruction and analysis of Minstrel, this chapter defines a new architectural class of story generation systems: Minstrel Style Storytelling (MS3) systems. Such systems are characterized by their use of graphs containing symbols to represent stories, toggling between constructive and reflective modes, and use of Case-Based Reasoning. This chapter first defines terms and standard characteristics of MS3s, then applies the MS3 model to three story generation systems. The MS3 model will continue to be used throughout the dissertation.

2.1 Defining Terms And Concepts

2.1.1 Story

According to the Oxford English Dictionary [Simpson et al., 1989], "story" has a number of definitions, two of which are:

> "An account of imaginary or real people and events told for entertainment"
> "An account of past events in someone's life or in the evolution of something"

For the purposes of this dissertation, both of these definitions are accurate. Additionally, stories are also understood to be individual outputs of MS3s. While most MS3s produce textual output, there's no reason one couldn't produce stories in other forms such as poetry, comic strips, or playable experiences.

2.1.2 Story Element

A story element is a unit of narrative content. A story is composed of many story elements put together. A story element is not a literal word or group of words taken from a story such as, "It was a dark and stormy night". Rather, these words are a representation of the concepts that the setting for the current story is in the past, at night, and is dark and stormy. These concepts (dark, stormy, etc.) could be considered to be individual elements or a single 'setting' story element. The granularity of story elements vary by the system involved and potentially by parameter settings of the system itself. In either case, Story Elements are considered to be the building blocks that all MS3s work with.

2.1.3 Simple and Complex Story Elements

While describing how to encode stories in a manner that computers can understand and manipulate, Schank and Abelson describe in detail the concept of Causal Chaining [Schank and Abelson, 1977]. This term covers the interconnections of cause and effect. When one story element is implicitly or explicitly described as a precondition for another, reasoning about the two actions together is an option and doing so generally produces better outcomes when it comes to story generation. Readers who encounter causal chains will generally infer a number of facts about a given story which the MS3 did not explicitly include. These inferences can be superfluous to the story or even happy accidents which further

engage readers; but one of the primary places where a reader's sense of verisimilitude is broken is when an MS3 contradicts facts that a reader feels are implied in a causal chain. Since MS3s routinely handle varying levels of causality between story elements, it is useful to draw a distinction between the causal chains which are simple to work with and those which are more complex. Consider the following 2 element story:

> The fork sat on the table.
> The mouse was scared to leave its hole.

This story contains two simple story elements which are completely disconnected from one another. With a small change however, both elements become interconnected:

> The cat sat on the table.
> The mouse was scared to leave its hole.

Now that these elements have an implied causal connection (element 2 is caused by element 1), they work together to provide more meaning. Although causal chaining could describe the complexity here, the connection is only in the implied threat of violence upon the mouse. By joining the two elements with the word "so" the connection becomes explicit but without this addition, no cause and effect or precondition relationship is guaranteed to exist between these two elements.

I find it useful in this case to augment causal chaining by looking at whether the one element gains meaning from the context provided by the other and vice versa. This slight change turns the answer to the question, "are these elements connected?" into an unequivocal "yes." By judging story elements in this manner it is possible to determine which elements are simple (unconnected) and which elements are complex (contextually connected to other elements). Complexity isn't a binary relation, the more contextual information a given element borrows

from its surroundings, the harder it is to reason about or alter — and thus the more challenging it is for story generators to work with. The following story features 3 story elements each with a different level of complexity:

> Thief steals the statue from Victim.
> Policeman arrests Thief.
> Policeman returns the statue to Victim.

In this story the first element is simple, it can stand alone and doesn't benefit from the context of the second or third elements. The second element is complex, it could stand alone but gains extra meaning if the first element is present to provide context. The third element is even more complex than the second since it benefits from the contextual information of both the first and second story elements.

2.1.4 Storytelling Domain

The Domain for a set of stories at its loosest definition is the genre or other overarching feature that the stories share. The more precise definition that is applied throughout this dissertation is that the domain is the collection of components that add world knowledge to the system. This includes many internal components (e.g., lists of analogous nouns, code to detect specific patterns and transform them, etc.) and external components (e.g., story library, templates, etc.). Many components of the system include some domain knowledge so separating pieces which are part of the domain from those which are not is often difficult. Existing MS3s have a wide range of domains covering almost exclusively fictional narratives[1] such as fairy tales and science-fiction.

[1] One notable deviation from this generalization is that Turner describes a single experiment in which Minstrel is used to produces "stories" that are solutions to mechanical problems (portrayed as a sequence of events).

2.1.5 Possible Story Space and The Story Likelihood Map

One of the difficult challenges of creating a story generation system is figuring out how to answer the question, "how does this story compare to that one in terms of X" where X can be nearly any property of a story and is often something ambiguous and subjective such as creativity, believability, or quality. Since many of those measurements seem impossible to accomplish computationally (although many have tried [Colton et al., 2011] [Gervas, 2009] [Ritchie, 2001] [Wiggins, 2006] [Zhu, 2012]), we found that the only useful ways to concretely analyze our system was by looking at the output of the system as a whole. This approach is very similar to Gillian Smith's work on characterizing the potential space of procedural content generators [Smith, 2012]. For our work, we invented two theoretical constructs, the Possible Story Space (PSS) and Story Likelihood Map (SLM) to answer the question, "If we make this change, can we remove some undesirable output stories without removing any other stories and without changing the likelihoods that various stories will be produced?" In addition to providing measures for incremental tuning of the generator, the PSS and SLM also provide insight into the quality of the story domain. This is discussed more in chapter 9 but what follows is a short description of the PSS and SLM.

The Possible Story Space is the set of all unique stories that a particular setup on a given story generation system can produce. Although some systems are made to only produce a specific set of output stories, MS3s (and most other story generation systems) use content provided in the domain (and thus as part of the setup for the system) to generate stories. Given this, by changing the domain, active system components, and generation parameters, one can dramatically alter the set of potential output stories. The PSS allows authors, engineers, and academics to observe general characteristics about the output of an MS3. While the informa-

tion one derives from a PSS does not point out any low-level flaws that an author can immediately address, it is useful for evaluating the effects of alterations and to determine larger problems such as when the system is not applying appropriate levels of constraint to its output.

To further characterize the generative space of the system, another invention of ours, the Story Likelihood Map (SLM) can be calculated. The SLM is a set of values describing the likelihood that any given story will be produced in a single run. The SLM can be seen as a heat map on top of the PSS and, in addition to being a great way to describe the general qualities of a story generation system, it can also forecast the output probabilities of the system and thus provide a slightly different set of insights.

2.1.6 Story Coherence

The term "story coherence" is a catch-all term I've chosen to assist in discussing problems that appear in stories produced by MS3s. All stories are coherent by default and as a story becomes less coherent, it is judged to more be "crazy" or "nonsensical" by human readers[2]. Coherence comes in two types, internal and external, each covering a slightly different concept. That being said, these concepts are fuzzy and although coherence flaws are generally easy to spot, many are not easily categorized as exclusively internally or exclusively externally incoherent (hence the use of coherence as a catch-all).

Internal coherence measures how much a story maintains consistency within itself. Flaws in internal coherence are often pointed out as sections that "make no sense" or are otherwise hard to understand from a logical perspective. This

[2]In the best case, this leads to what Meehan calls "mis-spun" tales [Meehan, 1981] which nicely covers all output that is close to but not-quite correct (and often quite comical as a result). At worst they lead to something completely incomprehensible.

does not cover ambiguity or poorly defined sections which also make little sense but are not necessarily internally incoherent in the way being covered here. There are two primary internal coherence flaws: violating previously defined reality and unexplained appearances. The first flaw, violating previously defined reality, happens when an item, person, or piece of information is in an unexpected state with no explanation as to how or why it achieved that state provided either before or after the fact. To provide examples, it would detract from a story's internal coherence should it depict a person entering a room and then entering it again or if a person is declared to be destitute and then purchases a bag of gems without some sort of explanation. The second flaw is similar: when an item, person, or piece of information suddenly appears (or an inconvenient thing disappears) without explanation. Often this second type of flaw is too minor to matter; readers will assume that a king has money or a knight has a generic weapon without having the source of these items being explained. That said, a knight who is losing a fight with a dragon and then pulls out a wand to disintegrate it would likely raise an eyebrow or two without some fairly good explaining as to why he didn't use it earlier and where he got it. Whatever the case may be, having a plot suddenly change course with the help of an unexplained and unexpected item, entity, or ability, is seen as "cheating" or "nonsensical" and degrades the quality of the story's coherence in the eyes of readers.

External coherence measures how much a story stays consistent with a reader's concept of the laws that govern reality as it pertains to the domain. Some examples of this are dead characters taking actions, animals talking, and wizards using cellphones. Generally speaking, any of these elements appearing in a story will cause the users to question whether or not the story generator is working properly. That being said, in the right domain, each of these three examples is actually

fine: people bitten by zombies have a tendency of continuing to act, creatures in many fairy-tales have the capacity to talk, and the urban fantasy setting is known to be populated by tech-wielding wizards who are expected to have one or more mobile devices.

As a final note, coherence is orthogonal to "interestingness" and "creativity". This is best demonstrated by a story about a dragon and a knight that Minstrel produced during a live demonstration at a conference (explanatory comment added in second-to-last line):

> Smaug hated Lancelot.
> Smaug decided to kill Launcelot.
> Smaug poisoned King Joshua.
> King Joshua died and became Poisoned King Joshua.
> Smaug gave Poisoned King Joshua to Lancelot.
> Lancelot was hungry and ate Poisoned King Joshua. **(kings are dragon food)**
> Lancelot died, Smaug was happy.

Although this story contained an external coherence flaw (a knight would never eat a king), this flaw proved endearing to readers who felt the story was quite creative and interesting mostly due to the fact that the system had almost but not quite gotten it right.

2.2 Components and Mechanisms Common to MS3s

Although no storytelling system contains exactly the same components or even types of components as any other storytelling system, it's certainly possible to create categories of storytelling systems based on their components or processes. Many different categories exist and in this dissertation I'll be discussing a category

of my own devising, the Minstrel Style Storytelling System (MS3). In this section I will outline the defining features of an MS3 but before I do that, a few things should be said about this category. First and foremost, this category is centered on Minstrel but is not defined by it. Some of the core MS3s have features that were not present in Minstrel and, I argue, help them accomplish the same kinds of reasoning that Minstrel was aiming for in a better manner than Minstrel itself did. Secondly, there are no strict rules for inclusion in this category. What makes the processes or components of a system fit into "Minstrel Style Storytelling" is sufficiently ambiguous to make a precise judgement impossible. Instead, the following list lays out a set of core concepts and components against which systems can be compared to determine whether they fit more or less into the MS3 category.

1. Symbols and Patterns
    - Stories are a collection of symbols and patterns which are manipulated by the system.
    - Story templates are used to provide structure for Construction.
    - Employs Natural Language Generation.
2. Construction and Reflection
    - Constructive Mode adds elements to the story.
    - Reflective Mode alters or removes existing elements.
    - System processes toggle between constructive and reflective modes.
3. Case-Based Reasoning
    - Constructs stories by using CBR methods to find appropriate story fragments.
    - Methods to intelligently select which CBR search results to use.
    - Curated Case Library.

2.2.1 Symbols and Patterns

As was discussed in section 1.3, there are a number of symbolic and analogical reasoning systems that perform computation on patterns of symbols for many reasons. MS3s do this as well and the most fundamental concept behind Minstrel-style reasoning is that a story is comprised of a handful of symbols laid out in a specific pattern. While humans may understand the symbols and the pattern that they are laid out in, from the perspective of the system, none of them hold any special meaning. As an analogy, a human presented with a handful of images containing variously colored boxes can tell the images apart and may even be able to identify repeated elements and configurations but beyond such pattern recognition, no further meaning would be forthcoming.

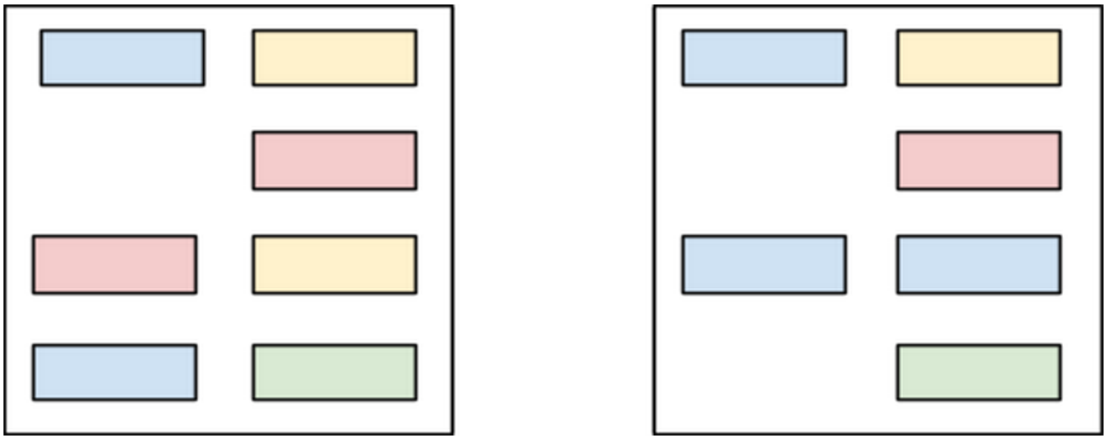

Figure 2.1: Although you can tell these two "stories" apart, identify similarities, and maybe make one of your own, you can't identify their topics, point out coherence flaws, or judge one to be superior.

Since the best way to represent this 'pattern of symbols' is a graph, MS3s generally represent their stories as graphs in some way. These graphs often vary from system to system since most MS3s borrow a story representation from elsewhere. Minstrel's Conceptual Dependency uses a graph of frames which are themselves a set of attributes while Riu's authors chose to base their story representation on

Force-Dynamics which is best represented by a large number of interconnected strings. The story representation certainly has an effect on the other components but isn't itself a factor that helps discriminate an MS3 from others.

One unfortunate repercussion of having a system that doesn't understand any of the symbols that it is manipulating is that these systems generate bland formless stories unless they are given an initial framework to work with. Thus, MS3s often contain one or more templates that are designed to provide the basic shape of a desirable story without providing too many constraints on the system's output. It should be noted that while in many cases these templates are obvious, some systems have methods to repurpose stories from the Case Library and use them as story templates.

Another signature module that any storyteller using a symbolic story representation must contain is a way to render text into language. Since MS3s manipulate symbols and patterns rather than actual phrases they are saved from having to deal with the complexities of language while they create the plot of a story. When they are done, however, they must have some method to turn the created story into something that an average person can read. Most systems choose to use a simple templated text rendering system in which story elements are paired with templated sentences to produce textual output. That being said, smarter rendering systems do exist which can make up for holes or flaws in the story graph, mitigate the repetition that is inevitably produced by templated text rendering systems, or even introduce new details to enhance the story.

2.2.2 Constructive and Reflective Modes

Although not all MS3s separate their workflow into separate distinct modes of operation, all MS3s spend time on at least two different creative endeavors:

Construction and Reflection. Construction is time spent adding new content to the story while Reflection is time spent removing undesirable content, modifying existing content, or detecting specific patterns and altering runtime parameters in some way as a result.

The concepts of Constructive and Reflective[3] modes are taken straight from Mexica which was built around these two concepts. While running, Mexica literally toggles between these two modes until the story is complete. The first mode adds a new element on to the story while the second checks over the newly added content to see if it needs filtration, can be mutated, or requires augmentation. In Minstrel and Riu, the Constructive and Reflective modes are merely useful abstractions (rather than concrete distinct processes). As an example, Minstrel's ALP subsystem (which is discussed in detail in Chapter 4) triggers Constructive and Reflective modules as needed. This more integrated example in which a single process interleaves the two modes into one workflow tends to be the norm in MS3s.

2.2.3 Contains Modified Case-Based Reasoning

As a rule, MS3s use Case-Based Reasoning (CBR) as the primary tool to add content during Construction. Specifically, MS3s query with and lookup fragments of cases (where cases are stories). There are many strategies to accomplish this, largely because the differing story representations adopted by MS3s are each supported best by slightly different CBR strategies. Implementation details aside, all MS3s take some piece of the current story graph (and perhaps some other data as well), and perform a CBR lookup on the Case Library which returns a result

[3]Mexica's implementation is informed by the concept of writing as design [Sharples, 1999] which specifies that creative writing is conceptualized as a cycle of cognitive engagement and reflection.

which in turn is adapted for and inserted into the story under construction.

MS3s often include a form of CBR that is capable of generalizing or altering its query in some (ideally) useful way if no results are found. These MS3s are often capable of using CBR to enhance their own CBR lookup procedures. By using CBR to find "realistic" or "likely" alterations to an unsuccessful query, the MS3 is able to create a new query without making it so general that it looses its meaning. An example of this from Minstrel is an attempt to decide what object a princess should find in a forest for some story that's being generated. If Minstrel is unable to find any stories in which a princess finds an object in the forest, it might decide to transform the princess into something similar and perform another query. By using a few CBR searches, the system might determine that hermits and princesses are fairly similar in their activities and thus decide to attempt a new search using a hermit instead.

There are two final elements that are often included in the MS3s to support CBR: a carefully curated Case Library and good result selection routines. Some combination of these two elements is crucial largely due to the fact that the systems don't understand the symbols that make up the stories in the library and thus without direction, will make choices that result in incoherent stories. The larger a story library gets, the more symbols there are and the higher the likelihood that two or more of those symbols will be put together in a pattern that is deemed inappropriate by human judges. Curation and careful design helps the stories and the domain work together in harmony and thus can represent more concepts while producing fewer undesirable elements. Further, some sort of intelligent selection routine that filters CBR results can help prevent incoherent and inconsistent patterns from being added to a story.

2.2.4 Two Systems Through the MS3 Lens

Before the primary MS3 systems are introduced, it is useful to examine two other CBR systems to distinguish MS3s from CBR story generators more broadly. The first, Say Anything [Swanson and Gordon, 2008], is a remarkable data-driven system that mines a huge corpora of online content to take a query sentence and find one to succeed it. Say Anything uses a CBR lookup with adaptation to find the next sentence and modify it so that it works within the context provided by the query sentence. Also, by recursively running the system, it is possible to generate stories. Looking at it from the perspective of Minstrel, it seems like a good candidate to be an MS3. Under the hood, Say Anything is essentially a set of really good algorithms for assessing whether two sentences have similar meaning and content. Applying these algorithms to a story element (in this case a sentence) lets the system find analogical elements just like an MS3 does but that's where the similarity ends. Say Anything doesn't ever get much below the text version of a sentence and thus can't be said to be manipulating symbols, do any sort of reflection and improvement of a story under construction, or make any intentional choices about the stories that it produces.

KI-CBR [Gervas et al., 2005] is another excellent CBR based storytelling system that shares many similarities with Minstrel but does not quite fit the framework of an MS3. To create a story with KI-CBR, a user dictates characters and a broad plot (in essence, supplying a template) and then the system uses CBR with adaptation to find story fragments which can be used to fill in the template. Interestingly enough, this system represents its stories as a sequence of Proppian functions (archetypical story events) rather than following the convention of merely using these functions to inform the creation of more specialized events. This forces KI-CBR to use some fairly complex natural language generation to

	Say Anything	KI-CBR
Symbols and Patterns:	0/3	1/3
- System manipulates symbols and patterns	-	-
- Templates	-	-
- Natural Language	-	+
Construction and Reflection	1/3	0/3
- Constructive Mode	+	+
- Reflective Mode	-	-
Case Based Reasoning	2/3	2/3
- Uses CBR	+	+
- Intelligent selection of results	+	-
- Curated case library	-	+

Table 2.2: MS3 attributes of Say Anything and KI-CBR

render english text from the proppian functions, satisfying another requirement of an MS3. Despite all of these similarities, KI-CBR has absolutely no Reflective mode. In other words, KI-CBR is a wonderful tool for sequencing together pre-written plot elements in a desired order but it does not mutate its queries and thus doesn't retrieve as varied a set of possible responses to a given query. Thus, it fails at one of the primary goals of MS3s: to invent and embed "new" content in its creations. In Table 2.2, Say Anything and KI-CBR are compared against MS3 attributes.

2.3 Overview of The Core MS3 Systems

The four best examples of MS3s are Minstrel, Mexica, Riu, and Skald [Turner, 1993, Pérez y Pérez, 1999, Ontañón et al., 2010, Tearse et al., 2014]. As I stated earlier, although the MS3 category is named after and centered on Minstrel, both Riu and Mexica have unique contributions that help define the category. Thus, understanding how each one works is clearly important to provide a complete picture of the category as a whole. At a fundamental level, Skald and Minstrel are virtually identical (the upgrades that make Skald unique do not alter its symbolic reasoning, construction/reflection loop, or CBR,) so in the interest of not presenting the same material twice, I will leave discussions of Skald for later chapters. Although more details on these systems can be found in the scientific literature cited above, this section will provide a brief overview of Minstrel, Riu, and Mexica using a common vocabulary.

2.3.1 Minstrel

Turner recounts that Minstrel started as an attempt to utilize Propp's theory of story morphology [Propp, 1968] to generate stories [Turner, 1993]. By the time it was completed, it had moved away from Propp's grammar-like approach, instead developing an approach where stories are written a piece at a time by finding useful snippets of other stories and transforming their content to fit into the current context and constraints. Originally Minstrel used a fairytale domain because such stories tend to be formulaic, allowing for a wide cast of characters, objects, and events so long as a standard structure is followed that helps convey a chosen moral. Later versions of Minstrel diverted away from morality themes in order to work with a larger number of genres but both systems use templates as pre-existing plots for its story construction. These stories tend to be short but highly detail-

oriented, often impressing readers with the chains of properly interrelated events and the inclusion of key details. This is especially impressive when one knows that all story details are represented as symbols; Minstrel has no deep ontology of its symbols and thus can't reason deeply about the symbols that it manipulates. The combination of detail oriented story elements and lack of understanding of what's being manipulated means that Minstrel needs to keep its actions, objects, and items simple since too much specificity can easily make components that have only one use and thus break all stories they are imported into.

The third element of the "Symbols and Patterns" requirement, text generation, is much less impressive in Minstrel than in some of the other MS3s. Story graphs are rendered into English using a handful of templated sentences which can be filled in with whatever details are required. These templates are selected based on what sort of node is being rendered and sometimes even what action that node represents.

As far as Constructive and Reflective modes are concerned, Minstrel is the least clear-cut of the core MS3s. Minstrel's main loop employs a planning algorithm which tests and applies a number of modular plans (similar to hierarchical task networks [Erol, 1996]) to the story. Minstrel's plans can be categorized as detectors (which find problems and thus fit into the Reflective mode), modifiers (which fix problems and thus also fit into the Reflective mode), and adders (which add to or fill in the current graph and are thus of the Constructive mode). Although there is nothing to prevent someone from creating a plan that both reflects upon the current story and adds to it, by convention Minstrel's story generation plans focus only on construction or reflection.

The third set of criteria, that an MS3 employs a CBR system to find appropriate story fragments, intelligently select amongst them, and maintains a well

curated story library is also easily satisfied by Minstrel. When filling in details of a particular node in the story graph, Minstrel makes a call to its 'TRAM' system which performs one or more CBR lookups on the story library, finds fragments matching the context of the node being operated upon, and uses those details to fill it. Minstrel uses a carefully designed story library to do these lookups for without such curation, lookups would either always fail or would have a high degree of incoherence in their results.

2.3.2 Riu

Riu is a system that tells stories about a robot, Ales. Unlike the other two core MS3s, Riu is designed to work interactively with a user to generate stories. A user is prompted with the beginning of a story and given a handful of options for how he or she would like the story to progress. One interesting quirk is that Riu can reject the user's choice. It only does this when the protagonist can "imagine" some potential outcomes of the chosen action which are undesirable to it, but this demonstrates some Reflective mode filtration capabilities on the part of the system.

Internally, Riu, like other MS3s, has a Case Library (called the Lost Memories), this is filled with pre-authored symbolic graphs representing the plots of stories. Also like other MS3s, Riu uses its own specialized form of story representation. Riu's authors chose to use Force-Dynamics [Talmy, 1988] [Talmy, 1985] as a way of encoding stories, which take the form of a graph of symbols (in this case strings) with links between them. This representation has proven to be fluid, abstract, expressive and by comparison, makes Minstrel's more heavyweight and detail oriented format seem like one used for a simulation (see Figures 2.2 and 2.3 below for a graphical comparison). A consequence of this more abstract and fluid format

is that Riu's scenes need to be short in order to maintain causal coherence. Thus if larger stories are desired, they tend to take the form of a series of detailed but disjoint events. This limitation of the system makes sense given the domain - Ales, like a human can be expected to recount an "average" day, as a handful of interesting events along with plenty of interstitial ones that are appropriately skipped with phrases such as "an hour or so later".

Like Minstrel, Riu uses templates both for story scaffolding and text rendering. That being said, Riu's form of text generation is one place where it is markedly more advanced than either Mexica or Minstrel. Each of the stories in the case library comes with templated text and a finite state machine allowing the same event to be told in a number of unique and coherent ways. This is a wonderful improvement as it allows Riu's authors to leverage a little extra authoring effort into a marked improvement in output diversity.

2.3.3 Mexica

Mexica is the third and last of the core MS3s selected for this discussion. This system is described as a "plot generation," rather than storytelling, system chiefly because it creates a graph of story events but does not include details or track objects. Despite this, Mexica does generate English text that tells a story and many of its capabilities are distinct and interesting from both Riu and Minstrel while its components fit solidly into the MS3 framework.

Mexica was originally programmed with a domain enabling it to tell stories about the Aztec people. Like Minstrel and (arguably) Riu, it is capable of telling other stories so long as it's provided with adequate domain knowledge. This takes the form of two things: a document outlining a list of viable actions and a set of Previous Stories. Each listed action is bundled with a set of preconditions,

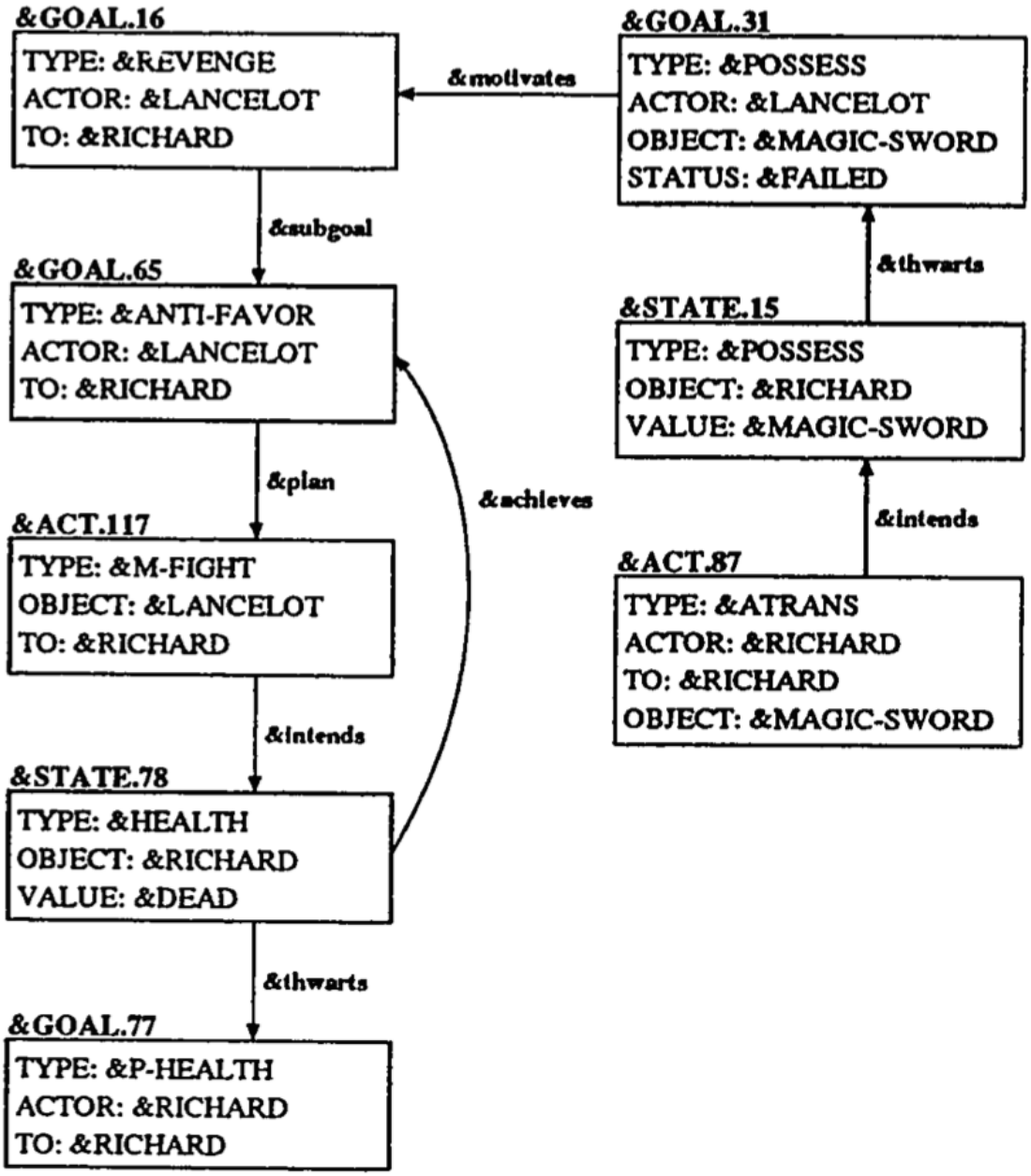

Figure 2.2: An example of a Minstrel story graph from Turner's dissertation [Turner, 1993].

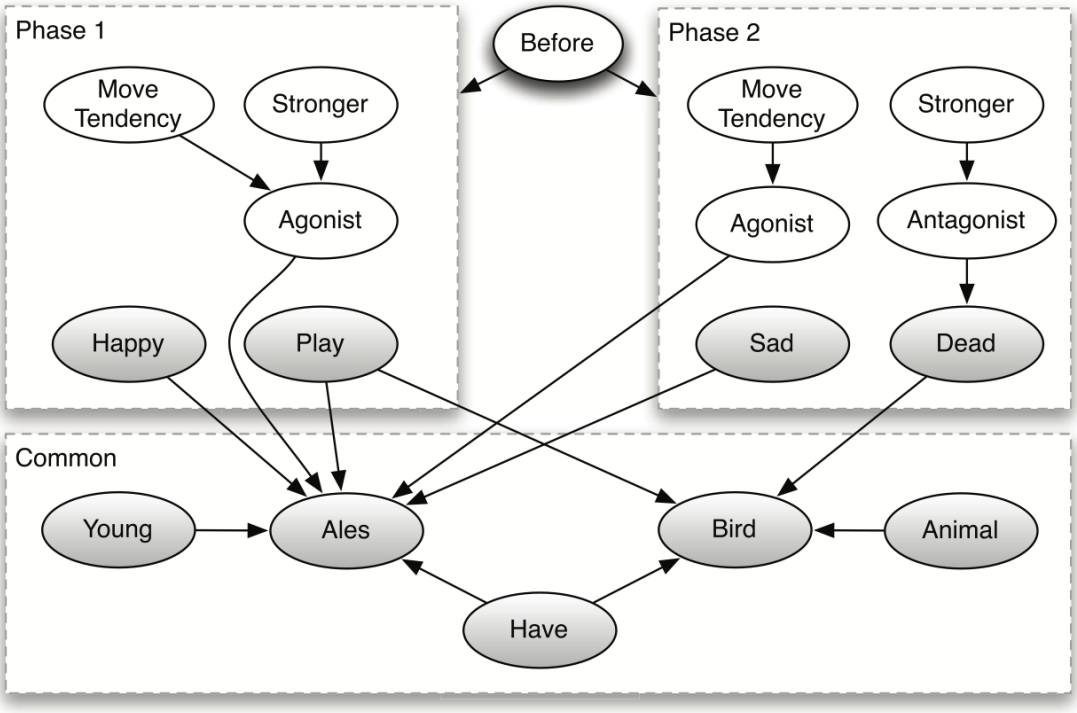

Figure 2.3: An example of a story graph from Riu [Ontañón et al., 2010]

postconditions, and an English rendering with placeholders in it. Mexica's stories take the form of a temporal progression of events and in keeping with the framework, every element of every story is a symbolic string with no attached rules for inferring meaning. Events in Mexica contain only the action itself, a subject and object for the event, and a list of context changes for each character who witnessed the event (which are governed by the action's postconditions). When the story requires a next event, it is selected by doing a CBR lookup in the Previous Stories to find situations in which one of the characters has the same or a similar context to one of the characters in the current story. When something appropriate is found, the matching character in the current story will take the action found in the retrieved story, causing the next event in the story. Since everything is primarily driven by these lists of character contexts, actions are essentially a description for how to transmute one character context (the precondition) into

another (the postcondition). That being the case, Mexica's searches are less computationally complex (being list comparisons) than the graph comparisons that Minstrel and Riu perform. Thus Mexica is in a unique position amongst MS3s where it is free to have comparatively many more actions in the domain (each containing their own English rendering) before the complexity costs of its search impairs the functionality of the system.

Mexica is the system from which the names of the Construction and Reflection Modes are taken. Although many storytelling systems have components that build new content (Construction) and then filter, consistency check, or modify that new content (Reflection), Mexica is the only core MS3 that explicitly toggles between the two modes rather than having intertwined components which behaviorally fall into one or the other of these two categories.

As far as Case-Based Reasoning is concerned, all three requirements have already been touched upon. When Mexica needs the next element of a story, it performs a CBR lookup to find the next event. When an exact match isn't possible, Mexica is able to relax its matching requirements so that only some percent of the current context elements must match an event in the Previous Stories in order for it to be used (leaving some preconditions unmatched). This process is effective at intelligently finding useful and appropriate events. As for the third requirement, Mexica's library is carefully constructed and curated. If it were not, inappropriate events would have an easier time working their ways into stories and break their cohesion.

2.3.4 A Comparison of Minstrel, Mexica, and Riu

Now that each of the core MS3s have been discussed individually, conclusions can be drawn by comparing and contrasting these systems with one another in

an attempt to better understand the capabilities of the class as a whole. This is summarized in Table 2.4.

	Minstrel	Mexica	Riu	Say Anything	KI-CBR
Symbols and Patterns:	3/3	2/3	3/3	0/3	1/3
- Manipulates symbols and patterns	+	+	+	-	-
- Templates	+	-	+	-	-
- Natural Language	+	+	+	-	+
Construction and Reflection	2/2	2/2	2/2	1/2	1/2
- Constructive Mode	+	+	+	+	+
- Reflective Mode	+	+	+	-	-
Case Based Reasoning	3/3	3/3	3/3	2/3	2/3
- Uses CBR	+	+	+	+	+
- Intelligent selection of results	+	+	+	+	-
- Curated case library	+	+	+	-	+

Table 2.4: Comparison of various Case-Based Reasoning with regards to MS3 attributes

Although each of these systems uses CBR to tell a story, it's clear, albeit unsurprising, that the MS3s stand out when judged by their own requirements.

What is striking about the core MS3s is that the same techniques are employed in similar ways to accomplish quite different results. Minstrel's operation revolves around a graph of frames, each of which is a self-contained cluster of details about a single event or state (e.g. "Knight fights Troll with Sword"). Connections between frames imply, but do not explicitly represent, a large amount of story meaning.

When interlinked frames are moved from one story into another, they are often surprisingly compatible or impressively not. This allows Minstrel to create stories with a large number of implied meaning which humans find either impressive or humorous depending on how well Minstrel connects the frames together.

Instead of trying to determine which details can optimally be used to fill a particular story, Mexica takes another approach with it's core question, "what happens next?" By maintaining a list of current contextual elements, Mexica is able to find events in stories with matching contexts and choose to import the next event for the current story from one those matches. Mexica does not track special objects and lets the reader decide how one event flows into another which makes it highly successful at generating coherent plots without getting mired in the details.

