Narrative Intelligence

Warren Sack <wsack@ucsc.edu>

Abstract: Research in Artificial Intelligence (AI) -- before the so-called ‘AI winter’ when funding from the military went south -- repeatedly engaged with stories: How might one write software to understand and generate narratives? From the time of this ‘winter,’ roughly the late-1980s, until the mid-1990s, fun topics, like storytelling, were put to the side. The US military had been open to funding all kinds of open-ended projects that were, in many cases, in close dialog with humanities scholarship -- especially literature and philosophy -- even if the ‘dialogs’ were most frequently heated arguments. But, when the military money went away, the new sponsors wanted research that was obviously ‘serious’ business: expert systems for oil exploration and credit card fraud detection, machine vision algorithms for surveillance, and so forth. But, now it is a new, virtual world where active investment comes from game companies and fun is fundable again. The problem is that no one did much in this area of fun - story generation - throughout the freeze and the thaw. As artists, designers and researchers, we now need to pick up the pieces from where they were left decades ago. What is a good story generation algorithm? What would make it better? Have we learned anything in the past years that could make new work in this area better than the ‘classic’ work of the 1970s and 1980s? I propose to do a rational reconstruction of one of the ‘classics’ (James Meehan's Tale-Spin story generator) and explore the means and meaning of reviving fun research in AI decades after its death.

Introduction

I met Narrative Intelligence (NI) in Santa Cruz, California. After college and then three years as a research assistant with the Yale Artificial Intelligence Project, I had moved to Santa Cruz with my partner who was pursuing her Ph.D. in the History of Consciousness Department (HistCon) at the University of California. I was a HistCon want-to-be, attending seminars unofficially, hanging out with graduate students and faculty, and making a living computer programming. I was struggling to connect the humanities discourses of Santa Cruz to what I knew about computers. NI showed up in Santa Cruz for a few days in April of 1991 for the *Second International Conference on Cyberspace* that HistCon alum Allucquére “Sandy” Stone organized.

The NI that I met that spring was a subset of the Narrative Intelligence Reading Group from the MIT Media Lab (see Davis and Travers, 2003). I had a great time talking with the co-founders of the group, Marc Davis and Michael Travers, and they invited me to come present my struggles with computation and the humanities to the group in Cambridge, Massachusetts. I went. I was seduced by the excitement and intellectual energy. I applied to the Lab as a graduate student and ended up staying at the Media Lab for most of the 1990s as a master's student, then as a doctoral student, and finally as a research scientist. The weekly meeting of the informal NI group was a heartbeat of my time at the Lab. I co-organized it with Amy Bruckman from 1994 until 1996.

NI at the Media Lab comprised MIT graduate students and faculty, but drew local members from other geographically close schools and research labs and integrated members from all over through a lively online discussion. We began by trying to find a way to synthesize a connection between artificial intelligence and literary theory, but ended by linking many more aspects of computer science and the human sciences.

In 1999 two of our remote members, Michael Mateas and Phoebe Sengers, organized a Narrative Intelligence workshop at the one of the main venues for Artificial Intelligence, the Association for the Advancement of Artificial Intelligence (AAAI) Fall Symposium Series. Michael and Phoebe gathered together many of us from the MIT Media Lab as well as many others from the United States and elsewhere. They edited a book from the proceedings (Mateas and Sengers, 2003). And, since then, there has been a regular series of conferences, workshops, and symposia on the topic.[1] Michael and I both ended up as professors here, at the University of California Santa Cruz; and, his group, the Expressive Intelligence Studio (that he co-directs with Noah Wardrip-Fruin), is now one of the places where Narrative Intelligence flourishes today.

[1] Marie-Laure Ryan points out that "narrative intelligence" was coined by the literary theorist Paul Ricoeur in his 1982 book entitled *Temps et Recits* (Paris: Seuil) (see Ryan, 1991, p. 233). Research in the area of Narrative Intelligence is currently published and exchanged in a variety of venues. Some of these for are specific to the topic such as the ACM Intelligent Narrative Technologies Workshops, the International Conference on Interactive Digital Storytelling (which superseded the previous two European conference series: Technologies for Interactive Digital Storytelling (TIDSE) and Virtual Storytelling – Using Virtual Reality Technologies for Storytelling (ICVS)), and the ACM Multimedia Workshop on Story Representation, Mechanism and Context. Others are conferences and workshops devoted more widely to artificial intelligence such as the Artificial Intelligence and Interactive Digital Entertainment (AIIDE) conference, or special workshops like the AAAI Workshop on Computational Aesthetics: Artificial Intelligence Approaches to Beauty and Happiness. Others are conferences and journals where Narrative Intelligence is discussed in the context of computer games: Game Developers Conference, the ACM Foundations of Digital Games Conference, IEEE Transactions on Computational Intelligence and AI in Games, ACM Workshop on Procedural Content Generation in Games. This list is not complete.

From the mid-1970s until the late-1980s there was large-scale support for AI research, especially from the United States Department of Defense (DoD). During that period, a lot of work was done connecting literary theory and computer science. The Yale Artificial Intelligence Project directed by Roger Schank had DoD funding and was one of the main sites where story understanding and story generation were investigated in computational terms. Complementary work was being done by literary theorists interested in employing formal tools of logic, mathematics, and computer science. For example, Teun van Dijk (then at the University of Amsterdam) founded the journal *Poetics* in 1971 which, for decades, has been a venue where ideas from structuralism, narratology, cognitive science, and computer science have mixed. There have been periods when funding for AI research has been less generous. These periods are referred to as "AI winter" (cf., Hendler, 2008). The last such "winter" occurred from the late-1980s through the mid-1990s.

Nevertheless, when I began graduate school in 1992, the AI winter in the United States showed no signs of melting. At the time, the MIT Media Lab was sheltered from the AI winter because relatively little of the Media Lab's funding came from the DoD. But, researchers in AI Labs had largely abandoned the field of narrative or were attempting to apply older findings for other purposes. For example, Roger Schank moved to Northwestern University to found the Institute of Learning Sciences were he and his group largely focused on the development of systems for training and education.

How then can ideas planted before the AI winter be transplanted to a new environment and survive? Posed as a more general question, it clear that this is not an idiosyncratic concern, but rather one of the basic issues of the humanities. To frame this question, one must first acknowledge that the "ideas" are of a particular material form: the ideas of AI are written as books, articles, and computer programs. In short, they are *texts*. Many scholars in the humanities have been faced with texts hundreds or thousands of years old written in lost or dying languages. The methods and techniques of philology have been employed for thousands of years in the analysis of the great classical texts of India, China, Europe to address this question: How can one articulate a relationship between the present and the texts of the past? In a recent article, Sheldon Pollock describes the history and future of philology, the paradigmatic discipline for most of the humanities and many of the sciences (e.g., evolutionary biology) like this: *"…philology is, or should be, the discipline of making sense of texts. It is not the theory of language -- that's linguistics -- or the theory of meaning or truth -- that's philosophy --*

but the theory of textuality as well as the history of textualized meaning. If philosophy is thought critically reflecting upon itself, as Kant put it, then philology may be seen as the critical self-reflection of language. Or to put this in a Vichean idiom: if mathematics is the language of the book of nature, as Galileo taught, philology is the language of the book of humanity." (Pollock, 2009).

Pollock attributes this idiom about mathematics and philology to Giambattista Vico, an early 18th century professor of rhetoric at the University of Naples. It is telling that this three hundred year old quip is still operative today. If we consider computer science, and thus AI as a sub-discipline of computer science, to be a science or a branch of mathematics, then we might understand older work in the discipline to be true, false, or a special case of a more general truth (e.g., as Galileo's ideas were later generalized by Newton and then Einstein). Or, we might have such an understanding if we were relatively unacquainted with contemporary research done in both the history of science and the social studies of science and technology. Yet, such a naïve understanding of the history of AI is what is the common understanding within AI itself. In 1985, the philosopher John Haugeland, coined the term "Good Old Fashioned AI" (GOFAI) to describe the first three decades of the field (Haugeland, 1985). GOFAI was largely developed using "symbolic systems" until the mid-1980s when "symbolic AI" was seen to have "failed." It was then "replaced" by subsymbolic, embodied, and statistical approaches to AI. To see this history told in more detail, one can consult most popular reference books and websites as well as the introductory chapters of many AI textbooks.

There are many problems with this common history of AI, but one most important to the topic of this chapter is this: the so-called "failed" ideas of symbolic AI addressed issues of narrative and practically any contemporary AI project that wrangles with the analysis and/or generation of stories is contingent on these so-called "failed" ideas of symbolic AI (e.g., Chambers and Jurafsk, 2009). In other words, symbolic AI never died, it was just incorporated in various ways into more contemporary work and relabeled. The automatic analysis of news stories is a good example. One of the first programs to automatically construct a summary of news stories was Gerald DeJong's 1979 dissertation program called FRUMP (DeJong, 1979). DeJong's Ph.D. was awarded for his research at the Yale AI Lab. His work inspired a subdiscipline known now as "information extraction," a kind of text-based data mining, and the basic data structure deployed in the newest of these systems is a variant of what in the 1970s and 1980s was called a "script" or a "frame." Analogously, most of the current technical proposals associated with what Tim Berners-Lee has dubbed the "Semantic Web" rely on techniques and technologies developed by researchers working on symbolic AI. Many other examples can be found in other areas of AI (e.g., natural language process) and computer science more generally (e.g., programming languages).

The kernel of an alternative to a "Galilean," science-and-mathematics-centered understanding of AI, is in the Vichean idiom. To quote two of the most prominent computer scientists Gerald Sussman and Harold Abelson: "*Underlying our approach to this subject is our conviction that 'computer science' is not a science and that its significance has little to do with computers."* (Abelson and Sussman, 1984, p. vii). Instead, Sussman and Abelson insist that computer science is a form of *"…procedural epistemology -- the study of the structure of knowledge from an imperative point of view, as opposed to the more declarative point of view taken by classical mathematical subjects*" (ibid.). In short, the alternative is to bring AI into the fold of the arts and the humanities.

From a Galilean perspective it's hard to understand why one might spend time with old texts. Why read Homer's *Iliad* if Warner Brothers produced a perfectly good movie in 2004 starring Brad Pitt and covering much of the same material? Newer is better, right? From a Vichean perspective the opposite opinion is equally clear: reading the classics is the core of a rigorous education. Less hyperbolically stated, most any artist or humanities scholar can see the value in studying and interpreting older works. In contrast, in science, it is commonly the case that the original texts of a subject (e.g., Galileo's writings on physics) are not read or interpreted in an introductory course.

The purpose of this chapter is to grapple with the contemporary significance of one of the "classics" of AI story generation, James Meehan's Tale-Spin program written for his 1976 Yale dissertation entitled *The Metanovel: Writing Stories by Computer* (Meehan, 1976).[2] The means to do this are means of the arts and the humanities: translation, interpretation, and imitation.

Micro-Talespin

[2] In the acknowledgements of his dissertation, Meehan states "*This work was supported in part by the Advanced Research Projects Agency of the Department of Defense and monitored by the Office of Naval Research under contract N00014-75-C-1111*" (Meehan, 1976, p. iv). Many projects of the Yale AI Lab were similarly funded. Searching the site http://www.dtic.mil/dtic/search/tr/tr.html for technical reports with the same ONR contract number (i.e., *N00014-75-C-1111) retrieves forty-five technical reports from Schank and his students. Meehan's is one of the forty-five.*

"*Today Tale-Spin is one of the most widely discussed digital fictions ever produced. It is not only a touchstone for computer science accounts of story generation but also broadly cited in writing about digital literature and the future of fiction*" (Wardrip-Fruin, 2008, p. 120).

At the beginning of my graduate studies I started engaging in a study of older computer programs in a practice I called a "literature review by critical reimplementation" (see Sack and Davis, 1994). In 1992, I began to work through a I book I knew as an undergraduate at Yale: *Inside Computer Understanding: Five Programs Plus Miniatures* (Schank and Riesbeck, 1981). When a dissertation was completed at the Yale AI Lab, former students were asked to reimplement the essentials of the dissertation program as a small "miniature" that could be used by others to study how the bigger programs worked. This practice resulted in a series of books of which *Inside Computer Understanding* (ICU) was the first (see also Riesbeck and Schank, 1989; and, Schank, Kass, and Riesbeck, 1994). I worked at the Yale AI Lab for three years after my undergraduate degree and spent much of my time then taking a dissertation "miniature" (Johnson, 1985) and scaling it back up to be faster, more efficient, and easier to modify and further extend (Sack, 1992). This experience taught me that the "miniatures" were not necessarily inferior to the original programs. Rather, they were frequently equally interesting, but pithier. This concision was often achieved by the Ph.D. graduates in collaboration with Christopher Riesbeck, an amazing programmer who was then a research scientist at Yale and the co-editor of the book.

