

University of Bath



DOCTOR OF ENGINEERING (ENGD)

Building Abstractable Story Components with Institutions and Tropes

Thompson, Matthew

Award date:
2018

Awarding institution:
University of Bath

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 22. May. 2019

Building Abstractable Story Components with Institutions and Tropes

submitted by

Matt Thompson

for the degree of Doctor of Engineering (EngD) Digital Media

of the

University of Bath

Department of Computer Science

May 2018

COPYRIGHT

Attention is drawn to the fact that copyright of this thesis rests with its author. This copy of the thesis has been supplied on the condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the thesis and no information derived from it may be published without the prior written consent of the author.

This thesis may be made available for consultation within the University Library and may be photocopied or lent to other libraries for the purposes of consultation.

Signature of Author

Matt Thompson

Contents

1	Introduction	8
1.1	Interactive Narrative	10
1.2	Interactive narrative for games	11
1.3	Normative Systems	12
1.4	Story Tropes	12
1.5	Answer Set Programming for Story Solving	14
1.6	Outline	14
1.7	Published Works	16
2	Literature Review	18
2.1	Narratology	18
2.1.1	Narrative Structure	19
2.1.2	Types of Narrative	22
2.1.3	Formalisms of Narrative	24
2.1.4	Other Types of “Story Component”	25
2.2	Implementations of Experimental Narrative	26
2.2.1	Story Generation	26
2.2.2	Characters as Intelligent Agents	28
2.2.3	Modelling Narrative with Logic	34
2.2.4	Normative Frameworks for Multi-Agent Systems	35
2.2.5	Modelling Characters with Emotional Models	37
2.2.6	Discussion	40
3	Tropes as Story Components	43
3.1	Tropes: a “Folksonomy” of Story Components	44
3.2	Why Use Tropes?	50
3.2.1	A Means of Abstraction	50
3.2.2	Conceptually Simple	52
3.2.3	A Library of Re-usable Examples	52
4	TropICAL: A Language for Story Tropes	54
4.1	Controlled Natural Language Syntax	55
4.1.1	Attempto Controlled English	55

4.1.2	Inform 7	56
4.2	Requirements	58
4.2.1	Use Cases	58
4.2.2	Requirement Specification	62
4.3	Language Design and Features	62
4.3.1	Entity Declarations	63
4.3.2	Sequences of Events	64
4.3.3	Obligations	65
4.3.4	Branching Events	66
4.3.5	Subtropes	68
4.4	InstAL Code Generation	71
4.4.1	Answer Set Programming (ASP)	71
4.4.2	InstAL: The Institution Action Language	74
4.4.3	Why use ASP and InstAL?	80
4.4.4	Compilation Strategy	81
4.4.5	Initial Conditions	84
4.4.6	Generation	86
4.4.7	Initiation	87
4.4.8	Termination	88
4.4.9	Branches	89
4.4.10	Obligations	90
4.4.11	Institutional Bridges	92
4.5	Answer Set Generation	94
4.6	Adding Constraints	94
4.7	Example Answer Sets (Traces)	95
4.7.1	Evil Empire	95
4.7.2	The Hero’s Journey	96
4.7.3	The Hero’s Journey <i>and</i> The Evil Empire	98
4.8	Summary	99
5	StoryBuilder: An Interface for Trope-based Interactive Story Creation	101
5.1	Interface	102
5.1.1	Overview	102
5.1.2	Usage Examples	108
5.1.3	Design Justification	109
5.1.4	Creating a Punch and Judy story with StoryBuilder	109
5.1.5	Combining Tropes Together	115
6	Using TropICAL with a Normative Multi-Agent System	117
6.1	Punch and Judy as Tropes	117
6.1.1	Issues with Propp Mapping	121
6.1.2	The Sausages Scene	122

6.2	An Interactive, Generative Punch and Judy Show with Institutions and Emotional Agents	125
6.2.1	VAD emotional model	125
6.2.2	VAD emotions in Jason	127
6.2.3	Agent decision making	128
6.2.4	Architecture	129
6.3	Summary	132
7	Evaluation & Validation	135
7.1	StoryBuilder Evaluation	135
7.1.1	Procedure	136
7.1.2	Participants	136
7.1.3	User Tasks	137
7.1.4	Limitations of the System	140
7.1.5	Analysis	141
7.1.6	Discussion of StoryBuilder Evaluation	152
7.2	A Full Specification of Punch and Judy	153
7.2.1	“Introduction” Scene	154
7.2.2	“Punch, Judy and the Baby” Scene	154
7.2.3	“Punch and the Policeman” Scene	156
7.2.4	“Crocodile and Sausages” Scene	156
7.2.5	“Punch and the Devil” Scene	157
7.2.6	Putting It All Together	157
7.2.7	Authoring the Character Agents	160
7.2.8	The Idiot	162
7.2.9	Code for Punch Agent	163
7.2.10	Combining Tropes	164
7.3	An Example of “Character Freedom”	165
7.3.1	Handling Trope Violations: The Director Agent	166
7.4	Summary	167
8	Discussion	168
8.1	Comparison with Planning Approaches	168
8.1.1	Accommodation and Intervention	168
8.1.2	Façade’s Beats and Drama Manager	169
8.2	Extent of “Character Freedom” Achieved	170
8.2.1	Prototype Implementation	170
8.2.2	Proposed Full Implementation	171
8.2.3	Advantages of Character Freedom	171
8.3	Evaluation Against Interactive Criteria	172
8.4	Complexity of Story Generation	173
8.4.1	A Simple Sequence of Events	174

8.4.2	Basic Story Branching	174
8.4.3	Five Branches	174
8.4.4	A Combination of Branches and Sequences	175
8.4.5	A Trope Containing a Subtrope	175
8.4.6	Two Simultaneous Tropes (Without Branches)	176
8.4.7	Two Simultaneous Tropes (With Branches)	176
8.4.8	Two Simultaneous Tropes (With Subtropes)	177
8.4.9	Two Simultaneous Tropes (With Subtropes and Branches)	177
8.4.10	Three Simultaneous Tropes (With Subtropes and Branches)	178
8.4.11	Analysis	179
9	Conclusions & Future Work	180
9.0.1	Summary of Contributions	180
9.1	Future Work	180
9.1.1	Opportunities for Future Research	181
9.1.2	Addressing Limitations in <i>TropICAL</i> and StoryBuilder	183
9.1.3	Methods for Character Agent Authoring	185
9.1.4	Application of <i>TropICAL</i> to Other Domains	188
9.1.5	Concluding Remarks	191
A	London Interactive Fiction Meetup Questionnaire	192
B	Full “Don’t Touch It, You Idiot” Institution	194
C	Generated InstAL Code	200
C.1	Branching Trope Example	204
C.2	“Evil Empire” Trope Example	206
D	Parser Implementation	209
E	Full Trace for “Evil Empire” and “Hero’s Journey” Tropes	212
F	AgentSpeak Code for Character Agents and Roles	215
F.1	Character Roles	215
F.1.1	The Idiot	215
F.1.2	The Narrator	215
F.1.3	The Hero	216
F.2	Character Agents	216
F.2.1	Code for Punch Agent	216
F.2.2	Code for Judy Agent	217
F.2.3	Code for Joey Agent	217
F.2.4	Code for Crocodile Agent	217
F.2.5	Code for Policeman Agent	217
F.2.6	Code for Devil Agent	218

F.2.7	Code for “Director” Agent	218
G	Thematic Analysis Transcripts	219
G.1	Participant A	219
G.2	Participant B	226
G.3	Participant C	236
G.4	Participant D	244
G.5	Participant E	255
G.6	Participant F	265
G.7	Participant G	272
G.8	Participant H	282

Acknowledgments

I would like to thank the following people for helping make this thesis possible:

Julian, for his always-patient and insightful supervision. Steve, for his industrial supervision and advice over pints and coffee. Andy for the partnership that made this EngD happen. Tobias and Dorothea for getting me to University and encouraging me to stop slacking. Sandra for teaching me the value of strength. Guillaume for a world-class education in cinema. Hashim, Dan and Milto for good times and moral support. Kwam for teaching me the transcendent nature of code. Richard for his advice on Clojure and testing. Finally, to my mother Janice and sister Natalie for all of their support and encouragement.

Summary

Though much research has gone into tackling the problem of creating interactive narratives, no software has yet emerged that can be used by story authors to create these new types of narratives without having to learn a programming language or narrative formalism.

Widely-used formalisms in interactive narrative research, such as Propp’s “Morphology of the Folktale” and Lehnert’s “Plot Units” allow users to compose stories out of pre-defined components, but do not allow them to define their own story components, or to create abstractions by embedding components inside of other components.

Current tools for interactive narrative authoring, such as those that use Young’s *Mimesis* architecture or *Façade*’s drama manager approach, direct intelligent agents playing the roles of characters through use of planners. Though these systems can handle player interactions and adapt the story around them, they are inaccessible to story authors who lack technical or programming ability.

This thesis proposes the use of *Story Tropes* to informally describe story components. We introduce *TropICAL*, a controlled natural language system for the creation of tropes which allows non-programmer story authors to describe their story components informally. Inspired by Propp’s *Morphology*, this language allows for the creation of new story components and abstractions that allow existing components to be embedded inside of new ones.

Our TropICAL language compiles to the input language for an Answer Set solver, which represents the story components in terms of a formal *normative framework*, and hence allows for the automated verification of story paths. These paths can be visualised as branching tree diagrams in the StoryBuilder tool, so that authors can visualise the effect of adding different tropes to their stories, aiding the process of authoring interactive narratives.

We evaluate the suitability of these tools for interactive story construction through a thematic analysis of story authors’ completion of story-authoring tasks using TropICAL and StoryBuilder. The participants complete tasks in which they have to describe stories with different degrees of complexity, finally requiring them to reuse existing tropes in their own trope abstractions. The thematic analysis identifies and examines the themes and patterns that emerge from the story authors’ use of the tool, revealing that non-programmer story authors are able to create their own stories using tropes without having to learn a strict narrative formalism.

Chapter 1

Introduction

In most media, stories are told linearly, where one event follows another sequentially until the conclusion is reached. This is the format which most printed works follow, due to the physical limitations of the medium. Though these are limitations that we have come to accept due to the popularity and wide proliferation of printed media, they are not limitations inherent to the act of storytelling. Even before written language, when stories were told around the camp fire, the audience would have opportunities to interrupt the narrator in order to add narrative details of their own. In its transition from the spoken to the printed word, the medium of storytelling has lost its interactivity.

Innovative authors such as John Barth, and works such as the “Choose Your Own Adventure” and “Fighting Fantasy” series of books have attempted to expand the possibilities of the printed word to allow for non-linear and interactive narratives. Working within the limitations of the medium, these books offer the reader choices such as “turn to page 50 to drink the water” or “turn to page 14 to run away”. These decision points create a branching narrative that can be described by a tree structure. Though this method for describing branching narrative in a book is innovative, it falls prey to the limitations of book size: the choices are very often only the *illusion* of choice, where the decision is usually between continuing the story and death of the reader’s character. If the tree of the narrative were to be visualised, it would contain a great many “dead end” leaves.

Interactive storytelling is the method of telling a story that a “reader” can meaningfully change through interaction. The “Choose Your Own Adventure” example just described is not an example of meaningful change in a story, as the only interactions are binary decisions that lead to a limited number of paths through the tale. Characters are not able to react or change their behaviour based on a sequence of actions the “reader” of the story has taken, for example. The structure of the story cannot change in real time in response to the actions of the “reader”. Even with new media such as computer games, crafting a dynamically changing story is challenging, and narrative interaction is limited to “multiple endings”, where a single decision point at the end of the game allows the player to choose between multiple outcomes.

Interactive storytelling researchers have attempted to challenge the limitations of current computer game stories through the use of artificial intelligence techniques. Early systems such

as Talespin (Meehan, 1977) and Minstrel (Turner, 1993) focus on story *generation*, where a story is created that changes each time the software program is run. Later systems would then go on to use intelligent agents to simulate the characters in a story, each with goals and plans that mimic their motivations. More recent storytelling systems use planners (such as ones based on Young’s architecture (Young et al., 2004)) to direct the actions of these character agents to fit the pre-determined path of a story.

Though the field of interactive narrative generation has many active researchers and research centres, most of the tools produced from these projects are targeted towards other computer science researchers to use. To learn esoteric techniques such as how to use goal-based planner systems in order to describe branching narratives is a challenge even for professional programmers. Similarly, describing stories with declarative, logic-based languages such as Prolog or Answer Set Programming languages requires authors to master new and very different programming paradigms. If such methods are challenging to programmers, they are far beyond the reach of non-programmers such as fiction authors. The creation of interactive narrative authoring tools that can be used by authors with no previous programming experience remains a challenge.

Another challenge in the field is the identification of a suitable formalism for narrative. Researchers still use formalisms created almost a century ago, such as Aarne-Thompson’s “Tale-Type Index” (Aarne and Thompson, 1987), or Propp’s “Morphology of the Folktale” (Propp, 1968), due to inertia, familiarity, and the fact that they are “good enough” for most purposes.

The contribution of this thesis is the interpretation of Propp’s theory to use *story tropes* as re-usable story components that can be combined together to form non-linear stories. These tropes are integrated into a multi-agent system through the concept of of a *social institutions* (also known as normative frameworks) for the governance of intelligent agent characters in an interactive narrative. These institutions are defined as story tropes using controlled natural language. The normative framework can be thought of as the director that governs the behaviours of the actors in the story, telling them what they are *permitted* and *obliged* to do as part of the story. The tropes can be considered as a screenplay of sorts, a user interface that contains a high-level description of what the director has to do. This thesis introduces the concept of story tropes as a story formalism for the first time into the interactive narrative literature, where tropes are formal, composable components of stories. Tropes describe identifiable patterns that recur throughout multiple stories, which become familiar to audiences through repeated exposure. Examples include *The Hero’s Journey*, where the story’s protagonist leaves home to go on a quest, where they must overcome many challenges and obstacles before returning triumphant, or the *MacGuffin*, where the main characters of a story are chasing after an object (such as the Ark in *Indiana Jones and the Raiders of the Lost Ark*, or the falcon statue in *The Maltese Falcon*). A story is composed of tropes that may describe its structure (such as with the *Three Act Structure* or *Climax* tropes) as well as specific scenes that must take place (such as the *Deus Ex Machina* or *Betrayal* tropes). Describing the story in terms of such tropes gives an author the ability to piece together a high-level version of the story structure, while still leaving the small details of the story to the plans of the char-

acter agents. Additionally, tropes can be composed in different ways. As with formalisms such as Propp (1968) and Lehnert (1981), tropes can be composed one after another to tell a sequential series of scenes. However, the real flexibility of tropes comes with their ability to be composed *hierarchically*, describing the structure of a narrative as well as its phases. This is due to the fact that tropes can describe different levels of abstraction. A top-level trope would be the *Three Act Structure* mentioned above, which can three main “acts” that consist of sequences of tropes. However, different paths through the narrative could lead to different chains of tropes within the three-act structure. Another example of composing tropes hierarchically could be Vonnegut’s “Man in Hole” story shape (Vonnegut, 2009), where events get worse for the hero until the very end of the story. At the start of this top-level trope, progressively “worse” tropes could be strung together, followed by progressively “better” tropes. More detailed explanations appear in Section 1.4 on page 12 and Chapter 3 on page 43.

This dissertation proposes a method for authors to construct non-linear stories for multi-agent systems using a controlled natural language syntax. Our language, named *TropICAL*, allows a non-programmer story author to describe their stories in terms of re-usable tropes, which they can then combine together to form the shape of a story. The TropICAL language, and its development environment StoryBuilder, are described in Chapter 4 on page 54 and Chapter 5 on page 101 respectively.

In order to put our narrative authoring system into context, the next Section of this introduction contains a high-level overview of interactive narrative, describing its past and future use in and beyond entertainment. From this, we determine the criteria that make a story interactive, against which we evaluate our system in Chapter 7 on page 135.

1.1 Interactive Narrative

Though media such as computer games allow a player to influence events through interaction with its characters and objects, the story of the game itself is usually predetermined. In order to create media in which the narrative is shaped by a person’s interactions, one of two approaches are usually taken: emergent narratives or branching narratives.

Emergent narratives occur when a story (of sorts) unfolds as a result of a player’s interactions with the rules of the game world. Examples of this appear in the games *Minecraft*, *Civilisation*, *Dwarf Fortress* and *The Sims*. In these cases, no author has sat down and written a branching narrative for every eventuality in the game. Instead, the narrative is created by the story world itself in response to a player’s actions.

The advantage of this type of narrative is easily apparent from an author’s point of view: it saves them the work of having to write all the branches of a story. In a story with any nontrivial amount of branching, the author would have to spend a great many hours writing down all of the alternative paths through it. In the end, a player might not even see the result of a fraction of the effort put into a game’s writing. An emergent story solves this problem by procedurally generating a narrative.

However, it could be argued that these emergent stories are only “stories” in the loosest

possible sense — they probably have no shape or structure, no recurring motifs, and nothing like a climax near the end. They do not take the player on an emotional ride, but merely describe a sequence of events. Indeed, it greatly decreases the influence of the writer as an artist, leaving a story totally in the hands of the player. If a game author has a certain story they want to tell in a non-linear fashion, or a message that they wish to convey, they must resign themselves to hoping that it somehow emerges from the unscripted interactions of a player with the game.

Branching Narratives, on the other hand, give authors total control over what happens in a non-linear story. They are able to structure each possible branch of the narrative, ensuring some branches terminate in climaxes while others lead to despair. They can insert artistic themes into their stories, and put their own personal flourishes into every single aspect of it. The price of this control is the amount of work it requires for the creation of each story branch, as well as the limited amount of branching an author’s time constrains them to write.

Ideally, we could combine the best features of both types of non-linear narrative: the labour-saving generative element of emergent narratives and the structural control of the branching narratives, sharing control of the narrative between author and actor.

1.2 Interactive narrative for games

Computer games are a new medium for artistic expression. The element of interactivity, combined with visual art, music and storytelling, allows the creation of ever more fantastic worlds. Game creators are now experimenting with ways to make a narrative itself interactive. Imagine playing through a version of Romeo and Juliet where there is a possibility that the characters could escape their fates. Or playing a detective in a game where the story changes depending on how quickly you can piece together clues. This is the new frontier that we are exploring with this research: the domain of *interactive* storytelling.

We are defined by the choices we make throughout our lives. These choices form a story that describes our own personal history. If a player can watch somebody’s life unfold and witness the choices that they made, and those that were forced upon them, they would have a better understanding of how they came to be who they are. This kind of deep understanding is only possible through experiencing a story where the decisions made have real consequences. Traditional fiction, and perhaps computer games, enable this to some extent. But the story is still experienced passively by the consumer in these media. A truly interactive narrative would go deeper, as though you had truly lived as another person, forced to make the same decisions in the same situations. This has the potential to enable a whole new level of narrative immersion.

Our initial attempt at creating an interactive narrative “game” uses intelligent agents with emotional models to act out an interactive Punch and Judy puppet show. As the audience boos or cheers at the actions of the puppets, encouraging or scolding them, the agents’ emotional state is changed, affecting their decision-making process. Though this game may not offer much explanation for the irrationally violent actions of “Punch”, the audience can use the

system to explore their expectations of what a Punch and Judy show should be. This work is described in Section 6.2 on page 125.

The Punch and Judy show in Section 6.2 on page 125 also makes use of a *social institution* (or *normative framework*) to regulate the actions of the agents playing the roles of the puppets. Because the normative system only guides the actions of the agents, rather than forcing them to choose specific courses of action, their emotional models may affect their decision making in such a way that the agents choose to break away from the constraints of the behaviour suggested by norms, at times of extreme emotional distress for example.

1.3 Normative Systems

In the field of multi-agent systems, a *normative framework* or *social institution* coordinates the actions of the agents using the language of deontic logic: permissions and obligations that describe what each agent *may* or *must* do at each point in time.

Normative systems are often used to model legal contracts, so that the contracts are described in terms of social norms. This way, the contract is described as a social institution that each party adheres to. In the case of a contract dispute, the party in the wrong is the one that has broken the social contract, either by carrying out an unpermitted action, or by not executing an obliged action before its deadline. As in a social contract, there is nothing that physically stops either party from breaking the norms. The only barrier is the prospect of a possible penalty for deviating from the agreed-upon rules.

In our system, we model the narrative as a social institution, which regulates the behaviour of the character agents, rather than strictly regimenting their actions. This means that each character is technically able to break away from the story if they wish, but they would suffer grave consequences for doing so. This makes for an interactive storytelling system with a greater degree of unpredictability and variety, where an agent that is desperate to carry out a plan, or caught in the darkest state of an emotional model, may do something rash and spontaneous. This means that the simulated characters have more agency to do what they want, even though they are encouraged towards certain actions by the bounds of the story.

Describing a story in terms of permissions and obligations does not come naturally to most story authors, however. For this reason, we use story tropes as a kind of *user interface* through which non-programmer artists and authors may compose the social norms that govern the behaviour of the character agents.

1.4 Story Tropes

Tropes describe commonly-seen themes and elements of stories. In a sense, they are similar to clichés, in that an audience becomes familiar with them through repeated exposure. However, tropes are not clichés. Clichés are story elements that have been repeated ad nauseum, that are so often used and unoriginal that they evoke tedium from their audience. Tropes may perhaps become clichés if they are used to the point of becoming tiresome, though some

are so embedded into the collective subconscious, that their presence is taken for granted. Joseph Campbell argues that *The Hero's Journey* is a trope that transcends time and culture, representing a shared human blueprint for the rite-of-passage into adulthood (Campbell, 2008). In this way, stories and tropes could be considered a cultural language for describing common human themes. Clichés could not be described as such: they are simply overexploited ideas.

The themes, scenes, characters and structure of a story can be described in terms of tropes. Even individual lines of dialogue can be a trope, such as James Bond's witty put-downs when he defeats an enemy. Tropes describe parts of a story in an abstract way, which means that they can be easily identified in multiple stories.

Examples of tropes are:

- **The Hero's Journey:** A hero answers a call to adventure, and leaves home to go on a journey. The hero defeats the villain along the way, returning back home triumphant.
- **The Evil Empire:** The villain is part of an empire, which tries to stop the hero at all costs.
- **The MacGuffin:** There is an object that the hero needs, the search for which is used to drive the plot.
- **Chekhov's Gun:** If a gun appears somewhere in the first act, it must be fired by the end of the third.

Tropes can describe a story at several layers of abstraction, meaning that tropes can contain other tropes as sub-tropes. For example, *The Hero's Journey* can contain the *Hero*, *Quest*, and *Call to Adventure* sub-tropes. This allows a degree of expressivity unavailable in other narrative models described in the *Literature Review* chapter (Chapter 2 on page 18).

In interactive narrative generation, where a player may take one of many possible paths through a story, parts of the story are often procedurally generated in order to save an author the time and effort of manually writing out all the branches of the narrative. However, this means that the author must give up some control over the structure and content of the story. By describing the intended contents of the story in an abstract manner using tropes, an author is still able to give it structure. In our case, the procedural generation occurs through the use of intelligent agents to model story characters, which interact with the player's actions. These character agents are guided towards pursuing actions that fit the tropes in the authored narrative, thus gently making sure they meet the expectations of the author while allowing them some degree of freedom with which to pursue their own goals. Additionally, the player's allowed actions are constrained according to the defined tropes of the story, so that the narrative emerges as a result of the interactions that are taken as a result of the allowed actions, along with the way that the character agents respond to these interactions.

1.5 Answer Set Programming for Story Solving

In Chapter 4 on page 54, we describe TropICAL, our programming language for defining of story tropes. The requirements for this language are:

1. Easy to learn for authors who are unfamiliar with programming
2. Able to express sequences of events and branching events
3. Must have a mechanism for abstraction (embedding sub-tropes within tropes)

In order to satisfy requirement 1, TropICAL uses a controlled natural language syntax resembling that of the popular interactive fiction programming language Inform 7, designed to allow authors with no previous experience with programming to describe their story in terms of tropes. An author uses this language to combine and sequence their tropes together, using keywords such as “X Then Y” and “X Or Y” to specify sequences of events or branches (Requirement 2). Requirement 3, the mechanism for creating abstractions, is implemented by referring to other tropes by name (for example, *Then the “Quest” trope happens*).

When the tropes are compiled into the normative language InstAL (Cliffe et al., 2007), they are defined strictly in terms of permissions and obligations. InstAL allows the trope descriptions to be compiled a second time into AnsProlog, an Answer Set Programming (ASP) language, which can then be used with an Answer Set solver such as Potassco’s Clingo (Gebser et al., 2011) to generate all the different possible actions of each character at any point in the story, given the constraints expressed in the described tropes.

1.6 Outline

The structure of this thesis is as follows:

Chapter 2: A **Literature Review**. We approach the literature survey from two perspectives: those of *narratology*, and *interactive narrative*. In order to evaluate the current state of narrative formalisms, the literature review first covers narrative research (narratology) from social science (Section 2.1 on page 18), before examining existing implementations of interactive narrative and narrative generation systems from computer science (Section 2.2 on page 26).

Chapter 3: Using **Tropes as Story Components**: This chapter examines the use of *tropes* as story components, analysing the process of turning an informal trope description into something more formal that can improve upon existing narrative formalisms. This is linked to the work in Chapter 6 on page 117 by showing how tropes can be used as a “user interface” through which story authors would be able to describe the social institutions that govern story characters’ behaviours. Just as a computer user interface exposes the useful parts of a piece of software, hiding complexity that is irrelevant to the user, our tropes allow authors to create their story components

in controlled natural language, without having to worry about the complex code that results from the translation process.

- Chapter 4: We use the requirements that emerge from Chapter 2 on page 18 to Chapter 3 on page 43, along with those from specific use cases, to inform the design and implementation of our **TropICAL** programming language for tropes. This chapter explains the full rationale behind the language design, and gives examples of TropICAL code compiled to their formal institutional equivalents. The chapter concludes with examples of its use to build several stories. Work from this chapter appears in [ALIA2016], and in a legal context in [JURIX2016].
- Chapter 5: Keeping a mental model of the branches of a story is simple when only one trope is involved, but can be difficult when several tropes are active during a story. Branching narratives can be difficult to check once they start becoming at all complex. Tool support is needed to help visualise the changes in a story. For this reason, this chapter describes the implementation of the **StoryBuilder** tool, designed for the interactive creation and visualisation of tropes, allowing the user to combine tropes to describe the story world for an interactive narrative. We have carried out a qualitative evaluation of the tool through a thematic analysis of eight participants building their own stories with the tool.
- Chapter 6: This chapter, **Using TropICAL with a Multi-Agent System**, describes the integration of the stories described with TropICAL into a working story simulation environment with multiple agents. The chapter gives examples of converting tropes to social institutions in terms of norms, and how these norms would be used to govern a multi-agent system containing agents that have emotional models. This chapter is a development of *Governing Narrative Events with Institutional Norms*, which is listed as the [CMN 2015] conference paper in the list of papers below (Section 1.7 on the following page).
- Chapter 7: This chapter describes the **Evaluation and Validation** work done on the TropICAL and StoryBuilder systems. The chapter begins with a usability study conducted on the StoryBuilder tool, continues with a validation of the TropICAL system through a worked example of the full Punch and Judy story, and concludes with an example of “character freedom”, where a character is able to break away from the constraints of the story.
- Chapter 8: After the work done on evaluation and validation comes the **Discussion** chapter. This chapter reflects on the benefits and limitations on our approach by comparing it with existing planner-based approaches from the literature review, and describes to what extent “character freedom” has been achieved. The chapter concludes with an evaluation against the criteria for interactive stories described in the “narratology” section of the literature review.

Chapter 9: The final chapter, **Conclusions and Future Work**, reviews the research done, evaluating its potential impact and discusses possible future work.

1.7 Published Works

The following list includes all conference presentations and papers published by the author which are related to this report.

Conference Presentations:

[HSWI@ESWC 2014] Artfinder: A Faceted Browser for Cross-Cultural Art Discovery Matt Thompson, Julian Padget and Steve Battle, Human Semantic Web Interaction Workshop, European Semantic Web Conference, May 2014, Crete, Greece.

[DHIS 2015] Every Object Tells a Story Matt Thompson, Julian Padget and Steve Battle, Digital Heritage Meets Interactive Storytelling 2015 (conference presentation), April 2015, York UK.

[AISB 2015] An Interactive, Generative Punch and Judy Show Using Institutions, ASP and Emotional Agents Matt Thompson, Julian Padget and Steve Battle, AISB AI & Games Symposium, April 2015, Canterbury, UK.

[CMN 2015] Governing Narrative Events with Institutional Norms Matt Thompson, Julian Padget and Steve Battle, Sixth International Workshop on Computational Models of Narrative, May 2015, Atlanta, USA.

Published Proceedings:

[COIN@IJCAI 2015] An Interactive, Generative Punch and Judy Show Using Institutions, ASP and Emotional Agents Matt Thompson, Julian Padget and Steve Battle, International Workshop on Coordination, Organisation, Institutions and Norms in Multi-Agent Systems, July 2015, Buenos Aires, Argentina. (Thompson et al., 2015b)

[ICIDS 2015] Telling Non-linear Stories with Interval Temporal Logic Matt Thompson, Julian Padget and Steve Battle, International Conference on Interactive Digital Storytelling, December 2015, Copenhagen, Denmark. (Thompson et al., 2015a)

[JURIX 2016] Describing Legal Policies as Story Tropes in Normative Systems Matthew Thompson, Julian Padget and Ken Satoh, JURIX 29th International Conference on Legal Knowledge and Information Systems, December 2016, Nice, France. (Thompson et al., 2016)

[ALIA 2016] Governing Narrative Events With Tropes as Institutional Norms
Matt Thompson, Julian Padget and Steve Battle, 2nd International Symposium on Artificial
Life and Intelligent Agents, April 2016, Birmingham, UK (proceedings yet to be finalised).

Chapter 2

Literature Review

This research covers multiple fields of study such as narratology, interactive storytelling, emotional modelling and artificial intelligence, therefore an extensive literature review covering these fields is necessary. This section starts with a look at the field of *narratology*, or narrative theory, to gain some insights into the themes and components that make up stories. Looking at different formalisms that have been created for narrative and which themes and motifs recur in stories should better inform the creation of techniques with which to generate stories.

We also examine current approaches to interactive narrative generation, focussing particularly on the use of planners, drama managers, social norms and logical modelling for describing stories. Most of these implementations are designed with multi-agent systems in mind, where the story forms a set of rules which govern these agents. The start of this part of the literature survey also examines non-interactive story generation through means of techniques such as generative grammar, in order to provide a historical context for the research that follows.

As part of the examination of implementations of interactive narrative, this review especially focuses on agent-based systems. The section concludes with an overview of emotional models that can be used to model distinct characters using agents.

2.1 Narratology

Narratology is a deep field with many sub-fields. This review examines the parts of it that might best inform the modelling of narrative by computers, as well as the construction of interactive narrative.

The first part of the overview of narratology examines research into categorising different types of narrative, both traditional and experimental. This draws from classic narratological texts, as well as work done on “cybertext” and experimental narrative in the interactive age. This examination of recent research into non-linear narratives is essential for the construction of interactive narratives for games or simulations.

Formalisms of narrative attempt to explain how stories work by dividing them into commonly occurring themes and motifs. This is a natural fit to the modelling of narratives by computer. This overview of narratology starts with an examination of formalisms for this

reason.

After the overview of narrative models there follows a section on the use of formal logic for narrative modelling. The section ends with descriptions of other types of story components, taxonomies and ontologies used in the literature.

2.1.1 Narrative Structure

What is the difference between a narrative and a sequence of events? If we recount the events that happened to us during the course of a day, would that “count” as a narrative? When is the retelling of events a simple listing of facts, and when is it a story?

Narratologists from Bal (2009) onwards refer to the chronological ordering of events as *fabula* (from the Russian *фабула*, “scene”), and the retelling and reordering of those events in a narrative as *syuzhet* (*сюжет*, “plot”). In fact, Russian formalists Propp (1968) and structuralists Shklovsky (1991) were the first to use these terms in the context of narrative, before their rediscovery by modern narrative theorists.

The concepts of *fabula* and *syuzhet* are key to the understanding of narrative structure. Thanks to Aristotle’s *Poetics* (Halliwell, 1986), we understand that a story must have a beginning, a middle and an end. These three parts of a story describe how the *syuzhet* are organised, which can refer to its *fabula* (chronological) events in any order. For example, the beginning of a movie could be set at the present day in the life of the protagonist, the middle could be a flashback to an earlier time in her life, and the end could return to the events following the start of the movie.

Though Aristotle identifies the beginning, middle and end as three key story divisions, other theorists divide the *syuzhet* of a story further. Joseph Campbell’s work *The Hero with a Thousand Faces* (Campbell, 2008) describes how almost every story is a variation of the *Hero’s Journey* (which Campbell also calls the *monomyth*). Like Aristotle’s beginning, middle and end, the Hero’s Journey has three acts: the Departure, the Initiation and the Return. Campbell divides these acts further into seventeen distinct stages:

Departure

1. The Call to Adventure
2. Refusal of the Call
3. Supernatural Aid
4. Crossing the Threshold
5. Belly of the Whale

Initiation

6. The Road of Trials
7. The Meeting with the Goddess

8. Woman as Temptress
9. Atonement with the Father
10. Apotheosis
11. The Ultimate Boon

Return

12. Refusal of the Return
13. The Magic Flight
14. Rescue from Without
15. The Crossing of the Return Threshold
16. Master of Two Worlds
17. Freedom to Live

In summary, the hero begins the story at home or in some otherwise familiar setting, where she is called away on an adventure. After possibly rejecting this call, the hero leaves home (crosses the threshold) and sets off for the land of adventure. After facing many trials and possibly defeating an enemy, the hero returns home once again. Having endured the many trials of the journey, our hero becomes stronger in spirit and character.

As a classic example, consider the plot of Tolkien's *The Hobbit*. At the start of the tale, Bilbo Baggins is a comfortable but risk-averse hobbit who refuses to leave his comfortable surroundings when Gandalf first visits to send him on a quest. Of course, he does eventually leave home to fulfill the quest, returning home a changed character.

Campbell argues that the *monomyth* acts as a shared cultural memory of sorts, being the universal template for the rite-of-passage tale. He says that it is passed down as myth through many different cultures, replicating in much the same way as genes (or memes) do and that it can be thought of as a *metamyth*, or the spiritual history of humanity.

Other analysts have described several different types of commonly-seen plot. Harris (1959) makes the case for just three types of plot, having either a happy ending (where the protagonist is virtuous), an unhappy ending (with a selfish protagonist) or being a tragedy (where the protagonist is struck by fate).

Booker (2004) describes seven basic plots, where each may have either a happy or unhappy ending:

- Overcoming the monster
- Rags to riches
- The quest

- Voyage and return
- Comedy
- Tragedy
- Rebirth

Georges Polti famously divides narratives into the *thirty-six dramatic situations* (Polti, 1921), derived from his analysis of classical Greek and both classical and contemporary French texts. Polti's thirty-six types of plot are:

- | | |
|---------------------------------------|---|
| 1. Supplication | 19. Slaying of kin unrecognized |
| 2. Deliverance | 20. Self-sacrifice for an ideal |
| 3. Crime pursued by vengeance | 21. Self-sacrifice for kin |
| 4. Vengeance taken for kin upon kin | 22. All sacrificed for passion |
| 5. Pursuit | 23. Necessity of sacrificing loved ones |
| 6. Disaster | 24. Rivalry of superior vs. inferior |
| 7. Falling prey to cruelty/misfortune | 25. Adultery |
| 8. Revolt | 26. Crimes of love |
| 9. Daring enterprise | 27. Discovery of the dishonour of a loved one |
| 10. Abduction | 28. Obstacles to love |
| 11. The enigma | 29. An enemy loved |
| 12. Obtaining | 30. Ambition |
| 13. Enmity of kin | 31. Conflict with a god |
| 14. Rivalry of kin | 32. Mistaken jealousy |
| 15. Murderous adultery | 33. Erroneous judgment |
| 16. Madness | 34. Remorse |
| 17. Fatal imprudence | 35. Recovery of a lost one |
| 18. Involuntary crimes of love | 36. Loss of loved ones |

Although the above lists are useful in the categorisation of different types of story, they do not tell us about how each story is structured, only the overarching themes. Author Kurt Vonnegut recognises that a story must also contain moments of rising and falling action, such as times where the hero experiences a struggle or climactic victory. In order to describe the

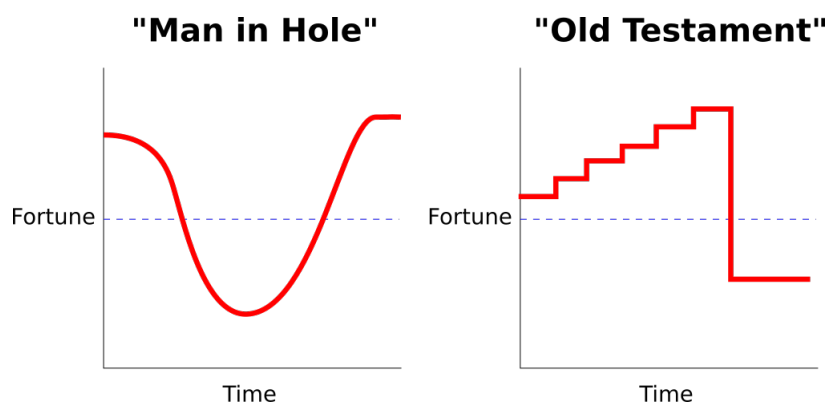


Figure 2-1: Two of Kurt Vonnegut’s “Shapes of Stories”

“ups and downs” of a narrative, Vonnegut describes several different “shapes” of stories in his rejected Master’s thesis (Vonnegut, 2009). He plots the events of a story as points in a 2D space with “Beginning-End” (time) on the X-axis and “Ill Fortune-Great Fortune” (fortune of protagonist) on the Y-axis. Figure 2-1 shows two examples of story shapes. In the “Man in Hole” example, things start out well for the story’s protagonist, but then some misfortune falls on them. They spend much of the story overcoming challenges before finally ending the story with a triumph. In the “Old Testament” example, the protagonist(s) experience gradually increasing levels of fortune, before finally being “cast down” to a grave level of misfortune at the end of the story.

All of the above categorisations describe stories as a whole, sorting narratives into one of several different “types” of story. This works well for linear narratives within traditional media, but what about experimental and non-linear narratives? In the next section, we review the literature relating to these less traditional types of stories.

2.1.2 Types of Narrative

The rise of the Web in the 1990s brought with it great interest in the future of narrative in cyberspace. Aarseth’s work, *Cybertext* (Aarseth, 1997) describes the creation of a new form of narrative, for which he coins the term *ergodic literature* (from the Greek words *ergon* and *hodos*, meaning ‘work’ and ‘path’). In this new form of narrative, some amount of work or effort is required by the reader in order to traverse the path that the story takes.

Aarseth makes a distinction between the narrative as written by the author, and the way in which it is traversed by the reader, calling the former *textons* and the latter *scriptons*. In ergodic literature, the *scripton* is produced by the effort that the reader goes through in interpreting the *texton*. In the context of a game, it is as though the game interface is a gateway that allows access to the narrative at different times. Using classical music as a metaphor, the *texton* can be thought of as the *score*, and the *scripton* the *performance*.

In Section 1.1 on page 10, we assert that how generative a narrative is and its level of interactivity are two different variables in an experimental narrative. However, Aarseth identifies seven different methods of story traversal: *dynamics*, *determinability*, *transiency*, *perspective*,

access, linking and user function.

Dynamics describe whether or not the content and number of scriptons changes. In a simple, static story with branching choices (such as in a *Choose your own adventure* story), both the number of textons and scriptons are fixed, since all paths have been written out beforehand. A dynamic story would still have a fixed number of textons, but the scriptons would be generated as the user traverses the path of the narrative.

Determinability is how deterministic the narrative is, whether or not the same interactions will result in the same scripton being produced.

Transiency means to what extent scriptons are produced as time flows, or whether user interactions are required to produce them.

Perspective is whether or not the user/reader plays a role as a character in the narrative.

Access is whether a user has access to all scriptons at any point in traversing the narrative, or whether their access is restricted.

Linking is whether or not parts of the scripton are linked to other parts, and whether these links are conditional (if they rely on a user having already traversed part of the scripton).

User functions: the functions the user uses to traverse the text. This could be interpretive (which is implicit in any traversal of the text), explorative (traversing the scripton according to whim) or configurative (specifying parts of the scripton in advance), for example.

By performing correspondence analysis (a process similar to principle component analysis) on a diverse corpus of 23 texts “*ranging from ancient China to the Internet*”, Aarseth filters these seven variables down into two numerical axes which account for 49 percent of the variation between stories. Using these axes, he groups classic tales such as *Moby Dick* and more experimental narratives such as William Gibson’s *Agrippa* and Michael Joyce’s *Afternoon*. By grouping these stories into categories, he intends to show how emerging media are enabling new types of story.

Chris Crawford’s *Chris Crawford on Interactive Storytelling* (Crawford, 2012) provides a scathing assessment of the relationship between narrative theory and computer science. A veteran of the games industry, he argues that ‘soft’ science theories such as those of Aarseth et al are entirely removed from ‘hard’ science, and are therefore an example of bubble intellectualism and impossible to implement.

Crawford himself provides a useful examination of experimental narrative in computer games, defining interactivity as:

A cyclic process between two or more active agents in which each agent alternately listens, thinks, and speaks.

He argues that for game narratives to be truly interactive, they must be more social. Characters in a story must be able to react with the player as though they were people in real life. In turn, the player should have some degree of freedom in the way in which they interact. Rather than presenting branching story points as choices, a better way to interact would be socially, through talking to agents in the game. This is the approach that Façade takes (Mateas and Stern, 2003), which Crawford acknowledges as the most successful attempt

at interactive storytelling to date. A detailed description of Façade’s implementation appears in Section 2.2.2 on page 30.

In order to determine whether Crawford’s assertion that narratology research is too far removed from its practical implementations to be of use, we provide an overview of these implementations and their underlying research in Section 2.2 on page 26. Has narrative theory research informed the creation of computer-generated or interactive narrative at all, or do they all take approaches grounded in computer science and artificial intelligence? If narrative theory has not been used, then we must ask: why not?

2.1.3 Formalisms of Narrative

Attempts to organise recurring themes, roles and motifs of narrative go back at least a century. The Aarne-Thompson tale-type index (Aarne and Thompson, 1987), first published in 1910 and later refined by Stith Thompson in 1928 and 1961, is well known amongst folklorists as a classification and analysis method for traditional folktales and myths. Aarne-Thompson’s index is a taxonomy of tale themes, arranging tales into categories such as *animal tales* and *jokes and anecdotes*, and then sub-categories (*tales of magic* and *numskull [sic] stories* being two examples). This taxonomy is only two levels deep however, and only serves as a useful way to categorise individual stories or tales. In order to break down and analyse components of tales, we must dig deeper.

In *Structural Anthropology*, Claude Lévi-Strauss seeks to discover why myths and legends are so similar across cultures and history (Lévi-Strauss, 1963). He concludes that there are global laws that govern the way in which people create stories, therefore these laws can be modelled as a set of rules for describing myths.

His theory is that myths describe opposing forces which are resolved through mediation. The example he gives in *Structural Anthropology* describes how Native American legends often contain ‘trickster’ characters in the form of ravens or coyotes. As scavenging animals, these tricksters symbolically act as mediators between life and death.

While interesting, Lévi-Strauss’ ideas bring us no closer to developing a formal model of narrative structure. For that, we must go even further back in time, and turn to Vladimir Propp.

Propp’s Morphology of the Folktale

A notable narrative formalist is Vladimir Propp, creator of *The Morphology of the Folktale* (Propp, 1968), a formalism for Russian folktales. Propp’s formalism, though originally limited in scope, generalises well, and is still used by researchers to procedurally generate stories (Grasbon and Braun, 2001; Gervás et al., 2005; Hartmann et al., 2005). Drawing from a corpus of one hundred Russian folktales, Propp identifies thirty-one distinct *story functions*, each of which is identified by a number and symbol. These functions are executed by characters fulfilling certain roles, each of which has a *sphere of action* consisting of the functions that they are able to perform at any given point of the story. Stories are created by chaining

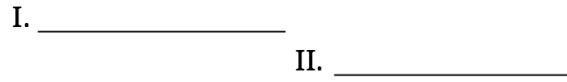


Figure 2-2: One Propp function following another

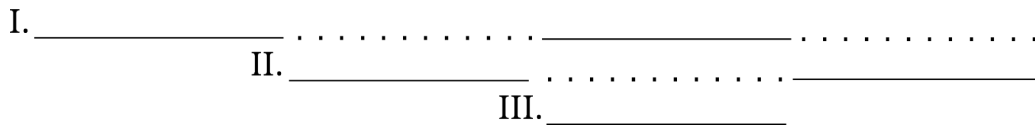


Figure 2-3: Multiple simultaneous functions

story functions together, with subplots expressed as parallel chains of story functions.

In this formalism, characters have *roles*, such as *hero*, *villain*, *dispatcher*, *false hero*, and more. Characters performing a certain role are able to perform a subset of *story functions*, which are actions that make the narrative progress. For example, the *dispatcher* might send the *hero* on a quest, or the *victim* may issue an *interdiction* to the *villain*, which is then *violated*.

Propp defines a total of 31 distinct story functions, each of which is given a number and symbol in order to create a succinct way of describing entire stories. Examples of such functions are:

- One of the members of a family absents himself from home: *absentation*.
- An interdiction is addressed to the hero: *interdiction*.
- The victim submits to deception and thereby unwittingly helps his enemy: *complicity*.
- The villain causes harm or injury to a member of the family: *villainy*.

Each of these functions can vary to some degree. For example, the *villainy* function can be realised as one of 19 distinct forms of villainous deed, including *the villain abducts a person*, *the villain seizes the daylight*, and *the villain makes a threat of cannibalism*.

In a typical story, one story function will follow another as the tale progresses in a sequential series of cause and effect (figure Figure 2-2). However, Propp’s formalism also allows for simultaneous story functions to be occurring at once (figure Figure 2-3).

2.1.4 Other Types of “Story Component”

Lehnert’s *plot units* are a more recent narrative formalism (Lehnert, 1981). However, these plot units only describe stories as three types of event: positive, negative and mental. These events occur with respect to a single character in the story, so an author must always author story components with concrete characters in mind, making them difficult to re-use. Similar to Propp’s system, the order of composition must always be in a certain sequence, and plot units cannot refer to other plot units. Again, we are left without a means of creating abstractions for our story components. Chapter 4 on page 54 addresses this issue by describing how tropes allow the nesting of components to allow story authors to create their own abstractions.

1. Attempt \rightarrow Plan + Application
 \Rightarrow MOTIVATE(Plan, Application)
2. Application \rightarrow (Preaction)* + Action + Consequence
 \Rightarrow Allow(AND(Preaction,Preaction,...),
 {CAUSE | INITIATE | ALLOW}(Action,Consequence))

Figure 2-4: Example rules from Rumelhart's story grammar

2.2 Implementations of Experimental Narrative

2.2.1 Story Generation

Inspired by Chomsky's theories of generative grammar (Chomsky and Halle, 1968), researchers in generative narrative strive to build their own 'universal grammar' for narrative.

Rumelhart (1975) provides one early and influential model for the grammatical generation of natural language. Figure 2-4 shows two example rewrite rules from this grammar. Each rule pair consists of a syntactic rule (following the ' \rightarrow ' symbol) and a semantic interpretation rule (following the ' \Rightarrow ' symbol). The '+' symbol denotes items happening in a sequence, the '|' showing possible alternatives. '*' denotes one or more item being generated. Capitalised words (such as ALLOW, MOTIVATE, etc) describe relationships between items. For example, MOTIVATE is a relationship between a character's thought and their reaction to that thought.

Other systems draw inspiration from generative grammar, such as GESTER (Pemberton, 1989), which generates stories based on a grammar synthesised from old French epic tales. Lang (1999) describes a declarative model for narrative, consisting of lists of first-order predicate calculus expressions. These expressions describe events, states, goals and beliefs which combine to form a narrative. More specifically, it combines:

- A **grammar interpreter** to search for a sequence of grammar rewrites which would produce a convincing narrative.
- A set of **temporal predicates** to describe the occurrence of events over time and enforce temporal constraints on story components.
- A **world model** which describes the set of actions that characters may perform and fluents that may alter over the course of the narrative.
- A **natural language output unit**, which takes the sequence of events produced by the story grammar and converts it into readable natural language sentences.

This combination of using a grammar interpreter, world model and natural language output unit is especially common amongst generative grammar approaches.

While generative grammar approaches may be effective for procedurally creating prose, they are less well suited to the creation of *interactive* narratives. Once the grammar rewrite rules are specified, the user is entirely passive, unable to affect the way in which the story is

ONCE UPON A TIME GEORGE ANT LIVED NEAR A PATCH OF GROUND. THERE WAS A NEST IN AN ASH TREE. WILMA BIRD LIVED IN THE NEST. THERE WAS SOME WATER IN A RIVER. WILMA KNEW THAT THE WATER WAS IN THE RIVER. GEORGE KNEW THAT THE WATER WAS IN THE RIVER. ONE DAY WILMA WAS VERY THIRSTY. WILMA WANTED TO GET NEAR SOME WATER. WILMA FLEW FROM HER NEST ACROSS THE MEADOW THROUGH A VALLEY TO THE RIVER. WILMA DRANK THE WATER. WILMA WASN'T THIRSTY ANYMORE.

GEORGE WAS VERY THIRSTY. GEORGE WANTED TO GET NEAR SOME WATER. GEORGE WALKED FROM HIS PATCH OF GROUND ACROSS THE MEADOW THROUGH THE VALLEY TO A RIVER. GEORGE FELL INTO THE WATER. GEORGE WANTED TO GET NEAR THE VALLEY. GEORGE COULDN'T GET NEAR THE VALLEY. GEORGE WANTED TO GET NEAR THE MEADOW. GEORGE COULDN'T GET NEAR THE MEADOW. WILMA WANTED TO GET NEAR GEORGE. WILMA GRABBED GEORGE WITH HER CLAW. WILMA TOOK GEORGE FROM THE RIVER THROUGH THE VALLEY TO THE MEADOW. GEORGE WAS DEVOTED TO WILMA. GEORGE OWED EVERYTHING TO WILMA. WILMA LET GO OF GEORGE. GEORGE FELL TO THE MEADOW. THE END.

Figure 2-5: Example TALE-SPIN output

being generated. For this to happen, the narrative needs to be part of a system that reacts to the actions of the user, such as in a computer game.

James Meehan's TALE-SPIN (Meehan, 1977) is an influential early approach to story generation using planning. In TALE-SPIN, the author describes a story domain, its characters and their goals, and a natural language story is produced as output. It works by using a problem-solver to resolve each character's goals over the story domain. Figure 2-5 is an example of TALE-SPIN's output.

TALE-SPIN's strong planning component is evident in the reading of its output. Sentences are terse, with one event leading directly to another in order to achieve some goal. One problem with this character-led approach is that the goals of the author are not necessarily taken into account. If the author intends for a character to die at some point in the story, it seems unnatural for a character to have the goal of dying to fulfill this intention.

Turner criticises TALE-SPIN's stories as seeming "pointless and somewhat boring" (Turner and Dyer, 1986), going on to create the MINSTREL system for story generation (Turner, 1993). Using the legendary world of King Arthur's court as a story domain, MINSTREL strives to generate stories with an authorial purpose.

MINSTREL attempts to address TALE-SPIN's shortcomings by introducing two types of schema: author-schemas and character-schemas, both of which combine to represent the elements of a story. The author-schemas describe the goals of the author of the system, allowing story creators more control over the structure and content of their narrative. This allows authors to specify a 'point' or moral to their story, something that is not possible to achieve with TALE-SPIN. Character-schemas describe character-level goals in a similar

manner to those of TALE-SPIN's.

The comparison of TALE-SPIN and MINSTREL highlights a challenge that has dominated story generation for decades: the balance of *character* and *plot* (as Riedl and Young (2010) highlights). Especially with approaches based on multi-agent systems, the regulation of character actions to conform to an underlying theme or structure is a challenging problem.

However, modelling characters with agents is a promising approach to take in order to achieve a story which is both generative and interactive. In such a system, an author can specify the story world and character models, creating the 'big picture', and the agents would be able to fill in the details (such as dialogue, sub-plots, and relationships). An apt metaphor would be that of animation. In large animation studios such as Disney, the lead animator draws the key frames of a sequence, and a team of other animators work to fill in the gaps in between¹. This is what the combination of a managed narrative with agents could achieve: the author would be the 'lead animator' in such a system, with the agents being the assistants.

The implementations until now have focused mainly on *generation*, and little on *interaction*. Character models have been mentioned, but these do not react in real time to a user. For that, we need a multi-agent system.

2.2.2 Characters as Intelligent Agents

Story worlds are usually populated with characters. Interactive story worlds such as games contain characters with very basic scripted behaviour. At even the highest level of game character simulation, AI techniques are usually used to govern basic behaviour such as movements and actions. Governing character behaviour to fit within the context of a narrative is a more challenging problem. This section examines different approaches to tackling this challenge.

Planner-based Systems

The most prevalent approach to the generation and management of plot in interactive narrative is to use planners. With a planner, an author sets the goals for the story (certain situations that they would like to see happen), and the planner tells the character agents what to do to make sure these "story goals" are achieved. When a player that is interacting with the system takes an action that compromises the story goals, the planner must re-plan to make sure the goals can still be achieved, by restricting the actions of the player or intervening in some other manner.

Young (1999) argues that planners are a good method for regulating plot, later creating the *Mimesis* architecture for integrating a planner with character agents in an interactive game environment, (Young et al., 2004). Young describes how narrative systems must re-plan when a player makes narrative-breaking actions, by either restructuring the narrative mid-story (*accommodating* the action) or preventing the action from executing (*intervening* on the action).

¹This process is described at the following website: <http://www.justdisney.com/animation/animation.html>, accessed 20160805.

Given its influence over subsequent approaches to interactive narrative generation, it is worth looking at the Mimesis architecture more closely. It is designed to integrate into the *Unreal Tournament* game engine, and has five components: the *mimesis controller*, the *story planner*, the *discourse planner*, the *execution manager* and the *MWorld*.

Figure 2-6 on the following page shows how these components work together to form the Mimesis architecture. Once an author has created the pre-defined libraries of actions needed by the story planner, the following steps are taken to determine the course of the story:

1. All the components connect to the Mimesis Controller (MC) via socket connection. It then acts as a message router.
2. The game initiates a plan request containing the state of the game world, a list of possible actions, and the goals for the plan.
3. The *story planner* responds with a *story world plan*, a data structure that describes the actions (selected from the list) that must occur over time in order to meet the plan request's goals.
4. Once the story world plan is created, it is sent to the *discourse planner*, along with a list of actions that can occur in the game engine (such as camera movements, voice-overs and background music). The discourse planner then creates a sequence of these actions that best fit the story world plan.
5. The discourse planner then sends the narrative plan to the *execution manager*, which builds a directed acyclic graph (DAG), where the nodes are the actions within the plan and the edges are the temporal constraints between the orderings of the actions. The execution manager removes nodes from the DAG in order, sends the node's actions to the game engine, and updates the graph.
6. The *MWorld* is essentially the environment in which the story occurs, consisting of the game engine, coordination code, and class definitions for actions, and discourse planners. It is the *MWorld* that receives actions from the execution manager to be executed, and executes these actions in the game engine.

Mimesis uses DPOCL (Decomposed Partial-Order Causal-Link Planner, (Young et al., 1994)) plans for its story planning. DPOCL plans are composed of *steps* (the plan's actions), *ordering constraints*, *decomposition links* describing the hierarchical structure of a plan, and *causal links* between pairs of steps. It uses *refinement search* (Kambhampati et al., 1995) as its plan reasoning process, searching through the space of possible plans represented as a directed graph, with each node in the graph being a plan or partial plan. Mimesis specifies the initial planning problem for DPOCL using the current and goal states of the story.

A key feature of Mimesis is its strategies for handling of user actions which potentially interfere with the story plan, making its goals unachievable. In the intervention strategy, Mimesis simply prevents the user's action from having any effect in the game world. With

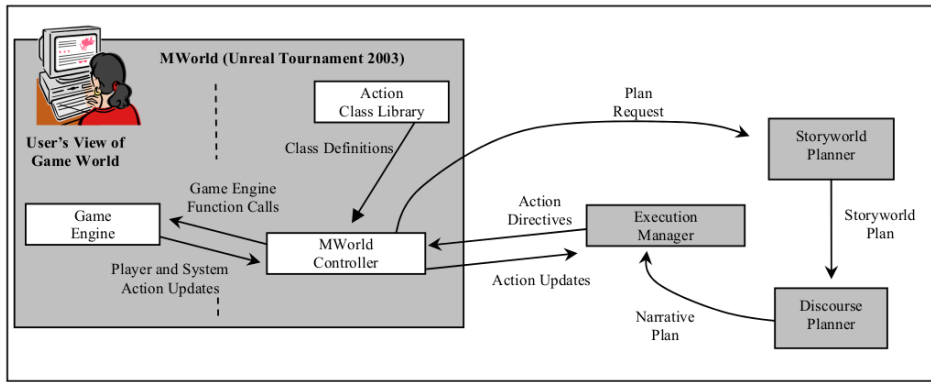


Figure 2-6: The Mimesis architecture, from Young et al. (2004)

accommodation, Mimesis replans the story events, restructuring the plan so that the interfering actions are taken into account and worked around.

Cavazza et al. (2002)'s *I-Storytelling* system implements Young et al. (2004)'s architecture with ideas from Barthes' narrative units (Barthes and Duisit, 1975), using characters with behaviour described by Hierarchical Task Networks (HTNs) to generate its stories. Each character has a main task, which is divided into subtasks to create a task hierarchy, with each task node having pre- and post-conditions. The story emerges from the outcomes of each character's plans, and the narrative structure as a whole is not planned.

Riedl et al. (2003) describes a further development of Young's architecture, allowing the story author to create plans for the overall narrative in addition to its characters. Rather than using a narrative model, this version of Mimesis models the player to track their expected level of suspense while interacting with the story. Like the *I-storytelling* system, its plans are hierarchical, using the Longbow hierarchical partial-order causal link planning system (Young et al., 1994).

A disadvantage of these planner-based systems is that they require the story author to think in a planner-oriented manner. They must consider the goals of both the story and the character, plans to achieve these goals, and re-planning when goals are not met or when situations change. This is a drastic change from the usual story writing methods of authors, where the focus is on structure, plot, themes and characters. Though graphical user interfaces such as Mimesis' Bowman system (Thomas and Young, 2006) could be used to assist plan-driven story authoring, a complete shift in creative workflow is still required, making this approach inaccessible to non-technical writers.

Drama manager-based Systems

Carnegie Mellon University's OZ project (Mateas, 1999) uses a *drama manager* to structure its narrative, which observes the actions occurring in the storyworld and "directs" its characters to conform to shape the story. A screenshot from the game is shown in Figure 2-7 on the next page.

Mateas and Stern's *Façade* (Mateas and Stern, 2003) has players interact with the char-

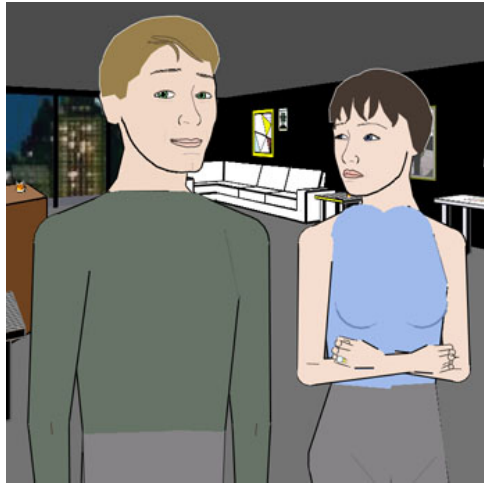


Figure 2-7: A screenshot from *Façade* (Mateas and Stern, 2003)

acters of the story through natural language. In this game, the player attends the party of a young couple (Grace and Trip) celebrating their wedding anniversary. As the course of events unfold however, the player learns that all is not as happy as it seems.

The player interacts with the characters by typing in natural language sentences, to which Grace and Trip respond. Though the characters are implemented through agents, the story is controlled using a drama manager. In all, their system consists of using Natural Language Processing (NLP), a novel character authoring language and a novel drama manager to create an interactive narrative.

Several custom-designed languages were used to create the game, including a language called ‘A Behaviour Language’ (ABL) for the agents and a special language for the sequencing of the beats. ABL represents situations as character goals, maintaining a tree of all the active goals and behaviours that are happening at any time.

In *Façade*, the smallest unit of narrative action is called a *story beat*, taken from McKee’s book on authorial style for screenwriters (McKee, 1997). The simulation constantly monitors what the user is doing and how it may lead from the current story beat to another. Story beats have preconditions and effects on the state of the narrative, so it is the drama manager’s job to work out when it makes sense to initiate a certain beat.

‘Beats’ have a very fine granularity, with 200 or so updating every minute of the simulation. They consist of a set of ABL behaviours, which advance the narrative yet still allow interaction to change to other beats. Only one beat can be active at a time.

A beat can have 5 types of goal:

- **Transition-in:** characters express their intentions
- **Body:** a dramatic question/situation is posed to the player
- **Local/global mix-in:** react to the player before end of the beat
- **Wait-with-timeout:** wait for the player’s reaction

- **Transition-out:** final reaction to the player’s action in the beat

A beat goal is a series of steps for an agent to perform, which can be:

- staging (where to walk to, face)
- dialogue to speak
- where and how to gaze
- arm gestures to perform
- facial expression to perform
- head and face gestures to perform
- small arm and head emphasis motions triggered by dialogue (head nods, hand flourishes)

As an example, there is a behaviour called “Fix_Drinks”, which specifies a sequence of agent behaviours where the characters Grace and Trip have an argument while Trip asks the player what they would like to drink. If the player decides not to go along with the beat (in this case, by not choosing a drink), then the beat will be aborted and replaced with another.

Façade has become popular as a game outside of academia, with playthroughs of the game reaching millions of views on Youtube. This shows the promise of interactive narrative as being a unique and engaging new form of entertainment. Unfortunately, no other implementation of interactive narrative seems to have captured the public imagination since the release of Façade.

Façade’s popularity seems to reinforce Crawford’s assertion (as we mention in Section 2.1.2 on page 22) that interactive narratives must be social in nature. The gameplay comes entirely from the conversations and interactions between Grace, Trip and the player. Much of the excitement comes from the social consequences of certain conversation paths or actions. By modelling characters as agents, Mateas and Stern Mateas and Stern (2003) have created a truly interactive experience. However, by also using a drama manager to manage the agents, they have used these agents to tell a story.

How might these agents be made more convincing? Outside of writing rules for their behaviour consisting of character goals and beliefs, how might an author create truly unique and idiosyncratic characters? To address the question, we next examine different types of emotional models in psychology, and how each might be used to model characters as agents.

Social Norms

Versu (Evans and Short, 2014) is an interactive drama system that uses a multi agent system as characters. The characters’ actions are coordinated with *social practices*, which describe types of social situations and is described by the authors as a successor to the Schankian script. These social practices are implemented as reactive joint plans, which agents can choose to participate in or not. Rather than directly telling the agents what to do, these social practices

Listing 2.1: nextHopInfo: caption

```
1 A(agent).sex!G(gender).
```

Listing 2.2: Description of “Brown” character.

```
1 brown.sex!male;
2 brown.class!upper;
3 brown.in!living_room;
4 brown.in.living_room.activity!watching_tv;
5 brown.relationship.lucy.evaluation.attractive!40;
6 brown.relationship.lucy.evaluation.humour!20.
```

merely *suggest* courses of action, leaving each agent to decide for itself what to do based on its individual goals.

The authors decide against using a drama manager to control the agents’ actions because they want to take the *strong autonomy* approach to agent governance. This means that they prefer to give each agent some degree of autonomy by allowing it to make the final decision on which course of action to take, rather than blindly following a drama manager. Suggesting actions with social norms achieves this goal. Rather than describing typical story events in terms of social norms, however, in Versu the social norms *are* the story. The gameplay revolves around the avoidance (or purposeful subversion of) awkward social situations.

Each character has a role, which is governed by a social practice. For example, a *greeting* practice involves characters with the *greeter* and recipient roles. The greeting practice would tell the greeter in which manner they are to greet the recipient, and the recipient how to respond. It is noteworthy that these actions are merely suggested, and not enforced.

Exclusion logic (Evans, 2010) is used to describe the social practices of the system. Exclusion logic manages the frame problem by organising related fluents in a tree structure. A single update at the right branch can change a set of related facts en masse. This is achieved through use of an exclusion operator (“!”). Listing 2.1 shows an example of the exclusion operator in use. The example specifies that an agent can have only one gender. The *Praxis* language implementation of the exclusion logic in Listing 2.1 has a type checker which ensures that no character can have multiple genders.

Listing 2.2 is another piece of *Praxis* code showing how the attributes of a character called “Brown” are built up as a tree structure. Exclusion logic’s exclusion operator allows an author to express the fact that a variable can only have one value. For example, if ‘the ‘Brown’ character changes location from the living room to the kitchen, *brown.in!living_room* is terminated when *brown.in!kitchen* holds. This also terminates any attributes further down the tree, so in this case the *brown.in.living_room.activity* of *watching_tv* will also be terminated when the location updates.

Versu takes the *constitutive* view of social practice, as opposed to the *regulative* view (Friedman, 1992). This means that rather than restricting an agent’s possible actions based on its permissions and obligations, they participate in a certain social practice by taking an action. Their actions are only restricted by what is possible in the story world, and what the agent

desires to do. This way, agents can choose whether or not to take part in certain social interactions.

Many of the components we aim to have in our story telling system appear in Versu: the use of social norms to gently encourage story-conforming behaviour rather than demanding it, and the use of formal logic to determine which behaviours are possible. However in Versu the social norms *are* the story, rather than describing the story components that invisibly govern the behaviour of characters. In order for this kind of governance to occur, an institution-based solution is preferable, based on events, agent actions and standard deontic logic. Because character actions are constrained by the structure of a story, a *regulative* view of social practice is more suited to the expression of story components as social norms.

Many of the advantages of using exclusion logic can be gained by using an institution-based approach. Non-inertial fluents can be used to ensure that variables can only ever have one value. Standard deontic logic (Von Wright, 1951) is enough to provide the rest of what is needed.

2.2.3 Modelling Narrative with Logic

Although most recent research focuses on the use of planners to manage the drama in a story, there is also much interesting work which makes use of formal logic to model narrative. Though often used for the generation of linear story text, it is increasingly being applied to non-linear narratives as well. Logic-based approaches are generally based on either temporal logic variants or some kind of linear logic.

Ceptre (Martens, 2015) is a language for modelling generative interactive narratives using *linear logic*, a formal logic designed to describe resource usage.

A Ceptre story begins with an initial state Δ_0 . Each state iteration Δ_i is examined repeatedly, and a subset S of it is updated with rule r . The next state, Δ_{i+1} , has the subset S replaced with S' , the new subset with the consequences of the applied rule r .

The rules are specified using the combination of logical statements with two operators: $*$ (tensor) and $-o$ (loli). The tensor operator is used to concatenate statements, while the loli operator expresses state transitions in the form $S -o S''$. The rules use *replacement semantics*, which means that everything from state S will disappear unless stated to be in state S' . A $\$$ operator is used to mark facts in S that the author wishes to remain in S without explicitly stating so.

Listing 2.2.3 shows an example from Martens (2015) that describes a “murder” rule and its consequences.

```

1 do/murder
2   : anger C C' * anger C C' * anger C C' * anger C C' *
3   $at C L * at C' L * $has C weapon
4   -o !dead C'.
```

In this case, four instances of the “anger” predicate with the same arguments has a significant meaning: a character’s emotion is treated as a resource. The fact that *anger CC'* appears four times means that a character is *four times* as angry at another character. Depending on

how many times the “anger” statements appear in the new state, this anger can rise or fall at the next step in the story. In this case, the emotion is not only treated as a pre-condition, but also as a resource. These resources can also be specified through the addition of a number to the name of the resource.

Ceptre introduces a *stages* feature that allow authors to structure a program through the use of independent components. A stage is a unit of computation that runs to quiescence, meaning that it terminates when no more rules are able to fire. At this point, another stage may commence.

The central motivation behind Ceptre’s design is its ability to use “proofs as traces”, or *computation as proof search* (Hodas and Miller, 1991). Ceptre uses a *sequent calculus*, where a sequent $\Delta \vdash A$, Δ is a state, and A is a goal formula. If a complete proof tree can be formed with that sequent as its root, then the sequent can be said to be *derivable*. Thus Ceptre takes a sequent as input and creates a proof as output, declaring failure if a proof cannot be created. Ceptre looks at the left side of the sequent, using *forward chaining* to choose which rules to try in order to reach the goal formula.

A key feature of Ceptre is its representation of resource management in stories. Rules are able to either produce or consume resources. This has interesting implications for the representation of causal structure in linear logic. For example, if two rule applications consume different sets of resources from the same state, they are occurring concurrently and independently. However, if one rule produces resources that are consumed by another rule, then these rules have a causal, dependent relationship.

2.2.4 Normative Frameworks for Multi-Agent Systems

Social Institutions (or *Normative Frameworks*) govern the behaviour of agents within a multi-agent system. The agents act according to norms described within the institutions that govern their society, which act as a social contract that the agents follow in order to cooperate. We use these institutions to govern the character agents in our story world (described in Chapter 6 on page 117) because they regulate agent behaviour without strict enforcement of their actions. This gives the characters in a story the autonomy to break away from a predefined narrative in certain situations.

Noriega’s Ph.D. thesis (Noriega, 1999) uses a fish market as a metaphor for institutional governance of agents. The thesis explains that Spanish fish markets seem to use a *downward-bidding* auction protocol that emerges from a social institution with refined, socially acknowledged conventions. Noriega explains how agent-mediated institutions may use these social conventions to model such a market, with an agent governor to ensure that the conventions are enforced.