Operating somewhere in-between Minstrel's detail shuffling and Mexica's event sequencing, Riu takes a scripted set of events and tries to find a pleasing set of motivations and sensible results for the events. This is an interesting focus since Riu attempts to create novel and interesting support for pre-existing story elements rather than having complete control over the story as Mexica and Minstrel do.

These three ascending levels of operational detail are clearly where the systems' strengths lie ; the authors did not include mechanisms to support creativity in areas that are not the focus for the system. Minstrel's templates are fixed so all stories produced with the same template will follow the same basic path. Riu has a similarly pre-defined set of events that it progresses through every time it tells a story. Neither of these systems can do any event resequencing. Similarly, the details of Mexica's events are either non-existant or fixed (depending on your point of view). The resequencing that Mexica focuses on uses blocks that all have a

predefined set of preconditions and postconditions and thus there is no machinery in place to let any of those change. Beyond even the lack of machinery, each of the systems' basic story representations are designed to facilitate the strength of the system but hinders operation at other levels of detail. Mexica's lists of contexts don't have the fidelity to facilitate a high level of detail, Minstrel's story graphs are too cumbersome and full of details of different levels of importance to ever be translatable into a context list for Mexica, and although Riu's Force-Dynamics representation could be translated into a context list for Mexica, it lacks the patterned structure required to ever become a Minstrel-style story graph.

To briefly summarize these arguments: these three systems are (unsurprisingly) each specialized for unique goals. While they do operate using similar machinery, they demonstrate that MS3s can successfully function at different levels of detail and with different goals. While they all aim to generate stories, they do so in different manners and have different aptitudes and weaknesses (these will be discussed in more detail in later chapters.)

Despite these differences, Mexica, Minstrel, and Riu are all CBR systems that do some degree of symbolic or analogical reasoning. Each system has its own special twists on old concepts and an array of custom built components but it is satisfying to know that almost every custom piece has an analogous part in other MS3s. There are a few papers that attempt to compare and contrast the qualities of individual storytelling systems [Zhu, 2012] [Pérez y Pérez and Sharples, 2004a] [Wardrip-Fruin, 2009] but generally, nobody has spent time looking at the many similarities between systems. Each new research paper essentially examines its target in a vacuum despite the fact that many of these systems have similar enough components to present a common framework. To my knowledge the following table is the first ever translation table of elements common to an entire class of

storytelling systems. Although I will continue to refer to elements mostly by the names provided in this table, I hope that it will support direct discussion about systems on a component by component basis.

Concept	Minstrel Remixed	Mexica	Riu
Conceptual Representation	Conceptual Dependency	N/A (Custom Made)	Force-Dynamics
Story Element	List of Symbols called a Frame. Single Node in Graph	Single Action and the Story World Context of its actors.	Roughly one symbol plus all symbols linked to it in the story graph.
Story	Single Graph in Story Library	Single Element in Long Term Memory	Single Lost Memory
Story's Data Representation	Graph of Story Elements. Links indicating some contextual connections.	Contextual Structures (Atoms)	List of Scenes. Scene = graph containing story elements. Graph links => symbol relationship.
Case Library (For Recall)	Story Library	Previous Stories	Lost Memories
Templates	Generic Story Templates	N/A	Lost Memories, Pre-authored story
Recall	Recall	Exact Recall Strategy	Recall
Recall with Generalization	Adaptive Recall (with TRAMs)	Inclusive or Dynamic Recall Strategies	Recall
Recall Criteria	N/A	Tension Guidelines	N/A
Recall Retrieval Strategy	Retrieve similar patterns then weight by element matches	Retrieve based on % of keywords matched.	Retrieve by keyword matches then weight by pattern similarity

Concept	Minstrel Remixed	Mexica	Riu
Toggles: CBR construction and reflection?	Yes	Yes	Yes
Reflection: Story Mutators	Author Level Goals/Plans	Final Analysis	N/A
Reflection: Story Filters	Author Level Goals/Plans and TRAMs	Precondition Satisfaction while in Reflection Mode	Character's "Things To Avoid" list
Text Rendering Method	Fill english templates based on story element contents	Each action has a templated sentence	FSMs that help/vary rendering

Table 2.5: Comparing components and processes across Minstrel Remixed, Mexica, and Riu

Chapter 3

Five - A Simple Story Generator

Below is FIVE, a simple story generator similar to a number of systems that create output using a few simple rules and some pregenerated content (for example, [Montfort, 2008, Queneau, 1961]). FIVE is easily runnable by humans and is named for the amount of time it took me to create its base algorithm (five minutes) and is inspired by Mexica's event ordering storytelling style (see previous chapter for a discussion of Mexica). The chief intent when engineering FIVE was to make as simple a story generator as possible but leave room for easy modification. Since it doesn't include CBR, FIVE isn't quite a Minstrel Style Storytelling System (MS3) but its qualities as a story generator facilitate discussion of Construction vs Reflection, utilize the reader's proclivity to add meaning to stories, and provide a simple example for developing techniques for estimating the size of an MS3's story space. The domain model in FIVE is tuned to take advantage of the common concept of setup, action, result that is found in many stories as well as the fact that humans are excellent at taking a string of actions and interpreting them as a cohesive storyline. Tuning FIVE took an additional thirty minutes to complete, orders of magnitude less than our other story generators. I will use FIVE as an example throughout this dissertation to clarify various properties of

Minstrel style story generation, demonstrate improvements that could be made, and act as a base case to demonstrate how other story generators have improved upon this base case.

3.1 Example Stories Generated by FIVE

Here are three example stories generated by FIVE. The titles were added to provide some clarity but different readers have different interpretations and would title the stories differently. While these stories are by no means ideal, they do satisfy the requirement of conveying meaning through a series of events, being unique and different outputs, and generally making sense (albeit requiring a little creative gap-filling on the reader's part).

> **Insider Information**: peasant gave message to king. king asked for help from peasant. peasant shared secret with king. king bargained with dragon. dragon gained affection of king.
>
> **The Warning**: knight asked a favor of princess. princess bargained with peasant. peasant shared secret with princess. princess talked to knight. knight fled from peasant.
>
> **Vengeful Wizard's Unwitting Accomplice**: king gave message to wizard. wizard gave gift to king. king asked for help from wizard. wizard tricked king. king killed peasant.

3.2 Running FIVE

In order to run Five, the 'system' (i.e., you) will require a tool to generate random numbers from 1 to 6 and a buffer to store output in. If you so desire, you can operate in different story spaces by altering the initial conditions or the tables that your system will be using. Record your chosen initial conditions and then either follow the steps in the Generational Algorithm followed by the Filter

#	Actors	Initial Actions	Normal Actions	Final Actions
1	knight	stole from	bargained with	killed
2	king	fought with	asked for help from	gained affection of
3	princess	asked a favor of	tricked	received gift from
4	dragon	found secret about	gave gift to	publicly praised
5	peasant	gave message to	talked to	denounced
6	wizard	is in love with	shared secret with	fled from

Table 3.2: Random story element tables for FIVE

Algorithm or use the Optional Integrated Algorithm which is a slightly more complex version of the Generational Algorithm with the Filtration steps built in.

Initial Conditions

- Random Number Generator Weights: All numbers are equally weighted.
- TotalNormalActionsInStory: 3
- NumberofActorsInStory: 3

Generation Algorithm

- Randomly choose *NumberOfActorsInStory* actors from the list above.
- **To generate a sentence**: randomly choose two actors, an initial actor and a secondary actor and place the initial actor in your buffer followed by an action followed by the secondary actor.
- **Generate a sentence** using a random element of the *Initial Actions* list as the action.

- **Generate *TotalNormalActionsInStory* additional sentences** using randomly selected *Normal Actions*.
- **Generate a final sentence** by using a random element of the *Final Actions* list as the action.

Filter Algorithm If any of the following conditions are not met, the story is not acceptable and must be regenerated

- All actors chosen in step 1 must be unique.
- Every sentence must have its initial actor be the same as the secondary actor of the previous sentence (with the obvious exception of the first sentence.)
- Every sentence must have its initial actor be different than its secondary actor.
- No sentence may have the same action as any other sentence.

It should be clear that the Generation Algorithm is a very simple Constructive Mode while the Filter Algorithm is a simple Reflective Mode (see Section 2.2.2 for discussion of these two modes). Because FIVE is an extremely simple system, it is possible to calculate both its unfiltered generative space and its Possible Story Space (PSS). 6 actors of which 3 are chosen means 216 unfiltered possible combinations of which only 120 are valid (those in which the 3 chosen actors are unique). The first sentence has 6 possible actions, all of which are valid and 9 possible combinations of first and second actors of which 6 are valid (first actor is not also the second actor). The second, third, and fourth sentences have 216 possible sets of actions of which only 120 are valid (those sets which contain 3 unique actions) and each of these sentences have 9 possible combinations of first

Generative Element	Unfiltered Options	Valid Options
Actors	216	120
Initial Actions	6	6
Initial Actors	9	6
Sentence 2,3,4 Actions	216	120
Final Action	6	6
Sentence 2,3,4,5 Actors	6561	16
Total Unique Stories	99,179,645,184	49,766,400

Table 3.4: Calculating FIVE's Possible Story Space

and second actors of which 6 pass filter #3 (no duplicate actors) and only 2 pass filter #2 as well (first actor matches second actor of previous sentence). The final sentence has 6 more possible actions which all are valid and 9 possible sets of actors of which only 2 are valid. This means that FIVE can generate just under 100 billion (9.918 * 10^10) stories thanks to its domain and Generative elements and of those only 500 in a million are valid (or 49,766,400 to be exact) due to the filters that are its Reflective elements. This example provides a clear demonstration of the expansive and restrictive power that the Constructive and Reflective elements of an MS3 wield over its PSS.

Although it is unknown exactly how many times a human might be willing to generate and regenerate stories, I believe it is safe to argue that the rejection rate of FIVE's filters is sufficiently high to prevent any reasonable person from attempting to generate even a single valid story using the provided method. That being said, by simply attempting to generate a story or two it is clear that much more efficiency can be achieved by constructing a single algorithm that integrates the provided Constructive and Reflective elements. Below is the "optional" Integrated

Algorithm which is both the suggested method for humans to generate stories using FIVE and an example of the way that most MS3s achieve greater efficiency by creating similar integrated algorithms.

Optional Integrated Algorithm *(alterations from the Generation Algorithm are **bold**)*

- Randomly choose NumberOfActors **unique** actors from the list above.
- To generate a sentence: **Use the previous sentence's secondary actor for the initial actor (or in the case of the first sentence, a random actor) and choose a random actor that is not the same as the initial actor to be the secondary actor**. Place the initial actor in your buffer followed by an action followed by the secondary actor.
- Generate one sentence using a random element of the Initial Actions list as the action.
- Generate TotalNormalActionsInStory more sentences using randomly selected normal actions (**reselect actions if they have already been used in the story**).
- Generate a final sentence by using a random element of the Final Actions list as the action.

The Optional Integrated Algorithm is clearly far more efficient than the original Generation and Filter Algorithms but, even though both processes yield precisely the same output, it is far harder to exactly determine the size of the PSS under the Integrated Algorithm. This is also true of MS3s which use their own integrated algorithms and much more complex processes for Generation and Filtration. In future chapters I will use FIVE as a base case to aid in the clarification and discussion of various subsystems of MS3s.

Chapter 4

Skald: An Implementation and Upgrade of Minstrel

Although Minstrel is a well-discussed system that has been cited many of times and even been made the focus of papers and book chapters [Wardrip-Fruin, 2009] [Mateas and Sengers, 2003] [Pérez y Pérez and Sharples, 2004b] , it has never been directly explored by any researcher save Scott Turner, its creator. While many insights, descriptions, and implementation details can be found in Turner's dissertation and subsequent book [Turner, 1993][Turner, 1994], his work can best be characterized as an introduction of Minstrel, that is to say, an overview of the inner workings of the system along with a presentation of the proof-of-concept experiments designed to demonstrate that it works. Being a system that has been lauded as a landmark in the field of computational storytelling [Wardrip-Fruin, 2009], further exploration beyond Turner's initial experiments is warranted. This task is unfortunately made difficult by the fact that no code for Minstrel was ever made available. Due to these two facts, we set out to create a new MS3 that was as close to the original version of Minstrel as possible with the intention of further exploring the capabilities of Minstrel-Style Storytelling.

The new program, being a reimplementation of Minstrel, went by the moniker, "Minstrel Remixed" and took a number of years to create. Turner's dissertation being a presentation of academic research, rather than a document with detailed specifications about the program, naturally lacks sufficient detail to ensure that Minstrel Remixed has complete fidelity. On a few rare occasions this is doubly true as the ambiguities in Turner's book and dissertation sometimes hint at different specifications for the same subsystem[1]. Although these conditions forced us to make a few guesses and fill in conceptual holes here and there, we are convinced that Minstrel Remixed is a very close approximation of its progenitor.

As we analyzed Minstrel Remixed in a number of different ways, we noticed deficiencies that could be improved by upgrading the system with new functionality [Tearse et al., 2014, Tearse et al., 2012, Tearse et al., 2010b, Tearse et al., 2011a, Tearse et al., 2010a, Tearse, 2011, Tearse et al., 2011b]. As more potential improvements were identified, it became clear to us that we could support better story generation and explore more of the capabilities of MS3s by creating a new system based on the Minstrel Remixed codebase but which would break from our self-imposed limitation of strictly adhering to the outline provided by Turner. The result of these improvements to Minstrel Remixed, and arguably, Minstrel itself, is called Skald (the Norse word for 'minstrel'). Although Skald contains a large number of improvements over Minstrel remixed, these are all aimed to expand our capabilities in a few areas: transparency in story generation, exploration and measurement of the subtle workings of individual modules, improved stability, and better story output in terms of speed, size, and coherence. The details of the implementation of Skald are the content of this chapter and a copy of the source

[1]For example the dissertation seems to suggest that nouns are globally referenced objects while the book seems to suggest that they are locally referenced. This may not seem to be important but each requires a different implementation which could have effects on story generation.

code is publicly available for download at `https://minstrel.soe.ucsc.edu/`.

4.1 Overview

Skald is written in Scala 2.8 (a derivative of the Java programming language) and it includes the original components of Minstrel described in Turner's dissertation [Turner, 1993], as well as a few changes (which are conveniently grouped and discussed at the end of each section of this chapter). Turner's most broad description of Minstrel is as a system which simulates the actions of a human author in order to produce stories. At a high level, Skald does not deviate from these original specifications so Turner's original description still remains valid. Some modules are low level simulations of problem solving processes (akin to finding a replacement for absent jelly when one wants a peanut butter and jelly sandwich) while others are modeled on a storyteller's higher level authorial goals such as working from a given story outline or trying to enhance tragedy or suspense. Skald applies all of these modules to on an input story (be it empty or nearly complete), decides on an area of the story to be improved, attempts to flesh it out, and then moves on to another area. Once all of the various modules have checked over a story to ensure that it is complete and contains no flaws (that the modules can detect), Skald will store the story in its library for later reuse and then present it to the user.

4.2 Skald's Architecture

Skald has a complex architecture including many subsystems and components. These are organized largely into three separate subsystems with a fourth organizational system to help them interact: the story library, TRAMs, ALPs, and the

Bard respectively. TRAM stands for Transform Recall Adapt Method and "the TRAM system" refers to the Case-Based Reasoning (CBR) component of Skald. The TRAM system contains a number of individual TRAMs which each contain some logic for modifying search queries. The TRAMs and accompanying support mechanisms are used to do problem solving and generate the fine details of a story. ALP stands for Author Level Plan and these are used to direct the broader themes of a story as well as being used to enforce a variety of consistency constraints. Together the planning of the ALP system and the case-based reasoning of the TRAM system allow for complete stories to be pieced together. Finally, the Bard is responsible for managing communication between the components and initiating the story generation process. In Skald, each of these 4 modules is run as its own process which sends TCP/IP signals to one another. Although we never did so, this means that Skald could be run from 4 separate computers to speed up processing.

4.3 The Story Library

4.3.1 High Level Overview

The Story Library is the simplest of the four primary modules in Skald since it is essentially a data storage system with minor functionality. That being said, stories themselves are fairly complex data structures and understanding how they are represented and manipulated is crucial to understanding how other modules operate. In Skald, the most fundamental unit of a story is a symbol. These are grouped together into story elements that are called frames — and frames are grouped together into a graph which represents a story. The Story Library contains many such graphs in two categories: templates and stories. Templates

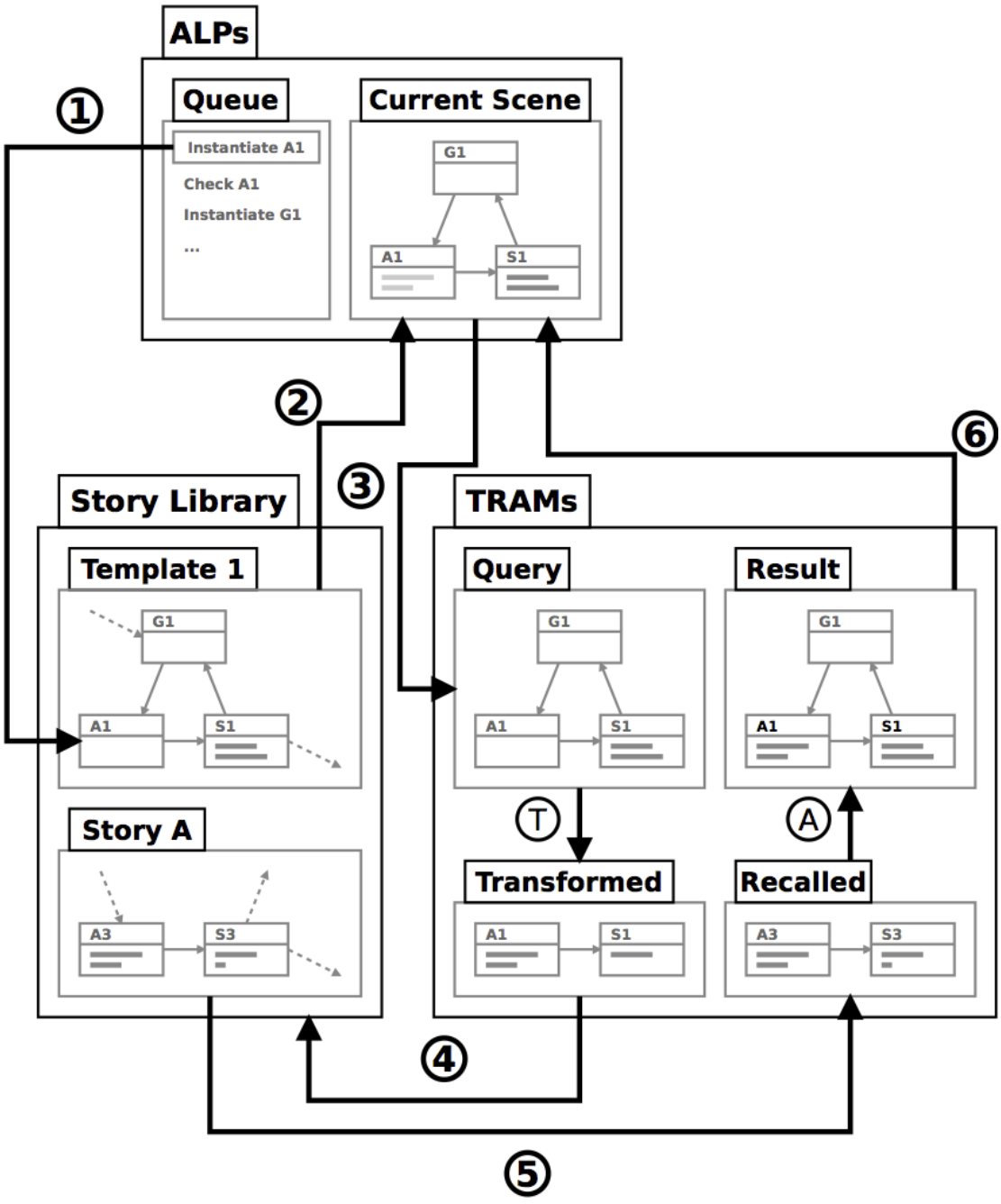

Figure 4.1: Interactions between the ALP system, the TRAM system, and the story library during the instantiation of a single frame. A full description of this image is found in Section 5.8.

are used to build the initial structures for new stories while the stories are used for CBR lookups to help fill in details of new stories.

In addition to the Minstrel style graphs, Skald also has the ability to represent its stories as a series of changes over time known as a Lineage. Using this construct, operators can view the story's creation at any point in time or roll a story back to a specific point in its generation in order to explore alternate choices. Lineages contain commands written in the Story Implementation Language (or SIL). Being easier to read and write than previous incarnations of Minstrel's graph representations, SIL is the language used for stories stored in files. Sets of SIL commands are routinely converted into graphs and vice-versa over the course of a story's generational lifespan.

4.3.2 Data Structures: Symbols, Frames, and Graphs

Original Symbol	Unifies To
Generic Authority Figure, "Judger"	Concrete King, "Arthur"
Generic Person, "Noticer"	Concrete King, "Arthur"
Generic Tool, "Thing"	Concrete Tool, "Shovel"
Concrete Knight, "Lancelot"	Concrete Knight, "Percival"
Generic Place, "Setting"	Generic Place, "House"

Table 4.2: Generic and Concrete symbols

Symbols are represented as either a string (e.g. "happy", "fight", etc.) or a more complex reference to a noun which has a name and a type. Both string and noun reference symbols can be either concrete or generic and no story can be complete if it contains any generic symbols. All stories contain a unification table that describes which symbols have been unified to which other symbols or,

in other words, which symbols to replace with which other symbols. Table 4.2 is an example of one such unification table with original symbols on the left and the symbols they unify to on the right.

A few things should be noted about this example which are true for the system as a whole:

- String symbols rarely appear in unification tables as they don't often need to unified (note: the table above contains only noun symbols).
- Symbols can have a many-to-one relationship (for instance Arthur who takes on the two roles of "Judger" and "Noticer" in the example)
- Symbols may not have a one-to-many relationship. An example breach of this rule would be if there were two people who the "Judger" unified to.
- Almost all entries in the table cause a generic symbol to unify to a concrete symbol but in rare situations concretes can unify to other concretes (generally when some impasse is being corrected) or generics can unify to other generics (which is supported, though the current implementation of the TRAMs in Skald don't make use of this feature).

As was mentioned before, symbols are grouped together into Frames in order to represent a complete story element. Frames have a number of types which are defined in the domain (our domains primarily use action, state, and goal types). Each type of frame has a pre-defined set of slots which contain key-value pairs, the mappings of which are derived from Roger Schank's Conceptual Dependency [Schank, 1972]. A partially defined example frame is displayed in Figure 4.2.

This example demonstrates a number of things:

- Frames can contain Empty and/or Unknown slots. These slots may be filled with any symbol over the course of story generation. Empty slots indicate

Key	Value
Type	“Fight”
Actor	Concrete Knight, “Lancelot”
Object	Generic Weapon, “Thing”
To	< Unknown Slot >
From	< Empty Slot >

Figure 4.2: Lancelot used a thing to fight with something

an acceptable absence of information while all Unknown slots must be filled for a story to be considered complete.

- Frames can contain generic nouns which function as slightly more specified Unknown slots - they provide constraints on what can be unified with the slot.
- Frames can also contain concrete nouns (“Lancelot”) or strings (“Fight”) which are fully specified and in nearly all cases are considered to be final.
- Frames are standardized and some slots aren’t used in all contexts. For example, in the frame for the Fight action, the ‘To’ slot indicates ‘the thing being attacked’ while the ‘From’ slot is unused.

In order to represent a story, a graph is constructed using frames as nodes. The edges connecting the frames are each labeled to provide an unambiguous meaning to the whole structure. Although the frame types and edge labels are all mutable components of the domain, they are some of the more universal components (most domains will have identical or similar frame types and edge labels). Most commonly seen frames are of the types “Goal”, “Act”, or “State” which roughly define mental action, physical action, and results respectively. These frames tend to be linked by “plans”, “intends”, and “achieves” links to form a GAS trio (which

stands for a Goal Act State trio of nodes). The end result is an act that achieves a goal by causing a change in state. The bulk of most stories are formed from one or more of these GAS trios and Figure 4.3 is a fairly simple story framework constructed by an initial state motivating one such GAS trio. It should be noted that node types and edge labels will vary by domain since many concepts can't be easily expressed solely with GAS trios. Thus different structures will be required for different genres of stories. Skald and Minstrel both make infrequent use of a number of edges beyond plans, intends, and achieves: precondition, subsumes, supersedes, motivates, thwarts, accidents, and hasSubgoal. An additional example is Minstrel's 'Belief' nodes which we removed from Skald since we had sufficient content and expressivity without including more complex simulation of characters' mental processes.

Once a graph is completely put together it can be stored in the Story Library either as a story or a GST (Generic Story Template). These two concepts are identical save that templates can contain generic symbols and unknown slots while stories require all of their slots to contain a concrete symbol or be empty. Using pre-loaded stories and GSTs, the story library starts out with a basis for generation, but it can also incorporate any new stories that are generated allowing the system to "learn" from what it creates.

4.3.3 Story Implementation

As was mentioned in the introduction to the Story Library section, the Story Implementation Language (SIL) is an alternate format for story data that has different affordances than the standard graph structure. SIL still contains all of the concepts found in graphs (frames, slots, etc.) but in SIL each alteration to the graph is represented as a single command and thus a story can be seen as a

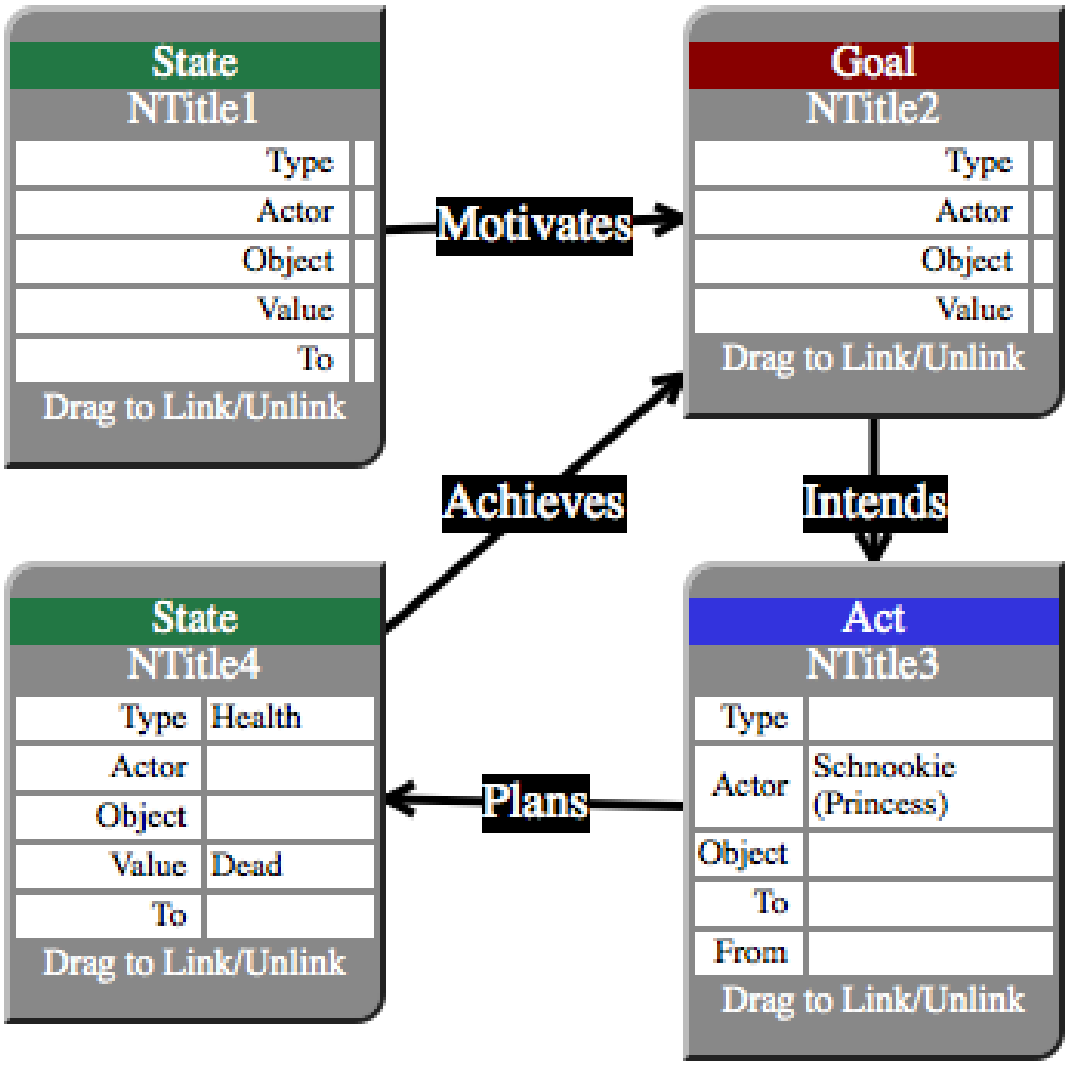

Figure 4.3: A Goal-Act-State trio where Princess Schnookie plots a murder.

list of commands rather than a graph. An example of a story template written in SIL is found in the left column of Table 4.5.

SIL is a simple language. The complete list of understood commands is as follows:

- new noun: name[String], type[String] , is_generic[Boolean]
- new stringref: name[String], value[String]
- unify: old[String], new[String]
- new node: name[String], type[String]
- new edge: from[String], to[String], label[String]
- update node: name[String], attribute[String] , value[String]
- delete node: name[String]
- delete edge: from[String], to[String], label[String]

New noun, *new stringref*, and *unify* are used to control the unification table for the story, while *new node*, *delete node*, *new edge*, and *delete edge* alter the structure of the story and update node is used to place content into the structure.

To facilitate authoring, Skald also has a stand alone authoring tool called StoryWriter that will compile simple English sentences into SIL files. Table 4.5 shows an example of a StoryWriter graph specification in the right column. For more details of the StoryWriter syntax and transformation into SIL, see the tools.storyWriter module in the downloadable Skald source code.

4.3.4 Lineages

One of the principals that we strove to uphold while creating Skald is to enable operational transparency in what would otherwise be a convoluted and opaque set of graph transformations. To facilitate this, the ability to view a story's progress

SIL	StoryWriter
meta: name, INSULT_BEGETS_ANGER meta: type, story	STORY INSULT_BEGETS_ANGER.
new noun: JERK, person, true	JERK is a generic person.
new noun: TARGET, person, true	TARGET is a generic person.
new node: S0, state update node: S0, actor, JERK update node: S0, type, feelingstowards update node: S0, value, negative update node: S0, to, TARGET	S0: JERK has feelingstowards of negative for TARGET.
new node: G1, goal update node: G1, actor, JERK update node: G1, object, TARGET update node: G1, type, c-mood update node: G1, value, angry	G1: JERK wants TARGET to be angry.
new edge: S0, G1, motivates	S0 motivates G1.
new node: A1, act update node: A1, actor, JERK update node: A1, to, TARGET update node: A1, type, ?	A1: JERK verb? TARGET.
new edge: G1, A1, plans	G1 plans A1.
new node: S1, state update node: S1, actor, TARGET update node: S1, type, mood update node: S1, value, angry	S1: TARGET is angry.
new edge: A1, S1, intends	A1 intends S1.
new edge: S1, G1, achieves	S1 achieves G1.

Table 4.5: 'Jerk insults someone' implemented in SIL and StoryWriter.

at any point in its creation was added to Skald. This is supported by a construct known as the Lineage which stores a story in SIL and contains some number of timepoints each associated with a list of SIL commands. In addition to the list of commands, each timepoint also stores a string detailing the reason for its alterations.

Using Lineages provides much needed transparency for system designers and authors by enabling two important abilities: temporal view of stories and rollbacks. The temporal view is simply the ability to see the entire list of graph alterations in order with reasons for each change. Because many different components touch the stories in many different ways it's often impossible to tell why something appears in a story when only looking at the final graph. By looking at a temporal view, finding both bugs and their causes is greatly simplified. The second ability that Lineages provide is the ability to rollback a graph to any timepoint. By looping a sequence of rollback and generation requests, authors can investigate probabilistic bugs. These either take the form of an improperly working module that is rarely triggered or some problem with the domain (e.g., a malformed chunk of story in the library which lowers the coherence of stories that include it).

4.3.5 Changes From Minstrel

Although the functionality of Skald's story library is the same as that of the original Minstrel, Turner's thesis didn't include the complete set of stories that he used for his library (See Appendix for the contents of our domains). As a result, our story library is different, including a few stories that were described in his paper, but mostly made up of stories that we created ourselves. This is one of the most significant changes in terms of its impact on story output, although it

doesn't affect the system architecture. The majority of our story library is a set of stories that we authored aimed at providing a diverse set of raw material for generation centered around a roughly medieval theme. Although this seemed like what Minstrel called for, we have found that a much more focused library may be necessary to support the quality of stories that Turner was able to produce (see chapter 7 for more details). In addition to simply altering the domain, Skald includes SIL, storyWriter, and Lineages, all of which were not in Minstrel.

4.4 TRAMs

"MINSTREL's main contribution to the computational creativity in writing is the concept of TRAMs, which demonstrate the potential power of small changes in story schemas." - Raphael PÃlrez y PÃlrez [Pérez y Pérez and Sharples, 2004a]

As the above quote says, TRAMs are one of the best contributions of Minstrel. They are arguably the heart of Skald's creative power and are certainly one of the most used modules of the entire system. TRAM stands for Transform, Recall, Adapt Method and is the CBR component of Skald. Together with the story library, the TRAM system can be used to fill in generic or empty symbols in a story or provide analogous nouns when asked to do so by other modules.

4.4.1 Functionality

TRAMs (Transform Recall Adapt Methods) support a modified form of case based reasoning. In Minstrel, many small fragments of stories are created over the course of creating a whole story and this is done by the TRAMs. The TRAM system is called with a search query which is a partially defined story fragment and returns a match for that query. One example query is, "someone does something to

a dragon which kills it" which may return "Lancelot fights the dragon which kills it." The system includes a number of TRAMs which can be recursively applied in any order to find a match for a given query.

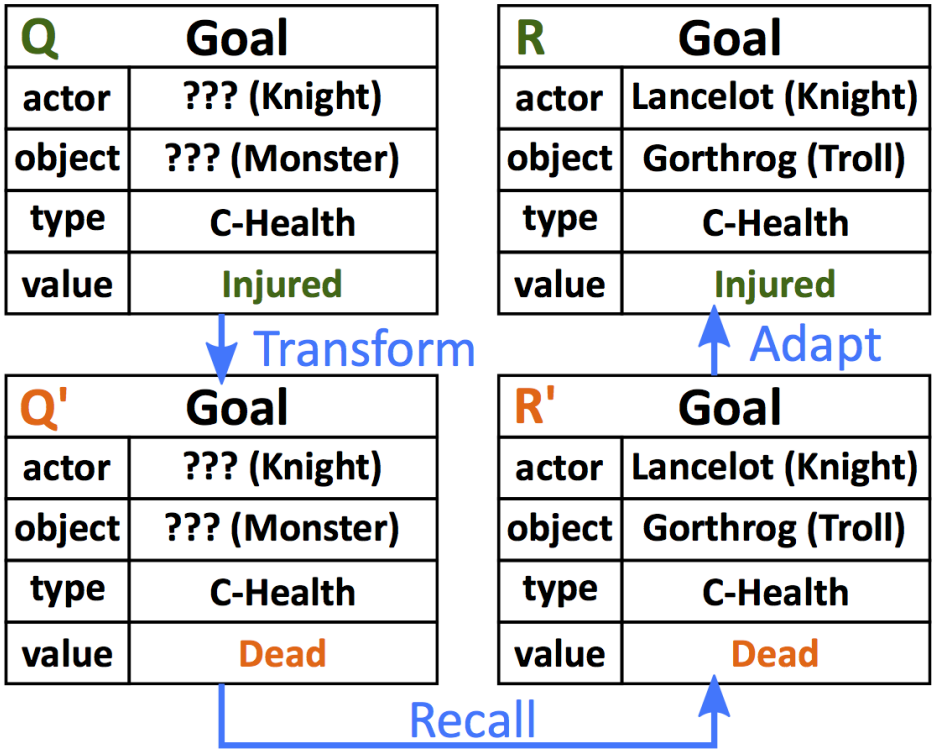

Figure 4.4: The TRAM system in action.