All of the miniatures had names with a "micro" prefix thus, as I worked my way through *ICU* book, I eventually got to the Meehan's miniature, micro-Talespin. As I worked through the book, I was translating the micro programs into a contemporary programming language, Common Lisp (Steele, 1990) so that I could run, modify, and share the code. The original versions of the micro programs were written in an earlier dialect of Lisp, UCI Lisp, which, coincidentally, was designed and implemented by Meehan after Yale when he was a professor at the University of California, Irvine (Meehan, 1979). Programming languages often undergo rapid change and so code written even a decade ago can be impossible to run on today's machines. Such was the case then and such is the case now. This is the case for most any language, unless, ironically, the language becomes unpopular. Common Lisp became unpopular between the early 1990s and today and so my 1992 translation of micro-Talespin into Common Lisp is still works fine today.

In 2006, Noah Wardrip-Fruin was working on his book *Expressive Processing: Digital Fictions, Computer Games, and Software Studies* (Wardrip-Fruin, 2010). One chapter of that book is devoted to Tale-Spin and so, as part of his research, Noah dug up my 1992 translation and posted it to the Electronic Literature Organization (ELO) website. His post is a succinct introduction to the program that emphasizes its connection to the arts and humanities: "*James Meehan's Tale-Spin, created as part of his 1976 dissertation,* The Metanovel: Writing Stories by Computer*, was the first major project in the area of story generation. Like one of Calvino's invisible cities, it creates an alternate landscape in which the inhabits live in a manner evocatively different from our own — with all actions the result of plans, the locations of items only learned by convincing someone to tell you, and no one feeling an emotion without knowing it. Like Aesop's fables, Tale-Spin's view of human nature was communicated through the interactions of iconic animals. But unlike the worlds of Calvino or Aesop, Meehan's world wasn't simply described — it was made to operate. In fact, its operation, rather than its description, was Meehan's primary work. (The text describing the world was produced by a bare-bones language generation program, called Mumble, designed primarily to fit in the small amount of memory left on the Yale AI lab's computer system when Tale-Spin was already running.) … In 1981 a simplified version of Tale-Spin was published as part of the book Inside Computer Understanding: Five Programs Plus Miniatures. This version, Micro-Talespin, was then translated into Common Lisp (a programming language used in many artificial intelligence projects) by Warren Sack in 1992. It includes the settings for five default stories, simple text output from Micro-Mumble, and also the ability to interact with the simulated world. The ELO website now hosts Sack's version, which requires that the computer running it have Common Lisp installed. GNU CLISP is an implementation of Common Lisp that works on Unix, MacOS, and Windows machines. To experience Micro-Talespin, start Common Lisp, load Micro-Talespin, and then, at the "?" prompt, type: (micro-talespin-demo *story1*). Next, try starting up with one of the other five stories*." (http://eliterature.org/showcase/meehan-and-sacks-micro-talespin).

The text of my Common Lisp translation is linked off of Noah's introduction at the ELO site. I will refer to this translation from time to time. When I do, I will cite the line numbers of the code. To find the same lines of code, load the text of the program into an editor that supports line numbers. Most editors used by programmers support this. For example, I use the Emacs editor that is free software. The micro-Talespin program comprises 1723 lines (i.e., about 30 pages) of code and comments.

What does the program produce? Here is what I see when I follow Noah's instructions, load the program, and run it:

```
? (load "microtalespin.lisp")
;; loading file microtalespin.lisp ... loaded
? (micro-talespin-demo *story1*)
once upon a time ... joe was near the cave.  joe knew that joe was near the cave.  the water was near the river.  joe knew that the water was near the river.
one day, joe was thirsty.  joe wanted not to be thirsty.  joe wanted to be near the water.  joe went to the river.  joe was near the river.  joe drank the water. joe was not thirsty.  the end.
```

One is struck immediately by the lack of proper capitalization, the crude punctuation, and the lack of pronouns. But, as Meehan notes in his dissertation, the module, Mumble, he wrote to output from data structures to English was not his focus: "*It's a quick 'n' dirty program, written in day, and many of its parts do not correspond to the way humans speak, but it produces adequate, if somewhat verbose, sentences*" (Meehan, 1976, p. 200).

I also translated micro-Mumble into Common Lisp and the file containing micro-Talespin includes micro-Mumble (see line 1103). When the micro-talespin-demo function is executed (see line 298), it generates a story with micro-Talespin and then transcribes the story into English with micro-Mumble.

Consider the first two sentences in the story above: "joe was near the cave" and "joe knew that joe was near the cave." In the data structures of micro-Talespin these two sentences are written as the following parenthesized statements (see lines 79 and 80): '(world (loc (actor joe) (val cave))) and '(joe (loc (actor joe) (val cave)))

The many parentheses are due to the way that the Common Lisp programming language represents lists. Simple lists look like this: '(this is a list). Thus the structures shown above are lists of lists. In micro-Talespin, when the token 'world' appears at the beginning of a statement it means that the rest of the statement is true in the story world, the diagesis. When a character's name appears at the beginning of a statement (e.g., 'joe'), it means that the rest of the statement is something that that character knows. This is a basic construct that allows one to declare that different characters know different things about the world or may believe things that are not true about the story world. For example, one might state, with interesting consequences, that Joe is actually at the cave, but he thinks he is on the mountain: '(joe (loc (actor joe) (val mountain)))

Russian Formalism and French Structuralism

Meehan's decision to make a sharp distinction between his stated focus (Tale-Spin, a program to generate stories) and an auxiliary project useful but not essential for his project (Mumble, a program to render a story into English) is a decision that has historical precedents much older than AI. The Tale-Spin/Mumble divide is a division of labor institutionalized in linguistics and philosophy as a difference between semantics and syntax where semantics is the study of meaning and syntax is the study of lexical and grammatical form. Accepting this distinction allows one to imagine that work done one side of the divide can be done independently of work done on the other.

But, even if one does not believe the syntax/semantics divide to be absolute, an argument is still to be had that the lexical and grammatical decisions made in oral storytelling are not that important to their meaning. Anthropologist Claude Lévi-Strauss maintained *"[m]yth is the part of language where the formula* traduttore*,* tradittore *[translators are traitors] reaches its lowest truth value. … Whatever our ignorance of the language and the culture of the people where it originated, a myth is still felt as a myth by any reader anywhere in the world. Its substance does not lie in its style, its original music, or its syntax, but in the story which it tells*" (Lévi-Stauss, 1963, p. 210).

In short, Meehan's distinction between Tale-Spin and Mumble is an implicit commitment to a *structuralist or formalist analysis of narrative,* methods in which a distinction is made between the story and the discourse, the way the story is told (cf., Prince, 2003). Or, in the older vocabulary of Russian Formalism (cf., Jameson, 1972), the story/discourse distinction was, in the 1920s, rendered as fabula/syuzhet. In his dissertation, in an exceedingly short discussion of this approach to the analysis of folktales (in which he does not cite any of the Russian Formalists), Meehan acknowledges that *"[a] now-obvious project for future work is to define a theory of the sujet [alternative spelling], to define the rules for altering the temporal order of the events in a story*" (Meehan, 1976, p. 156).

If then one can accept this inattention to discourse or *syuzhet*, one must find or define the "atoms," the units from which a "molecular" story is composed. But, when Vladímir Propp, a Russian Formalist, wrote *Morphology of the Folktale* in 1928 his approach was to look at the "bonds" rather than the "atoms" of the folktales: "*What methods can achieve an accurate description of the tale? Let us compare the following events: 1. A tsar gives an eagle to a hero. The eagle carries the hero away to another kingdom. 2. An old man gives Súcenko a horse. The horse carries Súcenko away to another kingdom. 3. A sorcerer gives Iván a little boat. The boat takes Iván to another kingdom. 4. A princess gives Iván a ring. Young men appearing from out of the ring carry Iván away into another kingdom, and so forth. Both constants and variables are present in the preceding instances. The names of the dramatis personae change (as well as the attributes of each), but neither their actions nor functions change. From this we can draw the inference that a tale often attributes identical actions to various personages. This makes possible the study of the tale* according to the functions of its dramatis personae *[emphasis in the original]"* (Propp, 1968, pp. 19-20).

Propp's "functional" analysis of folktales yielded the surprising insights that (a) there are only a limited number of functions (by his count 31); (b) the sequence of functions is always the same, regardless of the tale; and, thus, (c) all fairy tales have the same structure (Propp, 1968, pp. 21-23; and. p.64). Note that Propp's function are not simple actions; they are, instead, actions with context. Thus, it is not enough to know that one character killed another in order to determine the narrative function of the act of killing. A killing could be an act of villainy or heroism depending on its position within the larger series of functions that constitute the story.

Propp's analysis was restricted to an examination of 450 previously catalogued fairy tales (see Aarne, 1961; and, for a contemporary update to this system, see Uther, 2004) While Propp analyzed the actions and functions of the characters in the stories, Aarne's classification was based on the sets of motifs or character types that appeared in the tales. Thus, for instance, the tales Aarne numbers 1-299 are labeled "Animal Tales" and are subcategorized as, for example, "The Clever Fox or other animal" (numbers 1-69). The tales Propp analyzed are numbered 300-749 and categorized as "Fairy Tales" with subcategories "Supernatural Opponents," "Supernatural or Enchanted Relatives," "Supernatural Tasks," "Supernatural Helpers," "Magic Items," "Supernatural Power or Knowledge," "Other Stories of the Supernatural."

Metaphically, Aarne's classification is based on the "atoms" of the stories and Propp's on the "bonds" of the stories. Propp's analysis states that, if an action takes place in a folktale, it should be one of the 31 functions he has catalogued. However, he does not state that all tales contain 31 actions: tales can contain only a subset of the functions. Propp's sequence starts with these three functions: (i) Absentation: One of the members of a family absent's himself from the home. (ii) Interdiction: An interdiction is addressed to the hero. (iii) Violation: The interdiction is violated (Propp, 1968, pp. 26-27). At the end of the sequence of functions are these: (xxx) Punishment: The villain is punished. (xxxi) Wedding: The hero is married and ascends the throne (Propp, 1968, p. 63). In between, obviously, the hero is faced with a series of trials, aided and challenged by other characters.

For Propp, character types can be distinguished according to the functions they perform, not according to their appearance or attributes. Along with the thirty-one functions, Propp notes seven "spheres of action" (Propp, 1968, p. 79), each of which corresponds with a character type. Heroes, for instance, depart on a search and the beginning of a tale and are wed at the end of a tale. Along with heroes, Propp identifies villains, donors, helpers, princesses, dispatchers, and false heroes. Characters can play one or more types. And, one or more characters can play a type. Subsequent developments of narrative theory have elaborated and generalized these types in order to fit other story corpora. For example, in 1966 Algirdas Julien Greimas, a professor of semiotics, further developed Propp's functions and "spheres of action" into what he called an "actantial model" (Greimas, 1973). These further developments are still with us today. For example, sociologist and philosopher Bruno Latour and his colleagues have developed actor-network theory (ANT) based on Greimas' "actants." ANT has been used extensively to analyze the narratives of science and technology (e.g., Law and Hassard. 1999).

The anthropologist Lévi-Strauss elaborated Propp's method to analyze myths. He insists that *"[t]he true constituent units of a myth are not isolated relations but bundles of such relations,…"* (Lévi-Strauss, 1963, p. 211). He found Propp's series of 31 functions lacking insofar as it failed to take into account how many functions are similar to others and so how tales frequently feature repeated functions. Lévi-Strauss claims, just as in mathematics a new function can be composed using a given function (e.g., given f(x) one can compose its inverse as 1/f(x) or its negation as –f(x)), narrative functions can be identified as permutations or compositions of other narrative functions. Grouping these function permutations together, one can identify the "bundles of relations" he described earlier (Lévi-Strauss, 1984, pp. 183-184). For example, one might group together the departure of the hero with the return of the hero as inverses.