The idea of agent “societies” being governed by social conventions is realised through several different implementations. Artikis et al. (2009) present a framework for the executable specification of open agent societies, where an open society is one with heterogeneous agents, conflicting individual goals and unpredictable behaviour. Emphasis is put on the fact that individuals in such societies can choose to ignore the social norms that govern them, so long

as they are willing to accept the consequences of non-compliance. Their system is specified using two action languages (languages for specifying state transition systems): the $\mathcal{C}+$ language (Giunchiglia et al., 2001, 2004) and the Event Calculus (Kowalski and Sergot, 1986; Shanahan, 1999). The models are executing using the Causal Calculator, a software implementation of $\mathcal{C}+$. The social constraints of their agent society are expressed in terms of:

- The agents’ physical capabilities
- Institutionalised powers
- Permissions, prohibitions and obligations of the agents
- Sanctions and enforcement policies that deal with the consequence of non-compliance with obligations and performance of forbidden actions

Each agent in the society is assigned a social role that determines its permitted and obliged behaviours. The social constraints defined in $\mathcal{C}+$ are described in terms of *fluents* (symbols that characterise a state) and *actions* (symbols that characterise state transitions).

Fornara et al. (2007) propose a model for the definition of “artificial institutions” for the governance of open multi-agent system societies. They use the notion of *social commitment* to define the semantics of an Agent Communication Language (ACL). Commitments, being objective and external to an agent’s internal state, make it possible to verify whether or not an agent is conforming to the rules of its society. The authors use *norms* as event-driven rules that create or cancel sets of social commitments, and are triggered by events that occur in the multi-agent system. These norms are expressed as *obligations* and *prohibitions* on the agents’ actions. The system proposed consists of:

- A set of *entities* that have *natural* or *institutional* attributes, where natural attributes reflect physical properties of the agents’ world (such as the colour of a book), and institutional attributes reflect properties that only exist due to agreement between the agents (such as the price of a book).
- *Instrumental actions*, which are actions that occur in the agents’ physical environment, possibly triggering institutional actions.
- *Institutional actions*, which are events that change institutional attributes, with preconditions and postconditions.
- *Conventions*, which are the norms triggered by the institutional actions in the system.

Cardoso and Oliveira (2007) describe the creation of “virtual organisations”, where agent cooperation is regulated by appropriate norms. In this case, the enforcement of the institutional norms (“Institutional Reality”) is carried out by agents performing roles with certain institutional powers. The implementation of their system (the “Institutional Reality Engine”) uses a rule-based inference engine to determine when norms are violated or fulfilled.

Cliffe et al. (2007) develop *InstAL*, the Institution Action Language, a domain-specific language for the specification of social institutions that govern the behaviour of multi-agent systems. An InstAL institution consists of:

- *Domain fluents*: symbols to track the state of the institution.
- *Institutional Empowerments*: rules that determine which events have the institutional power to occur.
- *Exogeneous (or external) Events*: these resemble the instrumental actions of Fornara et al. (2007)'s system, as they describe the actions that occur in the agents' physical environment.
- *Institutional Events*: these events are the same as Fornara et al. (2007)'s institutional actions, which are events that are triggered by exogeneous events and occur within the institution.
- *Violation Events*: these are events that are triggered when a forbidden action is carried out, or if an obligation is not met, acting as the punishment for non-compliance.
- *Permissions*: the set of actions that agents are allowed to carry out.
- *Obligations*: the set of actions that agents are expected to take before certain deadlines.

Section 4.4.2 on page 76 contains a more detailed description of the InstAL semantics, including an example of how a scene from the “Punch and Judy” story is expressed in terms of tropes, then as an InstAL institution. This example is then expressed in InstAL syntax in Section 6.2.4 on page 129.

2.2.5 Modelling Characters with Emotional Models

In this section, we examine the use of emotional models in intelligent agents, so that we may create more “believable” characters to add to our stories. Part of the motivation for governing our character agents with the kind of social institution described in Section 2.2.4 on page 35 is so that the characters are able to break away from the path of the story in “extreme” circumstances. When agents are created with an emotional model, their emotions would be one way to provide such circumstances, allowing them to deviate from the narrative at times of extreme emotional distress. The reasoning behind this is described further in the introduction to Chapter 6 on page 117.

Usually it would seem odd to want to model emotion as part of a computational process. Emotion is such a seemingly irrational set of behaviours that they are easy to dismiss as ‘human imperfections’. However, as Gratch and Marsella (2004) observe, emotions may have a useful role to play in communication, so long as they are displayed at appropriate times.

For example, anger prepares the human body to fight by increasing the production of adrenaline. Fear similarly triggers the ‘fight or flight’ response, alerting the senses for danger and preparing the body to react.

In order to model human emotions using agents, we must first find a suitable psychological model to use. Marsella et al describe three main types of emotional model:

1. **Discrete** emotional models, which claim that humans have a set of innate, pre-defined emotional states which people may enter and leave.
2. **Dimensional** models of emotion, describing the spectrum of emotions as being points somewhere in continuous space. Implementations typically use two or three dimensions for simplicity.
3. **Appraisal** theories of emotion take an agent's mental processes into account. Their emotional state is derived from whether or not their goals have been achieved, and what effects current events are having on their circumstances, for example.

'Basic' emotions

Ekman first made a case for discrete, biologically-determined emotions, based on evidence from research into facial expressions (Ekman, 1992). He describes emotions as being *basic*, in two senses of the word: *i.* that there are a number of distinct emotions, each with its own different characteristics, and *ii.* that these emotions were developed through evolution for specific functions.

Ekman argues that these evolved emotions share nine characteristics:

1. Distinctive universal signals
2. Presence in other primates
3. Distinctive physiology
4. Distinctive universals in antecedent events
5. Coherence among emotional response
6. Quick onset
7. Brief duration
8. Automatic appraisal
9. Unbidden occurrence

These characteristics are shared by all of the 'basic' emotions as observed in humans and primates.

Discrete models of emotion suggest that there is a neural basis for emotion. For example, Armony et al describe how the amygdala in the brain is responsible for conditioned fear responses, and created a neural network to model it (Armony et al., 1997).

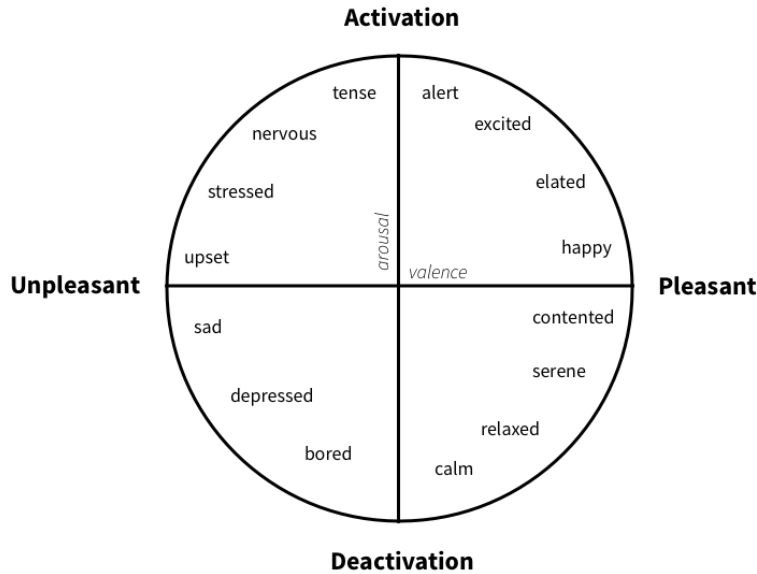


Figure 2-8: Russell’s circumplex model of emotion

Using a discrete model of emotion for agent-based characters would be relatively simple. Each basic emotion could have its own distinct set of behaviours as consequences, and triggering circumstances as preconditions.

However, a more fluid approach could be useful when modelling emotions with agents. It would be impossible to say that an agent is *angry and approaching furious* using a discrete theory of emotion. Nuanced levels of emotion and even combinations of several emotions add an extra level of texture to a character. Dimensional and appraisal theories of emotion address this challenge.

Russell’s circumplex model of emotion

Russell’s circumplex model of emotion is a well-known dimensional model (Russell, 1980). In this case, the dimensional variables are *valence* (how agreeable or otherwise a situation is to an agent) and *arousal* (how excited an agent is).

Russell’s original paper proposes a model similar to that shown in Figure 2-8, where the *x* axis is a person’s valence level and the *y* axis is their arousal level. He argues that the full range of human emotions lie as points along these axes. Eight such examples are shown in Figure 2-8.

This model is very easy to adapt to human-like agents. Ahn et al. (2012) adapt this model by adding a third dimension, dominance, to create conversational agents in a 3D environment. This ‘dominance’ dimension was first proposed in Mehrabian and Russell’s original work (Mehrabian and Russell, 1974), but later removed due to being perceived as the consequences of the *effects* of emotion (Russell, 1980), rather than being a component of emotion itself. Like Ahn et al, we found it useful to add the dominance-submission dimension, and so left it in our emotional model. This is the approach we take in creating my Punch and Judy simulation, and so it is described in more detail in Section 6.2.1 on page 125.

Appraisal theory

Appraisal theories of emotion lend well to simulation with agents, due to their taking a person’s beliefs, desires and intentions into account with respect to external events. Emotions arise when an event occurs and a person internally *appraises* its consequences with respect to their beliefs, desires and intentions. This fits well with the popular BDI architecture for intelligent agents.

Different methods of appraisal may be used in order to produce emotions. Gratch and Marsella (2004) use decision theoretic plans, but other approaches could include reactive plans, Markov-decision processes, or detailed cognitive models.

Though the Punch and Judy simulation described in Section 6.2 on page 125 uses a dimensional model of emotion, an appraisal-based model would be worth investigating due to its tight coupling with belief desire intention psychological models used in agents.

2.2.6 Discussion

Most narrative generation systems make use of formalisms such as Propp (Propp, 1968) in order to generate different parts of a story. Though Propp’s system predates computational narrative generation by several decades, neither narrative nor AI research have produced a formalism for narrative components to replace it entirely. This is likely because Propp’s formalism is “good enough”, and has worked for most researchers that have used it. There is also likely to be a network effect, where Propp has become the formalism that most researchers have heard of, and therefore the one that they end up using.

Though there have been attempts to create more expressive formalisms as described in Section 2.1.3 on page 24, none have been expressive enough to overcome Propp’s “good enough” properties. In order for a story model to significantly improve upon Propp, it should add features that it and other existing formalisms lack. We identify these features as:

- A means of *abstraction*
- Conceptually *simple* enough for non-programmers to grasp
- A library of re-usable *examples* for authors and researchers

A Means of Abstraction Current narrative formalisms lack a means of *abstraction*. Propp, for example, describes events in stories that all occur at the same level of abstraction. This means that one is limited to describing events that occur one after another, or in parallel, but not events that contain sub-events.

For example, suppose we define a “Quest” component of a story, which describes a sequence of events that occur (such as the hero leaving home and then defeating a monster). It is easy to think of other story components that could contain this “Quest” component, such as “Hero’s Journey”, “Rescue the World” or “Rite of Passage” story components. These components could themselves be used as part of other components. This gives us a means of creating abstractable,

reusable story components. Of all the story models described in Section 2.1.3 on page 24, none have any means of abstraction such as this.

The example we just described already hints at the use of story tropes that contain other tropes. This is the means of abstraction and re-use that we use in our system, described in Chapter 3 on page 43.

Conceptually Simple Story authoring tools are user interfaces which are used to write fiction. In the computing world, user interfaces are often simplified through the use of an *analogy*. For example, a *Desktop Environment* is a graphical user interface analogy created by Xerox in the 1970s. Where computer users would previously have had to learn and master a complicated command-line interface, the desktop environment metaphor presented an interface that resembled something that users were already familiar with: the top of a desk, with icons representing files spread out over a “desktop” surface.

When creating a story model, it is easy to fall into the trap of creating something complicated but arbitrary. As with Vonnegut’s “Shapes of Stories” metaphor (described in Section 2.1.1 on page 19, sometimes the simplest explanations are the easiest to grasp and most enduring. We believe that our “trope” analogy for modelling stories (Chapter 3 on page 43) is an elegant, expressive and accessible way of describing parts of stories.

Most story authors are already familiar with the concept of tropes. When questioned about their familiarity with the idea, all the participants at an interactive fiction meetup responded that they were familiar with the “trope” concept (sample size 19, see Section 3.2.2 on page 52). This means that tropes are a suitable analogy for the creation of a new narrative formalism.

A Library of Re-usable Examples Part of the problem of existing formalisms is that story authors that use them need to write their own story components based on a formalism’s description. Though these formalisms are usually described in papers with one or two examples, any story author would have to create their own formal descriptions, even for commonly-occurring story components such as quests or *The Hero’s Journey*. What is needed is a library of pre-existing formal descriptions of story components that authors can easily copy and paste into their stories.

As the concept of *tropes* is one that is already well-known to story authors and consumers of media, it is easy to find many examples of them in reviews of books, films and computer games. In fact, there is a whole website called “TV Tropes”, which is dedicated entirely to the description of tropes that recur throughout different types of media. This website takes the form of a wiki, with a very active community who contribute trope descriptions along with lists of the media they appear in.

For example, a trope called “Karma Houdini” has the following description:

“The character has done a number of things that deserve a karmic comeuppance, most importantly things that caused harm to the innocent. But when the time comes for the hammer to fall, that’s not what happens. At least, not on him. He

doesn't get what he deserves. Instead, he gets away scot-free. And he might even have reversed the Humiliation Conga that was being planned for him.

This is it. This is all there is to the story. The show is over. The book is finished. The author isn't going to write any more. The Word of God has been spoken. The guy has become a Karma Houdini.”

The site lists many examples from film and literature, including:

- **Treasure Island:** Long John Silver escaped scot free with a chest of treasure, and was never caught. Not bad, for a month's murder and betrayal.
- **The Talented Mr Ripley:** Villain Protagonist Tom Ripley killed some people to assume a new identity and enrich himself thoroughly. In the sequels, he killed to protect his new life, and sometimes as favors for others. He never faced justice.

The *TV Tropes* website is a Wikipedia of sorts for tropes. Since there are already so many examples listed on its website, it reduces the work needed to create a library of reusable formal descriptions of tropes.

These three omissions (means of abstraction, conceptual simplicity and a re-usable library of components) from existing narrative systems are addressed by our trope-based approach to interactive narrative generation. Our approach is an interpretation and extension of Propp's system, implemented with story tropes and normative multi-agent system. The theory of tropes is described in Chapter 3 on the following page, with Chapter 4 on page 54 describing the technical implementation of the TropICAL language for story tropes. The StoryBuilder tool, a user interface on top of TropICAL, is described in Chapter 5 on page 101

Chapter 3

Tropes as Story Components

This chapter describes the foundation of our narrative components: story tropes. Though our use of tropes for the description of stories has many advantages, the main benefit is that they offer a simple means to compose and thereby reuse existing story fragments. We describe this in more detail in Section 3.2.1 on page 50.

The literature review in Chapter 2 on page 18 highlights the need for more flexible and intuitive formalisms of narrative. We address this by using tropes to provide an expressive, informal alternative to a strict formalism such as Propp’s “Morphology”. Formalisms require their users to learn their constituent rules in order to be useful. Our trope-based approach aims to allow the user to describe the parts of their story in as close to natural language as possible, while still allowing for their translation to a formal representation. This is implemented through a controlled natural language approach to the specification of Tropes in our TropICAL programming language, described in Chapter 4 on page 54.

Story tropes are a familiar concept to many outside of the academic community, so their omission from the literature in either field of Computer Science or Narratology is worth addressing. This chapter contains a thorough description of what a story trope is, along with several examples.

Tropes are patterns that appear throughout various different media. Once one is familiar with a trope, it becomes easy to identify its use in any story. Take, for example, the *Hero’s Journey* trope first described in Chapter 1 on page 8. It is a template which is repeated so often in many different media, stories and contexts that it is instantly recognised even by those that are completely unfamiliar with the concept of tropes.

In this section we examine the concept of a “trope”, deconstructing examples to demonstrate widely-recognised trope patterns, and exploring tropes that operate at different levels of abstraction within a story. At the end of the section we put forward a formal definition of a trope, and how it fits within the wider context of a story.

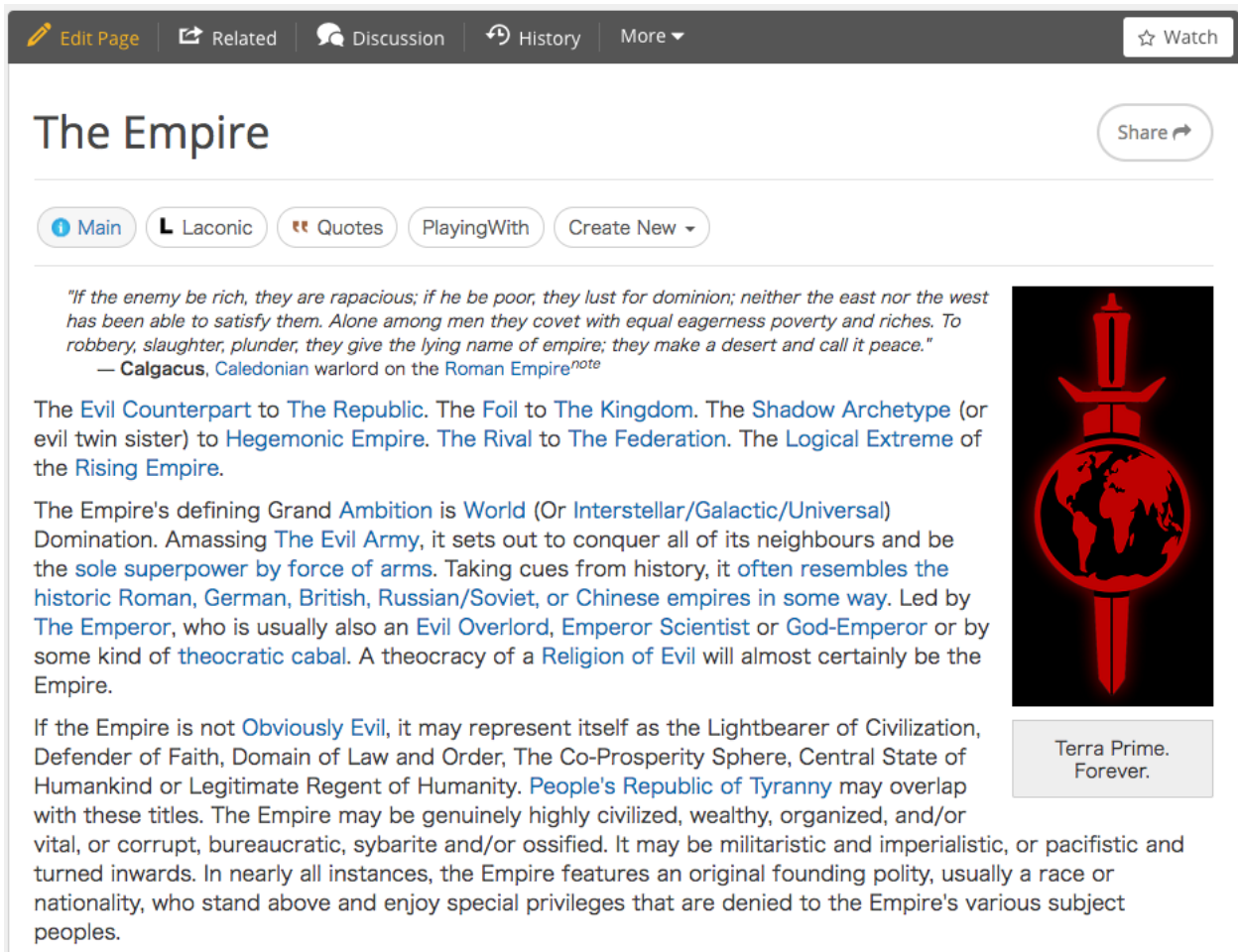


Figure 3-1: A screenshot of the “The Empire” wiki page from TV Tropes

3.1 Tropes: a “Folksonomy” of Story Components

The existence of a website called “TV Tropes” (TV Tropes, 2017a) makes the discovery of example tropes very simple. TV Tropes is a wiki for tropes, containing over 27,000 trope descriptions, along with the media that they appear in. For example, the “The Empire” trope appears in *Star Wars*, Asimov’s *Foundation* trilogy, the *Hunger Games* books and films, and the *Final Fantasy* series of games, and a great many more stories in media.

Figure 3-1 shows a screenshot of the “The Empire” page on the website. It clearly shows the description of the trope at the top of the page, and there are also instances of its use across different media at the bottom.

Tropes can also describe character archetypes. For example, this is how TV Tropes describes *Anti-Hero* characters:

An Archetypal Character who is almost as common in modern fiction as the Ideal Hero, an antihero is a protagonist who has the opposite of most of the traditional attributes of a hero. They may be bewildered, ineffectual, deluded, or merely apathetic. More often an antihero is just an amoral misfit. While heroes are typically conventional, anti-heroes, depending on the circumstances, may be pre-

conventional (in a "good" society), postconventional (if the government is "evil") or even unconventional. Not to be confused with the Villain or the Big Bad, who is the opponent of Heroes (and Anti-Heroes, for that matter).

TV Tropes further clarifies that there are subdivisions of Anti-Hero, depending on just how evil or cynical the character is. Batman, for example, would be a highly cynical Anti-Hero who is nevertheless morally good.

Shakespeare's Macbeth is a character who becomes more and more of an evil Anti-Hero, until he is too morally evil to still be a Hero and instead becomes a Tragic Villain.

Tropes can be very specific, referring to individual lines of dialogue. One example is "We Will Meet Again":

The standard phrase when the villain finds that he has been defeated by the heroes and there is no point in staying around with the immediate Evil Plan foiled.

Tropes can also be very abstract, referring to particular genres, types of story, or events in a story that move the action forward. Other than the previously mentioned "Hero's Journey" and "The Empire" tropes, another example could be the "Hilarity Ensues" trope:

Actions that are dangerous or illegal often lead to injury, arrest, job dismissal, expulsion from school, deportation, or other dire consequences. Thankfully for our fictional friends, both the Rule of Cool and the Rule of Funny keep them safe (the latter more prominently).

Metatropes are tropes about tropes, often intended as a knowing wink to the trope-savvy audience. One such example is "Lampshade Hanging", which TV Tropes describes as:

...the writers' trick of dealing with any element of the story that threatens the audience's Willing Suspension of Disbelief, whether a very implausible plot development, or a particularly blatant use of a trope, by calling attention to it and simply moving on.

In fact, even if an audience is unaware of the concept of tropes, they may be aware of the recurring patterns and themes that tropes describe. This enables genre-savvy writers (especially postmodernists such as Brecht and Vonnegut) to play with the audience's expectations. Ways to do this with tropes could include *inversion* (reversing the trope), *subversion* (making it look like the trope will happen, but then not using it after all), *parody* (using the trope in an over-the-top, exaggerated manner) and *deconstruction* (using the trope in a straightforward manner, but in way which forces the audience to analyse the trope itself).

Take, for example, the well-known "The Butler Did It" trope from murder mystery stories, where the butler of the house is revealed to be the murderer at the end of the story. TV Trope describes ways that an author could "play" with this trope:

- **Subvert** it: The butler is the prime suspect at the beginning, and is later found innocent.

- **Invert** it: The butler is the victim. Or the butler solved the crime. Or every suspect except the butler was part of the crime.
- **Parody** it: Butlers could learn their trade at butler college where they are taught cleaning, cooking, and murdering.
- **Deconstruct** it: The butler is a revolutionary serial killer, who purposely takes jobs as butlers to murder his rich masters. All the unfortunate implications of class warfare that this suggests are brought up and discussed.

Many other examples of using tropes in this way can be found on the “Playing with a Trope” page of the TV Tropes website (TV Tropes, 2017b).

A large and highly active community of users and contributors exists around TV Tropes. In addition to creating content for and curating the content on the website, they also work to create useful ways to visualise the usage of tropes in stories. For example, The Periodic Table of Storytelling (Harris, 2017) is a visualisation of tropes as “elements” in the “molecules” of a story. The table itself (Figure 3-2 on the following page) arranges the tropes into different “groups” according to the part of a story that they operate on. The leftmost groups describe the story as a whole, describing:

Story structure:

- Three Act Structure
- MacGuffin
- Chekov’s Gun

Story Modifiers:

- Darker and Edgier
- Tear Jerker
- Jumping the Shark

Plot Devices:

- Hand Wave
- Techno Babble
- Xanatos Gambit

In the centre of the table are different types of character such as: **Heroes:**

- Anti Hero
- Action Girl
- The Gunslinger

The Periodic Table of Storytelling

by James Harris second edition

clickable version: designthroughstorytelling.net/periodic



Figure 3-2: The “Periodic Table of Storytelling”, original by James Harris (<http://jamesharris.design/periodic/>), poster format by Deviant Art user Dawn Paladin (<http://dawnpaladin.deviantart.com/art/The-Periodic-Table-of-Storytelling-Second-Edition-425816342>)

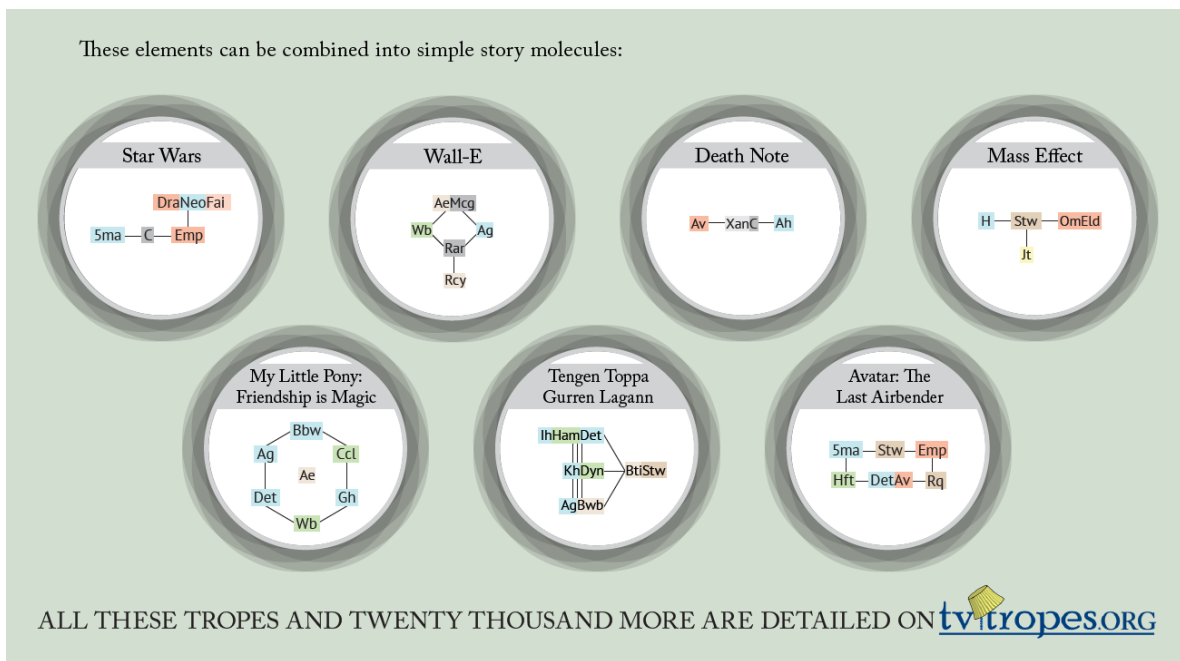


Figure 3-3: Example “Molecules” from the “Periodic Table of Storytelling”, original by James Harris (<http://jamesharris.design/periodic/>), poster format by Deviant Art user Dawn Paladin (<http://dawnpaladin.deviantart.com/art/The-Periodic-Table-of-Storytelling-Second-Edition-425816342>)

Archetypes:

- Mad Scientist
- The Fool
- Loveable Rogue

Villains:

- Evil Twin
- The Empire
- Obstructive Bureaucrat

The right third of the table contains self-referential tropes such as:

Metatropes:

- Lampshade Hanging
- Subverted Trope
- The Fourth Wall

Fandom and Audience Reactions

- Fanon

- Fridge Logic
- Freud Was Right

The story is then visualised as a molecule composed from tropes, linked together as atoms, as shown in the various examples in Figure 3-3 on the previous page.

By showing tropes as “atoms” that can be rearranged and combined into different “molecules”, this visualisation demonstrates our core idea of their use as reusable story components, but the “molecule” metaphor is unsatisfactory for a couple of reasons. Firstly, linking tropes together as atoms in a molecule does not communicate the different levels of abstraction at which tropes operate. Considering that our main purpose for choosing tropes as our method of describing narrative components, this means that the “molecule” metaphor used by the author does not match our intentions. The “Hero’s Journey” trope, for example, would describe the narrative arc as a whole, while the “Comeuppance” trope would describe just a single scene in the story. The metaphor is also not ideal because it presents unrelated tropes together in a story with no indication of which part of a narrative they affect. A “scoundrel sidekick” could be linked together with a “breaking the fourth wall” trope, even though one trope relates to a certain character, and the other may describe a single line of dialogue or action that occurs at a specific point in the story. In this visualisation, a trope that describes something as significant as the structure of the entire story would carry the same weight as a short event that occurs just once in the entire story. The arrangement and linking of the tropes in the example molecules is also quite arbitrary. The examples given on the *periodic table* web site form interesting shapes, but do not follow a consistent logic. In the *Star Wars* molecule, for example, the “Five Man Band”, “Conflict” and “The Empire” tropes are linked in a straight line suggesting a linear sequence, but three further tropes (“The Dragon”, “The Chosen One” and “You Have Failed Me”) are all linked to the “The Empire” trope in the molecule. While “The Dragon” refers to the Death Star in the movie, the “The Chosen One” trope is more closely linked to Luke Skywalker’s Hero role. The “You Have Failed Me” trope refers to a specific scene where villain Darth Vader punishes an under performing henchman with choking. It is not clear why the creator decided to link these specific tropes to the “The Empire” trope. This illustrates that such planar structures are insufficient to capture anything but the simplest stories.

Similarly, in the *GhostBusters* example, the “Five Man Band” and “Mad Scientist” tropes appear together in the same “atom”, which are linked to “Sealed Evil in a Can” and “Hilarity Ensues”. Again, it is not clear why those particular tropes are arranged together into the same atoms, or why they are linked together in this way. The most likely explanation is that this visualisation of the way that tropes link together in a story is not intended as a formal system for trope visualisation, and is merely a proof-of-concept work.

Working from the ideas presented in this visualisation, we develop our concept of tropes as logical, reusable components for the formal description of stories.

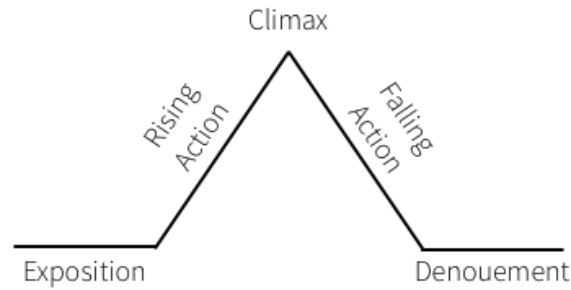


Figure 3-4: Freytag's Pyramid

3.2 Why Use Tropes?

Returning to the shortcomings of existing narrative formalisms we describe in Section 2.2.6 on page 40, we now describe how tropes are suitable for use as a narrative formalism that is able to overcome these limitations.

3.2.1 A Means of Abstraction

Most tropes form a hierarchy, with parent tropes such as the *Quest* containing child tropes such as *Redemption Quest*, *Sidequest* or a *Quest for Identity*. This hierarchy arises from story authors adding nuance to existing narrative archetypes through adaptation. This is often achieved by mixing different tropes together. For example, a “child” trope of *The Hero's Journey* could be *The Anti-Hero's Journey*, combining the *Anti-Hero* trope with the classic *Hero's Journey* trope. These child-tropes inherit some of the characteristics of their parents, but add subtle or major changes. For example, a *Quest for Identity* follows the *Quest* format, but is constrained so that the item that is the subject of the quest is the hero's own identity. This resembles the object-oriented programming mechanism of delegation, so one can imagine using such a process to avoid duplication of effort when authoring new tropes by only expressing how a trope differs from its parent.

Another method of abstraction is the embedding of sub-tropes as parts of parent tropes. This is how tropes can be defined and then re-used inside stories: by allowing any trope to be a component that can be dropped inside any other trope. The example we described in Section 2.2.6 on page 40 describes how the “Quest” trope could form one part of a larger trope such as the “Hero's Journey”. Another example of this would be the **Three Act Structure** (also known as Freytag's Pyramid) shown in Figure 3-4, which describes the shape of a story in terms of rising and falling levels of drama. This could be split into five (or perhaps more) sub-tropes:

1. **Exposition:** The setting of the scene, providing any background information that is relevant to the story.
2. **Rising Action:** A series of event drive the story forward, each increasing in dramatic intensity.

3. **Climax:** The turning point of the story. Some fateful event occurs as a result of the rising action, which could be a battle between the hero and the villain, for example.
4. **Falling Action:** The consequences of the climax play out, and the story shows how the characters are affected.
5. **Denouement:** This is the final resolution, where all the “loose ends” of the story are tied up.

This means that if we already have trope definitions for the “Exposition”, “Rising Action”, “Climax”, “Falling Action” and “Denouement” parts of a story, and want to create a “Three Act Structure” trope, we can simply express it in the following way:

1. The “Exposition” trope happens
2. Then the “Rising Action” trope happens
3. Then the “Climax” trope happens
4. Then the “Falling Action” trope happens
5. Then the “Denouement” trope happens

Returning to the concept of “story structure” described in Section 2.1.1 on page 19, we can use the “subtropes” we just identified to define other “story shapes” as described by Vonnegut (2009). For example, the “man in hole” story shape described in Section 2.1.1 on page 19 could simply be described as a rearrangement of the three-act structure:

1. The “Exposition” trope happens
2. Then the “Falling Action” trope happens
3. Then the “Nadir” trope happens
4. Then the “Rising Action” trope happens
5. Then the “Climax” trope happens
6. Then the “Denouement” trope happens

Note that we added an additional trope, the “Nadir” trope to describe the lowest point in the story. Otherwise, the rest of the story “shape” is easily described in terms of tropes that we have defined already.

The fact that we can describe an abstract trope such as “man in hole” in terms of more specific tropes addresses one of the shortcomings of the Periodic Table’s molecule model of tropes from Figure 3-3 on page 48: that all tropes have equal weight in a story. Though the above example only describes how these sub-tropes may be sequenced within their parent

trope, they may also be arranged as branching alternatives, so that either *Trope A* **or** *Trope B* occur. This is discussed further in Section 4.3.4 on page 66.

This re-use of existing trope definitions saves us the time and effort of the wasteful duplication of the steps already described within them. This is why abstraction is such a powerful and useful concept: it allows us to break down complicated stories into a series of smaller sub-stories, rather than having to describe the whole thing in one go.

3.2.2 Conceptually Simple

Most story authors are already familiar with the concept of tropes. In order to evaluate the suitability of their use for the description of narrative components, we presented a preliminary version of our trope-based TropICAL programming language (described in Chapter 4 on page 54) for story authoring to the Oxford and London Interactive Fiction meetup group.

After a brief presentation on the concept of tropes and how we intend to use them to create an authoring system for interactive narrative, participants were given a questionnaire with the purpose of discovering their familiarity with tropes, as well as finding out how suitable they thought tropes would be as a new kind of formalism for narrative components.

There were 18 responses to the questionnaire. The questions and responses are listed in Appendix A on page 192. The fact that all of the interactive fiction authors and games developers were already familiar with the concept of tropes demonstrates that they are conceptually simple enough for non-programmers to understand. Not only that, but most of them were already familiar with the TV Tropes website. This meets our requirement that authors should be able to create story components without having to learn an unfamiliar set of concepts such as those used by Propp (1968) or Lehnert (1981), which can only be discovered through reading academic books and papers.

3.2.3 A Library of Re-usable Examples

One of the major strengths of Propp’s system (Propp, 1968) is that the *morphology* is not only a theory: it is also a library of 31 story functions that can be put together by story authors to create a narrative. The authors need not create their own story functions, they can simply use the ones that Propp has already created for them.

Though the TV Tropes website does not contain formal definitions of tropes, it can be used as a resource on which to base the authoring of tropes in our system. In addition, it grants authors the flexibility to create their own tropes which are not already listed on the TV Tropes website.

TropICAL, our domain-specific programming language for trope-oriented story authoring, aims to simplify the authoring of story tropes for non-programmer users. In the same vein as Inform 7 (Reed, 2010), our language uses a constrained natural language syntax. Further details are described in Chapter 4 on page 54, where we describe the language in detail. In the same manner as a wiki, once a number of authors have contributed tropes, it could offer the capacity to be a library of reusable tropes for future authors to use in their stories.

To summarise: tropes have the potential to be a useful model to use for story components, as they fulfil the criteria we lay out in Section 2.2.6 on page 40:

- They provide a *means of abstraction* through subtropes as well as parent and child tropes.
- They are *conceptually simple* for authors to learn, given that all the authors in our preliminary study with the London Interactive Fiction Meetup group are already familiar with them.
- The TV Tropes website is a useful resource that can be used to aid the creation of a *library of re-usable examples*.

This chapter has described the concept of tropes, breaking down several examples and describing them as normative institutions. We then returned to our *Punch and Judy* example to describe a scene in terms of tropes rather than Propp functions.

The next chapter describes the design and implementation of *TropICAL*, our domain-specific language for describing tropes. TropICAL compiles controlled natural-language trope descriptions to formal representations in terms of InstAL institutions. After discussing requirements and design rationale, the chapter lists annotated snippets of TropICAL code for example tropes and their corresponding translations to *InstAL*.

Chapter 4

TropICAL: A Language for Story

Tropes

This chapter describes the design and implementation of TropICAL (the TROPe Interactive Chronical Action Language), our Domain Specific Language for story tropes. The purpose of this language is to allow for the creation of interactive narratives through the description of tropes in a constrained natural language. The tropes created through the language are designed to be reusable components that can go into a “library”, from which story authors can choose the tropes they wish for the particular story that they are creating.

The motivation for the creation of TropICAL is the lack of any methods of creating agent-based interactive narrative that are suitable for non-programmers to use. Inform 7 (Reed, 2010) enables non-programmers to create interactive fiction, but not in a way that can be integrated into a multi-agent system. The systems described in the literature review in Chapter 2 on page 18, such as those using the Mimesis architecture (Young et al., 2004), a drama manager and planner such as in Façade’s system (Mateas and Stern, 2003), or linear logic as in Ceptre’s system (Martens, 2015), all require the story author to be familiar with planner-based systems or the description of formal logics. The purpose of TropICAL is to greatly reduce the barrier to the creation of interactive stories with intelligent agents by allowing authors to describe the components of their story using constrained natural language. These story components integrate with a multi-agent system by describing the constraints on the actions of each character agent in the story, telling each agent what they are permitted and obliged to do at any point in the narrative.

Many authors using our system should not even need to write their own tropes using TropICAL, and will instead simply select the tropes that they need for their story from a pre-created “library” of tropes. This process is facilitated through the “StoryBuilder” user interface described in Chapter 5 on page 101

Our TropICAL language design is as follows:

- It is based around the idea of using *story tropes* as reusable components that can be combined to create stories

- It uses *controlled natural language* to describe these tropes
- The controlled natural language is then parsed and compiled to a data structure that describes the formal features of each trope
- The intermediate data structure is then used to generate InstAL code
- The InstAL code is compiled to AnsProlog, an Answer Set Programming language
- The AnsProlog code is then run through *clingo* (Gebser et al., 2011), an answer set solver, to generate all the possible sequences of events that can occur in the story.

This chapter begins by describing TropICAL’s constrained natural language syntax in Section 4.1. Section 4.2 on page 58 explores the use cases for our language, deriving a set of requirements from them. Based upon these requirements, the features and design of the language are described in Section 4.3 on page 62. The translation of TropICAL to InstAL code is demonstrated in Section 4.4 on page 71, with samples of generated InstAL institutions, Answer Set generation (Section 4.5 on page 94), constraints (Section 4.6 on page 94), and output traces (Section 4.7 on page 95). Finally, we show its extension for the description of legal policies (Section 9.1.4 on page 188). The chapter concludes with a summary in Section 4.8 on page 99, describing how some requirements from Section 4.2.2 on page 62 were met, while others are addressed through the *StoryBuilder* tool described in Chapter 5 on page 101.

4.1 Controlled Natural Language Syntax

TropICAL uses a *Controlled Natural Language* syntax (also referred to as *Constrained Natural Language*), meaning that it superficially resembles natural English, but is only a subset of the full language.

There are two types of Controlled Natural Language (CNL). The first type are *naturalistic* languages such as ASD Simplified Technical English (ASD-STE100, 2007), designed to be used in documentation for technical industries such as aerospace or defense, or Ogden Basic English (Ogden, 1944), a simplified language for teaching English as a second language. This type of Controlled Natural Language merely describes a subset of the parent language. The other type of CNL are *formalistic*, with a formal syntax and semantics, which can be mapped to rules in other formal languages such as first-order logic. Attempto Controlled English (Fuchs and Schwitter, 1996) is an example of this formal type of CNL, and the one which forms the basis of our syntax for TropICAL.

4.1.1 Attempto Controlled English

Attempto Controlled English (ACE) is a controlled natural language that is also a formal language, and was created by the University of Zurich in 1995. It is still under development, with version 6.7 of the language announced in 2013. ACE has been used in a wide variety of fields, such as software specifications, ontologies, medical documentation and planning.

ACE’s vocabulary has three components:

- predefined function words
- predefined fixed phrases (*there is, it is false that, ...*)
- content words (nouns, proper nouns, verbs, adjectives, adverbs)

The Attempto Parsing Engine (APE) has a lexicon of content words, and users can define their own content words. User-defined content words take precedence over the built-in lexicon.

ACE’s syntax is expressed as a set of construction rules (admissible ACE sentence structures), with syntactically correct sentences described as a set of interpretation rules, which contain the actual semantics of the sentences.

The simplest ACE sentences follow a *subject + verb + complements + adjuncts* structure:

```
1 A dragon sleeps.
2 The sky is blue.
3 A princess owns a castle.
4 A king gives a sword to a knight.
```

In this structure, every sentence must have a subject and a verb. Sentences that do not contain a verb are expressed with the *there is* structure:

```
1 There is a kettle.
2 There are 3 balls.
```

Details can be added to these sentences through the use of adjectives, and the sentences can be joined together with the *and* and *then* keywords.

Other ways of modifying the sentences include *negation*, adding *quantifiers*, or making the sentences *interrogative* or *imperative*. We do not go into further details in this thesis, as we are using ACE simply as an inspiration for our syntax, rather than following its design entirely. For more details, see the “ACE in a Nutshell” page¹ of its documentation for an overview of the language (Fuchs, 2017).

4.1.2 Inform 7

Besides ACE, the other main inspiration for TropICAL’s syntax is the *Inform 7* language.

Inform is a programming language for the creation of Interactive Fiction (also called *text adventure games*). All versions of Inform generate *Z-code*, which is interpreted by the *Z-machine* virtual machine for interactive fiction.

The syntax of the language has changed multiple times since its creation in 1993. The version of the language that we are most interested in is Inform 7 (Reed, 2010), which is its most recent incarnation. Prior to Inform 7, the language used traditional programming models such as procedural and object-oriented paradigms. With version 7, however, its creator Graham Nelson redesigned its syntax completely to allow authors to create their stories using controlled natural language, so that the experience of story creation became closer to that of writing a book.

¹Located at the following URL: http://attempto.ifi.uzh.ch/site/docs/ace_nutshell.html, accessed 20170826.

The simplest possible game authored with Inform 7 would be:

```
1 "Hello World" by "I.F. Author"  
2  
3 The world is a room.  
4  
5 When play begins, say "Hello, world."
```

The above code simply describes the name of the game (“Hello World”), declares that there is one room in the game, and sets the game to say “Hello World” to the player when the game starts.

An extended example, Will Crowther’s cave exploration simulation, would be described like this:

```
1 "Cave Entrance"  
2  
3 The Cobble Crawl is a room. "You are crawling over cobbles in a low passage.  
   There is a dim light at the east end of the passage."  
4  
5 A wicker cage is here. "There is a small wicker cage discarded nearby."  
6  
7 The Debris Room is west of the Crawl. "You are in a debris room filled with  
   stuff washed in from the surface. A low wide passage with cobbles becomes  
   plugged with mud and debris here, but an awkward canyon leads upward and  
   west. A note on the wall says, 'Magic word XYZZY'. "  
8  
9 The black rod is here. "A three foot black rod with a rusty star on one end  
   lies nearby."  
10  
11 Above the Debris Room is the Sloping E/W Canyon. West of the Canyon is the  
   Orange River Chamber.
```

The “Cave Entrance” example above is just one scene from the game. The code begins with a declaration of the “Cobble Crawl” room, followed by the description that appears when the player enters it. The next line declares that a wicker cage is in the room, which is an object that the player can interact with. The sentence after the declaration is the description of the item that appears when the player looks around the room (“There is a small wicker cage discarded nearby.”). The “Debris Room” follows the same format described above, with a “black rod” item inside the room. The final line of code simply describes how the rooms are arranged in the game, using words like “above” and “west of”.

Although Inform 7 is not based directly on ACE, we can see many of ACE’s features in it. For example, it uses the same syntax for simple statements such as *X is a Y*. Most statements in the listing above are pairs of sentences, with the first in each pair declaring the existence of an object or room, and the second being an uninterpreted string that is printed out as the description of the defined object.

Before version 7, Inform was already widely used as a language for the creation of Interactive Fiction. Given its continued popularity after the switch to a controlled natural language

syntax², it appears that the new programming paradigm is successful.

The wide adoption of Inform 7 by story authors with no other programming experience motivates our decision to create a controlled natural language syntax for TropICAL, our own trope-based programming language. The intended audience is the same, so an additional benefit would be that story authors who can code with Inform 7 would also be able to use TropICAL.

4.2 Requirements

The TropICAL language is designed to be used by authors of interactive narratives, enabling them to easily describe the “rules” of a story through tropes. It is designed to be used in interactive storytelling systems where intelligent software agents are acting out the roles of the characters in the story. These agents, through an architecture such as BDI (Belief, Desire, Intention), have their own goals and plans for how to achieve those goals. TropICAL does not assist in the authoring of these characters and their plans or lines of dialogue. Instead, its purpose is to guide the actions of these “character” agents so that certain behaviours are suggested by the rules of the author’s story. TropICAL is designed to be used with a system where the character agents have already been authored. It guides them by allowing the characters to know what they are permitted and obliged to do according to the story. For a description of how this is implemented in practice, see Section 6.2 on page 125 of Chapter 6 on page 117.

4.2.1 Use Cases

In order to design a set of tools that is suited to interactive story authors, we derive our set of requirements from specific use cases. These use cases come from the user stories of three hypothetical users: Alice, Bob and Charlie. In each case, we tell the user’s story, highlighting their different levels of technical experience, along with their various needs for and ways of using tools for interactive story creation.

Alice

Our first user story focuses on *Alice*, a user with some programming knowledge and specialist expertise in the field of intelligent agents:

Alice is an interactive narrative researcher using the Jason intelligent agent framework to create the characters of a story. The story is a classic detective noir story, with a private investigator as the main character and several suspects, one of which is the killer. She has coded their behaviour and dialogue as a set of plans, which each agent will follow according to their beliefs, desires and intentions as they “act out” the story.

²The Interactive Fiction Database at <http://ifdb.tads.org> (accessed 20170826) lists 1078 games created with Inform 6, and 914 created with Inform 7. Inform 6 has been in use since 1996, Inform 7 since 2006.

Alice sets the simulation running, but sees that although the agents are interacting with each other as intended, there is no structure or sense of drama to the story. It merely seems to be one event happening after another.

She decides to use the TropICAL framework to guide the actions of the character agents. The tropes that she chooses to describe her story are the “Three Act Structure”, “Chekov’s Gun”, “Murder Mystery” and “Rooftop Chase” tropes. These tropes are used to not only give the story structure, but also to encourage the characters in it to behave in a way that makes the simulation “feel” like a murder-mystery genre novel.

When using TropICAL, she selects the “Three Act Structure” and “Chekov’s Gun” tropes from a pre-written library of tropes. However, she cannot find tropes that describe what a “Murder Mystery” story is, or what should happen in a “Rooftop Chase” scene. She creates these by writing her own tropes in TropICAL:

```
1  ``Murder Mystery'' is a trope where:
2      The Detective is a role
3      The Killer is a role
4      The Victim is a role
5      The Killer kills the Victim
6      The Detective investigates the Killer
7      Then the Detective catches the Killer
8      Or the ``Rooftop Chase'' happens
9
10 ``Rooftop Chase'' is a trope where:
11     The Detective is a role
12     The Fugitive is a role
13     The Rooftop is a place
14     The Detective goes to the Rooftop
15     The Fugitive goes to the Rooftop
16     The Detective chases the Fugitive
17     Then the Detective catches the Fugitive
18     Or the Fugitive escapes
```

From Alice’s story, we can derive the following requirements:

- Alice R1. She needs a system for describing stories that can be used with a multi-agent system framework such as Jason
- Alice R2. The system needs to direct the behaviour of the agents so that their actions fit within the description of a story
- Alice R3. She needs to be able to write and re-use story components that fit the theme of her story (in this case, “film noir”).
- Alice R4. These story components must be able to express both sequences of events and diverging (“branching”) events

Alice R5. She wants to be able to select existing story components from a library or database

Alice R6. She also wants the ability to create her own story components and save them to the database

Bob

Our second hypothetical user, *Bob*, has no technical expertise at all:

Bob is a traditional story author interested in writing interactive narratives with branching plot lines. He has written a few stories in the style of the “Choose Your Own Adventure” books, using the traditional method of using paper and pen to write down the stories. He implements the story branches by labelling each scene with a name such as “The Hero Sets Off on a Journey”, or “The Villain Deceives a Victim”, and referring to them by name whenever a branching point occurs. For example, when the Hero of a story is tasked with a quest to complete, the decision could be to go to the “Leave Home and Go on a Journey” scene, or the “Stay at Home and Avoid Adventure” scene, which would be the names of scenes leading either to further choices, or the end of the story.

While he is writing the story, he finds it difficult to keep a mental model of the whole plot, with all its branches, in his head. As he creates new choices and branching points in the story, he finds himself having to pin up notes on a board, and connect them with string. Frustrated, he searches for software to help him with the process of authoring non-linear stories, eventually finding a piece of software called “Twine”. Twine’s user interface uses the analogy of notes on a pinboard connected with string, the exact process he is used to. He quickly inputs the story he is working on into Twine, and it produces an HTML website with hyperlinks to pages representing the scenes of his story. Next, Bob wants to get more ambitious by abstracting away different parts of his story and re-using them in different places. He wants his story to be heavily quest-driven, with the player being given many different quests and missions over the course of the game. He decides that all of these quests follow a common pattern:

```
1 The Hero is at Home
2 Then the Hero meets the Dispatcher
3 Then the Hero receives a Quest
4 Then the Hero completes the Quest
5 Or the Hero ignores the Quest
```

In this case, the actual content of the “Quest” would change for each individual quest, but the way the player receives and embarks on the Quest remains the same. Bob wants to write this combination of nodes and branches once, and then re-use it as a single node labelled “Quest”. Unfortunately, he finds no way to do this using Twine, and he is forced to copy and paste the existing nodes every time he wants to put a Quest into his story.

Bob shares some requirements with Alice:

Bob R1. He wants to be able to create story components with sequences of events and branching events

Bob R2. He wants to be able to save and re-use the components (to a database, for example)

Bob R3. He also wants the ability to create his own story components to add to the database

However, Bob also has some different requirements from Alice:

Bob R4. He needs to embed tropes inside other tropes (as “sub-tropes”)

Bob R5. He needs to be able to visualise all of the story branches that result when one or more tropes are combined together

Charlie

Our final user, *Charlie*, has technical expertise, and is also familiar with existing research techniques for interactive narrative generation:

Charlie is an experienced Interactive Narrative author that wants to automatically generate many of the small narrative details. By using intelligent agents to simulate the characters in the story, she hopes that their intelligence and autonomy will result in some surprising narrative arcs.

First, she authors the character agents using the Jason library for intelligent agents. To do this, she must give them a library of plans to execute in order to carry out certain goals. At any moment in the story, each agent has a set of *beliefs* about its environment, plans that they *desire* to carry out, and plans that they *intend* to execute.

She sets the simulation running, but finds that the characters actions do not follow the structure of a story. She starts out by directing their actions using a planner-based system, but finds that the resulting stories don’t make use of the agents’ intelligence. The planner is simply telling the agents what to do, based on its goals. She thinks that this is fine most of the time, but nonetheless desires to occasionally see some unexpected (but plausible) actions from the characters. For example, if a character is close to finishing a plan to achieve a goal, she wants them to choose to override what they are told to do by the director planner, and accept some penalty that comes with the violation of the story.

She attempts to write this functionality into the agents’ decision-making process, but this turns out to be too time-consuming, and she soon gives up. If only there were a better way.

Charlie’s only unique requirement is:

Charlie R1. A way to describe the story such that the agents are told what to do next, but are able to break away from these commands when necessary.

4.2.2 Requirement Specification

Based on the use cases above (Section 4.2.1 on page 58), we can create the following requirements:

- R1. The software must be able to integrate into a multi-agent framework such as Jason (Alice R1)
- R2. The system must direct the behaviour of agents to fit a story description (Alice R2)
- R3. The user must be able to write and re-use existing story components (from a library) (Alice R3, R5 and R6, Bob R2)
- R4. The components must be able to express sequences of events (Alice R4, Bob R1)
- R5. The components must be able to express branching (diverging) events (Alice R4, Bob R1)
- R6. The user must have the ability to nest existing components inside new components (Bob R4)
- R7. The user must be able to visualise the branches of the story that result in the addition or modification of components to the story. (Bob R5)
- R8. The story must tell the character agents what to do, but they should be free to break away from it in extreme circumstances, to add a degree of unpredictability (Charlie R1)

The next section describes the design of the TropICAL language, relating each design decision back to these requirements derived from the use cases.

4.3 Language Design and Features

In this section, we describe the main features of the TropICAL programming language:

- Entity Declarations (Section 4.3.1 on the next page)
- Sequences of Events (Section 4.3.2 on page 64)
- Obligations (Section 4.3.3 on page 65)
- Branching Events (Section 4.3.4 on page 66)
- Subtropes (Section 4.3.5 on page 68)

This section begins by showing how the entities (roles, objects and places) of a trope are defined by the author (Section 4.3.1 on the following page), then follows this with a description of how these entities' actions are composed together to form sequences of events (Section 4.3.2 on page 64). Normally, these events give the character entities *permission* to perform certain actions, so Section 4.3.3 on page 65 shows how an author may describe events that *must*

happen in the story through obliging characters to carry out specific actions. In cases where multiple story paths may be taken, TropICAL allows for the description of alternative narrative branches, the syntax for which is described in Section 4.3.4 on page 66. Finally, the means for creating trope abstractions by embedding existing tropes into new ones is demonstrated in Section 4.3.5 on page 68.

4.3.1 Entity Declarations

A trope may contain three types of entities that take part in a trope:

- Roles (characters acted out by the agents)
- Objects (items that are used or manipulated by the characters)
- Places (locations that the characters visit)

Before an author can use any characters, objects or places in their tropes, they must first declare them at the top of the description file. For example, to declare that the *Hero* and *Evil Villain* are character roles, one would write:

Listing 4.1: Role declarations

```
1 The Hero is a role
2 Evil Villain is a role
```

In TropICAL, the “The” at the beginning of role, object or place names is optional. If the name of an entity begins with “The”, the parser omits it from the name. Also, entity names can consist of multiple words, as is the case for the *Evil Villain* character. However, each word in a name must always start with a capital letter.

Similarly, to declare that our trope contains *Sword* and *Magic Potion* objects, the author would write:

Listing 4.2: Object declarations

```
1 The Sword is an object
2 The Magic Potion is an object
```

Finally, *Home* and *The Land of Adventure* locations in the trope are declared in the same way:

Listing 4.3: Place declarations

```
1 Home is a place
2 The Land of Adventure is a place
```

Once the author has declared the entities of their trope at the top of the trope definition, they are able to use them inside the events of the trope itself.

Entity Instances

In cases where multiple characters fulfil the same role (a story could have two hero characters, for example), or there are multiple objects or locations of the same type, these entity instances

can be declared separately. An example with multiple Heroes, Swords and Evil Lairs would be:

Listing 4.4: Entity instances

```
1 Harry Potter is a Hero
2 Luke Skywalker is a Hero
3 Excalibur is a Sword
4 Oathbreaker is a Sword
5 The Volcano is an Evil Lair
6 The Secret Cave is an Evil Lair
7 The Underground Bunker is an Evil Lair
```

If these separate instances are not declared, then each entity has just one instance with the same name as its type (a hero called “Hero”, a sword called “Sword”, an evil lair called “Evil Lair”, for example).

4.3.2 Sequences of Events

TropICAL is an event-based language: it describes events that happen as part of a story, where the events involve actions taken by roles (the agents in a multi-agent system) interacting with other roles as well as objects and places. The simplest possible statement in the language, given the declarations above, would be to write:

Listing 4.5: An event

```
1 The Hero goes to the Land of Adventure
```

This specifies that the first event that occurs in the trope is that the *Hero* character goes to a place called *The Land of Adventure*. It is important to note that this means the event *can* happen, but no that it *must*. We do this to fulfil requirement 8, so that we *guide* the actions of character agents to fit our story, rather than regimenting their behaviour. Section 4.3.2 describes how this is implemented when compiled. It is possible to tell the agents what to do in a stronger fashion using *obligations*, which we describe in Section 4.3.3 on the following page.

Typically, tropes tend to consist of a sequence of several events, as is stated in requirement R4. So we can extend this trope further to show how TropICAL expresses event sequences in order to this requirement:

Listing 4.6: Two consecutive events

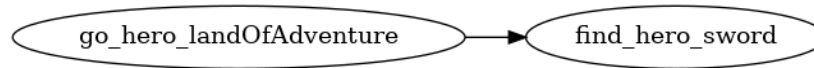
```
1 The Hero goes to the Land of Adventure
2 Then the Hero finds the Sword
```

This series of events can be visualised graphically (using a prefix notation for the verbs and objects) thus:

The *Then* keyword denotes that the event is the next stage in a sequence of events. However, this keyword is syntactic sugar, and can be omitted for the same effect:

Listing 4.7: “The” instead of “Then”

Figure 4-1: Two consecutive events

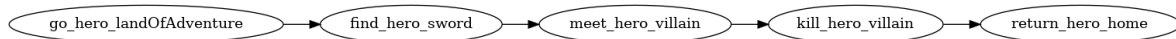


```
1 The Hero goes to the Land of Adventure
2 The Hero finds the Sword
```

Events can be strung together in a sequence that is as long as the author needs:

Listing 4.8: A sequence of events

```
1 The Hero goes to the Land of Adventure
2 Then the Hero finds the Sword
3 Then the Hero meets the Villain
4 Then the Hero kills the Villain
5 Then the Hero returns Home
```



The author can use any verb they wish to describe an event. The verbs are stemmed using *WordNet* (Miller, 1995) so that, for example, *goes* becomes *go* and *wanted* becomes *want* when the tropes are compiled. The reason for this is consistency: it allows an author to re-use verbs in any form throughout the trope. For example, they can write “The Hero goes Home”, “The Hero went Home” or “The Hero may go Home”, and the compiled code will be the same.

4.3.3 Obligations

All of the code examples up to this point express events that *may* occur in the story. When compiled to InstAL, and expressed in terms of norms, they describe the *permitted* behaviour of the agents in a story. This way, the character agents are aware of the actions available to them that follow the path of the author’s narrative. However, there are other actions that an author may want to strongly encourage an agent to take, so that they can direct their actions with more control (as per requirement R2). The syntax for this type of action is expressed with the *must* keyword, and is shown in listing Listing 4.9. In this case, the first action (*The Hero goes Home*) is expressed as a permission, but the second action (*Then the Hero must go to the Land of Adventure*) is an obligation.

Listing 4.9: An obliged event within a trope

```
1 The Hero goes Home
2 Then the Hero must go to the Land of Adventure
```

Deadline Events and Consequences

Obligations may have optional deadline events and consequences. The consequence event is triggered if the obligation has not been fulfilled before the deadline event has occurred. Listing 4.10 shows an example trope where the Hero is obliged to go to the Land of Adventure before the Villain character kills the Mentor character. If the Villain kills the Mentor and the Hero has not yet gone to the Land of Adventure, then a possibility opens up for the Villain to kill the Hero in the story.

Listing 4.10: An obliged event with a deadline and a consequence

```
1 The Hero must go to the Land of Adventure before the Villain kills the Mentor
2   Otherwise, the Villain kills the Hero
```

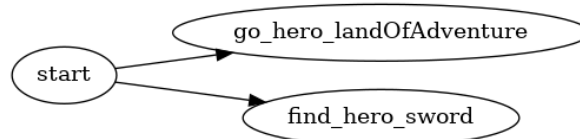
This mechanism could be used as a method of creating branches in the story. If the specified event happens before the deadline event, the path of the trope goes one way. If it doesn't, the story follows the path of the consequence event. 4.4.10 on page 90 explains in detail how obligations, deadlines and consequences are compiled to InstAL code.

4.3.4 Branching Events

As requirement R5 states, a trope author may want to describe alternative possibilities in their story, where either the player or a character in the story may choose from one of multiple actions to take. This is implemented using the *Or* keyword in TropICAL, along with a single indentation level of two spaces to indicate its dependency on the preceding statement:

Listing 4.11: Two branches

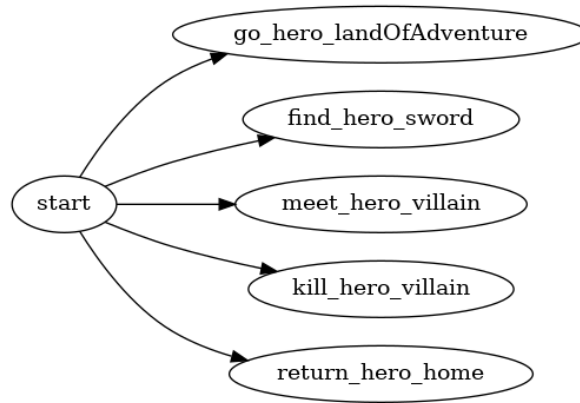
```
1 The Hero goes to the Land of Adventure
2   Or Hero finds the Sword
```



The above code describes two alternative events: either the Hero can go to the Land of Adventure, or the Hero can find the Sword. Multiple events can be chained together on the same level of indentation to create multiple possible alternatives:

Listing 4.12: Five branches

```
1 The Hero goes to the Land of Adventure
2   Or the Hero finds the Sword
3   Or the Hero meets the Villain
4   Or the Hero kills the Villain
5   Or the Hero returns Home
```

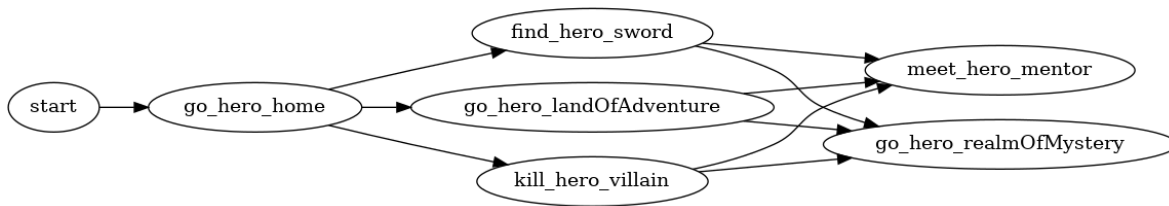


In the above example, the trope begins with five different possible alternative events, representing five paths through the story. These branching events can be combined with the event sequences described previously to create more complex tropes:

Listing 4.13: A combination of branches and sequences

```

1 The Hero goes Home
2 Then the Hero finds a Sword
3   Or the Hero goes to the Land of Adventure
4   Or the Hero kills the Villain
5 Then the Hero meets the Mentor
6   Or the Hero goes to the Realm of Mystery
  
```



In the previous example, only one event occurs in each branching story path before merging back into the main course of the story. In order to extend each story path further, we use a further level of indentation.

Listing 4.14: Extending branches

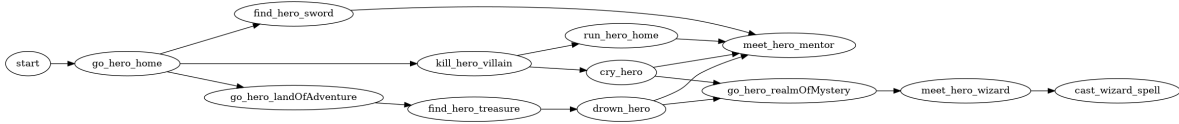
```

1 The Hero goes Home
2 Then the Hero finds a Sword
3   Or the Hero goes to the Land of Adventure
4     Then the Hero finds a Treasure
5     Then the Hero drowns
6   Or the Hero kills the Villain
7     Then the Hero runs Home
8     Or the Hero cries
9 Then the Hero meets the Mentor
10  Or the Hero goes to the Realm of Mystery
  
```

```

11   Then the Hero meets the Wizard
12   Then the Wizard casts a Spell

```



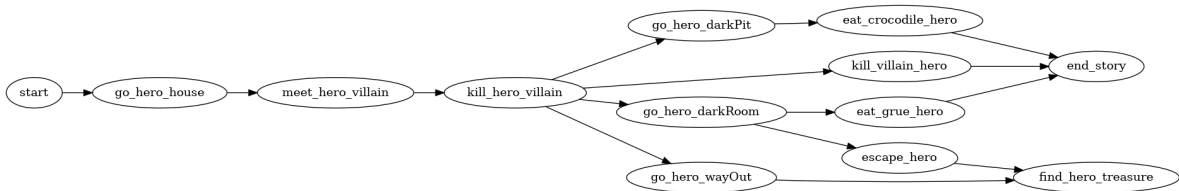
It should be noted that all branches will merge back into the “main” story path (the bottom level of indentation) once they have completed. To prevent this from happening, and terminate the story, the author may write *“Then the Story ends”*. In this example, the story will always end unless the Hero goes to the Way Out or escapes the Dark Room:

Listing 4.15: Terminating branches

```

1 The Hero goes to the House
2 Then the Hero meets the Villain
3 Then the Hero kills the Villain
4   Or the Villain kills the Hero
5     Then the Story ends
6 Then the Hero goes to the Way Out
7   Or the Hero goes to the Dark Pit
8     Then the Crocodile eats the Hero
9     Then the Story ends
10  Or the Hero goes to the Dark Room
11    Then the Hero escapes
12    Or the Grue eats the Hero
13    Then the Story ends
14 Then the Hero finds the Treasure

```



Using this technique, branches can be extended indefinitely through increasing levels of indentation. Deeply indented tropes are often undesirable, however, and is usually a symptom that a trope needs to be subdivided. We can achieve this by embedding subtropes inside of other tropes.

4.3.5 Subtropes

Requirement R6 states that we need a mechanism for the nesting of tropes, so that we may create new abstractions by combining existing tropes to create new ones. We do this through

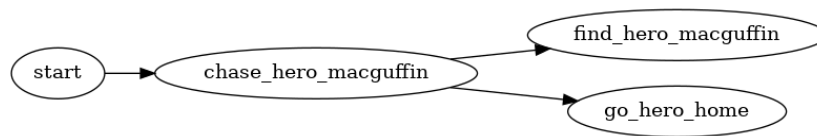
Subtropes, which are simply previously-written tropes that are embedded inside a new trope. Take this simple trope example:

Listing 4.16: Subtrope to be embedded

```

1 "Item Search" is a trope where:
2   The Macguffin is an object
3   The Hero is a role
4   Home is a place
5
6   The Hero chases the Macguffin
7   Then the Hero finds the Macguffin
8   Or the Hero goes Home

```



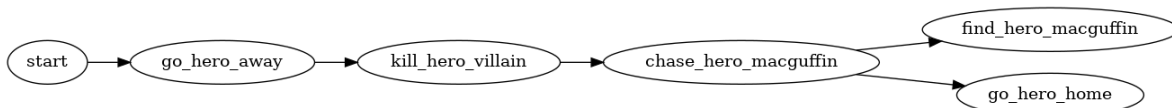
To embed this trope inside another trope, we refer to it by name by writing *Then the “Item Search” trope happens*. It is important to put the name of the trope inside quotation marks. The reason for this is that it makes subtrope embedding statements easier to parse, as trope names may contain combinations of role, object and place names along with verbs. Putting the name of a trope inside quotation marks ensures that the parser will not confuse it with any other kind of statement. An example of putting this trope at the end of another trope is shown in the following example:

Listing 4.17: Trope containing a subtrope

```

1 "Kill then Search" is a trope where:
2   Away is a place
3   The Hero is a role
4   The Villain is a role
5
6   The Hero goes Away
7   Then the Hero kills the Villain
8   Then the "Item Search" trope happens

```

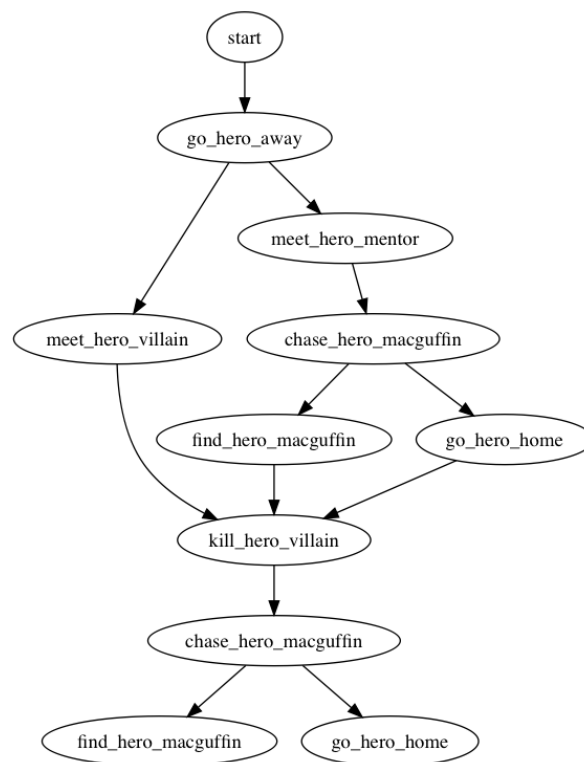


The “Item Search” subtrope is thus embedded inside the “Kill then Search” trope, with all of the events and branches of “Item Search” being appended to the end of “Kill then Search”. Rather than only being embedded at the end of a trope however, subtropes can appear at any point in a trope, or even at several points in a trope. Take this example that uses the “Item

Search” trope inside nested branches of its story:

Listing 4.18: Subtrope in multiple places

```
1 "Futile Search" is a trope where:  
2   The Hero is a role  
3   The Villain is a role  
4   The Mentor is a role  
5   Away is a place  
6   Home is a place  
7   The Land of Adventure is a place  
8   The Realm of Mystery is a place  
9  
10  The Hero goes Away  
11  Then the Hero meets the Villain  
12    Or the Hero meets the Mentor  
13    Then the "Item Search" trope happens  
14  Then the Hero kills the Villain  
15  Then the "Item Search" trope happens
```



An author can create complex branching narrative by combining these tropes together. To see visualisations of the story paths of multiple tropes combined, see Section 5.1.5 on page 115 of Chapter 5 on page 101.