The first step for each query is to attempt to recall fragments out of the story library which match without any transformation. Failing this, each TRAM has a specific transformation that it applies to the current query before recursing. Effectively, the TRAM system searches through a space of queries (connected via TRAM applications) to find a linear sequence of TRAMs which lead from the original query to a query that directly matches a story fragment in the library. When a result is found, that result is adapted to work with the original query

by reversing the transformations (e.g. changing result slots to the values of the original query). The benefit of this method is that it can use precise transformations in sequence to effectively find cases that are quite different from the original query. Of course, the TRAMs are not perfect, and the more TRAMs used, the higher the risk of a result which produces an incoherent result when combined with the rest of the story. Thus the TRAM system relies on a story library which contains examples that are fairly close to the queries it attempts to answer.

In Figure 4.4, an example query Q is shown in which a knight has a goal of injuring a monster but no stories are available in the library which can be used. As a result, a TRAM converts the query to one in which the monster is killed (Q') and this new query retrieves a result R' from the case library about Lancelot planning to kill Gorthrog the troll. This result is finally adapted back to a fragment R in which Lancelot plans to injure Gorthrog, satisfying all of the original requirements (C-Health is short for for 'Change-Health').

Figure 4.5 illustrates a TRAM search tree with 3 potential TRAMs. The query passed in is the top node in which a knight fights something. The three children of the top node are all possible through different TRAM applications. The left child generalizes the action, the middle child replaces the actor type with a more general actor type, the right child completely generalizes the actor. In the diagram, the middle child has been picked, transforming the query from a knight fighting something (top row) to a person fighting something (middle of second row). At this point Minstrel might recall a peasant fighting a troll, a constable fighting a bandit, or other similar situations. When adapted back, a story about a knight fighting a troll or bandit makes sense. If the recall were to fail however (if there are no stories in the library about people fighting) there are 2 further children (the bottom row in the diagram): relaxing the constraint that there is a

person involved or releasing the constraint of a fight being in the matching story. Without the fight involved, Minstrel might match a story about a princess eating a berry, yielding a much stranger story after adaptation: a knight fighting a berry. Despite this being a possibility, TRAM selection is weighted to prefer less loss of specificity in order to avoid queries with a low likelihood of being useful such as 'a person does something to a thing'.

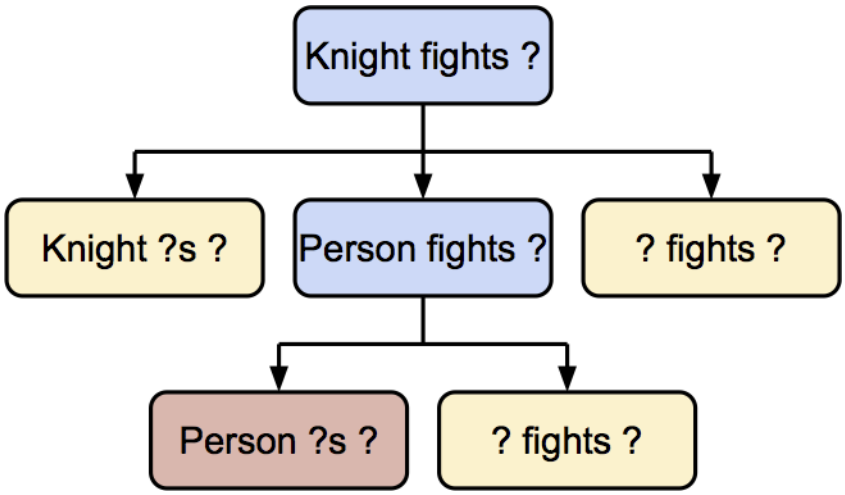

Figure 4.5: The TRAM search space.

Skald contains fifteen TRAMs and each of these can be applied in any order and often with multiple possible targets within a given query. TRAMs operate inside of a backtracking search tree, allowing for many different sets of TRAMs to be tried in order to eventually find a match for a given query. Because this search space often has a branching factor in the hundreds (each way to apply each TRAM is a branch in the search space), there are optional depth and exploration limits that can be used to keep the searches from taking too long.

4.4.2 TRAMs In-Depth

Although Minstrel's TRAM module is well documented in terms of research contributions, its implementation details are sparsely described. These details may seem trivial until one attempts to create a working TRAM system, at which point it becomes clear that writing a method to recall a frame from the story library given a query frame can be implemented in a number of ways, many of which will alter the possible output of the module. While the previous section gave a nice overview of the more theoretical aspects of how the TRAM module functions, this section describes the reconstructed algorithmic details.

Since the TRAM system's responsibility is to add details to partially specified frames, the most fundamental procedure that the TRAM system executes is called *recall*. This process is given one or more frames as a query and finds matching frames in the story library. Matches to a recall must have two properties: They must have the same structure as the original query (in terms of number of nodes, node types, edges, and edge types), and the content of the nodes must *generalize* to the original query. Generalization rules are fairly straightforward:

- All fields generalize to Empty or Unknown values
- String fields generalize to matching string fields (e.g. 'friendly' generalizes to 'friendly', <empty>, and <unknown>)
- Noun reference fields generalize to matching references as well as references further up the noun ontology (e.g. 'Lancelot the knight' generalizes to 'Lancelot the knight', some generic knight, a generic violent being, etc., <empty>, and <unknown>.

The complete ontology of Arthurian nouns used in Skald is listed in Figure 4.6.

As an example of content matching via generalization, a query containing a generic "person" would be matched by a result containing a "knight" (which is

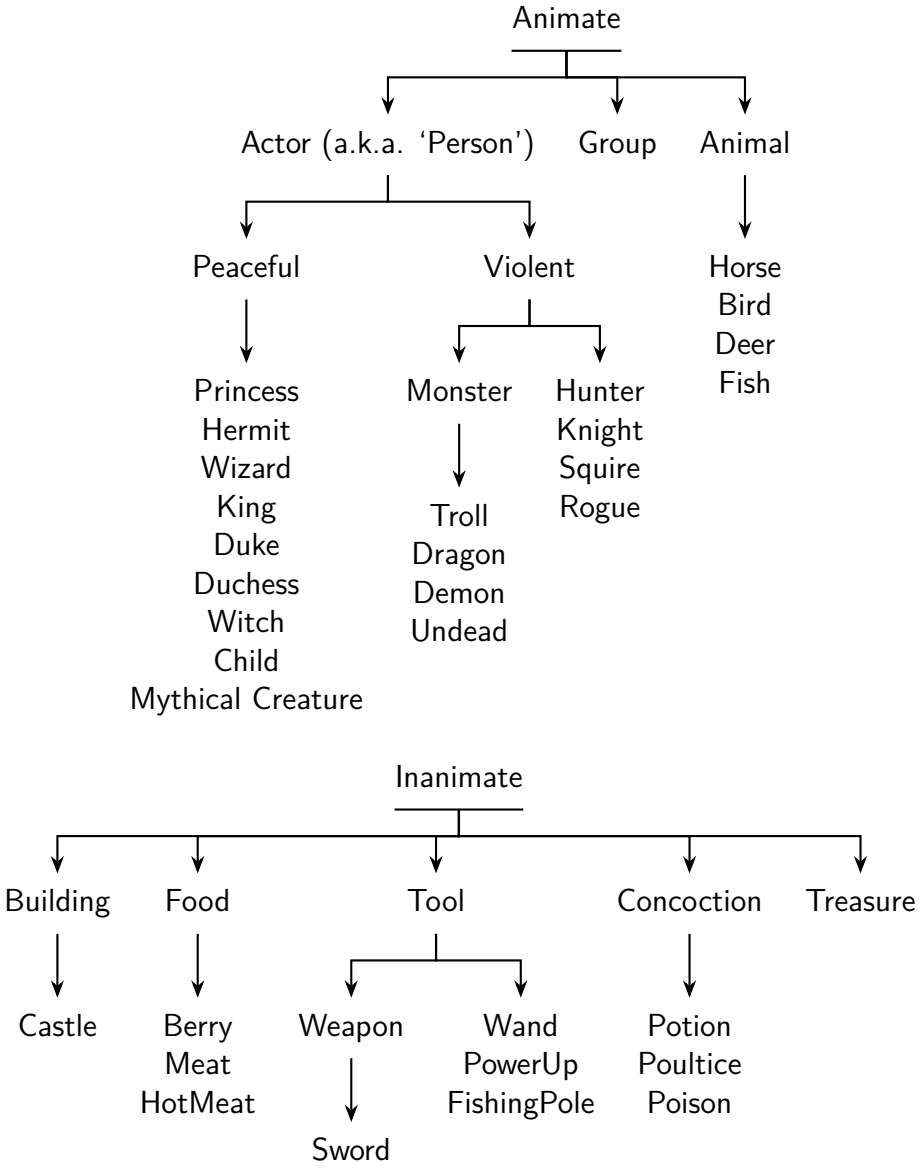

Figure 4.6: Arthurian noun ontology used in Skald

a type of person) but not one containing a "sword" (which isn't a person) or an "animate" (which is more generic than "person"). The fact that results for a *recall* have two requirements begs the question, which one should be used to search through the story library and which should be applied as a second-pass filter? Skald's strategy here is to first find structural matches to the query by extracting all subgraphs from the story library that contain nodes of the correct type with edges also of the correct type. Once these subgraphs are extracted, they are filtered to see if any of them have content which can generalize to the query. All subgraphs which passes this filter are returned as the results of the *recall*. It should be noted that we never tried to do this in the reverse order (which is incidentally how Riu does its recall) but we believe that searching for content matching and then filtering for structural matches should have little effect on the output of the TRAM system.

As was mentioned in the previous section, once the TRAM system is provided with a query, it tries to recall matches from the story library and then proceeds with a sequence of recursive transformations and recalls in order to find a match that is "close enough." An initial clarification is in order: the query includes both a subgraph to match on and a pointer to a single 'target' node in that subgraph. The nodes surrounding the target are treated as context which help make the query more specific. Only the target will be altered when a final match is found and adapted. A single loop of the TRAM application and lookup sequence consists of:

- Determing which TRAMs are applicable to the query by executing the TRAMs "test" method. For example, TRAM-GeneralizeActor tests whether the target node has an actor to determine if it is applicable to the query.

- Once the list of viable TRAMs has been established, a TRAM is "selected"

and used to transforms the original query

- The modified graph fragment is returned to the TRAM module as a recursive query if needed.

Minstrel's TRAMs operated as black boxes: they were provided with a query and returned a recalled and modified graph based on that query. Skald has been upgraded to function as a slightly-less-black box which provides both step-by-step tracking of transformations for post-hoc analysis and special exploration functions which can provide researchers with a clear picture not just of how the transformation did proceed but how it could have proceeded.

As was implied by the quotations marks in the itemized TRAM sequence above, the processes governing a TRAM's "test" and "selection" methods are more complex than their names imply. When an individual TRAM tests the query to see if it can be applied, in actuality it also generates a list of all possible inputs to its transformation step that would be valid with the given query and returns this list. A clear example of this comes from the "Generalize Actor" TRAM which finds actors which are similar to ones in the query, based on what actions those two types of actors perform in other stories, and then swaps in a different actor for that in the query. By using this TRAM on a hermit in a story, it may decide that suitable replacements for the hermit are a girl, a witch, or an advisor. When this TRAM returns an answer from its test, it will provide all three of these possibilities as returned values to be selected between. This means that at any recursion level of a TRAM call, all possible TRAMs and all possible variations of those TRAMs are able to be captured and displayed to researchers wanting to see how the system is working. This is invaluable for squashing bugs as well as ascertaining whether or not the domain has sufficient variation for an author's requirements.

Once all viable TRAMs and all viable variations are collected, one must still be selected for transformation and then, as was stated before, this transformed query is recalled against to continue the cycle. While using the system we noticed that some of the TRAMs seem to alter the query more than others and, in order to counteract the more disruptive influence of these TRAMs, included a toggleable setting to select from amongst the results either randomly by TRAM and then randomly by variation or in a weighted manner by TRAM and then randomly by variation. The results of this setting were discussed fairly extensively in [Tearse et al., 2011b] and upon further use we determined that making weighted selections yields better results (see Chapter 7 for more details).

4.4.3 Individual Trams

Now that the TRAM module has been described both at a high and low level, a brief description of each of the individual TRAMs is in order. Skald contains an unmodified copy of each of Minstrel's TRAMs. For more details on any individual TRAM, please see Turner's dissertation [Turner, 1993]. Each of these TRAMs is listed as the name of the TRAM in bold followed by its selection weight (higher means our weighting system is more likely to use it) and then a brief description of the transformation, a discussion, and an example. Although a more thorough analysis of these TRAMs is available in chapter 7, it should be noted that some TRAMs work well in environments with low amounts of domain knowledge (i.e., few stories in their library) while others are specialized for high domain knowledge environments and, for reasons discussed later, TRAMS that function well in low domain knowledge environments tend to have lower selection weights. This list is ordered by TRAM weight so the higher items in the list tend to be used before others when they can be. Another way of looking at this is that the first items

on the list are the "smartest" and least destructive while the lowest items on the list tend to be measures of last resort.

GeneralizeActor (10.0) : *Find a new type for a noun reference in the actor slot in the target node. Let's call the type of the noun that is being replaced 'T1'. Find a story containing a noun of type T1 taking an action which we'll call A. Now find a node in a different story in which action A appears and we'll call this node's actor's type 'T2'. If T1 and T2 are different, make the target node's actor have type T2.*

Generalize Actor is one of the best TRAMs for coherence since it does a good job of finding analogous actors to use as stand-ins for the one in question. The main downside of this TRAM is that in poorly populated domains, there may be no analogous actors and, even if one can be found, the limiting factor will often be some piece of more general context such as linked nodes or node types rather than the type of the actor, making this TRAM ineffective. Example:
Query: [context node] Adam the knight was hungry so [target node] Adam ? the ?

Transformed: The soldier was hungry so it ? the ?

Recalled: Bob the soldier was hungry so he ate the rations.

Result: Adam the knight was hungry so he ate the rations.

GeneralizeRole (8.0) : *Find an actor in the target node and generalize it up one level in the noun ontology.*

Like GenerelizeActor, this TRAM is quite good at yielding a new query that is similar enough to the original to provide high quality results. It is slightly less elegant than GeneralizeActor but at the same time functions better for us in less

defined domains since creating clearly defined noun ontologies is far simpler than creating stories. Thus, each of the sketchily defined domains we've created has had a robust noun ontology (and useful for this TRAM,) before any stories were created. Example:

Query: [context node] Adam the knight was hungry so [target node] Adam ate the ?

Transformed: The violent person was hungry so it ate the ?

Recalled: The bandit was hungry so it ate the poached venison.

Result: Adam the knight was hungry so he ate the poached venison.

GeneralizeObject (8.0) : *If there is an object involved in the target node (e.g. some tool or the direct object of the node), generalize the object up one level in the noun ontology.*

This is essentially the same thing as GeneralizeRole but applied to a different slot. As is demonstrated in the example, this and GeneralizeRole are excellent for dealing with situations where the current story is slightly too detailed for the story library. Example:

Query: [context node] Adam the knight was hungry so [target node] Adam ? the venison

Transformed: Adam the knight was hungry so he ? the meat

Recalled: Adam the knight was hungry so he ate the meat

Result: Adam the knight was hungry so he ate the venison

LimitedRecall (5.0) : *Remove all nodes that are not adjacent to the target node.*

This TRAM is essentially a less destructive version of DesperateRecall (which

is banished to the bottom of the list due to its destructive tendency). In Skald we turned off this TRAM since one of our time-saving measures is to pre-truncate queries before they're handed to the TRAM system. Example:

Query: [context node] Adam the knight was hungry so

[context node] Adam wanted to not be hungry so

[target node] Adam the knight ? the venison

Transformed: Adam the knight wanted to not be hungry so he ? the venison

Recalled: Adam the knight wanted to not be hungry so he ate the venison

Result: Adam the knight was hungry so he wanted to not be hungry so he ate the venison

IgnoreMotivation (5.0) : *If the target of the query is a goal node and there's a motivating state, remove it.*

This TRAM is essentially a more limited version of LimitedRecall. The underlying logic here is that when a query is provided in which there already exists a motivating state and an action that's the result of an undefined goal, oftentimes the action is more important for determining what the goal should be than the motivating state (i.e., the action one takes is more tightly coupled with one's goal than the reason for said goal). Skald's templates generally include ambiguous actions and concretely defined goals so this TRAM is rarely used. Example:

Query: [context node] Adam was angry so

[target node] he wanted to ? so

[context node] he fought with Bob

Transformed: Adam wanted to ? so he fought with Bob

Recalled: Adam wanted to kill Bob so he fought with Bob

Result: Adam was angry so he wanted to kill Bob so he fought with Bob.

IgnoreSupergoal (5.0) : *If the target node is a goal and it's a subgoal of another goal (its supergoal), remove the supergoal.*

Like IgnoreMotivation, IgnoreSupergoal is based on the concept that the query can be simplified by focusing on the current task at hand. Also like IgnoreMotivation, this TRAM is relatively specialized for a situation that has not often been part of the domains authored for Skald. This TRAM would be helpful for domains involving more mental processes and goal setting than Skald's action-driven stories. Example:

Query: [context node (supergoal)] Adam wanted Bob to attack Adam so
[target node (goal)] Adam wanted Bob to ? so
[context node] Adam insulted Bob.

Transformed: Adam wanted Bob to ? so Adam insulted Bob.

Recalled: Adam wanted Bob to be angry at Adam so Adam insulted Bob.

Result: Adam wanted Bob to attack Adam so Adam wanted Bob to be angry at Adam so Adam insulted Bob.

IntentionSwitch (5.0) : *If the target node intends to cause some state change, swap the link on that state change to an accidents link.*

This TRAM is predicated on the idea that actions have consequences that tend to be the same whether they are intentional or accidental. If a search for a story with an action that intentionally results in a specific state change comes up empty, finding a story that unintentionally results in that state change will likely yield a good enough analogy for use. Like the rest of the high weight TRAMs, this one doesn't really work in domains with few stories simply because the likelihood of finding accidental state changes matching the one specified in the query is very

low. Example:

Query: [target node] Adam ? the ? in order to [context node] break the ?.

Transformed: Adam ? the ? and accidentally broke the ?.

Recalled: Adam threw the hand mirror and accidentally broke it.

Result: Adam threw the hand mirror in order to break it.

AccidentSwitch (5.0) : *If the target node accidentally causes some state change, swap the link to make it intentional.*

This TRAM is the exact inverse of IntentionSwitch, see above for details.

RecallActs (5.0) : *If the target is an act node, remove all nodes that aren't joined with "plans" or "intends" links.*

This is one of the more targeted "remove nodes from the query" TRAMs. It operates on the assumption that actions are informed by their underlying goal and outcome more than anything else and thus removing other information improves recall chances without sacrificing much contextual information. This TRAM is fairly specialized but sees a fair amount of use since actions often have other adjacent contextual nodes which limit recall but are less significant such as a node describing a precondition. Although this TRAM is crucial since 'precondition', 'accidents', 'thwarts' etc. edges almost always make a search query too specialized to yield results, it also demonstrates a weakness in Skald since generalizing away these contextual cues can result in pretty serious non-sequiturs (e.g., owning a sword becoming a precondition to poison the evil queen). Example:

Query: [context node (this plans target)]Adam wanted the troll to be dead so [context node (precondition of target)] Adam owns a sword which allowed [target node] Adam ? the troll with the sword so

[context node (target intends this)] The troll was dead.

Transformed: Adam wanted the troll to be dead so Adam ? the troll with the sword and killed it.

Recalled: Adam wanted the troll to be dead so Adam fought the troll with the sword and killed it.

Result: Adam wanted the troll to be dead so Adam fought the troll with the sword and killed it.

RemovePreconditions (5.0) : *If the target is an act node, remove all preconditions.*

This is a version of RecallActs that is slightly more limited in scope since it will leave edges such as accidents, thwarts, etc. See above for details.

SimilarOutcomes (3.0) : *If the target is an act and there's a linked state node, generalize the state node's type and/or value.*

Essentially this TRAM operates on the principle that an action causing a state change in some object or person can be informed by other types of state changes in that object or person. For example, if I want to find an action that causes Adam to be angry, I might be able to use the same action that causes Adam to be sad... or maybe dead (to generalize both type and value). Example:

Query: [target node] Adam did ? and caused [context node] Bob to be angry.

Transformed: Adam did ? and caused Bob to be ?.

Recalled: Adam attacked Bob and caused Bob to be injured.

Result: Adam attacked Bob and caused Bob to be angry.

SimilarStates (2.0) : *If the target is a state, generalize its type and value.*

This is a more generic version of SimilarOutcomes. It has the same effect of generalizing a state node's type and/or value but requires the target node to be a state node and does not require an attached act node. See above for details.

GeneralizeConstraint (2.0) : *Take a single defined slot (has a concrete or generic value) in the target node and generalize it to 'unknown'.*

GeneralizeConstraint is similar to DesperateRecall (next in the list) in that it removes context which could otherwise be important but it does it at a much smaller scale and only alters the target node. Because this doesn't change the complexity of the story element, its matches are likely to fit into the story being built. It does occasionally cause absurd elements to crop up in stories due to its less-than-elegant removal strategy, but these tend to be local oddities rather than major coherence problems. This TRAM is often the only useful one when Skald is working with a domains with few stories in its library, but its utility sharply drops off in comparison to other TRAMs as the conceptual coverage of the domain increases. Example:

Query: [context node] Adam was hungry so [target node] Adam ? the ?

Transformed: ? was hungry so it ? the ? (both Adams get generalized at the same time)

Recalled: The troll was hungry so it ate the cow.

Result: Adam was hungry so he ate the cow.

DesperateRecall (0.5) : *Remove all nodes from the query that are not the target.*

This is both highly useful and highly disruptive since it essentially transforms the story element from a complex one (which relies upon the context of the nodes

around it for some of its meaning) into a simple element. This simple element is often far easier to match against because of its lack of contextual elements — but practice has proven that integrating new details found only after removing this context is often a source of story incoherence. Example:

Query: [context node] Adam was hungry so [target node] he ? the ?

Transformed: Adam ? the ?

Recalled: Adam destroyed the village

Result: Adam was hungry so he destroyed the village

4.4.4 Changes

The original Minstrel applied TRAMs randomly, but Skald supports multiple techniques for directing its TRAM searches. Included are deterministic and fully random search methods for testing purposes as well as a weighted random method for optimizing TRAM results both in terms of speed and quality. The weights associated with each TRAM (assigned by hand) correspond roughly to how much it modifies the original query. TRAMs are then selected randomly with probability proportional to their weights, so that TRAMs which have more drastic effects are chosen less frequently. By searching more intensely in the part of the search space closest to the original query (in terms of the distance from the original query rather than the number of TRAMs that have been applied), this technique provides quick results which generally make sense even when multiple TRAMs are required to get a result. In a previous analysis of Skald's creativity [Tearse et al., 2011b] we discuss the tradeoff being negotiated here between increased sanity and reduced creativity. Because a fully random search method is implemented within Skald, we were able to compare Minstrel's original design with our own and see how this subsystem affected the overall system. For more

information about these tradeoffs and the difference in outputs between random and weighted TRAM searches, see chapter 7.

4.5 Author-Level Planning

The author-level planning system pursues author-level goals (ALGs) by retrieving and executing author-level plans (ALPs) that serve specific functions in generating a story. Goals exist in a priority queue which re-enqueues failed goals at a lower priority. High level goals consider the story as a whole, and represent tasks such as deciding on a theme for the story or checking the story for opportunities to insert foreshadowing. At the same time, lower-level goals concern things such as filling out the details of a particular state or act within the story, or checking that a particular story node is consistent and properly motivated. Some of the ALPs encode micro-theories about storytelling (theories of consistent motivation, for example) that help Minstrel produce consistent output. Other ALPs rely on the story library to act as a model of a well-formed story, using the TRAM system to fill in pieces of the story under construction with appropriate material.

Skald does not simply generate free-form story structures. Instead, it relies on templates to structure its stories. These story templates contain rough specifications for important parts of the plot. Skald generates stories by selecting an appropriate template and then filling in the details of that template, adding extra scenes to the story as necessary along the way.

Once a goal is selected and a set of plans is found for it, plans are tried one by one until one succeeds. If all available plans fail, the system re-enqueues the current goal with half of its original priority and drops it entirely if this would put it below a minimum priority threshold. This convention allows goals to interact: if one goal cannot be solved initially, other goals are attempted in the hopes that

they will alter the story configuration and make the initial goal solvable. Once the goal queue is empty, story generation is done.

4.5.1 List of ALPs

Since ALPs are both Reflective and Constructive processes in Skald, it is fairly simple to divide the ALPs into a few different categories based on those distinctions. These categories are: Reflective Microtheory, Constructive Microtheory, Reflective Coherence Checker, Constructive Generalist, and General Machinery. These categories are explained in detail below. Generally speaking, the microtheory ALPs work together to try to weave in higher level concepts such as tragedy or suspense, the coherence checkers use their own limited understanding of how to find "bad" pieces of a story and repair them, the Constructive Generalist ALPs each know how to instantiate a chunk of story in a different way, and the General Machinery ALPs handle setup, teardown, and other basic functionality. What follows is a list of all of the ALPs along with a classification and description of functionality, and in some cases a brief discussion. As with the list of TRAMs, further details can almost always be found in Turner's dissertation (which covers many more details) or the source code of Skald itself. We only wrote one new ALP for Skald; ReassignDeadActors did not appear in the original Minstrel.

General Machinery:

- TellStory
- SkipInstantiation
- FindStoryTheme
- DontInstantiate

The general machinery ALPs help to control the flow of the ALP system and provide some default failure behaviors. TellStory is always the first ALP to be

called and enqueues FindStoryTheme (which finds a suitable GST to be used to construct the story skeleton from) as well as a series of Instantiation ALPs. DontInstantiate and SkipInstantiation handle general purpose failure cases for instantiate tasks which have timed out or returned an error. They abandon the author-level goal to fill in a particular node or leave it for later.

Reflective Microtheories:

- CheckForCharacteristic
- CheckSceneForSuspense
- CheckSceneForTragedy

This group of ALPs is part of the Reflective machinery in Skald. Their jobs are to look over the current story and enqueue constructive microtheory goals when a suitable pattern emerges. The patterns differ based on the ALP but one example is CheckSceneForTragedy which searches for events in which one character kills another. Although there can be many ALPs that resolve the goals that reflective microtheories enqueue (because there are multiple ways that something like 'tragedy' or 'suspense' can be added to the story), /colorredtragedy only has one (described below), which adds /colorblacka romantic connection between the two characters somewhere previous to what will soon be the tragic slaying of a loved one.

Constructive Microtheories:

- AddTragedyViaLovedOne
- AddSuspenseViaCharacterEmotion
- AddCharacterization
- AddSuspenseViaFailedEscape

As was mentioned in the Reflective Microtheories section, when a reflective theory finds a portion of the in-progress story that it would like to change, it enqueues a goal which one of the constructive microtheories must accomplish. In theory there could be many constructive theories to go with each reflective theory. In our tests of Skald, we turned off these ALP based micro-theories since they didn't interact with our domain particularly well. This is largely due to the small size of our domains which lead to extreme overuse or underuse of the theories (depending on whether their preconditions were satisfied in the story templates). More discussion on these microtheories can be found in chapter 7.

Reflective Coherence Checkers:

- CheckActConsistency
- CheckGoalConsistency
- MakeConsistentColocation
- ReassignDeadActors (New)
- CheckAffects
- CheckStateConsistency
- MakeConsistentMotivatingState
- CheckAffectsOthers
- MakeConsistentCause
- MakeConsistentSupergoal

In a sense, these ALPs act like inverse stories as far as Skald is concerned. Stories inform the system what pieces go together and what patterns they can form while Reflective Coherence Checkers tell the system what pieces don't go together and what patterns aren't allowed. In an ideal world none of these would be required (stories would just be generated properly all the time) but, from an

engineering perspective, creating little routines to find and fix the most egregious errors that have a habit of popping up is much simpler and more direct than creating a handful of stories hoping they'll help guide generation along more coherent paths. One clear example of this concept and this entire class of ALPs as a whole is ReassignDeadActors. Before we created it, Skald would fairly consistently kill actors off in various ways and then in the next scene have them doing things. This is perfectly reasonable since Skald doesn't know that killing things causes them to deanimate and no other state change bars future action. To prevent our science fiction and fairy tales from suddenly ending up awash in zombies we coded up the ReassignDeadActors ALP to find places where a dead entity was doing something and reassign that action to some other actor (or make one up if need be).

Constructive Generalisits:

- InstantiateEvidenceAuthoratativeSource
- InstantiateEvidenceMotivatingAct
- InstantiateEvidenceMotivatingState
- InstantiateSupersedingBelief
- GeneralInstantiate
- InstantiateAntifavor
- InstantiateFavor
- InstantiateThwartingState
- InstantiateBelief
- InstantiateDeception
- InstantiateRevenge
- InstantiateViolentReprisal

The final class of ALPs are responsible for generating the story rather than fixing it or implementing special microtheories. The most used ALP in all of Skald

is GeneralInstantiate which is used to actually fill in a node that isn't yet final. The rest of the generalists in this category encode small pieces of knowledge about how actions and events can be pieced together in a story.

4.5.2 Changes

One change to the ALP system is that Skald uses "Generic Story Templates" (GSTs) instead of what Turner called "Planning Advice Themes" (PATs). These two constructs share the same structure but differ thematically. Turner asserted that the stories Minstrel would generate should all revolve around "planning advice": they'd all have some moral that could be taken as a kind of advice encoded in a story. He believed that focusing on these types of stories was important, and so he called his story templates "Planning Advice Themes" and populated his template library exclusively with this type of content. The template used to generate "The Mistaken Knight," for example, consists of a character rashly doing something which they later regret, but which they are unable to take back. We have retained some stories that fall into this category, but we want to apply the Minstrel architecture more generally, and so in our system templates are just called "Generic Story Templates" and we have no ideological position on how they should be constructed.

Besides this change in the types of templates used, Skald also has a number of Minstrel's author-level plans disabled. After implementing the original set of ALPs based on Turner's descriptions, we found that many of the microtheory-based ALPs were prone to disrupting the story by adding irrelevant or incoherent details to the story. These microtheories worked well when they were carefully supplementing Minstrel's single long story but they don't work well in the much shorter stories that Skald produces. The final difference, was that we added an

ALP because of an issue we noticed: actors that were killed in the story would continue to initiate actions later on. This issue likely never came up for Turner but we felt the inclusion in Skald was appropriate. All of the ALPs included in Skald can be enabled or disabled, allowing comparisons between the original set and our own.

4.6 Bard

The Bard is the module that users interact with and which controls the flow of Skald as a whole. This subsection is only important from an engineering point of view since the Bard controls the importing of stories, TRAMs, and ALPs, the setting of flags, and the flow of information from one module to another. When a user requests that a new story be created, that request is made through the Bard which then triggers the ALP and TRAM cycle talked about at the beginning of this chapter. The Bard also contains a number of APIs that are intended to help Skald interact with other programs and interfaces. One such interface is included in Skald which provides a nice web-based interaction for users who would like to produce and view stories.

4.7 Boredom

Although not technically a subsystem on its own, Minstrel has a notion of boredom that is embedded within the TRAM system but acts on its own. Additionally, the concept of boredom is crucial to the longevity of the TRAM system. To add variability to the system, Minstrel is programmed to get bored with one solution to a problem and, as a result, to search for novel solutions. This is implemented in Skald as a table of query/solution signatures coupled with a boredom value.

Every time a query is given to the TRAM system and a solution returned, the boredom value of that signature is incremented. High boredom query/solutions won't be used, so as the boredom value for a pair rises, other solutions must be found for given queries. Without this method of enforcing variation, Minstrel would often generate duplicate stories, though the random nature of the TRAM searches contributes to variation even without boredom.

4.7.1 Changes

Skald's boredom system is different from Minstrel's boredom system. Originally, Minstrel only incremented boredom, and when a solution had been used twice, it would never be used again. While useful for forcing interesting results, this had the side effect of quickly exhausting the story library, resulting in increasingly incoherent stories after only a few had been generated. Our modified system doesn't produce quite as interesting results as quickly, but instead distributes variation across dozens or hundreds of stories.

Skald's boredom system fractionally decrements the values of all signatures in the boredom table with each call to the TRAM system, gradually moving them below the boredom threshold. Functionally this means that signatures refresh over time, allowing them to be reused in subsequent stories. Using this system, Skald establishes a cyclical pattern of boredom and the effect, once the pattern has been established, is similar to generating each new story using a random subset of the story library, always avoiding recent results. This randomization produces a sustainable loop that distributes the creative potential of the system evenly over a large number of stories.

4.8 Instantiating a Story Node

Figure 4.7 below shows the sequence of communication between the ALPs, the story library, and the TRAM system during the execution of a "GeneralInstantiate" author-level plan. When the ALP system selects and begins executing this plan, it performs the following procedure:

- [Step 1]: Move a copy of Template 1 into working memory. This copy will be called the 'current story'.
- [Step 2]: GeneralInstantiate will target a particular node in the current story and will mark that node and all immediately adjacent nodes as the 'current scene'.
- [Step 3]: The ALP system checks to see if the current scene is incomplete and if so, sends it to the TRAM system as a query.
- [Step 4]: The TRAM system begins imaginative recall to fill in the target node. The TRAM system then recursively attempts to transform the query and find a match (Step 'T').
- [Step 5]: Once a match in the story library has been found for the transformed query, it is adapted back (step 'A') and used to fill in the original query and become the TRAM result.
- [Step 6]: The TRAM result is returned to the ALP system where it is used to update the target node in the current scene.
- [Next ALP]: The ALP system then proceeds on to its next task which will generally be checking the newly instantiated node for defects or instantiating a different target node.