In his methodological description of the structural study of myth, Lévi-Strauss gives the nitty-gritty of how to do what he prescribes: "*The technique which has been applied so far by this writer consists in analyzing each myth individually, breaking down its story into the shortest possible sentences, and writing each sentence on an index card bearing a numbering corresponding to the unfolding of the story*" (Lévi-Strauss, 1963, p. 211). He then takes these cards and arranges them on a table into rows and columns. Each column corresponds to a function or permutation of a function. Each row can be read left to right as it comprises a series of one or more chronologically sequenced actions. His result is a large grid of index cards for a given variant of a given myth. To compare variants he does the same for each attempting to make the number of columns (and their corresponding functions) equal and so the comparison between variants of a myth is a complicated analysis of what is essentially a three dimensional table. Lévi-Strauss elaborates: "*It should be emphasized that the task of analyzing mythological literature, which is extremely bulky, and of breaking it down into its constituent units, requires team work and technical help. A variant of average length requires several hundred cards to be properly analyzed. To discover a suitable pattern of rows and columns for those cards, special devices are needed, consisting of vertical boards about six feet long and four and a half feet wide, where cards can be pigeon-holed and moved at will. In order to build up three-dimensional models enabling one to compare variants, several such boards are necessary, and this in turn requires a spacious workshop, a commodity particularly unavailable in Western Europe nowadays. Furthermore, as soon as the frame of reference becomes multi-dimensional (which occurs at an early stage,…) the board system has to be replaced by perforated cards, which in turn require IBM equipment, etc.*" (Lévi-Strauss, 1963, pp. 228-229).

Databases

Understanding a story, in Lévi-Strauss' terms, is tantamount to breaking it down into a multi-dimensional table and defining a set of relations between the rows and columns of the tables. In contemporary terms this is almost exactly what one does to define a relational database; i.e., the form of database used ubiquitously on computers and networks for applications of government, industry, and leisure. Websites, credit agencies, national intelligence agencies all use such databases. Coincidentally, the inventor of the relational model of database management was a researcher at IBM (Codd, 1970). Clearly, with the plethora of data we have today, the "frame of reference" has become “multi-dimensional” now, decades after Lévi-Strauss’ comments, and so there are no longer index cards, or even perforated cards, in the rows and columns of the relational databases; rather, all such rows and columns are declared as electronic data in the computers (IBM and otherwise). That contemporary relational databases essentially recapitulate the form and functions of a previous paper-based, bureaucratic technology should come as no surprise. This is a common development in computer technology; consider the computer “desktop,” “folders,” “trash bins,” “files,” actions like “cutting,” and “pasting,” etc. (cf., Sack, 2011).

Without diminishing the accomplishments of Lévi-Strauss and Propp, it may be worthwhile to consider the possibility that what counts as understanding at any given historical period is the ability to index and articulate previous forms of knowledge in the tropes of the latest media technology. To understand in a society where script is the latest technology would entail writing down oral stories (cf., Havelock, 1962). To understand in a print society would be to apply the techniques of print to the texts created by scribes (e.g., the introduction of chapters, footnotes, a table of contents, subject indices, and the whole apparatus of the library; cf., McLuhan, 1962). To “understand” in a society that employs computer programming would be to transcode printed texts into programs (cf., Lyotard, 1979). Following this line of reasoning, the database and the narrative are not oppositional forms as some have posited (e.g., Manovich, 1999). Instead, databases are just one necessary component to the contemporary understanding of narratives.

In micro-Talespin, the state of the story world and the beliefs of the characters and their relationships are recorded in a database where each entry in the database is a data structure defined as a list or lists of lists, like those shown above. The database is queried with many special-purpose functions such as memquery (line 825), knows (line 806), knows-loc (line 814), knows-owner (line 817), knows-if (line 820), relate (line 861), has-goal-of (line 838), and several other functions. There are another set of functions for asserting statements into the database and retracting statements from the database. Part of what makes micro-Talespin difficult to understand is the number and seeming diversity of these functions to access and modify the database. I will discuss, in a following section, how these functions can all be replaced with essentially three: assert, retract, and query.

In addition to these three basic database operators, it is convenient to have a means to compose compound queries and compound assertions, what can be called,

respectively, deduction rules and production rules. For example, it could be useful to have a rule to deduce whether or not a character is confused about its geographical location. Such a statement could be represented like this:
(confused (joe (loc (actor joe) (val ?where)))),
where a variable is the statement is rewritten with a prefixed "?". To implement such a rule we need to have a conjunctive query where first the database is consulted to see where the character is actually located (world (loc (actor joe) (val ?actual))); then the database is queried to see where the character thinks it is located (world (loc (actor joe) (val ?believed))); then the ?actual location is compared to the ?believed location; if they don't match (e.g., when Joe is actually in his cave, but he thinks he is on the mountain), the rule returns true, indicating that the character is indeed confused about its geographical location; otherwise, the rule returns false. Micro-Talespin does not implement many deduction rules, but those that it has are written as special-purpose Lisp functions.

Production rules are different from deduction rules. When a statement is added to the database, all of the production rules are checked to see if other statements also need to be added to the database, statements that are consequences of the first statement (see assert-fact (line 497), forward-chain (line 505) and conseqs (line 515)). In micro-Talespin each action that a character can perform has, associated with it, a Lisp function that enumerates all of the consequences of that action. For instance, if a character carries an object from one place to another, the consequences are that both the character's position and the position of the object are changed to the destination (ptrans-conseqs at line 689).

While it is perfectly feasible to state deduction and production rules as ad-hoc pieces of Lisp code, rules are much easier to understand, debug and extend if they can be written as simple if-then rules. I will discuss how that can be done in a later section of this chapter.

Functions

Lévi-Strauss' discussion of Propp employs a notation for functions that we are all familiar with from algebra. But, in most algebra classes variables are meant to stand in for numbers and functions define a relationship between sets of numbers; e.g., the successor function, $f(x) = x + 1$. Consequently, it is easy to combine functions with arithmetic operators like +, -, * and /. So, if $g(x) = x + 2$, then it perfectly clear what a definition like this means: h(x) = f(x) * g(x). But, Propp's functions are not defined on numbers but rather on texts, so we must interpret Lévi-Strauss' comments as metaphorical, rather than as a literal, technical proposal. To understand technically, namely computationally, the meaning of Propp's functions, we must turn to linguistics.

In his book, Propp assigns each of his 31 functions a one or two letter abbreviation and uses superscripts and some special symbols to indicate variants of a function. He states that all 450 folktales he has examined have a single morphology that can be described using this schema, or formula (Propp, 1968, p. 105):

$$\mathrm{ABC}\uparrow\mathrm{DEFG}\ \frac{\mathrm{HJIK}\downarrow\mathrm{Pr\text{-}Rs}^{0}\mathrm{L}}{\mathrm{LMJNK}\downarrow\mathrm{Pr\text{-}Rs}}\ \mathrm{QExTUW}^{*}$$

So, this describes a story form in which function A precedes function B precedes function C, etc. The horizontal line in the middle is not a division sign, as ordinary algebra might lead one to believe. Rather, it designates a branch point after function G which may, alternatively, be followed by function H or function L. The two branches then join again at function Q.

What is left unresolved in Propp's exposition is what it means to position one function next to another; e.g., AB. It is only clear that juxtaposition means chronological sequence (i.e., A comes before B). Yet, it is equally clear that each of these functions cannot be entirely independent of one another if they are meant to describe a coherent narrative that connects one part of the story to another. For example, each of the functions could have, at least, one parameter for each of the dramatis personae (heroes, villains, princesses, etc.), so that, for instance, the hero at the beginning of the story is not forgotten. Such a suggestion might imply a notation like this: A(hero,villain,princess)B(hero,villain,princess), etc. Yet, even then, this notation could lose important information. What if, halfway through a story, the hero becomes a villain? There is a lot of information that is presupposed in each function and that information is contingent upon what has happened in the story previous to the function. In short, we need a better notion, one that allows for the coding of narrative context.

As a linguistics graduate student in 1964, George Lakoff wrote a remarkable analysis of Propp's work (Lakoff, 1972) using the then-new formalism of linguistics introduced by Noam Chomsky: transformational grammar (Chomsky, 1957). Addressing some of the same problems discussed in the paragraph above, Lakoff argues that Propp's formalism is inadequate to the task of describing the form of folktales because it does not have a means to represent context. However, context can be addressed using the formalisms of transformational grammar. Lakoff rewrote some of the core aspects of Propp's work as a transformational grammar.

Although some linguists currently working in Chomskyan tradition are still concerned with the theoretical status of transformational grammars (see Chomsky, 1995), Lakoff is no longer so concerned. Lakoff's current thoughts about the topic of his 1964 paper can be found here (Lakoff and Narayanan, 2010). Nevertheless, the computational adequacy of a formalism with the descriptive adequacy to generate Propp's folktales can still be outlined using the essential features of transformations: "*Transformations are, roughly, combinations of permutations, additions, deletions, and substitutions. For a precise formulation see Chomsky (1961).*" (Lakoff, 1972, note 13). In other words, to introduce context and so coherence between Propp's function one needs the means of (a) permutation: to re-order their sequence; (b) substitution: to re-write abstract functions in more specific terms (which, ultimately, are rendered in English, or some other natural language); and, (c) additions and (d) deletions: to add and delete assertions from a database that records the current state of the story; thus, for instance, the assertion that a character that perishes at the beginning of the tale needs to be recorded so that same character does not appear alive later in the tale.

Sheldon Klein was a professor of Computer Science at the University of Wisconsin who worked with Lévi-Strauss in Paris at the Ecole des Hautes Etudes en Sciences Sociales during the year 1976-1977 (see http://www.cs.wisc.edu/news/sklein.memorial.pdf). Starting in the 1960s, Klein and his students wrote a number of story generators. In 1976, Klein and his colleagues wrote a computer program to simulate the narrative functions of Propp and Lévi-Strauss (Klein et al., 1976). To implement their simulation, Klein et al. developed a rule language that, although it was not a transformation grammar *per se*, provided the essentials: permutations, substitutions, additions, and deletions. Since they cite Lakoff's paper, this is certainly not coincidental. The system generated plausible versions of sixty of the Russian folktales analyzed by Propp and four indigenous Brazilian myths analyzed by Lévi-Strauss in his book the *Raw and the Cooked* (Lévi-Strauss, 1983). As can be seen from a review of their code (included in the published article), Klein et al.'s advance over Lakoff was in the details. Where Lakoff speculated that such a program could be written, Klein et al. wrote the program and, in so doing, identified a number of difficult details that were overlooked in the more theoretical discussions of Propp (e.g., Klein et al., 1976, p. 99).

Meehan cites this work by Klein in his dissertation. In fact, within the dissertation, Klein's work is the only work on story generation that Meehan reviews. Nevertheless, Meehan is extremely dismissive of Klein's work: "*This isn't a good model of either stories or storytelling because there's no implicit causality in the stories: nothing that says why one thing implies another, nothing that says what people need, nothing that says why people make the decisions they do, nothing that explains people's reactions to one another, nothing that represents the complexities of the physical world. The purpose of Tale-Spin is to address exactly these issues*" (Meehan, 1976, p. 9).

This strong rebuttal of Klein's work is unwarranted for two reasons. First, if we look into the code of Meehan's system, he too reinvents what is essentially a form of transformational grammar to operate as the main mechanism of Tale-Spin. This mechanism, in Meehan's descriptions, is called "planning," a term I will unpack shortly in more detail.

Second, Meehan states the purpose of Tale-Spin as an examination of various forms of causality (physical, logical, social, psychological). Yet, most students of narrative would agree that the backbone of stories is that which links events together that have no obvious or necessary connection: "*Narrative: The recounting … of one or more real or fictitious events communicated by one, two, or several … narrators to one, two, or several … narratees. … In order to distinguish narrative from mere event description, some narratologists … have defined it as the recounting of at least two real or fictive events (or one situation and one event), neither of which logically presupposes or entails the other.*" (Prince, 2003). Meehan's stated focus on mundane forms of causality and implication is probably the reason why most of what is output by Tale-Spin is difficult to read as a story. Instead, the output usually looks like a "mere event description"; e.g., the example output from micro-Talespin shown above which was, essentially, this: Joe is thirsty, he goes to get some water.