For technical details of the parser implementation, including the EBNF grammar of the TropICAL language, refer to Appendix D on page 209.

4.4 InstAL Code Generation

This section describes how and why language features translate to InstAL, with fully translated examples of InstAL code appearing in Appendix C on page 200.

4.4.1 Answer Set Programming (ASP)

Answer Set Programming (ASP) uses a *declarative* programming paradigm, meaning that programs described in an ASP language (such as AnsProlog) describe the problem to be solved, as well as its solution in terms of success criteria. Procedural programming languages (such as Java or Python) require the user to describe *how* to solve the problem. With the procedural approach, a *Sudoku* program would consist of the steps that are taken when a Sudoku game is played. Contrast this with the declarative programming approach, where the program would be created by logically describing the rules of the game, which are used to search for potential solutions when given number grids as input.

An ASP program consists of three elements: a knowledge base of “facts”, rules for generating sets, and constraints that limit the sets output from the solver to only “answer” sets.

The rules of an ASP program are of the following form:

```
1 <head> :- <body> .
```

This can be read as “head is true if body is true”. If the body part of a clause is empty, then the clause is known as a *fact*. This is the equivalent of just the head of a rule with no conditional body following it. For example:

```
1 hero(charlie).
```

An answer set is the minimal set of atoms that satisfy the rules program in an consistent manner. These atoms only appear in the answer set if a rule or fact from the program justifies its inclusion, without relying on the answer set itself.

ASP Example: A Sudoku Puzzle

As an example, we describe the creation of a Sudoku puzzle game in AnsProlog. The code for this example is taken from Schweizer (2017).

“Sudoku” is a famous number puzzle where numbers 1-9 are arranged on a 9×9 grid. Figure 4-2 on the next page shows an example of a Sudoku grid. The 9×9 board consists of 9 squares of 3×3 cells. The aim of Sudoku is to put numbers into each cell so that each row, column and square contains no duplicate numbers (numbers 1-9 appear only once).

In an ASP Sudoku solver, the program itself would consist of the game rules (that numbers 1-9 are arranged in $9 \times 3 \times 3$ squares), along with the constraints of the game:

- No number can be repeated inside a square
- No number can be in the same column or row

	2		5		1		9	
8			2		3			6
	3			6			7	
		1				6		
5	4						1	9
		2				7		
	9			3			8	
2			8		4			7
	1		9		7		6	

4	2	6	5	7	1	3	9	8
8	5	7	2	9	3	1	4	6
1	3	9	4	6	8	2	7	5
9	7	1	3	8	5	6	2	4
5	4	3	7	2	6	8	1	9
6	8	2	1	4	9	7	5	3
7	9	4	6	3	2	5	8	1
2	6	5	8	1	4	9	3	7
3	1	8	9	5	7	4	6	2

Figure 4-2: Left: an unsolved Sudoku puzzle. Right: The solution to the puzzle. (Images from Schweizer (2017))

Listing 4.19 shows AnsProlog facts that describe the layout of a Sudoku board. Line 1 states that the board contains numbers from 1 to 9, rows numbered 0 to 8 and columns also numbered 0 to 8. Lines 3 to 5 describe the nine squares of the Sudoku board, each containing three rows and columns of number cells, where square 0 contains rows 0 to 2, and also columns 0 to 2, square 1 contains rows 0 to 2 and columns 3 to 5, all the way to square 8 (both rows and columns are numbers 6 to 8).

Listing 4.19: AnsProlog facts for the Sudoku board

```

1 number(1..9). row(0..8). column(0..8).
2
3 square(0, 0..2, 0..2). square(1, 0..2, 3..5). square(2, 0..2, 6..8).
4 square(3, 3..5, 0..2). square(4, 3..5, 3..5). square(5, 3..5, 6..8).
5 square(6, 6..8, 0..2). square(7, 6..8, 3..5). square(8, 6..8, 6..8).

```

Once the layout of the Sudoku board has been defined, we can begin describing the rules of the game through *constraints*. A constraint limits the search space of possible solutions by describing behaviour that is not allowed as part of the system. For example, Listing 4.20 shows the AnsProlog rules that state that a number can only appear on a grid cell once. This works by stating that each number from 1 to 9 can only appear at location X, Y in the grid if no other number is already there. The listing omits numbers 3 to 8 to save space.

The constraints in Listing 4.21 on the following page describe the rules of the game itself. Lines 1 and 2 state that each number can occur only once in each row and column on the board, and lines 4 to 6 describe how each number (1 to 9) can appear only once in each of the 9 squares on the board.

Listing 4.20: AnsProlog rules limiting one number per cell in Sudoku

```

1 cell(X, Y, 1) :- row(X), column(Y),
2                 not cell(X, Y, 2), not cell(X, Y, 3), not cell(X, Y, 4),
3                 not cell(X, Y, 5), not cell(X, Y, 6), not cell(X, Y, 7),

```

```

4         not cell(X, Y, 8), not cell(X, Y, 9).
5
6 cell(X, Y, 2) :- row(X), column(Y),
7         not cell(X, Y, 1), not cell(X, Y, 3), not cell(X, Y, 4),
8         not cell(X, Y, 5), not cell(X, Y, 6), not cell(X, Y, 7),
9         not cell(X, Y, 8), not cell(X, Y, 9).
10
11         ...
12
13 cell(X, Y, 9) :- row(X), column(Y),
14         not cell(X, Y, 1), not cell(X, Y, 2), not cell(X, Y, 3),
15         not cell(X, Y, 4), not cell(X, Y, 5), not cell(X, Y, 6),
16         not cell(X, Y, 7), not cell(X, Y, 8).

```

Listing 4.21: AnsProlog constraints for the rules of Sudoku

```

1 :- cell(X, Y1, N), cell(X, Y2, N), Y1 != Y2.
2 :- cell(X1, Y, N), cell(X2, Y, N), X1 != X2.
3
4 in_square(S, N) :- cell(X, Y, N), square(S, X, Y).
5                 :- number(N), not in_square(S, N),
6                 square(S, _, _).

```

Now that we have described the rules of the game of Sudoku in AnsProlog, we can use a *solver* (Potassco’s *clingo* (Gebser et al., 2011), in this case) to compute models that represent its solution. Each of these models is an *answer set*, and represents a stable solution to the game.

Finding solutions for an ASP program is a two-step process. The first step is to *ground* any free variables in the program rules. This is done by generating facts to replace all the possible values of each variable. The Answer Set Solver then searches the space of these possible solutions to find those that are *stable* (ones that do not contain any contradictions).

If the solver is run with just the game description, it will generate answer sets for all possible combinations of numbers in the Sudoku board. However, a Sudoku game begins with some of the numbers of the board already filled in. It is up to the player to determine which numbers can fill in the rest of the grid. In order to do this, we must add facts that describe the initial conditions of a Sudoku game: the numbers that are filled in already. Listing 4.22 shows facts that represent a specific game of Sudoku by describing the cells that already contain numbers. When the solver is run with this input, the resulting answer set (shown in Listing 4.23 on the following page) consists of the game solution in terms of numbers and the cells they are located in. Figure 4-3 on page 75 shows the Sudoku game with the solutions from the answer set as blue numbers.

Listing 4.22: AnsProlog facts for a particular game of Sudoku

```

1 cell(0, 0, 3). cell(0, 4, 8). cell(0, 6, 6). cell(0, 8, 7).
2 cell(1, 1, 1). cell(1, 6, 4). cell(1, 8, 9).
3 cell(2, 0, 8). cell(2, 1, 9). cell(2, 4, 6). cell(2, 5, 7).
4 cell(3, 1, 6). cell(3, 3, 1). cell(3, 4, 9). cell(3, 6, 7).

```

```

5 cell(4, 2, 9). cell(4, 3, 6). cell(4, 4, 5). cell(4, 8, 2).
6 cell(5, 2, 2). cell(5, 7, 1).
7 cell(6, 1, 5). cell(6, 4, 4). cell(6, 8, 3).
8 cell(7, 1, 4). cell(7, 3, 2). cell(7, 7, 9). cell(7, 8, 8).
9 cell(8, 1, 8). cell(8, 2, 6). cell(8, 4, 3). cell(8, 6, 1).

```

Listing 4.23: Answer set corresponding to a particular Sudoku solution

```

1 cell(0,0,3) cell(0,1,2) cell(0,4,8) cell(0,3,9) cell(0,5,1) cell(0,8,7)
2 cell(0,6,6) cell(0,2,4) cell(0,7,5)
3
4 cell(1,0,6) cell(1,1,1) cell(1,6,4) cell(1,8,9) cell(1,4,2) cell(1,7,8)
5 cell(1,2,7) cell(1,3,3) cell(1,5,5)
6
7 cell(2,0,8) cell(2,1,9) cell(2,4,6) cell(2,5,7) cell(2,3,4) cell(2,8,1)
8 cell(2,2,5) cell(2,6,3) cell(2,7,2)
9
10 cell(3,1,6) cell(3,3,1) cell(3,4,9) cell(3,6,7) cell(3,2,8) cell(3,5,2)
11 cell(3,0,4) cell(3,7,3) cell(3,8,5)
12
13 cell(4,2,9) cell(4,3,6) cell(4,4,5) cell(4,8,2) cell(4,0,1) cell(4,1,7)
14 cell(4,6,8) cell(4,5,3) cell(4,7,4)
15
16 cell(5,2,2) cell(5,7,1) cell(5,1,3) cell(5,4,7) cell(5,6,9) cell(5,8,6)
17 cell(5,0,5) cell(5,3,8) cell(5,5,4)
18
19 cell(6,1,5) cell(6,4,4) cell(6,8,3) cell(6,2,1) cell(6,0,9) cell(6,3,7)
20 cell(6,5,8) cell(6,6,2) cell(6,7,6)
21
22 cell(7,1,4) cell(7,3,2) cell(7,7,9) cell(7,8,8) cell(7,2,3) cell(7,4,1)
23 cell(7,0,7) cell(7,5,6) cell(7,6,5)
24
25 cell(8,1,8) cell(8,2,6) cell(8,4,3) cell(8,6,1) cell(8,0,2) cell(8,3,5)
26 cell(8,5,9) cell(8,7,7) cell(8,8,4)

```

Prolog (Clocksin and Mellish, 2003) is another well-known language for declarative programming, widely used in academia and industry. The benefit of using ASP over Prolog comes from the fact that it is fully declarative. This means that the ordering of rules in AnsProlog programs do not change the meaning of the program, whereas they do in Prolog.

4.4.2 InstAL: The Institution Action Language

Cliffe’s thesis (Cliffe et al., 2007) introduces *InstAL*, a programming language for the specification of Social Institutions (such as those described in Section 2.2.4 on page 35) for multi-agent systems. InstAL code compiles to the AnsProlog language for Answer Set Programming, allowing the use of an answer set solver to generate answer sets corresponding to permitted and obliged agent actions. This section provides a brief introduction to the InstAL institutional model.

An institution is a collection of social norms that govern the behaviour of a set of inter-

	1	2	3	4	5	6	7	8	9
A	3	6	8	4	1	5	9	7	2
B	2	1	9	6	7	3	5	4	8
C	4	7	5	8	9	2	1	3	6
D	9	3	4	1	6	8	7	2	5
E	8	2	6	9	5	7	4	1	3
F	1	5	7	2	3	4	8	6	9
G	6	4	3	7	8	9	2	5	1
H	5	8	2	3	4	1	6	9	7
I	7	9	1	5	2	6	3	8	4

Figure 4-3: The solved Sudoku board for our example. Numbers from the input are black, solutions are blue.

acting agents. In the real world, social norms encourage certain behaviours and discourage others. However, people are still able to carry out the discouraged behaviours, as there is no physical barrier preventing them. Instead, the discouraged behaviours are punished with *social* consequences such as shaming or expulsion from a social group.

For example, when a teacher is lecturing a class, there is an implicit social contract for the students to remain silent while the teacher is talking. Though it is physically possible for a student to interrupt the teacher, they face the possibility of being told to leave the class if they do so. The institutions we describe in this chapter follow the same model: behaviours are permitted or obliged to happen in a contract that all agents follow. When the agents perform actions that are not permitted, or do not perform obliged actions before their deadline, they *violate* the norms in the institution. As a consequence, a punishment corresponding to the violated norm is applied to the offending agent.

Using a social institution as a story model governs character agents' actions instead of enforcing them. This allows agents to be guided by the norms described in the story model, but also allows them to break away from the model at times. An agent could either plan to break the norms of the story if it is willing to face the consequences, or it could deviate from the story in extreme situations (such as in times of extreme emotion). This potentially gives agents more autonomy to act within the guidelines of a story, so that their actions may generate a wider range of interesting scenarios. The extent to which our system achieves this “character freedom” (if at all) is discussed in Section 8.2 on page 170.

Norms and Institutions

As we describe in Section 2.2.4 on page 35 of the literature review, an institution consists of a set of ‘social’ norms describing the permitted and obliged behaviour of interacting agents.

Noriega’s “Fish Market” thesis (Noriega, 1999) describes how an institutional model can be used to govern the actions of agents in a fish market auction. Several (Artikis et al., 2009; Fornara et al., 2007; Cardoso and Oliveira, 2007) extend this idea to build systems where institutions actively regulate the actions of agents, while still allowing them to decide what to do. We build on the work of Cliffe et al. (2007) and Lee et al. (2013) to adapt it for the world of narrative, using an institutional model to describe story worlds in terms of tropes, through which the actors acquire powers and permissions appropriate to the character and the story function in which they are participating.

Institutional models use concepts from deontic logic to provide obligations and permissions that act on interacting agents in an environment. By combining this approach with our concept of *story tropes*, we can describe a Propp-style formalism of narrative in terms of what agents are *obliged* and *permitted* to do at certain points in the story.

The permissions an agent has, on the one hand, constrain the choices of actions available to them at any given moment. Obligations, on the other hand, affect the goals of an agent. Whether or not an agent actively tries to fulfil an obligation depends on the plan they are following, and their emotional state.

Describing Tropes as Institutions

The previous section describes how *social institutions* can be used to govern the behaviours of story characters to conform to a narrative arc. This section demonstrates how social institutions can be used to implement story tropes in a multi-agent system. Using this method, the tropes are described as social norms which govern the character agents of a story, where an institution describes the norms that govern a certain trope, and a story is a collection of tropes.

In order to describe story tropes in terms of social norms, we break them down into three components:

1. characters, which instantiate roles
2. objects, which instantiate types
3. places, which instantiate locations

Characters’ actions are described in terms of permissions and obligations. For example, a character in a certain role *may* go to the cinema, or a character *must* buy a ticket before the movie begins, otherwise they will not see it. Note that an obligation (which says that a character *must* do something) can have a deadline (“before the movie begins”) and a consequence (“they will not see the movie”). These are both optional in our system.

Returning to the tropes described in the introduction, we can express them in terms of social norms:

- **The Hero’s Journey:** The hero *must* leave home when they receive the call to adventure. Then the hero *may* kill the villain. Once this is done, the hero *may* return home.

- **The Evil Empire:** The villain has an empire, and *may* kill the hero.
- **MacGuffin:** The hero *must* search for an object. However, the hero *may* find it.
- **Chekhov’s Gun:** If a weapon appears in the beginning, it *must* be used before the end of the story.

Describing tropes in terms of permissions and obligations is enough for us to be able to specify them as social norms, which allows us to represent them formally through the use of *InstAL* (Cliffe et al., 2007), the Institution Action Language, a language for describing social institutions which compiles to AnsProlog, an *Answer Set Programming* (ASP) language. Once compiled to InstAL and AnsProlog, our tropes have a formal representation which (for example) allows us to use trope models and an ASP solver to determine which norms hold after agent or player actions have occurred in the story world. This means that we can use ASP to check whether certain sequences of events are possible within a story, given a set of trope descriptions.

As Section 4.4.2 on page 74 explains, an institution describes the *external* events that relate to actions in the agent’s environment, and *institutional* events that occur within the institution. The purpose of external events is to generate institutional events, which initiate and terminate social norms (permissions and obligations). When a trope is described as an institution, it must state which actions by the agents lead to the triggering of institutional events, and which norms these institutional events initiate or terminate.

Take the case of a *Star Wars* game, for example. Any agent has the ability to pick up a Lightsaber weapon. However, this action becomes institutionally meaningful if we state that it is the first event of the *Hero’s Journey* trope, where the Hero answers the Call to Adventure by obtaining a weapon. For example, an agent playing the role of Luke Skywalker in a *Star Wars* game may pick up a Lightsaber. If Luke Skywalker is a hero character, and a Lightsaber is a type of weapon, this would trigger an event inside the *Hero’s Journey* institution where a hero has picked up a weapon. As this corresponds to the first event in the *Hero’s Journey* institution, it would initiate the norms for the next events in the institution to happen, which could be permission to use the weapon, or an obligation to go to the land of adventure. For more details on InstAL, social institutions, and the formalism in Figure 4-4 on the following page, refer to Section 4.4.2 on page 74 and (Cliffe et al., 2007).

Using the same notation introduced in Section 4.4.2 on page 74, Figure 4-4 on the next page lists some external ($\mathcal{E}_{external}$) and institutional (internal, $\mathcal{E}_{internal}$) events for the *Hero’s Journey* trope. A wide range of external events such as *go*, *meet*, *kill*, *escape* generate the *intHerosJourney* internal event, but only if the external event meets certain criteria. These criteria could be whether or not an agent fulfils a certain role, for example. Figure 4-5 on page 79 shows examples of such internal event generation (\mathcal{G}). In the first example (rule 4.3), the *intHerosJourney* event is generated when Luke Skywalker goes to Tatooine, but only if Luke has the role of *hero*, and Tatooine’s location is *home*. Figure 4-5 on page 79 shows how internal events initiate (\mathcal{C}^\uparrow) fluents and norms (permissions and obligations) in a trope. Because the *Hero’s Journey* trope has several stages, this example only shows the first two

$$\mathcal{E}_{external} = \left\{ \begin{array}{l} \text{go}(\text{Agent}, \text{Place}), \\ \text{meet}(\text{Agent}, \text{Agent}), \\ \text{kill}(\text{Agent}, \text{Agent}) \\ \text{escape}(\text{Agent}) \end{array} \right\} \quad (4.1)$$

$$\mathcal{E}_{internal} = \left\{ \begin{array}{l} \text{intHerosJourney}(\text{Agent}, \\ \text{Agent}, \text{Agent}, \text{Place}, \text{Place}) \end{array} \right\} \quad (4.2)$$

Figure 4-4: External and institutional events (\mathcal{E}) for the *Hero's Journey* trope

events of the trope (the *phase* keywords are explained further in Section 4.4.7 on page 87). Rule 4.6 shows how the *intHerosJourney* internal event initiates the hero's permission to kill the villain, an obligation for the hero to go to the Land of Adventure before the villain kills the victim, and the next phase (phase C) of the *Hero's Journey* trope. These fluents are only initiated if the *intHerosJourney* internal event happens while the trope is in phase B (when $\text{phase}(\text{herosJourney}, \text{phaseB})$ holds), however. Fluent termination (\mathcal{C}^\downarrow) works in a similar manner to initiation, with permissions and obligations for previous trope phases being terminated once the next phase of a trope has been entered. Examples for the first two phases of the *Hero's Journey* trope are shown in Figure 4-5 on the next page.

While InstAL allows us to express tropes as social institutions, it would be difficult to use for non-programmers who are unfamiliar with logic programming paradigms. In order for story authors to be able to create their own tropes, a much more user-friendly language is needed. This is the motivation for TropICAL, and the reason we are describing tropes in that language rather than having users write InstAL code directly.

Fluents

These are properties that may or may not hold true at some instant in time, and that change over the course of time. *Institutional events* are able to *initiate* or *terminate* fluents at points in time. A fluent could describe whether a character is currently on stage, the scene of the story that is currently being acted out, or whether or not the character is happy at that moment in time. Domain fluents (\mathcal{D}) describe domain-specific properties that can hold at a certain point in time.

Institutional fluents consist of (institutional) *powers*, *permissions* and *obligations*. An **institutional power** (\mathcal{W}) describes whether or not an external event has the authority to generate a meaningful institutional event. Taking an example from Propp's formalism, an *absentation* event can only be generated by an external event brought about by a *donor* character (such as their leaving the stage). Therefore, any characters other than the donor character would not have the institutional power to generate an *absentation* institutional event when they leave the stage.

The generation relation \mathcal{G} for trope state \mathcal{X} and external event \mathcal{E} in the *Hero's Journey* trope:

$$\mathcal{G}(\mathcal{X}, \mathcal{E}) : \left\{ \begin{array}{l} \langle \{role(lukeSkywalker, hero), \\ location(tatooine, home)\}, \\ go(lukeSkywalker, tatooine) \rangle \rightarrow \{intHerosJourney(lukeSkywalker, \\ R, S, tatooine, T)\} \end{array} \right. \quad (4.3)$$

$$\left\{ \begin{array}{l} \langle \{role(lukeSkywalker, hero), \\ role(obiWan, dispatcher)\}, \\ meet(lukeSkywalker, obiWan) \rangle \rightarrow \{intHerosJourney(lukeSkywalker, \\ obiWan, R, S, T)\} \end{array} \right. \quad (4.4)$$

The fluent initiation relation \mathcal{C}^\uparrow for trope state \mathcal{X} and internal event \mathcal{E} in the *Hero's Journey* trope:

$$\mathcal{C}^\uparrow(\mathcal{X}, \mathcal{E}) : \left\{ \begin{array}{l} \langle \{phase(herosJourney, phaseA)\}, \\ intHerosJourney(hero, dispatcher, villain, \\ home, landOfAdventure) \rangle \rightarrow \left\{ \begin{array}{l} perm(meet(hero, dispatcher)) \\ phase(herosJourney, phaseB) \end{array} \right\} \end{array} \right. \quad (4.5)$$

$$\left\{ \begin{array}{l} \langle \{phase(herosJourney, phaseB)\}, \\ intHerosJourney(hero, dispatcher, villain, \\ home, landOfAdventure) \rangle \rightarrow \left\{ \begin{array}{l} perm(kill(hero, villain)) \\ obl(go(hero, landOfAdventure)), \\ kill(villain, victim), \\ viol(story, end) \\ phase(herosJourney, phaseC) \end{array} \right\} \end{array} \right. \quad (4.6)$$

The fluent termination relation \mathcal{C}^\downarrow for trope state \mathcal{X} and internal event \mathcal{E} in the *Hero's Journey* trope:

$$\mathcal{C}^\downarrow(\mathcal{X}, \mathcal{E}) : \left\{ \begin{array}{l} \langle \{phase(herosJourney, phaseA)\}, \\ intHerosJourney(hero, dispatcher, villain, \\ home, landOfAdventure) \rangle \rightarrow \left\{ \begin{array}{l} perm(go(hero, home)), \\ phase(herosJourney, phaseA) \end{array} \right\} \end{array} \right. \quad (4.7)$$

$$\left\{ \begin{array}{l} \langle \{phase(herosJourney, phaseB)\}, \\ intHerosJourney(hero, dispatcher, villain, \\ home, landOfAdventure) \rangle \rightarrow \left\{ \begin{array}{l} perm(meet(hero, dispatcher)) \\ phase(herosJourney, phaseB) \end{array} \right\} \end{array} \right. \quad (4.8)$$

Figure 4-5: Generation events, and fluent initiation and termination for the Hero's Journey trope

Permissions

(\mathcal{P}) are associated with external actions that agents are permitted to do at a certain instant in time. These can be thought of as the set of *socially permitted* actions available to an agent. While it is possible for an agent to perform other actions, societal norms usually discourage them from doing so.

Obligations

(\mathcal{O}) are institutional facts that contain actions agents *should* do before a certain deadline. If the action is not performed in time, a *violation event* is triggered, which may result in a penalty being incurred. While an agent may be obliged to perform an action, it is entirely their choice whether or not they actually do so. They must weigh up whether or not pursuing other courses of action is worth accepting the penalty that an unfulfilled obligation brings.

Events

Cliffe’s model specifies three types of **event**: *external events* (or ‘observed events’, \mathcal{E}_{obs}), *institutional events* ($\mathcal{E}_{insevent}$) and *violation events* (\mathcal{E}_{viol}).

External events are observed to happen in the agents’ environment, which can *generate institutional events* which occur only within the institutional model, leading to the *initiation* or *termination* of (domain) fluents, permissions, obligations or institutional powers. An external event could be an agent leaving the stage, an agent hitting another, or an agent dying. Internal events include narrative events such as scene changes, or the triggering of story functions such as Propp’s *absentation* or *interdiction* (described in Section 2.1.3 on page 24).

Violation events occur when an agent has failed to fulfil an obligation before the specified deadline. These can be implemented in the form of a penalty, by decreasing an agent’s health, for example.

Event Generation and Consequences

An **event generation** function, \mathcal{G} , describes how events (\mathcal{E} , usually external, but can also be internal) can generate other (usually institutional) events, conditional upon the current institutional state (\mathcal{X}). This is the counts-as relation.

Event generation functions follow a $\langle \text{preconditions} \rangle \rightarrow \{ \text{postconditions} \}$ format. The preconditions consist of a set of fluents that hold at that time, along with an event to have occurred. The postconditions are the events that are generated. The generation functions are used to generate internal, institutional events from external events.

Consequences consist of fluents, permissions and obligations that are *initiated* (\mathcal{C}^\uparrow) or *terminated* (\mathcal{C}^\downarrow) by institutional events.

A more elaborate example of an institution for the “Punch and Judy” story world, described in terms of story tropes, appears in Chapter 6 on page 117.

4.4.3 Why use ASP and InstAL?

Answer Set Programming is a useful tool for describing tropes and stories, as it allows us to formally construct a story model, against which possible sequences of events can be checked to see if they are allowed within that model.

As we are simulating our story world with a Multi-Agent System, we need to be able to determine which agent actions are allowed within the story model, given a history of past events that have already occurred. Using an answer set solver, we can query the model to check which agent actions are valid (according to the story) while the simulation is running.

Additionally, when designing our story model, the use of an answer set solver allows us to generate answer sets corresponding to all of the possible paths through a story. This can be useful when attempting to visualise the branches that are created when decision points are inserted into the TropICAL trope descriptions. The StoryBuilder tool described in Chapter 5 on page 101 makes use of this feature of ASP to generate tree diagrams to visualise the possible paths through a story during the process of creating tropes with TropICAL.

The advantage of using ASP over similar alternatives such as Prolog is its handling of negation-as-failure. Due to its use of the stable model semantics, queries to an answer set solver are less likely to get stuck in infinite loops. This makes ASP suitable for use at run time in situations such as a multi-agent system story model simulation.

Our TropICAL language compiles to InstAL before being compiled to the AnsProlog implementation of ASP. This offers several benefits. The first is that it greatly simplifies the process of code generation—rather than having to write a compiler to generate AnsProlog code, it can instead generate a smaller quantity of InstAL code, which can then generate the AnsProlog.

There are other benefits besides convenience, however. InstAL provides a ready-made formal model that describes event-driven scenarios in terms of social norms. As long as our tropes are described in terms of events that occur in a story, they can be translated into this formal model. This allows great flexibility in the definition of tropes, so that authors can describe them semi-informally in a language that resembles natural English. Once events of a trope are described this way, it is relatively straightforward to describe them in terms of permissions and obligations in InstAL.

Another benefit of using InstAL is that its institutional model allows us to observe when characters deviate from the story path, by generating violation events. When a violation event occurs, the potentially story-breaking action can be handled using a strategy such as changing the active tropes in the running simulation (Section 7.3 on page 165 gives an example of how this kind of scenario can be dealt with).

4.4.4 Compilation Strategy

The steps that the TropICAL compiler goes through to produce InstAL code are the following:

1. **Parse entity definitions:** The parser only looks at the first lines of the file that define the roles, objects and places (such as *The Hero is a role* or *The Sword is an object*)
2. **Parse the trope's events:** The parser then examines the rest of the trope, with the previously defined entity definitions inserted into the parser's grammar as keywords.
3. **Transform the parse tree into a hash map:** The hash map is an intermediate representation of the trope entities and events described in the TropICAL code.
4. **Generate InstAL institutions from the hash map representation:** The hash map representation of each TropICAL trope is converted into a corresponding InstAL institution.

Once the tropes written in TropICAL have been parsed and converted into an intermediate hash map, this data structure is then used to generate InstAL code. Each TropICAL trope compiles to a separate InstAL institution, each of which is contained in its own file. Example pieces of code of the anatomy of an InstAL institution are shown in Listing 4.26 on page 85.

An InstAL institution contains the following parts:

- *Institution Name*: the name of the institution (Line 1 of Listing 4.26 on page 85)
- *Type Declarations*: types of entity that appear in the institution, such as Agent, Object, and Place (lines 4 - 6)
- *Fluent Declarations*: fluents are facts that tell us something about the agents' environment (lines 9 - 10)
- *Exogenous Event Declarations*: exogenous (external) events are events that occur in the agents' environment, such as actions taken by the agents, as opposed to actions that occur within an institution (lines 13 - 14)
- *Violation Event Declarations*: events that are triggered when obligations are not fulfilled (line 17)
- *Institutional Event Declarations*: these are events that occur inside the institution to initiate and terminate norms (permissions and obligations) and fluents. They are generated by the exogenous events, and have a name that is prefixed with "int" (such as "intHerosJourney") (lines 20 - 22)
- *Obligation Fluent Declarations*: obligation fluents have three parts: the institutional event that is obliged to happen, the institutional event that acts as a deadline, and the violation event. The institutional (obligation) event must occur before the deadline event happens. If the deadline event happens before the obligation event, then the violation event is triggered (line 25, obligation events are described further in Section 4.4.10 on page 90)
- *Norm / fluent initiation events*: this part of an institution specifies which norms and fluents are initiated by its institutional (internal) events (lines 28 - 33, described in Section 4.4.7 on page 87)
- *Norm / fluent termination events*: this specifies which norms and fluents are terminated by institutional events (lines 36 - 41, described in 4.4.8)
- *Generation events*: institutional events that are generated by exogenous (external) events (line 44 - 51, described in Section 4.4.6 on page 86)
- *Norms / fluents that hold initially in the institution*: this describes the initial state of the institution when it becomes active, consisting of the norms corresponding to the trope's first event, and the fluents that describe the role, object and place entities in the trope (lines 54 - 62, described in Section 4.4.5 on page 84)

Intermediate Hash Map Representation

Once a TropICAL definition of a trope has been parsed, its parse tree is converted into a hash map, which is used as an intermediate data structure from which different outputs can be

produced. A hash map consists of key-value pairs, and is used as a table to look up values that correspond to certain keys.

In our case, we are generating the InstAL code from this hash map, but the purpose of using this intermediate data structure is to simplify the process of compiling to other programming languages. This would make it easier, for example, to add code that generates AnsProlog code directly rather than compiling to InstAL first.

The format of the hash map is as follows:

- **Label:** A string representing the name of the trope
- **Events:** A list of events that occur in the trope (see below for a description of an event)
- **Roles:** A list of strings describing character names for the trope
- **Objects:** A list of strings that are names of objects in the trope
- **Locations:** A list of strings naming places in the trope

Each event is a separate hash map consisting of a subject (which corresponds to a *role* key), a predicate (a *verb* key) and an object. The object key can be a *role*, *object* or *place*. In the case where both subject and object are roles, the keys are renamed to *role-a* and *role-b*. In the case where branching events occur in a trope, the event's only key is *or*, whose value is a list of alternative events that could happen.

Listing 4.24 shows a simplified “MacGuffin” trope in TropICAL. Once this trope has been parsed, its parse tree is processed and converted to the hash map representation shown in Listing 4.25. In the listing, a hash map is contained inside curly braces, with keys being unquoted words preceded by a colon and values immediately following keys. Lists of values are contained inside square brackets.

Listing 4.24: A simplified example of the “MacGuffin” trope

```
1 "MacGuffin" is a trope where:
2   The Macguffin is an object
3   The Hero is a role
4   The Villain is a role
5   Home is a place
6   The Hero chases the Macguffin
7   Then the Hero finds the Macguffin
8     Or the Villain finds the Macguffin
9   Then the Hero goes Home
```

Listing 4.25: Hash Map representation of the “MacGuffin trope from Listing 4.24”

```
1 {:label "MacGuffin",
2   :events [{:role "Hero", :verb "chase", :object "Macguffin"},
3             {:or [{:role "Hero", :verb "find", :object "Macguffin"}
4                  {:role "Villain", :verb "find", :object "Macguffin"}]}],
5   :roles ("The Hero" "The Villain"),
```

```

7  :objects ("The Macguffin"),
8  :locations ("Home")}

```

When a trope's hash map representation is converted into InstAL, each entity (role, place, object) is assigned a letter to be used as a variable in the InstAL events. For example, the *intHerosJourney* institutional event on line 20 of Listing 4.26 on the following page is declared with four parameters of types *Agent*, *Agent*, *PlaceName*, *PlaceName*. When this institutional event is used to initiate fluents on line 28, its parameters are input as variables *R*, *S*, *T* and *U*. In the original trope definition, the *Hero's Journey* contains the *Hero*, *Villain* and *Mentor* roles, and the *Land of Adventure* location. When the *intHerosJourney* event happens, its variables are filtered using the *if* keyword to ensure that $R = \textit{Hero}$, $S = \textit{Villain}$, $T = \textit{Mentor}$ and $U = \textit{Land of Adventure}$. An example of this appears on lines 32 and 33 of Listing 4.26 on the next page.

The following sections describe how each part of a TropICAL trope compiles to the components of an InstAL institution as show above and in Listing 4.26 on the following page.

4.4.5 Initial Conditions

The *initially* clause in InstAL specifies the fluents that initially hold when the institution is effected. Role, object and place declarations are translated into fluent declarations as part of this clause. For example, for a trope with the following declarations:

```

1 The Hero is a role
2 The Sword is an object
3 The Land of Adventure is a place

```

And the following entity instances:

```

1 Luke Skywalker is a Hero
2 The Lightsaber is a Sword
3 The Death Star is a Land of Adventure

```

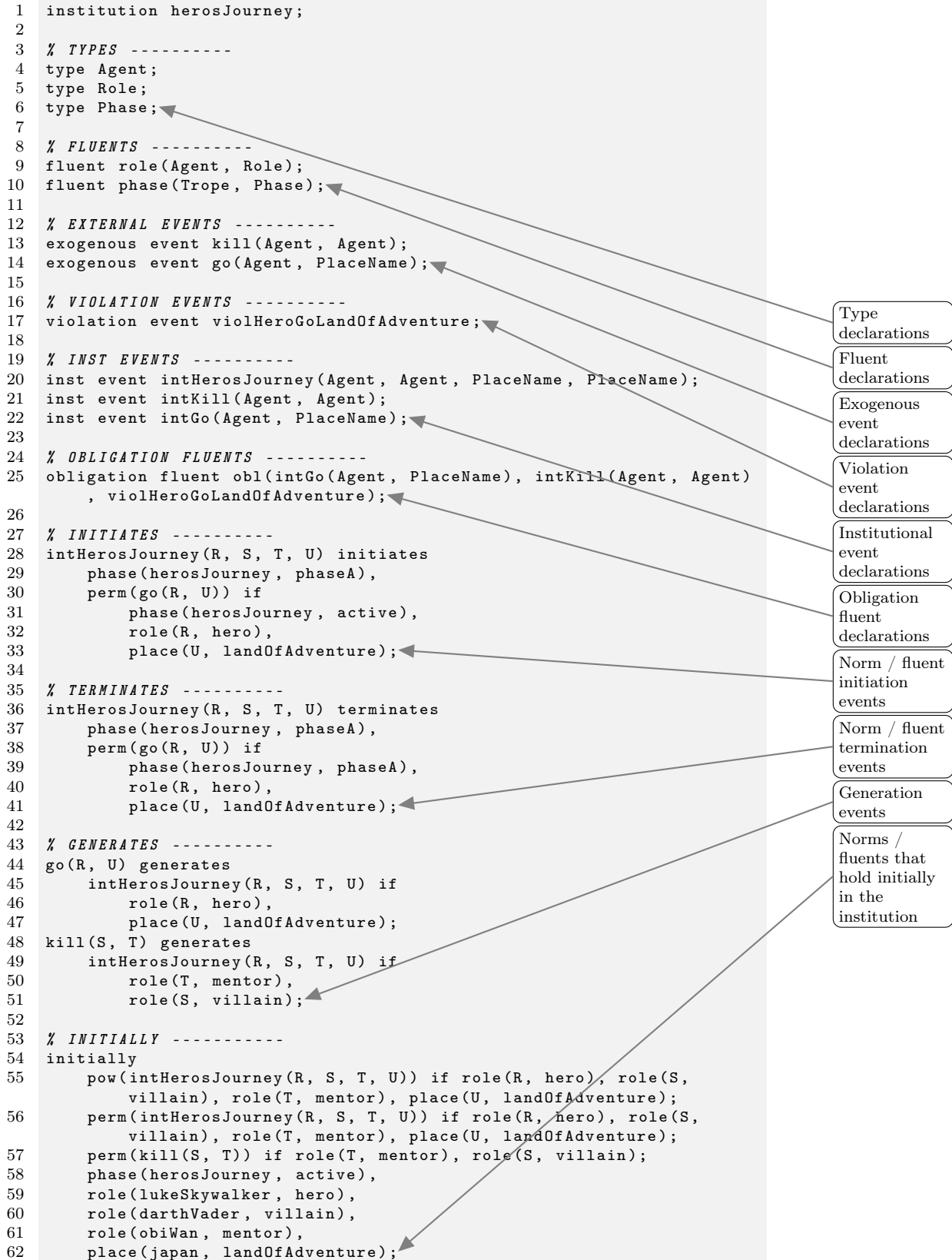
The following InstAL code is produced, with type and fluent declarations automatically appearing at the top of the file:

```

1 type Agent;
2 type Role;
3 type Place;
4 type PlaceName;
5 type Object;
6 type ObjectName;
7
8 fluent role(Agent, Role);
9 fluent place(PlaceName, Place);
10 fluent object(ObjectName, Object);
11
12 initially:
13     role(lukeSkywalker, hero),
14     object(lightSaber, sword),

```

Listing 4.26: Anatomy of an InstAL institution



```
15 place(deathStar, landOfAdventure);
```

The *initially* clause also specifies the permitted and obliged events that occur at the start of the trope. Say our trope contains a series of events (as in listing Listing 4.8 on page 65):

```
1 The Hero goes to the Land of Adventure
2 Then the Hero finds the Sword
3 Then the Hero meets the Villain
4 Then the Hero kills the Villain
5 Then the Hero returns Home
```

In this case, the first event of the trope (*The Hero goes to the Land of Adventure*) must be permitted to happen at the very beginning of the institution, inside the *initially* clause. In combination with the above entity declarations, this results in the following code being generated (omitting the type and fluent declarations this time):

```
1 initially:
2   perm(go(X, Y)) if role(X, hero),
3                     place(Y, landOfAdventure),
4   role(lukeSkywalker, hero),
5   object(lightSaber, sword),
6   place(deathStar, landOfAdventure);
```

The following rules apply as soon as the institution is active:

The Hero may go to the Land of Adventure

“Luke Skywalker” is a hero

“Lightsaber” is a sword

“Death Star” is a Land of Adventure

This means that the Luke Skywalker character is able to go to the Death Star at the beginning of the institution, as is specified in the trope.

4.4.6 Generation

Tropes usually consist of multiple events, so it is not enough to have one event permitted at the start of a trope. Once this first event has occurred, the next event in the sequence must have permission to happen.

As described previously in Section 4.4.2 on page 80, an institution consists of three types of events: *external events* (events that occur in the agents’ environment), *institutional events* (events triggered inside an institution) and *violation events* (events that occur when an obligation is violated in an institution). The institution watches for events to happen as *external* events in the environment, and then has these events generate *institutional* events that occur within the institution and create permissions or obligations for further external events to occur. Returning to Listing 4.8 on page 65 above, the external events that happen are:

- The Hero finds the Sword
- The Hero meets the Villain
- The Hero kills the Villain
- The Hero returns Home

These are translated into *generates* statements in InstAL so that they generate the required institutional events (denoted with an “int” prefix, following the naming convention introduced in Section 4.4.2 on page 80):

Listing 4.27: InstAL code resulting from the compilation of Listing 4.8 on page 65

```

1 find(R, T) generates
2   intSequence3(R, S, T, U, V) if
3     role(R, hero),
4     object(T, sword);
5 kill(R, S) generates
6   intSequence3(R, S, T, U, V) if
7     role(S, villain),
8     role(R, hero);
9 meet(R, S) generates
10  intSequence3(R, S, T, U, V) if
11    role(S, villain),
12    role(R, hero);
13 go(R, V) generates
14  intSequence3(R, S, T, U, V) if
15    role(R, hero),
16    place(V, landOfAdventure);
17 return(R, U) generates
18  intSequence3(R, S, T, U, V) if
19    role(R, hero),
20    place(U, home);

```

The *find* external event generates...

...the *intSequence3* institutional event if...

...it is the *hero* that finds a *sword*.

Each institution only has one institutional event that is triggered by the external events, which is named after the trope itself (in this case, the event is named *intSequence3* after the trope name “Sequence 3”). The permissions and obligations that this institutional event initiates depend on the *phase* of the trope that is currently active (see Section 4.4.7) below.

4.4.7 Initiation

External events generate an internal event for the institution, which permits the next event or events in a sequence to occur. To ensure that the events of the trope are permitted or obliged to occur one after another, we use a mechanism called *event phases*.

Event Phases

An *event phase* is a fluent that holds the state of the current institution. At the beginning of the trope, its state is simply *active*. This is represented in the institution through the fluent *phase(tropeName, active)*. Once the first event has occurred, the trope enters *phase A*, which means that the fluent *phase(tropeName, phaseA)* then holds, and the *phase(tropeName, active)* fluent is terminated (this is described in Section 4.4.8 on the following page). This initiation and termination process then repeats through all phases of the trope, through phases B, C and D if they exist, until the final event occurs and the last phase is terminated.

In the case of our example sequence of events from Listing 4.8 on page 65, there are five phases in total:

- The Hero goes to the Land of Adventure (active)
- The Hero finds the Sword (phase A)
- The Hero meets the Villain (phase B)
- The Hero kills the Villain (phase C)
- The Hero returns Home (phase D)

The events and phases of this trope are initiated in InstAL through the following generated code:

Listing 4.28: Institutional event initiation code for Listing 4.8 on page 65

```

1  intSequence3(R, S, T, U, V) initiates
2     phase(sequence3, phaseA),
3     perm(find(R, T)) if
4         phase(sequence3, active),
5         role(R, hero),
6         object(T, sword);
7  intSequence3(R, S, T, U, V) initiates
8     phase(sequence3, phaseB),
9     perm(meet(R, S)) if
10        phase(sequence3, phaseA),
11        role(S, villain),
12        role(R, hero);
13 intSequence3(R, S, T, U, V) initiates
14    phase(sequence3, phaseC),
15    perm(kill(R, S)) if
16        phase(sequence3, phaseB),
17        role(S, villain),
18        role(R, hero);
19 intSequence3(R, S, T, U, V) initiates
20    phase(sequence3, phaseD),
21    perm(return(R, U)) if
22        phase(sequence3, phaseC),
23        role(R, hero),
24        place(U, home);

```

The *intSequence3* institutional event initiates the following:

The fact that we are in *phase A* of the trope

It gives permission for the *hero* to find the *sword*

If the trope is active...

...and we have both a *hero* and a *sword* as part of the event.

4.4.8 Termination

The generated code that terminates the phases and permissions corresponding to those initiated in Listing 4.28 is shown in listing Listing 4.29 on the next page below. Once an external event occurs, its permission to occur again is terminated along with the phase in which it occurred:

Listing 4.29: Institutional event termination code for Listing 4.8 on page 65

```

1  intSequence3(R, S, T, U, V) terminates
2     phase(sequence3, active),
3     perm(go(R, V)) if
4         phase(sequence3, active),
5         role(R, hero),
6         place(V, landOfAdventure);
7  intSequence3(R, S, T, U, V) terminates
8     phase(sequence3, phaseA),
9     perm(find(R, T)) if
10        phase(sequence3, phaseA),
11        role(R, hero),
12        object(T, sword);
13 intSequence3(R, S, T, U, V) terminates
14    phase(sequence3, phaseB),
15    perm(meet(R, S)) if
16        phase(sequence3, phaseB),
17        role(S, villain),
18        role(R, hero);
19 intSequence3(R, S, T, U, V) terminates
20    phase(sequence3, phaseC),
21    perm(kill(R, S)) if
22        phase(sequence3, phaseC),
23        role(S, villain),
24        role(R, hero);
25 intSequence3(R, S, T, U, V) terminates
26    phase(sequence3, phaseD),
27    perm(return(R, U)) if
28        phase(sequence3, phaseD),
29        role(R, hero),
30    place(U, home);

```

4.4.9 Branches

To represent branching events in InstAL, multiple events are permitted to occur during one phase. Take as an example the combination of event sequences and branches shown in Listing 4.13 on page 67:

```

1  The Hero goes Home
2  Then the Hero finds a Sword
3    Or the Hero goes to the Land of Adventure
4    Or the Hero kills the Villain
5  Then the Hero meets the Mentor
6    Or the Hero goes to the Realm of Mystery

```

A fully compiled institution appears in Appendix C.1 on page 204, which uses the following entity instances:

```

1 Harry Potter is a Hero
2 Voldemort is a Villain
3 Dumbledore is a Mentor
4 The Gryffindor Sword is a Sword
5 Hogwarts is a Land of Adventure
6 The Chamber of Secrets is a Realm of Mystery

```

The branching events are implemented in the *initiates* and *terminates* clauses. In Listing 4.30 below, we see that three possible events are given permission to occur as part of Phase A: *The Hero finds a Sword*, *The Hero goes to the Land of Adventure*, and *The Hero kills the Villain*. In Phase B, two events corresponding to the possible branches are given permission to occur: *The Hero meets the Mentor* and *The Hero goes to the Realm of Mystery*:

Listing 4.30: Initiation events for the branching trope in Listing 4.13 on page 67

```

1 intBranch3(R, S, T, U, V, W, X) initiates
2   phase(branch3, phaseA),
3   perm(find(R, U)),
4   perm(go(R, X)),
5   perm(kill(R, S)) if
6     phase(branch3, active),
7     object(U, sword),
8     role(R, hero),
9     place(X, landOfAdventure),
10    role(S, villain);
11 intBranch3(R, S, T, U, V, W, X) initiates
12   phase(branch3, phaseB),
13   perm(meet(R, T)),
14   perm(go(R, V)) if
15     phase(branch3, phaseA),
16     place(V, realmOfMystery),
17     role(R, hero),
18     role(T, mentor);

```

This means that during Phase A, there are three possible narrative branches, which get closed off when any one of them occurs through the corresponding termination event in the institution.

4.4.10 Obligations

Obligations Without Deadlines or Consequences

In our tropes, obligations have optional deadline events and consequences. At the time of implementation, both of these were mandatory in InstAL, and so it was necessary to add “dummy” events for both if they were not specified in TropICAL. Returning to listing 4.9 on page 65 of Section 4.3.3, which specifies an obligation without a deadline or a consequence:

```

1 The Hero goes Home
2 Then the Hero must go to the Land of Adventure

```

Line 4 of 4.32 on page 91 shows the initiation rule for an obligation event with no deadline or consequence, and listing Listing 4.31 shows the fluent declaration for the obligation fluent. The *intNoDeadline* and *noViolation* events are dummy events that never occur either inside or outside of the institution. There are there only because InstAL always requires deadline and consequence events in obligation fluents. The first parameter of the obligation fluent specifies an institutional event that must occur before the deadline. In this case, the event is *intGo(hero, landOfAdventure)*. Listing 4.33 shows how the *go(hero, landOfAdventure)* external event generates the *intGo(hero, landOfAdventure)* institutional event. Due to the fact that it is this external event that triggers the obliged institutional event, it is given permission to occur on line 4 of Listing 4.32, along with the obliged institutional event itself.

Listing 4.31: Fluent declaration for the obligation event in Listing 4.9 on page 65

```
1 obligation fluent obl(intGo(Agent, PlaceName), intNoDeadline, noViolation);
```

Listing 4.32: Initiation rule for the second (obligation) event of the trope in Listing 4.9 on page 65

```
1 intObligation1(R, S, T) initiates
2   phase(obligation1, phaseA),
3   obl(intGo(R,T), intNoDeadline, noViolation), perm(go(R, T)),
4           perm(intGo(R,T)), pow(intGo(R,T)) if
5   phase(obligation1, active),
6   role(R, hero),
7   place(T, landOfAdventure);
```

Listing 4.33: Generation rule for the obligation event

```
1 go(R, S) generates
2   intGo(R,S) if
3   role(R, hero),
4   place(S, landOfAdventure);
```

The full InstAL institution that this generates appears in Listing C.1 on page 200 of appendix Appendix C on page 200.

Obligations with Deadlines and Consequences

The trope shown in Listing 4.34 describes a situation where an obligation with both a deadline and a consequence occurs.

Listing 4.34: Example of a trope containing an obligation with both a deadline and a consequence

```
1 The Hero must go to the Land of Adventure before the Villain kills the Mentor
2   Otherwise, the Villain may kill the Hero
```

Listing 4.35 on the following page shows the fluent declaration of the obligation event in Listing 4.34. The deadline event *intKill(villain, mentor)* is an institutional event, and so is generated by the *kill(villain, mentor)* external event on lines 5 to 8 of listing 4.38 on page 92.

Listing 4.36 shows the obligation fluent itself. As this obligation is the first event to occur in this trope, it appears in the *initially* clause of the InstAL code, rather than as an initiation event.

If the violation event contained within this obligation were to occur, it would enable the story to begin another course of events. In this case, it initiates the permission of the villain to kill the hero. This is shown in the code of Listing 4.34 on the previous page.

Listing 4.35: Fluent declaration for an obligation fluent with both deadline and consequence events

```
1 obligation fluent obl(intGo(Agent, PlaceName), intKill(Agent, Agent),
    violHeroGoLandOfAdventure);
```

Listing 4.36: An obligation fluent with both a deadline and a consequence, translated from the trope in Listing 4.34 on the previous page

```
1 obl(intGo(R,U), intKill(S,T), violHeroGoLandOfAdventure), perm(go(R, U)),
    perm(intGo(R,U)), pow(intGo(R,U)) if role(R, hero), place(U,
    landOfAdventure);
```

Listing 4.37: Permissions generated by the violation event of the trope in Listing 4.34 on the previous page

```
1 violHeroGoLandOfAdventure initiates
2   perm(kill(R, S)) if
3     role(R, villain),
4     role(S, hero);
```

Listing 4.38: Generation events for the trope in Listing 4.34 on the previous page

```
1 go(R, U) generates
2   intGo(R,U) if
3     role(R, hero),
4     place(U, landOfAdventure);
5 kill(S, T) generates
6   intKill(S,T) if
7     role(S, villain),
8     role(T, mentor);
```

The full InstAL code listing for this institution appears in listing Listing C.2 on page 202 of appendix Appendix C on page 200.

4.4.11 Institutional Bridges

In Section 4.3.5 on page 68, we describe how *subtropes* are implemented in InstAL through the embedding of existing tropes inside of new ones. This is translated into InstAL code through the use of a *Bridge Institution*, a separate institution which is used to link two other institutions together. The concept of a bridge institution is developed by Li (2014), and involves the creation of an intermediary institution to coordinate events and fluents between two other institutions. The two institutions linked are referred to as the *source* and *sink*

institutions, where an event that occurs in the source institution has the power to generate an event inside of the sink institution.

The *Item Search* and *Kill then Search* tropes described in listings Listing 4.16 on page 69 and Listing 4.17 on page 69 can be connected as the *sink* and *source* of a bridge, respectively. In this case, the first event of the sink institution (the *Item Search* trope) is given permission to occur when the final event of the source institution (the *Kill then Search* trope) happens.

Listing 4.39: Bridge for the *Item Search* and *Kill then Search* tropes

```

1  bridge killThenSearchItemSearch;
2
3  source killThenSearch;
4  sink itemSearch;
5
6  cross fluent ipow(killThenSearch, perm(chase(Agent, ObjectName)), itemSearch)
   ;
7  cross fluent ipow(killThenSearch, phase(Trope, Phase), itemSearch);
8
9  intStartItemSearch xinitiates phase(itemSearch, active);
10 intStartItemSearch xinitiates perm(chase(R, S)) if
11     role(R, hero),
12     object(S, macguffin);
13
14 initially ipow(killThenSearch, perm(chase(R, S)), itemSearch),
15     ipow(killThenSearch, phase(itemSearch, active), itemSearch);

```

Listing 4.39 shows the bridge that links the *Item Search* and *Kill then Search* institutions together. It starts by defining *killThenSearch* as the source institution, and *itemSearch* as the sink. Then a cross fluent for the first event in the subtrope (which is *itemSearch*, the sink institution) is declared in line 6. This line only states that this is a fluent that is to be initiated in the sink institution, from the source institution.

Lines 10 to 12 describe the actual sequence of events that lead to the *The Hero chases the MacGuffin* event being triggered inside the sink institution: when the institutional event *intStartItemSearch* occurs inside the source institution, it gives the Hero permission to chase the MacGuffin in the sink institution. The same mechanism is used on lines 7 and 9 to declare and initiate the first phase (the *active* phase) of the sink (*Item Search*) institution from the source (*Kill then Search*) institution. The phase is initiated from the same institutional event inside the source institution: the *intStartItemSearch* event.

Listing 4.40: The generation event for the final action in the *Kill then Search* source institution

```

1  kill(R, S) generates
2     intStartItemSearch if
3     role(S, villain),
4     role(R, hero),
5     phase(killNSearch, phaseA);

```

Listing 4.40 shows the generation event corresponding to the final action that occurs in the *Kill then Search* trope: *The Hero kills the Villain*. The line after this event in the trope

embeds the *Item Search* subtrope within this trope: *Then the “Item Search” trope happens.* For this reason, the *intStartItemSearch* institutional event is initiated, which further initiates the permission for the first event in the *Item Search* institution to happen through the bridge institution in Listing 4.39 on the previous page.

Full listings for the generated InstAL code for these tropes, along with several other examples, appear in Appendix C on page 200.

4.5 Answer Set Generation

Once the code has been translated from TropICAL into InstAL, it can be compiled further into AnsProlog, an Answer Set Programming (ASP) language, using a process described by Cliffe et al. (2007). We use the Potassco project’s Clingo (Gebser et al., 2011) solver to formally verify that certain sequences of events conform to the tropes that form our institutions. This solver can also be used to generate all of the possible sequences of events that fit a story described using one or more tropes, by generating all of the *answer sets* that correspond to a set of institutions.

4.6 Adding Constraints

When our answer sets (traces) are generated by the answer set solver, we want to see sequences of events that can happen as part of a story with a set of given tropes. However, the answer set solver may include some uninteresting events as part of the grounding process, such as *null* events, the dummy events we used to replace unspecified deadlines and consequences for obligations, and events that otherwise violate the rules in each institution.

Listing 4.41 shows the AnsProlog rules used to constrain the answer sets produced by the solver, so that they only consist of events that are relevant to our stories. Line 1 specifies that events are only valid if they are not violation or null events. Line 2 filters for events that occur inside institutions. Line 3 ignores any answer sets where a valid event occurs after a sequence of non-valid events. Lines 5 and 6 ignore our dummy deadline events for obligations where no deadline is specified. Finally, lines 7 and 8 remove any *null* events at all from the answer sets.

Listing 4.41: Constraint rules to remove invalid and irrelevant events

```

1 validEvent(I, In) :- instant(I), inst(In), event(E), occurred(E, In, I), not
    occurred(viol(_), In, I), E != null.
2 validEvent(I) :- validEvent(I, In), inst(In).
3 :- validEvent(I2), not validEvent(I), I < I2, instant(I), instant(I2).
4 :- instant(I), not validEvent(I), occurred(viol(_), In, I).
5 deadEvent :- observed(noDeadline(X), I, T).
6 :- deadEvent.
7 nullInst :- occurred(X, null, T).
8 :- nullInst.

```

The next section lists example answer set (trace) outputs of the solver with two tropes, both separately and combined.

4.7 Example Answer Sets (Traces)

Answer Sets (also called “traces”) are produced by the answer set solver when given the institutions compiled as ASP code, along with a set of constraints. The traces generated represent all of the *possible* sequences of events that may occur, given the described institutions. It is worth noting that by default this also includes any event that violate any of the institutions, as these violations are not prevented by the institutions. Instead, *violation events* are generated which carry consequences with them. By using a constraint such as the one in Listing 4.41 on the preceding page, we can tell the answer set solver to generate only sequences of events that do not contain these violation events. This is what we have done in this section. For visualisations of these answer sets, refer to the diagrams in Section 5.1.1 on page 104.

4.7.1 Evil Empire

Our first example trope is a simple sequence of events called the “Evil Empire”:

Listing 4.42: The “Evil Empire” trope

```
1 ``Evil Empire'' is a trope where:
2   The Empire is a role
3   The Hero is a role
4
5   The Empire chases the Hero
6   Then the Empire captures the Hero
7   Then the Hero escapes
```

This trope, once compiled to InstAL, then AnsProlog, and run through Clingo, produces four answer sets. Each answer set corresponds to a possible sequence of events that can happen in the trope. One answer set contains all the events of the tropes in sequence, another contains just the first two events, another has just the first event and one answer set contains no events at all. The compiled institution is listed in Appendix C.2 on page 206. The traces listed below are solved by looking up to three events into the trope, as that is its maximum length due to the fact that only three events occur in it (*The Empire chases the Hero*, *The Empire captures the Hero*, *The Hero escapes*). The sequences of events in the four answer sets are:

1. Nothing happens
2. The Empire chases the Hero
3. The Empire chases the Hero, Then the Empire captures the Hero
4. The Empire chases the Hero, Then the Empire captures the Hero, Then the Hero Escapes

Listing 4.43 on the next page shows one of the four traces from the solver output. This trace contains each event of the trope, happening one after another. The other traces stop short of the final event, so that trace one has zero events, trace two has only the first event, and trace three has the first two events of the trope.

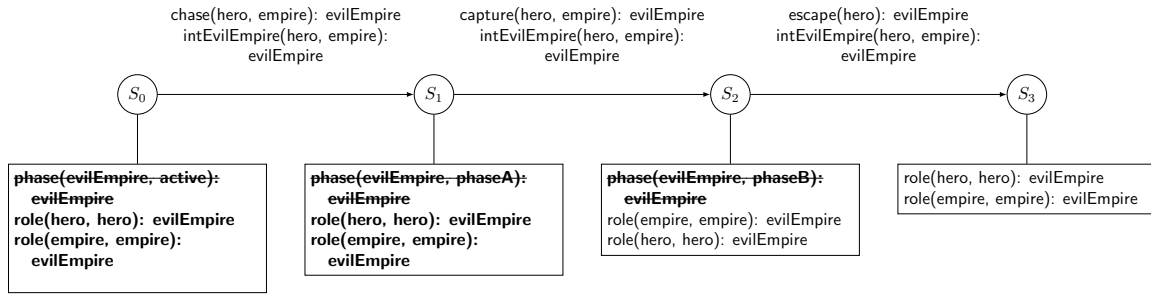


Figure 4-6: Visualisation of the “Evil Empire” example trace in Listing 4.43

Figure 4-6 on the following page shows a visualisation of the events that occur in the answer set in Listing 4.43 on the next page.

Listing 4.43: Example trace for the “Evil Empire” trope

```

1 Answer Set 4:
2
3 Time Step 1:
4
5 holdsat(phase(evilEmpire, phaseA), evilEmpire)
6 holdsat(perm(capture(hero, empire)), evilEmpire)
7 occurred(intEvilEmpire(hero, empire), evilEmpire)
8 occurred(chase(hero, empire), evilEmpire)
9
10
11 Time Step 2:
12
13 holdsat(phase(evilEmpire, phaseB), evilEmpire)
14 holdsat(perm(escape(hero)), evilEmpire)
15 occurred(intEvilEmpire(hero, empire), evilEmpire)
16 occurred(capture(hero, empire), evilEmpire)
17
18
19 Time Step 3:
20
21 occurred(intEvilEmpire(hero, empire), evilEmpire)
22 occurred(escape(hero), evilEmpire)

```

The events that appear at the start of the listing (starting with the *holdsat* predicate) on line 5 describe the *fluents* that hold for a certain institution at that particular time step in the answer set. After these statements, the events that have *occurred* in that institution appear. These events can be both institutional or external events, which can be distinguished due to the institutional event names beginning with “int” (such as *intEvilEmpire*, for example). External events usually trigger institutional events with the institution name, which is why the *chase* (line 8), *capture* (line 16) and *escape* (line 22) events all trigger the *intEvilEmpire* institutional event on lines 7, 15 and 21 of Listing 4.43.

4.7.2 The Hero's Journey

```
1  ``The Hero's Journey'' is a trope where:
2    The Hero is a role
3    The Villain is a role
4    Home is a place
5    The Evil Lair is a place
6
7    The Hero goes to the Evil Lair
8    Then the Hero kills the Villain
9      Or the Villain escapes
10   Then the Hero goes Home
```

The answer set solver produces six traces for this trope. The branching alternatives (the Hero can either kill the Villain or the Villain can escape at one point in the trope) add an extra three answer sets to the output. If this trope were just three events long, without the branching alternatives, then only three answer sets would be produced (corresponding with stories of one, two and three events long). The six answer sets are:

1. Nothing happens
2. The Hero goes to the Evil Lair
3. The Hero goes to the Evil Lair, Then the Hero kills the Villain
4. The Hero goes to the Evil Lair, Then the Hero kills the Villain, Then the Hero goes Home
5. The Hero goes to the Evil Lair, Then the Villain escapes
6. The Hero goes to the Evil Lair, Then the Villain escapes, Then the Hero goes Home

The fourth answer set produced is shown in listing Listing 4.44 as an example, and a visualisation appears in Figure 4-7 on the following page.

Listing 4.44: Example trace for the “Hero’s Journey” trope

```
1  Answer Set 4:
2
3  Time Step 1:
4
5  holdsat(phase(herosJourney,phaseA),herosJourney)
6  holdsat(perm(kill(hero,villain)),herosJourney)
7  holdsat(perm(escape(villain)),herosJourney)
8  occurred(intHerosJourney(hero,villain,evilLair,home),herosJourney)
9  occurred(go(hero,evilLair),herosJourney)
10
11
12 Time Step 2:
13
14 holdsat(phase(herosJourney,phaseB),herosJourney)
```

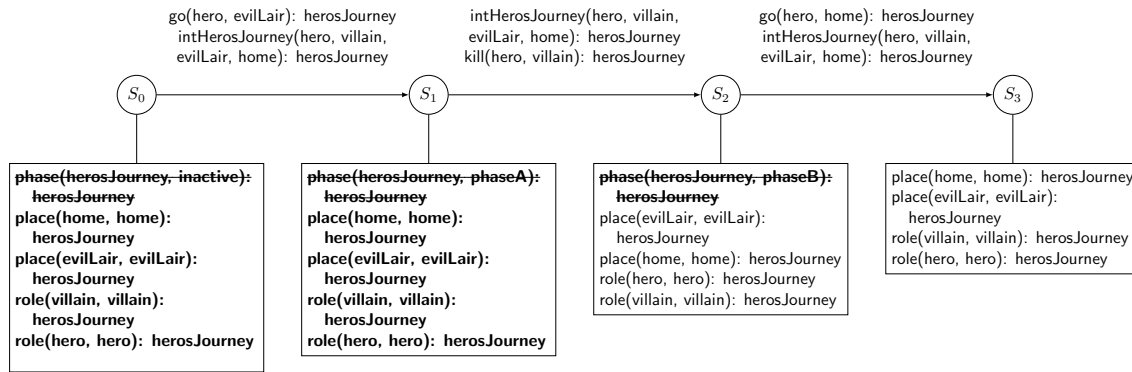


Figure 4-7: Visualisation of the “Hero’s Journey” example trace in Listing 4.44 on the preceding page

```

15 holdsat (perm (go (hero , home)) , herosJourney )
16 occurred (kill (hero , villain) , herosJourney )
17 occurred (intHerosJourney (hero , villain , evilLair , home) , herosJourney )
18
19
20 Time Step 3:
21
22 occurred (intHerosJourney (hero , villain , evilLair , home) , herosJourney )
23 occurred (go (hero , home) , herosJourney )

```

So far, the *Evil Empire* and *Hero’s Journey* tropes produce unremarkable results. Due to their simple natures, they only produce a small number of answer sets when run through a solver. The results are more interesting when both tropes are input into the solver, however.

4.7.3 The Hero’s Journey and The Evil Empire

Using both institutions as inputs into the solver, combining the two tropes, produces 124 answer sets. Combining the tropes in this case means combining the sets of norms for both tropes in order to generate a branching narrative. For example, if the first event of Trope A is “The Hero goes Home”, and the first event of Trope B is “The Villain kills the Hero”, the combination of both of these tropes gives both of these events permission to happen. This means that the story can follow either Trope A or Trope B, adding an extra element of nonlinearity to the story. Due to the fact that this offers many different branching points in a story (where the story may follow either one trope or another), the number of possible different stories increases greatly, in this case to 124 different combinations.

Again, the reason that the solver produces so many answer sets is that it includes stories that are less than the full length of every event from both tropes combined (which is six events in this case). This is desirable when we are combining tropes, as sometimes we need to incorporate events from a trope without needing it to run its full course. In this example, if the Hero’s Journey has come to an end, then that would seem like a more suitable place to finish a story, rather than waiting for all of the events in the Evil Empire trope to occur.

An example of a trace that runs the full six event length of both tropes put together is shown in Listing E.1 on page 212 in Appendix E on page 212. A visualisation is shown in Figure E-1 on page 214 in the same appendix.

These trace outputs do not assist the reader in visualising the possible sequences of events that they represent. For this reason, we suggest that the reader refer to the visualisations produced by the StoryBuilder tool in Section 5.1.1 on page 104.

4.8 Summary

Returning to the requirements listed in Section 4.2.2 on page 62 (listed again here for the reader’s convenience), we can now examine our language specification to see if the requirements are met:

- R1. **The software must be able to integrate into a multi-agent framework such as Jason:** The “Punch and Judy” example in Section 6.2 on page 125 describes how the normative framework of our system integrates with the Jason multi-agent framework. The output of an answer set solver such as Clingo can be used to add beliefs to an agent’s mental model to let it know what actions it is permitted and obliged to carry out as part of a story.
- R2. **The system must direct the behaviour of agents to fit a story description:** The use of permissions and obligations to describe story actions to agents in either a weakly (permissions) or strongly (obligations) enforced way gives the author two ways to direct agent behaviour. At times when the author wants agents to conform tightly with a trope description, they can give the agent an obligation with a strict consequence for non-compliance.
- R3. **The user should be able to write and re-use existing story components (from a library):** Not addressed in TropICAL (see below).
- R4. **The components must be able to express sequences of events:** Section 4.3.2 on page 64 shows how sequences of events can be described in TropICAL.
- R5. **The components must be able to express branching (diverging) events:** Section 4.3.4 on page 66 describes the implementation of branching story structures in TropICAL.
- R6. **The user should have the ability to nest existing components inside new components:** This is an important feature of our language. Section 4.3.5 on page 68 describes the use of subtropes to meet the need for users to create their own abstractions.
- R7. **The user should be able to visualise the branches of the story that result in the addition or modification of components to the story.:** Not addressed in TropICAL (see below).

R8. **The story must tell the character agents what to do, but they should be free to break away from it in extreme circumstances, to add a degree of unpredictability:** This is achieved through the use of permissions to regulate agent actions.

As a text-based programming language, TropICAL addresses the requirements mentioned above (numbers 1, 2, 4, 5, 6 and 8). However, there are two requirements that cannot be fulfilled through a text-based programming paradigm alone: selecting pre-existing story components from a library (requirement 3), and visualising the branches of the story as it is modified (requirement 7). These requirements are best addressed through an Interactive Development Environment (IDE) with which we can add graphical features for the browsing, selection and visualisation of tropes. The next chapter (Chapter 5 on the following page) introduces *StoryBuilder*, a browser-based tool that adds these features to our TropICAL language.