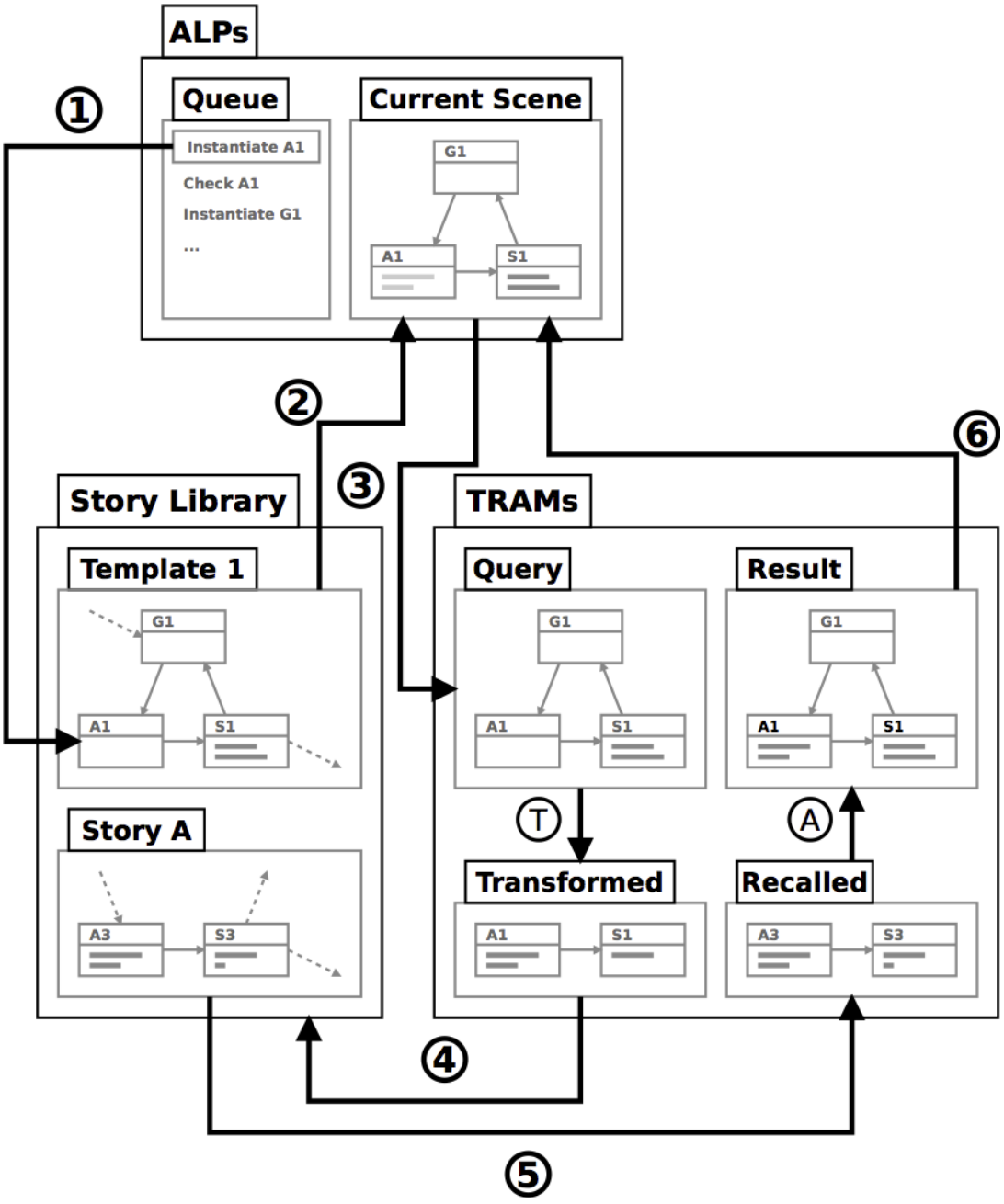

Figure 4.7: Interactions between the ALP system, the TRAM system, and the story library during the instantiation of a single frame.

4.9 Analysis of an Example Story

To demonstrate the differences between what we can produce and Turner's examples qualitatively, we include an example story here. Note that Minstrel originally used a natural language generation framework called Rhapsody to turn its graph structures into English sentences. Although we have reconstructed the core Minstrel components, our natural language generation processes can't compare with Rhapsody so this story was translated from a graph structure into English by hand with the idea of mimicking the style of Turner's original output.

> **Smaug, Mason, and Lancelot**
> Once upon a time there was a dragon named Smaug. Smaug wanted a knight named Lancelot to be dead. One day, Smaug found a king named Mason and fed him poisoned lamb, causing Mason to become poisonous. Later, Smaug presented Lancelot with a present: Mason. As planned, Lancelot was hungry and ate Mason. Later on because of this, Lancelot died. Smaug felt good.

This story was picked both to show the general subject and depth of our stories and to demonstrate some particular issues that we have run into. Chapter 7 provides a more detailed analysis of the quality and creativity of the outputs of Skald. As will be discussed there, we do frequently run into stories that have quirks similar to those this one has.

It is immediately apparent that this story is much shorter than Turner's example "The Mistaken Knight." The main reason for this is that we had disabled some of Turner's ALPs because of the frequency with which they produced bogus content. In "The Mistaken Knight," for example, an ALP which tries to add justifications for what it considers inconsistent events added the entire scene where Lancelot and Andrea met and fell in love, starting just from a node representing that Lancelot loved Andrea. When we turn this ALP on in our system, it adds such scenes (in this example it might add a scene explaining why Smaug wanted

Lancelot to be dead) but they are usually nonsensical. This is an example of how the story library needs to be tuned to match both the template(s) used and the specifics of the recall system: by adding stories to our library containing reasons for dragons wanting to kill knights, we would be able to use the aforementioned ALP with this template without trouble. However, this wouldn't be of much use when using another template which didn't involve a similar situation, so our approach was to turn off the offending ALPs rather than expand the story library to support each one.

This story further illustrates the brittleness of Skald: it doesn't make sense that Lancelot would eat a king. This story is based on a template where someone poisons a piece of food in order to kill their enemy. In this version, the main character is a dragon, and its enemy is a knight. So far this is an interesting instantiation of the template, but then the poisoned food becomes a poisoned king, which does not make much sense. Of course, Skald does not have the common-sense knowledge to realize its mistake: it can only assume that the transformations and adaptations applied by the TRAMs are small enough to keep things sensible. In this case, Skald found a story in its library about a dragon eating a king. Because of this, it assumed that a king was something that could be eaten (by anyone) and thus the king is used as the poisoned food in this story.

This shows how Skald's story library must be well-matched to its story templates and the TRAMs even when special ALP-based micro-theories aren't involved. If the library had contained a scene where a knight ate something, then when searching for a food item to substitute, it would likely have found that scene instead of the scene where the dragon eats a king, since the original query in this case is "Lancelot eats something," and Lancelot is a knight. Of course, finding that scene first depends on the set of TRAMs used: here a TRAM which gen-

eralizes actor types allows "Lancelot" to become "a knight" in fewer steps than it takes for "Lancelot" to become "any actor" (which would be able to match "Smaug the dragon" in "Smaug the dragon eats a king"). At the same time, the TRAM weights that we introduced in Skald allow the generalize actor TRAM to "cost less" than the generalize constraint TRAM, which would turn "Lancelot eats something" directly into "any actor eats something." So in order to avoid this kind of incoherent situation, a balanced approach considering the story library, the story templates, and the TRAM set is required. This kind of issue is what makes Minstrel (and the current version of Skald) a brittle system.

4.10 Rational Reconstruction

Our work on Minstrel Remixed was a rational reconstruction project: we reconstructed the original Minstrel system as a means of understanding it. The differences between Minstrel, Minstrel Remixed (our reconstruction), and Skald (which moves further away from the original) are shown in Table 4.7 below. Many of the changes described in this chapter are shown in Table 4.7, as well as some other details that vary between the systems. Whereas Minstrel Remixed is a very close reconstruction of Minstrel, Skald includes outright modifications that we have been working on which help make it more robust and which make it better at generating story fragments for interactive contexts. Note that for the most significant changes in Minstrel Remixed (the TRAM selection algorithm and the boredom algorithm), our code can be recompiled to mimic the original Minstrel for the purposes of comparing between the two implementation choices.

Feature	Minstrel	Minstrel Remixed	Skald
Architecture	Monolithic	Monolithic	Modular
Language	LISP	Scala 2.8	Scala 2.9
Source Code	No	Yes	Yes
Library Lang.	?	XML	Formulaic English
Interfaces	Command Line (CLI)	Web, CLI	Web, CLI, API
Count(StoryLibrary)	?	14	14
Count(Templates)	?	6	5
Count(Active ALPs)	33*	27	25
Count(TRAMs)	25*	13	13
Are Nouns Global?	?	Yes	Yes
TRAM Selection	Random	Weighted	Weighted
Boredom Mechanic	Static	Dynamic	Dynamic

* Note that Turner did not use all of the ALPs or TRAMs that he described in his dissertation as some were added for experimentation. We do not know exactly which ones were active to create Minstrel's primary story.

Table 4.7: Comparing features between Minstrel, Minstrel Remixed, and Skald.

4.10.1 Unknowns

Besides trying to alter Minstrel to serve our purposes, we also had to make some design decisions without knowing what Turner did. Turner's 900+ page dissertation includes many specifics, including example traces, but even with these references some things remained unclear. For example, when we started creating story graphs to be stored in the library (and generating story graphs as output) we weren't sure what scope the nouns should have. Should the Knight Lancelot be treated as the same entity even when he appears in different stories? Or should each story get its own instance? This choice has some subtle effects on how stories get built, especially when it influences graph matching. Turner never made a clear statement about this point in his dissertation and the examples that he gave did not resolve the issue. In this case, we tried both but decided that globally scoped nouns improved story quality. Perhaps that decision would be made differently in a different domain, but the domain that Turner used has a strong and consistent set of characters across many stories. Details like this likely seem obvious to the original author but upon reconstruction are revealed as important and potentially confusing choices.

Looking at the source code for Minstrel might have answered many of our questions. That being said, working from the original source code might also have made it difficult to discover authorial biases. As an example, had we used Turner's original story library, we might never have discovered how sensitive the system was to changes in the library. At one point we did contact Turner and learned that the source code for Minstrel was available, and that the tapes it was stored on could be shipped to us. Just like books, however, bits can decay and become unusable over time (a problem which digital arts historians have already started to grapple with [Rothenberg, 1995, Montfort and Wardrip-Fruin, 2004]).

In this case, technological obsolescence at both the hardware and software levels stood in our way: to read Minstrel's source code, we would need to procure a tape reader along with hardware and software environments that could run the variant of Lisp that it was written in. In the end, we did not look at the original source code, both because of the difficulties involved and in order to avoid biasing our reconstruction.

4.10.2 Reflection on the Reconstruction

Ultimately, reconstructing Minstrel proved quite difficult. Missing details forced us to experiment with several systems, and some things, like the full contents of the original story library, we were simply unable to reconstruct. As much as some of these things are opportunities to learn more about how the implementation of imaginative recall influences its performance, in some cases, having more information would have been helpful. Simply saying that researchers should describe their systems more thoroughly or that they should spend more time making sure that their code and data is preserved isn't productive, however; the pace of research is often dictated by outside terms, and even the most thorough author will miss things due to how close they are to the work. As the number and complexity of influential systems grows, spending the effort to look deeper into these systems to investigate these unknown design choices will become more important. It is unclear what the best solutions is, but creating more open-sourced projects and documentation describing the rationale behind important decisions would certainly help future rational reconstruction efforts.

4.10.3 Improving on Minstrel

Minstrel was constructed both to test and showcase the idea of imaginative recall. To that end it was a success but Turner specifically did not create Minstrel as a general-purpose story generator (see the quote on pp. 195-6 of [Wardrip-Fruin, 2009]). One of the goals of our work is to discover alterations that could significantly improve its robustness and thus its usefulness as a story generator within a larger system rather than as a demonstration of a specific model of creativity.

One such alteration is the boredom mechanic: Turner's original mechanism strictly limited how many stories could be generated from a given library by rapidly invalidating all stories in the library (Turner gives an example of having his system run out of good material after about 5 stories [Turner, 1993]). This is fine for demonstrating that imaginative recall works, but we modified this boredom mechanism to "use up" the story library more slowly (see [Tearse et al., 2012] for more details).

We also altered the TRAM selection algorithm to deal with the issue of library reliance. Although the original Minstrel produced interesting output, we found that without a library tailored to match both its templates and its transformation methods, it would often produce nonsense. To show that this reliance on a strong library could be ameliorated, we ran an experiment comparing our modified version (using weighted random TRAM searches and our updated boredom algorithm) to a version of our code without those changes. Using each version, we generated 75 stories from a single template, recording the outcome of each call to the TRAM system. Each case resulted in about 500 total TRAM calls, which was enough to compare them with some accuracy.

Compared to the strict reconstruction, our modified version reduced the failure

rate of tram searches from almost 19% to only 3.5%, reduced the average number of TRAMs tried per search from 58 to 16, and reduced the average number of TRAMs used when no direct match was found from 2.4 to 1.4 (full details of this study are in [Tearse et al., 2012]). The data also spoke to tuning of our library compared to Turner's: he reported in [Turner, 1993] that 88% of TRAM searches in his original system found a result from the story library without using any transformations (i.e., they matched some part of the library directly). Using his boredom algorithm with our library, we found that only 59% of our searches matched directly, and using our modified boredom 72% of our searches matched. Even with our modifications to boredom, this corresponds to more than a 2-fold increase (12% vs. 28%) in the use of transformations for retrieving cases: Turner's story library matched his templates (and the nodes added by his ALPs) much more closely than ours does.

Our story library is clearly a significant factor in the difference in quality between Minstrel's original stories and the ones we can generate, but this very fact reveals properties of the imaginative recall algorithm. Rather than tailoring our library to the templates that we use in order to increase direct recall, we have focused on changing the system to make better use of the story library that we have, since this latter approach promises improvements which would be useful to other people using the system.

The changes to both the boredom system and the TRAM selection mechanism are motivated by our plans to work with Skald as a component of a larger system. However, by experimenting with both our modified version and a version that mimics Turner's more closely, we can take design choices that Turner made and subject them to experimental analysis. These alterations are opportunities to understand more about how Turner's theories of imaginative recall can be put

into practice.

Chapter 5

Development of Problem Planets

The entire previous chapter was dedicated to detailing the workings of Skald. Although the engineering of such a system is itself a major contribution and significant amount of work, exploring some of the capabilities of Skald was a natural follow-up — if only to thoroughly test out the newly-created system. This chapter details a small interactive storytelling game called Conspiracy Forever which built on an early version of Skald as well as a full-blown system called Problem Planets which was built using the 1.0 version of Skald. These explorations are important because they help to examine the boundaries of what Skald can do and what it's like to interact with the system in various different configurations.

Problem Planets, and to a lesser extent, Conspiracy Forever added new interfaces, engineering methods, domains, and even modes of interaction to Skald's basic story generation. During the construction of Conspiracy Forever and at the outset of the Problem Planets project, we were under the impression that Skald just needed the right modules and internal plumbing to be a successful system. We knew that Skald's weaknesses lay in handling context-sensitive story elements and thus one of the main themes for our development of Problem Planets was finding ways to help Skald maintain and utilize context. Many of the improve-

ments we added or attempted revolved around the idea of simplifying stories and story element representations (to make context easier to handle), offloading context storage and injection into other systems, and guiding users along pathways that require no context to be coherent.

While our efforts yielded many interesting insights in ways to handle context, the most impactful ones arose from a growing sense that even a perfectly tuned version of Skald with some new context-handling modules would be missing something that was required for its success. When we finally came to the conclusion that we needed to revisit our basic assumptions about storytelling systems, we discovered that there were in fact two missing pieces that we had overlooked: Audience and Domain. The bulk of the lessons provided by the project were fueled by either the careful engineering of either its domain or user interactions to reduce complexity, improve coherence, and ultimately better handle the tricky problem of working with context in stories.

While exploring ways to change with the complexity and coherence of our stories, we came to realize that as far as our systems were concerned, the old success metric of "can it produce an interesting story" was lacking a crucial detail: our systems were interactive and thus there was far more involved in a "successful" play through than having generated a pleasing story at the end. We already had a notion that the human reader is an important piece of the storytelling system and that we want to provide them with a sense of agency by understanding their intent and reacting to it but we refined these concepts into two far more actionable principles:

- Humans are more than willing to fill in the gaps of a story so long as the details provided are coherent.
- It is far easier for the MS3 to proactively guide players than reactively

attempt to understand their actions.

These two lessons can not be overstated. The former allows us to dramatically reduce the amount of detail presented in stories (thus allowing for longer and more coherent stories) and the latter teaches that a bit of cognitive framing in the right place can make a user's actions predictable and thus easier to plan for.

At the same time as we were learning how to engineer our experiences with our users, we were also learning about how to engineer our domain, primarily by performing a complete refactor of our original domain for Problem Planets. By intentionally re-engineering our domain we discovered that a reduction in the number of symbols in the domain (when done intentionally and with care) can lead better and longer stories, less complex story representation, and a higher level of domain knowledge about each story element. To complement this improvement, we also learned that providing tools for quickly authoring stories (for the story library) in a uniform manner and with low friction for authors is crucial for the production of a tuned and healthy domain.

Examples, variations, and further details about the themes of audience and domain are detailed in the rest of the chapter and lead to many of the successes of Problem Planets. Prior to discussing this complex system however, it is useful to describe its much less developed precursor since the (mostly accidental) engineering of audience interaction and domain played a large role in its successes as well.

5.1 Conspiracy Forever

Conspiracy Forever is a prototype video game that was designed and implemented using only the TRAM system from an early version of Skald. Gameplay

involved a dozen or so players using their web-browsers to explore a 90's hacker drama. Each player was an individual conspiracy theorist with a home server where they were instructed to amass information. In order to do so, the players could connect to other servers (which were either randomly seeded or the 'home' servers of the other players) and download from it, upload to it, or manipulate the files found therein. Each 'file' was a one of 9 handwritten story fragments (each was a single GAS trio) that was then transformed using one of Skald's TRAMs to obfuscate one of the details from the original fragment. The goal of the game was for players to piece together a story from the fragments that they thought best fit together — and while players did a fair amount of this, they also enjoyed piecing together the most outrageous fragments and swapping them under increasingly evocative names. This worked well because the story fragments were extremely short and self contained. This prototype was developed only far enough to run a half dozen group play sessions, after which it was abandoned. By that time it had succeeded in demonstrating a different way to use story generation in gameplay: Generate many partial story pieces and prompt humans to search for narratives among them. Figure 5.1 displays Conspiracy Forever's styling through an image from the start of the experience and Figure 5.2 shows how one initial story fragment was transformed into two variants which were given to players.

5.1.1 Findings

Conspiracy Forever (CF) was an interesting exploration performed prior to the completion of Skald and, as such, many of the features of the fully running system were not available. Although this handicap meant that we could not use it to test out a number of subsystems, that very fact enabled the most striking finding provided by CF (which hints at what its successor would eventually more

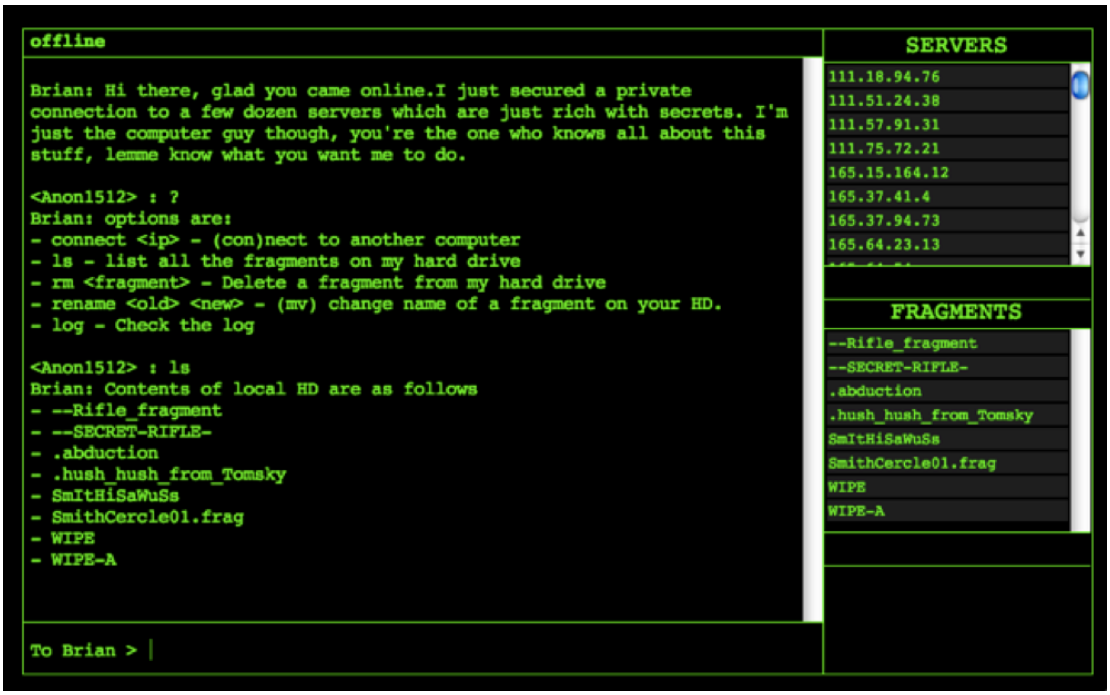

Figure 5.1: The start screen plus a listing of a player's starting information fragments early in a game of Conspiracy Forever.

Initial Story Fragment (rendered in English):
Walter wants Nikola Tesla to be powerful so...
Walter develops the X303 Rifle which achives...
Walter's goal to Make Nikola Tesla powerful.

Fragment variant 1, GeneralizeActor(Walter) => ?
Something (Actor) wants Nikola Tesla (Famous Person) to be powerful so ...
Something (Actor) develops X303 Rifle (Weapon) causing ...
Nikola Tesla (Famous Person) is now powerful which achieves its goal.

Fragment variant 2, GeneralizeRole(Nikola Tesla) => Conspirator
Walter (Scientist) wants something (Conspirator) to be powerful so ...
Walter (Scientist) develops X303 Rifle (Weapon) causing ...
Something (Conspirator) is now powerful which achieves its goal.

Figure 5.2: A base story fragment and two transformed variants used in Conspiracy Forever.

thoroughly explore): even when the system only used fragments of stories, humans were more than willing to amass a disparate collection of story fragments, piece together a story that they considered to be compelling (filling in the wide gaps with their imaginations), and describe the whole experience as an entertaining success. In hindsight, CF had an advantage over Minstrel Remixed and Skald: its story fragments were too short to be incoherent and players were acting as builders rather than judges. CF likely increased the engagement of its users and the perceived strength of the stories that its users were constructing by intentionally replacing random details in its seeded story fragments with evocative mysterious entities such as "Unknown" or "Something (Conspirator)".

The second major finding that CF provides relates to the first: a well-chosen domain and interface can disguise flaws in an MS3 and even has the potential to turn them into advantages. In this initial version of Skald, errors existed in the TRAM system which made some transforms error and return a null pointer rather than a generic name (e.g. 'Something'). The evocative "Unknown" labels in CF were actually the English rendered version of those null names. Upon further reflection, if I had thought to replace these unknowns with "Missing Data" or "Redacted," CF could have further capitalized upon the spy/conspiracy trope, turning these errors into even stronger thematic elements. Although CF is our first demonstration of the advantages imparted by a carefully selected domain, it is by no means the only demonstration of this concept, which is discussed further in this chapter and the succeeding ones.

5.2 Problem Planets

Problem Planets (P2) is the first game that uses a completed version of Skald as its engine. When starting the P2 project, our goal was to create an interactive

game to demonstrate the power of Skald. When playing P2, the player is tasked with a mission to travel to new worlds, and new civilizations, and boldly fix whatever problems are encountered. To do so, the player has a coarse level of control over the choices of a robot which tries to solve problems based on the player's directions. A little "gaminess" was added in the form of fuel, gold, and reputation resources which fluctuated to different degrees based on a user's choices. But by and large P2 was designed to operate like a computerized Choose-Your-Own-Adventure (CYOA) novel. Just to add some chaos (and because it was absurdly easy to implement) we also added an uncontrollable second robot to the mission, Robot Zed, whose secret purpose was to help demonstrate the range of potential stories in the game. He accomplished this largely by taking over on some missions and making the worst possible choices. This provided the player with some non-interactive stories to read and also "helped" players by providing a consistent character, the zany/bumbling sidekick, as well as a method for us to add chaos, difficulty, and levity to the game. Figures 5.3 and 5.4 demonstrate what players see when playing P2.

P2 was never formally completed (and so Zed never did get to wreak havoc on unsuspecting populations). That being said, Skald was "complete" before P2 started and new feature development for Skald was driven by the needs of P2. Over the course of its creation, we ended up with a great deal of experience with MS3 domain engineering, an exploration of a more Mexica-esque theory of story generation implemented in Skald (this path bypasses a number of context-related issues), new insights in framing a story for an audience, and a handful of new tools, and external APIs designed to help Skald offload challenging tasks to other programs. The remainder of this chapter is devoted to describing P2 and providing an in-depth analysis of the upgrades required to make Skald capable of creating

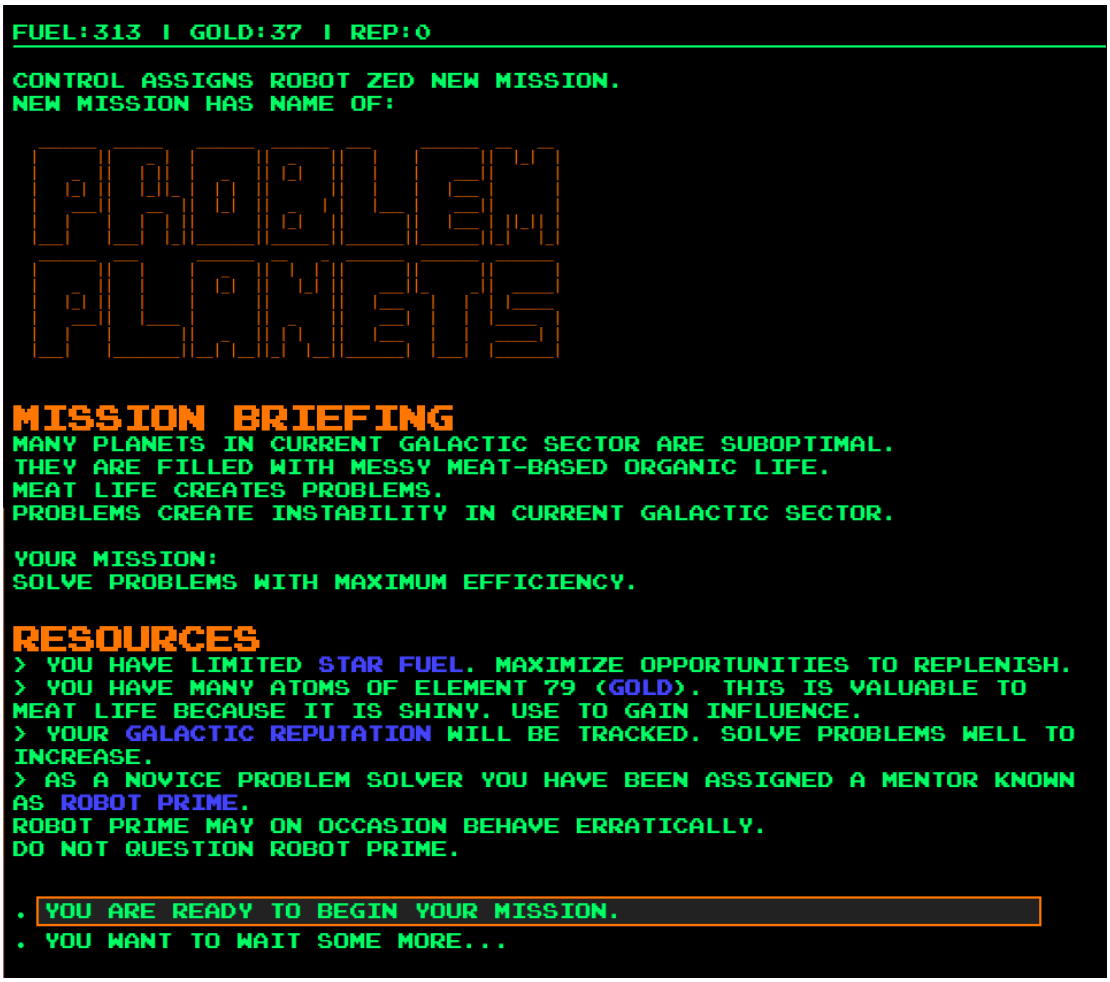

Figure 5.3: The adventure begins!

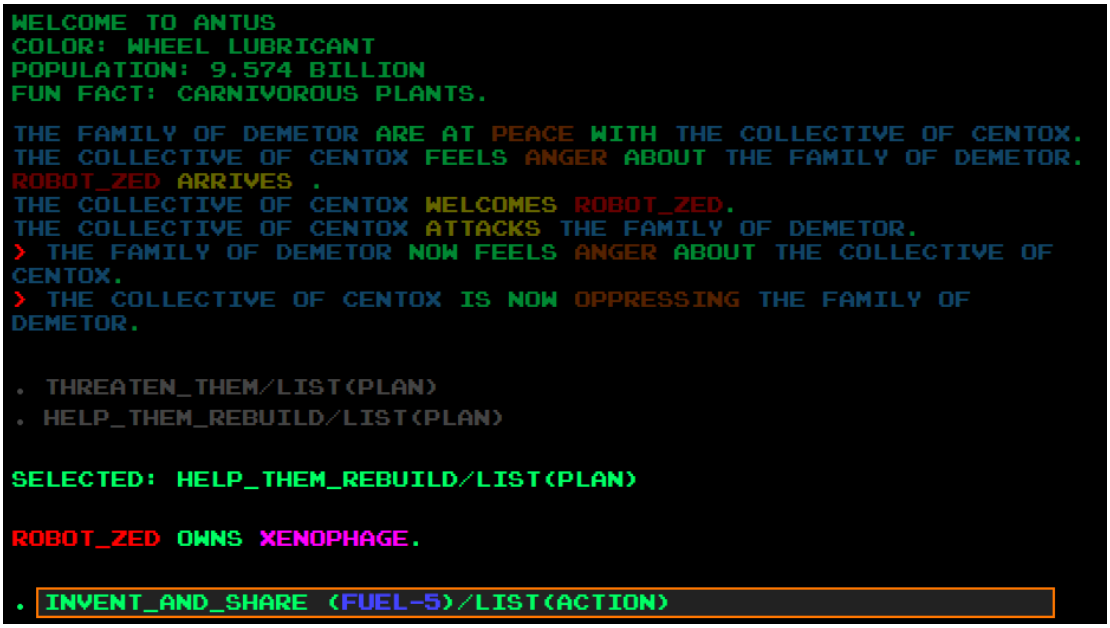

Figure 5.4: Midway through an adventure on a planet.

interactive stories for this project. As an example, here is the transcript from a whole story from Problem Planets:

> WELCOME TO HELLICUS
> COLOR: WHEEL LUBRICANT
> POPULATION: 5.716 BILLION
> FUN FACT: PAN-GALACTIC RADIO BEACON.
>
> THE ZARKIANS ARE AT WAR WITH THE HEGEMONY OF ANTASH.
> ROBOT_ZED ARRIVES.
> THE HEGEMONY OF ANTASH WELCOMES ROBOT_ZED.
> CHEMICAL_GUILD IS AFRAID OF ROBOT_ZED.
> THE HEGEMONY OF ANTASH INVENTS AND CREATES RADIOACTIVE GERBILS.
>
> **OPTIONS:**
> THREATEN_THEM/LIST(PLAN)
> APPEASE_THEM/LIST(PLAN)
>
> **SELECTED:** THREATEN_THEM/LIST(PLAN)

THE HEGEMONY OF ANTASH OWNS RADIOACTIVE GERBILS.

OPTIONS:
SUPPLY_THEIR_ENEMY (GOLD+10, FUEL-10)/LIST(ACTION)
DESTROY_FAVORITE_GOOD (FUEL-5)/LIST(ACTION)
PUNISH_THEM (FUEL-5)/LIST(ACTION)
PUNISH_THEIR_ENEMIES (FUEL-5, GOLD-10)/LIST(ACTION)

SELECTED: PUNISH_THEM (FUEL-5)/LIST(ACTION)

ROBOT_ZED ATTACKS THE HEGEMONY OF ANTASH.
THE HEGEMONY OF ANTASH IS NOW AFRAID OF ROBOT_ZED.

- LOST 5 FUEL.

OPTIONS:
THEY_SURRENDER_THE_GOODS/LIST(PAYOFF)

SELECTED: THEY_SURRENDER_THE_GOODS/LIST(PAYOFF)

THE HEGEMONY OF ANTASH IS AFRAID OF ROBOT_ZED.
THE HEGEMONY OF ANTASH TRADE WITH ROBOT_ZED.
ROBOT_ZED NOW OWNS RADIOACTIVE GERBILS.
ROBOT_ZED ARE NOW AT PEACE WITH THE HEGEMONY OF ANTASH.
THE HEGEMONY OF ANTASH NOW FEELS ANGER ABOUT ROBOT_ZED.

VISIT THE NEXT PLANET.

5.2.1 Authoring Tools

The first of these new features was driven by the need for a better authoring experience. Even with the Story Implementation Language (SIL was defined in section 4.3.3) offering a much less cumbersome way to encode stories than our original XML-based implementation, attempting to create a story proved to be a

tedious, slow, and deliberate engineering feat rather than a quick creative process. To make matters worse, errors in story encodings or improper use of words and conventions were common — and only possible to detect by observing the output produced by the system. This restricted the speed at which we could generate our domain to an unacceptable pace, given P2 was an experiment in iterative domain development as much as it was meant to be a working demonstration of Skald's capabilities. In the context of the iterative design of an interactive experience, this made domain development too slow and cumbersome. Our desire to quickly and efficiently write and debug stories drove the creation and design of the StoryWriter module mentioned in 4.3.3 which permits authors to write in English phrases and reduces the friction of getting an idea into a form understandable by Skald.

5.2.2 Adequately Steerable Stories

One interesting potential of computerized story generation that we've been distinctly aware of throughout Skald's development is the potential for interaction. Our initial focus was trying to rebuild and improve upon Minstrel, which didn't leave time to focus on this capability. But since one of the goals of the P2 project was to create a game, it was clear that the time to build interactivity into Skald had come.

In order to create a minimally interactive story generation loop in Skald, we merely needed to take a choice which is normally made by some aspect of the system and have a user make it instead. Once this was accomplished, in theory we would be able to easily scale up this behavior — so that users made more choices — until the level of interactivity was where we wanted it. There are a number of choice points within Skald but we figured that only a few of them would be acceptable for humans to manipulate. A large number of choice points have to

do with TRAM selection, ALP selection, or choosing the order in which nodes are filled in, but we believed that these choices would be too low level to be satisfying for users.

We explored providing users with control over a player character by allowing them to select actions for their character in the course of story generation. We dropped this idea as well after creating a few pilot stories to test it out on. These tests showed that it was too hard to steer stories — The user wasn't in control of most of the story and the system had no idea what the user's goals were so their respective influences on the direction of the story were often at odds to one another. In addition, as these stories grew in length, the system's inability to know which contextual tidbits from the past might have a bearing on any given other story element meant incoherent elements appeared more and more the larger the story became.

At the same time, we found that performing a "high speed escape" sequence from an action movie was perfectly doable because it contains a lot of direct cause and effect actions and no context needed to be incorporated into future actions. Thus, we knew the system was functional and some of our ideas were correct but that something needed to be done differently because anything involving more complex story elements was nearly impossible to create while maintaining coherence.

After all of the previously mentioned strategies had been tried out we opted to create a new feature for Skald that came with its own new type of choice point. We would stop creating large stories from a single scene template but instead create vignettes which were stories comprised of multiple smaller scenes. This allows Skald to completely fill in a smaller scene and then let the user choose which scene should come next. We reasoned that this extra guidance would help keep user's

goals aligned with the story goals and that we could maintain contextual elements within a vignette and ignore them afterwards which would effectively remove a maximum size from our stories. After performing some initial tests, we found that using this method allowed us to develop scene templates that were modular enough to create adequately steerable stories and immediately incorporated it.

5.2.3 The Vignette Generator

After observing stories generated by Skald in the preparatory stages of P2, it became clear that we didn't want to follow Minstrel's path of creating large stories via a large chain of cause and effect facts with a baked in contextual link between the setup and conclusion of the story. This sort of story would be obviously repetitive to a reader who was exposed to the same template a second time and isn't particularly expandable or multi-purpose. Thus, we needed a different method that would allow us to create larger free-form stories without running into the systemic deficiencies that tend to reduce coherence in such stories in proportion with their size. This is all due to the fact that Skald's handling of contextual elements is clumsy since it has no way to differentiate important from trivial details. The result of this is drawing random elements into future story events, a practice which only rarely allows Skald to properly maintain consistent focus on important story elements such as the main character, the primary goal of the story, etc. Since our goal with P2 was to create a number of story arcs that the main character and player would traverse, we needed a method to clear out unimportant contextual elements from time to time and guarantee that the important ones were consistently reintroduced when Skald dropped them. Our solution to this was to deviate from Minstrel's method of story expansion (ALPs selecting areas to add extra nodes to) and instead try to build what we termed,

"Vignettes," which are long stories constructed by stringing together a series of short stories which we named "Scenes." Each scene describes a plot point in the vignette and thus, by carefully controlling the initial conditions of each scene, we can create a cohesive vignette without ever needing to encounter the incoherence introduced by working with larger stories.