Plans

"*Plans are worthless, but planning is everything.*" (Eisenhower, 1957)

"*At the heart of Tale-Spin is a problem solver, a program which implements a new theory of planning. Accordingly, the stories Tale-Spin produces are essentially accounts of what happened during the course of solving one or more problems. This is consistent* *with the theory that all stories are about problems. … The theory of planning is based on ideas of Schank and Abelson.*" (Meehan, 1976, p. 39).

Planning, in Tale-Spin, is carried out using a number of explicit problem-solving procedures for transportation (dprox), acquisition of objects (dcont), acquisition of information (dknow), transfer of information (tell), persuasion (persuade), bargaining (bargain), and requesting (ask). The parenthesized terms refer to the ideas of Schank and Abelson (1977). Micro-Talespin implements a number of these procedures as Lisp functions. Each such function composed in one of three ways: either (a) it is an action, a conjunct of preconditions and a set of additions and deletions that modify the database, if the preconditions are satisfied; or, (b) it is a method, a sequence of steps that, after they have been executed, solve the given problem; or, (c) it is a disjunct of alternative ways of solving the given problem.

For example, "ask" is a plan (see line 135) with two preconditions: (1) the character making the request has to believe the queried character is not deceitful; and, (2) the queried character has to like the character making the request. If these two preconditions are met, then a "tell" procedure is run.

The "tell" procedure is a method (of type (b)) that entails two steps: to tell a character something first go to them (dprox), then say (mtrans) the information you have to tell (see line 468).

The "dprox" procedure (line 418) is a disjunct of type (c). To move a character or an object to the location of a second character of object, one can either (1) find out the location of the second character or object (dknow) and then go there (ptrans); or, (2) if it is a character that is to be moved, try to persuade that character to move themselves to the desired location.

Notice that these procedures "bottom out" in a set of "primitive actions" (e.g., mtrans (an abbreviation for "mental transfer") and ptrans (an abbreviation for "physical transfer")). These "primitive actions" were the topic of Roger Schank's own dissertation (cf., Schank, 1972). He maintained that all verbs could be composed from 11 primitive actions. In Tale-Spin, when a primitive action is asserted into the database, a special procedure is run that adds all the consequences of that action into the database too. For example, ptrans-conseqs (line 689) updates the position of the character or object that has moved. These kinds of procedures can be phrased as production rules, as was discussed above in the section on Databases.

Gathering all of these requirements together, one can say that any implementation of a story generator like Tale-Spin requires the means to (i) assert statements into a database; (ii) retract statements from a database; (iii) query a database; (iv) compose deduction rules; (v) compose production rules; (vi) define actions; (vii) define methods; and, (viii) define alternatives (i.e., disjuncts of actions or methods).

If one compares these requirements to the definition of transformations given above (in the discussion of Lakoff's reformulation of Propp's functions in a transformational grammar), they can be seen to match up. Additions and deletions are supported by assertions to and retractions from the database. Permutations can be articulated with alternative orderings or sets of actions or methods. Substitutions are simply a form of abstraction that can be enacted by defining methods that call sequences of other methods or actions.

To gloss over the differences, in this manner, between Propp, Lévi-Strauss, Klein, Meehan, Lakoff, Schank and Abelson, is to engage in what Harald Abelson and Gerald Sussman have called "metalinguistic abstraction": "*Programming is endowed with a multitude of languages. … These languages have means of combination and abstraction, such as procedure definition, that are appropriate to the larger-scale organization of systems. …* Metalinguistic abstraction -- *establishing new languages -- plays an important role in all branches of engineering design. It is particularly important to computer programming, because in programming not only can we formulate new languages but we can also implement these languages by constructing evaluators. … An evaluator (or interpreter) for a programming language is a procedure that, when applied to an expression of the language, performs the actions required to evaluate that expression. It is no exaggeration to regard this as the most fundamental idea in programming: The evaluator, which determines the meaning of expressions in a programming language, is just another program. To appreciate this point is to change our images of ourselves as programmers. We come to see ourselves as designers of languages, rather than only users of languages designed by others. In fact, we can regard almost any program as the evaluator for some language.*" (Abelson and Sussman, 1996).

Interpretation

What Abelson and Sussman advocate – and illustrate stunningly – in their textbook, *The Structure and Interpretation of Computer Programs* (known in geekdom as “SICP”), is an interpretation of interpretation that for most contemporary humanities scholars probably seems like a farfetched idea: given a set of problems, invent a new language to better express those problems, then, interpret these problems in the newly-invented language. This project has a long history in philosophy and mathematics documented in, for example, novelist and linguist Umberto Eco’s book *The Search for the Perfect Language* (1995).

But, in a medium, like computer programming, where the design and implementation of a new language is tractable, this approach is not farfetched. Rather, from a humanities perspective, this takes us from a focus on the concerns of philology (reading), to a focus on the concerns of grammatology (writing): How can old texts (computer programs) best be rewritten (not just read)? The answer is far afield from my 1992 translation of micro-Talespin from an old language (UCI Lisp) to a new dialect of that same language (Common Lisp).

Instead, what Abelson and Sussman urge is a close reading of a specific computer program that fosters an interpretation of that program as if it was an incomplete evaluator of some other language. Then, the challenge is to define a complete evaluator for the language implied by the problem. Specifically, the eight requirements for Tale-Spin-like systems enumerated above (from assertions to alternatives) should be considered as basic building blocks for a language within which it will be straightforward to implement the specifics of Meehan’s program (e.g., the functions for dprox, mtrans-conseqs, etc.). If done well, this language should also provide a medium with which it will be straightforward to implement Propp’s functions, Lévi-Strauss’ “bundles of relations,” the transformational grammar of Lakoff/Chomsky, and so forth.

The outcome of such an interpretation of Tale-Spin will be a computer programming language and a rewrite of Tale-Spin in that new computer programming language. Performing such an interpretation is analogous to translating poetry from, say, French into English. A lot of trial and error is entailed. Sometimes one has to coin new words or odd phrases to be true to the original. If the author or the original has been translated before, one can frequently borrow ideas from previous translations. The outcome is a new work that strongly resembles the original, but cannot be the same. Hopefully, it is as good as the original. If one is lucky, it is better.

Here, in this chapter, I can present my interpretation, but the work that went into the interpretation – the trial and error, the borrowing, etc. – will mostly remain invisible because many of those details are too boring or too intricate to present. The remainder of the chapter will be devoted to this "metalinguistic" interpretation of Tale-Spin. First, some details about syntax will be described. Second, the eight requirements enumerated above will be addressed. Third, the re-definition of Tale-Spin and Mumble will be sketched out. Fourth, some examples of output from the system will be presented.

The reader can also consult and play with my code directly online. Load this web page http://fdm2.ucsc.edu/~wsack/JavaScript/Spinner/. The code in JavaScript and can be found in eight files: (1) utilities.js (2) llpl.js (3) shop.js (4) qa.js (5) initial.js (6) spinner.js (7) mumbler.js (8) spinner.html. It comprises about 5000 lines of code; i.e., about 90 pages of code of which mumbler.js is about 12 pages and spinner.js is about 17. To view the output from the system, open the web page spinner.html and then view the JavaScript Console in the browser window. For example, in the Chrome browser, one selects the "View" menu, then the "Developer" sub-menu then the "JavaScript Console" sub-sub-menu. In Firefox, install the Firebug plug-in (available here: http://getfirebug.com/) then, once it is installed, in Firefox first select the "View" menu then the "Firebug" sub-menu. The JavaScript Console will appear in a tab in the lower portion of the web page. As I write this, only the Chrome browser seems to be fast enough to execute the code.

JSON

Currently, the *lingua franca* of data structures exchanged between websites and programming languages is JSON, the JavaScript Object Notation (http://www.json.org/). In Lisp, data structures are coded as lists, as shown above (e.g., '(this is a list)). Other languages, have a variety of syntactic constructs for defining data structures. Today, most programming languages suppose JSON, even languages that are not JavaScript. JSON objects have a very simple syntax. The empty object is denoted like this: {}. Objects can be filled by zero or more pairs. A pair is written like this: "name": "warren sack", or this "name": ["warren","sack"], or this "name": {"first": "warren", "last": "sack"}. So, an object describing someone can look like this:

```
{"name": {"first": "warren", "last": "sack"},
 “job”: “professor”,
 “children”: ”: [{“first”: “felix”, “last”: “sack”}]
 “spouse”: “jennifer”}
```

All of the data structures in Spinner will be defined in JSON syntax. However, there is one extra constraint on the Spinner data structures. We will call these data structures terms and they will be restricted to having one and only one pair in them. However, as can be seen above (with the definition of the “name” pair) JSON objects can be nested, so there is no loss of expressivity. To define the JSON structure above as a term, one would write it like this:

```
{“person”: {“name”: {“first”: “warren”, “last”: “sack”},
            “job”: “professor”,
            “children”: [{“first”: “felix”, “last”: “sack”}]
            “spouse”: “jennifer”}}
```

In addition to this constraint on JSON sytax, there is also one extension to that syntax: unification variables. A unification variable is written as a string prefixed with a question mark, like this: “?what”. Variables can appear on the right hand side of any pair (but not on the left hand side).

Unification

Given two terms, each of which may include zero or more unification variables, the function unifyPatterns (in the file utilities.js), will try to match the predicates together. unifyPatterns takes three arguments: two terms and a list of variable bindings. Let us set the bindings to the empty object, to start with. Type the following into the JavaScript Console after you have loaded spinner.html (note that the “>” is the prompt from the JavaScript interpreter, not something you type in:

```
> var bindings = {};
```

Now, try this:

```
> utils.unifyPatterns({"is": {"performer": "?name", "role": "?part"}},
                      {"is": {"performer": "joe", "role": "bear"}},
                      bindings);
```

The response from the JavaScript interpreter should be true. Now try typing this:

```
> utils.pp(bindings);
```

You should then see this:

```
{
  "?name":  "joe",
  "?part":  "bear"
}
```

In other words, after unification, the variable “?name” has been bound to the value “joe” and the variable “?part” has been bound to the value “bear”. Both terms can contain variables and so be patterns. So, switching the position of the two terms, yields the same result, as does this, where there is a variable in the first term and a variable in the second term, rather than no variables in the first and two in the second.

```
> utils.unifyPatterns({"is": {"performer": "joe", "role": "?part"}},
                      {"is": {"performer": "?name", "role": "bear"}},
                      bindings);
```

Here is something to watch out for: if two terms have the same left hand side (e.g., in the examples above, that is “is”), then they will unify even if one term does not have all of the pairs of the other term. Thus, this returns true:

```
> utils.unifyPatterns({"is": {"role": "?part"}},{"is": {"performer": "?name"}},bindings);
```

This can be the source of difficult bugs if one mistakenly writes something in JSON that is not a term because two unlike JSON objects (that are not terms) can unify. So, this returns true:

```
> utils.unifyPatterns({"a": "1", "b": "2"},
                      {"x": "3", "y": "4"},
                      bindings);
```

Two terms will not unify if they have a matching pair where the right hand side of the pair does not match. Thus, this will return false:

```
> utils.unifyPatterns({"is": {"performer": "joe", "role": "?part"}},
                     {"is": {"performer": "irving", "role": "bear"}},
                     bindings);
```

Database

As is the case for Tale-Spin, facts about the world are recorded as statements in a database. To create a new instance of the database system, first create a copy of the utilities:

```
> utils = makeUtilities();
```

Then use this command

```
> ds = makeInterpreter(utils);
```

And, initialize the database contents to contain nothing:

```
> ds.initializeDatabase([]);
```

The variable ds contains a database interpreter with a number of commands the three most important of which are assert, retract, and query. These commands are all defined in the llpl.js file. To record “Josephine is a bear” one could type this to the JavaScript prompt:

```
> ds.assert({"is": {"performer": "josephine", "role": "bear"}});
```

Now, the database can be queried to see ?who is a bear:

```
> ds.query({"is": {"performer": "?who", "role": "bear"}});
```

And, the response indicates there is only one bear in the database, Josephine:

```
{"is": {"performer": "josephine", "role": "bear"}}
```

A term can be removed from the database with the retract command; e.g.,

```
> ds.retract({"is": {"performer": "josephine", "role": "bear"}});
```

In a previous section of this paper, *relational databases* were mentioned in the course of a discussion of Lévi-Strauss' methodology for discovering the structure of a myth. The Spinner database is not a relational database of rows and columns, but rather a store of JSON statements. This is what is now commonly called a *document-oriented database* or sometimes a *NoSQL database*.