Chapter 5

StoryBuilder: An Interface for Trope-based Interactive Story Creation

Though TropICAL is useful as a programming language that allows for the construction of stories using controlled natural language, there remain several barriers that would prevent its use by non-programmer story authors. It is implemented as a compiler that is invoked from a command line interface with one or more trope definitions as input. Its output is a set of InstAL institutions and the AnsProlog that these institutions compile to. The AnsProlog can then be input into the Clingo solver in order to produce answer sets that correspond to the set of all possible story paths. This interface presents the following challenges to non-programmers:

- Users must be familiar with command-line tools
- Users must know how to use the Clingo solver
- Users must be able to parse and understand the answer sets that Clingo produces

These challenges would be difficult for a non-technical story author to overcome. Therefore, we offer StoryBuilder as a graphical user interface to enhance the usability of the TropICAL language. In addition to removing the barriers to TropICAL's use listed above, it addresses requirements R3 and R7 in the requirements listed in Section 4.2.2 on page 62:

R3. The user should be able to write and re-use existing story components (from a library)

R7. The user should be able to visualise the branches of the story that result from the addition or modification of components to the story.

These requirements suggest the implementation of an IDE (Interactive Development Environment) with visual tools that assist the selection and visualisation of tropes. StoryBuilder is designed to be an IDE that is specifically designed for trope-based story authoring. It takes

the form of a graphical user interface that runs in a web browser, sends snippets of TropICAL code to a server to be compiled and input into Clingo, and visualises the resulting answer sets.

Although libraries can be stored and reused in text-based programming languages through methods such as “import” statements, our intention is to create a graphical interface to simplify the exploration and viewing of tropes in the library. Using StoryBuilder, the user can select a pre-written trope from a dropdown box of existing tropes, which then allows them to see its source code and visualise its structure in the story with a tree diagram.

To address requirement 7, we parse the answer set outputs (traces) of our answer set solver (as described in Section 5.1.1 on page 104) to create a tree diagram of all of the possible events that they describe.

5.1 Interface

StoryBuilder is implemented as a web application with client (browser-based) and server components. The client, developed in the ClojureScript programming language (Hickey, 2017b) and compiled to Javascript, consists of the graphical user interface with which the user creates, selects and visualises tropes. The client passes TropICAL code to the server (written in Clojure (Hickey, 2017a)), which compiles the code to InstAL and runs the resulting AnsProlog code through the *clingo* answer set solver. The resulting answer sets are then passed back to the client as hash map data structures, which are then visualised as graph diagrams.

The following sections provide an overview of StoryBuilder’s user interface, explaining how it can be used for the creation, arrangement and visualisation of tropes.

5.1.1 Overview

Figure 5-1 on the next page shows what the user sees when they first start the *StoryBuilder* tool. On the left side of the screen are two tabs: *edit* and *arrange*. The *edit* tab allows the user to edit existing tropes, or to create a new trope of their own. While editing these tropes, their structures are visualised on the right side of the screen (which is blank in this initial screenshot). Figure 5-2 on the following page shows the same screen as Figure 5-1 on the next page, with the addition of annotations that explain the function of each item in the interface.

The *edit* tab

To edit a trope that has been created previously, the user selects it from a list of tropes in the dropdown box at the bottom-left of the screen. This causes the trope’s text to appear in the text editor that occupies most of the left-hand portion of the screen. From here, the user may amend the text in the editor, and click the blue *save trope* button to make the changes persist. The user may also click the red *delete* button to remove the trope from the database.

If the user selects a trope and then, at some point, clicks the green *refresh* button, a visualisation of its path structure appears on the right side of the screen. An example visualisation appears in figure Figure 5-3 on page 104, with annotations that further explain the features of the interface in the *edit* tab.

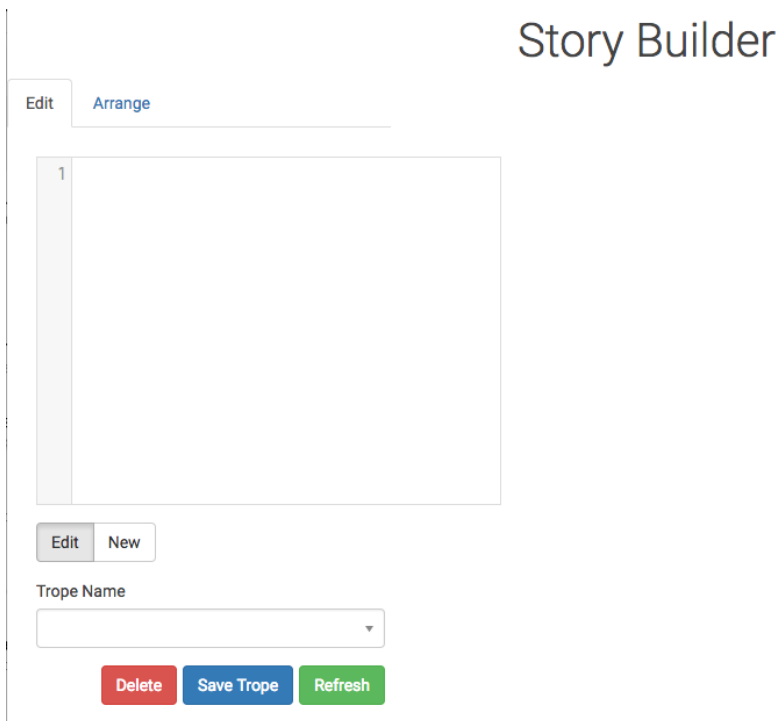


Figure 5-1: The initial view when *StoryBuilder* starts

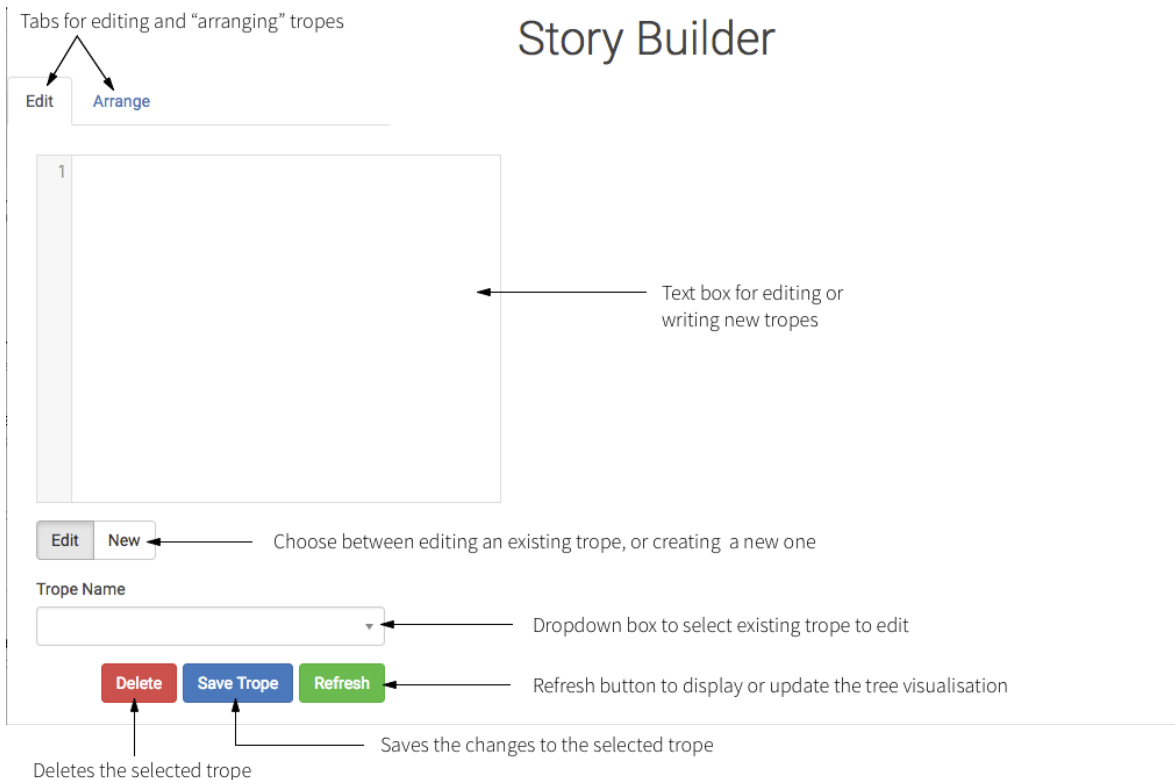


Figure 5-2: An explanation of the *edit* tab

The user also uses the interface in the *edit* tab to create a new trope. Figure 5-4 on page 105 shows the five steps that a user must take in order to create a trope:

Story Builder

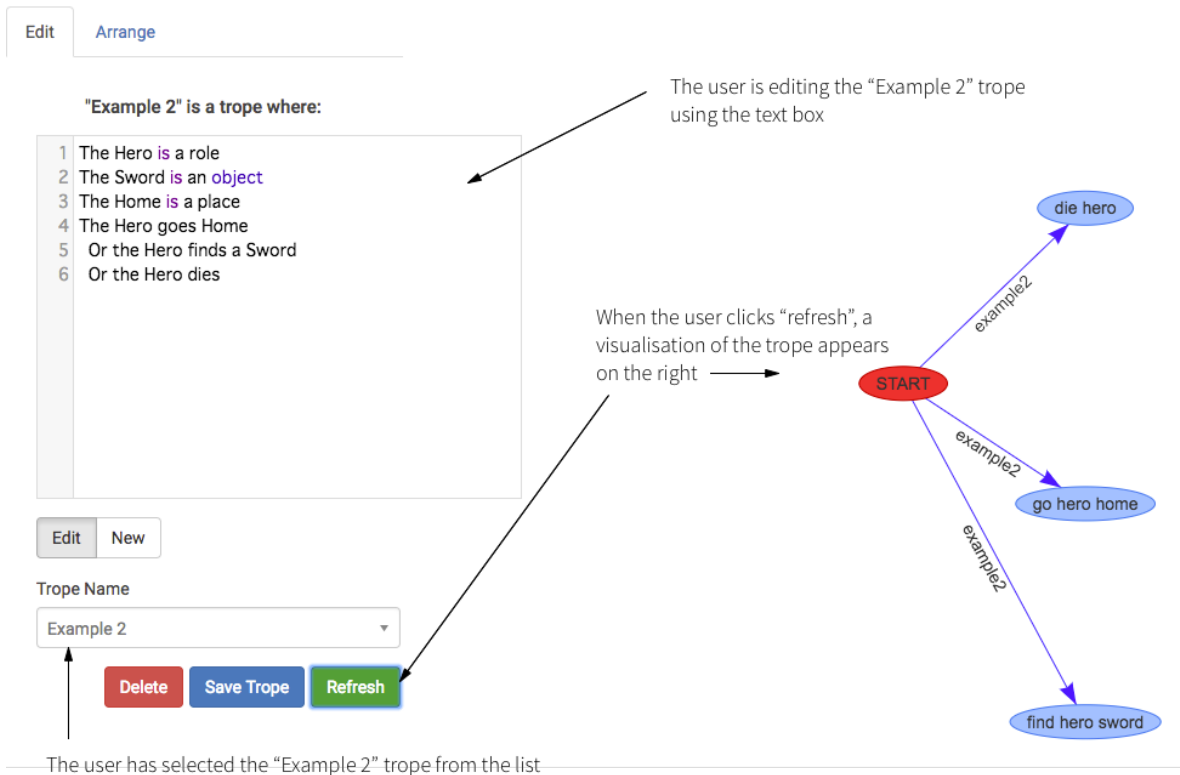


Figure 5-3: An explanation of how the *edit* tab is used

1. The user clicks on the "new" tab below the text area to create a new trope
2. The user types the name of the trope into the text box below the main text area (and below the "new" tab that they have just clicked on).
3. The trope name appears at the top of the interface as the first line of the *TropICAL* trope. For example, for a trope named "Chase Scene", the line above the text area will appear as: "*Chase Scene*" is a trope where:. The system prompts the user if the name is already in use.
4. The user types the *TropICAL* code into the main text area
5. The user clicks on the blue "save" button at the bottom of the screen to save the trope.

Once a trope is created, then the user may return to the *edit* tab to select and visualise it.

Trope Visualisation

The tree visualisation that appears on the right side of the *StoryBuilder* interface is produced from the output of running the *clingo* answer set solver on the *AnsProlog* translation of the trope being edited. The output, which we refer to as *answer sets* or *traces* in Section 4.5 on page 94, are lists of events that are possible in one run through the story. *Clingo* outputs one

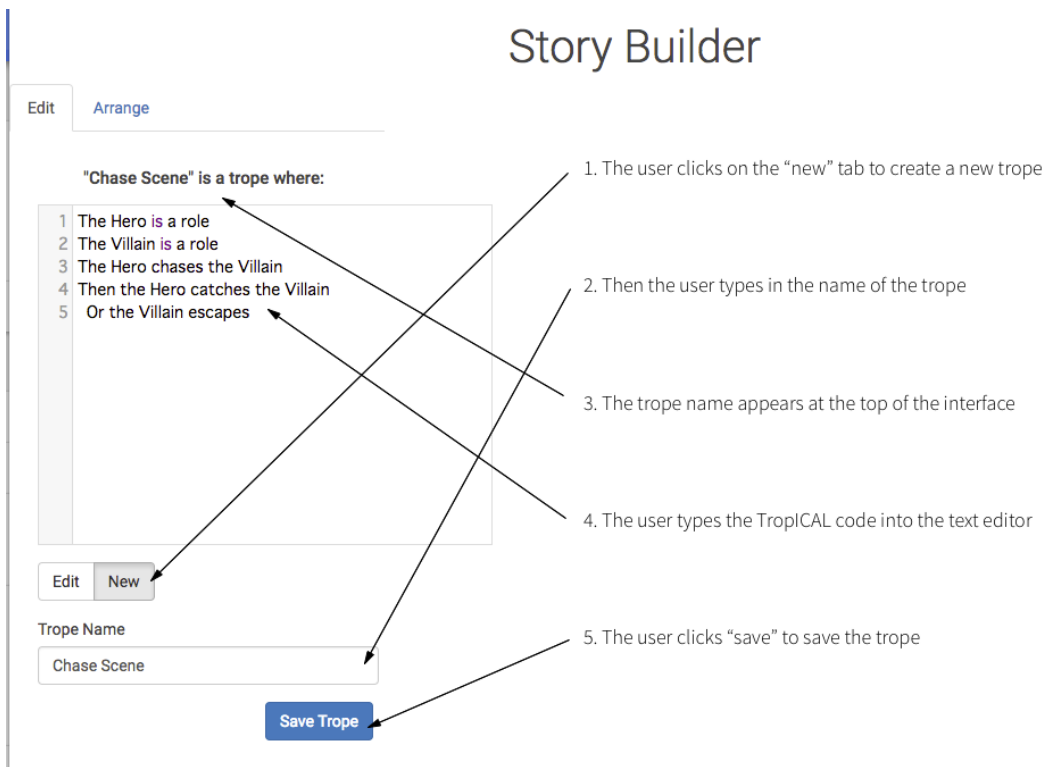


Figure 5-4: An explanation of how a new trope is created

answer set for each possible interpretation of the events that can occur in a story. Examples of these traces are explained further in Section 4.7 on page 95.

These traces are converted into a tree data structure by merging all events that occur at the same time step with the same predecessor events into one node. This eliminates redundant paths in the visualisation, reducing the visual clutter and making it easier for the author to understand. This data structure is used as the input to *vis.js*¹, a Javascript library for graph visualisation. This library draws the tree on an interactive canvas on screen, allowing the user to drag the nodes around and zoom in on them.

In the visualisation, each node is labelled with an event in *verb_subject_object* prefix notation. For example, a node corresponding to “The Hero goes to the Land of Adventure” would be labelled as *“go_hero_landOfAdventure”*. The reason for this labelling is to reduce the verbosity of the original controlled natural language statement in order to convey the essential information inside a graph node. The edges between each node are labelled with the name of the trope that corresponds to the event with the later time step. For example, if Event A is an *Evil Empire* trope event and is followed by Event B, a *Hero’s Journey* event, the edge linking the nodes together will be labelled with Event B’s trope (*The Hero’s Journey*). In order to allow for easier visual identification of the different story paths that different tropes allow, the edge for each trope has a unique colour, with Trope A’s lines appearing in blue, Trope B’s being red, and Trope C’s being green, for example.

¹Accessible at the following website: <http://visjs.org>, accessed 20170826.

The *arrange* tab

The *arrange* tab allows the user to combine multiple tropes into a story, and to visualise all the possible paths through that story.

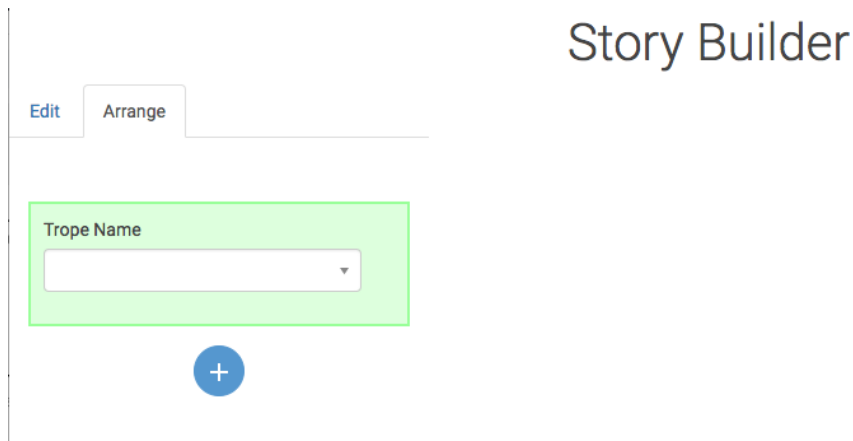


Figure 5-5: The view when the user clicks the *arrange* tab

Figure 5-5 shows the view that the user sees when the *arrange* tab is first clicked. The screen appears mostly blank, with the exception of a green rectangle on the left side of the screen, where the user may select from a list of pre-written tropes. The user may select extra tropes by clicking on the blue plus (“+”) symbol below the green box.

Once one or more tropes are selected, a tree visualisation appears on the right side of the screen to show the combination of paths that a story may take, based on the selected tropes. Figure 5-6 on the next page shows the steps that a user takes to combine tropes together in the interface, along with a part of the visualisation that appears as a result of selecting two example tropes. In order to combine multiple tropes together in this way, a user must go through these steps:

1. The user clicks on the “arrange” tab at the top-left of the screen
2. The user selects a trope from the dropdown box inside the green rectangle
3. The user waits for the resulting visualisation to appear
4. The user clicks on the blue “+” symbol below the selected trope on the left
5. The user selects another trope from the dropdown box that has just appeared
6. The user waits for the resulting visualisation to appear

A particularly important feature of the visualisation is that each graph edge is labelled with the name of the trope that links two events. In the example shown in Figure 5-6 on the following page, events that follow the *Example 1* trope are connected with a red arrow, and events that follow the *Item Search* trope are connected with a blue arrow. This allows the user to see the effect adding a certain trope has on the potential paths that their story may take.

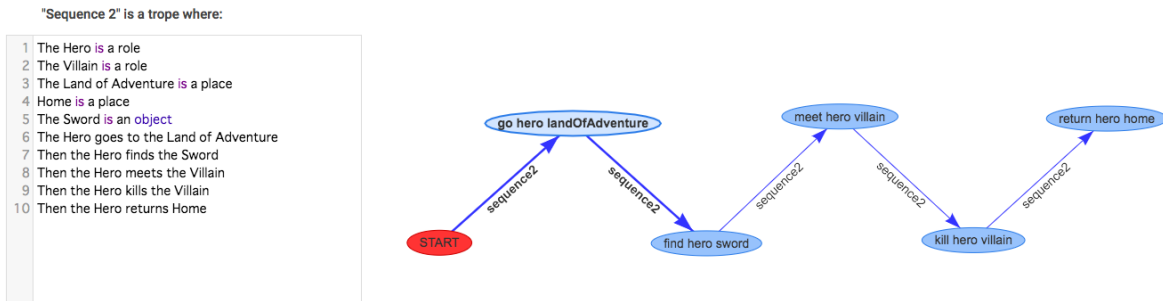


Figure 5-8: A sequence of events (from Listing 4.8 on page 65)

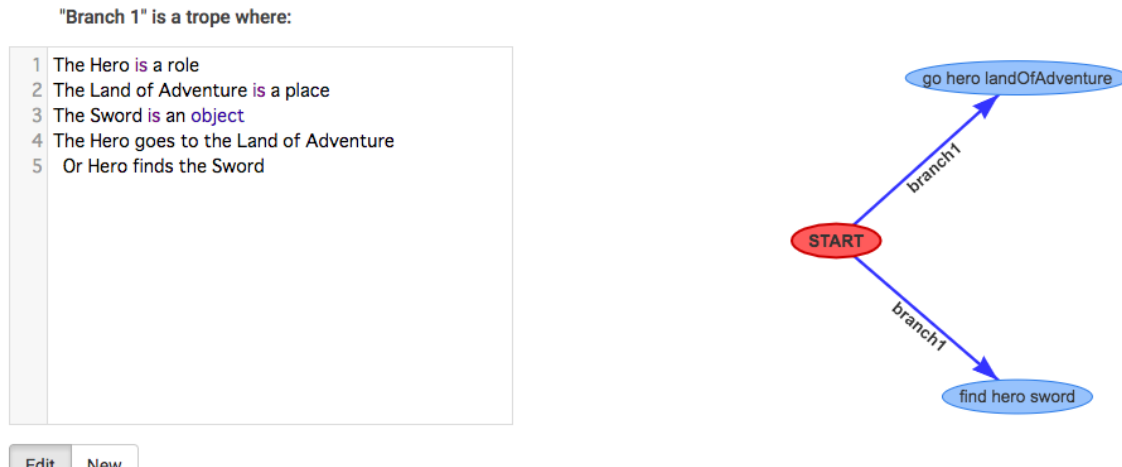


Figure 5-9: Two branches (from Listing 4.11 on page 66)

5.1.2 Usage Examples

The figures in this section show how the simple trope examples listed in Section 4.3.2 on page 64 to Section 4.3.5 on page 68 appear when edited and visualised using the *StoryBuilder* tool.

To demonstrate the visualisation of sequences of events, figure Figure 5-7 on the previous page shows a sequence just two events long, with the code from Listing 4.6 on page 64 as input. Figure 5-8 uses the longer sequence of five events from Listing 4.8 on page 65 as its input.

StoryBuilder's visualisation of branches are demonstrated in the figures that follow. Figure 5-9 shows the simple two-branch trope of listing Listing 4.11 on page 66. Figure 5-10 on the next page shows the visualisation of a trope with five different branches, as described by the trope in listing Listing 4.12 on page 66. Finally, the more complicated combination of sequences and branches from Listing 4.13 on page 67 is shown in Figure 5-11 on page 110.

The final two figures in this section demonstrate the use of *StoryBuilder* for embedding subtropes inside of other tropes. Figure 5-12 on page 110 is the *StoryBuilder* visualisation that corresponds to the subtrope that is to be embedded (described in Listing 4.16 on page 69). The actual embedding of this subtrope, as described in Listing 4.17 on page 69, and its resulting visualisation are shown in Figure 5-13 on page 111.

Edit
Arrange

"Branch 2" is a trope where:

- 1 The Hero **is** a role
- 2 The Villain **is** a role
- 3 The Land of Adventure **is** a place
- 4 Home **is** a place
- 5 The Sword **is** an object
- 6 The Hero goes to the Land of Adventure
- 7 Or the Hero finds the Sword
- 8 Or the Hero meets the Villain
- 9 Or the Hero kills the Villain
- 10 Or the Hero returns Home

Edit
New

Trope Name

Branch 2

Delete
Save Trope
Refresh

```

graph TD
    START([START]) -- branch2 --> A([find hero sword])
    START -- branch2 --> B([return hero home])
    START -- branch2 --> C([go hero landOfAdventure])
    START -- branch2 --> D([meet hero villain])
    START -- branch2 --> E([kill hero villain])
  
```

Figure 5-10: Five branches (from Listing 4.12 on page 66)

5.1.3 Design Justification

In the introduction to this chapter, we describe how the StoryBuilder tool addresses requirements R3 and R7 in the requirements shown in Section 4.2.2 on page 62:

- R3. The user should be able to write and re-use existing story components (from a library)
- R7. The user should be able to visualise the branches of the story that result in the addition or modification of components to the story.

StoryBuilder fulfills requirement R3 by storing every trope that the author creates in a database, so that they may be reused and amended if needed. Requirement R7 is met through StoryBuilder's use of a tree visualisation that updates as the story author is editing the tropes in the tool.

5.1.4 Creating a Punch and Judy story with StoryBuilder

From this point in the thesis, we use the *Punch and Judy* story world for our example narratives. *Punch and Judy* is a traditional British-Italian puppet show often seen at seaside resorts. Its first recorded appearance in the UK is in Covent Garden, London in 1662, having

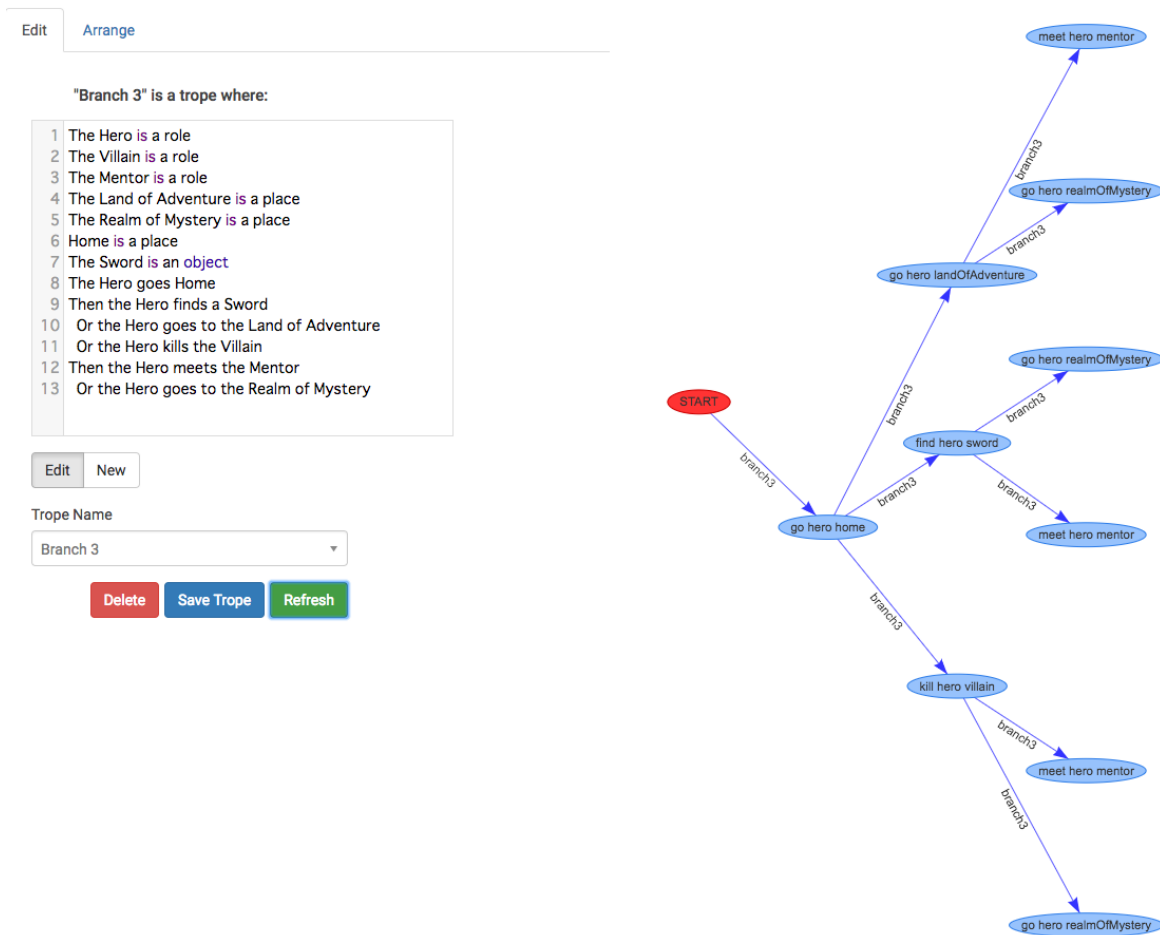


Figure 5-11: A combination of branches and sequences (from Listing 4.13 on page 67)

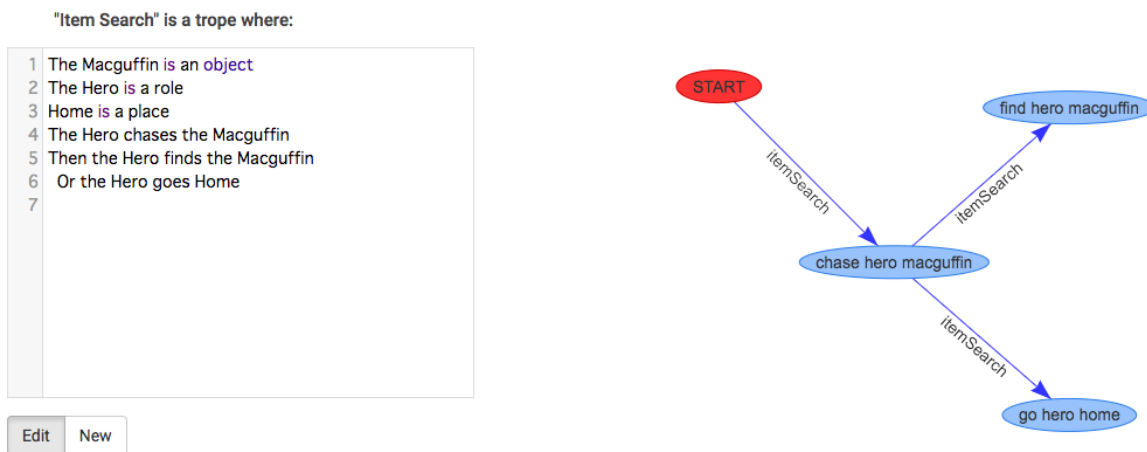


Figure 5-12: Subtrope to be embedded (from Listing 4.16 on page 69)

been brought into the country by Italian puppeteer Pietro Gimonde. Since then it has become a British cultural institution, delighting children and adults alike over the course of centuries.

The show itself is an incredibly violent farce, where the titular character *Punch* is a homicidal maniac with a hair-trigger temper. Most scenes start with him having an argument with

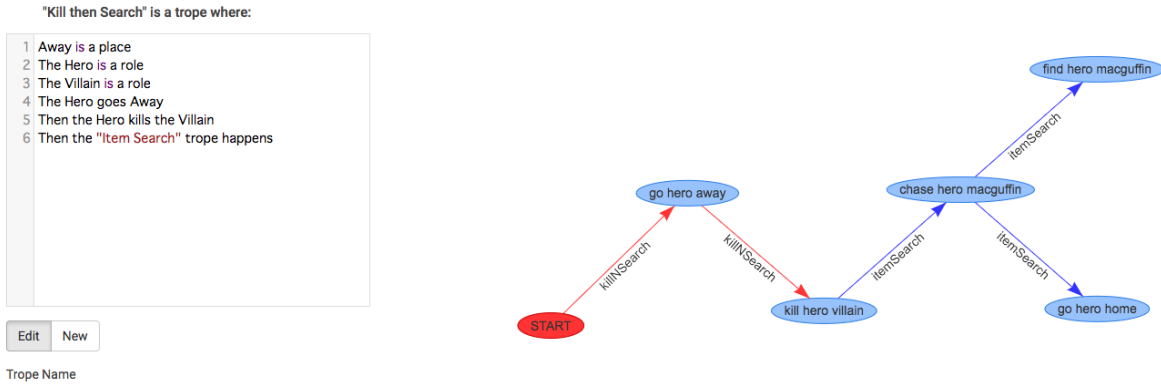


Figure 5-13: Trope containing a subtrope (from Listing 4.17 on page 69)

another puppet, and end with Punch beating the other puppet to death. He starts by killing his wife and child, then the policeman that comes to arrest him, before finally murdering the Devil himself.

Interestingly, there is no comeuppance for Punch's killing spree, as most modern audiences would expect. Instead, the story shows Punch as a buffoon, a ridiculous caricature of an angry man who blunders his way through life, yet is victorious nonetheless. The story has no moral message, only farcical violence.

Despite its arguable nihilism, *Punch and Judy* has many features that make it a suitable narrative domain for the purposes of testing our research:

- It has many recognisable patterns, or *tropes*. For example, most scenes feature some kind of audience participation, and most end with slapstick comedy.
- It can be as simple or sophisticated as we choose to make it. Some scenes are relatively simple to model, such as the scene where Punch beats the policeman. Others are more difficult, such as where Punch is asked to guard some suasages, but then loses them to a crocodile. Combining several scenes into a complete and recognisable *Punch and Judy* narrative is, we contend, a worthwhile challenge for a narrative generation system.
- Following from this, its scenes can either be self-contained and modelled on their own, or can depend on the context of the previous scenes. For example, Punch's foes increase in importance as his rampage progresses. In some versions, he kills his wife, then a policeman, then the executioner sent to hang him, then finally the Devil. This makes it useful for modelling a narrative with a state that changes over time.
- Most (British) people know *Punch and Judy* when they see it. They are able to recognise how characters in the story are supposed to behave, and what should happen in each scene and in the story as a whole. This helps with evaluation of the system: the audience will be able to say whether or not the generated tropes are *authentic* or not.

The minimal set of *Punch and Judy* characters would be Punch, his wife Judy, their baby, Joey the clown (who narrates the story and interacts with the audience, and is immune to

Listing 5.1: The “Audience participation” trope

```

1 ``Audience Participation'' is a trope where:
2 The Character is a role
3 The Audience is a role
4 The Character shouts at the Audience
5 Then the Audience shouts at the Character

```

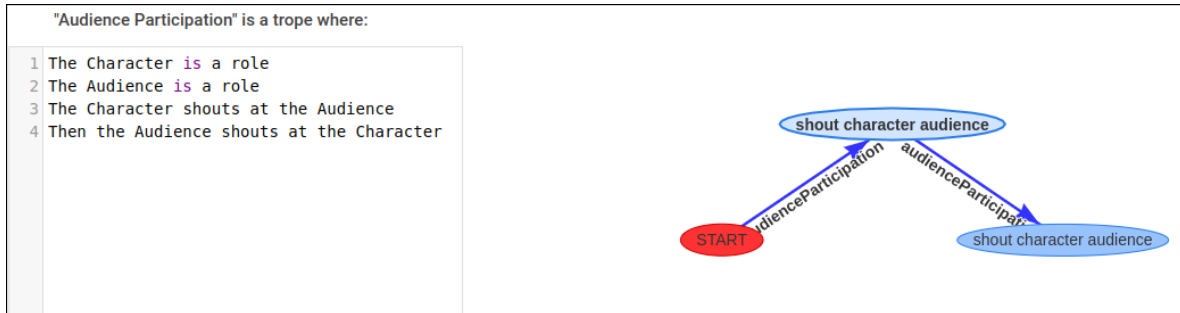


Figure 5-14: The “Audience Participation” trope in StoryBuilder

Punch’s attacks), a crocodile and a Policeman. Extended versions include other animals that Punch chases (such as a monkey or cat), and the previously mentioned hangman and Devil characters. These are not needed for a full Punch and Judy show, however. Indeed, many performers show just one or two scenes in a show, with the first featuring Punch, Judy and the baby, and then finishing with the scene where Punch wrestles with a crocodile for some sausages. The tropes that appear in Punch and Judy can be divided into two types: those that are present throughout the story, and those that describe particular scenes of the story. One trope that is a constant feature of Punch and Judy shows is audience participation, where the characters shouts something at the audience with the expectation of a response. For example, the Punch character may shout something like: “Where is he?”, with the traditional audience response being “He’s behind you!”.

There are also particular scenes that always feature in a Punch and Judy show. The two scenes that we will describe are the “Babysitting scene”, where Judy leaves Punch with the Baby, and the “Crocodile Sausages” scene, where a crocodile eats some sausages that Punch has been trusted with looking after.

Listing 5.1 shows a basic description of the events that happen in audience participation: a character calls out to the audience, then the audience calls back. When integrated into a multi-agent system, each character agent will be assigned the “character” role, so that the social norms from this trope will apply to them. Figure 5-14 shows how this trope appears when it is input into StoryBuilder.

Because this trope is present throughout the Punch and Judy story, it can be either inserted as a subtrope into other tropes, or (perhaps more effectively) passed to the answer set solver along with the other tropes in the story, so that its events can occur at any point. This is done via the “arrange” tab in the user interface, described in Section 5.1.5 on page 115 below.

Listing 5.2 on the next page shows a trope to describe the “Babysitting” scene from Punch and Judy. We have decided to put a branching point in this trope, where Punch may either kill

Listing 5.2: The “Babysitting” trope

```

1  ``Babysitting'' is a trope where:
2  Punch is a role
3  Judy is a role
4  The Baby is a role
5  Away is a place
6  Home is a place
7  Judy kisses Punch
8  Then Judy goes Away
9  Then Punch chases the Baby
10 Then Punch kills the Baby
11   Or the Baby escapes
12 Then Judy returns Home

```

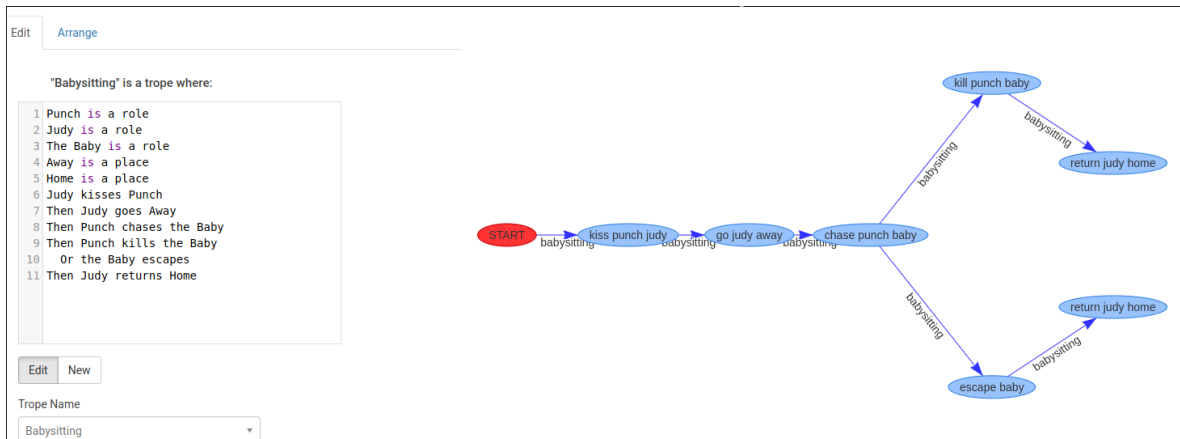


Figure 5-15: The “Babysitting” trope in StoryBuilder

the baby, or the baby escapes. This gives the option of a more “family-friendly” alternative of the scene, suitable for younger audiences. The StoryBuilder visualisation appears in Figure 5-15.

Listing 5.3 on the next page shows the TropICAL definition of the “Crocodile Sausages” scene in Punch and Judy. This scene is a simple sequence of events, but we will see how to interpose events from other tropes between them below. Its StoryBuilder visualisation appears in Figure 5-16 on the following page.

However, these tropes can be further simplified and modularised. Both the “Babysitting” and “Crocodile Sausages” feature events where a character leaves the stage, only to return later. With this knowledge, we can abstract these events into the “Absentation” trope, shown in listing Listing 5.4.

Listing 5.4: An “Absentation” trope

```

1  ``Absentation'' is a trope where:
2  Absentee is a role
3  Home is a place
4  Away is a place
5  The Absentee goes Away
6  Then the Absentee returns Home

```

Similarly, many Punch and Judy scenes end with two characters chasing each other, such

Listing 5.3: The “Crocodile Sausages” trope

```

1  ``Crocodile Sausages'' is a trope where:
2  Punch is a role
3  Joey is a role
4  Crocodile is a role
5  The Sausages are an object
6  Away is a place
7  Home is a place
8  Joey meets Punch
9  Then Joey leaves behind the Sausages
10 Then Joey goes Away
11 Then the Crocodile appears
12 Then the Crocodile chases Punch
13 Then the Crocodile eats the Sausages
14 Then Joey returns Home

```

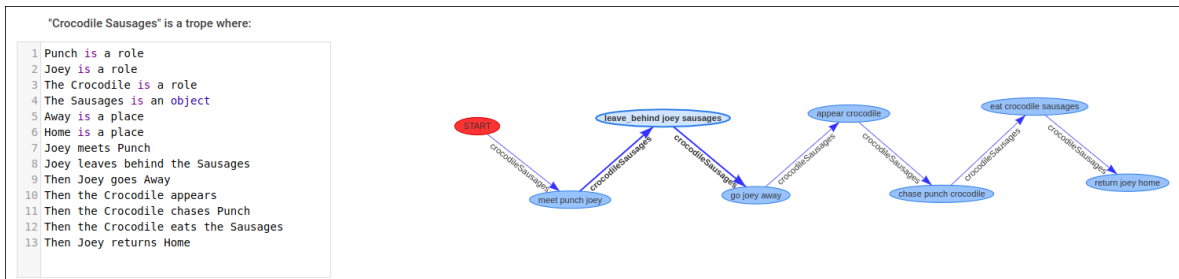


Figure 5-16: The “Crocodile Sausages” trope in StoryBuilder

as when the Crocodile chases Punch, or the scene where the Policeman chases Punch. From this, we can create the “Chase Scene” trope of Listing 5.5.

Listing 5.5: A “Chase Scene” trope

```

1  ``Chase Scene'' is a trope where:
2  Pursuer is a role
3  Pursued is a role
4  The Pursuer chases the Pursued
5  Then the Pursuer catches the Pursued
6  Or the Pursued escapes

```

Creating these new abstractions allows us to rewrite our tropes without repeating ourselves. Listing 5.6 on the next page shows how the “Babysitting” trope would be rewritten by incorporating the “Absentation” and “Chase Scene” tropes as subtropes. Listing 5.7 on the following page does the same for the “Sausages” trope.

Listing 5.6: The “Babysitting” trope, redefined with a subtrope

```
1  ``Babysitting'' is a trope where:
2    Punch is a role
3    Judy is a role
4    The Baby is a role
5    Away is a place
6    Home is a place
7    Judy kisses Punch
8    Then the ``Absentation'' trope happens
9    Then the ``Chase Scene'' trope happens
```

Listing 5.7: The “Sausages” trope, redefined with a subtrope

```
1  ``Sausages'' is a trope where:
2    Punch is a role
3    Joey is a role
4    Crocodile is a role
5    The Sausages are an object
6    Away is a place
7    Home is a place
8    Joey gives the Sausages to Punch
9    Then the ``Absentation'' trope happens
10   Then the Crocodile appears
11   Then the Crocodile chases Punch
12   Then the Crocodile eats the Sausages
13   Then Joey returns Home
```

5.1.5 Combining Tropes Together

Sometimes we want to combine two or more tropes together into a story, so that the characters are free to choose to follow the events of any of the tropes that may be happening. In this example, we describe a scene where the events of the “Babysitting” scene are combined with the “Audience Participation” trope. This would mean that both tropes are active at the same time, allowing the characters to perform the actions described in both tropes.

Using the “arrange” tab in StoryBuilder, we select both the “Audience Participation” and “Babysitting” tropes. The tree visualisation that appears on the right of the screen shows us all of the permutations that appear as the output of the answer set solver. Figure 5-17 on the next page shows the result of doing this in StoryBuilder. The tree visualisation on the right has been enlarged so that its nodes are legible. The nodes of the tree describe the events that are occurring, and the links between nodes are labelled with the name of the trope that their following nodes (events) correspond to.

This has the effect of “mixing” the events of the two tropes together. Example sequences of events in a scene with both the “Babysitting” and “Audience Participation” tropes would be:

1. Judy kisses Punch (*Babysitting*)

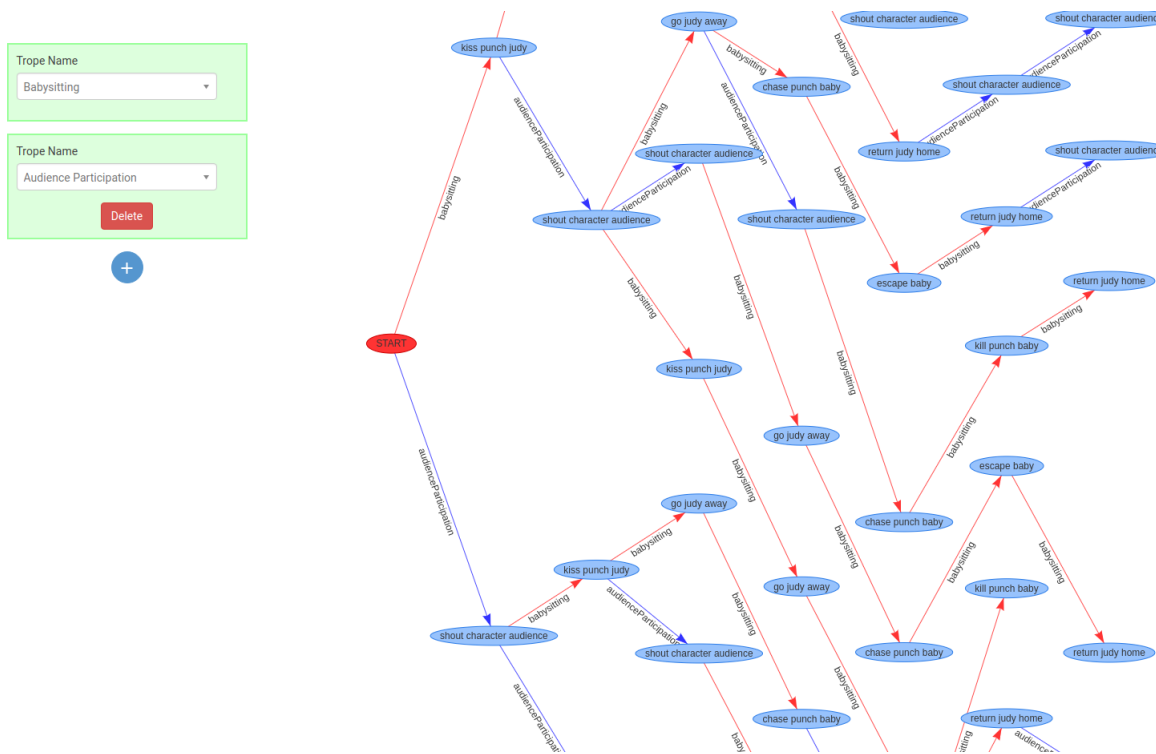


Figure 5-17: Combining the “Babysitting” and “Audience Participation” tropes in StoryBuilder

2. A Character (Judy or Punch) shouts at the Audience (*Audience Participation*)
3. Then Judy goes away (*Babysitting / Absentation*)
4. Then Punch chases the Baby (*Babysitting / Chase Scene*)
5. The Audience shouts at a Character (Punch) (*Audience Participation*)
6. Then the Baby escapes (*Chase Scene*)
7. Then Judy returns Home (*Babysitting / Absentation*)

This corresponds to one of the permutations of combining the events of the “Babysitting” and “Audience Participation” tropes in different orders, and would be just one of the many paths through the visualised tree.

Using this part of the StoryBuilder tool, the user is able to interactively explore the different combinations of events that different tropes add to their story. When these combinations of tropes are visualised, they are then able to refine their tropes further in order to build the non-linear narrative that they desire.

Further discussion on trope combination appears in Section 7.2.10 on page 164.

Chapter 6

Using TropICAL with a Normative Multi-Agent System

This chapter describes the integration of TropICAL into a multi-agent system consisting of multiple intelligent agents with character models. Using the *Punch and Judy* story world as our initial example, we formalise its narrative rules by describing them in terms of tropes. We then demonstrate how these tropes can then be expressed as social institutions to govern the intelligent agents in a multi-agent system.

This chapter begins by describing a scene from “Punch and Judy” in terms of story tropes in Section 6.1. These tropes are then translated into institutions through their expression in terms of norms (Section 6.1.2 on page 122). Section 6.2 on page 125 describes an implementation of this scene using the InstAL (Cliffe et al., 2007) language for institutions combined with agents with emotional models written using the Jason (Bordini et al., 2007) framework. The chapter concludes with a summary in Section 6.3 on page 132.

6.1 Punch and Judy as Tropes

We begin this section by building up a scene description in terms of tropes. Then we create an institution out of the scene we have described with our formalism.

On the TV Tropes page for Punch and Judy (TV Tropes, 2017c), the following tropes are listed (quoted directly from the site):

- **Amusing Injuries:** People are often beaten up.
- **Audience Participation:** The children are expected to reply to Mr. Punch’s Catch Phrase, “That’s the way to do it” with a shout of “Oh no, it isn’t!”
- **Black Comedy:** So black that many modern versions are often heavily censored compared to more historical stagings.
- **Catch Phrase:** “That’s the way to do it!”, “HE’S BEHIND YOU!”, etc.
- **Comedic Sociopath:** Mr. Punch

- **Commedia dell'Arte:** Punch is based on the *Pulcinella* character.
- **Crosses the Line Twice:** Good showings will definitely do this.
- **Hand Puppet:** All of the characters, except the baby, are puppets (though originally marionettes).
- **Head Bob:** Traditionally the puppets don't have articulated mouths, and use head bobbing to indicate which one is speaking.
- **Ironic Echo:** There's at least one rendition of the act where Punch ends up playing one trick too many on Snap the Crocodile, who promptly eats him (off-stage, of course) and returns repeating Mr Punch's "da-da-da" sound, culminating in a mock belch.
- **Karma Houdini:** In many versions, Punch is a psychopath who kills his own baby by throwing it out of a window, beats his wife to death with a stick, kills several other characters whom he encounters and finally outwits the devil himself to get away completely scot free.
- **Refuge in Audacity:** The entire show, especially the violence, is played as outrageous comedy.
- **Slapstick:** The style of the show, even named after the type of stick Punch uses.
- **Throw the Dog a Bone:** In some shows Judy will get her hands on Punch's stick and beat him with it. Though this is usually followed by Punch snatching it back and beating her with it.
- **Unsympathetic Comedy Protagonist:** Mr. Punch
- **Values Dissonance** Possibly the best example of this as Punch's domestic abuse of Judy is completely played for laughs.
- **Villain Protagonist:** Mr. Punch

Some of these tropes are useful in our construction of the story as a whole in terms of tropes, while others are too general or abstract to express as agent behaviours. For example, the *Commedia dell'Arte* trope would be difficult to express in a way that would influence our interactive narrative. Instead of describing the entire story of Punch and Judy, we will describe a single scene in terms of tropes, using one piece of the story as an illustrative example.

The common elements of Punch and Judy are easily described in terms of Propp's story functions. As our example, we use the scene where Punch battles a Crocodile in order to safeguard some sausages (which we refer to as the "Sausages Scene"). In this scene, Joey the clown (our narrator) asks Punch to guard the sausages. Once Joey has left the stage, a Crocodile appears and eats the sausages. Punch fights with the Crocodile, but it escapes. Joey then returns to find that his sausages are gone.

The corresponding Propp story functions are:

1. Joey tells Punch to look after the sausages (*interdiction*).
2. Joey has some reservations, but decides to trust Punch (*complicity*).
3. Joey gives the sausages to Punch (*provision or receipt of a magical agent*).
4. Joey leaves the stage (*absentation*).
5. A Crocodile enters the stage and eats the sausages (*violation*).
6. Punch fights with the Crocodile (*struggle*).
7. Joey returns to find that the sausages are gone (*return*).

Some story functions map to Punch and Judy better than others (for example, it is debatable as to whether or not the sausages can be considered a “magical agent”), but Propp’s formalism seems well suited to Punch and Judy for the most part. The advantage of using Propp for the Punch and Judy story domain is that the story function concept maps well to the idea of internal events in institutional models.

To help with this translation from Propp to tropes, we refer to the TV Tropes page on Vladimir Propp, which maps his story functions directly into tropes (TV Tropes, 2017d):

1. Joey tells Punch to look after the sausages (*Rule Number One*).
2. Joey has some reservations, but decides to trust Punch (*Deal with the Devil*).
3. Joey gives the sausages to Punch (*Mentor Archetype*).
4. Joey leaves the stage (*Parental Abandonment*).
5. A Crocodile enters the stage and eats the sausages (*Don’t Touch It, You Idiot!*).
6. Punch fights with the Crocodile (*Earn Your Happy Ending*).
7. Joey returns to find that the sausages are gone (*Where It All Began*).

Here are the tropes mentioned above, described in more detail:

Rule Number One (interdiction)

TV Tropes actually describes two separate versions of this trope:

- a situation where a character makes rules to govern a dangerous or uncomfortable situation (one such example being “The first rule of Fight Club...”, which is in itself a trope of its own).
- when a Mentor Archetype conveys advice or admonishments to another character, such as “Don’t use the dangerous forbidden technique!” or “Always believe in yourself!”.

In the case of our scene, the interdiction seems to be the second type listed here.

Deal with the Devil (complicity)

The classic incarnation of this trope is the 16th century legend of Faust selling his soul to Mephistopheles. It involves a desperate pawn (Faust) signing a magically binding contract with a corrupt, exploitative trickster (Mephistopheles, or any Satan-like character).

In this scene, Punch would be the corrupt exploiter, with Joey the Clown as his pawn.

Mentor Archetype (Receipt / provision of a Magical Agent)

TV Tropes describes this as “A more experienced advisor or confidante to a young, inexperienced character, particularly to a hero.”

Due to our stretching of the original Propp function definition, this trope does not fit what we want to express: the simple act of Joey giving the sausages to Punch. Joey does not really fulfil the Mentor role. Additionally, TV Tropes’ translation of “Receipt of a Magical Agent” from Propp to “Mentor Archetype” is questionable: there is no mention of a Magical Agent in this trope, only of the Mentor who provides it. We must look for a better-fitting trope in this case.

Parental Abandonment (absentation)

This trope is straightforward: the protagonist is abandoned by their parents (emotionally or physically). In the case of the trope, it is described as something that drives or influences the protagonist, such as in the case of the character Bruce Wayne: the death of his parents early on drove him to become the superhero Batman. This differs from Propp’s function (and our Punch and Judy example), in which the absence of parental or supervisory characters leads to mischief from the protagonist, and the violation of the earlier interdiction. This trope appears to be defined flexibly enough to fit this interpretation as well, however.

Don’t Touch It, You Idiot! (violation)

The title of this trope does not entirely convey the nuance of its meaning: TV Tropes defines it as any order or interdiction that is inevitably violated at some point later in the story. This actually fits well with Propp’s original definition, which stated that the “interdiction” and “violation” story functions must always go together, as one inevitably leads to the other.

Earn Your Happy Ending (struggle)

This trope states that the characters in a story must face far more difficulty than usual, overcoming more obstacles than most characters would have to face. However, the characters get a happy ending as a result of their struggles.

Again, this does not fit our scene where Punch fights the crocodile for the sausages. Though it does describe a struggle of sorts, it is more of a comedy fight than anything arduous for the characters involved. We can probably find a better match for this trope.

Where It All Began (return)

This trope does not match the definition of “return” that we have used from Propp: in our case, it describes the return of a supervisory character some time after they went away during

the *absentation* function. TV Tropes' definition describes more the return of the protagonist to their hometown at the end of the *Hero's Journey*. In this case, we need to find a trope that is the counterpart to the "Parental Abandonment" function from earlier, which describes the return of the "parents" of that particular trope. The problem is the slight mismatch in the definition of the trope against the Propp function we are using. In the most literal case of the trope, the "parents" cannot return: they are dead. Again, this indicates that perhaps we must find tropes that more closely match Propp's definitions of *absentation* and *return*.

6.1.1 Issues with Propp Mapping

From our deeper analysis of TV Tropes' mappings of Propp story functions to tropes, a number of issues have arisen:

- Our use of Propp's story functions may be a little too "flexible". This means that the mappings of the functions we have used add an extra layer of interpretation to the translation, taking us away from the original intended meaning.
- Tropes, by their very nature, are a little ambiguous and open to interpretation. The same trope could even be expressed in multiple different ways, such as the "Rule Number One" trope.

In place of TV Trope's suggested "struggle" trope, *Earn Your Happy Ending*, a more suitable trope to use would be the *Chase Fight*:

Chase Fight: An X meets Y cross between a Chase Scene and a Fight Scene.

This is far more suited to our purposes, as the scene simply consists of the crocodile fighting Punch by chasing him around the stage.

Similarly, in the place of the "absentation" and "return" tropes, *Parental Abandonment* and *Where It All Began*, TV Tropes has a more suitable pair to use:

Put on a Bus: a character is written out of a story so that they may (possibly) return later.

The Bus Came Back: when one of the (main) characters returns back into the story.

Though this trope pair better captures the essence of the leaving and return of Joey, what if we also wanted some of the nuance of the *Parental Abandonment* trope? The beauty of the capturing tropes as institutions is that we can use both sets of tropes and let the player and character agents decide the outcome, which could be a set of actions from a mixture of both tropes.

For simplicity, we can remove the "Deal With The Devil" trope, as well as "Rule Number One". The "Don't Touch It, You Idiot" trope includes the interdiction that we wanted to express through the "Rule Number One" trope. Also, as the "Deal With The Devil" trope

involves a lot more than the simple complicity with which Joey goes along with Punch's plans, it can be safely omitted.

This leaves us with just four tropes that describe our scene:

- **Don't Touch It, You Idiot:** Joey gives the sausages to Punch, telling him not to eat them
- **Put on a Bus:** Joey temporarily leaves the stage
- **Chase Fight:** The Crocodile appears and chases Punch around the stage
- **The Bus Came Back:** Joey returns to the stage to scold Punch

Trope Roles

The character roles that appear in trope descriptions in TV Tropes differ greatly from the *Dramatis Personae* defined in Propp's morphology. For each of the four tropes, we identify the following roles:

- **Don't Touch It, You Idiot:** the *owner* (Joey) and the *idiot* (Punch)
- **Put on a Bus:** the *absentee* (Joey)
- **Chase Fight:** the *chaser* (Crocodile) and the *pursued* (Punch)
- **The Bus Came Back:** the *returnee* (Joey)

Clearly, this means that each character must adopt multiple roles throughout the course of a narrative. In this scene alone, Joey the Clown plays the roles of *owner*, *absentee* and *returnee*. Punch himself must be an *idiot* and the *pursued*.

These roles are not strictly defined in the descriptions found in TV Tropes, and must be inferred by the reader. Furthermore, these roles do not describe character archetypes such as *Hero* or *Comedic Sociopath*. One interesting way to approach this issue could be to have an archetype inherit certain character roles. For example, a *Comedic Sociopath* could automatically fill the roles of *murderer*, *idiot*, *pursuer*, and *chaser*. This idea, while promising, is outside the scope of this thesis, and so is discussed further only in the "future work" Section 9.1 on page 180.

6.1.2 The Sausages Scene

This section describes the translation of the *Don't Touch It, You Idiot* trope from Punch and Judy into a formal institution.

First, we define the sequence of events that form the trope:

- The *owner* tells the *idiot* to protect an *item*
- Then the *owner* goes away

- Then the *idiot* breaks the *item*
- Then the *owner* returns
- Then the *owner* fights the *idiot*

This is the simplest possible interpretation of the trope. It is possible for other interpretations to exist: for example, rather than the *owner* returning and fighting the *idiot*, something bad might happen to the *idiot* instead. In our TropICAL language, alternative outcomes can be expressed using the *or* operator (Section 4.3.4 on page 66) allowing the creation of more flexible tropes which cater for branching story lines.

Translating a simple version of this trope into TropICAL code produces the code in Listing 6.1.

Listing 6.1: The “Don’t Touch it, You Idiot!” trope in TropICAL

```

1 "Dont Touch it You Idiot" is a trope where:
2   The Owner is a role
3   The Idiot is a role
4   The Item is an object
5   Away is a place
6   The Owner drops the Item
7   Then the Owner goes Away
8   Then the Idiot breaks the Item
9   Then the Owner returns
10  Then the Owner fights the Idiot

```

The following listing shows the domain fluents in the institution for the above TropICAL code, which include all the actions of the characters, as well as their roles:

$$D = \{\text{owner, idiot, item, away, inactive, phaseA, phaseB, phaseC, phaseD, phaseE, done}\}$$

Also based on the above sequence events is the list of actions that may be permitted to occur within the trope:

$$P = \{\text{perm(drop(owner, item)), perm(go(owner, away)), perm(break(idiot, item)), perm(return(owner)), perm(fight(owner, idiot))}\}$$

Trope Phases

Arranging these four tropes in a linear sequence describes the *sausages* scene for the most part, though our use of the *Don’t Touch It, You Idiot* trope in place of both of Propp’s *interdiction* and *violation* story functions introduces a need to have two different *phases*, as explained in Section 4.4.7 on page 87. The first phase is triggered by one character warning another character not to do something, the second is when the warned character performs the forbidden action.

$$\mathcal{C}^\uparrow(\mathcal{X}, \mathcal{E}) : \left\{ \begin{array}{l}
\langle \{ \text{phase}(\text{dontTouchItYouIdiot}, \text{inactive}) \}, \\
\text{intDontTouchItYouIdiot}(\text{owner}, \text{idiot}, \text{item}, \text{away}) \rangle \rightarrow \\
\quad \{ \text{phase}(\text{dontTouchItYouIdiot}, \text{phaseA}), \\
\quad \text{perm}(\text{drop}(\text{owner}, \text{item})) \} \quad (6.1) \\
\langle \{ \text{phase}(\text{dontTouchItYouIdiot}, \text{phaseA}) \}, \\
\text{intDontTouchItYouIdiot}(\text{owner}, \text{idiot}, \text{item}, \text{away}) \rangle \rightarrow \\
\quad \{ \text{phase}(\text{dontTouchItYouIdiot}, \text{phaseB}), \\
\quad \text{perm}(\text{go}(\text{owner}, \text{away})) \} \quad (6.2) \\
\langle \{ \text{phase}(\text{dontTouchItYouIdiot}, \text{phaseB}) \}, \\
\text{intDontTouchItYouIdiot}(\text{owner}, \text{idiot}, \text{item}, \text{away}) \rangle \rightarrow \\
\quad \{ \text{phase}(\text{dontTouchItYouIdiot}, \text{done}) \} \quad (6.3)
\end{array} \right.$$

Figure 6-1: Phase fluent initiation in the “Don’t Touch It, You Idiot” trope

$$\mathcal{C}^\downarrow(\mathcal{X}, \mathcal{E}) : \left\{ \begin{array}{l}
\langle \{ \text{phase}(\text{dontTouchItYouIdiot}, \text{inactive}) \}, \\
\text{intDontTouchItYouIdiot}(\text{owner}, \text{idiot}, \text{item}, \text{away}) \rangle \rightarrow \emptyset \quad (6.4) \\
\langle \{ \text{phase}(\text{dontTouchItYouIdiot}, \text{phaseA}) \}, \\
\text{intDontTouchItYouIdiot}(\text{owner}, \text{idiot}, \text{item}, \text{away}) \rangle \rightarrow \emptyset \quad (6.5) \\
\langle \{ \text{phase}(\text{dontTouchItYouIdiot}, \text{phaseB}) \}, \\
\text{intDontTouchItYouIdiot}(\text{owner}, \text{idiot}, \text{item}, \text{away}) \rangle \rightarrow \emptyset \quad (6.6)
\end{array} \right.$$

Figure 6-2: Phase fluent termination in the “Don’t Touch It, You Idiot” trope

At first, each trope is “inactive”, reflected in the *phase* fluent as ($\text{phase}(\text{intTroveName}, \text{inactive})$). After each event that occurs in the trope, its phase is updated, starting at *phaseA*, then *phaseB*, through to *phaseZ*. Once a trope has finished (its final event has happened), its *phase* fluent is set to ($\text{phase}(\text{intTroveName}, \text{done})$). It is set to *done* rather than *inactive*, because there may be situations where we want to check whether or not a trope has already appeared, so that it may not be repeated for example.

Returning to our *Don’t Touch It, You Idiot* example, Figure 6-1 shows how a short version of the *Dont Touch It You Idiot* trope with just two phases would be described with *phase* fluents that are initiated according to their corresponding events, with Figure 6-2. The left side of Rule 6.1 shows a set of fluents that describe its initial state. In this case, the only fluent is $\text{phase}(\text{dontTouchItYouIdiot}, \text{inactive})$, which states that the *Don’t Touch It, You Idiot* trope is inactive. Beneath the set of state fluents is the institutional event that initiates the norms and fluents that appear to the right of the arrow. In the case of Rule 6.1, the *intDontTouchItYouIdiot* institutional event is initiating *Phase A* of the trope ($\text{phase}(\text{dontTouchItYouIdiot}, \text{phaseA})$), and the permission of the *owner* agent to drop the *item* ($\text{perm}(\text{drop}(\text{owner}, \text{item}))$). Rule 6.2 shows that *Phase B* and permission for the *owner* to go *away* are initiated by the *intDontTouchItYouIdiot* institutional event if the *dontTouchItYouIdiot* trope is in *Phase A*. Rule 6.3 simply sets the trope’s phase to *done* if it is in *Phase B*, without initiating any norms.

The termination rules in Figure 6-2 simply terminate any phase fluents that hold (represented by the empty set in Rules 6.4, 6.5 and 6.6), so that the new phase fluent replaces its predecessor.

Extending this example to include all events, along with the corresponding permissions and obligations that they grant our story characters, results in the institution described in Appendix B on page 194.

6.2 An Interactive, Generative Punch and Judy Show with Institutions and Emotional Agents

The previous section defines a scene from the Punch and Judy story world as an institution, in terms of external events that can happen, the institutional events that are generated and the norms that are initiated and terminated during the scene. This is our formal model for the scene, describing the norms that govern the agents that act out our characters in the story. This section explains how these norms are used with a multi-agent system to guide the character agents towards courses of action that fit with our scene description.

In this example, the characters in our Punch and Judy simulation are provided with emotional models that affect their decision making and behaviour. We choose to do this so that agents would be able to deviate from the narrative in times of extreme emotion, providing some level of unpredictability to the story. A description of how this might work appears in Section 7.3 on page 165.

This section describes a fully realised and implemented interactive puppet show built with institutions and emotional agents. In this show, the characters interact on screen in accordance with the Punch and Judy story world. At certain points in the show, the audience are encouraged to cheer or boo at the characters. When this happens, a character puppet's emotional state is changed in accordance with the audience response. A working prototype of this system was demonstrated at the CDE (Center for Digital Entertainment) and the AISB (Society for the Study of Artificial Intelligence and the Simulation of Behaviour) conferences in 2015, with live audiences interacting with and changing the narrative.

6.2.1 VAD emotional model

In order to make the agents acting out the Punch and Judy show more believable, we apply an emotional model to affect their actions and decisions. For this, we use the valence-arousal (circumplex) model first described by Russell (1980). To give each character its own distinct personality, we extend this model with an extra dimension: dominance, as used by Ahn et al. (2012) in their model for conversational virtual humans. This dominance level is affected by the reactions of the audience to the agents' actions. For example, Judy may become more dominant as her suggestions to hit Punch with a stick are cheered on by the audience, emboldening her into acting out her impulses.

The VAD model, an extension of Russell's circumplex model (Russell, 1980), illustrates how valence, arousal and dominance values map to identifiable emotions. Valence, arousal and dominance can each have a value of low, medium or high. This allows the agents to have a total of 27 distinct emotional states, shown in Figure 6-3 on the following page. The valence and arousal levels of each agent are affected by the actions of other agents. For example, a

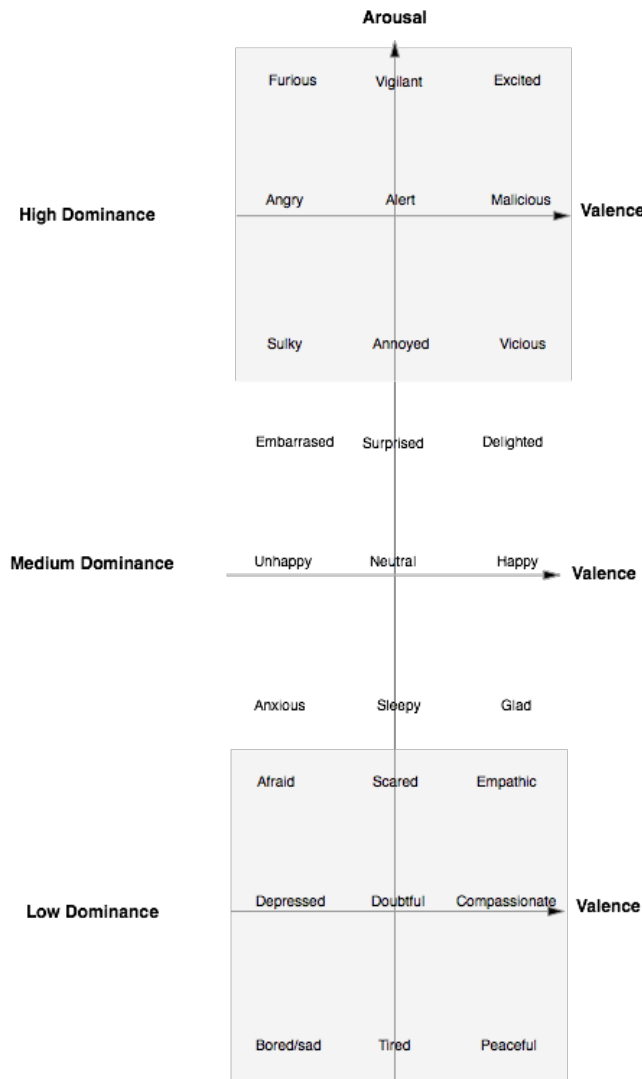


Figure 6-3: VAD emotional values (figure adapted from Ahn et al. (2012))

character being chased around the stage by Punch will see their valence level drop while their arousal increases. According to Russell’s circumplex model of emotion (Russell, 1980), this would result in them becoming *afraid* (if their dominance level is low).

An agent’s emotional state affects its ability to fulfil its institutional obligations. An agent that is *furiosus* might have no problem carrying out an obligation that requires them to kill another agent. If that same agent is *happy* or *depressed*, however, they might not have the appropriate motivation to perform such a violent action.