In order to accomplish this new plot-driven story framework planning, a new module needed to be created. The Vignette Generator (VG) module was our solution, which operates by taking the current story and choosing a set of scenes which might follow the current scene, offering them to the player, generating the selected one, and repeating. The VG is also responsible for identifying and storing important state information and reintroducing it into each new scene's template prior to generation to ensure that it is taken into consideration. Figure 5.5 is an example of one of our original sets of branching templates used to create a cohesive narrative.

We originally opted to treat the VG's planning as a constraint solving problem and developed an external API that would allow us to send completed stories to, and receive instructions for the next scene from, a Vignette Generator written in Answer Set Programming (ASP) and called as an external program. Our intention was that the ASP Vignette Generator would be able to quickly parse through thousands of combinations of facts from the "past" of the story along with a large library of small chunks of potential futures in order to provide a handful of linked future fragments as a template for story construction. This ASP-based system was unfortunately not implemented. Instead, an internal module was written which finds only the next story fragment rather than planning out an entire future. This system performs well, but lacks the planned ability of the ASP Vignette Generator to generate cohesive long-term plots. A full story template comprised

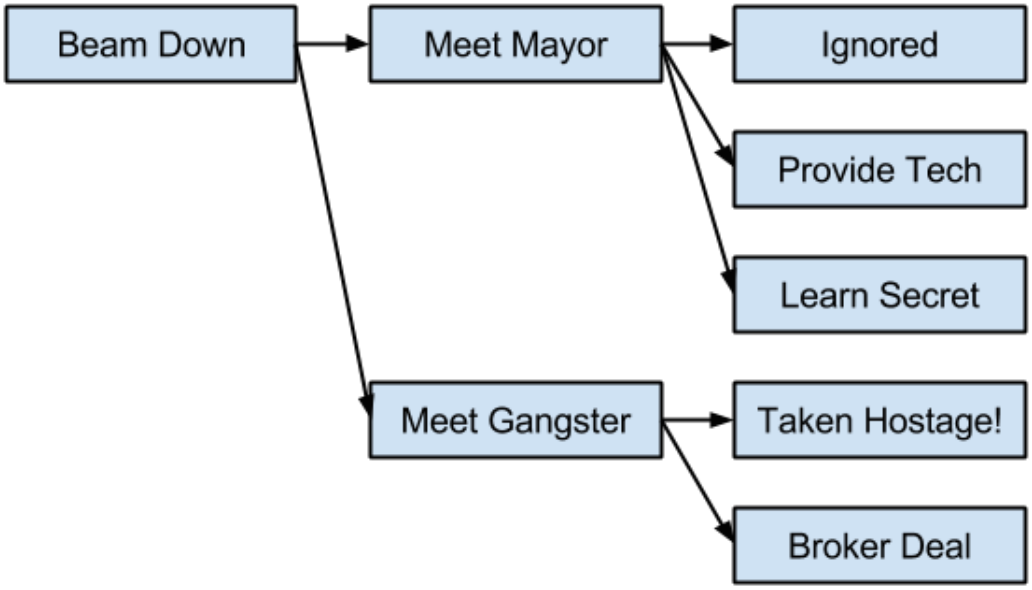

Figure 5.5: One starting template followed by those which could follow it.

of 4 vignettes has been provided for reference, see Figures 5.6, 5.7, 5.8, and 5.9.

In order to assist the VG in creating cohesive vignettes, new metadata was introduced to help tie scenes together, annotate gameplay values, and help players understand the choices they are making. There are many new tags which I will use the remainder of this subsection to describe. "RES" indicates that using this template will cause the player's in-game resources to change, for example, "04: RES FUEL-5" from Figure 5.7). "TEXT" indicates the text that the system should display to help a player understand what the template will do before picking it. An example of this is "06: TEXT FIND_MICROBE_TO_CLEAN_[E]." from Figure 5.7. "HIDE_NODE" indicates story nodes which are helpful for construction but will confuse readers or do not contribute to the meaning of the story. Hidden nodes are a convenient way of providing extra contextual information so that templates have a higher chance of generating a cohesive story without

adding repetitive and unwanted text to the end result. An example of this is "05: HIDE_NODE G0 G1" from Figure 5.6.

Arguably the most important new metadata element is the ARC_GOAL tag. Examples of this are seen in the plan and payoff vignettes in Figures 5.7 and 5.9. This points to the goal of the vignette which is selected by a player after reading the Setup. The VG will only allow Payoffs which have an ARC_GOAL which matches the one in the current Plan (although payoffs can still achieve or thwart that goal). This helps drive the plot forward and ensures that the player's stated goal is attended to rather than some other random detail. For this reason, all of our Plan and Payoff templates contain a reference to the ARC_GOAL.

The next new piece of metadata is tagging templates as SETUP, PAYOFF, PLAN, or ACTION templates. The intention here is that a template marked as SETUP contains the initial conditions for the start of a story along with a statement of its ARC_GOAL. After seeing a filled SETUP scene, players are given a choice between a number of PLAN scenes, each of which can be achieved through a number of ACTION scenes which cause a state change. After a PLAN/ACTION pair is accomplished, an appropriate PAYOFF is selected which ideally achieves the ARC_GOAL. It should be noted that PAYOFFS don't necessarily have to accomplish the ARC_GOAL (in which case the players would be punished) and although our current model includes only one PLAN/ACTION pair, there's no reason multiple couldn't be chained together to achieve larger state changes. Figure 5.10 demonstrates how one tree of SETUP/PAYOFF/PLAN/ACTION templates can be traversed to create a story.

The final addition to template metadata was the inclusion of the PRE and POST tags which denote preconditions and postconditions. In order for a new scene to be a valid choice for the VG to display, its preconditions must match the

```
/* Setup: Using Dangerous Things Hurts Environment. */
01: GST 1S_DGOOD.
02: PRE S0.
03: POST S_POST.
04: TAG SETUP.
05: HIDE_NODE G0 G1.
06:
07: B is a generic person.
08: A is a generic place.
09: E is a generic ecosystem.
10: W is a generic dangerous_good.
11: T is a generic good.
12:
13: G0: A wants to own W.
14: G1: A wants to own T.
15: A1: A buys from B.
16: S1: A owns T.
17: A2: A uses the T to invent W.
18: S_POST: E has status of polluted.
19:
20: G0 hassubgoal G1.
21: G1 plans A1.
22: A1 intends S1.
23: S1 achieves G1.
24: A2 precondition S1.
25: G0 plans A2.
26: A2 accidents S_POST.
```

Figure 5.6: Fixing The Environment Template. Setup fragment, vignette 1/4.

```
/* Plan: Get a microbe */
01: GST 1P_MICROBE.
02: PRE S1.
03: POST G2.
04: RES FUEL-5.
05: TAG PLAN.
06: TEXT FIND_MICROBE_TO_CLEAN_[E].
07: HIDE_NODE S1 G2 G3.
08: ARC_GOAL G3.
09:
10: R is a generic robot.
11: E is a generic ecosystem.
12: M is a generic microorganism.
13:
14: S1: E has a status of polluted.
15: G2: R wants to own M.
16: G3: R wants E to be healthy.
17:
18: S1 motivates G2.
```

Figure 5.7: Fixing The Environment Template. Plan fragment, vignette 2/4.

```
/* Action: Invent microbe */
01: GST A_INVENT.
02: TAG ACTION.
03: TEXT FORMULATE_[M]_BY_YOURSELF.
04:
05: PRE G1.
06: POST S2.
07: RES FUEL-15.
08: HIDE_NODE G1.
09:
10: R is a generic robot.
11: M is a generic microorganism.
12:
13: G1: R wants to own M.
14: A2: R invents M.
15: S2: R owns M.
16:
17: G1 plans A2.
18: A2 intends S2.
19: S2 achieves G1.
```

Figure 5.8: Fixing The Environment Template. Action fragment, vignette 3/4.

```
/* Payoff: Microbe fixes Plant */
01: GST 1Y_MICROBE.
02: PRE CS1.
03: POST S2.
04: TAG PAYOFF.
05: HIDDEN_RES REPUTATION+25.
06: RES FUEL-5.
07: TEXT CLEAN_[E]_WITH_[M].
08: HIDE_NODE CS1 S1 G1.
09: ARC_GOAL G1.
10:
11: E is a generic ecosystem.
12: R is a generic robot.
13: M is a generic microorganism.
14:
15: CS1: R owns M.
16: S1: E has status of polluted.
17: G1: R wants E to be healthy.
18: A1: R uses M to repair E.
19: S2: E has status of healthy.
20:
21: S1 motivates G1.
22: G1 plans A1.
23: A1 intends S2.
24: S2 achieves G1.
```

Figure 5.9: Fixing The Environment Template. Payoff fragment, vignette 4/4.

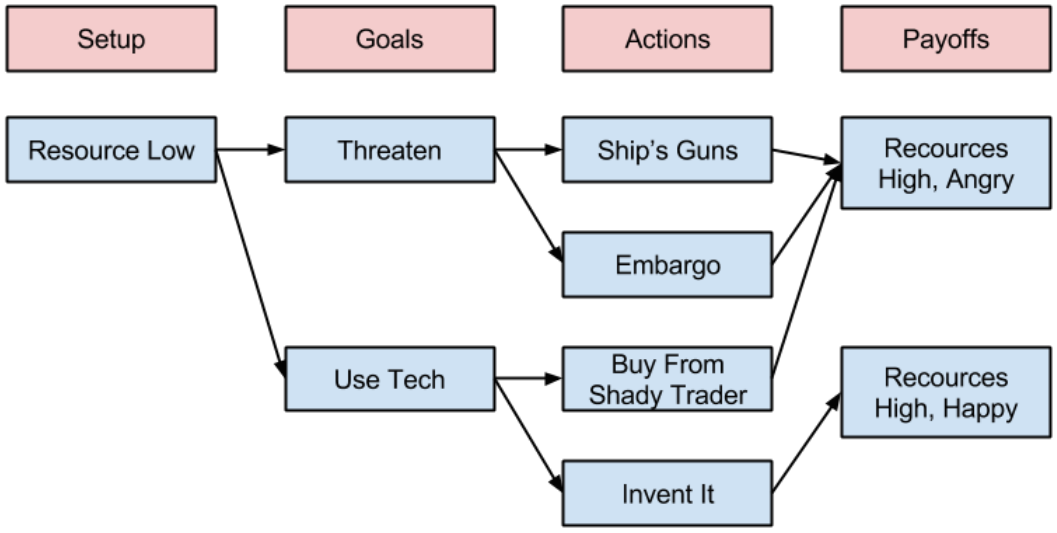

Figure 5.10: Branching tree of Templates of the form Setup, Plans, Actions, Payoffs.

postconditions of the previous scene. In a sense, this allows our scenes to operate in a manner similar to Mexica (which I discuss in section 2.3), where each scene is a way of getting from one set of states to another set of states. By using the PRE and POST tags to help us search, we are able to usefully integrate scenes into contexts they weren't originally intended for and still arrive at the proper conclusion for the vignette. This is exemplified by the structure of our templates: there are clear families of SETUP, PLAN, and PAYOFF templates which all revolve around a specific problem (e.g., pollution, overly aggresive neighbors, etc.) but each of our ACTION templates fits into a family of solutions (e.g., obtain technology, make someone like you). Although we author many of our problem families with specific actions in mind, if there's a pre/postcondition match outside of our intended actions, that just means there's one more option that we didn't need to author.

In a sense, the VG is a tool that uses Mexica's plot generation principles to help out where Skald is weakest: handling context in long stories. By creating a

scaffold that directs Skald in producing tiny stories, we enabled a new flavor of story generation within Skald. For further discussion on the differences between Skald and P2's stories, see Section 5.2.5.

5.2.4 Engineering a New Domain

Aside from all the upgrades that were required to bring Problem Planets into being, we also decided to create a brand new domain to support our authorial aims. While building this new domain we made a number of knowledge engineering discoveries leading to improved cohesion, consistency, symbolic and conceptual coverage, and overall quality — and by extension ultimately an improvement in the quality of the stories produced by the system. Despite this involving no programming, it is a technical achievement that amplifies the successes provided by the rest of the improvements created for Problem Planets.

The fact that the domain supports science-fiction scenarios was mentioned earlier but what was not mentioned is that we chose it intentionally for the same reason that we chose the Conspiracy Forever domain: suspension of disbelief. It also turned out to be an excellent domain due to the wide variability of stories told in the science-fiction realm. We consciously chose to focus on problems between large groups (be they cities, nations, or other organizations) in order to simplify our reader's expectations about the depth of our characters. We also found that the science-fiction domain could believably include stories about environmental challenges (e.g., dwindling supplies and rising levels of pollution) and societal progress (e.g., trading for better standard of living) on top of the standard Skald stories about violence and magic (or, in this case, technology).

Orthogonal to our choices about the themes of the domain was our effort to see what it was like to author a domain with an intentionally limited and well

defined set of symbols and patterns. As I will discuss in the next section, our choices changed over time but in the end all three of our authors agreed that our disciplined use of symbols and patterns was incredibly beneficial. Table 5.2 is a comparison of the lists of symbols contained in P2 and Skald's Arthurian domains.

In all categories save the goals, P2 has fewer symbols than the Arthurian domain despite the fact that the P2 domain is at least the same size if not bigger[1]. This is a result of the planned nature of the P2 domain which contrasts with the clearly ad-hoc nature of the Arthurian domain. Please note that the Arthurian domain contains duplicates (e.g., c-own, c-ownership, c-possess), near-duplicates (e.g., ask_help_from, ask_favor_from), highly specialized symbols (e.g., lock, raw_food), and five times as many noun categories. All of the duplication in the ad-hoc domain is unintended, the result of unconstrained story writing to fit ever-changing needs over the development of Skald. This can and should be avoided by creating a set of symbols prior to domain generation and insisting that all stories use only those symbols. As can be expected, the result of this error causes Skald to lose generation potential, since the symbols share the same meaning for the user but not for the algorithms. A TRAM search to find possible outcomes of someone giving an object to someone else using the verb 'give' won't match stories that use the verb 'pTransfer' or 'pGive,' meaning fewer potential matches and less information to generate content with. A similar loss of generative power is caused by having a large ontology: As ontology size increases, each symbol gets less coverage which means that TRAMs such as GeneralizeRole will have less options to choose from. In essence, Skald will 'know' less about each symbol in the ontology and have fewer options that aren't randomly selected. The TRAM system prefers to find matches with fewer transforms so the larger the ontology,

[1]P2 has more stories and more lines of SIL — but has smaller stories, more repetition, and fewer concepts than Skald's Arthurian domain does — so a true comparison of domain size is nearly impossible.

Symbol Type	Skald's Arthurian	Both	Problem Planets	Counts (Skald/P2)
States	hunger, affect, condition, know, lock, mhealth, ownership	feeling, health	stance, status, supply	9 / 5
Verbs	create, eat, give, askHelpFrom, askFavorFrom, ingest, mTransfer, pGive, pTransfer, transform, win	attack, buy, own	embargo, invent, meetsWith, oppresses, repair, steal, tradeWith	14 / 10
Goals	change-hunger, change-lock, change-ownership, change-possess	change-health, change-own	change-feeling, change-oppress, change-stance, change-supply	6 / 6
Nouns	animate, actor, peaceful, princess, hermit, wizard, king, witch, child, violent, knight, bandit, monster, troll, dragon, demon, zombie, animal, horse, bird, deer, inanimate, food, rawFood, berry, meat, badFood, tool, magic, fishingPole, weapon, mundaneWeapon, magicWeapon, potion, treasure, nature, rock, tree	-	organization, robot, microorganism, person, trade good, dangerous good, ecosystem	38 / 7

Table 5.2: Symbols in the domains of Skald and Problem Planets.

the less interesting the potential matches to a given search as it will skew towards smaller sets of nouns rather than potentially interesting ones far away. There is a case for having a large and specific ontology since intelligent TRAMs can find better contextual matches in some cases but I think that in general, an ontology that is more dense has more opportunities for good use of its TRAMs.

Beyond the general cleanups that separate the P2 domain from the slightly messy Arthurian domain, there are a number of less obvious optimizations that we consciously included when designing the P2 domain. The first optimization is to restrict the set of symbols to include as few as possible while still allowing authors to express their ideas. By using fewer symbols we're able to create greater coverage for the symbols meaning that Skald is able to find more matches for a given symbol and thus have a greater chance of providing acceptable context or identifying context-based subtleties.

One such subtlety, the creation and use of what we call, 'Conventions,' is chiefly responsible for our ability to express many different stories despite our extremely restricted domain. A convention is what one gets when the authors of a domain agree that a particular pattern of symbols has a special meaning above and beyond what its symbols alone convey. One concrete example is the P2 convention for the idea of 'X declares_peace_with Y.' Lacking a declares_peace_with verb, we instead use the meets_with verb since a meeting of ambassadors (in some fashion) is generally how an organization goes about peace negotiations. The resulting state change of such a declaration is as one might expect: X has a stance of 'peace' with Y. This is an example of symbol composition and it allows us to avoid introducing a specialized verb for declaring peace. This in turn improves the interplay between stories, because these more generic peace talks can have extra outcomes informed by other stories that include the 'meet_with' action

(e.g., insulting someone, establishing new trade-routes, etc.) and other stories can benefit from the context of the current meeting (i.e., other meetings might result in a change of stance to 'peace').

Another important element of our domain engineering is to remove redundant symbols by identifying those which require or imply the existence of another symbol and finding ways to either combine them or make one (or both) more generic. The original version of P2 included the verb 'declare_war_on' which is a great example of a mandatory coupling between the verb and its required result, 'x has stance of war with y.' Having a 'declare_war_on' action which doesn't result in a stance of war (even if just for a moment) simply doesn't make sense. Thus, anything the TRAM system finds to follow such a symbol that isn't its coupled symbol will add incoherence into the story. To make matters worse, coupled symbols require that TRAMs and ALPs match against two or more nodes (which is far harder to do than a single node,) so finding content that works with a coupled set of symbol is harder. Thus, coupled symbols should be avoided if at all possible and replacements are often easy to come up with once one is looking for them. As an example, we replaced the coupled declare_war -> stance_of_war pair with the more generic meets_with -> stance_of_war. Although the coupled verb is slightly easier for authors to understand, the new more generic verb is far easier for Skald to understand and work with. Meetings between countries can lead to war but we also encoded that meetings can lead to peace, embargoes, or trade agreements which means that it is easy to use details from one kind of meeting for any other kind of meeting which results in a win for generation potential and clarity.

A discovery related to the idea of coupled symbols revolved around the ambiguous concept of the "generality" of the symbols in the domain. Two symbols

in the domain sometimes have a relationship of one being a generalization of the other and it is a good idea to notice that and combine these into one symbol. An example of this is the verb ‘ transfer ’ which is a more general form of the far more specific verb, ‘give.’ Most comparisons of symbol generality aren’t as easy as this. However, after a few hours of thinking about the P2 domain, it is possible to create 2 generality classes amongst our verbs:

> **More General**: attack, buy, own, meets_with, repair, trade_with, embargo
> **More Specialized**: invent, oppresses, steal

While creating the domain for P2 we came to realize that having a high level of generality for all of our symbols improved the uniformity of our output stories and maximizes the generativity of TRAM recall. More importantly, however, having specialized symbols such as ‘assassinate,’ ‘coffee,’ or ‘out of gas’ is dangerous because they fill the roles of more general symbols but with added implied context which can lead to loss of coherence when the wrong patterns are formed (e.g., Alex drank the coffee to help get to sleep). Engineers of domains should be aware of the generality of their symbols and be prepared to deal with the implied context of those symbols which are more specialized. An example from P2 is the ‘invent’ verb which fills the role of the more general concept, ‘create.’ Having both verbs in the domain caused problems since they had similar contextual cues and recall would occasionally use invent improperly (for instance inventing a baby). To resolve this we removed ‘create’ from our domain in preference of invent. In exchange for using a more specific and theme-enhancing symbol, we had to give up using the full power of the ‘create’ verb. The tradeoff between expressivity and flavor should be considered by domain engineers while they are selecting the symbols to include in their domain.

5.2.5 The Development of P2

P2 was developed over 18 months and in that time a number of distinct strategies for storytelling were created and replaced, the domain that P2 was operating on shifted dramatically a handful of times, the class of stories that we were aiming to create altered, and the game as a whole was redesigned. Many of these changes were a natural part of the life-cycle of a new creative endeavor and all would have gone faster (or been discarded in the design phase) had we known then what was discovered while developing P2. The previous section discussed how P2 changed Skald, hinted at bits and pieces of its design cycle, and outlined how P2 operated when work on it ceased. This section provides the second half of that story: how it all came about. Not only will this provide a blueprint for future engineers aiming to create new MS3s and domains, but it should also provide some qualitative data in support of the assertions from the last section.

The goal of P2 was to create a concrete demonstration that Skald could be used to generate stories for an interactive game. To accomplish that end, our three-person team (two programmers and an author) agreed upon a choose-your-own-adventure style of interaction in which the player is presented with a branching narrative, piece by piece, in the form of chunks of text with a small handful of options. We also agreed upon an overarching theme of "environmental problems" but took a few months of tinkering with game designs to finally decide to create a narrative that would emulate serial science-fiction shows of the 80s and 90s. A player would choose a handful of crew members from the player's spaceship to send down to a particular planet in order to interact with socio-political figureheads with the goal of altering some current set of environmental issues. Once a player "solved" a planet they would move on to another planet with a new set of problems. Each planet was to be represented as its own independent vignette

but some of the actors in the stories would be recurring characters/powers (e.g., arms dealers, alien scientists, and other interstellar empires) who were conserved, along with their relationships to the crew, between vignettes. In additional to the repeating characters and empires, there would be a larger fixed storyline (which was never pinned down) that the players would slowly uncover in their dealings. To keep the individual scenes in the vignettes just as interesting as the overarching elements, we planned a diverse range of actions for players to undertake, including, but not limited to: brokering deals between political figures, taking scientific measurements from the field, gathering information from scientists or activists, having crew members be taken prisoner, rescuing imprisoned crew, and even romantic liaisons between crew members and actors in the story.

In order to accomplish all of this, we would create a large noun-ontology as well as a big base of acceptable verbs and states, in order to support all of the story variety. Since a planet was represented as a vignette which would take place over a number of scenes (i.e., individual stories) we would copy all "current" state nodes out of a completed scene and into the next (to provide context). Additionally, all repeating characters and crewmembers, as well as all states relating to them, would be kept track of and injected into vignettes when needed. In order to accomplish the choose-your-own-adventure style of interaction, we would simply create large templates, populate them with characters (including the crew the player had chosen to send down to the planet), and then generate the story temporally, starting at the beginning. Every time one of the player's crewmembers takes an action, we would use recall to determine a list of acceptable actions for that character to take at that time and ask the player which of those actions they would like to take (mimicking how Riu handles its interactions).

The Rise and Fall of the Eco-Graph

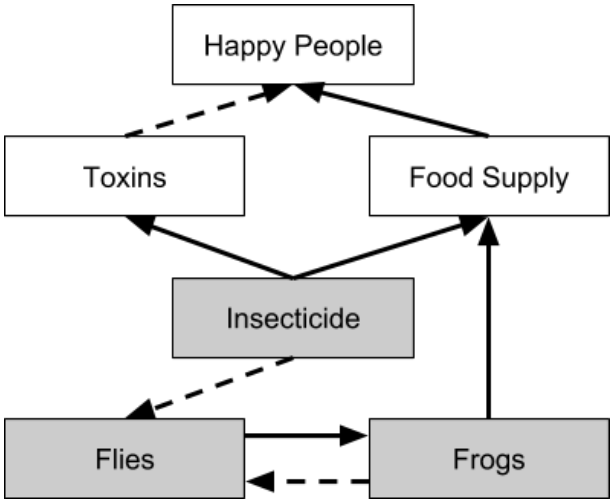

Figure 5.11: A simple ecograph. Solid arrows indicate a change in the origin will increase the destination. Dashed arrows cause a decrease in the destination when the origin is increased. Grey boxes are alterable by players.

Although the initial version of P2 was impossibly large in scope, we didn't realize that until much later. After creating the software support for the Vignette/Scene model of our stories, as well as the definitions for our noun ontology and set of states and actions, we came to the mistaken conclusion that P2 could be a little more of a game and less of a story generator. As a result, we created the eco-graph, a description of the interconnected environmental elements that the players would be manipulating for each planet. The eco-graph was a collection of nodes with edges describing the pressures that the nodes exerted upon other nodes. An example of this is a simple two node graph with a "frogs" and a "flies" node along with two edges, one indicating that as frogs goes up, flies goes down, and the other indicating that as flies goes up, frogs goes up. Our eco-graphs tended to have a dozen interrelated nodes in them, and the point of the game was for players to discover the nodes and edges of the eco-graph through story actions. Additionally, as players interact with characters, they would change the

way various people felt about them and about the nodes in the eco-graph, allowing them to push around the variables in the graph (for instance, by pleasing a politician, the player might acquire a favor which could be used to increase the frogs' population).

As it turns out, the eco-graph is itself a game and could keep users entertained with no story involved at all. Our attempt to add in "gamey" elements to what was otherwise supposed to be a storytelling system simply provided us with a distraction from the original goal of creating a player-controlled story. Additionally, eco-graphs being manipulable by players changing the mind-states of the political entities associated with the nodes has a large amount of subtle context required to make it read properly. We realized early on that the eco-graph element was an entire game unto itself. When we made that realization we should have removed it but we didn't because we thought it would make a nice backbone to guide player choice and make the experience more like a game.

In order to create stories that let players manipulate the eco-graph, we thought we'd end up with story fragments such as, "You compliment the frog lobbyist who is now happy. Frogs +1, Mosquitoes -1" or "You abduct the pesticide magnate, Mosquitoes +1, Frogs +1." Unfortunately, this sort of story is not so simple for a computer to understand. P2 needed some concept of which actions are positive and which are negative in order to know that the abduction mentioned above suppresses the supply of pesticide. We quickly realized that things were not so simple: if the frog lobby were about to vote to enact some frog overpopulation control, abducting the head of the lobby might actually cause no change whereas failing to abduct this leader would cause a negative change! Due to these complications we opted to tie promotion and suppression activities to the happiness of the people involved. If entities felt positively towards you and your crew then

they would give you the option to promote their ecological factor and if they felt negatively towards you, they might just make a change without you. By building out a tiny model of this and playing it a few times, it became clear that this feelings-based game created a situation in which players were rewarded by taking indecisive feel-good actions and resulted in an undesirable game that felt more like controlling a politician shaking an endless series of hands than a robotic starship captain.

One additional problem with our happiness-based approach demonstrated a major limitation for open-exploration style games implemented in an MS3. Because the eco-graph presented a puzzle to players which could be solved in a number of ways and required a fair amount of exploration, it was impossible to know the player intent at any point — which in turn meant we could not have actors take action in order to thwart or support the player's goals. This realization made clear to us the importance of either having a clearly stated mission for the player or asking them to state their objectives, so that P2 could work with them in order to create some challenge and, ideally, a nice story arc.

Because we couldn't see a way to salvage our politically-driven game, we scrapped the eco-graph concept and returned to an only slightly gameified interactive story. We resolved to have the entire game aspect be involved with managing the player's resources, which would be spent to follow certain narrative paths (e.g., buy terraforming tech from a trader) and be the tangible rewards at the end of each vignette (e.g., the grateful population gives you their spare fuel).

Template Based Story Arcs

One of the clear problems we were seeing with the eco-graph-centered stories was that they had no clear narrative direction or, to put it another way, they

seemed to be transcripts of a simulation rather than narratives. With the ecographs out of the way, adding narrative flow to our stories was the next clear goal. In order to resolve this and clear up some of the chaos in the resource system at the time, we opted to categorize all templates and stories as either setups or payoffs — with the intention that a complete story would consist of a setup and then a payoff leading to a single plot element. To enhance the cohesion between setups and payoffs, we created a new rule: the postconditions (state nodes) of the setup must match the preconditions (also state nodes) of the payoff. This allowed us to create a single setup with multiple viable payoffs depending on how the setup was filled in. To create player interaction, we gamified the setup/payoff model by including a visible cost and hidden benefit for each payoff and then programmed P2 to allow players to choose between acceptable payoffs.

The tidy mini-stories created by the setup/payoff model were quite nice and for the first time in P2's development we felt that we were on the right track as far as story generation was concerned. That being said, it took a lot of authorial work to generate the templates (which were written in XML, which was a tedious and error-prone process). Our author requested something more like English which resulted in the StoryWriter module. This module reduced our authoring/checking/debugging time from 20-30 minutes per story down to 2-5 minutes. With a 10x improvement on our content generation we found ourselves able to prototype and test in bulk rather than one story at a time, which brought into focus two large issues: we were largely unable to reuse our payoff templates for other setups, and players didn't seem to have much meaningful choice within a story.

Largely to combat the first of these problems we chose to rewrite our story library from scratch using an reduced set of nouns, verbs, and states which were

all intentionally selected for similar levels of generality and lack of contextual requirements (as was discussed in 6.2.4). The new stories using the reduced symbol set were, as expected, far more reusable, simply because there were more matching symbols for the same quantity of story library. The drawback was that we had to start using conventions, a measure which had the potential to cut both ways: it enabled more matching between stories but could be a source of incoherence because of this increased matching between previously unrelatable elements. Fortunately, the inclusion of conventions had only a positive impact, although the reason for this could be simply that we tried to limit our use of conventions as much as possible.

While performing the story library rewrite that was mentioned above, we were able to work on the issue of meaningful choice and knowing our user's intention. To accomplish this we used the new reduced symbol set and the increased utility of templates that it brought about to expand our setup/payoff model of modular templates in a fashion that would give us a better understanding of what a user wanted to do. We reasoned that between the beginning and ending of a story there is plenty of room for a series of goals and actions which will lead to increased variety, the system knowing what the user's goals are, and, hopefully, making the stories contain more choices that are seen as meaningful. The resulting arc displays a story setup to the user, then lets them choose between a few goals for resolving the presented problem, then provides another choice between a few actions for accomplishing the chosen goal, and finally displays the result as a payoff. To further enhance our situation, we also engineered the templates comprising each of the 4 arc situations (i.e., setup, goal, action, payoff) such that we could reuse templates as much as possible. In order to do this we created a few classes of templates for each situation and created the next piece of the story with a

class of templates in mind rather than a specific one. An example of this is the 'ecology-damaged' class of setup templates which contains a handful of differently flavored templates leading to a world with a damaged ecology. This class of setup templates is followed by a choice between 'use technology,' 'threaten planet,' and 'attack planet' goal classes, each of which can access the 'use biological agents' and 'use technological gizmo' templates as well as a few more for threaten and attack. Based on the goals and actions, one of a number of multi-use payoffs is selected which fits the choices the user has made. Because our content is applicable in multiple ways, we are able to multiply our authorial contributions by some factor compared to our previous methods. We did this working under the assumption that the new setup/goal/action/payoff arcs would allow users to guide the flow of the story — which is another quality that was missing from our previous methods.

5.2.6 The Authoring Experience - P2 vs. Skald

Because P2 uses Skald's story generation potential in a novel fashion, it's useful to examine the different experiences that authors have working with the two systems. From a high level, working with Skald feels a little more like creating random stories whereas building the small templated scenes for P2 feels a bit more like defining simple rules. Because both systems can be said to use the stories and templates as rules, this indicates that for better or worse, P2 puts authors a little closer to the workings of the system. Some authors may prefer having more clear connections between their work and the output of the system that P2 affords. Others may prefer simply writing stories and being surprised what comes out of a system more like Skald.

P2 feeling like authors are 'closer' to the system extends to other attributes as well. One example is the observation that authoring for both Skald and P2

can feel like a creating a handful of mad libs which the system then uses a bit of intelligence to fill out. Since a mad lib is quite literally a templated story, this has potential to be an accurate statement. Because P2's templates are smaller and have less room for variation, they are often small story chunks with only one or two blanks left for the system to fill out. Authors creating such small templates are correct in feeling like they've just produced a small mad lib. Skald's templates on the other hand are larger and have the potential to contain more blanks, many of which can be interrelated in interesting ways. Often times, authors can tell all the potential scenes that a P2-style template will produce, while the results of a Skald template are less easy to predict. This is another reason why authoring for P2 feels a bit like manipulating rules for a story machine while authoring for Skald feels more like an artistic pursuit.

The final major difference between authoring for the two systems, from an experiential point of view, derives from the comparative ALP use of the two systems. ALPs, especially groups of ALPs all being applied to the same story, produce varied and unpredictable results. In P2 this level of variety would break the state-based scene-to-scene matching, making it impossible to get to the end of most stories. Thus, while Skald uses ALPs as its main source of variety, P2 has them all but turned off — and utilizes primarily scene sequencing to provide variety. This difference reinforces the previous message that working with P2 is a more predictable experience akin to creating rules, tuning a machine, or performing some sort of engineering feat. In contrast, working with Skald's unpredictable ALPs feels more like an art in which an author must predict how a handful of complex processes might interact.

5.2.7 Findings

Although our supporting evidence is purely qualitative, the creation and exploration of P2 yielded a number of both positive and negative findings. Like Skald before it, the majority of work on P2 involves improvements to its coherence or meaningful expansion of its PSS. Many features improve one at the expense of the other and thus there is a constant struggle to find features, implementations, conventions, and other structures in the system, stories, and templates that are a net win (i.e., improves coherence without constraining the PSS too much or vice versa). Almost all of the negative findings about Skald or P2 can be linked back to this struggle in some way.

Our experience with P2 has two related negative findings: It is extremely difficult to get either long or highly detailed stories to work properly. This is predictable behavior: a simple story with 2 nodes, each of which has 2 potential facts, A and B has 4 potential stories — AA, BB, AB, and BA — while increasing its length (more nodes) or level of detail (more potential facts) will result in more possible stories. Since there's always been a tendency for changes in the PSS to inversely relate to changes in coherence, the fact that making longer or more detailed stories leads to less coherent stories is not surprising. While these findings were predicted, the useful lessons of these two findings lies in the specific reasons for these problems.

Our long stories were failing largely due to the fact that there is no way to attach a value to contextual elements indicating that some are more important to the story than others. Most stories have a repeated central element such as a hero, villain, macguffin, recurring plot element, etc. The inability for Skald or P2 to highlight some story element as "important" prevents it from focusing on any given element, causing longer stories to be aimless, an undesirable trait which

is obvious to readers. Additionally, because recall is performed on a target node and its immediate surroundings, it is weighted more by nodes that are closer to the target in the story graph. Thus the further a node gets from the "head" of the story (i.e., the part currently being generated) the less likely it is to be used. This makes both Skald and P2 act like they have long-term amnesia and makes it extremely difficult to create any meaningful persistent context. ALPs tend to handle this type of issue (e.g. detecting where new details should be added and doing so), but we found that they couldn't select meaningful context and thus did not generate contextual stories with the reliability we wanted. In P2, switching to our new method of building stories in four pieces, including bringing some context over from piece to piece manually, did allow for longer stories than the monolithic model that Skald uses. It would be possible to further ameliorate the situation by including some new programming (be it in ALPs or new modules) that creates contextual cues, chooses and injects repeated elements, and generally find ways to temporarily import useful facts so that new content isn't generated without important context.

While we could extend story length by introducing the four piece story method (and we could extend them further by adding more pieces), we did not find a good way to maintain coherence while making stories more detailed. The problem with producing highly detailed stories comes from the lack of details in the templates and story library. Complex story elements (pieces of the story which have implied contextual information, discussed in section 2.1) present a problem to Skald because it can't tell what assumptions humans will draw given a particular pattern. Since Skald can't tell what sets of elements will hold implicit meaning for readers, the more detailed a story becomes (and thus the more context there is for humans to draw assumptions from) the more likely it is that complex story

elements will arise and cause divergence between the computer's understanding of the story and the reader's. The larger this divergence, the more opportunity there is for incoherence. Thus, highly detailed stories tend to be difficult to create correctly. To make matters worse, as was mentioned previously, Skald's recall and ALP systems tend to work only with context proximal to the node being constructed — and thus events being described by a large number of nodes will only be partially "in memory" during construction, often leading to important contextual cues being ignored by the generator. Unlike the problems with large stories which can be fixed by extra engineering, the difficulties with complex story elements are a difficult problem that I see no clear solution for. It may be possible to clearly describe all relevant details and connections in every story, and then determine which details to render into text and which to use merely as context, but this seems to be one if not two AI-Complete problems.