Production Rules

It is frequently the case that one term implies many others. For example, when we assert that someone is a bear, we might also want to assert that that someone has fur, teeth and claws. To do so in the Spinner database, one can write a production rule. At the top of the production rules is term to be matched. If a new assertion matches the top of the rule, then the other terms listed in the rule (the consequents) are also asserted into the database.

```
{"-->": {"is": {"performer": "?bear", "role": "bear"},
         "consequents": [{"possesses": {"owner": "?bear", "possession": "fur"}},
                         {"possesses": {"owner": "?bear", "possession": "claws"}},
                         {"possesses": {"owner": "?bear", "possession": "teeth"}}]}}
```

One production rule can trigger another. So, for example, we might have one rule that states that if someone is a bear, they are also a mammal. And, then a second rule that states that if someone is a mammal, they are also warm blooded. If such was the case, asserting that Josephine is a bear would result in the additional assertions that she is a mammal and that she is warm blooded.

Deduction Rules

One might state that Josephine is at the cave using a term like this:

```
{"positioned": {"theme": "Josephine", "goal": "cave"}}
```

And, that Josephine is carrying a fish could be expressed like this:

```
{"carries": {"agent": "Josephine", "theme": "fish"}}
```

But, then where is the fish? You and I know that, since Josphine is carrying it, the fish is wherever Josephine is. One could devise a means to update the position of everything a character is carrying every time a character moves, or one could write a deduction rule so that the position of a carried item could be deduced when needed. Here is a deduction rule to do that. It states that if a character is carrying something, then that something is positioned at the same place as the character.

```
{"<--": {"positioned": {"theme": "?x", "goal": "?place"},
         "and": [{"carries": {"agent": "?character", "theme": "?x"}},
                 {"positioned": {"theme": "?character", "goal": "?place"}}]}}
```

Deduction rules start with an arrow that points to the left (<--) while production rules start with an arrow pointing to the right (-->). Production rules cause a set of assertions to be added to the database (the consequents of the rule). Deduction rules do not assert anything into the database. They simply determine if a term can be deduced from the terms that are already in the database. The conclusion of a deduction rule is listed first. The body of the rule is listed second following the “and”. The body of the rule is simply a list of queries into the database (i.e., a conjunctive query). If all of the queries in the body of the rule return successfully, then the conclusion of the rule is said to be true. Note that the body of the rule can call other deduction rules.

Also, deduction rules can have multiple definitions, thus providing alternative ways of deducing a term. For example, in addition to the rule above, one might also state that if someone is a bear, then it can be assumed that they are in the cave:

```
{"<--": {"positioned": {"theme": "?x", "goal": "cave"},
         "and": [{"is": {"performer": "?x", "role": "bear"}}]}}
```

The addition of such a rule may allow us to deduce that Josphine is in several different places. This may be useful if we are trying to generate possible places to look for her. Or, it may be problematic if no characters are suppose to be in two places at once. It all depends upon what the other rules look like; i.e., the other rules that employ these rules.

In computer science terms, the Spinner deduction rules are implemented using a form of Horn clause resolution (Robinson, 1965) and so the Spinner database rules are essentially the same thing as the Prolog logic programming language, a programming language invented in the 1970s (Colmerauer and Roussel, 1993).

Consequently, five of the eight criteria for Tale-Spin-like story generators can be encapsulated in a programming language interpreter that is essentially an implementation of a logic programming language, like Prolog. It allows one to (i) assert statements into a database; (ii) retract statements from a database; (iii) query a database; (iv) compose deduction rules; and, (v) compose production rules.

Planning

What remains of our list of criteria are these: the means to (vi) define actions; (vii) define methods; and, (viii) define alternatives (i.e., disjuncts of actions or methods). Recall that these actions and methods are the core of what Meehan saw as the core of his project: to implement a new theory of planning (Meehan, 1976, p. 39). Furthermore, Meehan saw planning in Tale-Spin as a cognitive simulation, a step-by-step copy of how people go about the task of creating a story. This claim – that Tale-Spin is a cognitive simulation – was taken seriously when it was made in the 1970s. Today, this claim would be a tough sell in the journal of *Cognitive Science* (a journal which Roger Schank co-founded).

However, the Abelson's and Sussman's path of metalinguistic abstraction provides us with a different possibility. The question posed is not this: What does a cognitive simulation of storytelling look like? Instead, the question to be asked is this: In what sort of a programming language can processes (specifically, methods and actions) be written so that Tale-Spin-like stories can be computed? Meehan's answer to this question, and the answer still current in the literature of narrative intelligence is this: a planning language is the right choice in which to write a story generator.

Following Meehan and then two other students of Roger Schank: Natalie Dehn (see Dehn, 1981) and Michael Lebowitz, a few years later (see Lebowitz, 1987) an extensive literature has grown around the idea that stories are best represented as plans where plans are sequences of actions that have an expected outcome (see, for example, Young, 1999; Riedl and Young, 2004).

The history of this literature was quickly sketched in a recent paper: "*With the development of new media, such as Interactive Storytelling (IS) and computer games, a major new application area for AI planning is emerging. In this area, planning technology is used to generate narratives for entertainment systems that feature 3D interactive presentation of the narrative using animations. This approach has its roots in the adoption of planning as a technology for virtual agents which was later transferred to reasoning about virtual actors (Geib, 1994). It was first proposed for IS in (Young, 2000) and since then it has emerged as the core technology for IS prototype systems (Cavazza, et al., 2007;Riedl and Young, 2010). In addition, planning has been used in recent computer games, including FEAR and KILLZONE, for controlling the behaviour of non-player characters"* (Porteous et al., 2011).

Cavazza and Pizzi have also written a concise introduction to narratology for artificial intelligence researchers (Cavazza and Pizzi, 2006). So, why is this seen as a natural fit from a technologist's point of view? I.e., the fit between planning and narratives? Recall the short definition of narrative by the narratologist Gerald Prince cited above: "*…the recounting of at least two … events … neither of which logically presupposes or entails the other*" (Prince, 2003, p. X). And, what causes an event? Some action is usually the cause of an event. Thus, plans, seen as sequences of actions, if recounted in the past tense, might be considered to be a good representation of story plots, which are sequences of events.

In the literature of AI and cognitive science, plans were seen as a cognitive construct at least by the time of the publication of the book *Plans and the structure of behavior* in 1960 (Miller, Galanter and Pribram, 1960). They were seen as analogous to, or even equivalent to computer programs (that encode sequences of actions). The earliest planning algorithms were implemented in the late 1950s and run as computer programs to solve logic puzzles, prove mathematical theorems, and play games, like chess (see Newell, Shaw and Simon, 1959). For a detailed history of planning, see chapter 8 of Phil Agre's book *Computation and Human Experience* (Agre, 1997). In the contemporary literature of planning (cf., Ghallab, Nau and Traverso, 2004) plans are sometimes posited as cognitive constructs, but, more frequently, they are seen simply as a technology. If one chooses the latter point of view, it is possible to think of planning systems as a genre of programming languages. This allows one to reconsider Tale-

Spin, not as a cognitive simulation, but as a partial implementation of a programming language evaluator, a planner.

Actions – or as they are more commonly described in the literature, operators – in planning systems are commonly represented using what is called a STRIPS notation (Fikes and Nilsson, 1971). Actions, in this notation, have (a) a set of preconditions that must be true before the action can take place; (b) a set of additions that are terms asserted into the database after the action has taken place; and, (c) a set of deletions that are terms retracted from the database after the action has taken place. In JSON, one can write an action like this:

```
{"action": {"description": "fly from one place to another",
            "task": {"flies": {"self_mover": "?self_mover", "source": "?source",
                               "goal": "?goal"}},
            "preconditions": [{"capability": {"entity": "?self_mover",
                                              "event": {"flies": {"self_mover": "?self_mover",
                                                                   "source": "?source",
                                                                   "goal": "?goal"}}}}],
            "additions": [{"positioned": {"theme": "?self_mover", "goal": "?goal"}},
                          {"believes": {"cognizer": "?self_mover",
                                        "topic": {"positioned": {"theme": "?self_mover",
                                                                 "goal": "?goal"}}}}],
            "deletions": [{"positioned": {"theme": "?self_mover", "goal": "?source"}},
                          {"believes": {"cognizer": "?self_mover",
                                        "topic": {"positioned": {"theme": "?self_mover",
                                                                 "goal": "?source"}}}}]}}
```

This is an action that describes flying. To fly a character must be capable of flying. This precondition is asserted in the production rule associated with the term declaring a character to be a bird and, for instance, is not associated with the production rule executed when a character is declared to be a bear. When a character flies from a source to a goal, two deletions are retracted from the database: (1) that the character is at the source and (2) that the character thinks it is at the source. And, two additions are made to the database: (1) that the character is at the goal and (2) that the character thinks it is at the goal. This action definition, along with many others, is defined in the file spinner.js.

Actions can be organized into sequences. Such an organization is called a method and is akin to a function definition in most conventional programming languages. Methods are a means of abstraction and composition in the planner used for Spinner. Here is a definition of a method for threatening a character in order to acquire something the character possesses.

```
{"method": {"description": "acquisition of something by threatening",
               "task": {"dcont": {"character": "?character", "desire": "?desire"}},
                "preconditions": [{"ownerIsKnown": {"cognizer": "?character",
                                                    "owner":"?owner", "object": "?desire"}},
                                  {"dominates": {"agent": "?character", "patient": "?owner"}},
                                  {"carries": {"agent": "?owner", "theme": "?desire"}},
                                  {"is": {"performer": "?character", "role": "bully"}}],
             "subtasks": {"ordered": [{"dprox": {"character": "?character",
                                              "objective": "?owner", "place": "?place"}},
                            {"tells": {"speaker": "?character",
                                       "addressee": "?owner",
                                       "message": {"desires": {"experiencer": "?character",
                                                               "theme": {"carries":
                                                                  {"agent": "?character",
                                                                   "theme": "?desire"}}}}}},
                              {"threatens": {"speaker": "?character", "addressee": "?owner",
                                            "message": "?message"}},
                              {"surrenders": {"donor": "?owner", "theme": "?desire",
                                             "recipient": "?character"}}]}}}
```

Note that both actions and methods have “task” slots. These are the declaration of a function name (functor) and the arguments for the task that will be achieved if the action or method can be executed. In this case, this is a “dcont” method, shorthand in the Schank and Abelson notation for “delta control,” a method for acquiring control of something. Methods also have a preconditions slot. Just as it is possible to list a number of deduction rules with the same name and arguments, it is possible to list a number of actions and methods with the same task description as alternative ways to get something done. So, for example, dcont includes declarations for robbing, stealing, being given an object from a friend, bargaining, trading: essentially the repetoire originally implemented in Tale-Spin.

In this definition of the dcont method, the owner of the desired object needs to be known: ownerIsKnown is a reference to a deduction rule that evaluates a set of conditions. Each of these conditions could be separately listed in the preconditions slot, but a deduction rule provide a means to abstract and compose sets of preconditions. Also, for this version of dcont to be applicable, the character applying the method needs to dominate the owner of the object; the character also needs to be a bully; and, the owner needs to be carrying the desired object. If those preconditions are met, then a sequence of subtasks is attempted.

Each subtask can be fulfilled by either an action or another method. Subtasks are either *ordered*, in which case they need to be explored in the order listed, or *unordered*, in which case they can be explored in any permutation. Here the subtasks are first a dprox (delta proximity, the transportation of the character to the location of the owner which could be fulfilled by, for instance, flying (as declared above), walking, running, etc.); then a telling in which the character informs the owner of his/her desire to have what the owner possess; then the character threatens to do something bad to the owner (this can be fulfilled in various ways; various threats are generated; e.g., threatening to kill the owner, threatening to rob something else from the owner; etc.); then, the surrender of the object by the owner to the character.