It is important to note that the emotional model is part of the agent belief state, and not held in the institution. We want to explore how the characters of the story might be able to choose actions based on their emotional state. While the institution could theoretically calculate the emotional state for each agent in turn and dictate this to them along with the norms of the narrative, it makes sense to decouple this feature from the narrative institution in order to separate the characters from the events of the story.

Agents’ emotional states change according to their interactions with the audience. This is

Listing 6.2: Emotional rules for a character with medium dominance

```

1 emotion(sleepy) :- valence(0) & arousal(-1) & dominance(0).
2 emotion(neutral) :- valence(0) & arousal(0) & dominance(0).
3 emotion(surprised) :- valence(0) & arousal(1) & dominance(0).
4 emotion(anxious) :- valence(-1) & arousal(-1) & dominance(0).
5 emotion(unhappy) :- valence(-1) & arousal(0) & dominance(0).
6 emotion(embarrassed) :- valence(-1) & arousal(1) & dominance(0).
7 emotion(glad) :- valence(1) & arousal(-1) & dominance(0).
8 emotion(happy) :- valence(1) & arousal(0) & dominance(0).
9 emotion(delighted) :- valence(1) & arousal(1) & dominance(0).

```

unrelated to what is happening in the narrative, and so this underscores the decision not to include any emotional modelling in the institution. Also, we want the agents to have some degree of freedom within the narrative world. They should be allowed to determine their emotions themselves, so that in extreme emotional states they can perform ‘irrational’ or ‘extreme’ actions that may not necessarily fit into the narrative.

The actions of other agents in the simulation also affects characters’ emotional states. Section 7.2 on page 153 gives a further example of an agent emotional model that take the actions of other agents into account.

6.2.2 VAD emotions in Jason

Emotions are implemented as beliefs inside an agent. An agent believes it has a certain level of valence, arousal and dominance, and it works out its emotional state based on a combination of these three factors. When the audience cheers or boos them, this changes the belief holding the relevant emotional variable, and their emotional state as a whole is recalculated.

Valence, arousal and dominance values can take values of -1 (low), 0 (medium) or 1 (high). Listing 6.2 shows the emotional belief rules for an agent with medium dominance (a dominance level of 0). Note that an agent maintains beliefs about both its current emotion label (such as sleepy or happy) and the separate valence, arousal and dominance values at the same time. Similar sets of rules handle the belief emotion for the other dominance levels.

Every time an emotional variable (valence, arousal, or dominance) changes, an agent’s emotion is changed according to the rules in Listing 6.2. While an agent’s valence, arousal and dominance belief values affect the way it makes decisions internally, the results of combinations of these values (sleepy, happy, etc) are broadcast as external actions. The reason for this is that an agent’s emotional state may affect the way in which the character is animated: changing the speed at which they move or turning their smile into a frown, for example. For this reason, whenever an emotional change takes place, the new emotion is published as an external action of the agent so that observing entities may perceive it. The Bath sensor framework described in Section 6.2.4 on page 129 provides the means for this evidence of the agent’s internal state change to be received by the animation system and reflected accordingly in the display.

Listing 6.3 on the next page shows the AgentSpeak rules describing how an agent’s valence and dominance levels are changed by the audience cheering or booing their actions. These AgentSpeak plans describe what the agent should do in response to a goal addition (denoted

Listing 6.3: AgentSpeak rules for changing an agent’s emotional values from audience responses

```

1  +!changeMood
2    <- ?emotion(Z);
3      emotion(Z).
4  +response(_, boo) : asking
5    <- -+valence(-1);
6      -+dominance(-1);
7      !changeMood.
8  +response(_, cheer) : asking
9    <- -+valence(1);
10     -+dominance(1);
11     !changeMood.

```

by a `+`! at the start of the plan name) or a belief addition (prefixed by a simple `+`). In the case of listing 6.3 on page 128, the `+`!changeMood plan on lines 1 to 3 updates the agent’s emotional state based on its valence-arousal-dominance values and broadcasts the result as an external action. The `+`response plans (Listing 6.3, lines 4 - 7 and 8 - 11) raise or lower an agent’s valence and dominance levels depending on whether the agent perceives a “boo” or “cheer” response from the audience.

An agent announces what they intend to do, then waits three seconds. During this time, they have the belief that they are “asking” the audience, and listen for a response. A “boo” reduces an agent’s valence and dominance, while a cheer raises them. For each response, the `changeMood` goal is triggered, which looks up and broadcasts the agent’s emotional state to the other agents and environment.

6.2.3 Agent decision making

The agents choose which goals to pursue according to three factors: their permitted actions, their obliged actions and their emotional state. Though obliged actions are given priority, and while agents’ decisions are generally constrained by their permitted actions, an agent’s emotional state has the final say in its decisions. In this way, an agent will follow the social norms of the narrative, but only according to their own mood.

Agent goals and plans The agents are implemented using a belief-desire-intention (BDI) psychological model using the Jason platform (Bordini et al., 2007). An agent’s knowledge about the state of their world and themselves are stored as *beliefs*, with new information coming in from the environment getting added to their belief base as *percepts*, which are ephemeral and only last for one reasoning cycle of an agent.

Agents are created with goals and plan libraries. Any goal that an agent is set on carrying out at any point is an *intention*, whereas a goal that an agent has but is not yet pursuing is a *desire*. Plan libraries describe the steps agents need to take in order to achieve goals, as well as how to react to changes in agents’ environments.

Norms as percepts When an event occurs, it is added to the event timeline, which is used to query the ASP (Answer Set Programming) solver to obtain the set of norms that hold after

the new event has occurred. The new permissions and obligations are then added to each agent as *percepts*. Each time this happens, the set of permitted and obliged actions that an agent sees is changed to be only those that apply at that instant in time, with the previous norms being discarded.

Agents choose between permitted and obliged actions based on their emotional state at the point of decision making. Obligated actions are given a higher priority over permitted ones for most of the emotional states that an agent can be in, though not always. If an agent is in a sulky mood, for example, they may decide to ignore what they are obliged to do by the narrative, even though they know there will be consequences.

For example, in the scene where Joey gives the sausages to Punch, Punch may see that he has permission to eat the sausages, drop them, fight the crocodile, run away (leave the stage) or shout for help at the crocodile or audience. His obligation for the scene, in accordance with the Punch and Judy narrative world, is to either eat the sausages himself, or let the crocodile have them. Note that his obligation in this case is not to guard the sausages as asked to by Joey. While Joey’s entrusting of the sausages is certainly an obligation in itself, Punch’s main obligations are to the narrative. Lesser obligations towards characters in the story can be implemented as having a lower priority than those of the story itself.

6.2.4 Architecture

Multi-Agent System: We use the Jason (Bordini et al., 2007) framework for belief-desire-intention (BDI) agents, with a VAD emotional model as described above.

Institutional Framework To describe our institutional model, we use InstAL (Cliffe et al., 2007), a domain-specific (action) language for describing institutions that compiles to AnsProlog, a declarative programming language for Answer Set Programming (ASP). A more detailed introduction to InstAL appears in Section 4.4.2 on page 74.

InstAL describes how external events generate institutional events, which then initiate or terminate fluents that hold at certain points in time. These fluents can include the permissions and obligations that describe what an agent is permitted or obliged to do at any point.

For example, if an agent with the role of *owner* leaves the stage, it generates the *intDontTouchItYouIdiot* event in the institution, but only if the *phaseA* phase is active (i.e., the *phase(phaseA)* fluent holds, meaning that the first event in the trope has already occurred). The InstAL code that generates the institutional event from the external event is shown in Listing 6.4.

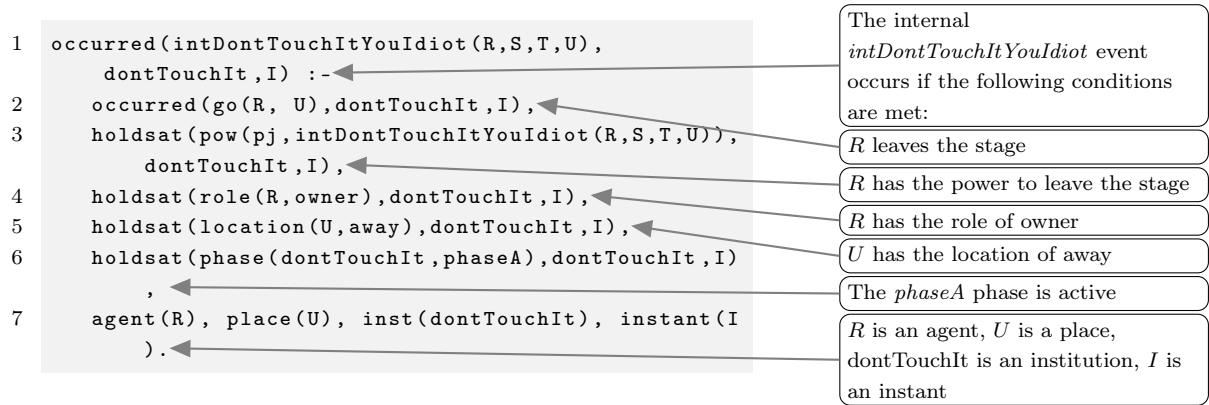
Listing 6.4: The “Don’t Touch it, You Idiot!” trope in TropicAL

```

1 go(R, U) generates
2   intDontTouchItYouIdiot(R, S, T, U) if
3     role(R, owner),
4     place(U, away);

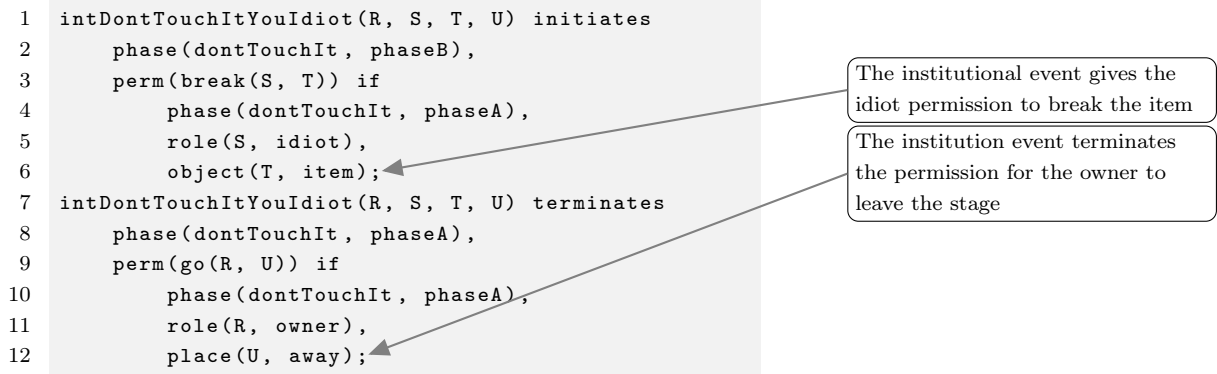
```

The InstAL generation event is compiled to the AnsProlog code shown in Listing 6.2.4 on the next page

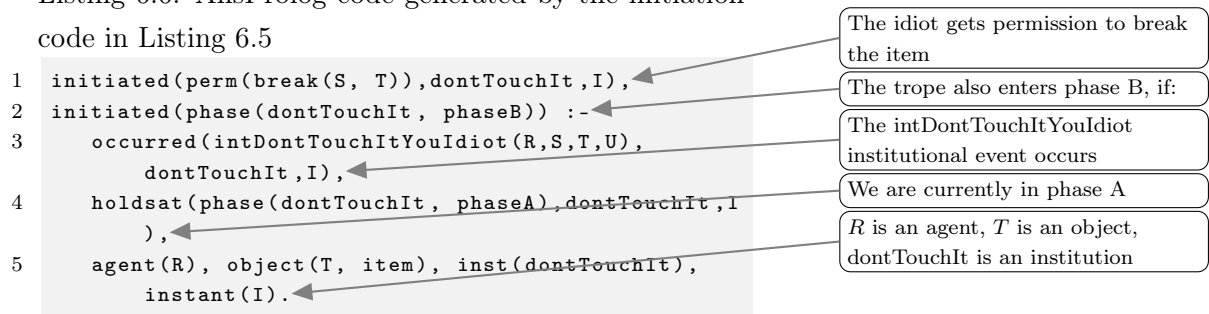


The *intDontTouchIt* institutional event gives the idiot permission to break the item if there is one. It also terminates the permission of the absented agent to leave the stage, as they have already done so. This is described in Listing 6.5. The AnsProlog generated by the InstAL code is shown in Listing 6.6.

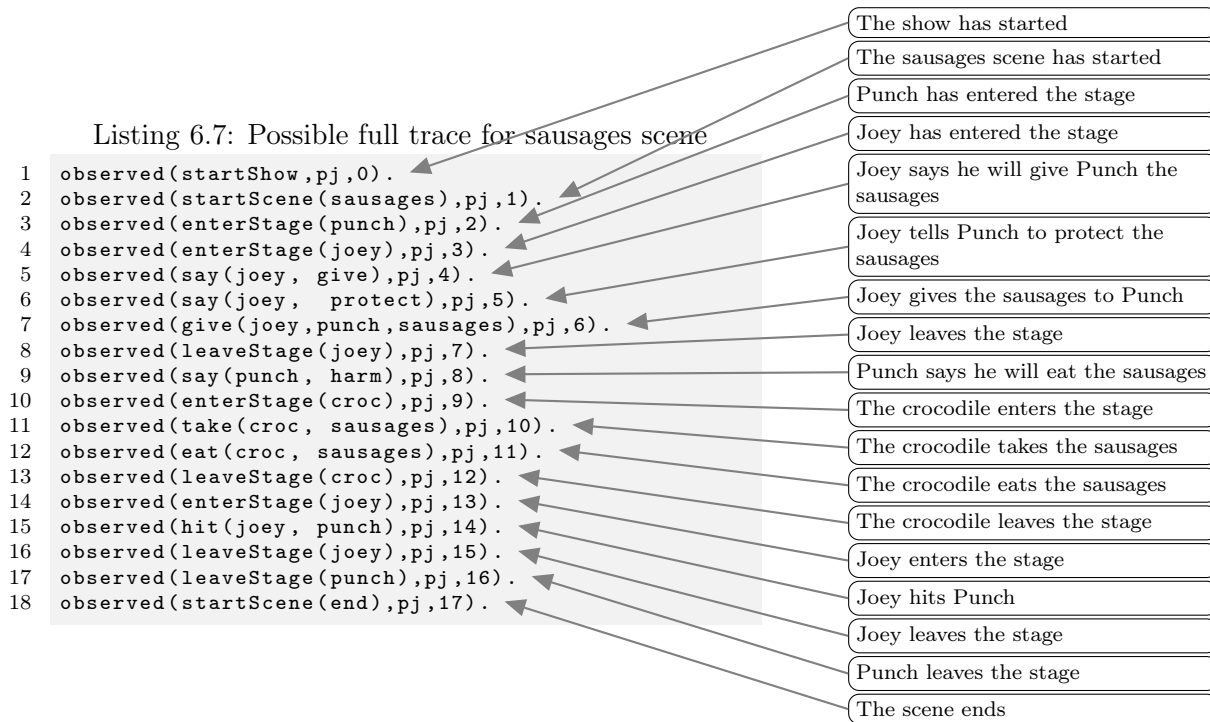
Listing 6.5: Initiation and termination events for the second event of the “Don’t Touch It, You Idiot” trope



Listing 6.6: AnsProlog code generated by the initiation code in Listing 6.5



By combining statements such as these, we can build a complete description of the sausages scene in terms of agent norms. A full listing is shown in Listing B.1 on page 194 of Appendix B on page 194. InstAL rules like those shown here are compiled into AnsProlog, then we use the *clingo* answer set solver (Gebser et al., 2011) to ground the program, and ‘solve’ queries by finding all permissions and obligations that apply to any agents, given a sequence of events as the query input. The agents’ percepts are then updated with their permitted and obliged actions from that time instant onwards. Thus, the institutional model acts as a social narrative sensor, interpreting actors’ actions in the context of the combination of the concrete narrative



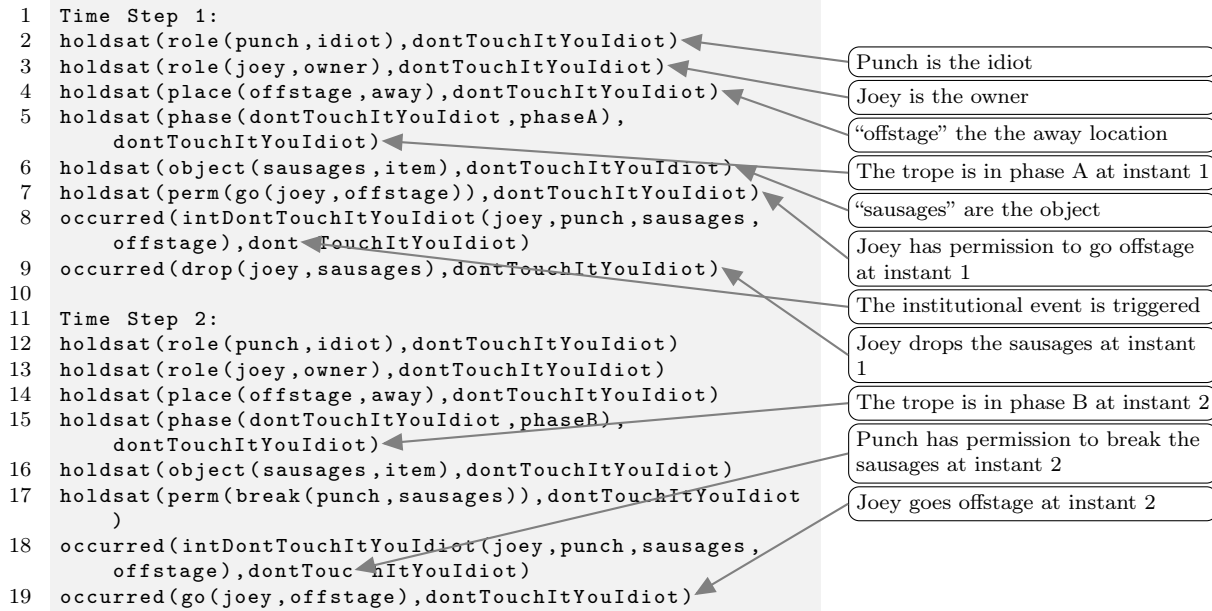
and the abstract story moves which detach (instantiate) the norms that guide the actors in the direction of the conclusion of the story arc.

A query is simply a list of external events in chronological order, also called a *trace*. A possible trace describing the actions of agents acting out the *sausages* is described in Listing 6.7 on page 131. The ‘pj’ in the trace is the name of the institution that observes the events, while the number is the enumeration of events in the sequence.

Each *observed* event triggers a corresponding *occurs* event inside the institution, as determined by the *generates* relation. Listing 6.8 on page 132 shows an extract from an answer set output for the trace queried against the ASP description of the sausages scenario, for events 1 to 2 of the scene. Starting with an initial set of fluents that hold at instant 1, only fluents that have been initiated and not terminated hold at the next instant. For ease of reading, some fluents such as institutional powers are omitted from the listing. Figure 6-4 on page 133 shows a visualisation of the answer set for the trace in Listing 6.7.

Bath Sensor Framework The components communicate using the Bath Sensor Framework (BSF) (Lee et al., 2013), through publish / subscribe-style communication between distributed software components, in this case connecting intelligent agents with their virtual environments. It currently uses the XMPP publish/subscribe protocol for communication between agents, environment and other software components. Each agent subscribes to receive notifications of environment changes via the appropriate topic node in the XMPP server, which relays messages between publishers and subscribers. If any environment change occurs, all subscribed agents are informed of the changes.

Listing 6.8: Sausages scene trace from ASP



Audience Interaction The puppet show is designed to be run in front of either a single user’s computer, or on a large display in front of an audience. The user/audience is instructed to cheer or boo the actions of the characters of the show, which will be picked up by a microphone and ‘heard’ by the agents. This will then affect the emotional state of the agents and change the actions they make in the show. Their actions are constrained by the set of ‘Punch and Judy’ world norms as described in the institutional model.

There are many different ways in which the audience’s responses can affect the outcomes of the show. If the audience craves a more ‘traditional’ Punch and Judy experience, then they can cheer Punch into beating and killing all of his adversaries (including his wife, Judy). Alternatively, a more mischievous audience could goad Judy into killing Punch and then taking over his role as sadist and killer for the rest of the show. The narrative outcomes are dependent on how the audience responds to the action, yet still conform to the rules of the Punch and Judy story world.

6.3 Summary

Our implementation of the interactive Punch and Judy show demonstrates the use of social institutions to model narrative.

However, this section only describes one scene of the Punch and Judy show. Section Section 7.2 on page 153 on page 153 describes how the full Punch and Judy story world would be implemented using TROPICAL, INSTAL and AGENTSPEAK.

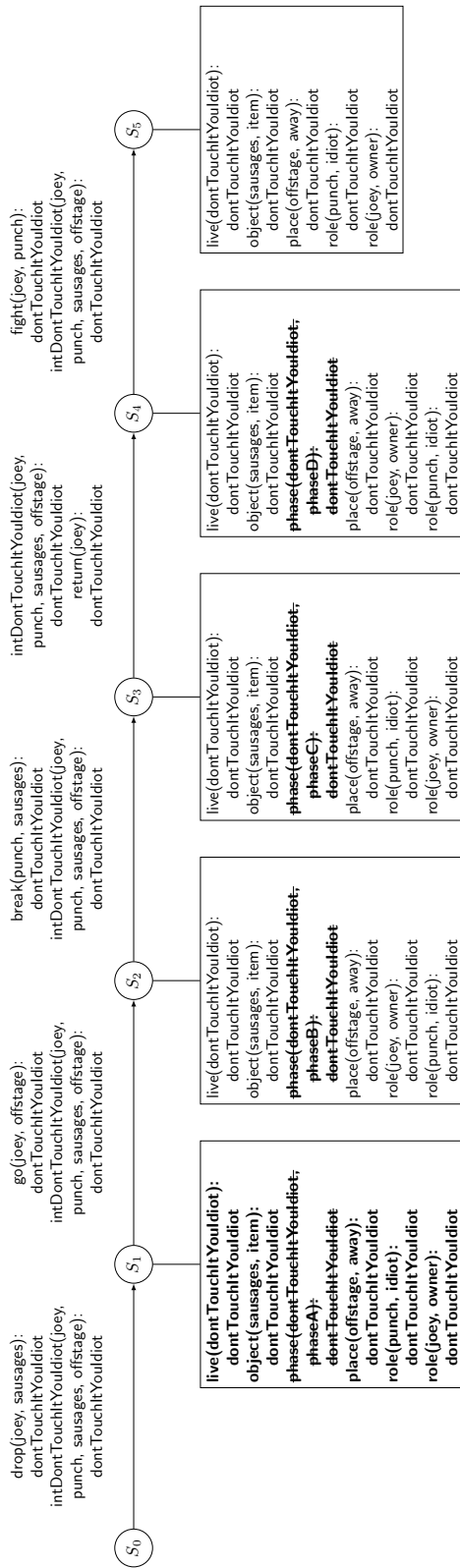


Figure 6-4: Trace visualisation

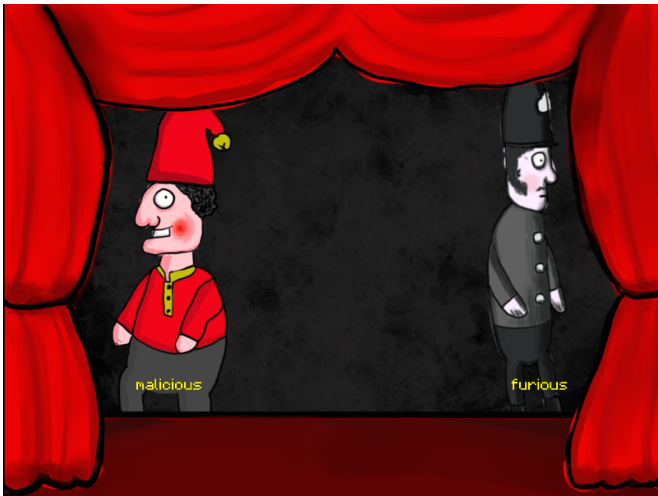


Figure 6-5: A screenshot of the Punch and Judy show

The animation engine that shows the visual output of the agents actions is written in Javascript and the Phaser game framework. It runs entirely in a browser, and communicates with BSF using the Strophe XMPP library.

If the user allows the program access to their microphone, they can cheer or boo the actions of the agents by shouting into the microphone. Otherwise, they can simulate these actions by clicking on 'cheer' or 'boo' buttons at the bottom of the screen.

Chapter 7

Evaluation & Validation

This chapter provides a full evaluation of both the TropICAL and StoryBuilder tools. First, a user study of the StoryBuilder user interface, with discussion and a thematic analysis of user interview transcripts, appears in Section 7.1. Next, a validation of the TropICAL language is described through a complete description of a Punch and Judy story in Section 7.2 on page 153. A discussion on how tropes are combined together is in Section 7.2.10 on page 164, and the chapter concludes with an examination of the extent to which “character freedom” has been achieved, and the potential benefits of this, in Section 7.3 on page 165.

7.1 StoryBuilder Evaluation

In order to evaluate the effectiveness of both TropICAL and StoryBuilder for the construction of tropes and nonlinear stories, we carried out a usability study, evaluated qualitatively through a thematic analysis (described by Clarke and Braun (2014), and in Section 7.1.5 on page 141).

The StoryBuilder tool is not integrated into a multi-agent system. This is due to its focus on story structure design and visualisation using tropes, rather than for the creation of the character agents themselves. For this reason, the analysis is not designed to evaluate requirements 1, 2 and 8 from Section 4.2.2 on page 62, only those related to the use of tropes for story construction:

- R3. The user must be able to write and re-use existing story components (from a library)
- R4. The components must be able to express sequences of events
- R5. The components must be able to express branching (diverging) events
- R6. The user must have the ability to nest existing components inside new components
- R7. The user must be able to visualise the branches of the story that result in the addition or modification of components to the story

Additionally, the study aims to examine the user experience of using tropes to construct stories in StoryBuilder with the following supplemental requirements:

- S1. To evaluate the suitability of TropICAL’s controlled natural language syntax for non-programmer story authors
- S2. To examine the use of tropes as reusable story components
- S3. To see if tropes allow a natural means of creating abstractions of story patterns
- S4. To study whether the story tree visualisations generated from the answer set solver output assists in non-linear story creation

These requirements are difficult to measure compared to those described in Section 4.2.2 on page 62, and so are used as the basis of the questions asked throughout the thematic analysis:

- How easy or difficult did you find the language to learn? (Requirement 1)
- Could you create the story you wanted by combining tropes together? (Requirement 2)
- Were you able to create the abstractions you wanted using subtropes? (Requirement 3)
- Did the visualisations assist you in the creation of tropes and stories? (Requirement 4)

The aim of the thematic analysis is to identify themes that emerge from the participants’ answers to these questions.

7.1.1 Procedure

The study begins with a brief explanation of the syntax and features of the TropICAL language: entities (roles, places, objects), events (sequences and branches) and subtropes. Once they have become familiar with the language and seen some visualisations of example tropes, they are asked to complete seven tasks. The tasks start with the user making simple changes to existing tropes, and give the user increasing freedom, finally giving them the opportunity to create tropes for a story of their own. Detailed descriptions of the tasks are given in Section 7.1.3 on the next page.

7.1.2 Participants

Eight participants were recruited for the study:

- **Participant A** is a computer science student, with good programming experience
- **Participant B** is an architect and former English teacher with a little programming experience
- **Participant C** is a creative writer with no programming experience
- **Participant D** is a computer science student with good programming experience
- **Participant E** is a computer science student with good programming experience

- **Participant F** is a creative computing student with some programming experience
- **Participant G** is a computer scientist that works on well-known interactive fiction tools
- **Participant H** is a computer science graduate student with good programming (and answer-set programming) experience

The studies were conducted using Google Hangouts, with each participant loading StoryBuilder by going to a web address (<http://mthompson.org/storybuilder>) and sharing a screen capture of their interactions with the system. Each study was recorded using screen capture software. The full transcripts, with annotations, are listed in Appendix G on page 219.

7.1.3 User Tasks

This section describes the tasks that the users carried out as part of the study.

Task 1: Edit an Existing Trope

The user is instructed to get a feel for the language by browsing through the examples provided, then presented with this task:

In the 'edit' tab, select an existing trope from the dropdown box. Edit its text on the left, and then save it. Once this is done, check to see that the visualisation seems to be as expected.

This task is designed to examine how easily the user is able to read and understand the TropICAL languages before making small changes to existing tropes, testing requirement R3.

Task 2: Create a Trope

The second task gives the user some specific character roles and places to use in their trope, and then instructs them to combine them together into a sequence of events:

Given the following character roles:

- Hero
- Villain
- Mentor

And the following places:

- Home

- Evil Lair
- Land of Adventure

Construct a trope using Tropical. Your statements will be simple sequences at first, i.e:

```
The Hero goes home
Then the Hero goes away
Then the Villain kills the Hero
```

Make sure to declare your roles and objects first.

This task is designed to allow the users to gently begin creating their own tropes, by limiting their role and place choices at first, as well as limiting their trope complexity by only allowing sequences of events without branches. This also tests requirements R3 and R4.

Task 3: Adding objects to your trope

This task extends task 2 by allowing the user to add objects to their trope.

Add the following objects to your trope:

- Weapon
- Evil Plan

For example:

```
The Hero takes the Weapon
Then the Villain makes an Evil Plan
```

Task 4: Add branches to your trope

In task 4, the user is asked to add some branching points to the trope that they have made:

Using the "Or" keyword, add branches to your story. For example:

```
The Hero goes to the Land of Adventure
Then the Hero kills the Villain
```

Or the Villain kills the Mentor
Or the Mentor goes home

This task introduces the concept of branches to the user for the first time, testing requirement R5.

Task 5: Put two tropes into a story

In this task, the user takes two existing tropes from the database, and combines them together with the “arrange” tab:

Using the "Arrange" tab in Storybuilder, select two (or more) tropes from the dropdown boxes to visualise all the possible paths through a story with those tropes.

Once this task is completed, the user is able to see a visualisation of all of the possible combinations of events that result from combining two tropes together. This task tests requirement R7.

Task 6: Tropes within Tropes

This task introduces the user to the concept of a *subtrope*, asking them to embed an existing trope inside of a new one:

Create a new trope that uses an existing trope as a "subtrope". For example:

"Hero May Leave" is a trope where:

The Hero is a role
Away is a place
Home is a place
The Hero goes Away
Or the Hero stays at Home

"Crime Flee Dilemma" is a trope where:

The Hero is a role
The Puppy is a role
The Hero kills the Puppy
Then the Hero panics
Then the "Hero May Leave" trope happens

In the above example, the first trope is used inside the second trope.

As this task introduces the concept of *subtropes* to the user, it tests requirement R6.

Task 7: Free Story Creation

Finally, the user is given the opportunity to create whatever they want. They can create new tropes, embed subtropes, and combine them into a story:

Using a combination of the techniques learned up until now, please create a story of your own, using any combination of characters, places, objects, and subtropes.

Once you have created the story, you can explore the possible paths through it with the tree visualisation.

This task tests requirements R3 to R7 by asking the user to use all of the components of TropICAL and StoryBuilder to freely create a story of their own.

7.1.4 Limitations of the System

Due to time restrictions and bugs in StoryBuilder, the following limitations applied to the participants' use of the software:

- Tropes can have a maximum of five events in a sequence (not including branches). This limitation is to prevent a combinatorial explosion from the answer set solver when combining multiple complicated tropes together.
- Subtropes must go at the end of a trope. This limitation is due to time limitations in institutional bridge generation.
- Branches cannot be extended beyond one event (as described by indenting an extra level after an *Or* statement, described in Listing 4.14 on page 67 of Section 4.3.4 on page 66), again due to time limitations and the combinatorial explosion this would introduce. Fluents would need to be added to each trope's InstAL institution to track the state of each branch, adding much complexity to the code generation process.
- Branches cannot be cut off (by writing "Then the story ends", shown in Listing 4.15 on page 68), and so always merge back into the main story. Omitted due to the same reasons as above.

- The tropes must be event-based. This means that updating the state of entities by writing statements such as “The Hero has an Apple”, or “The Apple is in the Barn” does not work. Verbs other than *has* or *is* must be used in these cases, for example: “The Hero takes the Apple” or “The Apple rests in the Barn”. This would introduce complexity in code generation due to the need to understand the meaning of certain verbs. For example, if the Apple is in the Barn, and then the Hero takes the Apple, this means that the Hero must have the Apple. In order for a fluent such as $has(R, S)$ if $role(R, hero)$, $object(S, apple)$ to track the state of the apple, the semantics of verbs such as *take*, *give* and *drop* would have to be understood by the system. This could be achieved with the aid of libraries such as *WordNet*, but would be out of the scope of this dissertation.
- Though the TropICAL parser can translate obligations, deadlines and violation events, the participants were not asked to test this due to the complexity that it would add. The aim of the study was to examine the use of tropes as reusable story components, and teaching the complexity of obligations with deadlines and consequences would have introduced a steeper learning curve for the users.

Despite these limitations, enough of the TropICAL language and StoryBuilder interface were implemented to test the usage of building a story using tropes as re-usable components.

7.1.5 Analysis

Transcripts for the interviews were analysed using the thematic analysis method described by Hayes (2000) and Clarke and Braun (2014). This method involves careful reading of the transcripts and identification of categories (referred to as “themes”) into which patterns identified in the responses can be organised.

We identify twenty different themes from the participant transcripts listed in Appendix G on page 219. These themes fall into four categories:

1. **Features:** what the user can or can’t express with TropICAL and StoryBuilder.
2. **Syntax:** words and indentation used in the TropICAL language
3. **Limitations:** things that the user wanted to do, but couldn’t, and any bugs in the system
4. **Usability:** how user-friendly both the StoryBuilder interface and TropICAL language are for story authors

The full list of twenty themes is as follows:

Features

1. Different characters in different tropes
2. State management

3. Event with three entities
4. Event semantics
5. Other types of entities
6. Subject must be an agent
7. Story structure
8. Conditionals

Syntax

9. Spaces in words
10. Using any verb
11. Use of “the”
12. Entity declarations
13. Capitalisation

Limitations

14. Five event limit
15. Branch length limit
16. Subtrope at end
17. Error messages

Usability

18. Ease of use
19. Visualisation
20. Tropes

The themes, identified when a pattern appears in the experiences of two or more participants, are explained in detail below:

Features

- **Different characters in different tropes**

This theme comes from participants asking questions such as “what happens if I have two different heroes in two different tropes?”.

For example, if there are two tropes active in one story, such as *The Hero’s Journey* and *The Chase Scene*, both tropes have Hero characters in them. Does our system allow us to use two different characters for each hero role, one per trope?

This feature is present in an earlier version of StoryBuilder, but not included in the prototype. Its implementation in TropICAL is described in Section 4.3.1 on page 63. The user was able to select named characters to fulfil roles. In this case, they could select (for example) “Luke Skywalker” to be the hero in the *Hero’s Journey*, and “Harry Potter” to be the hero in the *Chase Scene*. In order to simplify the StoryBuilder interface for the study, this feature was removed. This means that the same character plays multiple roles in the system used for the study.

This theme appears in these lines of the transcript:

- Participant A, line 3
- Participant D, line 241

- **State management** Some participants wanted to be able to write statements such as these:

- “The Hero is in the Barn”
- “The Villain has a Sword”
- “The Mentor is worried”

The intention is to update the state of these characters, objects and places. Though this was not implemented at the time of the study, creating fluents in InstAL to keep track of state would be simple:

- `in(hero, barn)`
- `has(villain, sword)`
- `is(mentor, worried)`

The only difficulty would be updating the state when changes occur later. For example, if the hero has an apple, and then gives it to the villain, then the `has(hero, apple)` fluent would have to be terminated.

This theme appears in these lines of the transcript:

- Participant A, lines 26, 142 259

- Participant B, line 275
- Participant D, line 312
- Participant F, line 76

- **Event with three entities**

Some participants in the study wanted to express events involving three entities, such as “The Mentor gives the Sword to the Hero”. In this case, the three entities are *Mentor*, *Hero* and *Sword*. The TropICAL parser was designed to handle statements such as these, but is not currently robust enough to handle some of the examples that the participants wanted to express.

This introduces another question: how complex do we want statements to be? Would it be useful to allow events involving four entities, such as “The Hero frees the Princess by giving the Sausages to the Dog”? The fact that no participants tried to enter statements with more than three entities indicates that this kind of event might be more complex than most users need, though further studies would be needed to determine if this is the case.

This theme appears in these lines of the transcript:

- Participant A, line 137
- Participant D, line 349
- Participant G, line 47

- **Event semantics**

A common pattern in the study was the users expecting the language to have semantics beyond allowing the agents to carry out certain actions. The actions that TropICAL permits and obliges the agents to carry out are simply seen by the agents as keywords that they themselves have to interpret. This means that the semantics of the verbs that are permitted and obliged by the system must be described in the plans of the agent. For example, if an event states “The Hero goes Home”, then the Hero agent only sees that it has permission to go home. It is up to the Hero to interpret this to mean that it physically has to move to another location, which should be described in a pre-existing plan.

The expectation that the events would have semantics created surprising effects for the participants while visualising their tropes. One user (Participant A, line 169) wrote a series of events where one character is killed, such as “The Villain kills the Hero”, and expected all subsequent branches featuring the Hero to disappear after that event. This was not the case, as the solver has no way of knowing that the *kill* verb should remove the Hero agent from the story.

This theme appears in these lines of the transcript:

- Participant A, line 169
- Participant D, lines 171, 352
- Participant H, lines 3, 59

- **Other types of entities**

Some users felt constrained by the fact that entities could only be roles, objects or places. Participant B wanted to describe the health state of agents with statements such as “Healthy is a Condition” and “David is Healthy”. Participant F expected to be able to define any type of entity at first, and had to be corrected.

This theme appears in these lines of the transcript:

- Participant B, line 193
- Participant F, line 7

- **Subject must be an agent**

There were users who wanted objects to be able to perform actions. For example, Participant B wanted to be able to write “The Sun is an object, The Sun shines”, but had to define the Sun as a role rather than an object. This constraint is due to the fact that TropICAL describes social norms. This could be addressed by allowing objects to be simulated by agents in addition to roles.

This theme appears in these lines of the transcript:

- Participant B, line 325
- Participant D, lines 143, 324

- **Story structure**

Three users in the study stated that they would like to have more control over story structure. Participant C stated that she would like the character roles to be dynamic and changeable, so that (for example), a Princess character could later be revealed to be a Ninja.

Participant F expressed a desire for the linear sequencing of tropes, rather than combining them together. This is possible if the tropes are sequenced as subtropes within another trope, that happen one after another. For example:

“Trope Sequence” is a trope where:
 The “Trope A” trope happens
 Then the “Trope B” trope happens
 Then the “Trope C” trope happens
 Then the “Trope D” trope happens

Participant G was concerned that the system was only able to express tropes at a certain level of abstraction, as it could only describe things that happen in terms of character actions, and could not describe changes in the abstract shape of the story. For example, TropICAL forces users to describe specific actions and events such as “The Hero fights the Villain”, but cannot express that some kind of climactic scene happens in an abstract way. The participant wanted to be able to express tropes such as Vonnegut’s “Man In Hole” described in Section 2.1.1 on page 19: “Things are good, then things are bad, then things are good again”. To be able to express tropes at this level of abstraction, subtropes or events would have to be tagged with their level of “drama”. This could perhaps be achieved through sentiment analysis, but this would be outside the scope of this work.

This theme appears in these lines of the transcript:

- Participant C, line 285
- Participant F, line 181
- Participant G, line 217

• Conditionals

A couple of participants stated that they would like to add some kind of conditional “if” statement to the language, so that the user may have preconditions for sequences of events. For example:

```
The Hero goes Home
If the Villain goes Home:
    The Hero fights the Villain
    Then the Hero kills the Villain
    Or the Villain escapes
Then the Hero goes Away
```

This would be a good feature to add to the language. In particular, if one could add several preconditions, the resulting tropes could become very expressive:

```
If the Hero meets the Mentor
And the Hero finds the Sword
And the Hero kills the Villain
    Then the Hero goes Home
    Then the Hero rests
```

However, due to the event-based nature of TropICAL in its current form, this is not too dissimilar from simply chaining permitted events together. The only difference is that the preconditions above can occur in any order, while those below must happen in a strict sequence:

The Hero meets the Mentor
Then the Hero finds the Sword
Then the Hero kills the Villain
Then the Hero goes Home
Then the Hero rests

For conditionals to be truly useful, we need to first implement ways to modify and check for changes in state for each entity (as described earlier under “State Management”). Then the preconditions can be rephrased in the following way:

If the Hero has met the Mentor
And the Hero has the Sword
And the Villain is dead
 Then the Hero goes Home
 Then the Hero rests

These preconditions, while enabling the user to express new combinations of events, add extra complexity to the language that was not needed during the evaluation of the suitability of tropes as re-usable story components. This is the reason for its omission during the study.

This theme appears in these lines of the transcript:

- Participant D, line 196
- Participant H, line 303

Syntax

- **Spaces in words**

Two participants were unsure if spaces were allowed in the names of entities of a trope. For example, “Danger Mountain” and “Land of Adventure” are both understood by the parser to be entity names. Participants (especially those familiar with programming) expected that the system would only be able to handle single-word entity names.

This theme appears in these lines of the transcript:

- Participant A, line 16
- Participant D, lines 47, 87

- **Using any verb**

This was another issue where users were confused by the parser being “too clever”. Many users started writing their tropes with the assumption that only a restricted subset of verbs were permitted. In fact, any English-language verb is allowed by the parser, due

to the fact that the verbs are fed into WordNet (Miller, 1995) to be interpreted (as described in Section 4.3.2 on page 64). Once users were told this fact, they had no problem choosing their own verbs to use in their tropes.

This theme appears in these lines of the transcript:

- Participant A, line 28
- Participant B, line 99
- Participant F, lines 28
- Participant G, lines 15

- **Use of “the”**

The TropICAL parser is designed to ignore any use of “the” in the name of an entity. For example, it will translate “The Hero” into just “Hero”. This caused some confusion amongst users, especially when the parser would inconsistently ignore the “the”s. Sometimes, for example, if a role was declared with a name starting with “The”, the parser expected to see it referred to with the “the” later on, as though it were part of the name. This was not by design, however, and was the result of a bug in the parser.

This theme appears in these lines of the transcript:

- Participant A, line 103
- Participant B, lines 60, 307
- Participant C, line 145
- Participant D, line 40
- Participant E, lines 111, 121
- Participant F, line 10
- Participant H, line 340

- **Entity declarations**

Some users expected to be able to put their entity declarations (such as “The Hero is a role”) at any point in the trope definition. However, the parser expects all of the entity declarations to be at the beginning. Though this was explained to users at the start of the study, some still wanted to be able to put the entity definitions anywhere in the text.

This theme appears in these lines of the transcript:

- Participant A, line 122
- Participant C, line 125
- Participant D, lines 162, 269

- **Capitalisation**

Entity names must be capitalised in TropICAL. This was a point of confusion for many participants, who often forgot to capitalise their entity names. This theme occurred so often that it would be worth considering adapting the parser to allow for entity names with lowercase or flexible capitalisations.

This theme appears in these lines of the transcript:

- Participant B, lines 48, 107
- Participant C, lines 64, 83, 111, 211
- Participant D, lines 137, 180, 188, 339
- Participant E, line 61
- Participant F, lines 19, 41

Limitations

- **Five event limit**

As explained above (Section 7.1.4 on page 140), the *clingo* answer-set solver that StoryBuilder uses was limited to producing answer sets of only five events in length for the duration of the study. This was done to reduce the amount of time that participants would have to wait when combining many tropes together. For individual tropes, the solver is quick to produce answer sets that are at least twenty events long, even with multiple branches. When multiple such tropes are fed as inputs into the solver, however, it may take some time to process.

Since many users found the five-event limit too constraining, it would be a worthwhile exercise to find ways to refine the queries that are input into the answer-set solver so that it can return fewer results in the answer sets.

This theme appears in these lines of the transcript:

- Participant A, line 153
- Participant C, line 236
- Participant D, line 342
- Participant E, lines 56, 139
- Participant G, lines 58, 189
- Participant H, lines 208, 322

- **Branch length limit**

In the prototype implementation used for the study, trope branches were limited to being only one event long. The path of a trope could only diverge for one event, before merging back into the main story path. Many participants wanted to be able to extend these

branches so that the alternative story paths could be longer. This was included in the design of the language, but not implemented for this proof-of-concept version for the study.

This theme appears in these lines of the transcript:

- Participant A, line 191
- Participant D, lines 56, 436
- Participant F, line 58
- Participant G, line 84

- **Subtrope at end**

Again, this was another limitation of the proof-of-concept prototype. Embedding subtropes inside a trope only worked with the answer-set solver if they were embedded as the very last event of the trope. A more complete implementation could allow subtropes to be embedded at any point in its parent trope.

This theme appears in these lines of the transcript:

- Participant A, line 241
- Participant B, line 344
- Participant C, line 232
- Participant E, line 274
- Participant H, lines 171

- **Error messages**

When a piece of TropICAL code does not compile in StoryBuilder, or if zero answer sets are produced by the solver, the error message “Compile Error” is displayed in place of the tree visualisation on the right. This is not very helpful to the user, as it does not give them enough detail to be able to identify the errors in their code in order to fix them. Solving this issue would require parsing the errors that come from the TropICAL parser, the InstAL compiler and the clingo answer set solver to identify the source of the error.

This theme appears in these lines of the transcript:

- Participant B, lines 10
- Participant G, line 53

Usability

- **Ease of use**

Participants described the system as “intuitive” (Participant A, line 118), “simple to use” (Participant F, line 222), and “self-evident” (Participant G, line 207). Features described as making the system easy to learn for non-programmers included the lack of a need for punctuation (Participant B, line 28)

A criticism of the language came from Participant D, describing the controlled natural language syntax as being perhaps counter-intuitive to users that are already programmers (Participant D, line 74). Participant E stated that the need to declare the entities in advance may be a barrier to learning for non-programmers (Participant E, line 19). The former issue is difficult to avoid when using a controlled natural language syntax targeted towards non-programmer users. The latter problem is difficult to address without developing an extremely sophisticated parser that is capable of identifying the subjects, verbs and objects in a natural language statement. The solution of requiring the user to define the subjects and objects up front (the entities) greatly simplifies the implementation and increases the predictability of the parser.

This theme appears in these lines of the transcript:

- Participant A, line 118
- Participant B, line 28
- Participant D, lines 74, 306
- Participant E, line 19
- Participant F, lines 211, 222
- Participant G, lines 207

• **Visualisation**

Participants found the visualisation of all possible paths through the story useful. Participant B in particular praised the ability to be able to see all the permutations of the story, stating that such a visualisation could also be applicable to domains such as architecture (for visualising different combinations of building designs). Participant E stated that the visualisation helped him to understand what was happening in his TropICAL code.

This theme appears in these lines of the transcript:

- Participant B, lines 371, 399, 409
- Participant E, lines 29, 222, 238

• **Tropes**

Participants found the use of tropes as story components to be useful. Participant C, as a story writer, found that it was a natural fit with the way that she thought about story structure (Participant C, lines 263 and 267). Participant E stated that tropes are a useful way to divide a story into modules (Participant E, line 311). Participant F

identified that the ability to embed tropes inside of other tropes prevented the problem of having unreasonable levels of nesting and indentation (Participant F, line 91).

This theme appears in these lines of the transcript:

- Participant C, lines 263, 267
- Participant E, lines 311, 409, 415
- Participant F, line 91

7.1.6 Discussion of StoryBuilder Evaluation

The purpose of this analysis is to evaluate the experience of authors using tropes as a reusable component for interactive story authoring. The assumption of existing interactive story authoring systems is that a narrative formalism is needed to be learned by the story author before they are able to describe a narrative in terms of machine-readable components.

The most notable result of the study is that non-programmer story authors were able to become comfortable with the TropICAL language syntax and trope concept very easily (Participant B, line 28, Participant C, lines 263 and 267). In fact, a comment from one experienced programmer participant suggested that he had more difficulty learning the language *because* of his programming experience (Participant D, line 74). This was an interesting theme that developed among the programmer group: they did not trust the “cleverness” of the language parser, preferring the lack of ambiguity that comes from a strictly defined programming language over the flexibility of the controlled natural language syntax that TropICAL uses.

The participant with the most experience in creative writing, Participant C, praised the use of tropes the most, explaining that it matched the way that she thought of story structure when writing stories (Participant C, lines 263 and 267). Other users’ comments were more focused on TropICAL’s controlled natural language syntax, with Participant B (line 28) describing how its lack of “punctuation” (brackets and braces) eases the learning curve for non-programmers.

Most of the frustrations that participants experienced when using StoryBuilder emerged from bugs or limitations from its implementation. The fact that tropes were limited to five events in length, and that branches could not be extended prevented most users from expressing the kinds of stories that they wanted to using the prototype. However, this is not due to the limitations of using tropes for the modelling of stories, but due to the incomplete nature of their implementation in StoryBuilder.

The expression of tropes in terms of social norms was a limiting factor for some story authors, however, with Participant G explaining that it limited the level of abstraction at which he could describe story events (Participant G, line 217). It is true that each trope must be defined in terms of agent actions and events, but these tropes can be inserted into a story as abstractions once they are defined. This allows a user to be able to describe a story more abstractly with statements such as “If trope A happens, then trope B happens or Trope C happens”. Due to the fact that trope A, B and C can be pre-written tropes from a library, a story author would not have to worry about their implementation in terms of agent actions at the lower level of abstraction.

Finally, the use of story visualisation alongside the controlled natural language syntax helped authors to understand the story structures that they were creating with the language. Participant B (lines 371, 399 and 409) describes how seeing all of the possible permutations could help story authors to refine the structure of their stories. Participant E (lines 29, 222 and 238) stated that the visualisations helped him see how changes to the TROPICAL code affected the resulting branches of the story he was creating.

Our data demonstrate that authors can create complex branching stories in TROPICAL and StoryBuilder with minimal instruction, regardless of their level of programming experience. All were able to create their own tropes and combine them into non-linear narratives, without the need to learn any kind of pre-defined formalism. Once the authors' stories are created, their compilation into a formal institutional representation allows for the use of an answer set solver to generate all of the possible paths through the story. This demonstrates that it is possible to allow the informal specification of stories through tropes and controlled natural language in a way that can be converted into a formal representation. The fact that the visual output of the formal translation of the tropes (the branching story tree visualisation that is generated by the answer set solver) matches the authors' expectations suggests that the translation from informal to formal is successful with this system.

Though this section has evaluated the usability of the StoryBuilder tool through a user study, the next section provides a more technical validation of the TROPICAL language by providing a fully worked example of the Punch and Judy story, including the TROPICAL code needed for each trope, their compiled InstAL institutions, how the tropes can be combined together, and the AgentSpeak code to control the agents that interact with the compiled institutions.

7.2 A Full Specification of Punch and Judy

In Section 6.1 on page 117, we described one scene from Punch and Judy in terms of tropes. The scene chosen in this example is one where a crocodile appears on stage to steal some sausages that Punch is guarding. This section gives an example of a *full* Punch and Judy show described in terms of tropes, including the “sausages” scene described previously.

Though each retelling of the Punch and Judy story may use a different number of scenes in varying order, we use the following scenes for our “minimal” Punch and Judy example:

- **Introduction:** Joey the clown introduces the show and interacts with the audience
- **Punch, Judy and the Baby:** Judy asks Punch to look after the baby, which he then kills. Punch then also kills Judy.
- **Punch and the Policeman:** A policeman comes to arrest Punch, and is killed.
- **Crocodile and Sausages Scene:** Joey asks Punch to look after some sausages, which are then eaten by a crocodile.
- **Punch and the Devil:** Punch fights the Devil, killing him.

7.2.1 “Introduction” Scene

In the *introduction* scene of *Punch and Judy*, Joey the Clown appears on stage to introduce the show to the audience. During the scene, Joey provokes an audience response by asking the audience to, for example, summon the *Punch and Judy* characters from backstage.

The tropes that appear in this scene would be:

- **Narrator:** Joey the Clown appears to be a narrator of sorts, who introduces the show. Due to his narrator status, he appears to be immune from *Punch*’s attacks.
- **Audience Participation:** Joey asks the audience to call out certain phrases during the introduction, often getting them to repeat themselves by calling out “Louder! I can’t hear you!”, for example.

The essence of the “Narrator” trope is simply to have a character that is able to talk to the audience. Listing 7.1 shows how this trope would be described in TropicAl.

Listing 7.1: The “Narrator” trope

```
1 "Narrator" is a trope where:  
2   The Narrator is a role  
3   The Audience is a role  
4   The Narrator talks to the Audience
```

In the “Audience Participation” trope, characters can talk to the audience, and the audience is also able to talk back. Listing 7.2 shows how this trope could be translated to TropicAl. In this example, certain characters can be assigned the role of “Audience-aware character”. Any character with this role would be able to talk to the audience, and the audience would be able to talk back to them.

Listing 7.2: The “Audience Participation” trope

```
1 "Audience Participation" is a trope where:  
2   Audience-Aware Character is a role  
3   The Audience is a role  
4   The Audience-Aware Character talks to the Audience  
5   Then the Audience talks to the Audience-Aware Character
```

7.2.2 “Punch, Judy and the Baby” Scene

In this scene, *Punch* and *Judy* appear on stage. After some comic dialogue, *Judy* announces that she needs to leave the stage, leaving *Punch* to take care of their baby. *Punch* chases the baby around the stage, finally hitting it with his stick, knocking it off stage and killing it. *Judy* returns to find that the baby is missing, and *Punch* and *Judy* have an argue. The argument ends with *Punch* hitting his wife with a stick, killing her.

This scene contains the following tropes:

- **Audience Participation:** as defined above

- **Don't touch it, you idiot!:** as defined earlier in Section 6.1.2 on page 122, this is where Judy tells Punch to look after the baby.
- **Put on a bus / the bus came back:** Judy then leaves the stage, only to come back later to discover that Punch has killed the baby
- **Chase fight:** Punch chases Judy around the stage with a stick
- **Domestic Abuse:** Punch beats and kills Judy

We define the TropICAL translation of the audience participation trope in the previous section (Section 7.2.1 on the previous page).

The combination of “Don't Touch It, You Idiot!” and “Put On a Bus / The Bus Came Back” is a common occurrence in Punch and Judy - it also appears in the “Sausages and Crocodile” scene. For this reason, it makes sense to create a new trope which is a hybrid of the two. We call this trope “Setting the Scene for Mischief”, and its TropICAL interpretation appears in Listing 7.3

Listing 7.3: The “Setting the Scene For Mischief” trope

```

1 "Setting the Scene for Mischief" is a trope where:
2   The Owner is a role
3   The Idiot is a role
4   The Item is an object
5   Away is a place
6   The Owner entrusts the Idiot with the Item
7   Then the Owner goes Away
8   Then the Idiot breaks the Item
9   Then the Owner returns
10  Then the Owner fights the Idiot

```

Listing 7.4: The “Chase Fight” trope

```

1 "Chase Fight" is a trope where:
2   The Pursuer is a role
3   The Pursuee is a role
4   The Pursuer chases the Pursuee
5   Then the Pursuee fights the Pursuer
6   Then the Pursuer kills the Pursuee
7   Or the Pursuee escapes

```

The “Domestic Abuse” trope simply describes a situation in which a husband and wife fight with each other, as described in the TropICAL code in Listing 7.5.

Listing 7.5: The “Domestic Abuse” trope

```

1 "Domestic Abuse" is a trope where:
2   The Wife is a role
3   The Husband is a role
4   The Wife fights the Husband
5   Or the Husband fights the Wife

```

7.2.3 “Punch and the Policeman” Scene

In this scene, Punch is confronted by the Policeman character, who attempts to arrest him for the murder of his wife and baby. Punch chases the Policeman around the stage, hitting him with a stick and finally murdering him as well.

This scene contains the following tropes:

- **Slapstick**: the characters hit each other in a comedic way
- **Chase Fight**: one character chases another and fights them, as defined above
- **Mini-boss**: the protagonist must defeat an enemy to continue the quest

The slapstick trope can be defined in TropicAL with the code shown in Listing 7.6.

Listing 7.6: The “Slapstick” trope

```
1 "Slapstick" is a trope where:
2   The Slapper is a role
3   The Slapped is a role
4   The Slapper slaps the Slapped
5   Then the Slapped slaps the Slapper
```

The “Chase Fight” trope already appears in the previous scene where Punch kills his baby and wife (Listing 7.4 on the previous page), so we can re-use that trope for this scene.

Listing 7.7: The “Mini-boss” trope in TropicAL

```
1 "Mini-boss" is a trope where:
2   The Hero is a role
3   The Mini-Boss is a role
4   The Hero fights the Mini-Boss
5   Then the Hero defeats the Mini-Boss
6   Or the Hero dies
7     Then the story ends
8   Then the Hero escapes
```

The translation of the “Mini-boss” trope into TropicAL code is shown in Listing 7.7.

7.2.4 “Crocodile and Sausages” Scene

This scene can simply re-use the “Setting the Scene for Mischief” trope from the previous scene with Punch, Judy and the baby. In this case, Joey the Clown takes the place of Judy as the “Owner”, telling him to look after an “Item”. In this case, the item takes the form of some sausages instead of a baby.

The only difference here is that instead of the “Idiot” breaking the item, it is broken by the “Crocodile”. This means that in order to use the “Setting the Scene for Mischief” trope, we need to modify it so that the “Mischief” that occurs can include events other than the “Idiot” breaking an item. Listing 7.8 on the next page shows a trope named “Mishandling of an Item” that describes the various ways in which an item can be mishandled. Listing 7.9 on

page 157 is the redefined “Setting the Scene for Mischief” trope, with the replacement of the Idiot breaking the Item with the “Mishandling of an Item” trope.

Listing 7.8: The “Mishandling of an Item” trope in TropICAL

```
1 "Mishandling of an Item" is a trope where:
2   The Idiot is a role
3   The Accomplice is a role
4   The Item is an object
5   Onstage is a place
6   The Idiot breaks the Item
7   Or the Idiot loses the Item
8   Or the Accomplice goes Onstage
9   Then the Accomplice breaks the Item
10  Or the Accomplice loses the Item
```

Listing 7.9: The “Setting the Scene for Mischief” trope with “Mishandling of an Item” embedded

```
1 "Setting the Scene for Mischief" is a trope where:
2   The Owner is a role
3   The Idiot is a role
4   The Item is an object
5   Away is a place
6   Home is a place
7   The Owner tells the Idiot to protect the Item
8   Then the Owner goes Away
9   Then the "Mishandling of an Item" trope happens
10  Then the Owner returns Home
11  Then the Owner fights the Idiot
```

7.2.5 “Punch and the Devil” Scene

This scene often (but not always) closes the Punch and Judy show. In it, the Devil himself confronts Punch. Punch, however, is not intimidated, and he chases the Devil around the stage before beating him to death with his stick.

This scene plays out in exactly the same way as the “Punch and the Policeman” scene, so we can re-use the same tropes: **Slapstick**, **Chase Fight** and **Mini-Boss**.

7.2.6 Putting It All Together

Having defined the tropes for each *Punch and Judy* scene in the previous section, we can now describe the scenes themselves in terms of the tropes and the characters’ roles within them. Then we can finally define the story itself in terms of a sequence of these scenes.

The story begins with the “Introduction” scene shown in Listing 7.10 on the following page, with the character of Joey playing the part of *Narrator* and the player assuming the role of the *Audience*. These roles are only used for the “Narrator” trope, the only one to happen in this scene. Though the “Audience Participation” trope also occurs, this trope is used when we

define the story as a whole (see Listing 7.15 on page 160). This is because the trope appears in every scene of *Punch and Judy*, and so adding it to the story definition has the effect of adding the trope to every scene.

Listing 7.10: The “Introduction” scene in TropICAL

```
1 "Introduction" is a scene where:
2   Joey is a Narrator
3   The Player is an Audience
4
5   The "Narrator" trope happens
```

Listing 7.11 shows the definition of the “Punch, Judy and the Baby” scene in TropICAL. As this scene contains two different tropes, each with multiple roles, the characters themselves must assume multiple roles in the scene. This means that Punch becomes both *Idiot* (for the “Setting the Scene for Mischief” trope) *Husband* (for the “Domestic Abuse” trope) and *Pursuer* (for the “Chase Fight” trope).

Judy’s roles are *Wife* (“Domestic Abuse”), *Owner* (“Setting the Scene for Mischief”) and *Pursuee* (“Chase Fight”). The Baby has the sole role of *Item* from its sole appearance in the “Setting the Scene for Mischief” trope. The three tropes are sequenced to happen one after another in the scene.

Listing 7.11: The “Punch, Judy and the Baby” scene in TropICAL

```
1 "Punch, Judy and the Baby" is a scene where:
2   Punch is an Idiot, Husband and Pursuer
3   Judy is a Wife, Owner and Pursuee
4   The Baby is an Item
5
6   The "Setting the Scene for Mischief" trope happens
7   Then the "Chase Fight" trope happens
8   Then the "Domestic Abuse" trope happens
```

The next scene in the puppet show is the “Punch and the Policeman” scene, which is shown in Listing 7.12. In this scene, Punch has the roles of *Hero* (“Mini-Boss”), *Pursuer* (“Chase Fight”) and *Slapper* (“Slapstick”). The Policeman’s roles are *Mini-Boss* (“Mini-Boss”), *Pursuee* (“Chase Fight”) and *Slapped* (“Slapstick”).

Listing 7.12: The “Punch and the Policeman” scene in TropICAL

```
1 "Punch and the Policeman" is a scene where:
2   Punch is a Hero, Pursuer and Slapper
3   The Policeman is a Mini-Boss, Pursuee and Slapped
4
5   The "Slapstick" trope happens
6   Then the "Mini-boss" trope happens
7   Then the "Chase Fight" trope happens
```

The fourth scene of the show is the one in which the Crocodile steals the sausages. Listing 7.13 on the following page shows the TropICAL code for the “Crocodile and Sausages”

scene. This scene is interesting, as it contains two tropes that are simultaneously active: the “Setting the Scene for Mischief” trope and the “Chase Fight” trope. The “Chase Fight” cannot begin until the Crocodile character appears however, which is only made possible as part of the “Mishandling of an Item” trope that appears midway through the “Setting the Scene for Mischief” trope.

In this scene, Joey the Clown assumes the role of *Owner* (“Setting the Scene for Mischief”), Punch is both *Idiot* (“Setting the Scene for Mischief”) and *Pursuee* (“Chase Fight”) and the Crocodile is both *Accomplice* (“Mishandling of an Item”, embedded in the “Setting the Scene for Mischief” trope) and *Pursuer* (“Chase Fight”).

Listing 7.13: The “Crocodile and Sausages” scene in TROPICAL

```
1 "Crocodile and Sausages" is a scene where:
2   Joey is an Owner
3   Punch is an Idiot and Pursuee
4   The Crocodile is an Accomplice and Pursuer
5   The Sausages are an Item
6   Offstage is Away
7   Onstage is Home
8
9   The "Setting the Scene for Mischief" trope happens
10  And the "Chase Fight" trope happens
```

The final scene, “Punch and the Devil”, is shown in Listing 7.14. This scene contains exactly the same tropes as the “Punch and the Policeman” scene: the “Slapstick”, “Mini-Boss” and “Chase Fight” tropes. In fact, it could be useful to combine these tropes together into a new abstraction (perhaps named “Slapstick Mini-Boss Fight”) if we find ourselves using them even more often, as this combination seems to be a common one in Punch and Judy puppet shows. Instead of the Policeman assuming the roles of *Mini-Boss* (“Mini-Boss”), *Pursuee* (“Chase Fight”) and *Slapped* (“Slapstick”), the Devil instead takes these roles. Punch assumes the same roles as in the other scene.

Listing 7.14: The “Punch and the Devil” scene in TROPICAL

```
1 "Punch and the Devil" is a scene where:
2   Punch is a Hero, Pursuer and Slapper
3   The Devil is a Mini-Boss, Pursuee and Slapped
4
5   The "Slapstick" trope happens
6   Then the "Mini-boss" trope happens
7   Then the "Chase Fight" trope happens
```

Now that we have defined the scenes that form a *Punch and Judy* show, we can combine them together to form a *story*. Listing 7.15 on the following page shows the TROPICAL definition of the entire *Punch and Judy* show, which is formed from the linear sequencing of the previously defined scenes. This is also where we add any tropes that we want to be active throughout the entire story: in this case, we want the “Audience Participation” trope to apply at any time, so that the characters also have the option to call out to the audience for a

response.

Instead of combining the scenes linearly, as shown in the listing, the *Or* keyword can be added to express alternative options for scene combinations in the same way that it is used to describe alternative sequences of events for tropes.

Listing 7.15: The full “Punch and Judy” story in TropICAL

```
1 "Punch and Judy" is a story where:  
2   The "Audience Participation" trope happens  
3   Punch is an Audience-aware Character  
4   Joey is an Audience-aware Character  
5   Judy is an Audience-aware Character  
6  
7   The "Introduction" scene happens  
8   Then the "Punch, Judy and the Baby" scene happens  
9   Then the "Punch and the Policeman" scene happens  
10  Then the "Crocodile and Sausages" scene happens  
11  Then the "Punch and the Devil" scene happens
```

The InstAL institutions that result from the compilation of these tropes are listed as examples in the source code repository for the TropICAL library, which is hosted on Github (Thompson, 2017).

7.2.7 Authoring the Character Agents

The institutions that are compiled from TropICAL code are designed to be used to govern the agents of a Multi-Agent System. In this example, we use the Jason (Bordini et al., 2007) framework for authoring character agents, and give examples of plan libraries for the character agents in the “Punch and Judy” story world.

Jason’s agent plan libraries are implemented in a Domain-Specific language named “AgentSpeak”. In the full system, the intention is to generate AgentSpeak code for each character role (such as the “Hero”, “Idiot” and “Pursuer”), and to use “include” statements to inherit the plans from these roles into plan libraries for specific agents (such as Punch, Judy, or the Policeman). The evaluated prototype described in Chapter 7 on page 135 does not include this functionality, however.

A character role’s “sphere of action” consists of everything they are permitted / obliged to do in *all* of the tropes in the library. For each role, a separate AgentSpeak plan library is generated. Each of these libraries for the Punch and Judy example are listed in Section F.1 on page 215 of the appendix, with the code for each character agent appearing in Section F.2 on page 216. A worked example of the “Punch” agent is explained below.

Common Agent Code

Each character agent shares two plan libraries that are the same for each agent: one library to describe how each agent moves around the story environment, and another that contains the emotional model for each agent. Listing 7.16 on the following page shows the shared

plan library for agent emotions, which contains the “Valence, Arousal, Dominance” emotional model for the agents. The listing shows examples of how each of these three emotional variables may change as the result of different actions being observed in the environment, such as an agent’s valence and dominance decreasing as a result of being slapped, or arousal increasing and dominance decreasing as a result of being pursued.

If the agent’s emotional state is “anxious”, “scared”, “furious” or “angry”, a “desperate” belief is added, indicating that the agent is in an extreme emotional state. This can be used to allow the character to permit actions that would otherwise not be permitted by the trope institutions, so that they can break away from the constraints of the narrative. An example of this situation is described in Section 7.3 on page 165.

Listing 7.16: AgentSpeak code for agent emotions

```

1 emotion(annoyed) :- valence(0) & arousal(-1) & dominance(1).
2 emotion(alert) :- valence(0) & arousal(0) & dominance(1).
3 emotion(vigilant) :- valence(0) & arousal(1) & dominance(1).
4 emotion(sulky) :- valence(-1) & arousal(-1) & dominance(1).
5 emotion(angry) :- valence(-1) & arousal(0) & dominance(1).
6 emotion(furious) :- valence(-1) & arousal(1) & dominance(1).
7 emotion(vicious) :- valence(1) & arousal(-1) & dominance(1).
8 emotion(malicious) :- valence(1) & arousal(0) & dominance(1).
9 emotion(excited) :- valence(1) & arousal(1) & dominance(1).
10
11 emotion(sleepy) :- valence(0) & arousal(-1) & dominance(0).
12 emotion(neutral) :- valence(0) & arousal(0) & dominance(0).
13 emotion(surprised) :- valence(0) & arousal(1) & dominance(0).
14 emotion(anxious) :- valence(-1) & arousal(-1) & dominance(0).
15 emotion(unhappy) :- valence(-1) & arousal(0) & dominance(0).
16 emotion(embarrassed) :- valence(-1) & arousal(1) & dominance(0).
17 emotion(glad) :- valence(1) & arousal(-1) & dominance(0).
18 emotion(happy) :- valence(1) & arousal(0) & dominance(0).
19 emotion(delighted) :- valence(1) & arousal(1) & dominance(0).
20
21
22 emotion(tired) :- valence(0) & arousal(-1) & dominance(-1).
23 emotion(doubtful) :- valence(0) & arousal(0) & dominance(-1).
24 emotion(scared) :- valence(0) & arousal(1) & dominance(-1).
25 emotion(bored) :- valence(-1) & arousal(-1) & dominance(-1).
26 emotion(depressed) :- valence(-1) & arousal(0) & dominance(-1).
27 emotion(afraid) :- valence(-1) & arousal(1) & dominance(-1).
28 emotion(peaceful) :- valence(1) & arousal(-1) & dominance(-1).
29 emotion(compassionate) :- valence(1) & arousal(0) & dominance(-1).
30 emotion(empathic) :- valence(1) & arousal(1) & dominance(-1).
31
32 desperate :- emotion(anxious) | emotion(scared) | emotion(furious) | emotion(
    angry).
33
34 +!changeMood
35     <- ?emotion(Z);
36     emotion(Z).

```

```

37
38 +slapped
39     <- -+valence(-1);
40     -+dominance(-1);
41     !changeMood.
42
43 +won
44     <- -+valence(1);
45     -+dominance(1);
46     !changeMood.
47
48 +defeated
49     <- -+valence(-1);
50     -+dominance(-1);
51     !changeMood.
52
53 +moved
54     <- -+arousal(1);
55     !changeMood.
56
57 +pursued
58     <- -+arousal(1);
59     -+dominance(-1);
60     !changeMood.
61
62 +pursuer
63     <- -+arousal(1);
64     -+dominance(1);
65     !changeMood.