In addition to the negative findings that we discovered while working on Problem Planets, we were also able to successfully demonstrate that creating a new module for plot sequencing allowed P2 to generate stories in a more Mexicaesque manner. Working with this, we were able to create a new module that provided a novel way to use Skald's templates: as building blocks rather than monolithic story containers. We also discovered that this new use of templates was a far more transparent method of creating longer stories than relying on ALPs to expand generation beyond the boundaries of the initially chosen template. /colorredIt should also be noted that a number of our successes came about while attemping to upgrade our knowledge representation by cleaning up our domain. It is highly suggested that the engineers behind other MS3s and story generators consider the benefits not only of careful domain engineering but also examining their own practices to uncover other successful techniques.

Chapter 6

Interviews

After the years of system building, testing, and tweaking that resulted in Minstrel Remixed, Skald, and Problem Planets, we produced a number of promising results which became the subject of a handful of papers [Tearse et al., 2014, Tearse et al., 2011b, Tearse et al., 2011a, Tearse et al., 2010b, Tearse et al., 2012] and are examined over the next few chapters. While learning how our systems functioned and finding ways to improve them, we also discovered a number of problems. Oftentimes these problems were either fixable or could be glossed over, but some we were never able to overcome and provided limitations which guided the scope of what we could expect from our output.

Truly understanding a thing requires knowing all of its characteristics, not just those which are positive or helpful. To this end, this chapter is dedicated primarily to the issues that arose that seem to be endemic to MS3s. Since this discussion is about the problems exhibited by our systems and the limitations that they lead to, it concerns a number of perhaps unpublishable findings. To add some evidence that these findings are valid (and to further compare our three exemplar systems) I interviewed Dr. Perez-y-Perez and corresponded with Dr. Zhu (the creators of Mexica and Riu, respectively) to get their insights into their creations. Through

these conversations I have concluded that there are some interesting behaviors that all three systems clearly have in common. It stands to reason that the more MS3s exhibit similar behaviors (be they strengths or weaknesses), the more likely it is that these patterns are inherent to the entire class of storytelling systems rather than being an implementation-derived attribute.

6.1 Six Troublesome Categories

I was able to distill the limiting attributes of Skald and P2 down to six individual categories about which I formed a number of questions for Dr. Perez-y-Perez and Dr. Zhu. I will attempt to collate the conversations that I had based on those categories into one discussion here, but it is important to note that this process is inexact since one conversation was through email while the other was done through a video conference. Additionally, although I started with the same five topics and asked many of the same questions, I modified or added questions to these interviews based on the knowledge of Riu and Mexica that I had gathered from books, papers, and conference presentations.

6.1.1 Authoring is Difficult

Few systems are stress tested on different domains or by people other than their authors. This leads to a big question looming over many projects: Does the system only work in specific conditions when coaxed in the right way? This certainly applies to our work on Minstrel: If we want a high likelihood of producing a specific class of stories, we have to know what type of story we're aiming for and do a lot of support work to make sure they can be created. For us that means tailoring custom starting templates, writing a handful of stories to support the

more complex actions that we're looking to see, etc. Once that's all put together, asking the system to generate a handful of stories will guarantee that a few at least fit our original specifications. Without proper elements and patterns loaded into the generative space, our system has no chance of producing a specific desired story.

When asked whether Mexica exhibits similar issues, Dr. Perez-y-Perez indicated that the question doesn't really apply to Mexica since it avoids predefined structures as much as possible and that its recall behaviors act on a small scale that wouldn't benefit from the types of setup that I mentioned in my question. He also indicated that Mexica regularly produces some percentage of undesirable stories but that his research is centered around the artifacts that Mexica is capable of producing and the processes that it takes to create them. Thus far questions of the reliability of Mexica's story generation across distributions of stories has not been a focus of his work.

When asked the same question, Dr. Zhu wrote,

> Riu heavily relies on the SME analogy making algorithm, which is known to be rather "brittle" with respect to the way knowledge is represented. We configure SME in a way that is the least strict possible, but some brittleness still remains. For example, when authoring a new story world (composed of the main story graph and a collection of memories), it is important to decide (beforehand) a vocabulary of actions/predicates/adjectives/etc. that will be used in the frame-based representation. If those are not authored carefully, SME will not be able to find any meaningful analogy, and the system will not perform as expected.

Thus it is clear that Riu and Minstrel both share the issue that their generative components fare far better with a carefully authored and well defined domains. As the Computer Science adage goes, "Garbage in, garbage out"; it is not particularly surprising that an inappropriately prepared domain (the input in this case) yields

undesirable outputs. This state of affairs would not be a problem if domains weren't also difficult to author.

While working with Minstrel, we found that the best way to author its domain was to hand-create a number of concrete stories and then loop between generating stories and tuning our story library while aiming for some goal. We also spent a lot of effort setting up proper templates to interface with our story library and goals. This all takes a week or two's worth of concerted effort for a modest-sized domain. Unfortunately, building a domain requires exponentially more work as it increases in size (since every new symbol will need to have its relationships with a large number of other symbols defined), and adding to a domain requires similar effort be spent refactoring existing pieces. Testing a domain is also difficult because it is a subjective exercise which requires many iterations—and any bugs which do show up in the end results are often attributable to a confluence of many elements rather than a single easily fixable flaw.

My interviews indicated that since Riu shared similar engineering approaches, creation timeline, and testing strategies, it too is difficult to author for. Dr. Perez-y-Perez reiterated that Mexica's goal is to be a research tool (rather than a generation engine) and thus the domains don't require as much tuning or testing since there is no "desired" output to be aimed for. That all being said, he also did acknowledge that authoring a domain takes a lot of knowhow to do properly and that producing a domain takes a significant amount of reflection and effort.

Thus, all three of our exemplar systems seem to share the same weakness: that proper domain engineering is difficult, time consuming, and required in order for the system to work "properly." Dr. Zhu indicated that she and her team may be able to remedy this problem by using a natural language processor to automatically extract properly formatted domain content from a corpus of stories.

Until that effort is complete or some way is devised to simplify domain engineering for MS3s, the difficulty of hand-building a domain presents an upper-bound upon the amount of knowledge that is reasonably stored in the domains of these systems.

6.1.2 Abstraction is Required

One of the main reasons that authoring is expensive is because correctly teaching Minstrel which symbols are "like" other symbols (and thus are good candidates for TRAM swaps) is hard. In order to teach the system that a sword and a knife are interchangeable, stories about swords must be created alongside duplicate stories about knives, thus making them equally likely to be chosen when a mutation is called for. The original Minstrel included a strategy to reduce this burden: a hierarchical ontology for nouns. This allows it to understand that some nouns are "closer" to other nouns and thus, when a sword is mutated in a story, the change that another weapon is swapped in for the sword is the most likely, while tools are less likely and other nouns such as houses or animals are less likely yet.

While working on our domains, we were unable to find other situations where adding in hierarchies made sense, but we did find two other strategies which help us to increase the amount of expressiveness displayed by a story library of a given size. The first strategy is abstraction: replacing a number of similar symbols with a single abstract symbol. An example of a simplistic abstraction that we used in Minstrel Remixed is that the verbs 'give', 'steal', and 'create' are all represented as the verb 'transfer'. Using this abstraction, stories about stealing are suddenly related to stories about giving and stories about creating. The result is that many patterns in the story library are suddenly "near" one another, making mutation between these similar patterns possible. Thus, where separate sets of patterns were required to support multiple very similar verbs, only one set of patterns

is required to support the single abstract symbol. This reduction in authorial burden is significant and improves the performance of the system dramatically. The downside of this is a loss of specificity: using the example abstraction, every story has one flavor of transferring objects from one character to another. A can give a thing to B and then feel angry or sad towards B but the nuances of theft or accidental loss are forsaken.

This loss of specificity is sometimes not acceptable. The simplest solution is to create a separate symbol. Unfortunately, this symbol now exists as a separate entity and as far as the system is concerned, is as similar to its original symbol as any other symbol (i.e., stealing is as similar to transferring as it is to eating or killing). Thus, all existing patterns to support transferring need to be recreated for stealing by adding them to existing stories or creating new ones.

In some cases, specificity in the results can be regained by a slightly more complex use of abstraction. An example of this is that we found a way to "cheat" while handling emotions by realizing that the specificity is only important to the rendered English results and not to the story representation. All emotions are represented as a noun having a positive, negative, or neutral affect towards another (or the same) noun. We found that the system does not need to comprehend the difference between anger and sadness to successfully add them into a story. Both of these negative affects can be reasonably understood to motivate the same set of actions and that the important part about these emotions is that they are rendered as a real emotion for the reader. Thus, the statement, "Kyle's negative feeling for Sam caused Kyle to attack Sam" is functionally equivalent to, "Kyle's anger for Sam caused Kyle to attack Sam." From the system's perspective these two statements are identical but, from an aesthetic perspective, the first example is less desirable. Thus, when handling emotions, Problem Planets uses the context

surrounding the emotion to alter which English word is used to represent the emotion. This sort of post processing allows us to use abstractions while simulating some of the specificity that was lost when we condensed sadness and anger into the same 'negative affect' symbol.

The way we handled emotions is close to our second strategy for improving the expressivity afforded by a given size of story library: convention. In Problem Planets, we employed a number of conventions to allow us to represent similar but nuanced concepts as particular patterns of symbols which were identified by the ALP system and acted on appropriately. An example of this is that we found that one of our domains needed a way for states to declare war on one another but that overloading the already existent 'meet_with' verb was better than creating a new 'declare_war' verb. Thus, declaring war is now always a discussion followed by a state change. We detect this convention and alter the wording when we render the text to English but it also means that complex concepts such as declarations of war and declarations of peace are very similar patterns and thus much closer to one another than two separate and unrelated symbols might be.

Riu shares these difficulties :

> Exactly the same is required in Riu. For example, we use the term "go-to" to represent "walk", "swim", "run", etc. Also, we needed some conventions, for example, we could represent the fact that "Ales" is "sad" as: (is Ales sad) or as (sad Ales), but if SME is to find those two facts similar, they must be represented using the same convention...

When later asked if Dr. Zhu et al. explicitly tried to include or avoid the use of abstraction or convention, they indicated that initially they happened to build using those two concepts and eventually (much like we did for Problem Planets) ended up designing their domain to take advantage of these concepts. Also, much like Minstrel uses local context for story lookups, Riu uses a more structural abstraction as well:

> . . . all stories have an additional layer of annotation using "force dynamics." We intentionally added this layer of abstraction because SME is designed to find structural, not surface, similarities. We then create stories where the different force dynamics are heightened. In short, force dynamics helps SME find high-level analogies between scenes, not just surface ones (e.g., character wearing the same color of clothing).

While Minstrel and Riu have many obvious similarities in this area, Mexica does not seem to have any truly parallel structures. Dr. Perez-y-Perez indicated that since Mexica is a plot generator, its results are already a level of abstraction away from what Minstrel generates (more on this in section 6.1.5). He also said that Mexica has an input format that's somewhat difficult to write stories for and doesn't allow much flexibility. As a result, it may not really allow for abstraction or convention.

Abstraction, convention, and other methods of indicating to the system that one symbol is "closer" to another symbol (and thus a better candidate for mutation in some situations) are all clear wins that show up in both Minstrel and Riu. The upside of using these tools is that when they're consciously employed during the design of a domain, they greatly increase its expressivity while only marginally increasing the difficulty of engineering it. The downside of their use is that troubleshooting becomes more challenging. Additionally, since more domain knowledge is stored in patterns rather than individual symbols, it is harder for humans to add to such a domain without explicitly knowing the conventions and abstractions that are in use.

6.1.3 Nonsense and Creativity Go Hand-In-Hand

Minstrel exhibits a tradeoff between novelty and creativity. At one extreme, our system could generate stories that make complete sense but are never novel

(i.e., copying the stories from the domain) while on the other, as the system is tuned to produce more novelty, it also generates more undesirable stories.

An almost ideal example of both sides of this was provided in section 2.1.6, wherein the dragon Smaug poisoned a king and fed him to his rival, the knight Launcelot. This story is clearly nonsensical (a knight wouldn't eat a king) but as a near miss (similar to Tale Spin's misspun tales), is also only one step away from being a novel and interesting artifact created by Minstrel Remixed. The whole thing was made possible because the domain does not have extensive knowledge encoded into it about what eats what and there is no current mechanism for easily encoding negative rules such as what does not eat what. Thus, there is no way for Minstrel Remixed to know that it is breaking the reader's expectations about the world. When asked about these types of results from Riu, Dr. Zhu indicated that her system displayed very similar behavior:

> Some of the most surprising stories were generated because Riu did not understand the semantics of a noun or verb, and the generated story should not make sense, but happens to make sense by chance, resulting in a surprising story.

While one of Minstrel's research goals is to model creativity, Riu is not intended for that purpose and as such, Dr. Zhu and her team haven't focused on analyzing the ratio of creative vs. nonsensical outputs produced by Riu. The research done on Mexica is similarly not helpful to answering this question since Dr. Perez-y-Perez is focused on demonstrating that his system can produce individual creative artifacts rather than improving Mexica's ability to maintain coherency across the entire range of outputs.

Thus, while Minstrel is the only system for which this particular trouble is being explicitly measured, I believe that a simple mechanical explanation for this behavior is sufficient. Boden defines one important component of creativity to

be novelty, meaning that a creative system must produce output that is different from its inputs [Boden, 2004]. Minstrel Remixed can produce more novel outputs based on a given input by increasing its possible story space. This can be performed either by increasing its generative potential (i.e., more mutations of story fragments as they're being added to the story) or relaxing its filters (i.e., the ALPs which check story fragments for acceptability). In either case, a common sense reasoner which is judging the outputs will have some set of filters which will reject some of the outputs as nonsensical while accepting others. Assuming Minstrel Remixed doesn't have the same set of filters as the judge, increasing the number of potential novel outputs will increase both the set of "creative" (meaningful) results and the nonsensical (incoherent) ones. Since the judges in this case are assumed to be humans, in order for an MS3 to produce more creative outputs without also producing more nonsensical ones, it would need a filtration system which was analogous to a human's (which as of yet we have not been able to achieve).

6.1.4 Context is Hard to Get Right

Minstrel is designed to fill one node of the current story at a time (generally through recall). We found that filling out details of a single node at a time would often fail to build cohesive stories because it can't handle context, generally by failing to correctly infer causality or understand chained actions. As an example, when Minstrel might build a simple node in which Kim buys bread from Billy, the resulting state could be that Kim now owns the bread or Billy doesn't or that Billy has more money but it doesn't know which of these results are contextually important or applicable since it's only looking at a single node. Minstrel has the same failure mode for complex chained actions such as constructing a narrative

that uses multiple steps. If one story has elements A->B->C, Minstrel might be trying to fill in a node in some other story that matches to C but be unable to comprehend that A and B are required for C to make sense.

Sometimes this lack of contextual understanding isn't problematic and creates interesting story fragments. These happy coincidences can't be relied on since they are essentially randomly occurring and a good solution looks the same to Minstrel as an irrational one. In addition to sometimes reducing the cohesion of a story, Minstrel's lack of contextual understanding often prevents it from being able to build complicated structures. Since causality, inference, and sequences of actions are included in almost all day-to-day human storytelling, their absence in Minstrel's stories is unexpected to human readers who then unconsciously add their own details to the story.

When asked about this, Dr. Zhu indicated that Riu has exactly the same flaw and provided this generated story:

> Herman, Julian's father, owned a small motorboat. One night Julian snagged the keys and took the boat from the dock to a nearby island without permission. The boat ran out of gasoline in the middle of the bay. Julian didn't realize his mistake until halfway up bay, and then Julian had to turn around and drive back to the dock.

Notice that the last sentence doesn't make sense. Riu understood that it needed to create some sort of boat trouble to cause the boat to stop and turn around and it found one but since it has no notion of causality, it chose the incorrect one!

Minstrel has some notion of preconditions and postconditions and these can be used to a small degree to help provide contextual cues. Dr. Zhu said in our discussion that she is considering adding similar structures to Riu in an attempt to verify the validity of a generated story. Perhaps further investigation in this direction will yield some good results but our (somewhat rudimentary) explorations

in this area didn't yield any decent results and we were forced to use fancy template construction and highly constrained stories to ensure contextual continuity in Problem Planets.

Dr. Perez-y-Perez has a slightly sunnier outlook on how Mexica and context interact. He stated that the ACAS constant (which decides how similar recalled fragments are) can be tuned to either accept a lot of similarity (presumably bringing the important context with it) or to allow very little similarity (which would likely exhibit the same results as Minstrel and Riu). He believes that there's a range of values of the ACAS constant which may ensure that sufficient context exists for any structure that is imported while still allowing some variation. He also thinks that having thousands of stories in the library to choose from could help Mexica find good matches for some less cohesive setups, situations where a more limited story library would be forced to guess and abstract more.

It is clear that all of the example MS3s wrestle with issues of context and that different avenues are being considered for how to handle the tough problems involved with using contextual clues effectively. The ideas of dramatically increasing the story library or upgrading the system to better handle context (for instance, by annotation of contextual links and using these annotations to guess at the most valuable elements to appear later in the story) are both future work ideas worth pursuing.

6.1.5 No Level of Story Detail is Good

We found that Minstrel's story representation was a limitation in many situations. Sometimes this was a problem of not having the right symbols or a good way to connect two non-standard concepts but in this section I'd like to address the issue of complexity of story fragments. The principle issue here is that good

stories combine fragments which have different levels of detail; a skilled storyteller may skip briefly over the setup of a story and focus heavily on details and the complexity of the plot during the climax. Often times plot twists or other clever patterns in a story rely upon a few details lining up just right. Conversely, a litany of highly detailed trivia is often understood to be boring. For MS3s, details and story complexity are a double edged sword which provide the opportunity for brilliant strokes of creativity or incredibly obvious incoherence.

We have found that working at different levels of detail provides for different feeling output. The stories that we generated with Skald were much more detailed than the tiny Vignettes of Problem Planets, and I feel that the stories that Skald produced were more extremely good or bad because of the details. That being said, since one of the goals of Problem Planets was to create a narrative that made sense as much as possible, the lack of details to cause issues was a boon.

Mexica lies on the Problem Planets side of the details line; it is a plot generator and as such doesn't provide the ability to encode helper objects (in Mexica a man may prepare a meal but he will never explicitly use a knife to do so) and tends towards the generic. At the same time, the lack of details mean that each plot element is far less specific and the plots can be longer and more complex.

Riu on the other hand, generates better analogical matches when stories were described in more detail. The main drawback as pointed out by Dr. Zhu is that the computational complexity is exponential and as story complexity increases, generation times quickly grow to become unreasonable. Interestingly, Riu and Problem Planets both "solve" the problem of how to make large complex stories by creating a handful of independent story chunks and splicing them together. Unfortunately, this method prevents relations between earlier chunks and later chunks of the story.

To my knowledge, no story generation system exists which aims to vary the fragment complexity over the course of a story. The fact that unnecessary details can force the generator into a corner that it doesn't know exists, and can't write its way out of, suggests that a low level of details is the optimal way to create consistent stories. However, most of our examples of "good" stories require a high level of detail to remain interesting to human audiences. There is something to be said for letting the audience infer their own details (something I will be talking about in the conclusion) but I believe that there is no correct setting here; all levels of detail cause just as many problems as they solve.

6.1.6 System Requires an AI Complete Component

We sense that Minstrel has some sort of AI Complete problem lurking under the hood preventing us from being able to guarantee reasonable stories and novelty at the same time. To put it another way, it seems that there's an unsolvable problem preventing MS3s from being able to guarantee novelty and rationality while still giving an author the flexibility of adding new content. If we could employ the cognitive capacities of a teenager as a filter we could weed out nonsensical and inconsistent stories and focus on amplifying the creative potential of the system. Similarly, with a little common-sense reasoning, handling context should be simple since various contextual facts could be ranked as more or less relevant resulting in much more accurate and focused generation.

Dr. Zhu agreed wholeheartedly with the idea that there's some sort of difficult common sense reasoning problem that needs to be solved and it may be AI-complete:

> Absolutely, we think that there are two basic problems that are rather AI-complete, and that Riu somehow "dodges":

- knowledge representation: Riu does NOT contain any semantic representation of what the different keywords used actually mean. You can replace 'go-to' by 'F12345', and Riu's operation would be identical. So, Riu has no way of knowing that a 'mountain' cannot 'walk', for example. And only a very limited way to know that a 'mountain' is more similar to a 'rock' than to a 'human' (since 'mountain' is a subtype of 'inanimate' and 'human' is a subtype of 'animate'). This keeps showing up again and again in the generated stories in one form or another.

- inference: in addition to not having a semantic representation of concepts, Riu does not perform any form of inference (causality is not represented), to determine whether the actions being generate[d] are actually possible. It purely relies on having a sufficiently large repository of memories, to find one that is sufficiently similar to the scene at hand so that this problem does not arise.

When asked about this issue, Dr. Perez-y-Perez pointed out quite accurately that there are many difficult problems lurking but that they're an opportunity to help us to tackle the question, "what is creativity?". Since we're pioneers, he pointed out, we shouldn't be worried about these problems, we should use them to reflect.

Although his answers don't appear to support or contradict the premise that there's a difficult problem revolving around AI-complete reasoning that may prevent MS3s from being fully realized, Mexica clearly suffers from both of the issues that Dr. Zhu pointed out. Although my goals while creating Skald and Problem Planets were more focused on creating a functional system which had a high likelihood of generating "good" stories, upon reflection, perhaps Dr. Perez-y-Perez's goal of further understanding creativity is a more attainable one to strive for while working with MS3s.

Chapter 7

Story Generation

While working on Minstrel Remixed (MR) and Skald, I had three primary goals: build an MS3 that can generate stories, explore whether the system that I developed for static story generation could be expanded such that it can perform interactive storytelling, and determine what lessons from the first two goals can be applied to story generation in general. This chapter uses the first of those goals as a format to discuss story generation. In the next chapter I will discuss the second and third goals. Because many of the goals were purely exploratory, I will discuss what we learned from both good and bad performance of the various systems created in my efforts. As a general structure for this chapter, the first four sections present findings specific to the design and engineering of MS3s while the fifth sections discusses domain and system engineering and the final section summarizes my findings.

7.1 Building an MS3 that can generate stories

The obvious goal for Minstrel — to generate stories — was clearly accomplished by by the Minstrel Remixed team upon the completion of the first version

of Skald. Since there are many alterable qualities in the outputs, after accomplishing the primary goal, our team refined our aims to a series of characteristics that we wanted Problem Planets (P2) and Skald to exhibit. First and foremost (and largely in reaction to the output quality of MR), our new story generators should be able to consistently build "good" stories. Upon further reflection, we decided that we would like our "good" stories to be:

- Long(ish): At least long enough to tell an interesting story. We would like a real plot arc or some simple character development but will accept 5 or 6 actions.
- Coherent: Both internally (story elements should not contradict one another) and externally (story elements shouldn't contradict the reader's knowledge of the story's contextual reality).
- Creative: Stories should be novel with respect to the domain. Copying a pattern of story elements from a few sources and compositing them together is not "creative."
- Complex: There should be relevant details included in a story. Actions should all have feelings associated with them, descriptions of tools used, or some other type of decorator. Ideally these details should be relevant to the outcome of the story but since relevance is very difficult to measure objectively, it won't be part of the formal definition.
- Contextual: Story elements that show up in the story should be referenced later on. Ideally, these recurring elements should generate a cohesive arc or theme and avoid being a sequence of actions performed by one character in disjoint contexts.
- Consolidated: As much as possible, filler should be kept out of stories. Each action should be relevant.

Although Turner's research using Minstrel was an exploration of what the system could produce, we care about the authoring experience as well. Authoring Minstrel's domains is a complex task that takes significant effort and in which it is easy to make mistakes. It is never clear whether an incoherent pattern is a result of poor recall, planning, or bad input data, so making the authoring experience as simple and straightforward as possible speeds up research and clarifies experimental results. Minstrel Remixed was tedious to write domain content for but we made improvements to the authoring process which were described in Chapter 4. An MS3 is a tool and if any tool is more work to use than the extra utility it provides, it would be better to forgo the use of that tool. To that end, we created a set of requirements for our new systems:

- MS3s built after Minstrel Remixed should be able to generate stories in minutes at most.
- The system should take less effort for engineers to build a domain than it would take to manually write a set of stories for that domain.
- Authoring should be straightforward, simple, and predictable.

MR, Skald, and Problem Planets provided a far greater number of insights about the importance of good domain and system engineering than we had anticipated. It certainly felt like the majority of our discoveries were focused on those topics rather than the rest of story generation or interaction. The next few sections will first discuss qualities of stories and story generation and will later cover domain and system engineering.

7.1.1 Coherence and the Reader

Humans are quite good at identifying coherence flaws while MS3s struggle to create coherent stories. Additionally, human authors don't consciously struggle

with coherence when writing their stories, thus when they work with an MS3, they must be careful to consider their choices in light of coherence to a much greater degree than they might otherwise. This lack of deep, common-sense reasoning about coherence makes it difficult for an MS3 to generate stories which don't violate the expectations of human readers. Thus the most difficult and important attribute to pay attention to when working with an MS3 is coherence. Coherence is the first attribute we tried to tinker with while working on MR and something we were still trying to manipulate at the end of P2. There are no known algorithms to calculate a level of coherence for a story nor general processes to automatically detect coherence flaws. The closest we ever came to developing a coherence tool is generating simple filters in our ALPs to detect when a noun from our story was both dead and taking action (and those wouldn't have worked if we'd included resurrection or the undead in our domain). Coherence is the most difficult attribute for computers to quantify and the primary reason that humans will read a story and immediately label it as bad, wrong, or broken before they render any other judgments about it.

In hindsight, the tricky aspect of coherence is understanding that there is only one known way (at the moment) to generally detect coherence flaws and it is part of the subsystem of the human brain known as "common sense." It's possible to systemically avoid some flaws (e.g., don't include the concept of death in your domain and your system can't include animate dead things) or to codify some of the detection algorithms for determining when something doesn't make sense (e.g., detect when an object is both dead and taking action) but there are always a large number of edge cases preventing such algorithms from being perfect (e.g., a person's last will and testament allows them to "take action" after death). Every new concept added to the domain brings multiple caveats, requirements, and

details along with it as potential coherence flaws when it is misused. Since MS3s are designed to create novel combinations of symbols as part of their mutation process, they have the potential to misuse every symbol every time it ends up as part of the output. Thus it rapidly becomes impossible to build a domain of any reasonable size and produce all of the detection and repair algorithms for each symbol's subtleties and common sense requirements. In short, while it is possible to detect and repair the most egregious coherence flaws in an MS3's output, it is currently impossible to do this in a generalized manner without including a human's common sense in the generation or filtration processes.

In my opinion, MS3s will never reach their full potential until common sense can somehow be built into the system. Fundamentally, they are built to explore all of the nooks available in a story space and the further afield they search, the harder it is to keep inconsistencies out of the output. Despite the fact that MS3s have this seemingly fundamental flaw, the goal of building these systems is to explore what they can do and what we can learn from them, not to figure out how to produce 100% consistent stories.

Despite the fact that coherence flaws can't be systemically eliminated by brute force, there are a number of things one can do to evade them or employ a reader's common sense in order to help build the story and minimize inconsistencies. A few of the more effective techniques we've found are:

- Reduce detail.
- Reduce the use and/or complexity of complex story elements.
- Choose a domain which is more tolerant of inconsistency.
- Change story presentation.

As was discussed in 6.1.5, one simple way to optimize for coherence is demonstrated wonderfully by Mexica which doesn't represent objects or much of the

detailed mechanical state one might find in a video game. Actors can fight other actors, have important discussions, or even save the princess, but the saved state is all about relationships between characters and not about their injuries, location, or whether they've acquired some magic sword. As a result, Mexica's stories are a series of plot points which guide a reader's imagination through the steps of the story without trying to do anything fancy with concrete details. Coherence flaws often crop up around details that a system doesn't have guidance for and can't understand, so removing as many details as possible is a strategy that is highly successful and should not be overlooked.

Mexica's successful strategy could be seen as attempting to maximize the involvement of the reader's ability to infer details without them being present. Put another way, it attempts to globally reduce detail in order to maximize inference and minimize any conflicting information from the story. For stories that require a bit more fidelity, it is helpful to attempt a lesser form of this: exclude fragments which tend to cause conflicts between the story's facts and a reader's inferences. In our experience, the largest source of these types of inconsistencies are complex story elements. Because complex elements are tied together with one or more inferences which are not known to the system it is impossible for the MS3 to accurately reason about the implied inferences. This makes it easy to introduce new elements which break the reader's expectations. An example of a story containing such implications is the story from 2.1.3, (Thief steals the statue from Victim. Policeman arrests Thief. Policeman returns the statue to Victim). In this example, the second sentence makes much more sense if the first is present. It is impossible and undesirable to completely remove these implications from an output story but, where possible, it is useful to identify stories from the library which employ implicit logic and rewrite them to be explicit.

The third storytelling strategy that we've found to work well is to reduce the expected coherence of the domain. In a political drama, everything has to make sense and even minor details which conflict can draw a reader's attention. A short hop away is the conspiracy drama genre. The context of these stori es is based on conflicting information being presented to the actors and thus some degree of internal inconsistency is easily tolerated. Similarly, a story about robots can get away with actions, beliefs, or behaviors which would not be accepted by most readers when transposed onto characters in a fantasy domain. Although we didn't attempt it, I believe that a dream domain would provide even more tolerance for inconsistency than any of the domains that I've mentioned.

Presentation is an important element of story generation but was never our focus. Despite this, while working on P2, we noticed that presenting a string of facts yielded a much higher tolerance for incoherence than when those facts were 'translated' into a paragraph of prose by Skald. This is largely due to the fact that some statements end up being contextually interesting to a human reader even if the way they were fit into the story by the MS3 does not make sense. As an example, here is a vignette from P2:

> THE HEGEMONY WANTS TO TRADE WITH ROBOT_ZED.
> THE FEDERATION ARE AT WAR WITH THE HEGEMONY.
> ROBOT_ZED ARRIVES.
> THE FEDERATION ATTACKS THE HEGEMONY.
> THE HEGEMONY IS AFRAID OF THE FEDERATION.
> THE HEGEMONY CREATES RADIOACTIVE GERBILS.

This vignette creates an interesting cliffhanger: what will the Hegemony do with their radioactive gerbils to deal with the Federation? If this is rendered using Skald's prose, we learn that there is a logical flaw in the story because the motivation for the Hegemony creating the radioactive gerbils is to thwart its own desire to trade with Robot Zed. Even in the 'correct' version of the story where

the creation of the gerbils was motivated by a desire to trade with Robot Zed, it is still less interesting than the implied interpretation that the gerbils are created by the Hegemony in response to their fear of the Federation.

It is important to know how well one's system can convert its data representation into human readable text and find a manner of presenting the output to readers that puts them into a more accepting mindset. As an example, when we realized that Skald's English generation subsystems weren't particularly great, we decided to make P2 look and operate like a text adventure from the 1980s and 90s. This presentation seemed to make users far more forgiving of the stilted language and somewhat clunky stories that were produced.

In summary, coherence is not well understood but is clearly among the most important attributes of the output of an MS3. There are some methods that can be employed to lower the appearance of incoherent elements or reduce their impact but the only way to effectively remove coherence flaws is through human level common sense which is the primary impediment stopping MS3s from being able to create consistently high quality stories and thus being useful for more than research purposes.

7.1.2 Story Qualities: Length and the 5 'C's

Every story generator has a point in an abstract space defining story quality that is an average of all of its output. For my purposes here, I care about a hypothetical six dimensional story quality space: Length, Coherence, Creativity, Complexity, Contextuality, and Consolidation. I defined these attributes at length in the beginning of this chapter but in brief, length is the number of nodes in a story, coherence is how well the story fits together (both internally and in the context of a reader's expectations), creativity is how much novelty the story

includes, contextuality is how much the story introduces and then uses context, and consolidation is the idea that every piece of a story should serve the narrative in some way. For the rest of this discussion I will refer to the six dimensional space defining story quality in relation to these traits as LC5-Space. Since many of the axes of LC5-Space are concepts for which there is no existing scoring metric, this concept is merely used in the abstract to further discussion. That all being said, all story generators have a point in LC5-Space that represents the average scores of all possible outputs weighted by their likelihood. In MS3s, this point, the LC5-Average, is governed by a combination of the subsystems that are active, their tuning, and the domain used for story generation. Much of the work that goes into an MS3 is spent trying to get the system's LC5-Average to a place that fits the project's goals.

As an example, in one of our papers [Tearse et al., 2011b] we present findings about how tuning the boredom mechanic in MR could change the ratio of sense and nonsense in the output of the system. This is an example of the balance between the power of the system to generate creative output and its ability to maintain consistency. To describe this work with regards to the LC5-Average for MR, we were exploring how the boredom subsystem can be tuned in order to shift the LC5-Average for the system and finding that the creativity and coherence variables were inversely related.

To describe this in more general terms, tweaking a system's LC5-Space coordinate is accomplished by modifying its capacity to generate new content or filter existing content. FIVE is a perfect toy system to demonstrate such activity: In FIVE one can easily increase the length of stories by incrementing the number of sentences generated. Similarly, the "creativity" can be augmented by enlarging the size of the noun/verb lists. These tweaks will clearly increase the system's

generative power alongside its LC5-Average length and creativity scores. By the same token, a new filter can be added to remove some incoherent stories from the output pool and thus increasing the average coherence score of the system through augmenting the power of the system's filters.

While working on MR, Skald, and P2, it was incredibly satisfying to make a change to successfully move a system's LC5-Average in a desired direction. At the same time, one of the most frustrating findings for the whole project was that our systems seemed to have an "output power" that described some curve in LC5-Space above which the system could not go. Instead, we could tune the system to spend the output power differently but very few of our changes (if any) were able to actually increase the output power of the system. Thus, every change we made to increase some score also seemed to decrease some other score. This is exhibited in the above example with FIVE. Increasing the generative power of the system by increasing story length will add extra events to the story. Since these events are randomly generated and each one has some implied contextual connection with all previous events (at least in the mind of humans), increasing the length of these stories will also decrease the coherence.

While there are many individual points to be made about the qualities of the stories generated by MS3s, throughout this chapter, I will routinely return to the concepts of **LC5-Space**, **LC5-Average**, **output power**, **generative power** (how many different stories the generative components of the system can create), and **filtering power** (how many different stories the filtering components of the system can weed out) of the system as a whole under the various modifications and tunings that I will be discussing. These concepts provide generalized ways to evaluate and discuss the system-wide effect of specific changes and their intended (or otherwise) impacts.

Attribute	Machine Measurable	Directly Manipulable	Inversely Influences	Directly Influences
Length	Yes	Yes	Coherence, Consolidation	
Complexity	Yes	Yes	Coherence, Contextual	Length
Contextual	Estimate	Yes	Coherence	
Creativity	Estimate	Poorly	Coherence	
Consolidation	No	Poorly	Creativity, Length	Coherence
Coherence	No	No	?	?

Table 7.2: Attributes and interrelations of LC5 variables.

7.1.3 Measurability, Manipulability, and Influences

As I hinted while defining output power, nearly every alteration to MR, Skald, and P2 managed to improve some of the LC5 variables at the expense of deteriorating others. Most variables inversely influence other variables but some have direct relationships as well. Table 7.2 describes these relationships, whether an attribute could be measurable by machine, and whether it can be directly manipulated by altering the story library or components of the MS3.

While it is possible to pass stories through a handful of human judges to get some sort of ordinal rating system, I believe that machine measurability of all of these attribute is critical to be able to tune MS3s and truly understand what effects a component changes might bring about. Such a system could also be used as a filter to throw out stories that do not pass some criteria. Length is clearly measurable by counting the number of story elements in the graph. Similarly, complexity (how locally interconnected the story elements are) can be roughly measured by getting the average number of connections for each element of the

story. These may not work precisely for all MS3s but some analogous method should be available for any graph based storytelling system.