Given an initial state (described as a set of terms), a set of rules, actions and methods, and a list of tasks, the planner produces a sequence of actions that accomplishes the tasks. The mechanism that accomplishes the sequencing, the planner, is essentially an evaluator for the action and method language described above. There are a myriad of planner types today (see Ghallab, Nau and Traverso, 2004). One type that has proven practical for animating autonomous characters in computer games (see Kelly, Botea and Koenig, 2007) and for generating character discourse (see Strong and Mateas, 2008) has been Hierarchical Task Network (HTN) Planning, a technology that was developed for purposes other than games, dialog and narrative, recently, at the University of Maryland (see Nau et al, 2003). The HTN planner for Spinner can be found in the file shop.js.

The HTN planner is implemented as a search through a state space where each state is represented as a database of assertions about the *diegesis*, the story world (e.g., Josephine is a bear and is currently in the cave). Each time an action is executed the additions and deletions of the action change the database and thus form a new state to follow the preceding state. One can understand this branching space of world states as sets of "possible worlds" as discussed in philosophy and literature (cf., Ryan, 1991). The planner searches the state space by applying combinations of methods and actions until either all of the tasks have been fulfilled or no such state can be reached. The general definition of search is implemented in the file utilities.js (see the function makeSearch). The specifics of the planner's search are implemented in the file shop.js (see the functions plan, initialize, refinePlan, and the variable planSearch). This way of recasting a problem solving program as a search through an abstract state space is a well-known technique in AI (Nilsson, 1971).

The methods and actions language described above addresses the three remaining criteria for a language for story generators; namely, the means to (vi) define actions; (vii) define methods; and, (viii) define alternatives (i.e., disjuncts of actions or methods).

Semantics

One of the oddities of the many interesting programs that were written by Roger Schank's students was their usage of Schank's eleven Conceptual Dependency (CD) "primitive acts" (mentioned above). It was considered reasonable, at the time, to imagine that any verb in any language could be decomposed into these eleven primitives (e.g., ptrans, physical transfer, would suffice for walking, running, flying, riding, jumping, shipping, racing, etc.). This was odd because, at the time, there were many other linguists and philosophers working on the topic of semantics and so Schank's CDs were hardly the only possibility in sight (see, for example, Lyons, 1977a and 1977b). Today, working outside of the context of the Yale AI Lab of the 1970s and 1980s, it is worthwhile to consider some of those alternatives.

In particular, the work of Charles Fillmore, a linguist who has been developing since the 1960s what he now calls "frame semantics" (Fillmore, 1968; and, Ruppenhofer et al., 2010). If Schank was a minimalist, trying to shoehorn the semantics of all verbs into eleven primitive acts, Fillmore is a maximalist, coding the specifics of each verb into a "frame" that includes specific roles for each. In syntax, when studying verbs, one distinguishes the “subject” and “object” roles. In frame semantics, one identifies more specific, semantically meaningful, roles (called “frame elements”) for each verb. Thus, the verb “to fly” has the roles “?self_mover”, “?source” and “?goal” that could occur in a sentence of this form: ?self_mover flew from ?source to ?goal; e.g., “The monarch butterfly flew from Canada to California.” Related verbs employ similar or the same roles. Thus, “to walk” and “to run” also employ the roles used in the definition of “to fly.” Fillmore’s detailed analysis now covers much of the English language (currently 10,000 lexical units) and is available online at this URL: http://framenet.icsi.berkeley.edu/.

One can download all of the verb frames from Fillmore’s website in XML format. However, it is not difficult to translate XML into JSON. In principle, this could be done automatically and systematically. What appears in Spinner (specifically in the spinner.js and initial.js files) are some abbreviated JSON forms for Fillmore’s frames translated unsystematically by hand. If one were to expand the Spinner system it would worthwhile translating all of Fillmore’s database into Spinner JSON terms. In this manner, it is possible to rewrite Tale-Spin, and many of the other Schankian AI programs, without using Schank’s primitive acts as a basis for natural language semantics.

Institutions

According to the Nobel prize winning economist Douglass North, “*Institutions are the rules of the game in a society … They are a guide to human interaction, so that when we wish to greet friends on the street, drive an automobile, buy oranges, borrow money, form a business, bury our dead, or whatever, we know … how to perform these tasks*.” (North, 1990, pp. 3, 4, and 6). Institutions can be formal (e.g., the U.S. Constitution) or informal (e.g., how to tell a story to a child). Formal and/or computational models of institutions are also a burgeoning area of interest in a number of fields outside of the social sciences including the philosophy of language (e.g., Searle, 2010); multi-agent systems (e.g., Jennings, 1993); artificial intelligence as applied to legal reasoning (e.g., Fornara et al., 2008); and, web services (e.g., Singh and Huhns, 2005). What was not clear when Meehan was writing Tale-Spin was that the large bulk of it is an attempt to codify simple social institutions like how to get from place to place, how to bargain, beg, buy, or steal. What is clear now is that to recode Tale-Spin, one can borrow from many other areas of research where institutions have been a focus.

For example, there is an emerging consensus across these various fields that institutions can be modeled as recurrent sets of commitments made by participants through speech acts (cf., Hewitt, 2007; Winograd and Flores, 1986). In a sense, the actions and methods of Spinner encode a form of deontic logic (a logic of obligation, permission, and prohibition) that is implemented by adding and deleting commitments between characters to and from the database. Thus, for example, methods like barter (see the spinner.js file) require that characters promise that a future event will occur (specifically the exchange of goods) before that event happens. The promise action adds a commitment into the database that is resolved when the event (the exchange of goods) takes place. The implementation of the promise action in Spinner is comparable to the analysis of the philosopher John Searle gives in his 1969 book *Speech Acts* (Searle, 1969, p. X). That event is then implemented with an action that deletes the commitments from the database (see the definition of the gives action). Many other speech acts have been formalized in this manner (e.g., Searle and Vanderveken. 1985; Vanderveken, 1990) and the philosophy of action has been extended to include the using of AI planning actions and methods (Bratman, Israel and Pollock , 1988).

In his critique of Klein's story generator (cited above), Meehan states that the purpose of Tale-Spin is an examination of various forms of causality (physical, logical, social, psychological). Institutional analysis in the social sciences, as it has developed since at least the early twentieth century (cf., Durkheim, 1982) provides a framework for exploring these various aspects of the everyday social world. Tapping some of the insights from this past century of social science work would improve upon the planning methods and actions used in Tale-Spin.

Mumbler

To render the database statements of Tale-Spin into English, Meehan threw together the Mumble program (see above). I have followed suit with the Mumbler program, although it would be possible to do a better job. One way to do this better would be, again, to more systematically employ Fillmore's FrameNet database. The FrameNet database includes thousands of sample sentences listed under the verb frames they illustrate and annotated with the semantic roles. For example, the frame for self_motion verbs (run, walk, fly, etc.) includes the sample sentence "A dog ran up" with "dog" annotated with the "?self_mover" role and "up" annotated as the "?goal" (see the definition of the the action for the verb "to fly" given above). These example sentences could be translated into many possible templates for render the frames into English.

Instead, I have simply thrown together a number of templates on my own, with no reference to the lexicographical research literature. The code discussed in this section can be found in the file mumbler.js. Here is one example, for the verb “to carry”:

```
{"carries":
   {"agent": "?agent", "theme": "?theme",
    "past": {"list": ["?agent","carried the","?theme"]},
    "present": {"list": ["?agent","carries the","?theme"]},
    "infinitive": {"list": ["?agent","to carry the","?theme"]},
    "interrogative": {"list": ["will","?agent","carry the","?theme"]}}}
```

The first two lines of the template are used to match the Spinner term about a character carrying some object. For instance, the first line would match this term:

```
{"carries": {"agent":  "joe", "theme":  "honey"}}
```

The following four lines of a text template express the statement in different tenses. The Mumbler program, given a tense and a statement, will render the statement into English in the appropriate tense. The above in the past tense would be rendered as “joe carried the honey.” If a role is not instantiated (i.e., it is still a variable), then it is render as “something.” It would make more sense to do a little more computation and render it as “someone” or “somewhere” or “something” depending on the context, but, again, that was neither Meehan’s focus nor the focus of this essay.

Very sophisticated work has been done on rendering the *syuzhet* from the *fabula* in more contemporary research such as Nick Montfort’s Curveship system: “*Curveship can tell events out of order, using flashback and other techniques, and can tell the story from the standpoint of particular characters and their perceptions and understandings*” (see Montfort, 2011). Obviously, there is an enormous distance between Mumbler and Curveship.

In Spinner, statements can be nested one into another. Some of the Mumbler text templates reflect this possibility. For example, the first few lines of the template for “to inform” look like this:

```
{"informs":
   {"informer": "?informer", "addressee": "?addressee", "message": "?message",
    "past": {"list": ["?informer","informed","?addressee","that",
                     {"past": {"assertion": "?message"}}]},
```

Note that the template includes a reference to another template: the {“past”: {“assertion”: “?message”}} statement directs Mumbler to render that part of “to inform” in the past tense. Thus, to render this into English

```
{"informs": {"informer":  "joe", "addressee":  "irving",
             "message": {"desires": {"experiencer":  "joe",
                                     "theme": {"carries": {"agent":  "joe",
                                                           "theme":  "honey"}}}}}}
```

The “informs” template is called and the message (which begins with “desires”) is then rendered using the template for the verb “to desire” which, in term, needs to call the template for “carries” to finally output this: “joe informed irving that joe desired joe to carry the honey.” Neither pronouns nor capitalization are supported by Mumbler.

A text template of this sort exists for task mentioned in a method or action, every deduction rule, and every functor used in a term that does or might occur in the Spinner database. It is extensible. After adding a new verb or a new method to Spinner, one should also write a new text template for Mumbler so that the output can be rendered in English.

In addition to text templates, Mumbler contains a simple transcoding system defined in a number of deduction rules. The search states generated by the planner are rewritten as Spinner database terms and the deduction rules of Mumbler select portions of these search states to output in English. The top-level rule is {“narrate”: {“tense”: “?tense”}}. It calls two other rules. The rule narrateInitialState gathers all of the assertions in the initial state and then outputs “once upon a time…” and then each assertion as an English sentence. The rule narratePathToSuccess gathers together the sequence of states that led to the successful plan (and the tasks that transformed one state into another). For each state in the path to a successful plan (excluding the initial state), the rule finds the task that was solved in that state (via a method or action) and records the task along with any database additions made at that point. It output the task and the terms representing the additions into English.

Mumbler is used in one other way in the Spinner program. If the user wants what was called in micro-Talespin an “interactive” version of a story, the Mumbler program is used to pose questions to the user. In micro-Talespin an “interactive” story calls the function find-out (line 874) if, during plan generation, a query returns false for statements about a character’s state (specifically, their state of hunger or thirst) and for statements concerning social relations between characters (e.g., “Does Joe like Irving?”). Tale-Spin’s “interactive” mode is similar and, in my opinion, does not constitute an interesting form of interaction.

To duplicate this “interactivity” in Spinner, the user can replace the function conjoin (defined in the file llpl.js) with a different version of conjoin (defined in the file qa.js) which asks the user if a query is true or false after it has been found that the statement is not currently in the database. If the user answers “yes” the statement is true, the statement is asserted into the database and the planner continues its work. If the user responds to the qa.conjoin question with “no,” then the statement is recorded as having been negated and the planner continues it work. See the function interactiveConjoin defined in the file qa.js for details. The Mumbler program is used in render the statements in question into the interrogative before they are printed out in English for the user to respond to.

Example Stories

In the file initial.js are defined the initial conditions and tasks for six example stories. For all of the stories, the planner solve two tasks: (1) a character must acquire some food and then (2) eat it. For example, the tasks for the first five example stories (story1, stpry2, story3, story4, story5) are all the same and are these:

```
{"tasks": {"ordered": [{"dcont": {"character": "joe", "desire": "honey"}},
                       {"eats": {"ingestor": "joe", "ingestibles": "honey"}}]}}
```

Depending upon the initial conditions, the planner ends up using different actions and methods to accomplish the tasks. For example, story2 is about Joe stealing the honey from Irving. The output for this story, like all of the outputs from Spinner, begins with “once upon a time…” followed by a translation of the initial conditions into English, then the statement “one day…” followed by the series of events sequenced by the planner. In the case of story2, the last half of the output reads like this:

"*one day . . . joe wanted to have the honey. joe wanted to go to honey. joe wanted to learn the answer to this question: is honey at the elm tree. joe moved from the cave to the elm tree. joe walked from the cave to the elm tree. joe was at the elm tree. joe believed that joe was at the elm tree. joe stole the honey from irving. joe carried the honey. joe ate the honey*."