```

7.2.8 The Idiot

The “Idiot” character role appears in the “Mishandling of an Item” trope, which is used in the “Punch, Judy and the Baby” and “Crocodile and Sausages” scenes in *Punch and Judy*. Listing 7.17 shows the AgentSpeak plans for this role. The code describes the actions of the character role in terms of what happens in the “Mishandling” trope: the character either breaks or loses the item in the trope. In the AgentSpeak code, this is performed by the agent trying to carry out plans for breaking and losing the item simultaneously. However, only the plan that is permitted by the institution is carried out. An agent perceives from its environment if it has permission to carry out an action, and so it checks its beliefs to see if it has permission to break or lose the item. If the agent is in a “desperate” emotional state, they are allowed to execute a plan regardless of their permission to do so.

Listing 7.17: AgentSpeak code for the “Idiot” role

```

1 // "Mishandling of an Item" trope
2
3 +startTrope(mishandling)
4     <- !breakItem;

```

```

5
6 +startTrope(mishandling)
7   <- !loseItem;
8
9 +!breakItem : not (desperate | perm(break(X))) & type(X, item)
10  <- !breakItem;
11
12 +!breakItem : (desperate | perm(break(X))) & type(X, item)
13   <- break(X);
14
15 +!loseItem : not (desperate | perm(lose(X))) & type(X, item)
16   <- !loseItem;
17
18 +!loseItem : (desperate | perm(lose(X))) & type(X, item)
19   <- lose(X);

```

7.2.9 Code for Punch Agent

Listing 7.18: AgentSpeak code for the Punch agent

```

1 // A lot of Punch's behaviour is inherited from the roles that he enacts in
   various tropes:
2
3 { include("movement.asl") }
4 { include("emotions.asl") }
5
6 { include("idiot.asl") } // "Punch, Judy, and the Baby", "Crocodile and
   Sausages"
7 { include("husband.asl") } // "Punch, Judy and the Baby"
8 { include("pursuer.asl") } // "Punch, Judy and the Baby", "Punch and the
   Policeman", "Punch and the Devil"
9 { include("hero.asl") } // "Punch and the Policeman", "Punch and the Devil"
10 { include("slapper.asl") } // "Punch and the Policeman", "Punch and the Devil
   "
11 { include("pursuee.asl") } // "Crocodile and Sausages"
12
13 { include("audience-aware-character.asl") } // All scenes
14
15 // The user is also able to author any other plans they want to add in order
   to customise Punch's behaviour

```

The AgentSpeak code for the Punch character agent is shown in Listing 7.18. As well as inheriting the common code for agent movement and emotions, Punch’s behaviour consists of the character roles from the tropes he acts out. In this case, Punch plays the “Idiot”, “Husband”, “Pursuer”, “Hero”, “Slapper”, “Pursuee” and “Audience-aware Character” in various scenes of the story, so these are the AgentSpeak plan libraries that his character’s plan library is composed from. The code for each of the plan libraries for each of these roles is listed in Section F.1 on page 215 of the appendix.

7.2.10 Combining Tropes

This section describes an example of a situation where a user wants to combine multiple tropes together. There are two ways in which TropICAL and StoryBuilder allow this: by allowing tropes to happen “at the same time”, and by sequencing tropes to occur at certain points in other tropes. Section 4.3.5 on page 68 describes the use of subtropes in TropICAL, which are used to implement the latter case of sequencing tropes within scenes. Their implementation through institutional bridges is described in Section 4.4.11 on page 92. This section examines how our system allows multiple tropes to occur at once.

As an example, we can imagine that a user wants to use the following three tropes in their story: the *Hero's Journey*, the *Evil Empire* and the *MacGuffin*. Using the StoryBuilder tool, the user can select all three tropes to occur at once using the dropdown menus in the “arrange tab” (shown in Figure 5-6 on page 107). Alternatively, if not using StoryBuilder, the following code in Listing 7.19 can be passed to the TropICAL compiler along with the trope definition files.

Listing 7.19: Simultaneous tropes in TropICAL

```
1 "Example Quest" is a scene where:  
2   The "Hero's Journey" trope happens  
3     And the "Evil Empire" trope happens  
4     And the "MacGuffin" trope happens
```

Due to the fact that each trope is expressed as a sequence of permissions and obligations in an institution, a scene with multiple simultaneously active tropes means that multiple institutions are active at once. This gives each character the choice of participating in each institution in the order of their choice.

Let us examine the “Hero” character role in this scene, for example. As all three trope institutions are active from the start of the scene, the character has permission to carry out the first event of any of the three tropes, as long as the permission applies to that character. Figure 7-1 on the next page shows an example where the first events of each institution permit the agent to perform an action, and so the agent is able to choose between these actions, in a similar manner to story branching.

If the first event of the “Hero’s Journey” trope is for the Hero to leave home, and the Hero does so, then the second event in the trope becomes permitted or obliged to occur, as shown in Figure 7-2 on the following page. Now we can see what happens if the Hero decides to follow the “MacGuffin” trope next: the second events of both the Hero’s Journey and MacGuffin tropes become permitted, as well as the first event of the Evil Empire trope (as the Hero has not chosen an action that follows that trope yet). Figure 7-3 on the next page shows the situation after two choices from the Hero.

A trope ends when its final event has occurred. The scene will end when either all the tropes have reached their points, or if the “The story ends” event occurs.

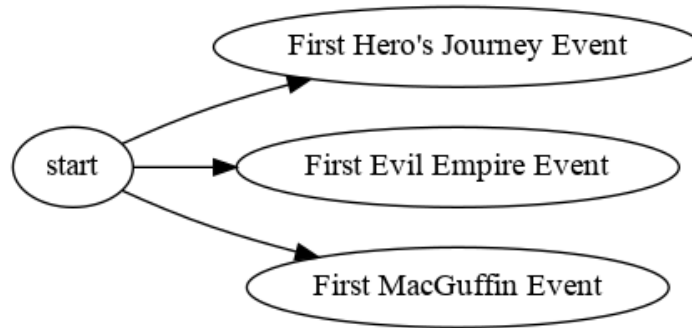


Figure 7-1: An agent can choose between the events of all three tropes at the start of the scene.

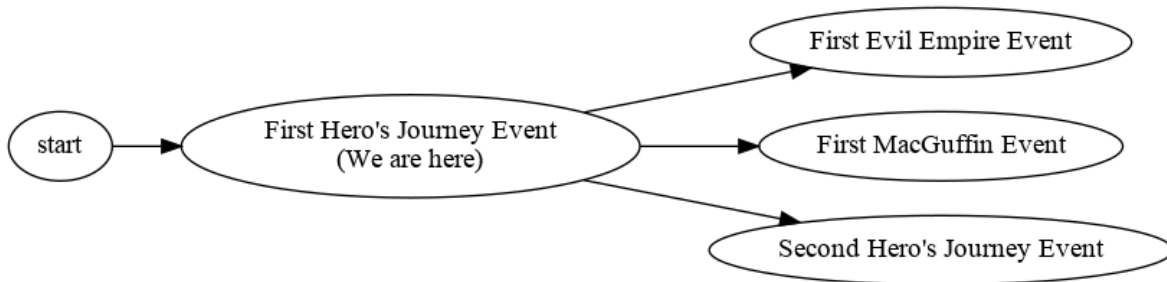


Figure 7-2: Now that the agent has chosen to perform the first event from the *Hero's Journey* trope, the second event from that trope is now permitted, along with the first event of the other two tropes.

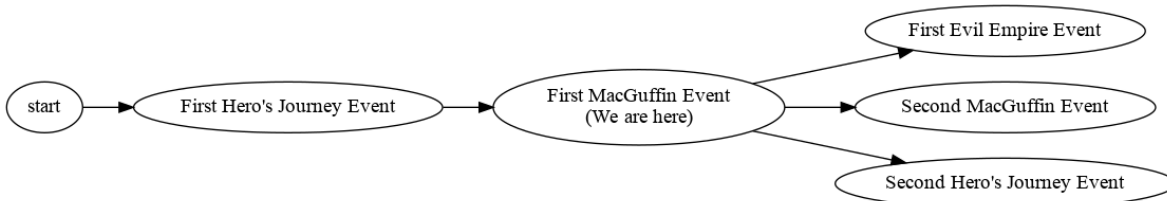


Figure 7-3: After following the *Hero's Journey*, then the *MacGuffin* tropes, there are still three paths for us to choose from.

7.3 An Example of “Character Freedom”

In this section we show an example where a character has the freedom to escape the main story path, and the potential consequences of such a scenario.

Imagine a run-through of the “Sausages scene” from *Punch and Judy* (described in Section 7.2.4 on page 156), where Punch has been given the sausages to look after, and Joey has left the stage. Now, the crocodile appears on stage and starts to chase Punch. As Punch is being pursued by the crocodile, his emotional state deteriorates, until his emotional state is “scared”. This adds the “desperate” belief to the Punch agent, which allows him to execute plans containing actions that he is not permitted to perform.

As well as being an “Idiot” in the “Mishandling of an Item” trope from the “Sausages” scene, Punch can have many other roles. In order to encourage as wide a variety of behaviours

from Punch as possible, an author can choose to attach extra roles to him. We could say that, in addition to everything else, Punch is also a “Thief”. When the “Thief” role code is included in Punch’s plan library, it adds the plans corresponding to the tropes that contain “Thief” character roles, such as the one shown in Listing 7.20.

Listing 7.20: “Theft of an Item” trope

```
1 "Theft of an Item" is a trope where:
2   The Owner is a role
3   The Thief is a role
4   The Item is an object
5   Away is a place
6   The Owner gives the Item to the Thief
7   Then the Thief goes Away
8   Then the Thief sells the Item
```

As Punch is also a “Thief”, he plans to steal the Item and escape. Until now, he has not been permitted to do so. However, now that he is in a “desperate” emotional state, he can carry out the actions from the “Theft of an Item” trope.

7.3.1 Handling Trope Violations: The Director Agent

Another, more flexible, way to enable character freedom would be through the use of a “director” agent. This agent can watch for violation events that occur, and activate tropes that closely match the roles, objects and places currently participating in the story.

So, in the “Sausages scene” example, once Punch becomes distressed, he can perform *any* of the actions described in *any* of the plans in his library. Once performed, this action would generate a violation event, and would be observed by the “director” agent. This agent can then decide which trope to activate by choosing a trope that has the same (or similar) character roles, objects and places as the currently active trope. An example of how this could be achieved is shown in the AgentSpeak code shown in Listing 7.21. The director agent could even make use of more advanced techniques such as case-based reasoning to help find a suitable trope to activate, in case no exact match can be found.

Listing 7.21: AgentSpeak code for the “Director” agent

```
1 trope(herosJourney);
2 role(hero, herosJourney);
3 place(landOfAdventure, herosJourney);
4 object(magicalAgent, herosJourney);
5
6 getSimilarTropes(X, T)
7 :- 1{trope(T) &
8     trope(X) &
9     role(R, X) &
10    role(R, T) &
11    object(O, X) &
12    object(O, T) &
13    place(P, X) &
```


Once a new trope has been activated, the agents get permission to perform the events it describes. The trope that has been violated can then be deactivated by removing permission for any of its events to occur.

7.4 Summary

This evaluation of StoryBuilder and TropICAL has shown that while users of the StoryBuilder system were able to use it for story construction, some improvements to usability, discoverability, and user feedback (such as helpful error messages) could be made. The validation of the TropICAL language showed that it has the flexibility to describe a full story (in this case, Punch and Judy) using controlled natural language trope descriptions, and an example of how the system could potentially enable character freedom is described.

The next chapter discusses the issues and limitations that arise from this evaluation in more depth, comparing our system with the existing planner-based approaches that appear in the literature, discussing the extent to which “character freedom” has been achieved, and analysing the levels of interactivity achieved against criteria based on narrative theory.

Chapter 8

Discussion

Following the evaluation of the StoryBuilder and TROPICAL systems for authoring interactive narrative, this section discusses the relative merits and limitations of these systems against existing approaches in agent-based story systems, and guidelines from narratology. The chapter begins with a comparison with the planning systems for interactive storytelling described in Section 2.2.2 on page 28, noting the similarities and comparative benefits and drawbacks our approach provides. Next follows a discussion in Section 8.2 on page 170 of the extent of “character freedom” our system provides, noting the limitations of our system in this respect, and the possible benefits of extending the system to provide more freedom for the agents to break away from the story. The chapter concludes with an analysis based on interactive criteria from narratology, and an examination of the complexity of the branching stories produced by using an answer set solver on the ASP translations of different trope examples.

8.1 Comparison with Planning Approaches

In this section, we discuss the relative merits and shortcomings of our implementation, as demonstrated in the evaluated prototype described in Section 7.1 on page 135, by comparing it with two planner-based approaches mentioned in Section 2.2.2 on page 28 of the literature review: Young’s *Mimesis* architecture (Young et al., 2004) and Mateas and Stern’s *Façade* system (Mateas and Stern, 2003).

8.1.1 Accommodation and Intervention

In implementations of Young’s architecture, such as *Mimesis* (Young et al., 2004) and *I-Storytelling* (Cavazza et al., 2002), the author specifies the situations that they would like to occur in the story through the story goals. For example, if the story were about a hostage rescue, the story goals could be to first find the hostages, and then escort them back to safety. A story planner module, given these goals and a logical description of the story world, is able to direct the intelligent agents in the story environment towards making sure these goals are achieved (by making sure that the hostages are not prematurely shot by the captor agents, for example).

However, what if the player’s actions threaten to prevent the story goals from occurring? The architecture has two strategies: accommodation (allowing the action to occur and re-planning around it) and intervention (preventing the action from taking place, or having any meaningful effect).

Compare this with the approach that our system takes. Our tropes are similar to the specification of story goals, except that they specify actions rather than goal states. The way in which they prevent story-breaking actions is similar to the *intervention* strategy of Young’s architecture: it makes sure that disallowed narrative actions are not permitted to occur. However, they may still occur if an agent is motivated enough to perform a narrative-threatening action, due to circumstances such as reaching an extreme emotional state.

The equivalent of *accommodation* in our system would be a director agent that sees the violation, and selects an appropriate trope to initiate to replace the one that has been violated. Such an agent is not present in the prototype system, however, and so further work is needed in order to implement this approach.

Given that our system, as implemented, does not offer an alternative for Mimesis’ accommodation strategy, does it offer any benefits over the system as a whole? The main motivation behind the creation of TropICAL is to ease the description of interactive story worlds for non-programmer story authors. This is achieved through the creation of a controlled natural language parser for narrative events. In the case of Mimesis, events are described as functions with parameters, such as *Go(Sam, Vault)* or *Take-out(Owner, gold, vault)*. In addition to being in a controlled natural language format, TropICAL’s stories are expressed in terms of tropes, which can contain multiple events, and can refer to other tropes. The goals expressed by story authors using Mimesis are limited to describing basic agent actions. Though the abstraction capabilities of our prototype are limited to chaining tropes at the end of existing tropes, the system still allows re-use and chaining together of tropes as abstractions.

8.1.2 Façade’s Beats and Drama Manager

On the other hand, Mateas and Stern’s *Façade* (Mateas and Stern, 2003) implements something similar to our concept of tropes through its use of *story beats*. Story beats are abstractions over multiple agent actions, describing a series of steps that the agent(s) need to perform in order to carry out a meaningful part of the story. Beat goals can refer to other beat goals as subgoals, so it is possible to create new abstractions with them. Beat goals are authored with a domain-specific language named “A Behaviour Language” (ABL). Though this language has a syntax that would be familiar to most Java or C programmers, it does not resemble natural English, and would likely take time for non-programmers to learn.

Façade uses a “drama manager” to observe the actions that are unfolding in a story and schedule a series of “beats” with appropriate pre-conditions to occur. This means that if an agent were to “break away” from the story, they would simply end up triggering and acting out an alternative set of beats. As the player can do whatever they like in the story, the “accommodation” approach is the default in this case. In comparison, our approach is restrictive, “intervening” by default. Due to the accommodation approach giving the player more

freedom to do as they wish in the story world, this arguably makes for a more entertaining experience, even though there is a resulting trade-off in story structure. Beast must have a fine level of granularity in order to make this approach effective. In contrast, a trope can have either fine or very coarse granularity, describing either a single dialogue line or an entire story. The coarser the granularity of a trope, the more difficult it should be to “break”, in order to ensure that its structure is enforced.

In summary, the primary benefit of our StoryBuilder and TropICAL tools, as demonstrated in the evaluation in Section 7.1 on page 135, is to allow non-programmer story authors to create non-linear narratives through a controlled natural language syntax. Additionally, they are able to combine pre-created narrative components from a library together in order to build their stories.

However, the implementation of our system has several shortcomings when compared with Mimesis and Façade. The main shortcoming is the lack of an implementation of a system to “re-plan” or “accommodate” story-breaking actions, such as Mimesis’ mediating story planner or Façade’s drama manager allow. Section 7.2.6 on page 157 describes how such a system could be implemented through a violation-detecting “director” institution. Another shortcoming is the lack of fine-grained control that an author can have over character agents, being limited to describing high-level goals without being able to describe the exact steps an agent would need to take to achieve them. This would still require the authoring of agent plans in a language such as AgentSpeak. Possible approaches for the generation of this code are discussed in Section 9.1.3 on page 185.

8.2 Extent of “Character Freedom” Achieved

This section addresses the issue of “Character Freedom”, introduced in Section 1.3 on page 12 of the introduction. One of the benefits of using intelligent agents for interactive narrative is that the agents have some degree of autonomy to “think for themselves”. In the context of interactive narrative, the benefit of this would be to allow the agents to generate story details for themselves, without the author having to explicitly write them into the story.

Another benefit of “character freedom” as suggested earlier is that it could add a degree of “unpredictability” to the generated story. In this section, we discuss whether such freedom has been achieved in the implemented and evaluated prototype, then whether the full system as proposed in Section 7.2 on page 153 would achieve it.

8.2.1 Prototype Implementation

The evaluation in Chapter 7 on page 135 examined the suitability of TropICAL and StoryBuilder as tools to assist non-programmer story authors in the creation of interactive narratives. The qualitative evaluation produced several themes that suggested positive results for this use case. However, in regards to its ability to give agent characters more freedom in a story world, the implementation is lacking. Though the institutions generated by TropICAL and StoryBuilder describe the permissions and obligations that apply to the character agents,

no code is generated to handle the cases where the norms are violated—the cases where the characters decide to “break away” from the suggested path of the story.

When an action occurs inside an institution that is not permitted, it triggers a violation event. In the prototype system, the violation events are being generated, but are ignored by the answer set solver due to the constraints shown in Listing 4.41 on page 94. These violations are important to keep track of, as they tell us that an agent is trying to break away from the permitted course of the story.

8.2.2 Proposed Full Implementation

Section 7.2 on page 153 describes a potential implementation of TropICAL that includes a “director” agent that watches for these violation events, and changes the violated trope to a similar alternative. In the example described, the agents are able to perform un-permitted actions if they are in certain “extreme” emotional states, which trigger violation events in the institution corresponding to the active trope.

One concern with this approach is that actions that are permitted in one institution (or trope) would not necessarily be permitted inside others. In a situation where multiple tropes are active, any action performed is likely to trigger violation events in at least one of the active institutions. The “director” agent would have to observe how many institutions have produced violation events as a consequence of an action, and only initiate a new trope when violation events are generated for all of the active tropes.

8.2.3 Advantages of Character Freedom

The prototype implementation of the storytelling system does not allow for character “freedom”, so the advantages such freedom would offer cannot be assessed based on its evaluation. If the full system described above were to be implemented, its main benefit would be the addition of a degree of unpredictability in a story, as agents would be able to break from its constraints in certain situations (such as at times of extreme emotional stress).

An intended advantage of giving the story characters more freedom is the potential for them to generate extra story details, saving the story author the work of having to write every detail of the story themselves. In practice, however, our system is too restrictive to allow this: an action is seen as either being permitted or forbidden, with no in between. In order to enable the agents to generate extra “detail” not described in the story outline, they would have to perform actions that have not been specified in it. These actions would all be violations in the system we have designed, unless specified explicitly in a trope description. This is the main limitation of our restrictive, forbidden-by-default model. Only the actions that the author specifies are allowed to occur.

In summary, though offering characters more freedom in a multi-agent system could in theory allow them to generate story details for themselves, the restrictive narrative model we are using is not an ideal fit to allow this to happen. The ability to detect and deal with events that violate a story model is useful, however, and occasionally allowing agents to perform

forbidden events can add an extra element of surprise and unpredictability to an interactive story world.

8.3 Evaluation Against Interactive Criteria

This section refers back to Section 2.1.2 on page 22 of the literature review to evaluate the interactivity of our system against the criteria defined by narrative theory research.

Aarseth's theory of Ergodic literature (Aarseth, 1997) describes the different ways in which a user can traverse a story in order to make it interactive. Aarseth's analysis identifies seven methods of story traversal: dynamics, determinability, transiency, perspective, access, linking and user functions. Each of these variables describe how the story text (the "texton") is interpreted through traversal (the "scripton", or performance of the story). Our interactive story system, as assessed through these seven dimensions, implement the seven story traversal approaches to differing degrees:

- **Dynamics** (whether or not the content and number of scriptons changes): high. If a user is simply viewing all generated permutations of a story through the StoryBuilder tool, then it would appear that there is a fixed number of scriptons. However, this only represents the story's search space. When a story is experienced through a Multi-Agent System, the actions of both the player and other intelligent agents determine the resulting *scripton*. Therefore, the scripton changes every time a story is run, meaning that the *dynamics* of the system are high.
- **Determinability** (will the same interactions result in the same scripton being produced?): high. This depends on whether there is any probabilistic behaviour on the parts of the character agents: assuming there is none, then it is indeed the case that each story would be deterministic. However, the determinability of the system could be adjusted by story authors by giving the character agents plans that carry out actions with different probabilities.
- **Transiency** (to what extent scriptons are produced as time flows, or whether user interactions are required to produce them): high. As the agents in a Multi-Agent System will carry out their plans whether or not the player takes any action, the level of *transiency* could be said to be high.
- **Perspective** (whether or not the user/reader plays a role as a character in the narrative): Yes. The system is designed so that the user can take the place of any character role in a trope. They do also have the option of not participating in the system at all, and can simply run the story as a simulation, where every character is played by an intelligent agent. In the case where the player assumes the role of a character, the interactions available to them would be determined by the permissions and obligations of that character at each point in the story.

- **Access** (whether a user has access to all scriptons at any point in the story, or whether their access is restricted): No. By design, the story unfolds according to the order of events described by the trope author. As noted above, the user’s actions are limited to their permitted and obliged actions according to the active tropes. It would not be desirable to allow them to jump to any point in the story at any time.
- **Linking** (whether or not parts of the scripton are linked to other parts, and whether these links are conditional): No. Aarseth’s analysis is mostly concerned with hypertext stories, the predominant form of interactive story at the time, so this dimension (and possibly also the “access” dimension) is difficult to apply to our agent-based storytelling system. The implementation of subtropes could be an example of linking scriptons, however, as it gives an author a method by which they can link one part of a story with another.
- **User Functions** (the functions the user uses to traverse the text: interpretive (simply traversing the text), explorative (traversing the scripton according to whim) or configurative (specifying parts of the scripton in advance)): our system is interpretive, as the user is limited to acting out parts of the scripton as interaction possibilities emerge rather than, for example, being able to jump back and forth between points in a story.

In order to compare our system to other criteria for interactive narrative, we refer once again to Chris Crawford’s definition (Crawford, 2012):

A cyclic process between two or more active agents in which each agent alternately listens, thinks, and speaks.

Due to the fact that our system is implemented using intelligent agents, our approach seems more suited to this criterion, so far as its description of alternating actions goes. However, Crawford’s main assertion is that interactive stories must be *social*, that each agent must react to the *context* of what the other agents is saying. Due to the lack of a dialogue system in our implementation, it falls far short of Crawford’s definition of an interactive narrative system. However, with the addition of topic-based agent dialogue such as we propose in Section 9.1.3 on page 186, our system could eventually meet this definition.

8.4 Complexity of Story Generation

This section investigates the complexity in terms of the number of solutions (answer sets) produced, and the time taken to produce them, given different combinations of tropes using different language features. The intention is to identify the features of TropICAL that add complexity to the story generation process, and to discover how these features could be optimised so that TropICAL can generate narratives more efficiently, with more relevant answer sets of alternative paths through the stories.

The listing examples from Section 4.3 on page 62 are used at first to isolate individual TropICAL features (Listing 8.1 to Listing 8.6 on page 176), with Listing 8.7 on page 176 to Listing 8.15 on page 178 showing the combination of several tropes together.

The answer set solver was set to generate a maximum of 500 answer sets with a maximum length of 5 events per answer set.

All tests were performed on a 2016 Lenovo Thinkpad 13 with 8GB RAM and an Intel Core i5-6200U CPU.

8.4.1 A Simple Sequence of Events

Listing 8.1: A sequence of events

```
1 The Hero goes to the Land of Adventure
2 Then the Hero finds the Sword
3 Then the Hero meets the Villain
4 Then the Hero kills the Villain
5 Then the Hero returns Home
```

The trope in Listing 8.1 is a basic sequence, five events long, one event after another.

Answer sets: 6

Time taken: 4442 msec

Already, we can see a source of inefficiency: instead of generating just one answer set for the full sequence of five events (the only full sequence possible), an answer set is generated for zero events, then the first one, then the first two, until finally an answer set for all five events is generated.

8.4.2 Basic Story Branching

Listing 8.2: Two branches

```
1 The Hero goes to the Land of Adventure
2 Or the Hero finds the Sword
```

Answer sets: 3

Time taken: 670 msec

Again, there is one more answer set than expected here: the one that contains zero events. The other two are as expected, however: one answer set per alternative branch in the trope.

8.4.3 Five Branches

Listing 8.3: Five branches

```
1 The Hero goes to the Land of Adventure
```



```
2 Or the Hero finds the Sword
3 Or the Hero meets the Villain
4 Or the Hero kills the Villain
5 Or the Hero returns Home
```

Answer sets: 6

Time taken: 1153 msec

As before, we have one answer set per story branch, with an extra answer set for a sequence with zero events. The fact that each story branch is only one event long means that we only get one answer set for every branch.

8.4.4 A Combination of Branches and Sequences

Listing 8.4: A combination of branches and sequences

```
1 The Hero goes Home
2 Then the Hero finds a Sword
3   Or the Hero goes to the Land of Adventure
4   Or the Hero kills the Villain
5 Then the Hero meets the Mentor
6   Or the Hero goes to the Realm of Mystery
```

Answers sets: 11

Time taken: 6643 msec

Answer set generation for this trope takes a little longer than previous examples, due to the extra alternatives presented by adding events after story branches. For each of the three branching possibilities in the second event of the sequence, two branching alternatives are added to the end, giving $3 \times 2 = 6$ separate branches, with each branch being three events long. Two are added for a zero-length sequence and the first event, then one for each of the first possibilities (representing sequences that are only 5 events long).

8.4.5 A Trope Containing a Subtrope

Listing 8.5: Subtrope to be embedded

```
1 "Item Search" is a trope where:
2   The Macguffin is an object
3   The Hero is a role
4   Home is a place
5
6   The Hero chases the Macguffin
7   Then the Hero finds the Macguffin
8     Or the Hero goes Home
```

Listing 8.6: Trope containing a subtrope

```
1 "Kill then Search" is a trope where:  
2   Away is a place  
3   The Hero is a role  
4   The Villain is a role  
5  
6   The Hero goes Away  
7   Then the Hero kills the Villain  
8   Then the "Item Search" trope happens
```

Answer sets: 6

Time taken: 1681 msec

In this example, as there is only one branch in the subtrope, there are two possible chains of events, each only four events long, meaning that not many answer sets are generated for this example.

8.4.6 Two Simultaneous Tropes (Without Branches)

Listing 8.7: First of two simultaneous tropes (without branches)

```
1 The Hero goes to the Land of Adventure  
2 Then the Hero finds the Sword  
3 Then the Hero kills the Villain  
4 Then the Hero goes Home
```

Listing 8.8: Second of two simultaneous tropes (without branches)

```
1 The Mentor goes to the Realm of Mystery  
2 Then the Villain goes to the Realm of Mystery  
3 Then the Villain kills the Mentor  
4 Then the Villain goes Home
```

Answer sets: 61

Time taken: 7979 msec

This example shows that the answer set solver produces 61 different answer sets when two very basic tropes are active at the same time. The reason for this is that after every event occurs, there is always a choice between the next action in the same trope, or the next action in the other active trope. This means that automatically we have $m \times n$ possible full sequences of events that can occur (where m and n are the full sequence length of each trope), and the solver is also generating all sequence lengths less than m and n in this case.

8.4.7 Two Simultaneous Tropes (With Branches)

Listing 8.9: First of two simultaneous tropes (with branches)

```
1 The Hero goes to the Land of Adventure
2 Then the Hero finds the Sword
3   Or the Hero kills the Villain
4 Then the Hero goes Home
```

Listing 8.10: Second of two simultaneous tropes (with branches)

```
1 The Mentor goes to the Realm of Mystery
2 Then the Villain goes to the Realm of Mystery
3 Then the Villain kills the Mentor
4   Or the Villain goes Home
```

Answer sets: 109

Time taken: 13289 msec

As before, adding branches to tropes greatly increases the number of answer sets produced. The effect is even more dramatic when two tropes are combined, where the solver takes over ten seconds to produce 109 answer sets.

8.4.8 Two Simultaneous Tropes (With Subtropes)

Listing 8.11: First of two simultaneous tropes (with subtropes)

```
1 The Hero goes to the Land of Adventure
2 Then the Hero finds the Sword
3 Then the Hero goes Home
4 Then the ``Item Search`` trope happens
```

Listing 8.12: Second of two simultaneous tropes (with subtropes)

```
1 The Mentor goes to the Realm of Mystery
2 Then the Villain goes to the Realm of Mystery
3 Then the Villain kills the Mentor
4 Then the ``Item Search`` trope happens
```

Answer sets: 69

Time taken: 10849 msec

Adding a subtrope does not seem to increase the complexity as much as branching in the middle of a trope. Though there is a branch in the “Item Search” trope in Listing 8.5 on page 175, which is included at the end of both active tropes here, the fact that it occurs at the end of the trope reduces the branching complexity significantly.

8.4.9 Two Simultaneous Tropes (With Subtropes and Branches)

Listing 8.13: First of two simultaneous tropes (with subtropes and branches)

```
1 The Hero goes to the Land of Adventure
2 Then the Hero finds the Sword
3   Or the Hero kills the Villain
4 Then the Hero goes Home
5 Then the ``Item Search'' trope happens
```

Listing 8.14: Second of two simultaneous tropes (with subtropes and branches)

```
1 The Mentor goes to the Realm of Mystery
2 Then the Villain goes to the Realm of Mystery
3   Or the Villain goes Home
4 Then the Villain kills the Mentor
5 Then the ``Item Search'' trope happens
```

Answer sets: 125

Time taken: 34696 msec

As expected: adding branching events to the middle of the two simultaneous tropes greatly increases the number of answer sets produced, and the time taken to produce them.

8.4.10 Three Simultaneous Tropes (With Subtropes and Branches)

Listing 8.15: Third of three simultaneous tropes (with subtropes and branches)

```
1 The Dog goes to the Forest
2 Then the Bear goes to the Forest
3   Or the Bear goes to the Den
4 Then the Bear kills the Dog
5 Then the ``Item Search'' trope happens
```

Answer sets: > 500

Time taken: 156772 msec

Of course, adding a third simultaneous trope increases the complexity of generated answer sets significantly.

It seems that the factors that have the largest effect on story complexity in these results are (from most to least significant):

1. Multiple simultaneous tropes
2. Branches in the middle of a story
3. Branches at the end of a story
4. Subtropes in a story
5. Sequences of events

8.4.11 Analysis

Based on the experiments above, the current implementation of the system is too slow for practical usage. The major source of inefficiency that emerges from this evaluation is the generation of *null* events for every possible length of event chains. This could be made much more efficient by adding a constraint to the solver that cuts out any answer sets below the maximum length of each possible sequence of events. Nothing would be lost in doing this: we can already infer from the answer sets that the events could be shorter if we simply remove events from them.

Chapter 9

Conclusions & Future Work

This chapter summarises the contributions made through the original work described in the previous chapters, and concludes with suggestions of improvements, extensions and future work to be done.

9.0.1 Summary of Contributions

The contributions of this thesis are:

- The use of story tropes to describe parts of a story using controlled natural language
- The formalisation of these tropes in social institutions.

Tropes are useful as a grammar that allows non-programmer story authors to express parts of stories informally. Social institutions gives agents more autonomy within a narrative, allowing them to break away from the story in some situations. The thematic analysis of *StoryBuilder* conducted in Section 7.1 on page 135 reveals that non-programmer story authors are able to create their own stories using tropes without having to learn a strict formalism or set of rules for story construction. They used tropes as ways to create abstractions of story patterns and embed them within other tropes. The combination of using an answer set solver with the visualisation of its output traces allowed the authors to see the effects of the changes they were making to their tropes and stories.

9.1 Future Work

The contributions of this thesis come from the use of tropes as a grammar for stories and their formalisation in social institutions. Though both techniques proved effective during the study described in Section 7.1 on page 135, some limitations and possibilities for expansion emerged during the evaluation of the system. These are addressed in Section 9.1.2 on page 183. In the first Section of this chapter, however, we address the potential future research opportunities that our trope theory and tools for non-programmer story authors create.

9.1.1 Opportunities for Future Research

What makes a trope a trope? This thesis introduces the concept of tropes for the first time into the interactive narrative literature, presenting it as a new way of describing story components. This opens up several new potential avenues of research surrounding the use and concept of tropes, both for interactive narrative and other domains.

One interesting area for research could be to explore the elements that make a trope a trope, rather than a cliché or an instance of a particular story. The existence of the TV Tropes wiki demonstrates that a community is able to compile a database of agreed-upon tropes, and identify examples of instances of these tropes in various forms of media. This suggests that tropes have features that multiple people can recognise independently, but what are these features? Are the people simply doing pattern matching across stories, or is there more to trope identification than this? A way to investigate these features would be to design a qualitative study where users are tasked with identifying tropes across several stories, and then interview them to discover how they identified the tropes, to find the lines in the stories that contained the information that caused them to identify the trope. This would assist in the discovery of elements that can be used to identify and define tropes.

Categories of trope abstractions: Another interesting study could revolve around the different levels of abstraction that tropes can take. Some tropes describe the “shape” of a story as a whole (such as the “Man in Hole” described by Vonnegut in Section 2.1.1 on page 19, or the “Hero’s Journey” trope). Is this the highest level of abstraction that a trope can describe? What would the next level of abstraction down from this be? What is the lowest level of abstraction that a trope can describe, while still being a trope? A study to identify the main categories of trope abstraction levels could involve asking participants to read several trope descriptions from the TV Tropes website, and to then sort them into categories of similar levels of abstraction. If patterns emerge from this study where the same abstraction layers seem to emerge from these trope groupings, then this could lead to a discussion about different categories of trope to describe different levels of abstraction.

Automatic Trope Analysis and Generation: Another interesting area for future research would be the automatic detection and extraction of tropes from story text. Using techniques such as Natural Language Processing (NLP), various stories could be analysed in an effort to detect common patterns and synthesise these into trope descriptions (possibly even in TropICAL). The generated tropes could then be given to human participants to see if they resemble existing tropes, or whether new tropes have been identified.

Determining the “emotional outcome” of tropes: Sentiment analysis techniques could also be applied to existing tropes to determine the emotional “direction” they make the story go: does one trope make the story build to a climax, while another turns it from comedy to tragedy? Analysing the sentiment of the start and end of a trope to see if it is “positive” or “negative”, and whether its polarity is reversed or remains the same at the end could help to

identify the part of a story “shape” it belongs to (or where it lies on “Freytag’s Pyramid” from Section 3.2.1 on page 50, for example).

Generating stories from tropes: Instead of analysing existing tropes using NLP, we could use NLG (Natural Language Generation) to generate free-text, natural language stories from TropICAL trope definitions. Using techniques such as Markov Chains, stories could be randomly generated to conform to a trope template, in the style of Talespin (Meehan, 1977) or Minstrel (Turner, 1993), both described in Section 2.2.1 on page 26. Rather than being used for dynamic, interactive narrative systems in a Multi-Agent System story world, tropes could instead be used to add structure to static, unchanging stories that are generated as the result of a computational process.

Human-Computer Interaction implications: The creation of tools such as TropICAL and StoryBuilder, which aim to enable non-programmers to author non-linear and interactive stories, has some interesting implications for Human-Computer Interaction (HCI) research. Further user studies could be designed to examine the needs of story authors when designing and creating non-linear narratives. Are visualisations of the story branches, such as the one that StoryBuilder provides, useful in the construction of such stories? How could they be improved? Would it help the visualisation if certain branches could be collapsed (hidden) and others shown? Would it be useful to add auto-suggestion features to the TropICAL language editor? Studies could be carried out to test these tools on users in order to elicit concrete User Interface improvements, which could then be implemented and compared in further studies.

Tropes and learning: Another interesting set of HCI studies could be to investigate the potential of desired interactive narrative for learning. Schank (1990) points out that humans intuitively think in terms of stories, and as a consequence learn facts more effectively when they are presented in the form of a narrative. This being the case, would it also be the case that certain types of stories are more suitable to learn from than others? Perhaps stories involving danger would shock the primitive parts of our brains into paying attention to their details more closely. One way to test this would be to study the recall abilities of participants when presented facts embedded within different types of tropes in stories. This type of study could be used to determine whether certain elements of tropes aid recall better than others.

Another study could compare the effects on recall of linear and non-linear or interactive narratives. Does a reader remember the details of a story better if they have some say in determining its course and outcome?

In summary, the potential research that emerges from this thesis are:

- How to identify the elements of a trope (what makes a trope a trope?).
- How to identify the main levels of abstraction of the story fragments that tropes describe.
- How to use Natural Language Processing to automatically extract tropes from story text.

- How to use Natural Language Generation to automatically generate story text from trope descriptions.
- How to use sentiment analysis to determine the “emotional outcome” of a trope.
- Discovering how to design better tools and user interfaces for interactive narrative authoring.
- Investigation to see whether certain tropes are more suitable as parts of stories design to aid learning.

9.1.2 Addressing Limitations in *TropICAL* and StoryBuilder

During the study described in Section 7.1 on page 135, users highlighted the following limitations of the StoryBuilder system:

Limitation 1: **Five event limit** - StoryBuilder was limited to producing answer-set “stories” of a maximum of five events in length to reduce the time needed to generate complex, multi-trope stories

Limitation 2: **Branch length limit** - Every time branching possibilities were created using the *Or* keyword, each branch could only be one event long before merging back into the parent branch of the story.

Limitation 3: **Subtrope at end** - Subtropes could only be embedded at the end of a trope, in place of the last event that occurs.

Limitation 4: **Error messages** - Error messages in StoryBuilder were vague and did not assist users in locating bugs in their TropICAL code.

Limitation 1 can be addressed by increasing the number of time steps that the answer set solver solves for, but complexity laws mean that the answer set solver will always take longer to solve for multiple tropes. More work is needed in refining the generated institutions, along with the constraint rules that are input into the solver, in order to make sure less answer sets are generated as a whole. An example of one way to approach this would be to remove the answer sets that contain any *null* events. This means that only answer sets of length n (where n is the maximum time step specified) would be produced. Each of these answer sets would contain the answer sets that are shorter than n as a subset of its events.

To solve Limitation 2, Fluents could be generated in the InstAL institution that keep track of the branch that the simulation is following, along with its phase. While we currently have “phases” for a trope as a whole, additional phases need to be added for each branch. This would require adding phase fluents for each branching point in the story, for example: `branchPhase(branch1, phaseA)`. Managing state in this way is quite procedural and awkward, however. Future work would include the creation of an alternative to the “phase” mechanism proposed for state management in institutions. For example, rather than storing the state of

a trope in a fluent, the same state could be inferred by checking to see which norms hold at a certain point in time.

Limitation 3 is an issue in the implementation of the prototype used for the study, and would only require time to be spent debugging the compiler to produce the correct output.

Limitation 4 could be fixed by referring back to the line of TROPICAL code that produces the error. This would involve embedding extra information (such as line numbers) in the output of the TROPICAL parser.

Users also suggested some improvements that could be made to the system:

Improvement 1: **Event with Three Entities** - Users want to be able to express statements involving a character and two other entities, such as “*The Mentor gives the Sword to the Hero*”, or “*The Villain tells the Hero to go Home*”.

Improvement 2: **Event Semantics** - Users want events to have consequences in the story world. For example, when a character dies, all further permissions for them to do anything should terminate.

Improvement 3: **Other Types of Entities** - Users want to have entities other than *roles*, *objects* and *places* in their tropes

Improvement 4: **Better Control Over Story Structure** - Users want to be able to arrange tropes in a specific sequence.

Improvement 5: **Conditionals** - Users want to be able to express events as preconditions with multiple events occurring as consequences.

Improvement 1 would require extending the parser to deal with verbs such as *give* or *tell* that have an “arity” of two (they refer to two entities). As the entities are declared at the start of each trope file, it would be simple to add this functionality to the parser by having it count the number of entities mentioned in a statement.

Improvement 2 would not be trivial to implement. One possible approach would be to use a tool such as ConceptNet (Liu and Singh, 2004) to identify key verbs that have meanings that could limit agent behaviour in a trope. If a character is *killed*, *defeated* or *murdered* for example, ConceptNet would identify that these verbs have the same meaning, and the consequences of this could be compiled into the resulting institution. In this case, all subsequent permissions for the character would be terminated. Another interesting example would be if a character could *become omnipotent*, granting them permission to do anything in the story world.

Improvement 3 would be implemented by extending the parser to allow other types of entities to be declared in a trope definition.

Improvement 4 is an improvement that would require implementation in StoryBuilder rather than TROPICAL. A drag-and-drop interface that allows users to select multiple tropes and arrange them in a graph would allow users to sequence the tropes so that they occur in any order that the user wishes. This would result in the generation of extra institutional

bridges that link the final event of one trope in the sequence to the first event in the next trope, so that permission for the first event in *Trope B* to occur is initiated when the final event in *Trope A* happens.

Improvement 5 would require extending the parser to look for an *if* keyword, with preconditions and consequences that follow it. These would then be compiled to the corresponding preconditions and consequences for generating an institutional event in the InstAL institution.

While this section described how the current implementation of TropICAL and StoryBuilder could be improved, the next section examines additional systems that could be authored to complement StoryBuilder and TropICAL, as well as how they can be applied to domains outside of interactive narrative.

9.1.3 Methods for Character Agent Authoring

TropICAL and StoryBuilder are tools that allow an author to create complex non-linear narratives. However, they are designed to be a single component of a system that also includes character agents that have their own behaviour defined by pre-written plans. The answer sets that emerge from the AnsProlog formalisation of TropICAL’s tropes only describe the constraints (in terms of norms) on the agents’ behaviour.

Our current implementation of the system integrates with the Jason (Bordini et al., 2007) multi-agent framework, which requires each agent plan library to be written in a language named “AgentSpeak”. The declarative paradigm that this language uses, while familiar to many Computer Science researchers, would take time for programmers familiar with today’s dominant paradigms (object-oriented, procedural, functional) to learn. For this reason, it would be even more challenging for non-programmer story authors to use. A simpler way to program the agents would enable authors to create the characters agent that fit into their TropICAL story without spending considerable extra time learning a new programming language.

Agent actions in a narrative would vary greatly according to the type of game world that they inhabit. An action game where the player visits many different locations to battle other characters without talking to them would be populated with agents that behave very differently from a dialogue-focused murder mystery game set in a country manor. *TopicScript*, the example language we describe in the next section, is a domain-specific language designed for the authoring of character dialogue for agents with an emotional model. The inspiration for its syntax comes from Inkle Studios’ *ink* programming language¹, which is also a language for describing branching dialogue trees. *TopicScript* is designed with the authoring of primarily dialogue-focused games in mind, aiming to allow authors to write branching story scripts for their agents as quickly as possible.

¹Described at <https://www.inklestudios.com/ink/>, accessed 20171025.

Agent Dialogue Authoring for Interactive Narrative

An important part of any narrative-based story is the creation of engaging character dialogue. When creating an interactive narrative using a multi-agent system framework such as Jason (described in Section 6.2 on page 125), these lines of dialogue would have to be placed at different points throughout the agents' plans. This would be a difficult system for non-programmer story authors to work with, so a domain-specific language similar to TropICAL could be devised to allow the author to create their story dialogue with a controlled natural language syntax. This language would then compile to AgentSpeak code (for example) to add the lines of dialogue to the agents. This section outlines an example of how such a language could be designed.

One way of authoring character dialogue to enable non-linear conversations would be to adjust the dialogue according to character emotion. When authoring character dialogue, their response to a topic can depend on their emotional state at the time. For example, asking a character about themselves will elicit a slightly different response if she is angry or if she is happy.

Our *TopicScript* programming language is designed to allow for fast authoring of this kind of affective dialogue. Its grammar is as follows:

```
1 <SUBJECT>: <TOPIC>: <QUESTION>
2   <EMOTION>: <RESPONSE>
3   <EMOTION>: <RESPONSE>
4   <EMOTION>: <RESPONSE>
5   <EMOTION>: <RESPONSE>
```

The subject is the object, place or character that is being asked about. The topic is some specific question that is being asked about the subject. For example, the player could ask about the crown in a painting of a king. In this case, the crown would be the topic and the painting would be the subject.

Template for general topics: Everything under General is character dialogue when there is no topic of conversation. It can be used as a template (for example) when a character is talking about something that is not in the database. This is useful as a kind of “fallback” option when the character is prompted to speak about something that it has no lines of dialogue for.

For example, if a character is happy and looking at something, they can say “Nice <something>!”. The <something> is represented with a `_` in the language.

Inserting detail about topics/objects into the dialogue: Each object/place/character in a story may have a string of descriptive text. This can be inserted into a character's dialogue with the [detail] placeholder. If this is used, but no description string exists, the character can have a (customizable) response such as “I don't know anything about it.”.

Example:

```
1 General: What's up?
2 Angry: You cad!
```

```

3   Angry _: I'm angry about the _, of course!
4   Surprised: I can't believe what's happening!
5   Happy, delighted _: The _ is the best!
6   Neutral, general: Not much.
7   Neutral, general _: Nice _.
8
9   Weather: How's the weather today?
10  Delighted: The weather sure is nice!
11  Neutral, general: Nice weather.
12
13  Painting of Richard III: subject: Who's that in the painting?
14  Tired, sad, unhappy: I have no idea.
15  Vicious, annoyed: It's Queen Elizabeth I.
16  Malicious: Why, that's my granddad!
17  General: It's King Richard III. [detail]
18
19  Painting of Richard III: crown: Is that a hat he's wearing?
20  Neutral, general: No, it's a crown. [detail]
21  Angry: No, you fool: it's a crown! [detail]
22  Scared: Be careful what you say! That's a crown he's wearing! [detail]

```

The language described above is designed around the concept of an agent’s dialogue adjusting to their emotions, but other variables could be taken into account. Instead of an emotional identifier, a line of dialogue could be labeled with an event (or series of events) that could have taken place, or fluents that hold, for example.

Roles and Emotional Traits

Another enhancement that could be made our storytelling system would be to allow a character’s role to determine their emotional states. Though TropicAl allows story authors to give roles to the characters in their tropes, the roles assigned to the agents do not affect their behaviour in any way. If the intelligent agents implement an emotional model such as the “Pleasure, Arousal, Dominance” model described in Section 6.2.1 on page 125, then their character role could determine their emotional state at the start of a story or scene. For example, a “Hero” character could start a story with high dominance, arousal and valence values (resulting in an “excited” emotion), while a “Villain” would have high dominance and arousal but low valence (resulting in a “furious” emotional state).

Furthermore, stories contain many different types of characters, each of which can implement one or more role archetypes. Often these archetypes inherit characteristics from one or more roles. For example, TV Tropes lists 224 different types of Hero, including the Anti-Hero (which itself has 9 sub-types of anti-hero, including the “Pragmatic Hero” and “Sociopathic Hero”), “The Fool”, “The Gunslinger”, “Right Man in the Wrong Place” and “Science Hero”. Many of these roles not only inherit traits from the “Hero” archetype, but also from other types of role, such as “The Sociopath” and “The Pragmatist”.

Many opportunities exist in the combining of all of the components described here into a complete IDE for story authoring. The story structure could be written in TropicAl,

visualised in StoryBuilder, and the dialogue described in TopicScript, with a character’s role determining their initial emotional state. The extension of the StoryBuilder IDE to allow for rapid prototyping and testing of the resulting multi-agent simulation would be another way to make the system easier to use for story authors.

9.1.4 Application of *TropICAL* to Other Domains

Due to the fact that tropes describe recurring patterns of behaviour, the concept can be applied to domains outside of storytelling where patterns can be described.

This section describes a worked example where TropICAL is used to compose legal contracts through the use of tropes to describe re-usable “policies”. Once compiled to a formal representation, these contracts can be verified using an answer set solver, in the same way in which we check and generate stories in Section 4.5 on page 94.

Legal Policy Descriptions

Though TropICAL is designed to describe non-linear stories in systems with intelligent agents such as interactive computer games, with minor changes it can also be adjusted to describe legal contracts. Where a story describes the constraints on character behaviour that any character can in theory break away from, a legal contract performs a similar role by describing the expected behaviour of its parties and the consequences for breaking those expectations.

In the case of contract law, though each individual contract between plaintiff and defendant would be different, common patterns exist between contracts. For example, contracts typically have a fixed duration and penalty for premature termination. There are usually clauses in a contract that describe the conditions for termination in the case of one of the parties performing certain unpermitted actions. These are what we refer to as a “policy” throughout this section: an abstract description of a commonly-occurring legal pattern that is analogous to a trope in a story.

In order to divide legal contracts into reusable components, the concept of tropes can easily be applied to capture fragments of the law. Some examples are:

- **The Warranty:** A *seller* sells an item to a *buyer*. If the item is defective, then the *buyer* has the right to return it within a certain period of time.
- **The Lease:** A *lessor* leases an item or property to a *lessee*. The lessee is obliged to pay rental fees on time, and to keep the item or property in good condition. The lessor is obliged to perform maintenance and necessary repairs on the item or property.
- **The Deposit:** A sum of money is given from one party to another with the understanding that it is to be returned upon expiration of a contract.

As with tropes that contain sub-tropes, policies can have sub-policies. For example, a “sales contract” policy could contain a “warranty” policy as a sub-policy. This can be thought of as adding a clause to a contract. Similarly, a “lease” policy could be a sub-policy of a “tenancy

agreement” policy, which keeps the spirit of the lease policy but adds the requirement for the lessee to not make excessive noise, for example. These policies describe the actions that both parties are permitted and obliged to carry out, in a sequence of events.

As an example of describing a legal policy in the manner of a story trope, consider the subleasing of an object or property from a lessor to a lessee. In such an arrangement, the following dispute could occur:

- The *lessor* leases property to the *lessor*.
- The *lessee* then subleases the property to a *third party*.
- The *lessor* cancels the lease contract, stating that the subleasing is a violation of the contract.

These policies can be expressed in terms of social norms, and translated into TropicAL code:

Listing 9.1: Example policies in TropicAL

```
1  ``Lease'' is a policy where:
2      The Lessor is a role
3      The Lessor is a role
4      The Thing is an object
5      The Lessor leases the Thing to the Lessee
6      Then the ``Maintenance of Confidence'' policy applies
7
8  ``Sublease'' is a policy where:
9      The Lessor is a role
10     The Third Party is a role
11     The Thing is an object
12     The Lessor may sublease the Thing to a Third Party
13
14  ``Sublease Permission'' is a policy where:
15     The Lessee is a role
16     The Lessor is a role
17     The Thing is an object
18     The Lessee may ask permission to sublease the Thing
19     If the Lessor gives permission to the Lessee:
20         The ``Sublease'' policy applies
21
22  ``Maintenance of Confidence'' is a policy where:
23     The Lessee is a role
24     The Lessor is a role
25     The Due Date is an object
26     The Lease is an object
27     The Thing is an object
28     The Lessee must pay the Lessor before the Due Date arrives
29     Otherwise, the Lessor may cancel the Lease
30     The Lessor must maintain the Thing
31     Otherwise, the Lessee may cancel the Lease
```

The use of TROPICAL to describe legal policies would be useful for lawyers that want to validate or generate arguments from any given set of policies. This would be valuable as a tool to aid lawyers in argument creation, for example. Suppose a defense lawyer is trying to argue that her client is not guilty. The system could generate traces (sequences of events) of a given length, whose outcome does not result on the violation of any contract or law. The lawyer would then be able to choose from amongst a list of generated arguments to make their case.

Adapting this idea to our *sublease* example described above, we can generate sequences of events where a lessee has subleased a property, and find out whether or not their actions have been in violation of a policy. We achieve this through the specification of constraints. For example, a prosecution lawyer wishing to find all sequences of events where both sublease event and violation events have occurred would specify this ASP constraint:

```
1 violEvent :- occurred(viol(X), I, T).
2 :- not violEvent.
3 subleaseEvent :- occurred(sublease(X, Y, Z), I, T), holdsat(role(X, lessee),
4 I, T).
5 :- not subleaseEvent.
```

This constraint specifies that we want to generate all possible sequences where the lessee has subleased something, and where a violation has occurred. Headless rules in ASP (where the leftmost part of the rule is simply “:-”) state what we do *not* want in generated answer sets, so in this example both headless statements are double negatives.

The solver outputs several answer sets, containing event sequences (traces) of a specified length. We can then parse these answer sets into a human-readable “executive summary” that can be used by lawyers. An example of such a summary would be:

```
1 Possibility 0:
2
3 The following occurred:
4 Alice Leases Bob House
5 Then:
6 Bob Subleases Charlotte House
7 VIOLATION: Bob Subleases Charlotte House
```

Each summary lists a number of “possibilities”, with each possibility corresponding to an answer set produced by the solver, listing events and violations that occur in a trace. To find an argument, a lawyer can simply read through the generated possibilities to find one that suits her needs.

Though we describe here the adaptation of TROPICAL to the context of legal policies, it could potentially be applied to any domain in which patterns can be identified, described in controlled natural language, and reused. TROPICAL could also be used, for example, to describe the actions of a house-cleaning robot, delivery drone, personal assistant chatbot or automated hotel-booking agent. The simple, pattern-based nature of tropes make them easily adaptable to a range of implementations.

9.1.5 Concluding Remarks

There is great potential for the use of intelligent agents in narrative-based interactive entertainment. This potential has yet to be realised however, with very few forms of interactive entertainment being released that take advantage of the possibilities that multi-agent systems offer.

A possible explanation for the lack of agents in entertainment is the requirement for creators to learn how to use experimental technology and development techniques in order to use the agents. Agent-oriented games such as *Façade* have only emerged from research groups that are already familiar with agent technology. There is a gulf between the world of research and practice in the realm of interactive narrative.

The development of approachable tools such as *TropICAL* and *StoryBuilder* is one way to bridge this gulf. We hope that the contributions described in this dissertation are able to find their way out into the wider world in some form so that they can enable the creation of more interactive narrative content.

Appendix A

London Interactive Fiction Meetup Questionnaire

What's your interest in Interactive Fiction?

- I'm an author: 5 (27.8%)
- I'm a game developer: 10 (55.6%)
- I write interactive fiction: 5 (27.8%)
- It's my hobby: 6 (33.3%)
- It's my job: 5 (27.8%)
- Other: 2 (11.1%)

What tools do you use to create Interactive Fiction?

- Inform: 3 (16.7%)
- Twinery: 7 (38.9%)
- Unity or other IDE: 8 (44.4%)
- Pure code: 4 (22.4%)
- I don't create interactive fiction or games with narrative: 3 (16.7%)
- Other: 6 (33.3%)

What kind of narratives are you interested in making?

- Linear: 2 (11.1%)
- Non-linear: 11 (61.1%)
- I'm not an author: 1 (5.6%)

- Other: 4 (22.2%)

Are you familiar with the idea of “tropes”?

- Yes, and I have visited the “TV Tropes” website: 15 (83.3%)
- Yes, but I hadn’t heard of “TV Tropes”: 3 (16.7%)
- No: 0 (0%)

How useful do you think tropes are for authoring interactive stories? (on a scale of 1 - 5)

- 1 (not useful): 0 (0%)
- 2: 2 (11.8%)
- 3: 9 (52.9%)
- 4: 2 (11.8%)
- 5 (extremely useful): 4 (23.5%)

Appendix B

Full “Don’t Touch It, You Idiot” Institution

This appendix lists the full TropICAL code for the “Don’t Touch It, You Idiot” trope, along with the institutional model it is derived from.

Listing B.1: Full InstAL code for the “Don’t Touch It, You Idiot” trope, compiled from TropICAL

```
1 institution dontTouchItYouIdiot;
2 % TYPES -----
3 type Identity;
4 type Agent;
5 type Role;
6 type Trope;
7 type Phase;
8 type Place;
9 type PlaceName;
10 type Object;
11 type ObjectName;
12
13 % FLUENTS -----
14 fluent role(Agent, Role);
15 fluent phase(Trope, Phase);
16 fluent place(PlaceName, Place);
17 fluent object(ObjectName, Object);
18
19
20 % EXTERNAL EVENTS: Dont Touch it You Idiot -----
21 exogenous event break(Agent, ObjectName);
22 exogenous event drop(Agent, ObjectName);
23 exogenous event go(Agent, PlaceName);
24 exogenous event return(Agent);
25 exogenous event fight(Agent, Agent);
26 exogenous event noDeadline;
27
28 % VIOLATION EVENTS: Dont Touch it You Idiot -----
```

```

29 violation event noViolation;
30
31 % INST EVENTS: Dont Touch it You Idiot -----
32 inst event intFight(Agent, Agent);
33 inst event intDrop(Agent, ObjectName);
34 inst event intReturn(Agent);
35 inst event intBreak(Agent, ObjectName);
36 inst event intGo(Agent, PlaceName);
37 inst event intDontTouchItYouIdiot(Agent, Agent, ObjectName, PlaceName);
38 inst event intNoDeadline;
39
40
41
42 % INITIATES: Dont Touch it You Idiot -----
43 intDontTouchItYouIdiot(R, S, T, U) initiates
44     phase(dontTouchItYouIdiot, phaseA),
45     perm(go(R, U)) if
46         phase(dontTouchItYouIdiot, active),
47         role(R, owner),
48         place(U, away);
49 intDontTouchItYouIdiot(R, S, T, U) initiates
50     phase(dontTouchItYouIdiot, phaseB),
51     perm(break(S, T)) if
52         phase(dontTouchItYouIdiot, phaseA),
53         role(S, idiot),
54         object(T, item);
55 intDontTouchItYouIdiot(R, S, T, U) initiates
56     phase(dontTouchItYouIdiot, phaseC),
57     perm(return(R)) if
58         phase(dontTouchItYouIdiot, phaseB),
59         role(R, owner);
60 intDontTouchItYouIdiot(R, S, T, U) initiates
61     phase(dontTouchItYouIdiot, phaseD),
62     perm(fight(R, S)) if
63         phase(dontTouchItYouIdiot, phaseC),
64         role(S, idiot),
65         role(R, owner);
66 % TERMINATES: Dont Touch it You Idiot -----
67 intDontTouchItYouIdiot(R, S, T, U) terminates
68     phase(dontTouchItYouIdiot, active),
69     perm(drop(R, T)) if
70         phase(dontTouchItYouIdiot, active),
71         role(R, owner),
72         object(T, item);
73 intDontTouchItYouIdiot(R, S, T, U) terminates
74     phase(dontTouchItYouIdiot, phaseA),
75     perm(go(R, U)) if
76         phase(dontTouchItYouIdiot, phaseA),
77         role(R, owner),
78         place(U, away);
79 intDontTouchItYouIdiot(R, S, T, U) terminates

```

```

80     phase(dontTouchItYouIdiot, phaseB),
81     perm(break(S, T)) if
82         phase(dontTouchItYouIdiot, phaseB),
83         role(S, idiot),
84         object(T, item);
85 intDontTouchItYouIdiot(R, S, T, U) terminates
86     phase(dontTouchItYouIdiot, phaseC),
87     perm(return(R)) if
88         phase(dontTouchItYouIdiot, phaseC),
89         role(R, owner);
90 intDontTouchItYouIdiot(R, S, T, U) terminates
91     phase(dontTouchItYouIdiot, phaseD),
92     perm(fight(R, S)) if
93         phase(dontTouchItYouIdiot, phaseD),
94         role(S, idiot),
95         role(R, owner);
96
97
98 % GENERATES: Dont Touch it You Idiot -----
99 fight(R, S) generates
100     intDontTouchItYouIdiot(R, S, T, U) if
101         role(S, idiot),
102         role(R, owner);
103 return(R) generates
104     intDontTouchItYouIdiot(R, S, T, U) if
105         role(R, owner);
106 go(R, U) generates
107     intDontTouchItYouIdiot(R, S, T, U) if
108         role(R, owner),
109         place(U, away);
110 break(S, T) generates
111     intDontTouchItYouIdiot(R, S, T, U) if
112         role(S, idiot),
113         object(T, item);
114 drop(R, T) generates
115     intDontTouchItYouIdiot(R, S, T, U) if
116         role(R, owner),
117         object(T, item);
118
119 % INITIALLY: -----
120 initially
121     pow(intDontTouchItYouIdiot(R, S, T, U)) if role(R, owner), role(S, idiot)
122         , object(T, item), place(U, away);
123 initially
124     perm(intDontTouchItYouIdiot(R, S, T, U)) if role(R, owner), role(S, idiot
125         ), object(T, item), place(U, away);
126 initially
127     perm(drop(R, T)) if role(R, owner), object(T, item);
128 initially
129     phase(dontTouchItYouIdiot, active),
130     role(joey, owner),

```

```

129     role(punch, idiot),
130     place(offstage, away),
131     object(sausages, item);

```

$$\mathcal{D} = \{\text{owner, idiot, item, away, inactive, phaseA, phaseB, phaseC, phaseD, phaseE, done}\}$$

Figure B-1: Domain fluents for the “Don’t Touch It, You Idiot” trope

$$\mathcal{W} = \{\text{pow(intDontTouchItYouIdiot(owner, idiot, item, away))}\}$$

Figure B-2: Empowerments for the “Don’t Touch It, You Idiot” trope

$$\mathcal{P} = \{\text{perm(drop(owner, item)), perm(go(owner, away)), perm(break(idiot, item)), perm(return(owner)), perm(fight(owner, idiot))}\}$$

Figure B-3: Permissions for the “Don’t Touch It, You Idiot” trope

$$\mathcal{E}_{\text{external}} = \left\{ \begin{array}{l} \text{drop(owner, item),} \\ \text{go(owner, away),} \\ \text{break(idiot, item),} \\ \text{return(owner),} \\ \text{fight(owner, idiot)} \end{array} \right\} \quad (\text{B.1})$$

$$\mathcal{E}_{\text{internal}} = \left\{ \begin{array}{l} \text{intDontTouchItYouIdiot(owner,} \\ \text{idiot, item, away)} \end{array} \right\} \quad (\text{B.2})$$

Figure B-4: External and institutional events (\mathcal{E}) for the “Don’t Touch It, You Idiot” trope

$$\mathcal{G}(\mathcal{X}, \mathcal{E}) : \left\{ \begin{array}{l}
\langle \text{drop}(\text{owner}, \text{item}) \rangle \\
\rightarrow \{ \text{intDontTouchItYouIdiot}(\text{owner}, \text{idiot}, \text{item}, \text{away}) \} \quad (\text{B.3}) \\
\langle \text{go}(\text{owner}, \text{away}) \rangle \\
\rightarrow \{ \text{intDontTouchItYouIdiot}(\text{owner}, \text{idiot}, \text{item}, \text{away}) \} \quad (\text{B.4}) \\
\langle \text{break}(\text{idiot}, \text{item}) \rangle \\
\rightarrow \{ \text{intDontTouchItYouIdiot}(\text{owner}, \text{idiot}, \text{item}, \text{away}) \} \quad (\text{B.5}) \\
\langle \text{return}(\text{owner}) \rangle \\
\rightarrow \{ \text{intDontTouchItYouIdiot}(\text{owner}, \text{idiot}, \text{item}, \text{away}) \} \quad (\text{B.6}) \\
\langle \text{fight}(\text{owner}, \text{idiot}) \rangle \\
\rightarrow \{ \text{intDontTouchItYouIdiot}(\text{owner}, \text{idiot}, \text{item}, \text{away}) \} \quad (\text{B.7})
\end{array} \right.$$

Figure B-5: Event generation in the “Don’t Touch It, You Idiot” trope

$$\mathcal{C}^\uparrow(\mathcal{X}, \mathcal{E}) : \left\{ \begin{array}{l}
\langle \{ \text{phase}(\text{dontTouchItYouIdiot}, \text{inactive}) \}, \\
\text{intDontTouchItYouIdiot}(\text{owner}, \text{idiot}, \text{item}, \text{away}) \rangle \rightarrow \\
\{ \text{phase}(\text{dontTouchItYouIdiot}, \text{phaseA}), \\
\text{perm}(\text{drop}(\text{owner}, \text{item})) \} \quad (\text{B.8}) \\
\langle \{ \text{phase}(\text{dontTouchItYouIdiot}, \text{phaseA}) \}, \\
\text{intDontTouchItYouIdiot}(\text{owner}, \text{idiot}, \text{item}, \text{away}) \rangle \rightarrow \\
\{ \text{phase}(\text{dontTouchItYouIdiot}, \text{phaseB}), \\
\text{perm}(\text{go}(\text{owner}, \text{away})) \} \quad (\text{B.9}) \\
\langle \{ \text{phase}(\text{dontTouchItYouIdiot}, \text{phaseB}) \}, \\
\text{intDontTouchItYouIdiot}(\text{owner}, \text{idiot}, \text{item}, \text{away}) \rangle \rightarrow \\
\{ \text{phase}(\text{dontTouchItYouIdiot}, \text{phaseC}), \\
\text{perm}(\text{break}(\text{idiot}, \text{item})) \} \quad (\text{B.10}) \\
\langle \{ \text{phase}(\text{dontTouchItYouIdiot}, \text{phaseC}) \}, \\
\text{intDontTouchItYouIdiot}(\text{owner}, \text{idiot}, \text{item}, \text{away}) \rangle \rightarrow \\
\{ \text{phase}(\text{dontTouchItYouIdiot}, \text{phaseD}), \\
\text{perm}(\text{return}(\text{owner})) \} \quad (\text{B.11}) \\
\langle \{ \text{phase}(\text{dontTouchItYouIdiot}, \text{phaseD}) \}, \\
\text{intDontTouchItYouIdiot}(\text{owner}, \text{idiot}, \text{item}, \text{away}) \rangle \rightarrow \\
\{ \text{phase}(\text{dontTouchItYouIdiot}, \text{phaseE}), \\
\text{perm}(\text{fight}(\text{owner}, \text{idiot})) \} \quad (\text{B.12}) \\
\langle \{ \text{phase}(\text{dontTouchItYouIdiot}, \text{phaseE}) \}, \\
\text{intDontTouchItYouIdiot}(\text{owner}, \text{idiot}, \text{item}, \text{away}) \rangle \rightarrow \\
\{ \text{phase}(\text{dontTouchItYouIdiot}, \text{done}) \} \quad (\text{B.13})
\end{array} \right.$$

Figure B-6: Phase fluent initiation in the “Don’t Touch It You Idiot” trope

$$\mathcal{C}^\downarrow(\mathcal{X}, \mathcal{E}) : \left\{ \begin{array}{ll}
\langle \{\text{phase}(\text{dontTouchItYouIdiot}, \text{inactive})\}, \\
\text{intDontTouchItYouIdiot}(\text{owner}, \text{idiot}, \text{item}, \text{away}) \rangle \rightarrow \emptyset & \text{(B.14)} \\
\langle \{\text{phase}(\text{dontTouchItYouIdiot}, \text{phaseA})\}, \\
\text{intDontTouchItYouIdiot}(\text{owner}, \text{idiot}, \text{item}, \text{away}) \rangle \rightarrow \emptyset & \text{(B.15)} \\
\langle \{\text{phase}(\text{dontTouchItYouIdiot}, \text{phaseB})\}, \\
\text{intDontTouchItYouIdiot}(\text{owner}, \text{idiot}, \text{item}, \text{away}) \rangle \rightarrow \emptyset & \text{(B.16)} \\
\langle \{\text{phase}(\text{dontTouchItYouIdiot}, \text{phaseC})\}, \\
\text{intDontTouchItYouIdiot}(\text{owner}, \text{idiot}, \text{item}, \text{away}) \rangle \rightarrow \emptyset & \text{(B.17)} \\
\langle \{\text{phase}(\text{dontTouchItYouIdiot}, \text{phaseD})\}, \\
\text{intDontTouchItYouIdiot}(\text{owner}, \text{idiot}, \text{item}, \text{away}) \rangle \rightarrow \emptyset & \text{(B.18)} \\
\langle \{\text{phase}(\text{dontTouchItYouIdiot}, \text{phaseE})\}, \\
\text{intDontTouchItYouIdiot}(\text{owner}, \text{idiot}, \text{item}, \text{away}) \rangle \rightarrow \emptyset & \text{(B.19)}
\end{array} \right.$$