Measuring creativity and the degree to which stories maintain and use context (how globally interconnected story elements are), are not as easily accomplished as length or complexity. Creativity is difficult to measure since most scholars [Turner, 1994, Sharples, 1999, Ritchie, 2001, Colton et al., 2011, Boden, 2004] include some form of "humans say so," "utility," or what I'm calling coherence, but it is possible to make rough estimates of the novelty of story elements or patterns by looking at how many of those elements or patterns don't exist as-is in the provided domain. This could be augmented by searching for elements and their neighbors to incorporate some of the extra meaning implied by complex story elements. Like creativity, it's possible to estimate the contextual score of a story by counting the number of reused story elements, causal links between non-adjacent parts of the story, or similar methods that work with the particular story representation used for a given system. These measurements will not take into account the quality of the use of the contextual elements, nor will this approach work well with common objects or states such as having money, being happy, etc. These confounds mean that accurate measurements of these two attributes aren't possible but the methods just described could be used to create rough estimates which are still valuable.

I have already argued that coherence isn't something that is measurable by a machine and I believe the sixth of the LC5 attributes, consolidation, is similarly impossible to measure. It may be possible to procedurally determine what parts of the story are important and what parts might be removed without sacrificing meaning but this seems to be an AI-complete task as well. While automated measuring of the other 4 attributes should certainly be implemented in order to

efficiently tune and tweak future systems, these final two attributes may require humans to read tens to hundreds of output stories in order to truly get an idea of where a particular system stands.

A second particularly interesting characteristic of the LC5 Variables is whether the attribute is directly manipulable or not. Length is once again a particularly simple demonstration: by changing the overall length of the story templates or the desired number of nodes, an author or system designer can directly change the length of the output stories. Complexity and Context are similarly directly manipulable by changing the number of connections between story elements. If this is done in general it will alter complexity while if it is done to intentionally increase or decrease mentions of a particular element at different temporal points in the story, it should alter contextuality as well as complexity. This could be accomplished coincidentally or enforced in other ways. For example, in P2 the Vignette Generator intentionally reuses entities across scenes.

Creativity and consolidation are only manipulable in very broad ways. We manipulated the creativity of MR and its successors through Turner's boredom mechanic but this drove stories to become more random, which increases both creative and nonsensical stories so we did not find it to be a reliable method. Similarly, since consolidation is not machine-measurable, it is hard to manipulate with finesse but for MS3s with templates, it's possible to alter them to make outputs more terse or long-winded. Also, the story library can be altered to provide stories where details are more or less packed in. While these methods work, they aren't truly manipulating consolidation, but are rather hoping to do so while altering related attributes like length and complexity. Once again, coherence has been discussed before and is impossible to directly manipulate.

The final and perhaps most important factor for a discussion of trade-offs is

the way that the various LC5 Variables interact with one another. First off, pretty much everything inversely influences coherence. Coherence flaws have a chance of popping up with every new element in a story as well as every interaction between elements which means that increasing length, complexity, or contextuality will reduce coherence. In addition to these three effects, coherence will also drop as creativity goes up since that means that more patterns will show up in outputs that aren't in our (assumed to be) completely coherent story library. It stands to reason that using random combinations of elements will cause coherence flaws. Consolidation is the only attribute with a positive correlation with coherence for as a story becomes shorter and containing only important details, it should be more likely to be coherent.

Speaking briefly about the remaining five attributes, given a story, increasing its length should decrease its consolidation as the added length will make it harder to keep everything tight. As complexity goes up, it is likely that output length will need to go up as well to describe the new complexity. Along with this, the contextuality should drop since it is unlikely that the added complexity will focus on the contextual elements and thus make more of the story about non-contextual elements.

7.1.4 Challenges and Trade-offs

With the various LC5 variables defined and their interactions described, it becomes a useful exercise to explore the trade-offs that we found while working on our MS3s. Additionally, I will describe some of the tricks we found to get around some of these trade-offs and will reframe some of the discussion in the creativity section in this light.

Length, Complexity, Context, and Coherence A clear theme has been that length, complexity, and use of context all reduce coherence. We could make stories longer but they tend to either be simplistic or have a high rate of incoherence. Similarly, we were able to make more complex stories but they were either short or likely to contain coherence flaws. In short, building a long, complex, and interwoven story is difficult to do correctly. This aligns nicely with our expectations as human authors and unsurprisingly, has been solved in film, novels, and video games in the form of a serialized story. Instead of creating a single long and complex story, these works are comprised of a number of small stories that connect one after the other with perhaps some overlying contextual element woven in at choice locations. P2 was our proof of concept attempt at this technique and the addition of the vignette generator along with a few custom tailored templates managed a passable short serialized story. We only strung together 3 or 4 micro-stories for P2 but there's no reason we couldn't have done a dozen or expanded the stories to contain more elements.

Context Assuming that one can maintain the coherence of output stories at a reasonable level, I think that one of the biggest challenges that MS3s face is their ability to make stories with more use of contextual elements. Any story that fits into a half-dozen lines is too short to demonstrate this ability – everything in the story is both the focus of the story as well as the context. When we tried to make Skald and P2 produce longer stories with more contextual elements, our naive approach was to write templates with repeated uses of the same noun placeholders, ensuring that 'Person A' would do a series of actions, some of them using the same 'Thing B' as a tool. This works quite well but we soon came to the conclusion that this form of contextual usage was not our system's accomplishment but rather our own and that we could no more praise Skald for using context well

than we could for it using good names (all of which we had also provided).

When we tried to simplify our templates and have Skald produce more of the story structure on its own, we found that the result was fairly aimless and that stories making good use of repeated contextual elements were rare and randomly occurring. This intuitively makes sense since we had created no subsystem or algorithm to help the system know what past context is important to any given choice. Accepting only stories that have repeated elements that we didn't expressly put there via a template will mean that the random re-use of elements should lead to more coherence flaws. These flaws will be centered around the misuse of those elements. Such a filter also yields too many stories that re-use unimportant or uninteresting elements and thus don't feel like there's anything contextual going on. To make matters worse, recall is weighted by proximity to the currently active node so this subsystem essentially ignores or 'forgets' elements of the story that are far away from the piece that is being filled in. This is especially true in episodic stories where Skald and other MS3s should start each episode with a blank memory. Thus without some sort of intelligent selection of elements to showcase for a second or third time later in the story, less context makes for more coherent and superior stories (to me at least).

I believe that the problem has two intertwined causes: current MS3s don't have a particular goal in mind for a particular stretch of story, and they don't have any good structures for indicating that an individual element should be reused later. We created rudimentary systems for this in P2 with each vignette having a cause, effect, and a few contextual nodes that were carried over from the previous vignette but a better system needs to be put in place if an MS3 is going to handle context well. I could see a simple version of this maintaining a list of annotated pointers to particular nodes in the story graph. If this list were properly

curated, it's likely that it could be used to tie relevant contextual elements back into the story without being random. Even if this system was not particularly smart about its selections, I believe that humans enjoy trying to figure out what a story is about, where it's going, and how everything fits together. So as long as a rough algorithm were doing something in a manner that was clearly not random, it would likely be enough to greatly enhance a user's perception of the system and its output.

Consolidation Consolidation is the LC5 variable that we investigated the least. Minstrel Remixed and Skald both tend to produce and incorporate a lot of story fragments that aren't particularly relevant to the story. In contrast, an MS3 producing consolidated stories should yield output that doesn't have any clearly identifiable uninteresting or unimportant segments. We figured that if interesting or useful fragments can be identified, removing the rest in some sort of intelligent way should yield "better" stories. The trouble here being this variable is far more subjective than any of the others and we did not develop a good way to measure it. We did determine that attempting to simply cut down templates to improve consolidation led to mad-libs style stories which were uninteresting and reduced the number of unique output stories dramatically.

Creativity: Turning up the heat Creativity is what Turner's Minstrel was perhaps known best for. The system could generate a few novel stories that were a little stilted but were quite impressive outputs nonetheless. Our original reason to start a project around Minstrel was to explore the creative potential of this landmark system. With that in mind, it should come as no surprise that our initial investigations all revolved around the creative subsystems and whether there were, as we suspected, ways to manipulate the amount of creativity employed

in story construction. As has been mentioned before, we first investigated the creativity of the TRAM system [Tearse et al., 2011b]. Since creativity is arguably demonstrated by creating novel and useful products, we found variables which could control the coherence of our output (as an approximation for utility) and the number of different possible outputs (as an approximation for novelty), although these were almost always inversely related.

This experiment was performed with Minstrel Remixed prior to the completion of the ALP system, much of the story library, or anything developed for Problem Planets. We created a partially filled out Goal-Act-State trio in which an unknown action causes a princess to die. This was used as a query for TRAM recall 1000 times. We then de-duplicated the thousand results into a set of unique results and then read each one, judging whether it was sensible (i.e. coherent) or nonsensical. This process provides us with a ratio of the sensibility of all possible answers. By applying each unique result's sensibility judgment to all of its duplicates, we can also provide an estimate of the chance of getting a sensible result if a single TRAM recall were requested. Our base case had a depth limit of 5 (the number of TRAMs that can be applied before a particular branch of the search tree is disqualified), used 7 TRAMs focused on Act nodes (GeneralizeConstraint, GenrelizeActor, GeneralizeRole, LimitedRecall, RecallActs, IntentionSwitch, SimilarOutcomes), 16 stories in the library, and uniform TRAM selection weights (e.g. random selection). Our experimental variables were to change the depth limit from 2 to 6 (denoted by D2 through D6), use an alternate query (in which a troll wants to change the health of something), use only 8 stories from the library, remove the two TRAMs which create the most broad changes (SimilarOutcomes and GeneralizedConstraint), and switch the TRAM selection method to use the weighted values described in Chapter 4.

The results of the control as well as each condition are summarized in Figures 7.1 and 7.2. These clearly demonstrate that we can control both the amount of uniqueness and coherence of TRAM results. Unfortunately, these results also show that the conditions with better total coherence ratios also have far fewer unique results. There were a number of other interesting things to note:

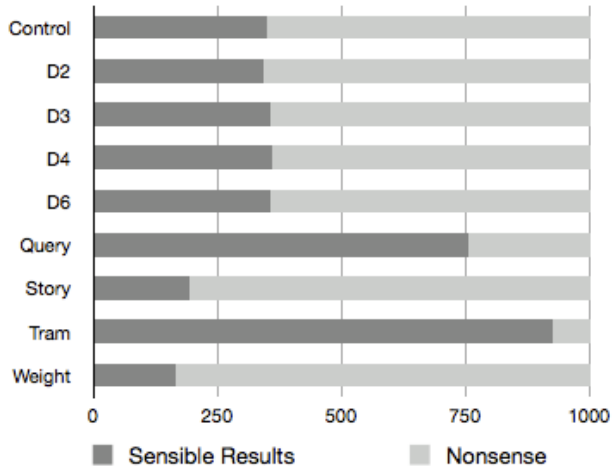

Figure 7.1: TRAM Experimentation: Sense to nonsense ratios of 1000 recalls in each of 9 experimental conditions.

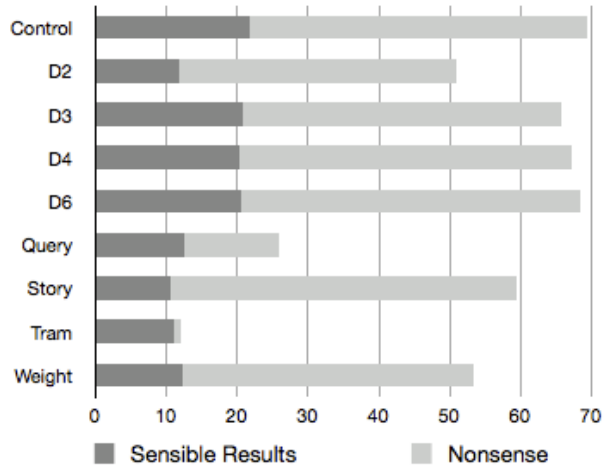

Figure 7.2: TRAM Experimentation: Sense to nonsense ratios after removing duplicate results.

- In 1000 recalls, there was a lot of duplication. The largest number of unique

story fragments was 69.

- Working with a reduced story library suppressed both coherence and unique recall results.

- Comparing the alternate query case to the control, it is clear that the initial query can create a lot of variance in results.

- TRAM choice is clearly important: when removing the most general TRAMs, we had a dramatic increase in coherence and a similar decrease in number of unique results.

- TRAM search depth doesn't seem to have much effect on coherence or uniqueness.

- Across the majority of our conditions, the unique result coherence ratios (averaging 30.55%) were far worse than the total result ratios (43.9%) indicating that the more common answers tend to be more coherent.

- Trying to understand why all the depth settings had similar results, we discovered that the vast majority of results were provided after only one or two TRAM applications. In Figures 7.3 and 7.4 we show how many TRAM applications were required to find results in the TRAM depth limit of 6 (Depth) case and Weighted TRAM Selection (TRAM Weight) case.

In addition to this exploration of the TRAM system, we also looked into the boredom mechanic built into Minstrel Remixed [Tearse et al., 2012]. This subsystem is simple: reduce a story fragment's chance of being looked up by recall after it has been used. Turner's version permanently reduced lookup chance to 0% after use but we found that this "turned up the heat" too much and after a few recalls had exhausted the most coherent patterns; after this our TRAM system

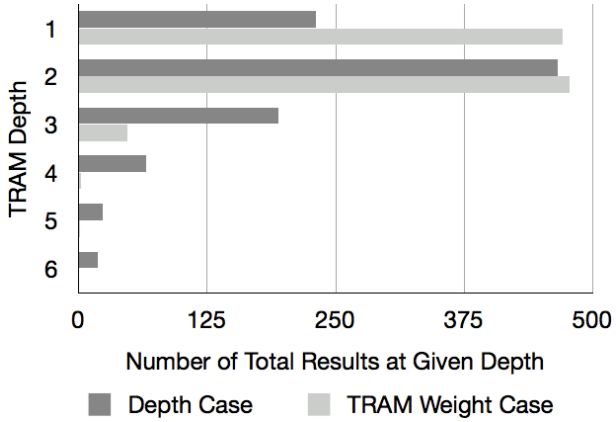

Figure 7.3: TRAM Experimentation: Based on 1000 recalls in Depth-6 and Weight trials, the number of additional results added at various TRAM depths.

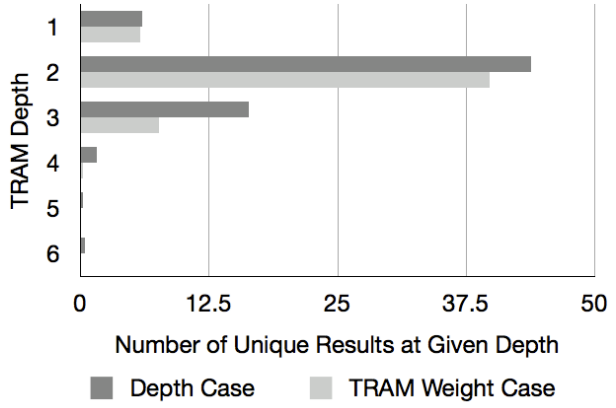

Figure 7.4: TRAM Experimentation: After removing duplicates, the number of additional results added at various TRAM depths.

would return nothing but incoherent results. We found that when attempting to produce more stories, it was beneficial to utilize a modified boredom system which would increase the lookup chance on specific results every time a TRAM recall was performed. This refreshed the available options over time and resulted in a wider array of fragments being used in TRAM recall than when the boredom system is turned off and more coherence than when it is set to permanently disqualify fragments and end up effectively making random choices.

We found the same issue in P2's Vignette Generator — and FIVE can even be modified slightly to display the same behavior. Time and again we found that if we built filters to constrain the generative systems, those filters could easily be tuned to reject all useful generated outputs and either end up in an endless loop (P2), breaking a rule in an effort to produce something (FIVE), or in the case of MR, demonstrating that the fragment transformation subroutines are subject to the old truism, "Garbage In, Garbage Out." In all of these cases, we found that there was a breaking point at which the systems will produce random outputs with high likelihood of incoherence. To put it another way, we learned that we could use the "creativity knobs" of our systems to increase the generative potential of our system but that this always lead to higher likelihood of coherence flaws which curbed their utility at some point. Instead, the best we could do was to tune the filters in an attempt to reject as many uninteresting stories as possible while not being overzealous and effectively eliminating the usefulness of critical subsystems.

Increasing Output Power The past handful of sections have been broadly about how the generative power of an MS3 seems to be fixed. We can tune each LC5 variable but at some point there is a limit after which further tuning takes place at a cost to another variable (generally sacrifices are made to coherence). Despite the fact that the LC5 variables seem to be part of a zero-sum system, we

have identified two ways to increase the generative power without incurring the trade offs mentioned above: to thoughtfully increase the size of the domain or to add new subsystems.

Increasing the size of the story domain is the most obvious of these two methods. If one wants more variation in one's stories, it is logical to assume that adding more variation into the fragments that are drawn upon to create the stories will result in more variation in the outputs. As has been discussed in section 5.2.4, this must be done carefully or else one risks adding complex or redundant story elements into the mix which will ultimately lead to unsustainable forks in the meanings or usage of symbols which will reduce the power gains made by adding them in the first place. By paying attention to carefully crafting the content of the story domain however, it is ultimately possible to increase the output power of the MS3. We demonstrated this very clearly a number of times in P2 by deciding we wanted to enable a new concept in a story, building a handful of stories for the library crafted around that concept, and then observing that the output would sometimes contain the new elements.

While adding more story content to the domain is a good way to increase generative power, another good way to increase the generative power is to add new ways to build or reason about stories. We did this most clearly by adding new subsystems to P2 such as the new vignettes, contextual nodes, and English rendering. By adding in branching serial vignettes connected by goals, we were able to keep the number of symbols in any given story down to a minimum and essentially eliminated coherence flaws. At the same time, we added in global nodes that shared important contextual elements between vignettes which allowed us to improve the use of context. The overhaul to our English rendering system allowed us to maintain expressiveness while reducing the actual size of our domain, thus

improving the coherence as well. These sorts of improvements can be made to Skald and the Minstrels as well by adding new ways for those systems to interact with their stories while they're being built. For example, the Generalize Object and Generalize Role TRAMs introduce the idea of nouns fitting into an ontology and having different levels of similarity between one another. By introducing these TRAMs, Skald is given a new way to reason about some of its story fragments and in so doing, avoid some coherence flaws and increase creativity by allowing some new fragments to be generated that otherwise would not be. Similarly, some Author Level Plans (ALPs) change the way that Skald builds stories and in so doing, can add context, creativity, and length. One example of such an ALP detects characters without an origin story and adds a handful of nodes to explain the background of those characters. It should be noted that while these improvements are good for the generative power of the MS3, they also come at the cost of adding complexity to the system which makes it more of a challenge to analyze whenever new issues or discoveries arise.

7.1.5 Domain and System Engineering

Having discussed the characteristics of our MS3 stories and how they can be manipulated, I will now focus the discussion upon domain and system engineering. As has been covered in the previous discussion, how the domain and system are designed, built, and used are tremendously important to the functioning of the MS3 as a whole. I will discuss three primary areas of domain engineering for MS3s: manipulating the reader's expectations, effectively working within the system provided, and being thoughtful when making domain choices.

After the discussion of domain engineering, I will move on to system engineering which has a number of topics to cover: English rendering, designing data

structures, story structure, authorial affordances, authorial leverage, and why making intentional improvement is so difficult with MS3s.

Domain Engineering: Character and Genre, Manipulating Expectations Arguably the most powerful design consideration for an MS3 is its domain. Authors must work very carefully to craft this component and the single most powerful choice that we made when crafting multiple domains was the selection of genre and tone. A well chosen domain and interface can disguise flaws in an MS3, or even turn those flaws into advantages, while a poorly chosen domain can expose the stories to extra scrutiny from readers. Choosing a genre that is designed to be confusing (such as a political drama) will certainly confuse whatever rudimentary story understanding or analogy systems are embedded within the MS3. On the other hand, choosing a genre which is too abstract (e.g., a dreamscape) tends to yield stories which either contain too much discontinuity for readers to follow or are so abstract as to be internally conflicting or to clash too much with the reader's sense of reality. Of the multiple domains we created, by far our most successful were a science fiction domain and a conspiracy theory domain. The success of these two domains lies in the tropes of their genres and in the habits of their readers. Both genres demand a high degree of suspension of disbelief from their readers who are more willing to accept cognitive leaps or omissions in the story simply because bizarre or extremely unlikely events are commonplace in these styles of fiction. Additionally, readers of science fiction or conspiracy theory novels are used to puzzling out a coherent storyline despite inaccurate or missing information. When put together, these two attributes support much more complex stories simply because they need less internal cohesion to satisfy readers.

In a similar vein, character choice is another important way to reduce the re-

quirements upon the logic and reasoning of an MS3. By choosing characters which aren't human and aren't anthropomorphizable, many otherwise unacceptable behaviors, beliefs, actions, and attributes become acceptable. Problem Planets, which is discussed in more detail in Chapter 5, used robots to great effect. This was possible because strange behavior based on some logic that is inappropriate for a given situation (from a human perspective) is acceptable and even desirable in order to better exemplify that character's robotic strangeness. Riu also selected a robot to be the main character of its debut stories and, while robots are the only non-anthropomorphic examples that we have created stories around, aliens, mad-scientists, young children, and other entities which act irrationally (from the point of view of a normal adult human reader), will all provide an extra measure of protection against a reader's sense of disbelief.

Domain Engineering: Generic Symbols and Fragments, Working with the System When building a domain, it is critical that authors and engineers keep in mind the way that their MS3's generation and filtration systems work. It is extremely easy to view the construction of a domain as orthogonal to the functioning of the MS3 as a whole, but working with this inaccurate viewpoint forced us to completely scrap half of Skald's stories and a few of P2's small prototype domains since they caused problems when the TRAMs and ALPs tried to work with them. The most important lesson that we learned from this was that each symbol should be appear as often as possible in the domain since each unique instance gives the MS3 more acceptable patterns to use the symbol in. This leads to more reuse of existing "good" usage patterns and less random guesswork on the system's part, which should improve coherence at the expense of a little creativity.

One easy way to improve a domain's symbol density is to try to make the domain as generic as possible. MS3s aren't likely to understand the nuances

and details associated with specialized objects, places, and people so the more specialized the symbols are that you put into the domain, the more unregulated implied meaning you're injecting into stories. To put it another way, naming symbols after non-generic familiar concepts (e.g., using 'berate', 'argue', or 'praise' instead of 'talk') invites cognitive dissonance. This means story elements that are more boring are far more likely to yield coherent stories. As an example, the symbols in the phrase, "A politician orders more trade goods for his town," fits in many more contexts than a more interesting version: "Senator Jim ordered 200 crates of Superman Comics for his town." Verbs should also be as generic as possible to avoid excess implications, but they should also describe a whole scene if possible. Although it's possible to describe an entire action sequence in which a hero leaps, punches, and wittily insults the villain, the only important aspects of that from the system's perspective are the primary action and its consequences. All other details are interesting for human readers but likely to cause unnecessarily complex story elements which are hard for the system to work with.

Following this advice yields a telltale stiltedness to an MS3's story graphs which, if not creatively rendered into language, will make logical sense but will appear far more generic than most human writers would produce. In essence, domains that include less nuance may be better for general story generation simply because the system optimizes "acceptable" coherent stories at the expense of the much more rare stories which make good use of clever subtext and connotations.

A related practice to increasing a domain's symbol density is to think about what you as a human would like the domain to contain at the same time that you conscientiously consider how each new symbol, fragment, and concept will compose with other domain elements and your system's generation and filtration components. As an example of this, in section 5.2.4 I discussed reworking P2's

domain to include predetermined conventions rather than what were effectively a random assortment of symbols and patterns. This refactoring was a huge improvement because the new conventions allowed us to take different human concepts which function in a similar fashion and make them symbolically identical (or incredibly similar) so the system can pool its knowledge about these concepts, thus improving its ability to successfully manipulate and predict outcomes when utilizing the concepts. By carefully considering how and why symbols and patterns are being used, it should be possible to find other ways to transfer knowledge about one pattern on to other similar patterns.

As a final note, we did try to generate a domain that was more detailed than the as-generic-as-possible type of domain that I have been championing in this section. Our results were promising but the time to results ratio appeared to be exponentially rising as we progressed so we decided to simplified it into P2's existing robots domain. It is possible that a complex domain could be made to work given enough time, looser coherence requirements, or the implementation of a powerful story validation system. In our opinion, none of these strategies or alterations were likely work for our goals and timeline for P2.

System Engineering: Rendering Text in English Assuming a story has been created and encoded as some sort of machine-readable data structure, translating that structure into text is a potentially complex process which has direct and dramatic consequences on the qualities of the story as it is perceived by the reader. By making certain presentation choices, inconsistencies can be smoothed over, work can be silently offloaded onto the reader, and the system as a whole can be made to read as more or less robotic. That being said, choices about the presentation can also introduce new incoherence into a story, hide clever patterns that were generated, or cause other deleterious side-effects.

The original Minstrel was created with a presentation layer already in place; Turner's lab had another project in it called RHAPSODY which handles the English rendering. Unfortunately, this system is not available, so we created our own extremely minimal text-generation system without giving it much thought. We used the same system for Skald and only while working on Problem Planets did we come to realize that the presentation of the stories could be consciously manipulated to fix or minimize inconsistencies and other issues. I have previously brought up the story of Smaug in section 2.1.6, the dragon who killed a knight by serving him a poisoned roasted king. In this story, if the detail of the king were removed by the presentation and instead, Smaug merely served up poisoned food, the story would be consistent with reader's expectations of the world and viewed much more favorably. Alternatively, the point could be made without the detail at all: Smaug could merely have been said to have poisoned the knight.

The two reworks of the Smaug story are representative of two separate philosophies that appear in MS3s. The latter strategy, removing details entirely, offloads a lot of the contextual and causal generation burden to the minds of the readers. The system doesn't try to answer questions such as, "what was the food?" or "how did the dragon get it?" and instead focuses on generating a good plot. This is the strategy that Mexica employs and that we started to veer towards with Problem Planets. Much of the incoherence that Minstrel, Skald, and Riu generate lie in their inability to understand all of the real world implications of the supporting details that they include in their stories. When we altered Problem Planets to remove the majority of its details, the number of inconsistent stories dropped dramatically but at the same time, most of its stories felt like incomplete sketches that were just waiting for some juicy detail to be injected.

Whether or not details are present in renderings, readers expect a natural

language version of any story that is presented to them. Simply rendering story fragments individually (without trying to link them together), is easy to do and does not risk reducing the coherence of a story. After all, putting together a dozen different sentence templates and creating a small algorithm to decide which template to use for a given story fragment is a rather simple task. The resulting story is passable but the lack of any causal links to guide a reader from one statement to another makes the story stilted.

> Jim owned the rapier. Jim attacked the troll using the rapier. The troll died. Jim is happy.

That being said, text generated straight from machine-readable data structures is boring and repetitive, and it is very difficult to get the context right. When generating human-readable stories, its important to present actions and state changes. Aside from state changes, there's a broad continuum of how much other stuff is presented by a system. Broadly speaking, the "other stuff" comprises contextual information such as supporting details and causal links and often times carefully managing one's place on this continuum is a major factor in the quality of the telling of a story (not to the confused with the quality of its content).

At the end of the continuum where all context (e.g., the entire state of the world) is provided as a contextual reference for each action and state change, stories become extremely boring and repetitive. Fortunately, I don't know of any story generation systems which provide such info dumps outside of some debug mode. On the other side of the continuum is the text-rendering system which provides no details, no context, and no indication of causation. This is essentially the point on the continuum where FIVE and Mexica lie. Neither of these systems attempt to provide details, context, or causality; these elements are instead left for the reader to infer from a list of events and state changes. Slightly up the scale lie Skald and Minstrel which attempt to provide *some* contextual detail for

their stories. Further up the scale is Problem Planets and then Riu, both of which employ much more complex text-generation algorithms.

Humans will imagine a surprisingly large amount of detail provided even just the sketch of a story. That being the case, the more details that are included in a story, the more constraints are placed upon the story that is forming in the mind of the reader. Sometimes, the details presented in a narrative are incompatible with the imagined narrative of the reader. These conflicts tend to be viewed in a wide array of ways: some may be seen as humorous or clever while others may be viewed as strange or even inconsistent. Taking this into account, systems that provide few details like FIVE and Mexica are offloading the maximum amount of context and causality generation work on their readers. Since this strategy also presents minimal details to the reader, it is also much harder to find inconsistencies in its output. Having played with a number of variations on FIVE, I can report that it generally feels like its stories skipped a step rather than presented a story with inherent inconsistency.

While the 'remove complicating details' philosophy seemed to be a cleaner path to enhance story coherence for Problem Planets, I would like to acknowledge that the first strategy I mentioned for reworking the Smaug story is perhaps more personally interesting to me. This method aims to keep in details that aren't as critical to the overall plot by hiding, abstracting, or transforming those that it believes will be hard for users to comprehend or likely to cause incoherence. This method is technically challenging and would be a completely separate subsystem of a story generation system but I believe there is are interesting opportunities in this area and will discuss them in further detail in the Future Work section.

System Engineering: Designing Data Structures to Help the System A large flaw in the design of Minstrel and its descendants is the fact that most of what

the system 'knows' about a story is represented in the story itself. When a TRAM decides that hermits and princesses are similar enough to do similar actions, this new knowledge is only represented by a new action showing up for the character. Similarly, when an ALP decides that further detail is needed somewhere, it adds some new story nodes to be filled in. In both of these situations, the story is altered but the fact that some small piece of intelligence decided that a given process should be executed is lost, generally to the detriment of the system's functioning and the resulting output.

Another way of looking at this is through the analogy of building a house. A questionable architecture firm draws up some fairly abstract plans and hands them off to a team of people whose job is to actually build the house. Most of the time the builders spend filling out all the details, but they also amend the plans when they run into ambiguity or details that clash with their professional opinion. If these builders don't communicate with one another, amendments to the original plan aren't coordinated. In this environment, one builder might construct the enclosure for an attached apartment, hand the space off to another worker to fill, and the result is an addition containing a single huge dining room.

An easy way to fix both the building and the story generation scenarios is to annotate the plans so that the reason for modifications are clear and further work can be coordinated. Any such annotations should remain hidden in the plans but should be able to affect change in the output. One such strategy that we found to work well to support more complex stories was to select a few specific details to be the focus of the story along with the main character. Special annotations were written into the story and the Vignette Generator was designed to track these chosen annotations and weave them back into the story periodically to provide readers with a sense of continuity. While MS3s generally perform poorly when

provided with a multitude of details (largely because they don't know which ones are most relevant to any given symbol or action) tracking one or two details (and allowing the rest to fall out of scope) successfully created continuity while not impeding generation.

MS3s contain internal representations of stories, but have limited representation of the story construction process. While we didn't explore this beyond the hidden nodes and vignette generator, it is clear that improvements could be made to help the existing structures work together. I believe that this sort of intentional design could provide stability and coherence in applications where those traits are desirable. With this in mind, I will outline some other potential systemic improvements allowing an MS3 to "know" more about its symbols and be more intentional about generation steps.

System Engineering: Authorial Affordances and Leverage One of the chief concerns for someone who is working on any sort of story generation is the simple question of efficiency, "Can I leverage the system I'm working on to generate stories faster or better than I could by hand?" This is called *authorial leverage* and is the topic of many academic endeavors [Walker et al., 2013, Sullivan et al., 2010, Chen et al., 2009]. Authorial leverage is also critical for MS3s since they're based on Case Based Reasoning (CBR) which generally needs a fair number of stories in the case library before it can effectively generate stories. Working on an MS3's domain has an exponential effect, as far as number of new stories that can be produced, so these systems are clearly advantageous from an authorial leverage point of view. One of the more interesting findings that has come along with this is that exponential generative potential means that small changes will have a large and unpredictable effect or, to put it succinctly, the system is chaotic. Depending on the research being performed, the inherently chaotic and untestable nature of

MS3s can either be a critical issue leading to a fixation on maintaining coherence (such as all of the work I have been discussing) or, as in the case of Mexica, a benefit since the goal of that research is to exhibit instances of creativity rather than to make claims about the overall potential story space.

Just as important as the question of authorial leverage, are the questions of authorial affordances: what knobs and levers do authors have, how responsive is the system to control, and is it simple to build new quality content? We found that there were a number of different variables that authors could change and that it is possible to create very different authorial experiences. Writing stories and templates for Skald has always felt out of control, an artistic pursuit in which any addition would lead to unexpected consequences that were both good and bad. In contrast, writing for P2 was a much more precise endeavor in which we could test the effects of new additions to the domain and control our outputs. This experience felt much more like writing mad-libs or a complex flow chart.

While I only have these two contrasting experiences to draw from, there are a number of very clear differences between the systems which create the different affordances. Firstly, we intentionally kept P2 templates shorter and made sure that the input and output states for each vignette were always fixed so the three phases of each story transitioned smoothly from one to another in a controlled manner. Secondly, we tightly controlled the domain in P2, allowing only carefully vetted symbols to appear. Thirdly, P2's templates were designed to reduce the appearance of indirect objects and prepositional phrases, both of which are invitations for both internal and external coherence flaws. There were no doubt many other more subtle changes between P2 and Skald but it is clear that MS3s are incredibly flexible and can be made to create dramatically different experiences both for readers and for authors.

System Engineering: Intentional Improvement is Hard As I mentioned in the previous section, MS3s are chaotic systems that have exponential generative potential. This is an extremely useful property since it tends to lead to novel output patterns, one of the key components of creativity. It also allows for a huge amount of authorial leverage which makes the story generator more tempting for authors to work with. The downside of working with a chaotic system should not be understated however: it is incredibly difficult to make improvements to MS3 with the intention of making a specific change in the output stories. Predicting how a given change will affect system output is hard. General improvements such as cleaning up the domain or adding filters for illogical patterns is simple enough but targeting a new behavior and trying to add it in is nearly impossible. In other words, trying to "teach" an MS3 how a car chase scene should be written, that some monsters require magical weapons to slay them, or when theft is an action that a character would take, is a challenging task. In my mind, this difficulty is arguably one of the main problems with MS3s after their general inability to prevent coherence flaws.

I believe that the unpredictability of MS3s comes from their chaotic nature and that in turn is derived from the simple fact that their Generative processes (their ability to generate new content) are exponential since each addition to the domain makes the rest of the domain more expressive (assuming the domain is cohesive enough for the knowledge in its stories to interact). In contrast, the Reflective processes (those which filter, redirect effort, or otherwise prune the output) are linear: each new rule or filter operates independently on the current output. Even a toy system like FIVE is capable of producing nearly 10^11 unique stories via Generative steps while its Reflective steps reduced that by a factor of only 10^5 stories. When dealing with these huge numbers it's hard not to over or undertune.

Although it is possible to calculate with precision how a change to FIVE will alter its unfiltered generative space as well as what stories it can generate and their likelihoods, most MS3s contain both Constructive and Reflective elements of such complexity that making such calculations is impossible. Suffice to say, the example provided by FIVE seems to align with our observations of MR, Skald, and P2 which were always far easier to write new content or generators for than they were to build new filters. Compounding this imbalance of generative vs. reflective power, MS3s tend to integrate generative and reflective methods into single systems. As an example, Minstrel's TRAMs both generate new content by trying to put new symbols into a given pattern, and filter that new content at the same time by only allowing certain symbols to be injected based on some rule. These integrated generative and reflective components are more efficient to operate and are simulations of some sort of human reasoning (e.g., every TRAM works by doing some flavor of fuzzy pattern matching based on a human decision process) so they're also easier to create and understand. At the same time, they're far from transparent and tracking down why some filter didn't fire or how a symbol got into a particular position is often impossible after only a few modifications from an integrated component.