In story1, Joe robs Irving of the honey. Story2 is listed above. In story3, Irving gives Joe the honey because they are friends (as declared in the initial conditions). In story4, Joe barters with Irving for the honey. Joe offers Irving, who is a bird, a worm for the honey. In story5, Joe threatens to kills Irving if he doesn't give him the honey. Irving surrends the honey to Joe. One can change the output from by editing just this line in the file spinner.html: story1 can be replaced with story2, story3, story4, story5, or story6:

```
var db = initialConditions.story2
```

Story6 is slightly different. The characters are a fox and a crow rather than, as in the previous examples, a bear and a bird (of no particular type). The tasks for story6 are these:

```
{"tasks": {"ordered": [{"dcont": {"character": "master reynard", "desire": "cheese"}},
                       {"eats": {"ingestor": "master reynard", "ingestibles": "cheese"}}]}}
```

Master Reynard, the fox, needs to acquire some cheese and eat it. Here is the second half (the action!) of the output from Spinner:

"*one day . . . master reynard wanted to have the cheese. master reynard wanted to go to crow. master reynard moved from the elm tree to the oak tree. master reynard walked from the elm tree to the oak tree. master reynard was at the oak tree. master reynard believed that master reynard was at the oak tree. master reynard requested crow for crow to sing something to something. crow believed that master reynard desired crow to sing something to something. crow sang something to something. crow dropped the cheese. cheese was at the oak tree. crow informed master reynard that* *something. master reynard believed that something. master reynard stole the cheese from crow. master reynard carried the cheese. master reynard ate the cheese.*"

In other words, the output from story6 is a garbled version of one of Aesop's fables: "The Fox and the Crow."

Animal Tales

"…Tale-Spin is able to generate stories with a 'point', such as the simpler of the Aesop fables" (Meehan, 1981, p. 199).

"*Once upon a time there was a dishonest fox and a vain crow. One day the crow was sitting in his tree holding a piece of cheese in his mouth. He noticed that he was holding the piece of cheese. He became hungry, and swallowed the cheese. The fox walked over to the crow. The end. … That was suppose to have been 'The Fox and the Crow', of course. The fox was going to trick the crow out of the cheese, but when he got there, there was no cheese. I fixed this by adding the assertion that the crow had eaten recently, so that even when he noticed the cheese, he didn't become hungry*" (Meehan, 1981, p 219).

It is tempting to imagine that Tale-Spin was intended to describe, in formal terms, the morphology or structure of Aesop's fables. This would make it analogous to Lévi-Stauss' work to analyze the structure of more than eight hundred American Indian tales (Lévi-Strauss, 1964; 1966; 1968; 1971). Or, perhaps, Meehan's project could be seen as a straightforward extension to Propp's project: whereas Propp analyzed the "Fairy Tales" of Aarne's collection (numbered 300-749), Meehan might be seen to be analyzing the "Animal Tales" of Aarne's collection (numbered 1-299 and beginning with tales about "The Clever Fox and other animals").

But, Meehan only ever describes in vague terms the stories he is interested to have Tale-Spin generate. Even if we assume that Meehan's corpus was Aesop's tales, that is, itself, problematic: "*The 'Fables of Aesop' are familiar to everyone, and books with this title are easy to find. So it may come as something of a surprise to learn that no fables known to have been written by Aesop exist. No surviving fable collection, and no individual fables, are old enough to have been written by Aesop, who lived, if he existed at all, in the sixth century BC*" (Mann, 2009, p. 1).

My point is that Meehan's goal was far afield from those of Propp, Lévi-Strauss, and their colleagues and students. Scholars of ethnology and literature have attempted to closely study very specific corpora of texts be they a defined set of Russian folktales, Shakespeare's plays, or the *Illiad*. But, Meehan's work was not this kind of scholarship.

"*Never trust flatterers. So says Aesop in the fable called "The Fox and the Crow." There are lots of ways to get a program to tell that story, the easiest being simply to have the computer read it in and print it out. Another method, used by Klein [see above], is to write some code which will produce each part of that particular story when the whole program runs. Such programs, however, are limited to telling exactly those stories that have been put into them, whole or piecemeal. But there are many stories that have 'Never trust flatterers' as their moral. How can we get a computer to write them?"* (Meehan, 1976, pp. 108-109).

In a nutshell, Meehan's work is about writing stories while Propp et al. have been studying and describing stories. Meehan did, nevertheless, also, have an object of study, but it was not stories: "*Tale-Spin is intended to model people. … We're interested in studying people, not computers, so we build models which correspond to the way in which we think people behave*" (Meehan, 1976, p. 4). However, the only empirical observations about human behavior that Meehan considers in his dissertation are anecdotal. Consequently, one might consider Tale-Spin not as a contribution to a failed branch of science as the conventional histories of AI would have us believe. Rather, Tale-Spin might be completely reconceived as a form of creative writing.

Noah Wardrip-Fruin explains this insight. "*Tale-Spin has in some sense lost its status as a simulation. There's no one left who believes that it represents a simulation of how actual people behave in the world. … As this has taken place, Tale-Spin has become, I would argue,* more *interesting as a fiction. It can no longer be regarded as an accurate simulation of human planning behavior with a layer of semisuccessful storytelling on top of it. Rather, its entire set of operations is now revealed as an authored artifact – as an expression, through process and data, of the particular and idiosyncratic view of humanity that its author and his research group once held*" (Wardrip-Fruin, 2010, p. 151). Or, perhaps, Meehan's dissertation title reveals him to be a fiction writer right from the start. The title is "The Metanovel: Writing Stories by Computer." Neither people nor simulations are mentioned in the title.

Conclusions

Unlike Wardrip-Fruin and most other humanities scholars writing about Tale-Spin (cf., Murray, 1997; Aarseth, 1997)) who found Tale-Spin's output to be underwhelming, Marie-Laure Ryan found it to be impressive. "*Nowadays specialists in artificial intelligence are busy trying to generate literary masterpieces, and one of the most impressive achievements so far has been the production, by a program named Tale-Spin written by James Meehan, of a story in which a fox, trying to get a cheese held by a crow, flatters the crow into singing, and causes him to drop the cheese. What should make the modern version of 'The Fox and the Crow' radically different from the fables of Aesop and La Fontaine is that it was produced by a brain aware of its own operations, using explicitly defined procedures, whereas the naturally written stories were the product of obscure intuitions about art*" (Ryan, 1991, p. 233).

However, Ryan's book, in which she makes this statement, is perhaps surprisingly similar to Wardrip-Fruin's book in which he coins the phrase "The Tale-Spin Effect" to mean a *"…surface illusion of system simplicity – which the available options for play (if any) can't alter*" (Wardrip-Fruin, 2010, p. 146). In other words, programs that display the "Tale-Spin Effect" do many complicated calculations invisible to the user and then output very simple – or perhaps, more accurately, *simplistic* or even *raw* -- texts or images. The Wardrip-Fruin and the Ryan books are similar because both provide detailed descriptions of the internal workings of Tale-Spin and entreat us, the readers, to give some thought to the workings of the processes that produce garbled versions of Aesop's fables.

Despite Ryan's enthusiasm, it seems clear to me that Tale-Spin's output was underwhelming. In fact, even now, the productions of every contemporary story generation program I have seen, except one, have been disappointing. The one exception is the program Terminal Time (Mateas, Vanouse and Domike, 1999) which delivers a compelling audience experience.

However, the internal workings of Tale-Spin are neither simplistic nor underwhelming. The point of this chapter has been to take descriptive analyses, like those of Ryan and Wardrip-Fruin, one step further to determine how the computational processes of Tale-Spin could be not just described but reinterpreted, reinterpreted using a method that Abelson and Sussman have called "metalinguistic abstraction."

In the course of this reinterpretation several ideas have been explored that might provide means for writing a story generator that is more interesting than Tale-Spin. First, one might replace Roger Schank's approach to semantics (the primitive acts of conceptual dependency) with the more nuanced and expansive approach of Charles Fillmore (frame semantics). Second, it may be worthwhile to recognize the Schankian school of pragmatics (scripts, plans, goals, cases, etc.) as a minor stream within the larger set of institutional analyses that have been conducted throughout the social sciences during the last century. Many interesting insights could be appropriated from these other studies. Third, if we understand Tale-Spin as the partial implementation of a programming language for planning actions and methods -- specifically an evaluator for an HTN planner – we can understand Tale-Spin as just one possible program written in that language. Other programs could be written that more closely resemble the work of Lévi-Strauss, or Propp, or Roland Barthes (e.g., Barthes, 1966). Or, perhaps more pertinently, one can recognize that computer programming itself is a new form of writing that one might use to explore fiction as a creative writer or an essayist.

Acknowledgements

I am most grateful for the friendship and intellectual excitement of the Yale AI Lab in the 1980s, the Narrative Intelligence reading group at the MIT Media Lab in the 1990s, and now the Games and Playable Media Research Center at UC Santa Cruz. I would like to thank my graduate students at UC Santa Cruz of the past two years who have worked through some of this material with me during a seminar, a studio, and some independent studies on the topic of Software Studies. I acknowledge the financial support of the Maison des Sciences de l'Homme in Paris where I was Directeur d'Etudes Associé for the summer of 2010 with the Program for the Study of the Use of Information and Communication Technologies in Migrations and where I was able to code much of the planner used for this project.

References

Espen Aarseth, Cybertext: Perspectives on Ergodic Literature (Johns Hopkins UP 1997)

Antti Aarne, *The Types of the Folktale: A Classification and Bibliography*, The Finnish Academy of Science and Letters, Helsinki, 1961 (originally published in 1910)

Harold Abelson and Gerald Jay Sussman with Julie Sussman, Structure and Interpretation of Computer Programs, Second Edition, MIT Press, 1996. Available online: http://mitpress.mit.edu/sicp/full-text/book/book.html

Phil Agre, *Computation and Human Experience* (Cambridge University Press, 1997)

Roland Barthes, “Introduction à l’analyse structurale des récits,” *Communications* 8: 1-27 (1966).

Michael Bratman, David Israel and Martha Pollock "Plans and Resource-Bounded Practical Reasoning," Computational Intelligence 4 (1988): 349-355

Marc Cavazza, David Pizzi. 2006. Narratology for Interactive Storytelling: a Critical Introduction 3rd International Conference on Technologies for Interactive Digital Storytelling and Entertainment, Darmstadt, Germany, December 2006

Cavazza, M.; Lugrin, J.; Pizzi, D.; and Charles, F. 2007. Madame Bovary on the Holodeck: Immersive Interactive Storytelling. In Proc. of the 15th Int. Conf. on Multimedia (ACMMM 2007), 651–660.

Nathanael Chambers and Dan Jurafsk, “Unsupervised Learning of Narrative Schemas and their Participants,” *ACL '09 Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2* (Association for Computational Linguistics Stroudsburg, PA): 602-610

Chomsky, Noam (1957), *Syntactic Structures*, The Hague/Paris: Mouton

Chomsky, Noam, “On the Notion ‘Rule of Grammar’.” In Roman Jakobson (ed.) Structure of Language and Its Mathematical Aspects. *Proc. 12th Symposium in App. Math.* Providence, RI: American Mathematical Society, 1961, pp. 6-24

Chomsky, Noam (1995). *The Minimalist Program*. MIT Press

Codd, E.F. (1970). "A Relational Model of Data for Large Shared Data Banks". Communications of the ACM 13 (6): 377–387).

Colmerauer, A.; Roussel, A. (1993). "The birth of Prolog". ACM SIGPLAN Notices 28: 37)

Marc Davis and Michael Travers, “A brief overview of the Narrative Intelligence reading group,” in Michael Mateas and Phoebe Sengers (Eds.), Narrative Intelligence. Amsterdam: John Benjamins. 2003, pp. 27-40.

Dehn, N. (1981). Story generation after TALE-SPIN. In Proceedings of the 7th International Joint Conference on Articial Intelligence, pp. 16-18

Gerald DeJong. Skimming stories in real time. PhD dissertation, Yale University, 1979.