Figure B-7: Phase fluent termination in the “Don’t Touch It, You Idiot” trope

Appendix C

Generated InstAL Code

Listing C.1: An obligation event with no deadline or consequence events

```
1 institution obligation1;
2 % TYPES -----
3 type Identity;
4 type Agent;
5 type Role;
6 type Trope;
7 type Phase;
8 type Place;
9 type PlaceName;
10 type Object;
11 type ObjectName;
12
13 % FLUENTS -----
14 fluent role(Agent, Role);
15 fluent phase(Trope, Phase);
16 fluent place(PlaceName, Place);
17 fluent object(ObjectName, Object);
18
19
20 % EXTERNAL EVENTS: Obligation 1 -----
21 exogenous event go(Agent, PlaceName);
22 exogenous event noDeadline;
23
24 % VIOLATION EVENTS: Obligation 1 -----
25 violation event violHeroGoLandOfAdventure;
26 violation event noViolation;
27
28 % INST EVENTS: Obligation 1 -----
29
30 inst event intGo(Agent, PlaceName);
31 inst event intObligation1(Agent, PlaceName, PlaceName);
32 inst event intNoDeadline;
33
34 % OBLIGATION FLUENTS: Obligation 1 -----
```

```

35 obligation fluent obl(intGo(Agent, PlaceName), intNoDeadline, noViolation);
36
37 % INITIATES: Obligation 1 -----
38 intObligation1(R, S, T) initiates
39     phase(obligation1, phaseA),
40     obl(intGo(R,T), intNoDeadline, noViolation), perm(go(R, T)), perm(intGo(R
41         ,T)), pow(intGo(R,T)) if
42         phase(obligation1, active),
43         role(R, hero),
44         place(T, landOfAdventure);
45 % TERMINATES: Obligation 1 -----
46 intObligation1(R, S, T) terminates
47     phase(obligation1, active),
48     perm(go(R, S)) if
49         phase(obligation1, active),
50         role(R, hero),
51         place(S, home);
52 intObligation1(R, S, T) terminates
53     phase(obligation1, phaseA),
54     obl(intGo(R,T), intNoDeadline, noViolation), perm(go(R, T)), perm(intGo(R
55         ,T)), pow(intGo(R,T)) if
56         phase(obligation1, phaseA),
57         role(R, hero),
58         place(T, landOfAdventure);
59 % GENERATES: Obligation 1 -----
60 go(R, S) generates
61     intObligation1(R, S, T) if
62         role(R, hero),
63         place(S, home);
64 go(R, T) generates
65     intObligation1(R, S, T) if
66         role(R, hero),
67         place(T, landOfAdventure);
68 go(R, S) generates
69     intGo(R,S) if
70         role(R, hero),
71         place(S, landOfAdventure);
72
73 % INITIALLY: -----
74 initially
75     pow(intObligation1(R, S, T)) if role(R, hero), place(S, home), place(T,
76         landOfAdventure);
77 initially
78     perm(intObligation1(R, S, T)) if role(R, hero), place(S, home), place(T,
79         landOfAdventure);
80 initially
81     perm(go(R, S)) if role(R, hero), place(S, home);
82 initially
83     phase(obligation1, active),

```

```

82     role(hero, hero),
83     place(landOfAdventure, landOfAdventure),
84     place(home, home);

```

Listing C.2: An institution with an obligation event containing a deadline and a consequence

```

1  institution obligation2;
2  % TYPES -----
3  type Identity;
4  type Agent;
5  type Role;
6  type Trope;
7  type Phase;
8  type Place;
9  type PlaceName;
10 type Object;
11 type ObjectName;
12
13 % FLUENTS -----
14 fluent role(Agent, Role);
15 fluent phase(Trope, Phase);
16 fluent place(PlaceName, Place);
17 fluent object(ObjectName, Object);
18
19
20 % EXTERNAL EVENTS: Obligation 2 -----
21 exogenous event kill(Agent, Agent);
22 exogenous event go(Agent, PlaceName);
23 exogenous event noDeadline;
24
25 % VIOLATION EVENTS: Obligation 2 -----
26 violation event violHeroGoLandOfAdventure;
27 violation event noViolation;
28
29 % INST EVENTS: Obligation 2 -----
30 inst event intKill(Agent, Agent);
31 inst event intGo(Agent, PlaceName);
32 inst event intObligation2(Agent, Agent, Agent, PlaceName);
33 inst event intNoDeadline;
34
35 % OBLIGATION FLUENTS: Obligation 2 -----
36 obligation fluent obl(intGo(Agent, PlaceName), intKill(Agent, Agent),
37     violHeroGoLandOfAdventure);
38
39 % INITIATES: Obligation 2 -----
40 violHeroGoLandOfAdventure initiates
41     perm(kill(R, S)) if
42     role(R, villain),
43     role(S, hero);
44
45 % TERMINATES: Obligation 2 -----
46 intObligation2(R, S, T, U) terminates

```

```

45     phase(obligation2, active),
46     obl(intGo(R,U), intKill(S,T), violHeroGoLandOfAdventure), perm(go(R, U)),
        perm(intGo(R,U)), pow(intGo(R,U)) if
47     phase(obligation2, active),
48     role(R, hero),
49     place(U, landOfAdventure),
50     role(T, mentor),
51     role(S, villain),
52     role(R, hero),
53     role(S, villain);
54 intKill(S, T) terminates
55     obl(intGo(R,U), intKill(S,T), violHeroGoLandOfAdventure), perm(go(R, U)),
        perm(intGo(R,U)), pow(intGo(R,U)) if
56     role(R, hero),
57     place(U, landOfAdventure),
58     role(T, mentor),
59     role(S, villain),
60     role(R, hero),
61     role(S, villain);
62
63
64 % GENERATES: Obligation 2 -----
65 kill(S, T) generates
66     intObligation2(R, S, T, U) if
67     role(R, hero),
68     role(S, villain);
69 go(R, U) generates
70     intObligation2(R, S, T, U) if
71     role(R, hero),
72     place(U, landOfAdventure);
73 kill(R, S) generates
74     intObligation2(R, S, T, U) if
75     role(T, mentor),
76     role(S, villain);
77 go(R, U) generates
78     intGo(R,U) if
79     role(R, hero),
80     place(U, landOfAdventure);
81 kill(S, T) generates
82     intKill(S,T) if
83     role(R, hero),
84     role(S, villain);
85 kill(R, S) generates
86     intKill(R,S) if
87     role(T, mentor),
88     role(S, villain);
89
90 % INITIALLY: -----
91 initially
92     pow(intObligation2(R, S, T, U)) if role(R, hero), role(S, villain), role(
        T, mentor), place(U, landOfAdventure);

```

```

93 initially
94     perm(intObligation2(R, S, T, U)) if role(R, hero), role(S, villain), role
      (T, mentor), place(U, landOfAdventure);
95 initially
96     obl(intGo(R,U), intKill(S,T), violHeroGoLandOfAdventure), perm(go(R, U)),
      perm(intGo(R,U)), pow(intGo(R,U)) if role(R, hero), place(U,
      landOfAdventure);
97 initially
98     pow(intKill(S, T)),
99     phase(obligation2, active),
100    role(hero, hero),
101    role(villain, villain),
102    role(mentor, mentor),
103    place(landOfAdventure, landOfAdventure);

```

C.1 Branching Trope Example

```

1  institution branch3;
2  % TYPES -----
3  type Identity;
4  type Agent;
5  type Role;
6  type Trope;
7  type Phase;
8  type Place;
9  type PlaceName;
10 type Object;
11 type ObjectName;
12
13 % FLUENTS -----
14 fluent role(Agent, Role);
15 fluent phase(Trope, Phase);
16 fluent place(PlaceName, Place);
17 fluent object(ObjectName, Object);
18
19
20 % EXTERNAL EVENTS: Branch 3 -----
21 exogenous event kill(Agent, Agent);
22 exogenous event go(Agent, PlaceName);
23 exogenous event find(Agent, ObjectName);
24 exogenous event meet(Agent, Agent);
25 exogenous event noDeadline;
26
27 % VIOLATION EVENTS: Branch 3 -----
28 violation event noViolation;
29
30 % INST EVENTS: Branch 3 -----
31 inst event intKill(Agent, Agent);
32 inst event intFind(Agent, ObjectName);

```

```

33 inst event intGo(Agent, PlaceName);
34 inst event intMeet(Agent, Agent);
35 inst event intBranch3(Agent, Agent, Agent, ObjectName, PlaceName, PlaceName,
    PlaceName);
36 inst event intNoDeadline;
37
38 % INITIATES: Branch 3 -----
39 intBranch3(R, S, T, U, V, W, X) initiates
40     phase(branch3, phaseA),
41     perm(find(R, U)),
42     perm(go(R, X)),
43     perm(kill(R, S)) if
44         phase(branch3, active),
45         object(U, sword),
46         role(R, hero),
47         place(X, landOfAdventure),
48         role(S, villain);
49 intBranch3(R, S, T, U, V, W, X) initiates
50     phase(branch3, phaseB),
51     perm(meet(R, T)),
52     perm(go(R, V)) if
53         phase(branch3, phaseA),
54         place(V, realmOfMystery),
55         role(R, hero),
56         role(T, mentor);
57 % TERMINATES: Branch 3 -----
58 intBranch3(R, S, T, U, V, W, X) terminates
59     phase(branch3, active),
60     perm(go(R, W)) if
61         phase(branch3, active),
62         role(R, hero),
63         place(W, home);
64 intBranch3(R, S, T, U, V, W, X) terminates
65     phase(branch3, phaseA),
66     perm(find(R, U)),
67     perm(go(R, X)),
68     perm(kill(R, S)) if
69         phase(branch3, phaseA),
70         object(U, sword),
71         role(R, hero),
72         place(X, landOfAdventure),
73         role(S, villain);
74 intBranch3(R, S, T, U, V, W, X) terminates
75     phase(branch3, phaseB),
76     perm(meet(R, T)),
77     perm(go(R, V)) if
78         phase(branch3, phaseB),
79         place(V, realmOfMystery),
80         role(R, hero),
81         role(T, mentor);
82

```

```

83
84 % GENERATES: Branch 3 -----
85 find(R, U) generates
86     intBranch3(R, S, T, U, V, W, X) if
87         role(R, hero),
88         object(U, sword);
89 go(R, V) generates
90     intBranch3(R, S, T, U, V, W, X) if
91         role(R, hero),
92         place(V, realmOfMystery);
93 meet(R, T) generates
94     intBranch3(R, S, T, U, V, W, X) if
95         role(T, mentor),
96         role(R, hero);
97 go(R, W) generates
98     intBranch3(R, S, T, U, V, W, X) if
99         role(R, hero),
100        place(W, home);
101 go(R, X) generates
102     intBranch3(R, S, T, U, V, W, X) if
103         role(R, hero),
104         place(X, landOfAdventure);
105 kill(R, S) generates
106     intBranch3(R, S, T, U, V, W, X) if
107         role(S, villain),
108         role(R, hero);
109
110 % INITIALLY: -----
111 initially
112     pow(intBranch3(R, S, T, U, V, W, X)) if role(R, hero), role(S, villain),
113         role(T, mentor), object(U, sword), place(V, realmOfMystery), place(W,
114         home), place(X, landOfAdventure),
115     perm(intBranch3(R, S, T, U, V, W, X)) if role(R, hero), role(S, villain),
116         role(T, mentor), object(U, sword), place(V, realmOfMystery), place(W
117         , home), place(X, landOfAdventure),
118     perm(go(R, W)) if role(R, hero), place(W, home),
119     phase(branch3, active),
120     role(hero, hero),
121     role(villain, villain),
122     role(mentor, mentor),
123     place(home, home),
124     place(landOfAdventure, landOfAdventure),
125     place(realmOfMystery, realmOfMystery),
126     object(sword, sword);

```

C.2 “Evil Empire” Trope Example

Listing C.3: Full InstAL code for the “Evil Empire” trope, compiled from TROPICAL


```

1  institution evilEmpire;
2  % TYPES -----
3  type Identity;
4  type Agent;
5  type Role;
6  type Trope;
7  type Phase;
8  type Place;
9  type PlaceName;
10 type Object;
11 type ObjectName;
12
13 % FLUENTS -----
14 fluent role(Agent, Role);
15 fluent phase(Trope, Phase);
16 fluent place(PlaceName, Place);
17 fluent object(ObjectName, Object);
18
19
20 % EXTERNAL EVENTS: The Evil Empire -----
21 exogenous event chase(Agent, Agent);
22 exogenous event escape(Agent);
23 exogenous event capture(Agent, Agent);
24 exogenous event noDeadline;
25
26 % VIOLATION EVENTS: The Evil Empire -----
27 violation event noViolation;
28
29 % INST EVENTS: The Evil Empire -----
30 inst event intChase(Agent, Agent);
31 inst event intCapture(Agent, Agent);
32 inst event intEscape(Agent);
33 inst event intEvilEmpire(Agent, Agent);
34 inst event intNoDeadline;
35
36
37
38 % INITIATES: The Evil Empire -----
39 intEvilEmpire(R, S) initiates
40     phase(evilEmpire, phaseA),
41     perm(capture(R, S)) if
42         phase(evilEmpire, active),
43         role(S, empire),
44         role(R, hero);
45 intEvilEmpire(R, S) initiates
46     phase(evilEmpire, phaseB),
47     perm(escape(R)) if
48         phase(evilEmpire, phaseA),
49         role(R, hero);
50 % TERMINATES: The Evil Empire -----
51 intEvilEmpire(R, S) terminates

```

```

52     phase(evilEmpire, active),
53     perm(chase(R, S)) if
54         phase(evilEmpire, active),
55         role(S, empire),
56         role(R, hero);
57 intEvilEmpire(R, S) terminates
58     phase(evilEmpire, phaseA),
59     perm(capture(R, S)) if
60         phase(evilEmpire, phaseA),
61         role(S, empire),
62         role(R, hero);
63 intEvilEmpire(R, S) terminates
64     phase(evilEmpire, phaseB),
65     perm(escape(R)) if
66         phase(evilEmpire, phaseB),
67         role(R, hero);
68
69
70 % GENERATES: The Evil Empire -----
71 escape(R) generates
72     intEvilEmpire(R, S) if
73         role(R, hero);
74 chase(R, S) generates
75     intEvilEmpire(R, S) if
76         role(S, empire),
77         role(R, hero);
78 capture(R, S) generates
79     intEvilEmpire(R, S) if
80         role(S, empire),
81         role(R, hero);
82
83 % INITIALLY: -----
84 initially
85     pow(intEvilEmpire(R, S)) if role(R, hero), role(S, empire);
86 initially
87     perm(intEvilEmpire(R, S)) if role(R, hero), role(S, empire);
88 initially
89     perm(chase(R, S)) if role(S, empire), role(R, hero);
90 initially
91     phase(evilEmpire, active),
92     role(empire, empire),
93     role(hero, hero);

```

Appendix D

Parser Implementation

This section of the appendix describes the technical implementation of the TropICAL parser.

We use the Clojure programming language (Hickey, 2017a) with the Instaparse parser generator library (Engelberg, 2017) to create the language parser for TropICAL. The method of parser creation with Instaparse is to describe the parser using a syntax that resembles *Extended Backus-Naur Form* (EBNF) grammar, combined with regular expressions. According to the author, its features are:

- Turns standard EBNF or ABNF notation for context-free grammars into an executable parser that takes a string as an input and produces a parse tree for that string.
- No Grammar Left Behind: Works for any context-free grammar, including left-recursive, right-recursive, and ambiguous grammars.
- Extends the power of context-free grammars with PEG-like syntax for lookahead and negative lookahead.
- Supports both of Clojure's most popular tree formats (hiccup and enlive) as output targets.
- Detailed reporting of parse errors.
- Optionally produces lazy sequence of all parses (especially useful for diagnosing and debugging ambiguous grammars).
- "Total parsing" mode where leftover string is embedded in the parse tree.
- Optional combinator library for building grammars programmatically.
- Performant.

Due to using this method of parser creation, our parser has two stages: first it parses the entity declarations (roles, objects and places) at the top of the trope description, then takes the parsed declarations and inserts them as keywords in the grammar for the rest of the language, creating a new parser specifically for those entity declarations. For example, if the

author writes “The Hero is a role”, “The Castle is a place” and “The Sword is an object”, then the *role*

The Extended Backus-Naur Form (EBNF) of the grammar is shown in listings Listing D.1 and Listing D.2. Listing D.1 describes the syntax for the entity declarations at the top of each trope definition file, as described in Section 4.3.1 on page 63. Listing D.2 shows the grammar for the rest of the trope definition files.

Listing D.1: EBNF grammar for the entity declarations in TROPICAL

```

1 text = tropedef defs trope
2 <tropedef> = label <' is a ' ('trope' / 'policy') ' where:\n'>
3 <defs> = (<whitespace> (chardef | objdef | placedef) <'\\n'?>)+
4 chardef = charname <' is a ' ('character' | 'role') '.'?>
5 objdef = objname <' is an object' '.'?>
6 placedef = placename <' is a place' '.'?>
7 <charname> = name
8 <objname> = name
9 <placename> = name
10 trope = (<whitespace> !(chardef | objdef | placedef) #'.*' <'\\n'?>)+
11 label = <'\"'> words <'\"'>
12 <whitespace> = #'\\s\\s'
13 <name> = [<'The ' / 'the '>] cwords
14 <words> = word (<' '> word)*
15 <cwords> = cword (<' '> [<'of'<' '>] cword)*
16 <cword> = #'[A-Z][0-9a-zA-Z\\-\\_\\]'*
17 <the> = <'The ' | 'the '>
18 <word> = #'[0-9a-zA-Z\\-\\_\\]'*

```

Listing D.2: EBNF grammar for the main trope definitions in TROPICAL

```

1 trope = (<whitespace> sequence)+ <'\\n'?>
2 (str "label = ''" label "'")
3 fluent = object <' ' is ' > adjective
4 adjective = word
5 outcome =
6 (<'\"\\n\"' whitespace whitespace> (event | obligation | happens) (or? | if
7 ?) <'\"\\n\"'>)+
8 happens =
9 <the?> subtrope <('\" happens' / ' trope happens' / ' policy applies
10 ') '.'?>
11 block =
12 <the?> subtrope <' policy does not apply' / ' does not happen' / ' trope does
13 not happen' > <'.'?>
14 sequence =
15 ((fluent | event | norms | happens | block) (or* | if*)) | ((fluent | event |
16 norms | happens | block) (<whitespace+ 'Then '> (block / norms / event /
17 fluent / obligation / happens) (or* | if*)))*
18 or =
19 <whitespace+ 'Or '> (fluent | event | norms)
20 if =
21 <whitespace+ 'If '> (fluent | event | norms)

```

```

17 action = !fverb verb [sp [particle sp] (character | object | place)] (crlf? |
    [sp particle? (character | object | place)] crlf?)
18 event = character sp action
19 <particle> = <'a' | 'at' | 'will' | 'of' | 'to'>
20 norms = permission | rempermission | obligation
21 fluent = (character | object | place) sp fverb sp [particle sp] (character |
    object | place) crlf?
22 fverb = (<'is'> sp 'at') | 'has'
23 violation = norms
24 (str "character = " (if (seq rs) rs "'nil'"))
25 (str "place = " (if (seq ps) ps "'nil'"))
26 (str "object = " (if (seq os) os "'nil'"))
27 subtrope = <'\'> words <'\'>"
28 label = <'\'> words <'\'>"
29 verb = words
30 rempermission = character <' may not '> action crlf?
31 permission = character <' may '> action crlf?
32 obligation = character <' must '> action (<' before '> deadline)? (<crlf
    whitespace+ 'Otherwise, '> <'the '?> violation)? <'.'?> crlf?
33 deadline = consequence
34 role-b = character
35 object-b = object
36 place-b = place
37 <crlf> = <'\\n'>
38 consequence = event
39 <whitespace> = #'\\s\\s'
40 <sp> = <' '>
41 <words> = word (sp word)*
42 <cwords> = cword (sp ['of' sp] cword)*
43 <cword> = #'[A-Z][0-9a-zA-Z\\-\\_\\']*'
44 <the> = <'The ' | 'the '>
45 <word> = #'[0-9a-zA-Z\\-\\_\\']*'

```

Appendix E

Full Trace for “Evil Empire” and “Hero’s Journey” Tropes

Listing E.1: Example trace for both the “Evil Empire” and “Hero’s Journey” tropes combined together

```
1 Answer Set 70:
2
3 Time Step 1:
4
5 holdsat(phase(herosJourney,phaseA),herosJourney)
6 holdsat(perm(kill(hero,villain)),herosJourney)
7 holdsat(perm(escape(villain)),herosJourney)
8 holdsat(phase(evilEmpire,active),evilEmpire)
9 holdsat(perm(chase(hero,empire)),evilEmpire)
10 occurred(intHerosJourney(hero,villain,evilLair,home),herosJourney)
11 occurred(go(hero,evilLair),herosJourney)
12
13
14 Time Step 2:
15
16 holdsat(phase(herosJourney,phaseA),herosJourney)
17 holdsat(perm(kill(hero,villain)),herosJourney)
18 holdsat(perm(escape(villain)),herosJourney)
19 holdsat(phase(evilEmpire,phaseA),evilEmpire)
20 holdsat(perm(capture(hero,empire)),evilEmpire)
21 occurred(intEvilEmpire(hero,empire),evilEmpire)
22 occurred(chase(hero,empire),evilEmpire)
23
24
25 Time Step 3:
26
27 holdsat(phase(herosJourney,phaseB),herosJourney)
28 holdsat(perm(go(hero,home)),herosJourney)
29 holdsat(phase(evilEmpire,phaseA),evilEmpire)
30 holdsat(perm(capture(hero,empire)),evilEmpire)
```

```
31 occurred(kill(hero, villain), herosJourney)
32 occurred(intHerosJourney(hero, villain, evilLair, home), herosJourney)
33
34
35 Time Step 4:
36
37 holdsat(phase(evilEmpire, phaseA), evilEmpire)
38 holdsat(perm(capture(hero, empire)), evilEmpire)
39 occurred(intHerosJourney(hero, villain, evilLair, home), herosJourney)
40 occurred(go(hero, home), herosJourney)
41
42
43 Time Step 5:
44
45 holdsat(phase(evilEmpire, phaseB), evilEmpire)
46 holdsat(perm(escape(hero)), evilEmpire)
47 occurred(intEvilEmpire(hero, empire), evilEmpire)
48 occurred(capture(hero, empire), evilEmpire)
49
50
51 Time Step 6:
52
53 occurred(intEvilEmpire(hero, empire), evilEmpire)
54 occurred(escape(hero), evilEmpire)
```

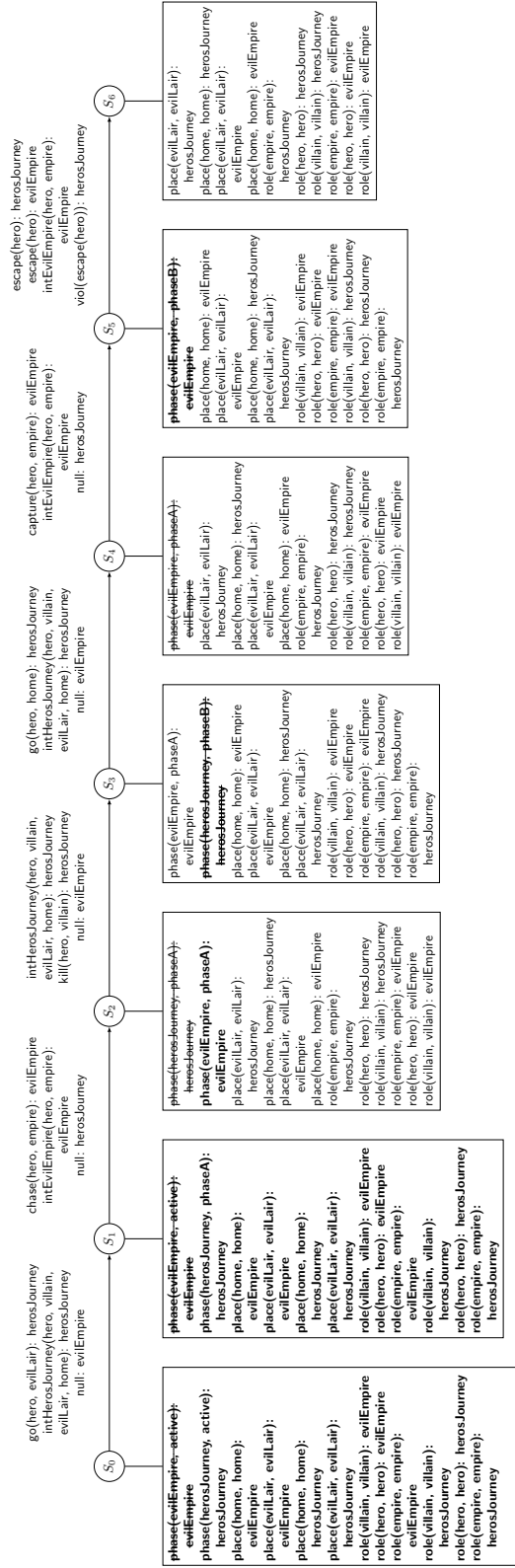


Figure E-1: Visualisation of trace for both the “Evil Empire” and “Hero’s Journey” in Appendix E on page 212

Appendix F

AgentSpeak Code for Character Agents and Roles

This appendix lists example AgentSpeak code for the various character roles in the Punch and Judy story, along with the code for the characters themselves.

F.1 Character Roles

F.1.1 The Idiot

Listing F.1: AgentSpeak code for the “Idiot” role

```
1 // "Mishandling of an Item" trope
2
3 +startTrope(mishandling)
4   <- !breakItem;
5     !loseItem;
6
7 +!breakItem : not (desperate | perm(break(X))) & type(X, item)
8   <- !breakItem;
9
10 +!breakItem : (desperate | perm(break(X))) & type(X, item)
11   <- break(X);
12
13 +!loseItem : not (desperate | perm(lose(X))) & type(X, item)
14   <- !loseItem;
15
16 +!loseItem : (desperate | perm(lose(X))) & type(X, item)
17   <- lose(X);
```

F.1.2 The Narrator

Listing F.2: AgentSpeak code for the “Narrator” role

```
1 +startTrope(narratorTrope)
```

```

2   <- !talkToAudience;
3
4  +!talkToAudience : not perm(talk(X, Y)) & role (X, narrator) & role(Y,
      audience)
5   <- !talkToAudience;
6
7  +!talkToAudience : (desperate | perm(talk(X, Y))) & role(X, narrator) & role(
      Y, audience)
8   <- talkTo(X, Y);

```

F.1.3 The Hero

Listing F.3: AgentSpeak code for the “Hero” role

```

1  +startTrope(miniBossTrope)
2   <- !fightMiniBoss;
3     !defeatOrDie;
4     !escape;
5
6  +!fightMiniBoss : not perm(fight(X, Y)) & role(X, hero) & role(Y, miniBoss)
7     <- !fightMiniBoss;
8
9  +!breakItem : (desperate | perm(break(X))) & type(X, item)
10     <- !break(X);
11
12 +!loseItem : not perm(lose(X)) & type(X, item)
13     <- !loseItem;
14
15 +!loseItem : perm(lose(X)) & type(X, item)
16     <- !lose(X);

```

F.2 Character Agents

F.2.1 Code for Punch Agent

Listing F.4: AgentSpeak code for the Punch agent

```

1  // A lot of Punch's behaviour is inherited from the roles that he enacts in
   various tropes:
2
3  { include("movement.asl") }
4  { include("emotions.asl") }
5
6  { include("idiot.asl") } // "Punch, Judy, and the Baby", "Crocodile and
   Sausages"
7  { include("husband.asl") } // "Punch, Judy and the Baby"
8  { include("pursuer.asl") } // "Punch, Judy and the Baby", "Punch and the
   Policeman", "Punch and the Devil"
9  { include("hero.asl") } // "Punch and the Policeman", "Punch and the Devil"

```

```

10 { include("slapper.asl") } // "Punch and the Policeman", "Punch and the Devil
    "
11 { include("pursuee.asl") } // "Crocodile and Sausages"
12
13 { include("audience-aware-character.asl") } // All scenes
14
15 // The user is also able to author any other plans they want to add in order
    to customise Punch's behaviour

```

F.2.2 Code for Judy Agent

Listing F.5: AgentSpeak code for the Judy agent

```

1 { include("movement.asl") }
2 { include("emotions.asl") }
3
4 { include("wife.asl") } // "Punch, Judy and the Baby" scene
5 { include("owner.asl") } // "Punch, Judy and the Baby" scene
6 { include("pursuee.asl") } // "Punch, Judy and the Baby" scene
7
8 { include("audience-aware-character.asl") } // All scenes

```

F.2.3 Code for Joey Agent

Listing F.6: AgentSpeak code for the Joey agent

```

1 { include("movement.asl") }
2 { include("emotions.asl") }
3
4 { include("narrator.asl") } // "Introduction" scene

```

F.2.4 Code for Crocodile Agent

Listing F.7: AgentSpeak code for the Crocodile agent

```

1 { include("movement.asl") }
2 { include("emotions.asl") }
3
4 { include("accomplice.asl") } // "Crocodile" scene
5 { include("pursuer.asl") } // "Crocodile" scene
6 label=lst:pjfull-crocagent,caption={AgentSpeak code for the Crocodile agent}}

```

F.2.5 Code for Policeman Agent

Listing F.8: AgentSpeak code for the Policeman agent

```

1 { include("movement.asl") }
2 { include("emotions.asl") }

```

```

3
4 { include("mini-boss.asl") } // "Punch and the Policeman" scene
5 { include("pursuee.asl") } // "Punch and the Policeman" scene
6 { include("slapped.asl") } // "Punch and the Policeman" scene
7 label=lst:pjfull-policeagent,caption={AgentSpeak code for the Policeman agent
  }}

```

F.2.6 Code for Devil Agent

Listing F.9: AgentSpeak code for the Devil agent

```

1 { include("movement.asl") }
2 { include("emotions.asl") }
3
4 { include("mini-boss.asl") } // "Punch and the Devil" scene
5 { include("pursuee.asl") } // "Punch and the Devil" scene
6 { include("slapped.asl") } // "Punch and the Devil" scene
7 label=lst:pjfull-devilagent,caption={AgentSpeak code for the Devil agent}}

```

F.2.7 Code for “Director” Agent

Listing F.10: AgentSpeak code for the “Director” agent

```

1 trope(herosJourney);
2 role(hero, herosJourney);
3 place(landOfAdventure, herosJourney);
4 object(magicalAgent, herosJourney);
5
6 getSimilarTropes(X, T)
7 :- 1{trope(T) &
8     trope(X) &
9     role(R, X) &
10    role(R, T) &
11    object(O, X) &
12    object(O, T) &
13    place(P, X) &
14    place(P, T)}1;

```

Appendix G

Thematic Analysis Transcripts

G.1 Participant A

Dialogue starts at at 16:00 in the recording.

1 **Participant:** So if "Example 1" defined a hero here, and in "Kill then Search"... oh I
2 guess... have they defined Hero there? Hero's a role here, and then in "Kill then Search" the
3 Hero's also a role... will that cause any confusion, because the Hero's being defined twice?

4 **Myself:** That's a really good question, actually. In an older version of this I had it so
5 that you could have different heros in the different tropes. At the moment they're using the
6 same Hero, just called "Hero". But you can actually choose different names for the different
7 Heros, and you'd have two heroes in the story. But just to simplify this whole thing, I decided
8 "if there's two heroes, just have them as the same person".

9 **Participant:** Fair enough.

10 **Myself:** OK yeah, so you've done the first task, which is great. So task 2 is to create
11 your own trope. So if you click on "New", the first thing you have to do is click on the trope
12 name in that box. And try to avoid uh... some special characters. Yeah, that's fine, "New
13 trope" is fine. Uh, so given the following character roles (Hero, Villain and Mentor). Um, so
14 you have to declare those. Yeah, exactly, very good. And then the three places: Home, the
15 Evil Lair and Land of Adventure.

16 **Participant:** How does it deal with spaces?

17 **Myself:** It's fine with spaces, yeah. Most of the time. I'll let you know if it isn't. The
18 problem is that, as you probably know with PhDs, is you have to do stuff which looks really
19 simple but under the surface is really complicated. It's really hard for me to actually get some
20 informative error messages up on here, so I'll try my best to work out what's going wrong
21 when it does go wrong. But for now, it's going alright.

22 **Participant:** OK.

23 **Myself:** So, I'd like you to construct your own tropes, just using the sequences and
24 branches for now. Actually, no, part of this one is just having sequences, so just one thing
25 after another.

26 **Participant:** OK, so I could do like: The Villain's in the Evil Lair.

Feature:
Different
characters
in different
tropes

Syntax:
Spaces in
words

Feature:
State man-
agement

27 **Myself:** Yeah.

28 **Participant:** I'm just trying to remember the exact syntax. It's "goes"?

29 **Myself:** That's fine. It should be fine with any verb.

30 **Participant:** Oh, OK.

31 **Myself:** And any form of any verb.

32 **Participant:** (types "resides")

33 **Myself:** Yeah, that should be OK. I hope! So, how are you finding this? Is it easy to
34 learn so far?

35 **Participant:** Yes, especially with the examples, so I can basically just copy everything
36 you've done so far. When you told me that any verb could fit in there, that wasn't immediately
37 apparent to me, so maybe if it said that explicitly?

38 **Myself:** Sure.

39 **Participant:** I was looking for like "oh, is 'goes' another keyword?", so to speak.

40 **Myself:** That's a good idea. I tried this before on some people, and it *wasn't* OK with
41 every verb, so I wasn't saying it before. But now it is! I changed it a lot. So, that's a good
42 point, I should make that explicit. Have you used any programming languages similar to this
43 before, with like a controlled natural language?

44 **Participant:** Uh, kind of, when I was working at Microsoft, they made their own in-house
45 one to do testing for "Fable Legends".

46 **Myself:** Oh wow, cool.

47 **Participant:** So it was kind of complicated. I wasn't directly involved in it, I only dabbled
48 in it very briefly, but this reminds me of that a little bit.

49 **Myself:** OK, wow. Very nice.

50 **Participant:** That was for like, designers trying to test out scenarios without any coding
51 experience. They could say like "The Hero is here" and "This sort of Villain appears" with
52 text, and it'd generate code for them, that was the idea.

53 **Myself:** Yeah, that sounds very similar to this use case. I've designed this for people who
54 aren't programmers, like story authors, so that they can create these interactive stories. But
55 yeah, keep going, sounds really good.

56 **Participant:** Uh, doing OK. "The Villain resides in the Evil Lair", uh... Besides... so,
57 we've got people in places. I'm trying to think what else we've got... They can find objects,
58 but I haven't got any objects they can find. Can I implement objects, or am I just supposed
59 to be using this stuff?

60 **Myself:** Uh, for this task... That's the next task. So this one's just roles and places.

61 **Participant:** Just roles and places... I'll just put everyone in places then, I guess. So,
62 "Hero arrives in Home". I do have to add the "in", right? That is the keyword?

63 **Myself:** No, you can delete "in". "in" is not significant at all.

64 **Participant:** Why is it purple, then?

65 **Myself:** Uh, because it's using a Python syntax highlighter, I think. I was trying to get
66 some kind of syntax highlighting, and I was too lazy to write my own. So that's why. That is
67 a bit confusing though, yeah. So "Home" has to be capital "H".

68 **Participant:** Ah. Oh yeah, that is how I defined it, yeah. "Then the Hero goes to the
69 Evil Lair".

70 **Myself:** OK, save that. It's not going to update that visualisation... This interface is a
71 little bit clunky, so you have to go to "Edit", and then select it. Where is it? You did save it,
72 right?

73 **Participant:** And it came up with the tick as well.

74 **Myself:** Uh oh... Let me refresh my one.

75 **Participant:** And it's saved the title "A new trope"

76 **Myself:** Ah OK, yeah, it seems to have saved the title as a blank string, and we can't
77 change that. I'll just do some fiddling around here. I wonder why it didn't like that... Ah,
78 it's saving all the titles as blank strings. Why is it doing that? It wasn't doing that before. It
79 might not like the fact that you're... no, it's not that. Hmm... I wonder if it's the case that
80 there's something here that it doesn't like... OK, I'm just editing it so that I know there's
81 not anything that will trip it up.

82 **Participant:** OK

83 **Myself:** I'm wondering if when it doesn't compile, then it saves it with a blank name.
84 But that wouldn't make sense. Hmm. Oh! OK, let's see. Ah yeah, so if it doesn't compile
85 (this is a bug I need to fix), it will save it with a blank name. So, the thing that's tripping it
86 up, by the look of it, is that you're missing an "a", which is "The Hero is a role". At the top
87 there.

88 **Participant:** OK, I've re-input it in, because I've lost it. Are you looking at the old...?

89 **Myself:** I'll re-input it in for you. So...

90 **Participant:** I've just got this new one here, if you look at the screen capture. Is that
91 one fine?

92 **Myself:** Oh, hang on... Don't save it though, because I've just saved one called "New
93 Trope", which is hopefully the same. I'm just going through the blank ones and deleting them
94 now. I wasn't aware of that bug, but I am now, which is good.

95 **Participant:** Great.

96 **Myself:** So yeah, if you go to "edit", and "select new trope"...

97 **Participant:** It's not there. Shall I refresh the page maybe, if it's on yours?

98 **Myself:** You might have to refresh the page, yeah. And OK, we get a compile error if you
99 try to refresh. So there's still something it doesn't like there. So as I said before, the trouble
100 is with this is that if it doesn't work, then it's really hard to get any useful error messages up
101 here.

102 **Participant:** I've got "the Evil Lair" was supposed to be capitalised with "T", that's
103 what I've missed off.

104 **Myself:** Ah, OK. I think it ignores the "the's" though, so it might not be that.

105 **Participant:** But "the" was part of its name, because I put "the Evil Lair" is a place,
106 right?

107 **Myself:** I think even then it ignores the "the's". Oh! I think if you... It doesn't seem to
108 like... Yeah it's the "in" in "resides in" it doesn't like, I think. So if you delete the "resides

Syntax: Use
of "the"

109 in". Because it seems to be fine with the verbs, but sometimes if you have extra particles after
110 those verbs, it doesn't like them. But yeah, it's OK if you remove the "in".

111 **Participant:** Still got an error.

112 **Myself:** Ah. Did you click "save", then "refresh"?

113 **Participant:** (clicks)

114 **Myself:** Ah, there you go. It takes a while sometimes. It's doing really complicated stuff
115 on a really slow server. OK, cool. So that's the first task done. So any feedback on that? I
116 know there are plenty of bugs, and error messages are a concern, but are there any thoughts
117 that you have?

118 **Participant:** I mean, yeah, when you look past the bugs, then the system is quite intuitive.

Usability:
Ease of use

119 **Myself:** OK, cool. Let's see... so if you go back to the tasks. Add a "Weapon" and
120 an "Evil Plan" to your trope. And again, don't have any branches, just do one thing after
121 another in sequences.

122 **Participant:** OK, do I have to define all these before I start? Can they be anywhere?

Syntax: En-
tity defini-
tions

123 **Myself:** Yeah, they do have to be...

124 **Participant:** So I define the roles, then I define the places do I then define the weapons
125 and whatnot?

126 **Myself:** That would make sense, but as long as all your definitions are at the start, then
127 that's fine.

128 **Participant:** It could be anywhere, as long as I'm using stuff that's already been defined?

129 **Myself:** Yeah, yeah, exactly.

130 **Participant:** I was just wondering if this is a kind of constructor of sorts, but it's not?

131 **Myself:** Uh, it's not object-oriented or anything like that. It's really basic.

132 **Participant:** OK, that makes it easier really. So, is "Weapon" an alternative to "place"
133 or "role"?

134 **Myself:** You'd have to put them as objects. So "Weapon" is an object, "The Evil Plan"
135 is an object. Cool. Yeah, good. I think if you have that blank line, it doesn't like it. But add
136 them into the trope somewhere.

137 **Participant:** So, could I say like: "The Mentor gives the Hero the Weapon"?

Feature:
Event with
three entities

138 **Myself:** Uh, yes, you can. Maybe. I can't remember if I did that or not. Maybe. See
139 what happens if you do that. Save it and refresh first to see if that works. I have a feeling
140 that "has"... I did some kind of half-baked thing with that that might not work any more.
141 So try that. No, yeah, OK, change it to something else.

142 **Participant:** Just "has" by itself, or is it the word "has" that's wrong?

Feature:
State man-
agement

143 **Myself:** It's the word "has" that I reserved as a keyword and then did nothing with. OK
144 yeah, try that. Sometimes it takes a little while. There we go, yeah.

145 **Participant:** OK, great. So... "The Mentor gives"...

146 **Myself:** Yeah, see if that works. That should work.

147 **Participant:** Look, it's compiled.

148 **Myself:** It's compiled, but it hasn't appeared. Ah, OK, so because this goes through an
149 answer set solver, I've limited it to just five steps. So if you remove one of those, like "The

150 Mentor lives at Home" or something. Because if I have more than five steps in the answer set
151 solver, it can take really, really long because it's searching through all the possibilities. Once
152 you have lots of branches in there it just takes ages. There you go, it says "Give Hero Mentor
153 Weapon".

Limitation:
Five event
limit

154 **Participant:** OK.

155 **Myself:** Cool. Alright. Any more thoughts on that task?

156 **Participant:** Um, I guess just the info that you can combine the two roles in one sentence
157 would be good. I asked you about that. It makes sense that it would, just maybe some people
158 would simply have to put the separate steps.

159 **Myself:** OK, yeah, yeah. That's something I should brief people on before. OK, cool.
160 So, task 4 is now: feel free to add some branches to your trope with the "Or" keyword. And
161 remember you have to indent it two spaces.

162 **Participant:** OK. So the "Mentor gives the Hero the Weapon". Kills the Hero... a real
163 twist in the tale up front!

164 **Myself:** Yeah! So save and try that. Ah so, it's sometimes very slow. And that hasn't
165 worked. Why doesn't that work?

166 **Participant:** There we go, it just came up.

167 **Myself:** It has! It's just incredibly slow. OK.

168 **Participant:** I'm not sure how the Hero then goes to the Evil Lair. But that's just a flaw
169 in my own story!

Feature:
Event seman-
tics

170 **Myself:** Sorry? Oh yeah, he kills the Hero, and then he goes to the Evil Lair even though
171 he's dead! How does that happen?

172 **Participant:** He respawns at the Evil Lair as a daemon of the Villain?

173 **Myself:** I like your thinking, yeah. So if you want to add more branches, you can chain
174 branches together. So at the moment there's two possibilities there, but if you add another
175 "Or" on the same level of indentation below that, it will create another possibility.

176 **Participant:** OK, so, I can do more than one. "The Mentor attempts to kill the Hero"...

177 **Myself:** I wonder if that's going to... Try that out and see if that compiles. I'm not sure
178 if it will. Get rid of that blank line.

179 **Participant:** OK, yeah.

180 **Myself:** Because I'm not sure if that will compile. No. Too complicated. Oh! What's
181 that? "The Mentor kills"... Oh no, it's... Hmm. Did you save that one?

182 **Participant:** It seems to have gone back to the previous.

183 **Myself:** OK, that's weird.

184 **Participant:** So I'll just say "attacks the Hero".

185 **Myself:** Yeah. There you go. Hmm. OK, so this is a limitation. Oh, what's this? That's
186 interesting. Ah yeah, so one limitation is that unfortunately I haven't implemented extending
187 a branch beyond one event. Because again, it makes things very very complicated. I just
188 wanted to demonstrate the simple nature of this language. So like, what you're trying to do...
189 I can see what you're trying to do there is you're trying to say "The Mentor attacks the Hero,
190 THEN the Hero escapes" on that branch, and then you'd branch it further. Unfortunately,

191 you're not able to continue a branch like that. We're just testing out very simple branching.

192 **Participant:** OK.

193 **Myself:** Cool. It's interesting that that still compiled, though, but it just put it onto the
194 end of every branch.

195 **Participant:** Of both possible... yeah.

196 **Myself:** OK, cool. OK, let's see. Task five: so the next task, go to the arrange tab on the
197 left. And with this you can select... so select one of the simpler tropes, like Example 1, then
198 click "+", then click another one, and again a simple one again like Example 2, and then wait
199 a bit, and then it will visualise both of them at the same time. Eventually. Again, this is very
200 slow. There you go. So, when you have both of those tropes happening in a story... so we've
201 got a story saying "OK, I want these two tropes", then it generates all the possible branches
202 with those two tropes happening. So it looks like Example 2 starts with three branches, and
203 with Example 1 there's only one possibility, so it's kind of like... this is a tree with all the
204 different combinations of events that you can do. So, yeah, just doing that WAS task five. Do
205 you think this is useful for, you know, if someone is trying to write a story with tropes like
206 this, is it useful to have this kind of visualisation?

207 **Participant:** Yeah, certainly, if some of the stuff we've mentioned had been ironed out
208 then I could definitely see this being a useful tool.

209 **Myself:** Cool, cool. Alright. Any other comments that come to mind?

210 **Participant:** Uh, yeah, well just back on the.. when you're refreshing and compiling, it
211 would be useful to have a little loading icon or whatever so that I know that it HAS loaded
212 or hasn't.

213 **Myself:** Yeah, I definitely agree. It's something I've tried to do, but unfortunately didn't
214 have time for. But, yeah, that's good to know. So next task, task 6: I want you to embed a
215 trope within another trope. So maybe like take the trope you've already made, and embed an
216 existing trope in it, or something like that.

217 **Participant:** I'm just going to take this bit I added in the previous one.

218 **Myself:** OK, cool. And remember it can only be five steps long at the moment.

219 **Participant:** OK, yeah. I'm trying to remember how to embed another trope...

220 **Myself:** Do you have the document open? So it's there, and the "Name trope happens".
221 Yeah. That's alright.

222 **Participant:** Example 3, let's go for it.

223 **Myself:** Example 3... was that a complicated one? I wonder... OK, if it was a com-
224 plicated one, it might take a while. Yeah, that was quite a complicated one. But yeah, we'll
225 be waiting a couple of minutes perhaps. No, not that long! Oh, there you go. New Trope,
226 New Trope... It's only got ones from your trope, why is that? I think it's longer than five
227 things long. So delete maybe "The Hero arrives Home", or something like that. Again, it
228 might be doing the previous one. Just wait a little while. Example 3... what was example
229 3? Example 3 is a bit complicated. Oh, OK, it's got... Ah again, yeah yeah, OK. So maybe
230 I need to extend it so that it does more than five events, because it's just done the first event
231 of Example 3, and that's all that it can handle, sadly. Um. Try another one, try embedding

232 a different one. Yeah, maybe I should consider having it do more than five events. But as
233 you can see, it's slow enough as it is. OK, Example 1. I think... hang on, that's already
234 five events. So it's just chaining Example 1 on the end there. See what happens if you put
235 Example 1 earlier in the trope.

236 **Participant:** OK.

237 **Myself:** Ah yeah, that's actually something I haven't tested. I always put them at the
238 end when I was testing it. So, I should have tested putting it earlier. But that's interesting
239 to know that that doesn't work. I'll have to go in and... Oh, or did it? No, it didn't, that
240 one's at the end. OK, so, any thoughts that have occurred to you doing that? Again, we've
241 encountered some more bugs, but as I said that's kind of the point of doing this exercise.

242 **Participant:** Uh, not a load relating to that task specifically. With this springy nature
243 thing, how do you compress it entirely?

244 **Myself:** There's no way of collapsing it. I'm just using a plain old JavaScript library. I
245 agree that it would be nice to collapse the nodes, wouldn't it, so that you could...

246 **Participant:** 'Cause I like it when it's simple, like bring it all out, then I add a few
247 branches and it's way too big, and then I wanna like... bring it in.

248 **Myself:** Yeah, that's a good idea. I'll see if I can change that. But yeah, for now, you're
249 stuck with it as it is, sadly.

250 **Participant:** I appreciate that it's not a high priority.

251 **Myself:** OK, so final task, task seven is: start with a new trope, just a blank one, and
252 create a... just think of a story and see if you can create it using tropes. And again, use
253 multiple tropes. Rather than have a trope called "Brand New Trope", think about what the
254 trope is doing, so is it describing the Hero going on a quest or something?

255 **Participant:** OK, let's have a "Revenge Trope".

256 **Myself:** Cool, cool. Yeah, yeah.

257 **Participant:** OK. So "The Villain is a role, the Hero is a role, the Villain kills...". So,
258 I could... say, like, "The Hero is Angry", I'm not relating that to any place or object, that's
259 fine as a statement?

260 **Myself:** Again, I used to have this thing... at the moment you do have to have verbs, I
261 know "is" is kind of a verb... but "is" and "has" are reserved as key words because I wanted
262 to do something else, and I forgot to un-reserve them, kinda thing. So, yeah, don't use "is", use
263 something else. So "The Hero gets angry", or "The Hero becomes angry". So save that, and
264 see if that compiles by going into the "edit" tab. Oh, OK. I reckon it doesn't like "Becomes
265 angry". "Angers" perhaps, "Angers" might work.

266 **Participant:** OK

267 **Myself:** Cool. So you've got that trope, now make another trope. So you've got some
268 revenge in your story there, what else do you want to happen in your story?

269 **Participant:** Uh... we could have a betrayal.

270 **Myself:** OK, cool.

271 **Participant:** "The Hero trusts the Mentor, the Mentor betrays the Hero".

272 **Myself:** Cool.

Limitation:
Subtrope at
end

Feature:
State man-
agement

273 **Participant:** I've got it really in for Mentors apparently, I didn't realise. I had the mentor
274 killing the Hero previously, now he's betraying him. I don't have much trust of mentors,
275 apparently. That's news to me.

276 **Myself:** Ah, yeah. Well, it's interesting all the subconscious stuff that's coming up through
277 this process of creation. I like it. OK, so you've got that, now if you go to the arrange tab,
278 you can combine those tropes together. Yeah, there you go. Again, when you have multiple
279 tropes, it seems to take a long time. Why is that? So, it's done the revenge trope, but now
280 you've added the betrayal... ooh, there's lots of possibilities.

281 **Participant:** Yes, whether we start with the revenge first, or the betrayal first.

282 **Myself:** Yeah, basically.

283 **Participant:** Either way, it goes badly for the hero.

284 **Myself:** Yeah, true.

285 **Participant:** "The Villain kills the Friend, then the Hero is trusting in the Mentor, then
286 he is betrayed by the Mentor, and that makes him even angrier, so he goes and kills the
287 Villain". That makes total sense! I mean, he was angry at the villain, but with the betrayal
288 on top of that it just led to the revenge. Boom. Narrative conceived, right there.

289 **Myself:** Excellent.

G.2 Participant B

Dialogue starts at 14:40.

1 **Myself:** Now I'll get you to do some tasks with the language. So, the first one is to take
2 an existing trope, any of those tropes there, and just edit it in some way. Change it in some
3 way that you see fit.

4 **Participant:** Sure. So just change something? "The Bucket is an object".

5 **Myself:** Ah... has it...? Oh no, it's OK. I was worried there for a second. So yeah,
6 that's changed.

7 **Participant:** Sorry about that, compile error.

8 **Myself:** One thing that is not ideal is that I don't have very detailed error messages yet.
9 It's kind of frustrating to explain, but there's lots of complicated stuff going on underneath
10 this that's kind of the research-y bit.

11 **Participant:** Oh, error detection is massive, Matt.

12 **Myself:** Oh got, yeah, especially in languages that only PhD students in our department
13 use, there's no error stuff there. So, yeah, it's kind of tricky. But I should be able to spot
14 anything that's going wrong and let you know how to fix it.

15 **Participant:** I'll slow down, though. In fact, you can just say "Next, next, next" when
16 you actually want me to click a button.

17 **Myself:** OK, sure. But that looks good - so you've changed something. I'd say that
18 counts as finishing the first task. So, obviously you have kind of a programming background,
19 you're used to learning languages, but would you say that this would be easy to pick up for
20 someone who just writes stories and has no programming background?

Limitation:
Error mes-
sages

21 **Participant:** Oh, hugely. I don't have a great eye for detail, and it's the detail that
22 kills programming. So I'm not having to keep a mental note of when you're using a comma,
23 semicolon, colon, you know as in I-can't-remember-what-language really depends upon that,
24 even HTML, CSS. The nice thing about this is that it's plain, using that conventional punc-
25 tuation, and symbols. Because it is a language - programming is a language, it's fluency. But
26 of course a newbie, those punctuation marks in my mind I kind of skip them, because they're
27 punctuation, but they're not, they're integral to the working. So the fact that you've kept it
28 without punctuation is fantastic.

Usability:
Ease of use

29 **Myself:** Great, great.

30 **Participant:** The idea that we just carry a new line to declare a new variable. Brilliant.
31 I think maybe a writer might be inclined to pop a full stop behind here, a period at the end
32 of each sentence.

33 **Myself:** Actually, you can do that, and it ignores it. So I've written that into it.

34 **Participant:** Oh, well done. How about, I know Americans, when they do listing they
35 like to use a semicolon.

36 **Myself:** Ah, that probably won't work in this case, that's actually probably not a good
37 thing to do.

38 **Participant:** The Americans I've met are very keen on semi-colons when they're running
39 through a list.

40 **Myself:** OK, OK.

41 **Participant:** So, apart from that, the only thing really, which would flick up in an author's
42 mind is the use of a double-space. But, you know, live with it.

43 **Myself:** Yeah, yeah.

44 **Participant:** Maybe the use of a tab instead of a double space would be more in keeping
45 with the convention of using tabs in word processors.

46 **Myself:** Yeah, that's a good point actually.

47 **Participant:** I noticed that you've got some capitals in there, are the capitals important
48 to the running of the program?

Syntax:
Capitalisa-
tion

49 **Myself:** Yes they are, very much. So if you have the name of some kind of entity like the
50 Hero, the Home, the Bucket, it's important to give that a name with a capital letter. That's
51 how it kind of identifies that.

52 **Participant:** If I could just ask, you're using capitals for nouns, then, be that abstract
53 nouns or physical nouns, that's fine. Quick question - Land of Adventure. We haven't used it
54 yet, can I just try something?

55 **Myself:** Yeah, go ahead. Ah, OK, I think I see what you're doing. I think you might
56 encounter a bug I have. Oh no, you don't. I have this bug which comes up sometimes, but
57 not always, where if you don't put a "the", if you declare it with a "the", it's supposed to
58 ignore the "the", but sometimes it doesn't and you have to use a "the" later. But in this
59 case it's fine, so I'm not quite sure what's going on. But no, that's good. So, yeah, "Land of
60 Adventure" works.

Syntax: Use
of "the"

61 **Participant:** It capitalised "Land"... an author would get bugged by that. Don't worry,

62 I'm just making an observation. "Land of Adventure", I like it in this flowchart where it's
63 capitalised to keep this word separate, that's really cool.

64 **Myself:** The camel case.

65 **Participant:** I was interested in how it managed to string that together, that noun.

66 **Myself:** Hmm, so when it compiles this language, it has to compile it to another language
67 called "InstAL", and then that is compiled to another language which looks a bit like Prolog,
68 called AnsProlog, and that's a kind of declarative... it's a language where you describe
69 the problem, and then the program works out how to solve the problem. Which is called
70 "Declarative Programming".

71 **Participant:** So you've given it a blanket rule where if you've got a couple of capitals,
72 and then an "of" in the middle...

73 **Myself:** Yeah, yeah. But in order to translate it to that language I've said: "if you have
74 an entity with multiple words, convert it to this format, so it's one word". So that's kind of
75 just an output of that. OK, you've got a compile error. Did you change anything?

76 **Participant:** Yeah, I changed the "o" to a capital.

77 **Myself:** OK, OK, yeah. That would explain it. It has to be exactly the same.

78 **Participant:** I hope you don't mind me tinkering with your, uh...

79 **Myself:** Oh no, that's OK. So I think in this case, you'd have to change the declaration
80 on line 4 as well, to be the same thing. But that might work.

81 **Participant:** Let's give that a go.

82 **Myself:** Hmm, no I don't think that's worked. I think hyphens probably trip it up.

83 **Participant:** Yeah, sure.

84 **Myself:** I wonder why. I'll have to look into that. OK, moving on to task 2, I'd like you
85 to create a trope from scratch. So if you click on "new" underneath that text box, and then
86 type in the trope name first in that text box there. So what would you like to call it? OK,
87 could I ask you to remove the apostrophe, just in case? I don't think it will have a problem,
88 but just in case. Before you click "Save", type in "David is a role", and maybe one other role
89 or object.

90 **Participant:** Hang on. Yeah.

91 **Myself:** Great, OK. Then click "save", and go to "edit". Actually, it's a good idea to go
92 to "edit" so you can visualise that it's working as you're creating the trope. So, there you
93 go. "Dave's typical day". So if you click refresh at this point, nothing will appear, because
94 nothing's happening in your story yet. So add an event.

95 **Participant:** Add something? Yeah, sure.

96 **Myself:** Oh yeah, of course you need a place.

97 **Participant:** Uh huh. I can't remember how you declare objects.

98 **Myself:** "... is an object": "Coffee is an object".

99 **Participant:** OK. How would I say "Drink is an action"? I don't need that do I?

100 **Myself:** Actually no, you can just use that. So you can say "David drinks the coffee",
101 and that's fine. OK, save that and see if that works.

102 **Participant:** Yeah, sure.

Syntax: Us-
ing any verb

103 **Myself:** Click refresh as well. There might be a compile error because it wants a capital
104 "C" for coffee.

105 **Participant:** Not really anything to show, is there?

106 **Myself:** No, it should show the first event, "David drinks Coffee", but it wants the Coffee
107 with a capital "C".

108 **Participant:** (interrupted by colleagues) Please continue.

109 **Myself:** So, where it says "David drinks coffee", the "c" in coffee must be a capital C. So
110 that's one convention I had to do. And then try "refresh" and see if that's fixed it. There you
111 go.

112 **Participant:** Oh.

113 **Myself:** So yeah, it doesn't automatically refresh the visualisation if you click "save". I
114 did try to implement that, but I had some problems.

115 **Participant:** Matt, you don't have to explain a damn thing to me, this is quite an
116 achievement.

117 **Myself:** That's very understanding of you David, thank you very much. So, see if you
118 can add some more events to your trope. I think for this task, don't put any branches in yet,
119 just put in one thing after another for now.

120 **Participant:** Yeah, sure.

121 **Myself:** And if you have "David goes to the cafe", so it's grammatically correct, that
122 should work as well. It should ignore the "the"s. If you see what I mean.

123 **Participant:** I'm with ya. I think because I'm halfway between a programming and
124 writing mind, I'm going to like, programming. Just, you know, real BASIC stuff. Let's see
125 what happens. Let's see if it knows "orders". I doubt it. Let's try.

126 **Myself:** It doesn't actually know any of these verbs, it's using a tool called "Wordnet" to
127 convert them from the words... to detect if it's a verb or not basically. So it should be fine
128 with any verb you try. Hopefully. So, if you click "save" and "refresh", hopefully it will be
129 OK.

130 **Participant:** Yeah, cool.

131 **Myself:** There you go. Cool.

132 **Participant:** Looks right. Can you hold the line for 20 seconds?

133 **Myself:** Sure

134 **Participant:** (talks to colleagues). OK, Matt.

135 **Myself:** OK, cool. So that looks good. That's a good first, er... Looks really good.

136 **Participant:** Except it's put "David drink Coffee" at the beginning.

137 **Myself:** Interesting... so "drink_david"... oh, yes, so it starts...

138 **Participant:** That should be there, shouldn't it.

139 **Myself:** Because it's difficult to determine where a verb goes in a sentence, and it's
140 converting this directly from your sentence structure here to something else entirely, I've just
141 put the verbs at the beginning. So it's verb and then...

142 **Participant:** Oh hang on, so I've done it wrong. It's because we've declared that... you
143 see how I've highlighted number 5? We declared that as a verb: "David drinks coffee".

Syntax:
Capitalisation

144 **Myself:** Ah, yes.

145 **Participant:** When really, the story starts at line 6: "David goes to the cafe".

146 **Myself:** Oh, I see. So you were saying it's the wrong way round? So David should go to
147 the cafe first?

148 **Participant:** Well, I was trying to declare that as a verb, something which subject does
149 with object. However, what you've just described with the compilation, it doesn't need that
150 at all.

151 **Myself:** Oh, I see. I see. So you were thinking that it would infer that coffee was something
152 that David drinks because you declared it. So if David orders a coffee, then it follows that
153 David will drink the coffee. That kind of thing.

154 **Participant:** No, what I was trying to do... trying to CTRL-z on this, hang on... I
155 popped that in as almost like declaring variables at the beginning to say the kind of things
156 which an object can do to other objects.

157 **Myself:** Ah, yeah. That's a very Prolog-style of programming, actually, that's quite
158 interesting.

159 **Participant:** I don't know what that means, but OK.

160 **Myself:** Yeah, so it's actually related to the language which this compiles to, which is a
161 very obscure language called "Prolog", it's like, if you're writing a Sudoku solver. So normally
162 in programming you tell it exactly how to solve a program, right? But using Prolog, you just
163 tell it the rules of Sudoku, and it would work out how to solve a given set of numbers that
164 you put into it. It's like a very different way of programming. So what you're saying is, by
165 telling it the things you can do to the coffee, you're telling it the problem, and it could...
166 that's exactly how you'd program in Prolog. I'm not sure if you follow me, but that's quite
167 interesting.

168 **Participant:** OK.

169 **Myself:** So, that's good. So if you move onto task 4 now, actually. So add some branches
170 to the trope. There is a limitation. So yeah, continue with this trope. One limitation I should
171 mention is that you can't have a trope which is more than five events long at the moment
172 because once you combine lots of tropes together it can get very very complicated with more
173 than five events. So, you've got that. Add some branches to it now with the "Or" keyword.

174 **Participant:** I'm with ya. Just give me a minute to get creative.

175 **Myself:** Cool. No problem.

176 **Participant:** Um... yeah... I'm just referring back to the notes about the syntax. OK.
177 So I would pop in to the market, so I can put that in or I can come down here two spaces "Or
178 David asks Matt how he is".

179 **Myself:** Ahh, now that's tricky. "Matt how he is"... I wonder if that would work.

180 **Participant:** Doubt it.

181 **Myself:** Yeah, I doubt it too. See what happens, though. See what happens. I reckon it's
182 going to have a compile error.

183 **Participant:** Refresh...

184 **Myself:** No, it doesn't like that. Doesn't like that.

185 **Participant:** OK, let me try something else.

186 **Myself:** Yeah, sure.

187 **Participant:** Let's just give it a blunt thing of getting rid of those on the end.

188 **Myself:** I have a feeling that "health" needs to be an object or something, because...
189 but then... yeah. Which obviously isn't ideal. But you could probably add another kind of
190 keyword, like a Subject or something, or a Topic. But try that.

191 **Participant:** Oh, I could change object... uhh "Condition".

192 **Myself:** That won't work, that won't work. So I've only told it to have "role", "place" or
193 "object". But "subject" won't work either. It can only be "role", "place" or "object".

194 **Participant:** Yeah, I'm with ya. I see you've got a slightly different colour variation on
195 "is an object". I would make that a lot clearer

196 **Myself:** Yeah, so something that I did, which I probably shouldn't have done, is that it's
197 um... in this text editor I'm using, it's using Python syntax highlighting, which I thought
198 was a good idea at the time, but it's highlighting words which are just confusing. So, yeah.

199 **Participant:** No worries, just a comment you can bin it as you wish. Let me save the
200 trope, and see what happens.

201 **Myself:** No, it won't like that. Definitely won't like that. If you change "Health is a
202 subject" to an object...

203 **Participant:** Please repeat, Matt.

204 **Myself:** So change "subject" to "object" and see if that works.

205 **Participant:** OK. Save, refresh.

206 **Myself:** Ah, there we go. So that works. So you can imagine... that's interesting. It's
207 good that you've brought up that kind of question. Because it means that if I want to have
208 people asking people about other things, or talking about certain topics, I can add another
209 thing which isn't a role, a place or an object, it could be a topic or a subject, and that can be
210 another kind of thing.

211 **Participant:** That would be a really good thing. Let's try it to see what happens.

212 **Myself:** I KNOW it's not going to work. But yeah, why not? I would have been amazed
213 if that had worked. Alright, that's good, so you've got the branching in there, once that works
214 again. Cool. So, that's task four done. OK, in task five, I'd like you to click on the "arrange"
215 tab. So far we've been in that "edit" tab on the left, so click on "arrange". And then click on
216 the plus to combine, to merge this trope with another. So imagine you have a story, you have
217 multiple tropes happening at the same time. Go with a simple one like "Example 1".

218 **Participant:** Sorry, did you say "press on the plus button"? I lost the audio.

219 **Myself:** Yes, that's right. On the plus, yes. Then click on one of the simple tropes like
220 "Example 1", or something like that. And wait a little while, because it's generating all the
221 possible possibilities of combining these tropes together. It should have done it by now... ah,
222 there you go. So you can zoom in on that by hovering your mouse over it and scrolling up. And
223 then you can see all the different possibilities... so basically this is saying "In this story, we
224 have these two tropes, and all the events here are based on following one trope or another, so
225 these are all the different possibilities of the different orders of following the different tropes".

Feature:
Other types
of entities

226 **Participant:** Hmm. I suppose what we need... what I'd love to see here is the relation-
227 ship between. Oh... by the way, that's fantastic, that's really quite formidable.

228 **Myself:** Thank you. "Formidable"... that's a good one!

229 **Participant:** I'll tell you later how we can use this in flowchart decision making.

230 **Myself:** Oh, wow.

231 **Participant:** Yeah, this is very interesting.

232 **Myself:** So you can see the colours of the arrows.

233 **Participant:** Are all the permutations coming through here?

234 **Myself:** Sorry... how many permutations...?

235 **Participant:** Yeah, "go_hero_home" can spill off into "ask_matt_david". Oh... "ask_-
236 matt_DAVID"...

237 **Myself:** Ah yeah, sometimes it gets those orders mixed up. Yeah, that's a bug that I need
238 to fix.

239 **Participant:** I like the way that the Hero's going to the Land of Adventure, and I'm
240 heading to the cafe.

241 **Myself:** That could be its own Land of Adventure, depending on how much coffee you
242 have, I think.

243 **Participant:** Not in Bath! OK, yeah, what can I do for you next?

244 **Myself:** OK, that looks good. So task 6 is to embed a trope in another trope. So go
245 back to the "edit" tab there. So that's the trope you've already created. We want to embed
246 that trope into a new trope. So click on "new", and create a new trope. So, "David's typical
247 day" has to be kind of a sub-trope of some other trope, maybe "David's typical week", I don't
248 know. So type the name at the bottom for your new trope.

249 **Participant:** Yep.

250 **Myself:** Yeah, that looks good.

251 **Participant:** "Save trope". Edit?

252 **Myself:** Actually, I don't think it works unless you put in an event. So before you click
253 on an event, put "David" in, and...

254 **Participant:** Do I need to declare variables again?

255 **Myself:** Yes, you do. Yeah.

256 **Participant:** (types)

257 **Myself:** Ahh... "David is an object". Not a role?

258 **Participant:** Ah, that's the one. Cheers, mate. I suppose, because an object can be so
259 many things like we just describe, a topic, a condition of health, an abstract noun. Um, yeah.
260 I was clumping in my mind "object" all together.

261 **Myself:** Oh yeah, yeah, sure. I understand. But it has to be "a" (role), "an" will trip it
262 up. Put in another role if you can, and maybe the first event before you save it. Because if
263 you create a trope and save it, and it doesn't compile, it can mess it up. So "A Boat is an
264 object", good. Actually, I think if you save that, that should work, and then go to "edit".
265 And see if that's appeared...

266 **Participant:** Not appearing... yet.

267 **Myself:** Oh, in the dropdown, yeah.

268 **Participant:** Hang on, hang on. There we go.

269 **Myself:** There we go. OK.

270 **Participant:** Yeah, I'm there.

271 **Myself:** Great. So have a couple of events happen and then have the events of "David's
272 typical day" happen after that.

273 **Participant:** (types)

274 **Myself:** Hmm. I think it won't like "Sun is shining". I think if you can say "The Sun
275 shines", that might be better.

276 **Participant:** Yep, good. No, that's interesting isn't it. Present conditional or present
277 tense.

278 **Myself:** Yeah... that should be OK. And then "the" rather than "then". "The Sun
279 shines". So you could have maybe "The Sun shines" and then 'The "David's Typical Day"...

280 **Participant:** "David goes to Boat".

281 **Myself:** Ah yeah, and then "David goes to the Boat".

282 **Participant:** Oh, I see. I see. "David goes to the Boat" OR... I think the syntax did it
283 that way... "David..."

284 **Myself:** So, you have to put "David's"... I think it has to be 'The "David's Typical
285 Day" trope happens'. And the, so "The Sun shines"... don't indent line 6. Put that back
286 onto where... so delete those spaces. And put that back down to two spaces. So yeah, try
287 that. I have a feeling it might work, it might not work though.

288 **Participant:** Save trope? Problem is, we've still merged the Hero Adventure as well.

289 **Myself:** Ah no, it'll forget that once you click refresh.

290 **Participant:** Oh, I see.

291 **Myself:** I have a feeling there's still this bug where if you try to embed a trope in a
292 branch, it's not working, which is unfortunate. Instead of "Or", could you just put "Then",
293 and unindent it, and see if that works.

294 **Participant:** I'm wondering if THAT's the problem.

295 **Myself:** "Then David goes to Boat". Ooh, good point. I think it should be OK. So, just
296 change line 7 to "Then the... ", because I'm pretty sure that's not working, sadly. And then
297 delete those spaces at the beginning as well. Hmm, still a compile error, huh? OK. I wonder.
298 "The Sun shines"... So perhaps it doesn't like... Hmm. OK, delete line 6 and line 7, so cut
299 line 6 and line 7 so you can paste them in later. And then see if that works. I'm wondering.
300 Did you click save? You did? So, it doesn't like "The Sun shines". That's interesting. "The
301 Sun is an object", "The Sun"... "is shining"... No, that definitely won't work. Uh.. "The
302 Sun shines"! Give me a second. So, I can edit it from my end very quickly and see if I can fix
303 it.

304 **Participant:** Go for it.

305 **Myself:** Because sometimes it can be stuff like too many "the"s, or not enough "the"s.
306 "David's life in Bath". Actually, if you could put "The Sun is an object, The Sun shines", see
307 if that works.