7.1.6 Summary

Over the course of this chapter I have discussed general trends that we found in the output stories of MS3s. The initial discussion talked about the metrics that we initially started off looking at which would later become the LC5 attributes: Length, Complexity, Consolidation, Contextuality, Coherence, Creativity, and most importantly, Coherence. I discussed how these variables can be measured and manipulated as well as the observation that these attributes seem

to be inversely related, so improving one facet of an MS3's outputs tends to decrease the others. Coherence is sacrificed more than any other attribute, so more of our observations, insights, and research revolved around ways to improve or maintain coherence. In the end, it seems that this attribute will always be the key factor of MS3s and for those systems which aim to have coherent stories, it will always need to be attended to. It is possible to use ambiguity or setting to convince human readers to help the story maintain coherence but, until other methods can be developed, engineers of MS3s will either need to expect only small uncomplicated stories or will need to accept that a non-trivial fraction of their outputs will have significant coherence flaws. Fundamentally I believe this is all due to MS3s' inability to understand the symbols that they are manipulating and this will always make them weak at understanding the complex implications and connotations that each symbol brings along with it. Since humans are excellent at detecting these flaws and MS3s are excellent at generating a large number of stories, these systems require powerful filters (which have not yet been developed) if they are ever going to consistently produce stories that are both internally and externally coherent.

Further discussion moved on to the experience of system and domain building for Minstrel remixed, Skald, and P2. While attempting to manipulate the LC5 attributes, we developed some new approaches in domain engineering and text rendering. I went into detail describing how the practices of minimizing the symbols in the domain, building in special structural nodes to maintain context, and stringing together short stories all simplified our efforts to author stories and made our output far more coherent. Additionally, I reported on the helpful trick of using the text rendering system to add flair into stories and translate complex ideas from something the computer can understand into something that is easier

for humans to understand (e.g., a meeting leading to a pair of countries feeling angry towards each other can be translated in the English story as a declaration of war). These upgrades were critical to the success of both Skald and P2 and, while they aren't directly applicable to the afore mentioned coherence issues, I am certain that these upgrades will make solving the coherence issues far simpler.

Finally, I discussed the experience of authoring for Minstrel Remixed and its successors. It is clear that the three different systems operated differently and provided fairly different challenges for authors who are building their domains. With P2 producing Mexicaesque high level "movie plots," Skald providing compact self-contained stories with some detail, and Minstrel Remixed producing often flawed but far longer and more detailed stories, it is clear that these systems can be tuned to produce outputs with different characteristics. We found that authoring for these systems is similarly variable. Minstrel Remixed is unconstrained and trying to add in a new concept is a simple matter of producing a few stories using the new verb or noun. Ideally the author thinks about how the new symbol is similar to or should interact with existing symbols but this isn't particularly important. On the other end of the spectrum, authoring for P2 feels like writing rules. Carefully crafting its very constrained domain was challenging but also rewarding since we managed to nearly eradicate the incoherence by planning all symbol interactions beforehand and being incredibly calculated with what showed up in the domain. Ultimately, our MS3s were successfully able to provide authorial leverage and demonstrated a number of the different ways such systems can be tuned.

Chapter 8

Interactive Storytelling

There is a party game in which a group of people pass around a piece of paper with each successive person adding a sentence until someone declares the effort complete. When I initially set out to rebuild Minstrel, one of my goals was to make an interactive version of Minstrel, and I thought that modeling it after this simple party game made a lot of sense. It would be relatively easy to have Minstrel create the framework of a story and then share the burden of filling it out with a human. The computer could fill in the first few nodes, introducing some nouns and states, and then the player would fill in the first action. Then the system and player would take turns, such that Minstrel would fill in consequences of actions and perhaps add some new states or nouns and the player would fill in the next action. The intention was something that would go like this:

> **Minstrel**: Once upon a time, there was a bear named Boris, a daffodil, and a woman named Penny. Penny was unhappy with Boris.
> **Minstrel**: *What happened next?*
> **Player**: Boris gave Penny the daffodil.
> **Minstrel**: As a result, Penny was happy with Boris. Everyone was happy but Boris was hungry.
> **Minstrel**: *What happened next?*
> ...

Once Interactive Minstrel could tell stories in this manner, I figured that it could build multi-stage quests that were superior to those built by hand for World of Warcraft [BlizzardEntertainment, 2004] or any of the other MMORPGs that were its contemporaries. These games' quests generally follow an incredibly simple Action-Number-Noun pattern such as: Kill 20 bunnies, collect 5 flowers, or make 3 townspeople happy (which generally just means 3 more nested Action-Number-Noun quests.) Since these games have a simple set of actions (e.g. fight, kill, flee, transfer object, or talk with) and generally have a fairly constrained set of inanimate objects, building an automated quest generator to replace the developer-supplied quests would be trivial. A system that could handle Boris and Penny should be upgradeable to track a small set of states for a half-dozen villagers, generate quests based on 'negative' states, mutate states based on how and which quests are accomplished, and thus supply a chain of quests that feed back into each other. I wanted to see if a program such as Minstrel, which could be configured to generate simple MMORPG quests, could be upgraded to create an interactive experience both by allowing players to 'solve' quests in multiple ways, and by creating future quests based on the actions and outputs of prior ones.

8.1 Three Step Stories and Player Intent

As a prototype and proof of concept, we set up Skald to make "3 step stories" in which Skald provides a setup, the player provides an action via text input, and then Skald finishes out the story. An example of this follows:

> **Minstrel**: Once upon a time there was a Knight named Lancelot, a Troll named Gorthorog, a Knight named Kay, and a Spear named Spear. Gorthorog did not like Lancelot. Lancelot did not like Gorthorog. Kay owned Spear.
>
> **What happened next?**

Player: Lancelot attacks Gorthorog.

Minstrel: As a result, Gorthorog dies. Kay feels happy towards Lancelot which motivates him to give Lancelot Spear.

In this example, Skald sets up some animosity between Lancelot and Gorthrog and even provides a subtle nudge to the player in the form of a spear owned by a third party (Kay). Players who choose to have Lancelot attack Gorthrog in this situation are almost always rewarded with an ending similar to the example. There is, of course, a second option which makes sense:

Minstrel: Once upon a time there was a Knight named Lancelot, a Troll named Gorthorog, a Knight named Kay, and a Spear named Spear. Gorthorog did not like Lancelot. Lancelot did not like Gorthorog. Kay owned Spear.

What happened next?
Player: *Lancelot gets Spear from Kay.*

Minstrel: As a result, Kay was unhappy with Lancelot. Gorthorog gets Spear from Lancelot and eats it, making Gorthorog like Lancelot.

When a player tries to prepare Lancelot for an attack on Gorthrog by having him take the Spear from Kay, there are a number of different possible results, some of which are attacks on Gorthrog, but many of which deviate significantly from the player's intention. The non-combat endings were disappointing and although the 'surprise twist' was novel to see once or twice, after a few iterations of this sort, it became clear that the system would ignore any implied motives and assume that the players' intent was completely contained in the action they selected. In short, this form of Skald performed well as a simulator but when players tried to help it create more complex stories, it failed far more frequently than we were comfortable with. Problem Planets initially had the same trouble and I believe that this issue will be a challenge for any interactive storytelling system that allows unconstrained player input.

We identified a few factors that contribute to this problem:

- Our choice of a simple interface doesn't let players input enough data to encode their intention.
- Humans don't think like MS3s and don't realize the amount of context that they would need to provide.
- Skald's structure doesn't support the idea of preparing for a particular type of action. Goals are achieved or thwarted by states and actions are just ways to 'complete' a goal.

Later in this chapter I will discuss how we eventually bypassed this problem by constraining player input and forcing them to input their intentions but these two choices meant that we were forced to abandon the open-ended aspect of our original goal.

8.2 Playing With Problem Planets

When the Minstrel Remixed team finished the above prototype and then started working on Problem Planets, one of our first observations was that audiences are less forgiving of incoherence when collaborating than when presented with a pre-generated story. I suspect this is due to the personal investment of a player who is both trying to understand what is going on and accomplish some personal goals while shaping the story. Whatever the reason, players who saw enough incoherent stories would start to explore the ways in which they can consistently break the system. This behavior reminds me of a the discussion of the Eliza Effect that Wardrip-Fruin presents in Expressive Processing [Wardrip-Fruin, 2009].

Eliza, an early chatbot designed to use simplistic scripts to hold conversations with humans, falls apart when its interactions are not tightly constrained and

humans find ways to break out of the script. In those situations, users either try to help the chatbot move the conversation along, or explore its incoherence, but can't authentically experience the magic of the system now that they have an idea of how it works. This is exactly what I found happened with Problem Planets; once the illusion was broken, users are left only with trying to see how far they can tip the system.

8.3 Constraints are a Solution

Thus far, I have discussed the main challenges that the Minstrel Remixed team found while working on Problem Planets. In the absence of a human-level intelligence to determine player intention, directing the player or limiting their options as much as possible is advantageous to maintain coherence. Additionally, we found that forcing the player to state their intention immediately after reading the story introduction allowed us to avoid the need of inferring player intention. To these ends, we discovered two successful strategies that we employed for Problem Planets: limiting a player's possible goals, and providing proactive guidance rather than reactive understanding. Put another way, it's far simpler to know what a player is doing if there are only a handful of reasonable choices (especially when the player is required to state their goal at the beginning.) Similarly, framing the story and choice in suggestive ways can make a user's choices far more predictable. For example, a player has to choose how to resolve an issue in which two political factions on a planet are at an impasse. Whether the player is provided with a simple sentence outlining the situation or information about each faction backed up by characters and narrative experiences, when the player is asked what they would like to do, their actions, motivations, and expectations are effectively unconstrained. In contrast, if the player is informed that the rich

corporations are subjugating the impoverished workers and that the impasse is a workers' strike for living wages, the player's reasons, choices, and expectations are far more predictable. However, the need to impose these constraints in Problem Planets means that we failed at creating the kind of open-ended collaborative story generation we hoped an MS3 could support. True collaborative storytelling is only marginally attainable without human level comprehension of the user's contribution in order to steer the story away from coherence flaws.

Chapter 9

Future Work

While the ultimate results of my investigations have lead me to the conclusion that we don't yet have all of the pieces required to create reliable interactive storytelling and that it is not likely to be cost-effective without some radical new approach or advances in machine based reasoning and common sense, there are still a large number of possible explorations that can be done to advance the computational storytelling field using Skald and a few likely fruitful directions that could be taken.

Likely the easiest next step for further development in this field would be to figure out how to consistently build high quality domains. Over the course of constructing Minstrel Remixed and its successors, the Minstrel team discovered a number of insights about domain construction. These insights were always identified in hindsight after we had already paid the costs to build a problematic domain and we developed all of our guidelines for constructing domains by attempting to troubleshoot these problems. Thus, our ad-hoc domain construction and refactoring was useful, but I believe that a procedure or set of best practices for building good domains could be developed which would allow domain construction to become a repeatable engineering task. This would make domain engineering far

more efficient and consistent. Given the fact that Minstrel and Skald already exist, I believe that building a procedure for domain construction could also benefit from developing a quick set of metrics to help quantify this development effort. Being able to score a domain on one or more useful qualities would make any sort of process far quicker and more direct when metrics can tell you if you have succeeded rather than requiring humans to evaluate stories by hand.

I believe that metrics hold the key to further explorations of Minstrel outside of their applicability to domain engineering. One of the largest impediments that the Minstrel team came across while doing our research was the fact that our ability to generate stories was many orders of magnitude faster than our ability to judge their LC5 attributes by hand. Doing this slow and repetitive process just once to determine the creative potential of the various settings of Minstrel's transformation system [Tearse et al., 2011b] was sufficiently tedious and subjective to convince the team not to attempt that form of analysis again. Upon further consideration, although I still have no better ideas about how to directly measure the qualities of individual stories, I wonder whether it might be fruitful to develop a method to judge the entire set of possible stories produced by a given storytelling configuration and derive useful information from that.

9.1 Measuring MS3 Configurations

After the Problem Planets project was complete, I was happy to discover that it is possible to calculate all possible transformations of a particular pattern of story elements along with their likelihoods. Given this, it is possible to take an initial story graph and exhaustively apply every possible TRAM and ALP to yield all possible stories produced from that initial state. With a little optimization (many of the states are visited countless times during this traversal and only need to be

calculated once,) it is possible to generate all output stories and their likelihoods in relatively short order (Skald uses a fairly small domain but I could generate stories in seconds and all possible stories given a starting condition in less than a minute). From there, it isn't too much of a challenge to simply run this process for all of the story templates in the domain and receive two very useful artifacts, a Possible Story Space (PSS) as well as a Story Likelihood Map (SLM). When analyzed, the PSS and SLM should be able to provide researchers and engineers with both diagnostic and measurement tools. The rest of this section outlines a few examples.

Metric 1: Level-N Coverage. Given a PSS, for all story fragments of size N, how many possible patterns are available? What ratio of the possible patterns already exists in the domain? This yields a ratio that I'm calling the domain's coverage. The coverage of a domain should provides a decent measurement of the ability of the system to generate novel and coherent outputs (which are generally understood to be primary components of creativity). Configurations (system tuning coupled with a domain) with low level-N coverage numbers would have little guidance about how symbols should be arranged in N-sized story fragments and would probably tend towards more incoherence and higher novelty (since many of the story fragments that it constructs don't exist in the library). At the same time, configurations with high coverage numbers would be the opposite, tending towards more coherence but less novelty. This metric would be wonderful for quickly understanding some abstract concepts about an MS3's configuration which aren't currently easy to grasp or observe. As an example of this, consider a configuration with high level-1 coverage but low level-2 coverage. This would indicate that all of the individual elements of generated stories are borrowed from some other story in the library (high level-1 coverage) but the larger blocks of the story have novel

rearrangements of the lower level patterns (low level-2 coverage). I would also imagine this metric could be used to evaluate changes: having a coverage number jump from 10% to 20% as a result of a single change probably would indicate that the alterations were effective. Also, High level-N percentages would indicate that a storytelling configuration is providing little authorial leverage for that size of story fragment. From a diagnostic point of view, engineers seeing extremely high or low numbers for most of a particular configuration's level-1,2, and 3 coverage values likely need to alter their configuration since its symbols are either far too sparse or dense. From a research point of view, it would be interesting to look at this metric and investigate if there's a particular percentage for each level (perhaps influenced by domain size,) which optimizes creativity, coherence, or any other desired output attribute. It would also be interesting to observe the differences between values of N. Correlations between coverage numbers at different levels might provide some interesting insights into how the domain and system interact at a more structural level. My suspicion is that high values of level-1 coverage is desirable to reduce the number of coherence flaws and that at some point (level 2 or 3 most likely), it is desirable to have low coverage numbers. This would indicate that the system has good support for how to correctly handle basic verbs and low level details (no more knights attacking with a spoon), but has the freedom to be creative with larger patterns in the story.

Metric 2: Associated symbols. Given a Story Likelihood Map (SLM) and two symbols x and y, it is possible to calculate how likely it is for x and y to appear within n nodes of each other. By iterating through every symbol in every story in the SLM and storing the minimum distance to every other symbol in that story along with the likelihood of that story, it is possible to do a little math and produce a comprehensive list of every pair of symbols, x, and y and what the percentage

chance of having them both appear in a single randomly generated story within 1, 2, 3, etc. nodes of one another. Combining these values into a single list, a researcher could, for example, look up the verb 'attack' and see the spread of subjects, objects, and direct objects that likely appear within a given distance of the 'attack' verb. In this example, one might expect the highest probabilities for symbols at distance n = 0 from the 'attack' verb to be weapons and violent beings. At n = 1, one might expect to see see that death, injury, and changes in emotional state are the most likely results of an attack and that nearly all attacks are predicated by the two combatants being in the same physical space.

Various observations about the Associated Symbols list would help with domain engineering. Symbols that are highly associated with very few other symbols should probably get more varied representation in the story library so the MS3 can 'learn' more about good patterns to use that symbol in. Alternatively, it is possible that if a few symbols are exclusively used with one another, for example, if every aggressive action in a hypothetical domain happened with a sword. In this case, the paired symbols should be merged into one to reduce the number of underrepresented symbols in the domain. Whatever the case, symbols that have a large number of weak associations are likely to be problematic or at least worth checking on. These symbols are likely to be generic and are likely be the inflection points around which a story can pivot from one topic to something completely different. It is likely that there is an optimal range, both in number and probability distribution, of associated symbols. I believe that this distribution could be very interesting to both observe and attempt to manipulate. From a diagnostic perspective, this sort of information would be incredibly useful to allow a domain engineer to look at the effects of alterations without having to generate dozens of stories. This method could also be used to flag symbols that are either highly or

weakly associated with other symbols. Additionally, aggregates could be created from the lists of associated symbols. These would give engineers an indication about how many elements appear to be outside of the ideal range (whatever that is determined to be,) and could thus serve as another metric indicating the health of a particular configuration.

In hindsight, if we had developed either the Level-N Coverage or the Associated Symbols metrics and used them on Minstrel Remixed, Skald, or Problem Planets, I believe the scores would indicate a large number of places that we could invest more time and effort filling in the domain (and thus teaching the system about the world). This would have provided us with far better domains to work with and likely would have helped us debug issues that we instead had to solve with guesswork. Given these two metrics, along with the PSS, and the SLM, there are a number of potentially fruitful research questions that could be asked. One such question is, could one find a way to reliably manipulate these numbers independently of one another? As an example, while working on P2, the Minstrel team refactored the domain and removed as many symbols as we could (which would shrink the PSS). I believe that we managed to increase our level 1 coverage numbers while we did this (more of our possible stories contained more nodes that existed somewhere in the story library). The question is, if we had taken the P2 domain prior to our refactor and removed stories and symbols randomly until we had a comparably sized PSS to our refactored one, would the level 1 coverage of the curated and randomly truncated configurations be the same? My suspicion is that this test would indicate that we managed to effectively guide our domain towards higher coverage through thoughtful effort.

9.2 Multiple Domains

A third interesting research direction to further MS3s would be to determine if a system might benefit from using multiple domains for the same story. Eliza can be made to work with different scripts to simulate different types of interactions [Weizenbaum, 1966]. An MS3 could be setup to do something similar but to have each of the different domains handle a different aspect of storytelling. For example, one domain could be used to create the overall plot for a story in a manner similar to Mexica, while alternative domains could specialize in creating special types of substories (e.g. action sequences, interpersonal interactions, etc.) which could then be inserted as replacements for nodes in the overall plot. Specialized domains would contain a smaller number of symbols than a generic domain and would likely suffer from fewer coherence issues. A multi-domain setup may suffer from decreased authorial leverage since information about how the world works would need to be injected into multiple specialized domains. Maintaining a large separation between the specialized domains would likely mitigate this since, for example, interpersonal scenes are unlikely to need to understand that dead things can't take actions.

Since MS3s heavily skew their outputs towards the content of their domains, these help to guide story generation towards patterns that are known to be good. It would be interesting to see the effect of a subsystem which would allow authors and users to provide negative feedback to an MS3. While attempting to engineer a domain to produce certain types of stories, I often found that Skald had produced a 'bad' story fragment in the middle of an otherwise fine output. If any given bad fragment cropped up too often, I would then have to figure out what combination of symbols and TRAMs/ALPs/stories ended up producing the fragment so I could limit its appearance or correct misbehaving components. I am proposing that it

would be far simpler to create a blacklist to which I could add a little rule that would detect that pattern and correct it. Skald's Author Level Plans (ALPs) are the only way to do this currently but they're much more general purpose and require far more effort to author than simply adding a pattern to a blacklist. Expanding on this idea, it would be interesting to create such a system that would allow readers to flag bad fragments and thus allow the MS3 to learn from user feedback. In addition to the already mentioned benefits of simplified domain engineering and improved story quality, such a system might be very interesting for users to interact with since users have never been asked to evaluate MS3 story fragments before. It would also be possible (depending on the implementation,) to allow users to indicate a bad fragment and have the MS3 regenerate that fragment, enabling a unique experience wherein users could manipulate a story after generation.

9.3 More Information About Symbols

My last suggestion for new research directions for Skald is to investigate whether making symbols more knowledge-dense could provide some additional power to the system. Specifically, I think that creating multiple different taxonomic categories which could all be used at the same time would help to provide more leverage to the domain engineers. Some of the most interesting TRAMs use the ontology to make logical transformations and this would expand that concept and enable more connections between symbols without authoring more content. This would simplify many of the tasks involved with engineering a domain since categories could be used to define broad attributes which apply to many symbols but don't correlate nicely with the standard ontology. For example, it is currently impossible to encode the concept that humans can ride horses, but not

young horses, or more generically, 'x can do y unless/except z.' This sort of rule is impossible to encode with Skald's current methods. One might approach it by creating many stories about humans riding adult horses and none about humans riding baby horses but this is an incredibly inefficient and indirect way to perform a simple correction. One might argue that the principles governing Minstrel and its successors dictate that stories should be generated from other stories, not restricted by rules. This is true but I believe that rules providing inferences on symbols could work synergistically with TRAMs to provide more variety and more coherence with less work.

9.4 Closing Thoughts

In closing, while there are a number of possible directions for further research, each one introduces more complexity to Minstrel. Minstrel Remixed, Skald, and Problem Planets are all improvements upon Minstrel, each one enabling new structures, different user experiences, and making incremental improvements upon the quality and complexity of the output stories. At the same time, each one also demonstrates the gap that exists between the idea that stories contain interchangeable pieces and making a collection of those pieces fit together nicely into a high quality story. Fundamentally I believe that an MS3 could bridge this gap and reliably produce high quality stories but that while Skald is closer to this goal than Minstrel was, it still has a number of challenges that it needs to overcome before it can do so.

Bibliography

[Barone, 2016] Barone, E. (2016). Stardew valley.

[BioWare, 2009] BioWare (2009). Dragon age: Origins.

[BlizzardEntertainment, 2004] BlizzardEntertainment (2004). World of warcraft.

[Boden, 2004] Boden, M. (2004). *The Creative Mind*. Routledge, London, 2nd edition.

[Bringsjord and Ferrucci, 2000] Bringsjord, S. and Ferrucci, D. A. (2000). *Artificial Intelligence and LiteraryCreativity. Inside the Mind of BRUTUS, a Storytelling Machine*. Lawrence Erlbaum Associates, Inc., Hillsdale, New Jersey.

[Bui et al., 2010] Bui, V., Abbbass, H., and Bender, A. (2010). Evolving stories: Grammar evolution for automatic plot generation. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE.

[Chen et al., 2009] Chen, S., Nelson, M. J., Sullivan, A., and Mateas, M. (2009). Evaluating the authorial leverage of drama management. In *AAAI Spring Symposium: Intelligent Narrative Technologies II*, pages 20–23.

[Cinematronics, 1983] Cinematronics (1983). Dragon's lair.

[Colton et al., 2011] Colton, S., Charnley, J., and Pease, A. (2011). Computational Creativity Theory: The FACE and IDEA Descriptive Models. *Proceedings of the Second International Conference on Computational Creativity*.

[Costikyan, 2007] Costikyan, G. (2007). Games, storytelling, and breaking the string. *Second Person: Roleplaying and Story in Playable Media*, pages 5–14.

[Dehn, 1981] Dehn, N. (1981). Memory in story invention. In *Proceedings of the third annual conference of the cognitive science society*, pages 213–215.

[Dyer, 1982] Dyer, M. G. (1982). *In-Depth Understanding. A Computer Model of Integrated Processing for Narrative Comprehension.* PhD thesis.

[Erol, 1996] Erol, K. (1996). *Hierarchical task network planning: formalization, analysis, and implementation*. PhD thesis.

[Fairclough and Cunningham, 2003] Fairclough, C. and Cunningham, P. (2003). A multiplayer case based story engine. Technical Report Bremond, Trinity College Dublin, Department of Computer Science, Dublin.

[Gervas, 2009] Gervas, P. (2009). Computational approaches to storytelling and creativity. *AI Magazine*, 30(3):49.

[Gervás et al., 2005] Gervás, P., Díaz-Agudo, B., Peinado, F., and Hervás, R. (2005). Story plot generation based on cbr. In *Applications and Innovations in Intelligent Systems XII*, pages 33–46. Springer.

[Gervas et al., 2005] Gervas, P., Diazagudo, B., Peinado, F., and Hervas, R. (2005). Story plot generation based on CBR. *Knowledge-Based Systems*, 18(4-5):235–242.

[Glassco, 2004] Glassco, B. (2004). Betrayal at house on the hill.

[Goldberg, 1985] Goldberg, E. (1985). Tales of the arabian nights.

[Gygax and Arneson, 1974] Gygax, G. and Arneson, D. (1974). *Dungeons and Dragons: Rules for Fantastic Medieval Role Playing Adventure Game Campaigns: Playable with Paper and Pencil and Miniature Figures*. Tactical Studies Rules (TSR).

[Infocom, 1977] Infocom (1977). Zork.

[Klein, 1965] Klein, S. (1965). Control of Style with a Generative Grammar. *Language*, 41:619–631.

[Klein et al., 1973] Klein, S., Aeschiliman, J. F., Balsiger, D., Converse, S., Court, C., Foster, M., Lao, R., Oakley, J. D., and Smith, J. (1973). Automatic novel writing: A status report. Technical Report TR186, Computer Sciences Department, The University of Wisconsin–Madison.

[Klein et al., 1971] Klein, S., Oakley, J. D., Suurballe, D. J., and Ziesemer, R. A. (1971). A program for generating reports on the status and history of stochastically modifiable semantic models of arbitrary universes. Technical Report TR142, Computer Sciences Department, The University of Wisconsin–Madison.

[Langley and Stromsten, 2000] Langley, P. and Stromsten, S. (2000). Learning context-free grammars with a simplicity bias. In *European Conference on Machine Learning*, pages 220–228. Springer.

[Lebowitz, 1984] Lebowitz, M. (1984). Creating characters in a story-telling universe. *Poetics*, 13(3):171–194.

[LucasArts, 1990] LucasArts (1990). The secret of monkey island.

[Mateas and Sengers, 2003] Mateas, M. and Sengers, P. (2003). *Narrative intelligence*. J. Benjamins Pub.

[Mateas and Stern, 2003] Mateas, M. and Stern, A. (2003). Façade: An experiment in building a fully-realized interactive drama. In *Game developers conference*, volume 2, pages 4–8.

[McDonald, 1999] McDonald, D. (1999). A rational reconstruction of genaro. In *Proceedings of the RAGS Workshop, Edinburgh*.

[Meehan, 1976] Meehan, J. (1976). *The Metanovel: Writing Stories by Computer*. PhD thesis, Yale University.

[Meehan, 1981] Meehan, J. (1981). TALE-SPIN. In Schank, R. and Riesbeck, C., editors, *Inside Computer Understanding: Five Programs plus Miniatures*, chapter TALE-SPIN. Lawrence Erlbaum Associates, Inc., Hillsdale, New Jersey.

[Montfort, 2008] Montfort, N. (2008). Nick montfort's 1k story generators. `https://procedural-generation.tumblr.com/post/134427592623/nick-montforts-1k-story-generators-2008-now`. Accessed: 2018-10-11.

[Montfort and Wardrip-Fruin, 2004] Montfort, N. and Wardrip-Fruin, N. (2004). Acid-free Bits: Recommendations for Longer-lasting Electronic Literature from the ELO's Preservation, Archiving, and Dissemination (PAD) Project.

[Musen et al., 1995] Musen, M. A., Gennari, J. H., and Wong, W. W. (1995). A rational reconstruction of internist-i using protege-ii. In *Proceedings of the Annual Symposium on Computer Application in Medical Care*, page 289. American Medical Informatics Association.

[Ontañón et al., 2010] Ontañón, S., Zhu, J., et al. (2010). Story and text generation through computational analogy in the riu system. pages 51–56. Association for the Advancement of Artificial Intelligence (AAAI).

[Packard, 1980] Packard, E. (1980). *The Mystery of Chimney Rock (Choose Your Own Adventure, #5)*. Bantam.

[Pajitnov, 1984] Pajitnov, A. (1984). Tetris.

[Partridge, 1991] Partridge, D. (1991). *A new guide to artificial intelligence*. Intellect Books.

[Partridge and Wilks, 1990] Partridge, D. and Wilks, Y. (1990). *The foundations of artificial intelligence: A sourcebook*. Cambridge University Press.

[Peinado and Gervás, 2006] Peinado, F. and Gervás, P. (2006). Minstrel reloaded: from the magic of lisp to the formal semantics of owl. In *International Conference on Technologies for Interactive Digital Storytelling and Entertainment*, pages 93–97. Springer.

[PÉrez and Sharples, 2001] PÉrez, R. P. Ý. and Sharples, M. (2001). Mexica: A computer model of a cognitive account of creative writing. *Journal of Experimental & Theoretical Artificial Intelligence*, 13(2):119–139.

[Pérez y Pérez, 1999] Pérez y Pérez, R. (1999). MEXICA: A Computer Model of Creativity in Writing. Master's thesis, Falmer, UK. Ph.D. dissertation.

[Pérez y Pérez and Sharples, 2001] Pérez y Pérez, R. and Sharples, M. (2001). MEXICA: A computer model of a cognitive account of creative writing. *Journal of Experimental and Theoretical Artificial Intelligence*, 13:119–139.

[Pérez y Pérez and Sharples, 2004a] Pérez y Pérez, R. and Sharples, M. (2004a). Three computer-based models of storytelling: Brutus, minstrel and mexica. *Knowledge-based systems*, 17(1):15–29.

[Pérez y Pérez and Sharples, 2004b] Pérez y Pérez, R. and Sharples, M. (2004b). Three computer-based models of storytelling: Brutus, minstrel and mexica. *Knowledge-based systems*, 17(1):15–29.

[Propp, 1968] Propp, V. (1968). *Morphology of the Folktale*. University of Texas Press.

[QuanticDream, 2010] QuanticDream (2010). Heavy rain.

[Queneau, 1961] Queneau, R. (1961). *Cent mille milliards de poèmes*.

[Riedl, 2004] Riedl, M. (2004). Narrative Planning: Balancing Plot and Character. Ph.D. dissertation.

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

[Ritchie, 1984] Ritchie, G. (1984). A rational reconstruction of the proteus sentence planner. In *Proceedings of the 10th International Conference on Computational Linguistics and 22nd annual meeting on Association for Computational Linguistics*, pages 327–329. Association for Computational Linguistics.

[Ritchie, 2001] Ritchie, G. (2001). Assessing Creativity. In *Proceedings of the AISB '01 Symposium on AI and Creativity in Arts and Science*, pages 3–11, Heslington, UK.

[RockstarGames, 2013] RockstarGames (2013). Grand theft auto v.

[Rothenberg, 1995] Rothenberg, J. (1995). Ensuring the Longevity of Digital Documents. *Scientific American*, 272(1):42–47.

[Schank, 1972] Schank, R. C. (1972). Conceptual Dependence: A Theory of Natural Language Understanding. *Cognitive Psychology*, 3(4).

[Schank and Abelson, 1977] Schank, R. C. and Abelson, R. P. (1977). Scripts, plans, goals and understanding: an inquiry into human knowledge structures. hillsdale, nj: L.

[Sgouros, 1999] Sgouros, N. M. (1999). Dynamic generation, management and resolution of interactive plots. *Artificial Intelligence*, 107(1):29–62.

[Sharples, 1978] Sharples, M. (1978). Poetry from LOGO.

[Sharples, 1997] Sharples, M. (1997). Storytelling by computer. *Digital Creativity*, 8(1):20–29.

[Sharples, 1999] Sharples, M. (1999). *How we write: Writing as creative design*. Psychology Press.

[Simpson et al., 1989] Simpson, J. A., Weiner, E. S., et al. (1989). *The Oxford english dictionary*, volume 2. Clarendon Press Oxford.

[Smith, 2012] Smith, G. M. (2012). *Expressive design tools: Procedural content generation for game designers*. PhD thesis, UC Santa Cruz.

[Studios, 2015] Studios, B. G. (2015). Fallout 4.

[Sullivan et al., 2010] Sullivan, A., Mateas, M., and Wardrip-Fruin, N. (2010). Rules of engagement: moving beyond combat-based quests. In *Proceedings of the Intelligent Narrative Technologies III Workshop*, page 11. ACM.

[Swanson and Gordon, 2008] Swanson, R. and Gordon, A. S. (2008). Say anything: A massively collaborative open domain story writing companion. In *Interactive Storytelling*, pages 32–40. Springer.

[Talmy, 1985] Talmy, L. (1985). Force dynamics in language and thought. *Parasession on causatives and agentivity*, 1:293–337.

[Talmy, 1988] Talmy, L. (1988). Force dynamics in language and cognition. *Cognitive science*, 12(1):49–100.

[Tearse, 2011] Tearse, B. (2011). Minstrel Remixed: A Reconstruction and Exploration. In *Proceedings of the 6th International Conference on Foundations of Digital Games*, pages 253–255.

[Tearse et al., 2010a] Tearse, B., Mateas, M., and Wardrip-Fruin, N. (2010a). MINSTREL Remixed: a rational reconstruction. In *Proceedings of the Intelligent Narrative Technologies III Workshop*, Monterey, CA. ACM.

[Tearse et al., 2011a] Tearse, B., Mawhorter, P., Mateas, M., and Wardrip-fruin, N. (2011a). Minstrel Remixed: User Interface and Demonstration. In *Proceedings of AIIDE 2011*.

[Tearse et al., 2012] Tearse, B., Mawhorter, P., Mateas, M., and Wardrip-Fruin, N. (2012). Lessons Learned From a Rational Reconstruction of Minstrel. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*.

[Tearse et al., 2014] Tearse, B., Mawhorter, P., Mateas, M., and Wardrip-Fruin, N. (2014). Skald: Minstrel reconstructed. *Computational Intelligence and AI in Games, IEEE Transactions on*, 6(2):156–165.

[Tearse et al., 2010b] Tearse, B., Wardrip-fruin, N., and Mateas, M. (2010b). Minstrel Remixed : Procedurally Generating Stories. In *Proceedings of AIIDE 2010*.

[Tearse et al., 2011b] Tearse, B. R., Mawhorter, P., Wardrip-fruin, N., and Mateas, M. (2011b). Experimental Results from a Rational Reconstruction of Minstrel. In *International Conference on Computational Creativity*.

[Theune et al., 2003] Theune, M., Faas, S., Heylen, D. K. J., and Nijholt, A. (2003). The virtual storyteller: Story creation by intelligent agents. In Göbel, S., Braun, N., Spierling, U., Dechau, J., and Diener, H., editors, *TIDSE 2003: Technologies for Interactive Digital Storytelling and Entertainment, Darmstadt, Germany*, pages 204–215, Darmstadt. Fraunhofer IRB Verlag.

[Thue et al., 2013] Thue, D., Bulitko, V., and Hamilton, H. (2013). Implementation cost and efficiency for ai experience managers. In *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*.

[Turner, 1993] Turner, S. (1993). *MINSTREL: a computer model of creativity and storytelling*. PhD thesis, University of California, Los Angeles.

[Turner, 1994] Turner, S. R. (1994). *The Creative Process: A Computer Model of Storytelling And Creativity*. Lawrence Erlbaum Associates, Inc., Hillsdale, New Jersey.

[Walker et al., 2013] Walker, M. A., Sawyer, J., Jimenez, C., Rishes, E., Lin, G. I., Hu, Z., Pinckard, J., and Wardrip-Fruin, N. (2013). Using expressive language generation to increase authorial leverage. *Intelligent Narrative Technologies*, 6.

[Wardrip-Fruin, 2009] Wardrip-Fruin, N. (2009). *Expressive Processing: Digital fictions, computer games, and software studies*. The MIT Press.

[Weizenbaum, 1966] Weizenbaum, J. (1966). Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45.

[Wiggins, 2006] Wiggins, G. (2006). A Preliminary Framework for Description, Analysis and Comparison of Creative Systems. *Journal of KNowledge Based Systems*, 19(7):449–458.

[Zhu, 2012] Zhu, J. (2012). Towards a mixed evaluation approach for computational narrative systems. In *International Conference on Computational Creativity*, page 150.

[Zhu and Ontanón, 2010] Zhu, J. and Ontanón, S. (2010). Towards analogy-based story generation. In *Proceedings of the First International Conference on Computational Creativity (ICCC-X)*, pages 75–84.