Émile Durkheim, The Rules of the Sociological Method, (Edited by Steven Lukes; translated by W.D. Halls). New York: Free Press, 1982

Umberto Eco,*The Search for the Perfect Language* (Wiley-Blackwell, 1995)

Dwight Eisenhower (Public Papers of the Presidents of the United States, Dwight D. Eisenhower, 1957, National Archives and Records Service, Government Printing Office, p. 818).

R. Fikes and N. Nilsson (1971). STRIPS: a new approach to the application of theorem proving to problem solving. Artificial Intelligence, 2:189-208

Fillmore, C. J. (1968). The case for case. In Bach, E. and Harms, R., editors, Universals in Linguistic Theory. Holt, Rinehart & Winston, New York

Nicoletta Fornara, Francesco Vigano, Mario Verdicchio, Marco Colombetti, “Artificial institutions: a model of institutional reality for open multiagent systems,” Artif Intell Law (2008) 16:89–105

Geib, C. W. 1994. The Intentional Planning System: It-PlanS. In Proceedings of the 2nd Int. Conf. on Artificial Intelligence Planning Systems (AIPS-94).

Malik Ghallab, Dana Nau, Paolo Traverso, *Automated Planning: Theory & Practice* (Morgan Kaufmann, 2004)

Greimas (1973) "Actants, Actors, and Figures." *On Meaning: Selected Writings in Semiotic Theory*. Trans. Paul J. Perron and Frank H, Collins. Theory and History of Literature, 38. Minneapolis: U of Minnesota P, 1987. 106-120

John Haugeland, Artificial Intelligence: The Very Idea (MIT Press, 1985)

Eric Havelock, Preface to Plato (Harvard University Press, 1963)

James Hendler, Avoiding another AI Winter, IEEE Intelligent Systems (March/April 2008 (Vol. 23, No. 2) pp. 2-4).

Carl Hewitt , “What Is Commitment? Physical, Organizational, and Social (Revised),” in P. Noriega et al. (Eds.): COIN 2006 Workshops, LNAI 4386, pp. 293–307, Springer-Verlag Berlin Heidelberg 2007

Fredric Jameson, The Prison-House of Language: A Critical Account of Structuralism and Russian Formalism (Princeton: Princeton University Press. 1972).

Jennings, N.R.: Commitments and conventions: The foundation of coordination in multi-agent systems. The Knowledge Engineering Review 8(3), 223–250 (1993)

W. Lewis Johnson, PROUST: Intention-based diagnosis of novice programming errors, PhD Dissertation, Yale University, 1985.

Kelly, J. P.; Botea, A.; and Koenig, S. 2007. Planning with Hierarchical Task Networks in Video Games. In Proceedings of the ICAPS-07 Workshop on Planning in Games

Klein, S., Aeschliman, Applebaum, Balsisger, Curtis, Foster, Kalish, Kamin, Lee & Price 1976. “Simulation d'hypothèses émisés par Propp et Lévi-Strauss en utilisant un système de simulation meta-symbolique.” *Informatique et Sciences Humaines*, No. 28, pp. 63-133, Mars. [A revised and expanded French translation of 'Modelling Propp and Lévi-Strauss in a Meta-symbolic Simulation System.' in Patterns in Oral Literature, edited by H. Jason & D. Segal, World Anthropology Series, The Hague: Mouton, 1977]; also accessible on the web here:

http://www.cs.wisc.edu/~sklein/Simulation-Meta-Symbolique%20d%27Hypotheses-Propp%20&%20Levi-Strauss.pdf

Lakoff, George, “Structural Complexity in Fairy Tales,” *The Study of Man*, Vol. 1 (1972), 128-150; also available at Lakoff’s website: http://georgelakoff.files.wordpress.com/2010/12/structural-complexity-in-fairy-tales-lakoff-1972.pdf

George Lakoff and Srini Narayanan, “Toward a Computational Model of Narrative,” Computational Models of Narrative: Papers from the AAAI Fall Symposium (FS-10-04)).

Lebowitz, M. 1987. Planning stories. In Proceedings of the 9th Annual Conference of the Cognitive Science Society

Law, John, and John Hassard. 1999. *Actor network theory and after*. Oxford [England]: Blackwell/Sociological Review).

Claude Lévi-Stauss, “The Structural Study of Myth,” in *Structural Anthropology* (New York: Basic Books, 1963): 210.

Claude Lévi-Strauss, Mythologiques 1: Le cru et le cuit. Paris: Plon, 1964.

Claude Lévi-Strauss, Mythologiques 2: Du miel aux cendres. Paris: Plon, 1966.

Claude Lévi-Strauss, Mythologiques 3: L’origine des Manières de Table. Paris: Plon, 1968.

Claude Lévi-Strauss, Mythologiques 4: L’Homme nu. Paris: Plon, 1971.

Claude Lévi-Strauss, 1983, *The Raw and the Cooked: Mythologiques, Volume 1*, Chicago: University of Chicago Press

Claude Lévi-Strauss, “Structure and Form: Reflections on a work by Vladimir Propp,” in Propp, Vladimir. Theory and History of Folklore. Ed. Anatoly Liberman. University of Minnesota: University of Minnesota Press, 1984.

John Lyons, Semantics, Volume 1 (Cambridge University Press, 1977)

John Lyons, Semantics, Volume 2 (Cambridge University Press, 1977)

Jill Mann, *From Aesop to Reynard, Beast Literature in Medieval Britain* (Oxford University Press, 2009

Lev Manovich, “Database as Symbolic Form,” *Convergence* June 1999 vol. 5 no. 2 80-99.

Michael Mateas and Phoebe Sengers (Eds.), Narrative Intelligence. Amsterdam: John Benjamins. 2003

Michael Mateas, Paul Vanouse and Steffi Domike, “Terminal Time: An ideologically-biased history machine.” AISB Quarterly, Special Issue on Creativity in the Arts and Sciences, number 102, 1999, pages 36-43

Marshall McLuhan, The Gutenberg galaxy: the making of typographic man (University of Toronto Press, 1962)

James Meehan, *The Metanovel: Writing Stories by Computer*, PhD dissertation, Yale University, 1976

James Meehan, The New UCI Lisp Manual (Lawrence Erlbaum Assoc Inc, 1979)

James Meehan, “Tale-Spin,” in Roger Schank and Christopher Riesbeck (editors), *Inside Computer Understanding: Five Programs Plus Miniatures* Hillsdale, New Jersey: Lawrence Erlbaum Associates. 1981.

Miller, George; Galanter, Eugene, & Pribram, Karl (1960). Plans and the structure of behavior. New York: Holt, Rinehart and Winston

Montfort, Nick. "Curveship's Automatic Narrative Variation." Proceedings of the 6th International Conference on the Foundations of Digital Games (FDG '11), pp. 211-218, Bordeaux, France. 29 June-1 July 2011

Munindar P. Singh and Michael N. Huhns, Service-Oriented Computing: Semantics, Processes, Agents (John Wiley & Sons, Ltd., 2005

Janet Murray, Hamlet on the Holodeck: The Future of Narrative in Cyberspace (Free Press, 1997; MIT Press 1998

Nau, D, et al, SHOP2: An HTN Planning System, Journal of Artificial Intelligence Research 20 (2003) 379-404).

Newell, A.; Shaw, J.C.; Simon, H.A. (1959). Report on a general problem-solving program. Proceedings of the International Conference on Information Processing. pp. 256-264

Nilsson, N., Problem-Solving Methods in Artificial Intelligence, New York: McGraw-Hill, 1971).

Douglass North, Institutions, Institutional Change and Economic Performance (Cambridge University Press: Cambridge, UK, 1990Jean-François Lyotard, The Postmodern Condition: A Report on Knowledge (University of Minnesota Press, 1979)

Sheldon Pollock, “Future Philology? The Fate of a Soft Science in a Hard World,” *Critical Inquiry* 35 (Summer 2009): 931-961

Julie Porteous, Jonathan Teutenberg, David Pizzi and Marc Cavazza, “Visual Programming of Plan Dynamics using Constraints and Landmarks,” in Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS), Freiburg, Germany, June 2011

Gerald Prince, A Dictionary of Narratology, Revised Edition (University of Nebraska Press, 2003)

Vladimir Propp (author), Louis A. Wagner (Editor), Laurence Scott (Translator) Morphology of the Folktale, 2nd edition, (Publications of the American Folklore Society) (University of Texas Press, 1968)

Riedl, M. O., and Young, R. M. 2004. An Intent-Driven Planner for Multi-Agent Story Generation. In Proceedings of the 3rd International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS).

Mark O. Riedl and R. Michael Young. Narrative Planning: Balancing Plot and Character. Journal of Artificial Intelligence Research, vol. 39, 2010

Christopher K. Riesbeck and Roger C. Schank (editors), Inside Case-Based Reasoning (Psychology Press, 1989).

Robinson J. A. "A Machine-Oriented Logic Based on the Resolution Principle." J. Assoc. Comput. Mach. 12, 23-41, 1965

Josef Ruppenhofer, Michael Ellsworth, Miriam R. L. Petruck, Christopher R. Johnson, Jan Scheffczyk, FrameNet II: Extended Theory and Practice, 2010, available online: http://framenet2.icsi.berkeley.edu/index.php?option=com_wrapper&Itemid=126

Marie-Laure Ryan, Possible Worlds, Artificial Intelligence and Narrative Theory (Bloomington: Indiana University Press, 1991

Warren Sack, “Knowledge Compilation and the Language Design Game,” in Intelligent Tutoring Systems, Second International Conference (Lecture Notes in Computer Science) Claude Frasson, Gilles Gauthier, and Gordon McCalla (editors), (Berlin: Spring-Verlag, 1992).

Warren Sack, “Aesthetics of Information Visualization,” in *Context Providers: Conditions of Meaning in Media Arts*, Christiane Paul, Victoria Vesna, and Margot Lovejoy, Editors (Bristol, UK: Intellect; and, Chicago: University of Chicago Press, 2011)

Warren Sack and Marc Davis, "IDIC: Assembling Video Sequences from Story Plans and Content Annotations," in Proceedings of the IEEE International Conference on Multimedia Computing and Systems, Boston, MA, May 14-19, 1994.

Schank, R. (1972). Conceptual dependency: A theory of natural language understanding. *Cognitive Psychology* 3 (4): 552-631

Roger Schank and Robert Abelson, Scripts, Plans, Goals and Understanding: an Inquiry into Human Knowledge Structures (Hillsdale, NJ: Erlbaum, 1977)

Roger C. Schank, Alex Kass and Christopher K. Riesbeck (editors) Inside Case-Based Explanation (Psychology Press, 1994)

Roger Schank and Christopher Riesbeck (editors), *Inside Computer Understanding: Five Programs Plus Miniatures* Hillsdale, New Jersey: Lawrence Erlbaum Associates. 1981.

Searle, John R. Speech Acts: An Essay in the Philosophy of Language. London: Cambridge University Press, 1969

John Searle, Making the Social World: The Structure of Human Civilization (Oxford University Press: Oxford, UK, 2010

Searle, John R., and Daniel Vanderveken. 1985. Foundations of illocutionary logic. Cambridge [Cambridgeshire]: Cambridge University Press

Guy L. Steele, Common Lisp: The Language, 2nd Edition (Woburn, MA: Digital Press, 1990).

Christina R. Strong and Michael Mateas. Talking with NPCs: Towards dynamic generation of discourse structures. In Proceedings of the 4th Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2008), Palo Alto, California, October 2008

Uther, Hans-Jörg, *The Types of International Folktales: A Classification and Bibliography. Based on the system of Antti Aarne and Stith Thompson*. FF Communications no. 284–286. Helsinki: Suomalainen Tiedeakatemia, 2004. Three volumes.

Vanderveken, Daniel. 1990. Meaning and speech acts. Cambridge [England]: Cambridge University Press

Winograd, Terry, and Fernando Flores. Understanding Computers and Cognition: A New Foundation for Design. Norwood, N.J: Ablex Pub. Corp, 1986)

Noah Wardrip-Fruin, Expressive Processing: Digital Fictions, Computer Games, and Software Studies (MIT Press, 2009).

Young, R. 1999. Notes on the use of plan structures in the creation of interactive plot. In Proceedings of the AAAI;

Young, R. M. 2000a. Creating Interactive Narrative Structures: The Potential for AI Approaches. In AAAI Spring Symposium in Artificial Intelligence and Entertainment. AAAI Press