Feature:
State man-
agement

Syntax: Use
of "the"

308 **Participant:** Could you repeat that? I lost audio.
309 **Myself:** Sure. "THE Sun is an object", and then "THE Sun shines". Sometimes that
310 trips it up.
311 **Participant:** Yeah, because it was the "is" as a declaration, wasn't it?
312 **Myself:** No, no, that's not working. OK.
313 **Participant:** Let's just try this.
314 **Myself:** No, I don't think it likes adverbs. "The Sun shines on...".
315 **Participant:** I tell you what, I could make this work by changing The Sun to The Weather.
316 And then say that "The Weather is good", then it would be subject-verb-object. Shall I give
317 it a go?
318 **Myself:** Yeah, try that. "The Sun shines, The sun is an object"...
319 **Participant:** Maybe the Weather needs to be declared as a role.
320 **Myself:** Hmm. Perhaps. Ah OK, interesting. So, for me, I have a feeling a person has to
321 be involved. There always has to be a role involved. So if you say: "The Sun is a role, The
322 Sun shines", that works. I think the way I've designed it is, because this is designed around
323 these intelligent agents, right? So it describes the constraints on these agents. So for every
324 event, an agent has to be in there somewhere, so there always has to be a role. So the sun,
325 er...
326 **Participant:** I say "subject", and you say "agent", is that correct?
327 **Myself:** I guess so... Like a role, yeah the subject would be... it has to be an agent.
328 Yeah, that's right, the subject has to be an agent of some kind. But OK, that's interesting. I
329 can't remember where we were. We were on some task.
330 **Participant:** We were going to embed a trope.
331 **Myself:** Yeah, OK, "The Sun shines", and then try embedding the trope. Ah yeah, OK,
332 there we go. And save then refresh.
333 **Participant:** Save the trope. Refresh.
334 **Myself:** There we go. Excellent.
335 **Participant:** It worked. Close. 'Cause really it's... what I'm aiming for is "If the Sun
336 shines, then David goes to Boat, Or then the "David's Typical Day" trope happens".
337 **Myself:** Yeah, I'm going to have to fix that bug where you can't for some reason embed
338 a trope ON a branch, if you see what I mean, which seems quite vital to me, discovering this
339 now talking to you, it seems like a very important thing to have. It's good that we've done
340 this study. OK, so final thing is if you just want to mess around with the tropes you have, you
341 can arrange them with that arrange tab, or change anything and just kind of like... yeah,
342 just create a story freely. I think you've kind of done a lot of that already, though. So yeah, I
343 think something you haven't explored much is the stuff on the "arrange"... so yeah, click on
344 "arrange", and see what happens when you combine tropes together.
345 **Participant:** Repeat the last 5 seconds.
346 **Myself:** Sure. Click on that "arrange" tab, and combine lots of tropes together. So play
347 with that a bit. So you've got "David's life in Bath", yeah. Ah, this might have some recursive
348 properties right? So you've got "David's Typical Day", which occurs inside "David's Life in

Feature:
Subject must
be an agent

Limitation:
Subtrope at
end

349 Bath", and then... oh, no, it's OK. No recursion, so it's OK.

350 **Participant:** It doubles up on "speak_matt_david", though.

351 **Myself:** I think that's because you have "David's Typical Day" in there twice, right?
352 Because "David's Typical Day" is contained inside "David's Life in Bath". And so, it will do
353 it as part of "David's Life in Bath", and then it will do it as part of "David's Typical Day"
354 as well.

355 **Participant:** Gotcha. Oh, is that your word recursive?

356 **Myself:** So that's what I was worried about, I was worried that it would get stuck in a loop
357 where it was doing "David's Typical Day" over and over again, like some kind of "Groundhog
358 Day" scenario. But no, it's OK.

359 **Participant:** Matt, it does feel that way sometimes.

360 **Myself:** (laughs). OK, cool. I think that's everything. So, I guess I've got some general
361 questions to ask you. I might have asked you all of them. How useful do you think it is to use
362 tropes to construct a story in this way, as a kind of component of story?

363 **Participant:** For who?

364 **Myself:** So if you're a story author, and you want to create a non-linear story with these
365 branches, how useful do you think it is to construct a story using tropes, in this way you have
366 been doing? I mean, if you're creating a computer game...

367 **Participant:** I think your layman author...

368 **Myself:** For yeah, a computer game author.

369 **Participant:** I like the idea of it being able to spit out all permutations. You know, I was
370 looking at some Google layout algorithm which compiled pictures of dinosaurs using botanical
371 (garbled) from the past.

372 **Myself:** So yeah, you're breaking up quite a bit there, could you repeat?

373 **Participant:** ...and that was amazing. You know, it's very much... Oh, I beg your
374 pardon. Can you hear me again, Matt?

375 **Myself:** I can hear you now.

376 **Participant:** Maybe I was just speaking too quickly.

377 **Myself:** I think our bandwidth is going down, though. Hang on. So you can stop the
378 screen sharing now, because we've done everything we need to do, and maybe that will conserve
379 some bandwidth.

380 **Participant:** (garbled)

381 **Myself:** So hang on, if I close that one. Hello? Ah, hello.

382 **Participant:** Hi there.

383 **Myself:** I think we were running out of bandwidth there.

384 **Participant:** That's fine.

385 **Myself:** You were just telling me about some Google software which generated all these
386 permutations of something.

387 **Participant:** Yeah, and I think for people who hit their creative end, of like just showing
388 different permutations of what is possible. I'm an architect, I deal with layouts and flows.
389 When I was in London, it used to be airports. They were hugely complex. And right now

390 I'm doing simplistic things of mainly loft conversions. But still, I have to give options to the
391 client, and so being able just to say... particularly when it's a stair layout. You know, if
392 I was able to just say "these are the options which are possible in this space for this new
393 staircase", that's brilliant, that saves hours and hours of work. You talk about authors and
394 you talked about before doing storylines. I can't speak to them, you'd have to ask them if
395 they think it's usable. I think for most creative people, the only time they would think it
396 would be more worthwhile to use a machine is if there's hundreds of permutations, or if they
397 just want all the permutations possible, or if they themselves have hit a creative wall. That's
398 my preconception, I don't know, you'd have to ask them. But what I'm saying that within
399 my line of work, you see, that's how I see (this being useful).

Usability:
Visualisation

400 **Myself:** OK, interesting that it could be useful for your line of work as well, though.
401 That's good to hear.

402 **Participant:** Let me get back onto the language stuff, because I know that's where you're
403 really aiming for. Well, I don't know. If your author was a Dali-ist, kind of author, who did
404 everything in abstractions, then that would be great. They don't have to keep taking the drugs
405 to come up with wild and wonderful things. Then it would be worth them constraining their
406 way of thinking to what the software demands. Then they would do this trade-off, thinking:
407 "Well, if I kind of put a bit of effort here, I would greater results that this way". I think for
408 your average Dick Francis, though, they would think: "No, I know what I want my characters
409 to do.". But that's me off the top of my head without knowing these people.

Usability:
Visualisation

410 **Myself:** I can definitely understand that.

G.3 Participant C

Dialogue starts at 13:20.

1 **Myself:** That's everything I have to explain about that language. Do you have any
2 questions about it?

3 **Participant:** I think it's pretty simple.

4 **Myself:** OK, cool. So do you have any programming experience at all?

5 **Participant:** None.

6 **Myself:** OK, that's good. That's what I wanted. Right, so if you go to the other tab with
7 the information in it. I've got some tasks that I'd like you to do. The first one is just to pick
8 a trope that's already there and edit it in some way. So you can do whatever you want to
9 it. There are some limitations that I have to explain. The first one is that you can only have
10 up to five events long, and I've put that limitation on there because once you have multiple
11 tropes combined, which we haven't done yet but we will do, it can get very complicated, so
12 I've limited it to just five events. So, yeah, pick any of those tropes and change it in some
13 way.

14 **Participant:** Right. How would you, like, edit it? What would you change, exactly?

15 **Myself:** So, I guess instead of the Hero killing the Villain, the Hero could meet the Villain,
16 or you could change the name of the Villain to "The Puppy". I don't know. Stuff like that.

17 **Participant:** That would be quite funny. I might do that one.

18 **Myself:** OK, cool. So you have to make sure to change it, where it says "The Hero kills
19 the Villain", it has to say: "The Hero kills the Villain Cat", because otherwise it won't work.
20 Ok, so then click "save". Oh yeah, you don't want it to kill the Villain Cat. So, I think it
21 has to be lower case "B". I think it would work either way, actually, but just in case. So click
22 "save", and then "refresh". There you go! Excellent. OK, cool. So that's the first task done.
23 Second task is to create a trope. But just a very simple one which doesn't have any branches.
24 So you can choose the Hero, Villain, or Mentor, and some places, so Home or Evil Lair.

25 **Participant:** Do I click on "new", or...?

26 **Myself:** Yeah, create a brand new one, so click on that "new"... there you go. So, type
27 the name first, in the "trope name" box.

28 **Participant:** Can I go rogue?

29 **Myself:** Yeah, that's cool.

30 **Participant:** Now, what was it called? It's not "object". How does...?

31 **Myself:** This is the name of the trope.

32 **Participant:** Yep. Oh, trope name? So just "rogue"?

33 **Myself:** Could you... I think... there used to be a bug where it didn't... so if you have
34 a thing called "Rogue" in the trope, it didn't like that. So could you call it "Rogue trope" or
35 something. I'm not sure if that bug's still there, but just in case, call it "Rogue trope".

36 **Participant:** Should "trope" be like, capital as well?

37 **Myself:** It doesn't... yeah, just in case. Yeah.

38 **Participant:** Trope... save?

39 **Myself:** Don't save it yet. Type some stuff first. Oh, sorry, so delete those words and
40 type it in the text box above. Don't delete it all, it still needs the name in that trope name.

41 **Participant:** (laughs). So type "Rogue Trope", and then...

42 **Myself:** Yeah. And then go into that text box above, and start typing "Rogue is a role",
43 or something like that.

44 **Participant:** Is it role?

45 **Myself:** Yeah, that's right.

46 **Participant:** "Rogue is a role".

47 **Myself:** Hmm. I think that will work. Yeah, sometimes it doesn't like lots of words. I
48 think that's OK. Probably. We'll see.

49 **Participant:** Should I save it or carry on?

50 **Myself:** Yeah actually, save it, and then actually go to edit and choose the trope from
51 there. And that way, when you save it you can visualise it. There you go: "Rogue Trope". So
52 actually, if you try to visualise it now, nothing will appear because there's no events.

53 **Participant:** No. There's no events happening. We need an event.

54 **Myself:** Yeah.

55 **Participant:** I think I have to click... for some reason it's stuck on the first...

56 **Myself:** It's stuck?

57 **Participant:** It's like frozen.

58 **Myself:** Oh, that's weird. Oh, it's alright now. OK. Haven't seen that before.

59 **Participant:** Carry on. So now what?

60 **Myself:** Oh wait. Oh no, this is OK. In the first task I was limiting you to certain
61 character roles, but I think it's OK. Just keep doing your own thing, it's fine.

62 **Participant:** Don't know how "Rogue" came to my head, I was like "yay!". Um, so...
63 What would I type? So, like, "Rogue searches for Knife", or...?

64 **Myself:** Yeah, that's fine. I think Knife has to be capital "K".

65 **Participant:** Oh yeah.

66 **Myself:** See if that works. Oh, get rid of the space at the end of that line. Yeah. OK.
67 Save that and see... oh. Click refresh just in case.

68 **Participant:** "Compile error"!

69 **Myself:** So it's not a very useful error message, sadly. Because it was difficult to get
70 anything more useful than that. It might... so there's a couple of things it might not like.
71 For starters, you've put "Knife", but not "Knife of Immense Power", where "Rogue searches
72 for Knife". That might be that. I have a feeling there might be other problems, though. But
73 try that, then click "save", and then "refresh".

74 **Participant:** No. Maybe it just doesn't like the word "rogue".

75 **Myself:** No, I think "rogue" should be fine. So delete... just call it "Knife" rather than
76 "Knife of Immense Power".

77 **Participant:** Aww.

78 **Myself:** Because it might not like that.

79 **Participant:** Gets rid of all the creativity!

80 **Myself:** OK, save and refresh.

81 **Participant:** Nope, compile error.

82 **Myself:** No, OK. Try "The Rogue is a role, The Knife is an object".

83 **Participant:** With a capital Knife? Capital K?

84 **Myself:** It shouldn't make a difference, but... and then "The Rogue searches for the
85 Knife". Yeah, try that. Ah, OK. It's still not working. I will have a look at your trope and
86 see if I can get it working. "The Rogue searches for the Knife", "The Rogue"... It might not
87 like... hmm. It should like "searches for", though, I'm sure I've used that before.

88 **Participant:** It sort of went purple.

89 **Myself:** Ah, yeah, I'm actually using the syntax highlighting for a different language, so
90 it doesn't mean anything, unfortunately. OK, "looks for" is OK. Hang on. "Search"... it
91 doesn't like "searches for", but it likes "looks for" for some reason. So save that and refresh.
92 Ah, that's weird. Why... I don't know why that is. So I...

93 **Participant:** Look Rogue knife?

94 **Myself:** Oh yeah, yeah. Again because the verb is always at the beginning. So it's like
95 "The Rogue looks for the knife". OK, keep going. Strange that it doesn't like "searches for",
96 but "looks for" is alright. Oh yeah, so try putting that in, yeah.

97 **Participant:** Otherwise the Knife is a bit boring. (indistinct)

98 **Myself:** OK, see if that works.

Syntax:
Capitalisa-
tion

Syntax:
Capitalisa-
tion

99 **Participant:** No!

100 **Myself:** OK. Try the "Knife of Power", because that...

101 **Participant:** Knife of Power! No, it just doesn't like knives of power at all.

102 **Myself:** That's weird. I wonder why that is. I think I just limited it to two words

103 for names of things, sadly. So you can call it "Power Knife". It's not very creative, sadly.

104 Although no, I didn't do that, because "Land of Adventure" is three words.

105 **Participant:** Yeah.

106 **Myself:** Hmm, I'm not sure why that... hang on. Oh no, it DOES like "Knife of Power".

107 Hang on... So I'm editing it myself here.

108 **Participant:** Yeah yeah. "Knife of Power"! (indistinct)

109 **Myself:** So it doesn't like "Knife of Immense Power", but "Knife of Power"... it's OK if

110 "Power" has a capital "P". So hang on, I think that's what I've done, I've said "Every word

111 has to be capitalised". But then "of" is OK.

112 **Participant:** Yeah, because it's a combine-y thingy, isn't it.

113 **Myself:** Yeah, but I haven't told it that. Ah yeah, so yeah, "Knife of Immense Power" is

114 fine if "Immense" and "Power" have capital letters at the start.

115 **Participant:** Oh yay! OK, cool. I like it.

116 **Myself:** Cool.

117 **Participant:** "Knife of Immense Power"!

118 **Myself:** Cool. Nice. So what happens next?

119 **Participant:** Ah, "Compile error".

120 **Myself:** Hmm. Try deleting the line at the end, the blank line. Oh! No, I know what it

121 is. So it doesn't like it when these declarations, when you say "The Vendor is a role", that has

122 to be before anything else, that has to be before any event. Save that and refresh. OK, cool.

123 Alright, I think that's enough. So if we go to task 3. Oh you've kind of done that already, it's

124 adding objects to your trope. So let's go to task 4: add branches to your trope. So add some

125 kind of branching event where something happens or something else happens.

126 **Participant:** A branch...

127 **Myself:** So to do that, on the next line you do "Or something else happens", but you

128 have to do two spaces and then "Or". OK, that sounds good. OK. Oh no! "Evil People is a

129 role"... try "The Evil People is a role". That doesn't make sense.

130 **Participant:** Evil People! "The Evil People"... oh yeah, because there's that down here.

131 **Myself:** Yeah, I think that's confusing it. OK, OK, cool. So the way I did program this,

132 or at least so I thought, was that it ignores the "the"s, but it isn't ignoring them. Or it's... I

133 guess it's ignoring them if they're there, but it's expecting them to be there, which isn't what

134 we want.

135 **Participant:** This one's a capital "The".

136 **Myself:** I think in the case of the... it doesn't matter if it's capital or not. But it just

137 wants the "the"s to be there. Which is annoying. Alright, I'll have to fix that. OK, so that's

138 good. You've got the branching going, that's great. Oh, so task five. This is cool. If you

139 scroll up to the top of the page. So this is where you want to combine two tropes together.

Syntax:
Capitalisation

Syntax: Entity declarations

140 So click on the "arrange" tab there. So you've got your trope there. If you want to combine
141 that with another trope, click "+", and then choose one of the other ones. Preferably a simple
142 one, like "Example 1". Then we can... because it takes a little while actually. Because
143 there's complicated stuff going on underneath where it's combining everything. And then
144 you'll see, you'll get this big tree appear... eventually. There you go. So this is all the
145 different possibilities, combining these tropes together.

Syntax: Use
of "the"

146 **Participant:** Wow! That's pretty cool.

147 **Myself:** And if you zoom in, so if you scroll... hover over and scroll up. Put your mouse
148 over the visualisation and scroll up with your scroll wheel.

149 **Participant:** Oh!

150 **Myself:** Oh no, it's not doing it, though. Do you have a mouse?

151 **Participant:** I have, like, a laptop.

152 **Myself:** Can you scroll... do you know how to scroll with your touchpad, your trackpad
153 on the laptop?

154 **Participant:** Yes, it's like this, but it's not doing it.

155 **Myself:** Oh, that's not good. I guess you could do it without scrolling. But you can see
156 that there's the blue lines and the red lines. And the blue lines are one of those tropes. I
157 can't quite read it. And the red lines are the other. So you can see which event follows which
158 trope, according to each decision.

159 **Participant:** Even the writing itself is tiny.

160 **Myself:** Yeah. You should be able to zoom in, but it's not working.

161 **Participant:** Ooh, I got closer somehow.

162 **Myself:** Ah yeah, it should be if you scroll up, or try scrolling down. It's not doing it
163 though, that's weird. You could try... let's see. No, I don't know what else would do it. I
164 think scrolling is the only way. You've managed to do it somehow, mysteriously.

165 **Participant:** Mysteriously! Don't ask me how, I do not know.

166 **Myself:** OK cool, so you've combined two tropes. You can try combining other ones now.
167 So instead of example 1, you could try a more complicated one. But it might take a little
168 while. Example 3 was pretty complicated.

169 **Participant:** I'm going to do 2. Ooh, nice!

170 **Myself:** Yeah.

171 **Participant:** That's pretty cool.

172 **Myself:** Cool. Alright, so the next task is... I want to to take your existing trope and
173 embed that into another trope. So create a new trope. And give it a new name.

174 **Participant:** I guess let's go stereotypes.

175 **Myself:** The Princess trope... oh good. OK. Actually, if you go to edit, see if it appears.
176 Sometimes if it doesn't compile, it gives it a blank name, which you don't want. Oh, it's given
177 it a blank name. OK, go back to "new", and put some stuff in there like "Princess is a role".
178 OK, and then click "save", and then go back to "edit".

179 **Participant:** Nope.

180 **Myself:** Hmm. Put a first event in there. Go back to "new". I'll just go and delete those
181 blank ones. Sorry?

182 **Participant:** "The Princess is a"... ah.

183 **Myself:** I think you have to put some event in there. Oh no, it's saved that one. I guess
184 it has to have... so if you go to "edit", and select the "Princess Trope", it's appeared there.

185 **Participant:** Cool.

186 **Myself:** I think it has.

187 **Participant:** We'll find out.

188 **Myself:** Yeah.

189 **Participant:** Oh yeah, it's there twice.

190 **Myself:** OK, cool. Choose one of them. Oh, actually delete one of them, it might get
191 confused.

192 **Participant:** Yep.

193 **Myself:** OK, so now add some kind of events in there.

194 **Participant:** Um. Ah - "Compile Error".

195 **Myself:** "The Princess hits the Guard"... oh, that's strange, that should work. OK,
196 "hits the Guard", I don't know why it doesn't like that. Try lowercase "t" rather than capital
197 "T". Try that. Oh, that's weird. It wanted a lowercase "t". Alright, OK. Add another event.

198 **Participant:** Cool. "Get Princess Key"!

199 **Myself:** So with the "Or", it has to be capital "O".

200 **Participant:** Plus two spaces.

201 **Myself:** Yeah, that's right.

202 **Participant:** That the one?

203 **Myself:** Yeah, that's right. OK, save... oh.

204 **Participant:** "Compile error", oh wait...

205 **Myself:** Oh no, it's OK. "Princess hits the Guard"... ah, yeah another thing is it looks
206 like... so the verb is always the first word, but then those words, it always puts in alphabetical
207 order I think, so it looks the "Princess hits the Guard" is the same as "The Guard hits the
208 Princess". That's not a great representation. Ah, OK, never mind. Alright, so, for your last
209 event, put like 'Then the "Rogue Trope" happens'. So maybe delete the spaces and put "Then
210 the"...

211 **Participant:** Capital "Then", or...?

212 **Myself:** Oh yeah, capital "T". And "Rogue Trope" has to be in double quotes.

213 **Participant:** Oh yeah.

214 **Myself:** "Rogue Trope"... I think you have to put 'The "Rogue Trope" trope happens".
215 Which is a bit redundant. Trope has to be lowercase "t".

216 **Participant:** OK.

217 **Myself:** OK, see if that works.

218 **Participant:** "Compile error"!

219 **Myself:** No! Huh, OK. Yeah, I'm not sure why that's not working. Try... so instead
220 of that branch of "Or the Guard hits the Princess", try just putting "The Guard hits the

Syntax:
Capitalisa-
tion

221 Princess".

222 **Participant:** "The Guard hits the Princess".

223 **Myself:** Yeah.

224 **Participant:** Nope.

225 **Myself:** Oh, that's weird.

226 **Participant:** Oh wait... yes.

227 **Myself:** Oh, it's OK. OK. Should have tested this a bit more. I'll have to fix this. So
228 it doesn't like there being some kind of branch before when it puts the sub-tropes in. OK.
229 That's good to know. What else? So we've done task 6. OK, for task 7, just mess around
230 with that and combining tropes with the "arrange" thing. So try creating a story for yourself
231 and combining them together using existing tropes, or you can create some more if you want.
232 Using that "arrange" tab from before.

233 **Participant:** But how can I edit, like, the Rogue trope, do I have to go back to the Rogue
234 Trope?

235 **Myself:** Yeah, you do. So notice that you've got five events total when you combine those
236 tropes together. So you can't have any more events.

237 **Participant:** Ah. Can I have an "Or"?

238 **Myself:** You can, yeah, you can add more "Or"s. It looked like it was going to work, but
239 it's not any more. Oh it's because you haven't put "The Princess is a role" at the top.

240 **Participant:** Oh, OK. Nice.

241 **Myself:** Cool. Do you want to try clicking on the "arrange" tab and combining tropes
242 together?

243 **Participant:** OK. Plus... it's kind of like combining two already though, hasn't it?

244 **Myself:** Yeah, yeah. Kind of. But it's put them in a kind of strict sequence, so that one
245 always happens after the others, but this kind of merges them all together.

246 **Participant:** Right. So, Example 3.

247 **Myself:** It sometimes takes a little while. I think Example 3 was quite complicated, so it
248 will be interesting to see what emerges.

249 **Participant:** I'll have no idea though, because it'll be too small.

250 **Myself:** Oh yeah, that's a good point. This one might be really small. When it does
251 appear.

252 **Participant:** It's taking a long time.

253 **Myself:** Hmm. OK, I'll ask you some questions while we're waiting for that. So, do you
254 think that this language is easy to pick up as a non-programmer? Is it easy to learn?

255 **Participant:** Yes, but obviously I needed your help sometimes because I didn't know why
256 it wasn't working.

257 **Myself:** Yeah, OK. So imagine all the bugs have disappeared. So in that case, do you
258 think it would be easy to use?

259 **Participant:** Yes, I would say so.

260 **Myself:** OK, cool. OK, great. So do you think that composing stories out of tropes like
261 this is a convenient way to make stories?

Limitation:
Subtrope at
end

Limitation:
Five event
limit

262 **Participant:** I'd say yes, but I think that it also helps that I am a creative writer, so I
263 automatically think of a story pattern.

Usability:
Tropes

264 **Myself:** Hmm, OK.

265 **Participant:** So like creating a story pattern on the builder is a lot easier, because I know
266 how to, like, combine them. So in my head, I always view that the Princess will meet the
267 Rogue eventually. And I had to create the second trope.

Usability:
Tropes

268 **Myself:** OK, so, I think that's good because it is kind of designed more towards creative
269 thinkers and story authors rather than programmers. So it's good that that helps.

270 **Participant:** Like if I didn't have that already in my head, I don't know... I think I'd
271 be a bit more lost.

272 **Myself:** OK. Is there anything you think that would be difficult to describe with tropes?

273 **Participant:** I dunno, everything is a trope, though.

274 **Myself:** Oh, it's finally done it! It took a while.

275 **Participant:** It's tiny.

276 **Myself:** OK, I guess if you can't zoom in, we can't really see the details. That's a shame.
277 So, OK, do you think there are tropes you wouldn't be able to describe with this language?

278 **Participant:** Probably if you wanted characters who aren't basic characters like the
279 Princess and the Rogue. It might be a bit more tricky. Like if you want characters that are
280 multi-layered as well, like the Princess, but she's also a Ninja, then you've got to say like "The
281 Ninja Princess".

282 **Myself:** Yeah, I understand what you're saying. You want something in there which
283 describes how a certain character behaves, right? Because she's a Princess, she does this, etc.

284 **Participant:** Yeah, but like say also in storytelling you don't reveal everything at once.
285 So like, normally she'd be a princess until it's suddenly revealed that she's a ninja.

Feature:
Story struc-
ture

286 **Myself:** Ah OK. So, she's not a ninja from the beginning, she can become a ninja at some
287 point in the story. Well, she is a ninja at the beginning but it's not revealed until later.

288 **Participant:** Yeah, but you don't know.

289 **Myself:** Yeah, OK, interesting.

290 **Participant:** The same with like, "The Knife of Immense Power", you don't know if "The
291 Knife of Immense Power" has a fallback that creates, like the person who touches it turns evil,
292 then say the rogue would become "The Evil Rogue", but like the system wouldn't recognise
293 that.

Feature:
Story struc-
ture

294 **Myself:** Hmm, I see. Yeah, so you'd want something... not only would you want a
295 character to be able to change roles at some point in the story, you'd want something like an
296 object which can do that. Yeah, OK, I see.

297 **Participant:** And like, "object development", I guess. It's the biggest part in the story
298 is like the build-up and the reveal. You can't really reveal when it's written in from the
299 beginning.

300 **Myself:** Yeah, that's a good point. OK, cool. Do you have any other general thoughts
301 about this tool? Do you think it's useful to be able to visualise this story like this?

302 **Participant:** I think it's very useful, but obviously it'd be handy if I can actually see it.

303 **Myself:** Yeah, sadly, you can zoom in on it, but if you have a mouse with a scrollwheel,
304 I think.

305 **Participant:** Yeah, which I don't.

306 **Myself:** OK.

307 **Participant:** It's a bit like, laptop bias.

308 **Myself:** Yeah, I guess so. I can do it with my laptop, but I've got like the two-finger scroll
309 thing, which works OK, where you drag two fingers over the trackpad. But that doesn't work
310 on every laptop I think.

311 **Participant:** Mine only does, like, the sidebar. It doesn't...

312 **Myself:** Ah, OK. I think that's a Windows...

313 **Participant:** Ah wait! It works.

314 **Myself:** Hey!

315 **Participant:** So what you said literally works, that's perfect.

316 **Myself:** So you can see you've got the three tropes there, you've got a green line as well
317 as the... ah because yeah, the "Princess Trope" includes the "Rogue Trope". Cool.

318 **Participant:** Fun times. I'm so glad we discovered this. So as long as someone knows
319 about the two scrolly-thingies, and it works, it's fine.

320 **Myself:** OK, cool. I'll stop recording now.

G.4 Participant D

Dialogue starts at 6:23.

1 **Myself:** Right, so have you got any questions about that language?

2 **Participant:** It just occurred to me: what happens if you do it recursively? What if
3 "Item Search" also calls "Kill Then Search"? Presumably it would get...

4 **Myself:** Uhh, yeah that's a very good question. That's a really good question! OK, you
5 can try it out. If you go to "Item Search"...

6 **Participant:** Let me just copy the name of this one, "Kill then Search".

7 **Myself:** I have a feeling it will not work, because this is very brittle. So yeah, put 'Then
8 the "Kill then Search"... and the "Kill then Search" has to be in quotes... "...trope
9 happens".

10 **Participant:** "...trope happens". Is that right?

11 **Myself:** Yeah. Save that.

12 **Participant:** Uhh... "save trope".

13 **Myself:** And then click "refresh".

14 **Participant:** Ah, OK. It's just ignored it, basically. It seems to have just thrown that
15 one out.

16 **Myself:** Ah, that's lucky. That's good. As long as that's not broken my thing running
17 on the server, then that's great. I wonder. I'll try that later and see what's happening in
18 the background. Because it shouldn't just ignore it. But I'm glad it hasn't broken it. Right,
19 so that's good, that's actually a good thing to bring up, which I kind of hadn't considered.

20 So yeah, save that one and do that again. So one thing I should mention is that this is like
21 a really basic proof of concept, I've got limited time to do this, so I've really implemented
22 the very basics of this language. There's stuff that you're going to notice which would be a
23 good idea to put in. And that's actually part of the point of this study, is like, just for you
24 to identify what I should be adding to this language. Because I've just put these basics, you
25 know, just events, one after another, and then very simple branching, and then calling other
26 tropes. But, yeah, I'd like to hear your opinion about what's missing. OK, so we'll start with
27 these tasks. Also, yeah, there are lots of bugs, which I'll point out some of them ahead of
28 time. So task 1 is: edit an existing trope, so select perhaps one of the simple ones like the
29 example tropes, and just edit it to change it however you want.

Limitation:
Five event
limit

30 **Participant:** OK, cool.

31 **Myself:** Did you... the "Home" declaration has disappeared. Was that not there before?

32 **Participant:** Oh, yeah. I don't think I changed that.

33 **Myself:** "The Hero goes Home"... hmm. Somehow it still works, though. Oh, OK.
34 It's because when I put it in... so every time I do a study, I copy and paste this in from
35 somewhere. And what it's done is it thinks "go home" is a verb with two words. So it kind
36 of still visualises it. So yeah, actually, could you put in that declaration of home, "Home is a
37 place"?

38 **Participant:** Yeah OK, so lowercase "place".

39 **Myself:** Yeah.

40 **Participant:** Is it "The" Home is a place?

Syntax: Use
of "the"

41 **Myself:** "Home is a place"... either should work. But just put "Home". Oh no, it's still
42 using the old one. You have to click "save", then "refresh".

43 **Participant:** Oh, OK.

44 **Myself:** Yeah, "go_hero_home", cool.

45 **Participant:** "go_hero_home", makes sense. Cool. Erm, so I could do something like:
46 "Sword is an object, Home is a place", er, "Danger Mountain is a place". How's it going to
47 deal with that space, is it going to be OK with that?

Syntax:
Spaces in
words

48 **Myself:** It should be OK with that.

49 **Participant:** Er, "The Hero goes Danger Mountain".

50 **Myself:** You might want "goes TO Danger Mountain". Well, see if that works. It might
51 do, it might not. So save it, and click "refresh". I think it's OK. Yeah, there you go. "go_
52 hero_dangerMountain".

53 **Participant:** Oh, OK. Cool. "Hero goes to Danger Mountain". Erm, yeah, "Then he
54 finds the Sword, or the Hero dies". Can you sub-say, can you say "The Hero finds the Sword",
55 and then does something, or something else? Like, it gets confusing with the nesting, doesn't
56 it, because that's one level of nesting, but can you go to a second level of nesting?

Limitation:
Branch
length limit

57 **Myself:** Ah, sadly not, because I've kind of limited it so that there's only one level of
58 nesting. Which is very limited.

59 **Participant:** Otherwise it would get very complicated, wouldn't it?

60 **Myself:** Yeah, but I think that is actually essential if I want to have like a proper branching

61 thing eventually. It's something that I didn't really realise until I started doing these studies.
62 But yeah, I think that is something that's come up that I'll have to add to this language after.

63 **Participant:** Makes sense. Erm, "The Hero finds a Sword, Or the Hero dies".

64 **Myself:** Ah, that's enough actually if you...

65 **Participant:** Cool.

66 **Myself:** So that's the first task done. Do you have any general comments so far about...
67 do you think that this language would be easy to learn for someone who hasn't done any
68 programming before?

69 **Participant:** Yeah. I think in some ways it might be easier for someone who hasn't done
70 programming before to learn than someone who has. Like one of the things that confuses me
71 a bit is intuitively thinking about the grammar, because there's obviously some clever thing
72 going on, kind of similar to what Ruby does, where Ruby has this whole thing about how
73 it translates grammar. It seems to be aware of whether "to" should go before something, or
74 things like this. So it's kind of less predictable in a way.

Usability:
Ease of use

75 **Myself:** Are you talking like, erm, with that Rspec thing? Is it Rspec? Ruby has this
76 kind of natural-language testing language, which is a DSL for writing tests or something. Is
77 that what you're talking about?

78 **Participant:** Er, that sort of thing, yeah, that's right. When you're working on Rails.
79 I did this years ago, so I hardly remember. There's this whole thing about the grammar
80 whereby if you name an object a plural, Ruby will recognise that and treat it correctly, and it
81 can generate plurals of objects and things like this.

82 **Myself:** Yeah, that rings a bell. Yeah. I haven't used Ruby for a long time. I'm probably
83 in the same situation as you, I worked with Rails a long time ago, and haven't used it since.

84 **Participant:** Same. Yeah, I'm just drawing the comparison because of this thing about
85 the grammar is slightly less predictable. Because I'm not sure if... shall I assume the in-
86 terpreter isn't clever enough, and just write it in a literal way, or should I just write it in a
87 human way and just trust the interpreter to do something clever and understand it?

Syntax:
Spaces in
words

88 **Myself:** That's a good point, it is quite opaque. You weren't sure if "Danger Mountain"
89 was going to be recognised because it's got a space there, and then putting the "to" after
90 "goes to"... It's not too bad, because once you've declared the things it knows that has to be
91 somewhere, it knows what those things are. And also the verbs are going into Wordnet, so it's
92 kind of stemming the verbs, like, you know, putting them into their basic forms, so it knows
93 that "goes" is "go", "finds" is "find", etc. But it's generally OK, it's reasonably robust, but
94 still quite easy to break. But yeah, that's a good point. I think it might be easier for someone
95 who's quite naive and not used to programming, who isn't aware of the complexity of these
96 things. Hmm, OK, yeah, good point. Right, OK, so if you go to the second task.

97 **Participant:** Alright. "Create a trope". Right, "Given the following character roles and
98 the following places, construct a trope using TropICAL."

99 **Myself:** So for this one, I just want you to create a one-thing-after-another sequence of
100 events. So before we start, type in the name of the trope, in the "trope name" box. So that
101 should be at the top, really.

102 **Participant:** Um, "Super Duper Trope".

103 **Myself:** Yep, that should be OK.

104 **Participant:** "The Hero is a Role". Yeah, is that right?

105 **Myself:** "role" should be lowercase "r".

106 **Participant:** "The Hero is a role, Villain is a role, Mentor is a role". It would be nice to
107 be able to just do "Hero, Villain, Mentor are roles", like that.

108 **Myself:** OK, that's the kind of comment I want, because obviously I've implemented the
109 most basic version of this language, but I do want to know how to improve it and make it
110 more intuitive, so thanks for that feedback.

111 **Participant:** Yeah, no problem, it seemed like a fairly straightforward thing to stick on
112 top.

113 **Myself:** I think... so this language is kind of based on this other language called "Inform",
114 which people used to write text adventures, and it looks a lot like this. And I think they have
115 a mechanism like that, where you don't have to keep typing "so-and-so is a so-and-so", you
116 can just do the words separated by commas, and just do it that way.

117 **Participant:** Makes a lot of sense.

118 **Myself:** Yeah. OK, so you've got all those things...

119 **Participant:** Um, so then just put them in some sequence and doing some things?

120 **Myself:** Yeah, so could you just get rid of the blank lines, because I think sometimes that
121 trips it up. It shouldn't do, but yep. OK. And that one.. yep. So have no blank lines.

122 **Participant:** OK. So, a thing happens... "The Hero"... let's say, start with "The
123 Villain goes"... How's it going to behave if I put "The" before? "... goes to the Evil Lair".
124 Um.

125 **Myself:** Actually, could I ask you to save it.

126 **Participant:** Oh yeah.

127 **Myself:** And you can't actually visualise it from this thing, but go to "edit", and select
128 the trope you've saved.

129 **Participant:** Right.

130 **Myself:** There you go. And then you can visualise it. And see... so "compile error".
131 It might be because you've got a blank line at the end, so try that. Then click "save", and
132 "refresh".

133 **Participant:** No, it doesn't like it.

134 **Myself:** Oh, that's interesting. So "The Villain goes to the"... I think it might want
135 "The" in front of the declarations, so "The Hero, The Villain, The Mentor".

136 **Participant:** Oh, right. OK.

137 **Myself:** And I think it has to be capital "T" as well.

138 **Participant:** Capital what, sorry?

139 **Myself:** Capital "T", at the beginning.

140 **Participant:** Oh, right.

141 **Myself:** So try that. There you go, cool.

Syntax:
Capitalisa-
tion

142 **Participant:** Nice, right. Um. "The Villain goes to the Evil Lair". Presumably the Evil
143 Lair can't do something, because it's a place?

Feature:
Subject must
be an agent

144 **Myself:** Oh, that's a good point. It can, I haven't actually put anything to stop it from
145 doing things. So you can try that and see what happens. So yeah, a good way to try to break
146 it. I think...

147 **Participant:** It looks like it hasn't, er...

148 **Myself:** Click "save" again. Ah there you go, "compile error".

149 **Participant:** OK, so it doesn't like places moving to other places.

150 **Myself:** Which, actually, I can't think of any reason why not. Because when this is
151 compiled, it's just like... it just has statements that, you know, one thing is a kind of other
152 thing, but it doesn't put any limitations on what those things can do. So, huh, I wonder why
153 it's not liking that. There might be another reason for it.

154 **Participant:** "Evil"... oh, have I...? No, I have spelled that right. "goes to the Land
155 of Adventure", "Land of Adventure is a place". It looks fairly sensible, doesn't it?

156 **Myself:** Yeah. Hmm, interesting.

157 **Participant:** I think it's fair enough to call it a compile error, if you're trying to make
158 one geographical location stand up and move to another one.

159 **Myself:** Ha, yes.

160 **Participant:** Erm, so I'll stick an object in there as well. What've we got? "The Sword
161 is an object".

162 **Myself:** So, you have to put declarations at the beginning.

Syntax: En-
tity declara-
tions

163 **Participant:** Ah, OK, that makes sense actually.

164 **Myself:** It's just another limitation which saved me a lot of work.

165 **Participant:** I think that makes a lot of sense actually, because otherwise you can't really
166 do it intuitively. Yeah. I'm trying to think of a word. You can't do it in time, basically.
167 Chronologically. Um... "The Villain goes to the Evil Lair, Or he"... can you say something
168 like: "Either this role does this, or it does nothing"?

169 **Myself:** Uhh... you can say it. It won't have any semantic meaning, but you can do
170 two spaces, then an "Or the Villain goes nowhere". But then nowhere isn't a place. It would
171 probably interpret that as a separate verb.

Feature:
Event seman-
tics

172 **Participant:** It would just, like, assume this is a thing, and it wouldn't bother to know
173 the meaning of it, it would just say... "it does this thing".

174 **Myself:** Yeah, exactly.

175 **Participant:** Makes sense.

176 **Myself:** Oh, no. I think you can say, maybe not in this version, but I had at one point
177 you being able to say "The Villain may not go to the Evil Lair", and it would prohibit the
178 Villain from doing that. But in this case, it's kind of the case that it's prohibited from doing
179 that by default, so there's no point in kind of specifying that. OK, so I think "Or" has to be
180 capital "O".

Syntax:
Capitalisa-
tion

181 **Participant:** Ah, right.

182 **Myself:** And be careful with that "the Hero", I think it wants a lower-case "t" as well.

183 Yeah, so I'm sorry about the lack of information in the compile messages, that's just the way
184 it turned out sadly, it's kind of hard to give any information that would be relevant to anyone
185 in that interface. Uh, "The Hero goes to the Land of Adventure, Or the Hero goes"... it
186 might have to have "Nowhere" as a place. So declare nowhere as a place. And it has to be
187 capital "N" when it's a thing. So when you put "The Hero goes to Nowhere", it's currently
188 not a capital "N".

Syntax:
Capitalisation

189 **Participant:** Ah right, yes.

190 **Myself:** Is that gonna work? No. Oh, yes! There we go. Cool.

191 **Participant:** So, "The Villain goes to Evil Lair, The Hero either goes Nowhere, Or goes
192 to Land of Adventure". Makes sense. Another thing that would be... again, it's strange
193 because the more you get into it, the more you think about it, the more you start turning it
194 into a regular programming language. One thing that would be useful would be able to say...
195 to have... what's the phrase I'm looking for? Sort of a... well, if functions basically, to be
196 able to say "If the Hero is in Nowhere, then he finds a Sword", or something.

Feature:
Conditionals

197 **Myself:** Yeah, yeah. That's something I definitely thought about and tried to do, but
198 couldn't implement in time for doing these studies. I was kind of also thinking of like the way
199 to describe that, would you put... would it make sense to have the "if" at the beginning,
200 or the end of a statement, or just like... so if you want to have your "if" dependent on lots
201 of events having occurred previously, it's kind of hard to kind of think of an elegant way of
202 doing that. You could just put like "if:", and then list the events that could happen as the
203 preconditions. But yeah... so how would you do it, you'd say "If the Hero goes to the Land
204 of Adventure, and...".

205 **Participant:** Something like "If the Hero is in the Land of Adventure, then the Hero finds
206 the Sword". I think it would work out OK because if you have, it might not be as complex as
207 it sounds, because if you have multiple ways the Hero can end up in the Land of Adventure,
208 then you can simplify all those things down to "If he ends up there, then he gets the Sword",
209 regardless of how he got there, if that makes sense.

210 **Myself:** Yeah, I think that makes a lot of sense. I was kind of keeping it event-based, so
211 that, you know, instead of "If the Hero is in the Land of Adventure", I had it so it was like
212 "When the Hero goes to the Land of Adventure". Then X, Y and Z happens. But I think
213 "If" makes more sense, because if you're familiar with programming, you know about "Ifs",
214 etc. And also it allows you to describe things other than events, like when you want... when
215 a Hero picks up an object, you want to be able to say "If the Hero has an object", but again
216 that would involve more work, and having these special... these fluents which describe what
217 state holds at what time. So for the time being, just for the point of this study, I'm keeping
218 it event-based, just really, really simple.

219 **Participant:** Makes sense.

220 **Myself:** OK, cool. So... right. I think you've kind of played with that task enough,
221 so if you move on to the next task, which was task 3. It says "adding objects", but I think
222 you've kind of done that, you had objects. So move onto task 4, which is adding branches.
223 Oh, again, you've done that. OK. That's cool. You're kind of moving through all the tasks

224 already. That's cool. So, next... OK, this one's easy. Go to the "arrange" tab in the interface.
225 And for this you can combine different tropes. So add a trope to your "Super Duper Trope".
226 Maybe one of the simple ones, like "Example 1", or "Example 3". Just wait... so sometimes
227 this takes a long time because it's sadly quite CPU intensive, and this is running on a server
228 somewhere.

229 **Participant:** Oh, wow.

230 **Myself:** There you go. So this has worked out all the different possibilities of combining
231 these two tropes.

232 **Participant:** That's really cool. "Go to Evil Lair"... So "go_home_hero", or "go_-
233 villain_evilLair", the Villain can then go to the Evil Lair, the Villain can then (garbled) the
234 Sword. So at each level it's going through one path of each trope.

235 **Myself:** Yes, that's right. So one thing I should mention is that I've limited this so that it
236 visualises a maximum of five events in a chain, because otherwise it could just take too long.
237 And also like a maximum of a hundred different possibilities, because again it could get stuck.
238 So sometimes it doesn't generate all the possible nodes. But in this case, the tropes are simple
239 enough that I think that's all the possibilities.

240 **Participant:** Hmm. What about if you had an object or a place with the same name, in
241 two different tropes? How does it handle that?

242 **Myself:** That's a really good question, and it treats them as the same thing in this case.
243 So it is kind of like they're really merged. Again, in an earlier version I had it so that you
244 could specify different names for the different roles, so that you could have a Hero called "Luke
245 Skywalker" in one trope, and a Hero called, I don't know, "Darth Vader", if you wanted that,
246 in another trope. And they would actually have different permissions and they would be
247 visualised as different things. But as you can imagine, that would increase the complexity of
248 this visualisation again, and we would have more and more stuff to be generated into this tree.
249 Again, I'm trying to keep it simple for this.

250 **Participant:** So, is it impossible then, for say, "The Hero finds the Sword", and within
251 another trope to say "The Villain finds the Sword"? Or would it allow that?

252 **Myself:** Yeah, it would allow that. So they could both find the sword according to the
253 two tropes.

254 **Participant:** OK, that makes sense.

255 **Myself:** Ah right, OK. Cool. So could you go to task 6, then. I'm going to ask you to
256 embed a trope inside the one you created already, perhaps. Or maybe, no, take the one you
257 created and embed that in a new trope.

258 **Participant:** Right, OK. So "Super Duper Trope" is this one.

259 **Myself:** Or maybe embed it in an existing trope if you can't be bothered to type it. Either
260 way. So type the name of a new trope here. This has still got the name of an old one.

261 **Participant:** "the Hero is a role"...

262 **Myself:** So yeah, be careful with the capital "T"s on the "the"s at the beginning.

263 **Participant:** Ah, right.

264 **Myself:** But that is something I need to bear in mind to fix, I think.

Feature:
Different
characters
in different
tropes

265 **Participant:** Yeah, that would definitely make it more robust, if can stop thinking about
266 that.

267 **Myself:** Yeah, and it's easy enough to fix, so that would be a good point.

268 **Participant:** Hmm. "The Hero finds the Smartphone, The Warm Bed is a place"...

269 **Myself:** You have to put that before the Hero finds the Smartphone.

270 **Participant:** Ah, right, yes, sorry. 'Hero goes to the Warm Bed, Or the "Super Duper"
271 trope happens'.

272 **Myself:** Hopefully that will work. So you have to save it, then go to "edit", and visualise.
273 Oh, it's compiled, so... Ah no. So I have a feeling it doesn't like putting that in a branch.
274 So just change it to 'Then the "Super Duper" trope happens'. So this is something which
275 is quite frustrating because the mechanism behind this is something which is super awkward
276 and difficult, but it always works when it's at the end of a trope. Oh, OK, it's not liking that
277 either. Interesting. Oh.

278 **Participant:** I think it was adding "the" fixed it.

279 **Myself:** OK, so it's the "the". So actually change that to "Or". Change that back to
280 "Or" and see if that works now.

281 **Participant:** Let's see. I can't remember if I had the "the" there before or not.

282 **Myself:** Oh, OK.

283 **Participant:** No, it doesn't like it.

284 **Myself:** OK, right, that's another thing to fix. Cool.

285 **Participant:** I wonder what's... this was working a second ago.

286 **Myself:** I think if you click "refresh" too much, it queues up all the things. Ah yeah, OK,
287 cool.

288 **Participant:** So I was just clicking too much, and it got confused.

289 **Myself:** Yeah, I think that was it. OK, cool.

290 **Participant:** "The Hero finds Smartphone", "The Hero goes to the Warm Bed", "The
291 Villain goes to the Evil Lair". Makes sense.

292 **Myself:** Cool. OK, sweet. Alright, so the final task is to just mess around with the
293 interface, basically, and make any kind of story you want. So you could do it by arranging the
294 existing tropes you have with the "arrange" tab, or by altering ones you've already made, or
295 creating new ones. Just spend a few minutes doing that. And while you're doing that I'll ask
296 you some open-ended questions about, you know, what your thoughts are when you're using
297 it.

298 **Participant:** Uh-huh. Uh... open-ended thoughts?

299 **Myself:** So, er, if you were making a story-based game, and you wanted to create a kind
300 of branching narrative, and assuming that the mechanism you're doing it is that you have
301 these intelligent agents as the characters, do you think this would be a good way of writing
302 the constraints for these characters, so that you can visualise...

303 **Participant:** Yeah, I think it makes a lot of... I was imagining it earlier that each
304 of these could relate to a plan in some intelligent agent, and then you'd be able to just
305 somewhat dynamically... where all the agents are actually behave in somewhat sensible

Syntax: Entity declarations

306 manners following a traditional story structure. I think that makes a lot of sense, Matt.

Usability:
Ease of use

307 **Myself:** That's exactly, thanks, that's exactly the intention of this language. It should be
308 OK with two words there. I'm pretty confident that that's fine.

309 **Participant:** Um... "The Farm Boy finds the...". So, one thing it can't express is
310 where it started at. You can say: "The Robot goes to the Starship", but then you can't say
311 something like: "goes to the Planet FROM the Starship". So you have to, say, make an event
312 arriving somewhere rather than declaring that they started somewhere.

Feature:
State management

313 **Myself:** Yeah, that's a good point. So I had these special reserved words for "is" and
314 "has", so you could say "The Robot is at the Planet they came from", "The Robot is"...
315 but you actually can't. I think it will break it, because I reserved those words and didn't do
316 anything with them because it turned out to be complicated. So yeah, would that solve your
317 problem though, if you wanted to say where it started from? You'd say "The Robot is at this
318 planet, Planet X, Then the Robot goes to Planet Y".

319 **Participant:** Yeah, then I could do things like: "The Farm Boy is at the Planet", or I
320 suppose I'd have to do it after declaring the planet, but yeah.

321 **Myself:** Yeah, yeah. That's right. That's how it would work.

322 **Participant:** You could say: "The Robot goes to the Starship, Then the Farm Boy finds
323 the Robot". Oh wait, is the Robot a role, or...? Oh, it's an object, OK. Oh, so I'm saying
324 an object goes... let's see if that works, that'll be interesting.

Feature:
Subject must be an agent

325 **Myself:** Oh, I have a feeling that's... OK, yeah.

326 **Participant:** I'm guessing it probably won't, because we had places moving around, and
327 that didn't work.

328 **Myself:** It doesn't like it because of the blank line, I think.

329 **Participant:** Oh right, sorry, yeah.

330 **Myself:** So try that. No, it doesn't like that either.

331 **Participant:** I don't see any other errors either, so it's got to be that, hasn't it?

332 **Myself:** "Starship is a place, Robot is an object"... yeah, so try "The Robot is a role",
333 then. Ah, there you go. So it doesn't like objects going to places. Interesting.

334 **Participant:** But it is happy with people, roles finding other roles, which makes sense
335 doesn't it? Seems sensible.

336 **Myself:** Yeah, yeah. I think that makes sense.

337 **Participant:** Um, so what else is there? He meets the Wise Man. Er... so "Farm
338 Boy..."

339 **Myself:** There's going to have to be a capital "H". Yeah, exactly.

Syntax:
Capitalisation

340 **Participant:** I'm trying to think where this actually branches off. Um.

341 **Myself:** So yeah, I've limited this to maximum five events long. At the moment it is five
342 events long. But you can still put branches in there, so that will... that's fine.

Limitation:
Five event limit

343 **Participant:** Right, OK. Or make subtropes as well?

344 **Myself:** Ah yeah, as long as it's the last event, I think. You could put "then"... so the
345 last event could be "Then the whatever trope happens".

346 **Participant:** Right, OK, so I could do something like, save that, then (indistinct) make

347 this much more modular and do, like, a "Receiving the Laser Sword". "The Farm Boy is a
348 role". Oh yeah, can you do... "The Wise Man is a role"... you can't do giving, right? I
349 can't say something like...

Feature:
Event with
three entities

350 **Myself:** You can, but again it won't have any semantic... oh yeah, it would. Kind of.
351 It would give an agent permission to give something to another agent. So it would work, it
352 would compile and work. Yeah, so actually that would be fine.

Feature:
Event seman-
tics

353 **Participant:** Let's try that. "Receiving the Laser Sword". "Compile Error".

354 **Myself:** So I think you have to put "an object".

355 **Participant:** Oh, OK.

356 **Myself:** "The Laser Sword is an object". And then "The Wise Man" has to be capital
357 "W". Oh no, so "Laser Sword" is fine, but when you say "The Laser Sword is an object", it
358 has to be "an object".

359 **Participant:** Oh right, OK.

360 **Myself:** Yeah, "The Laser Sword". Yeah, try that. There you go.

361 **Participant:** Makes sense. "Or the Farm Boy goes to Home".

362 **Myself:** So... oh, that's interesting. You should have indented the "Or" with two spaces,
363 but it works anyway. Interesting. Oh well. So yeah, don't forget to put "The Hut is a place".

364 **Participant:** Oh right, yes. (runs through the events) "Then the 'Receiving the Laser-
365 Sword trope happens".

366 **Myself:** Ah, I've got a feeling though. See what... OK. Why is that... so it was OK
367 before, was it? Or was it OK... delete that last one, see if...

368 **Participant:** Yeah, I guess it is that last one.

369 **Myself:** OK, so "Then the 'Receiving the LaserSword trope happens". OK, save that.
370 Try again. It looks right.

371 **Participant:** Is it doing something?

372 **Myself:** Maybe. It's hard to tell. Yeah, try again. Hmm. Refresh the page and try again.
373 Sometimes it gets stuck.

374 **Participant:** Is this stuff stored on my computer, or...?

375 **Myself:** It's stored on a server, so it's all good. It seems to be getting stuck for some
376 reason. That's interesting.

377 **Participant:** It was a fairly simple thing, wasn't it? It was one "or" node.

378 **Myself:** Yeah, and it was fine. It compiled and visualised that fine. I wonder why it's
379 getting stuck.

380 **Participant:** OK, so it did compile on the server, but it didn't come back to here.

381 **Myself:** Yeah. I can just load it up on my server and see what's happening.

382 **Participant:** I'm getting an error 500, for some reason. From the "new" endpoint, "sto-
383 ry/new".

384 **Myself:** Yeah. Hmm.

385 **Participant:** Yeah, it has a 200 first, and then a 500 afterwards.

386 **Myself:** That's interesting. Try having it just two events long. So get rid of everything
387 between "The Robot goes to the Planet" and the last event. Huh, weird.

388 **Participant:** Maybe it's because the trope it's going into has a nested clause? Something
389 about that confusing it?

390 **Myself:** No, it should be fine. Let me see. I'm just going to load up my logs. So...
391 "Receiving the LaserSword"... it's compiled that OK. So... it hasn't solved it.

392 **Participant:** Is it taking too long to solve, is that it?

393 **Myself:** That's probably it. It's probably just taking a really long time to solve for some
394 reason, but I can't imagine why. Huh. Yeah, I'm not sure. "Receiving the LaserSword"...
395 that's definitely the name of that trope. "... goes to the Planet, the Planet is a place, the Hut
396 is a place, Farm Boy goes to Home". So it's fine with that trope, it just doesn't like that...
397 Oh. OK no, it's compiled, it's kind of compiled it, but not... there's something it's not quite
398 doing right.

399 **Participant:** No.

400 **Myself:** So it's supposed to be creating this thing called a "Bridge Institution" which
401 describes the link between the first institution, the first trope in this case, and the second one,
402 which gives the first event of the second trope permission to happen. But it's not compiling
403 that at all. OK, well, that's something I'll have to look into. Alright, so if you go to "arrange"
404 and just play around with that for a bit.

405 **Participant:** It's not going to like "A New Hope" is it? Let's put another one in.

406 **Myself:** So yeah, that is the one it's doing. So this is the kind of thing that will take a
407 while sometimes. I had this running on my own server on previous studies, and it was really
408 slow. So I've moved it to "Mist", which is one of the nicer servers on Bath, which is a lot
409 quicker. So cool. Alright. I'm trying to think of any other questions I can ask you. I think
410 that about covers it, though. Have you got any other thoughts?

411 **Participant:** Not really. I think I can definitely see where you're going with it, I think
412 it's a really cool idea actually.

413 **Myself:** Oh, cheers.

414 **Participant:** So I'm imagining you can do something like export, you design the outline
415 of a story, export it to some format like JSON or XML, put it into some part of your system,
416 and then your agents begin to follow this series of actions where each node is a plan that
417 they execute, and once a plan is completed then all the other agents know that this event has
418 happened, and therefore the possible actions they have are this.

419 **Myself:** Yeah, exactly, exactly. I think that for for this to be a complete system, we need
420 to have another language for authoring the agents and their plans, and their dialogue, which
421 kind of hooks into this, but again that's outside the scope of this PhD. But it's something
422 I'd like to do, I think. Because a lot of this multi-agent system stuff is really inaccessible to
423 people that just want to make games, especially story-based games, who don't do a lot of
424 programming. It is quite esoteric, and just niche, and so it'd be cool to have a simple way to
425 use it, to do simple agent stuff like this.

426 **Participant:** And I guess you can then easily... if you decide to change the story in some
427 way, you can just recompile this, change a few things, and it will put it all where it needs to
428 go.

429 **Myself:** Yeah, exactly.

430 **Participant:** I mean, I think we've already talked about, like, my thoughts about the
431 language. I think the only thing that might improve it is if you can make it more robust,
432 like having whitespaces would be really useful, actually having comments so you can highlight
433 "here are my declarations, here are my things that happen", and such.

434 **Myself:** Yeah, yeah, that's a good idea. Certainly. I definitely agree.

435 **Participant:** And then multiple levels of nesting, that kind of stuff. I think with those
436 kinds of things it would be really, really powerful.

437 **Myself:** Yeah. Definitely. I definitely agree. Cool. Well, thanks very much.

438 **Participant:** No problem.

Limitation:
Branch
length limit

G.5 Participant E

Dialogue starts at 12:08.

1 **Myself:** That's the basics of the language. So what I want you to do now is, I've created
2 seven tasks, and they start of very simple and get a bit more complicated towards the end.
3 The first one is simply to edit a trope that's already there. So you can select one of those
4 tropes, maybe one of the example tropes, and just change it somehow.

5 **Participant:** OK. Adding a declaration, or something like that, you mean?

6 **Myself:** Yeah, you can do something like that. Change it however you want.

7 **Participant:** "The Shield is an object"... "Or the Hero finds the Sword, Or the Hero
8 dies"... there.

9 **Myself:** I think you have to click "save" first. Yeah, you have to save it and then "refresh".

10 **Participant:** OK. There we are.

11 **Myself:** Cool. Alright, so that's task one done. I'm suppose to ask you lots of open-ended
12 questions as well. So: do you think that this language would be useful for non-programmers
13 to learn? Would it be easy to pick up, so far? I've kind of told you the basics, and you're
14 starting to learn it. Is it easy to pick up, do you think?

15 **Participant:** Um. I think it's a very basic... going from my knowledge of programming
16 languages, it seems that the only thing you need to know is "if-then" statements, because
17 it's a pretty simple... as it stands right now it's pretty simple. The declarations themselves,
18 I think they might be hard for a non-programmer to understand, like why do you have to
19 declare these things beforehand?

20 **Myself:** Yeah, so in a previous version of the language I was trying to work out a way to
21 do it without declarations. So it'd work out what a character is, what an object is, etc based
22 on the verbs used. But that was way too hard. So I just thought that I've just got to put
23 those declarations in, sadly.

24 **Participant:** Yeah, I think that's the one thing that anyone might have difficulty working
25 out. Other than that... OK, so you know like: "The Hero does this, or the Hero does that,
26 and then the Hero does"... you have like two sets of "if" statements. I don't know what else
27 to call them. I mean OK, at first you're like: "How does that work?", and then I think the

Usability:
Ease of use

28 visualisation of those branching things... I think that actually quite helps to understand. I
29 think the visualisation helps you understand the code.

Usability:
Visualisation

30 **Myself:** OK, cool. That's good to know, actually. Thank you. OK, so on to the next
31 task. So this is like creating a trope of your own from scratch. And to kind of help you, I've
32 limited the scope of what you can do. So if you go to the other document you have open in a
33 tab. I think it was the information... was it that one? Yeah. And go down to "task 2: create
34 a trope".

35 **Participant:** Sure. "Given the following character roles: The Hero, The Mentor, and the
36 following places..." (continues to read task).

37 **Myself:** So yeah, don't use the branching just for this. So create a new trope from scratch
38 which is just a sequence of events, one after another.

39 **Participant:** OK, so I need to include the statements. Sure.

40 **Myself:** OK, to do that, click on "new", which is just under the text box. And then first
41 type in the trope name in that text box there.

42 **Participant:** Sure. "Task 2", I guess. Shall I just save it, and it will visualise?

43 **Myself:** Don't save it, because there's a bug which... I think I can't fix any bugs once I've
44 started doing the studies because it's different for every person. But there is a bug where, if it
45 doesn't compile then it appears as a blank trope with a blank name, and it's quite annoying.
46 But I think it's OK, though. So write your trope first, and then save it.

47 **Participant:** "The Hero is a role, The Villain is a role, The Mentor is a role". And then
48 "Home, Evil Lair, Land of Adventure". "Home is a..." place? Is that...?

49 **Myself:** Yeah.

50 **Participant:** "Evil Lair is a place, Land of Adventure is a place". OK.

51 **Myself:** Cool. And then just sequence events one after another. So...

52 **Participant:** Do I just copy and paste that, or...?

53 **Myself:** No, just create your own sequence of events.

54 **Participant:** OK.

55 **Myself:** So again, I've kind of limited it so you can only have a maximum of five events,
56 one after another. I'll explain why after you've done this task.

57 **Participant:** OK. So the only verbs I have is "goes"? At the moment?

58 **Myself:** You can use any verb. Literally any verb you want. Even verbs with multiple
59 words in them.

60 **Participant:** OK, so "Hero goes Home", for example?

61 **Myself:** Yeah, "Home" has to be capital "H". Yeah, that's right.

62 **Participant:** Um, "The Hero meets the Mentor".

63 **Myself:** Yeah.

64 **Participant:** You have... OK hold on, see what happens. "The Hero goes"... can I say
65 "to the Land of Adventure"? Is that possible?

66 **Myself:** Yeah, that's fine.

67 **Participant:** I tell you what, "Evil Lair"... "kills the Villain, Hero goes to the Land of
68 Adventure". There we go.

Limitation:
Five event
limit

Syntax:
Capitalisa-
tion

69 **Myself:** OK... 1, 2, 3, 4. That should be fine. So save that, hopefully that will work.
70 Now click on "edit"... you can't visualise it until you go to "edit", sadly, so click on "edit"
71 and then select your trope. Oh yeah, cool, that seems to work. And then click "refresh", and
72 that should. Oh no, it's a compile error. I wonder why that is... Right. Let's see... "The
73 Hero is a role... is a role, is a role, is a place, is a place... goes home, meets the mentor"
74 ...try changing it from... so every event that's after "The Hero goes Home", put "Then the
75 Hero meets the Mentor, Then the Hero goes to the Land of Adventure".

76 **Participant:** OK. Wait do I edit...?

77 **Myself:** Yeah, yeah.

78 **Participant:** "Then the Hero goes, Then the"...

79 **Myself:** Yeah, exactly.

80 **Participant:** OK.

81 **Myself:** That might... yeah, you have to click "save" first.

82 **Participant:** Yeah, cool.

83 **Myself:** Oh, it's not appearing at all. Hang on, let me try it on my computer.

84 **Participant:** Oh, you can see these things as well?

85 **Myself:** Yeah. So I have it open in my computer too. And it's communicating with the
86 same server, and it's all on a database somewhere, so... Let's see... task 2... yeah. So
87 unfortunately it's hard for me to get any more useful error messages than what's there.

88 **Participant:** Uh-huh. What if I just like, go off and then go back on it? Did I break your
89 machine?

90 **Myself:** Hang on a sec... so... "compile error"... this one should be OK, I wonder why
91 it isn't working. "Home is a place, Mentor is a role". I think the only thing for me to do is
92 for me to delete these events and see what happens if I... OK, so the first one's fine. Second
93 event is fine. Maybe it's "The Evil Lair" it doesn't like. Yeah, it doesn't like "The Evil Lair"
94 for some reason, I wonder why that is. Just change "The Evil Lair" to something else.

95 **Participant:** OK. "Lair". Shall I just make it one word. Or something?

96 **Myself:** Yeah, yeah. Try that.

97 **Participant:** Uh. Did I do that right?

98 **Myself:** It should... nothing's actually appearing at all.

99 **Participant:** Wait, is it... I meant to say "The Hero"... "Then the Hero"...

100 **Myself:** Yes, try that. Save it, and then refresh. Actually, there's nothing appearing at
101 all, so try refreshing your browser.

102 **Participant:** OK. It's just refreshing. There we are. OK. Task 2.

103 **Myself:** And then yeah, just click "refresh".

104 **Participant:** "Compile error".

105 **Myself:** Ah, so delete everything after "Lair", but copy it so that you can paste it in after.

106 **Participant:** OK, sure.

107 **Myself:** No, like after "Lair" in the sequence of events. So that seems to be where it was
108 getting stuck before. Did you click on "save"?

109 **Participant:** Yeah.

110 **Myself:** Refresh... so "Lair is a place"... hang on, let me... I'll try on my computer to
111 see where exactly it's going wrong. I wonder if it's expecting "THE Lair is a place".

Syntax: Use
of "the"

112 **Participant:** So should we just go to "Lair"?

113 **Myself:** Actually, if you go to line 5 and put: "The Lair is a place", then "The Land of
114 Adventure is a place". It shouldn't make a difference, though, because it's supposed to ignore
115 that. Yeah, try that.

116 **Participant:** (garbled) back in the lines, or should I take them out?

117 **Myself:** Yeah, take them out, so that we make sure this is working.

118 **Participant:** Oh there we are. It works.

119 **Myself:** OK, so put those two back in. So it's the "the" problem. So if you declare it as
120 a "the", it has to be a "the" later as well. Even though it's supposed to ignore the "the"s.
121 Alright. So put those two events back in, that you had.

Syntax: Use
of "the"

122 **Participant:** Sure. "Save" and "Refresh".

123 **Myself:** It takes a little bit of time, usually. But it's... there you go, yeah. The trouble
124 is that I'm running this on my own personal server which is really cheap, and I think it's an
125 ARM-based server or something. It's really slow. So that looks good. OK. So we can move
126 onto task 3, which is: just take that trope and add an object, which could be a weapon or an
127 evil plan.

128 **Participant:** OK. Evil Plan?!

129 **Myself:** Yeah.

130 **Participant:** Evil Plan... is that in the information?

131 **Myself:** It is, yeah. So go to "task 3" there.

132 **Participant:** "Then the Hero makes an Evil Plan"... OK. Sure, sure. OK, so the Evil
133 Plan's an object, I see. OK. It's "the Weapon is an object"?

134 **Myself:** That's right, yes. So they're both objects. Ah right, so I didn't explain why
135 it's five events long. So this whole thing is run through an answer set solver, which is kind
136 of like... it's a bit like Prolog in that it kind of searches all the different possibilities, so it's
137 doing this really expensive search through all these different possibilities of what can happen.
138 And if the number of events is larger than five for complicated examples, it just takes really
139 long. So I've limited it to a maximum of five events that can happen.

Limitation:
Five event
limit

140 **Participant:** The tractability is...

141 **Myself:** Yes, exactly.

142 **Participant:** OK, "The Hero is an object"... so I have to remove two of these essentially
143 to add the evil plans?

144 **Myself:** Yeah, that's right. Because you can't do branching yet.

145 **Participant:** OK. "The Hero goes to the Lair". "The Villain"... is that right? He makes
146 an evil plan?

147 **Myself:** Yeah, that should work. So save that and see.

148 **Participant:** Oh wait, hold on. I didn't do "then".

149 **Myself:** It might be OK without "then". But put it just in case.

150 **Participant:** Cool. OK, "save trope".

151 **Myself:** Oh no. OK, so sadly. . .

152 **Participant:** Wait, wait.

153 **Myself:** Oh. So, hang on. . . is that OK? Yeah, that's what you. . . oh no, that's what it
154 previously was.

155 **Participant:** I think "an".

156 **Myself:** Ah, perhaps that's it, yeah. "Finds the Sword". . . "The Sword". . . wait, oh no
157 you put "The Weapon" rather than "The Sword", that's why.

158 **Participant:** Ah, OK, yeah, sure, sure. Probably because I was thinking about the sword
159 from one of the previous ones.

160 **Myself:** Ah, yeah.

161 **Participant:** There we are.

162 **Myself:** OK, there we go. Cool. Right, so, task 4, if you go to the other document,
163 is just to add branches to the story you have there. So replace some of those sequences, or
164 actually I think you can just augment the sequence or any event there with branches, branching
165 possibilities.

166 **Participant:** Sure, OK. And that's two spaces to get the branches?

167 **Myself:** That's right, two spaces.

168 **Participant:** OK. Ah, OK.

169 **Myself:** I think maybe every time you add a line, just save it and refresh in case it doesn't
170 compile, then you'll know which line it is that goes wrong.

171 **Participant:** Ah, that's a good idea. Debugging. That's what I need to learn to do.

172 **Myself:** Yeah, it usually takes a little while. OK, there should be. . . ah, there you go.
173 Yep, cool.

174 **Participant:** OK. "The Hero goes to the Lair, either the Villain finds a weapon, Or the
175 Villain makes an Evil Plan, or the Hero goes to the Land of Adventure". Oh, wait a minute.
176 OK, OK. It looks as if, on the second. . . in either case, the Hero kills the Villain, but in this
177 second one where the Villain finds a weapon, there's no option.

178 **Myself:** Hmm. I wonder why that is. That's well spotted, actually. 1, 2, 3, 4, 5. . . I
179 think. . . yeah, that's a good point, why is that? Try refreshing it again. Hmm. I'm going to
180 have to work out why that is.

181 **Participant:** Sure. Hey, I found a bug for you!

182 **Myself:** Yeah, thank you. Yeah, but it's still the same. . . or in this case, neither of those
183 have been generated. Huh? "... goes to the Land of Adventure, kills the Villain" . . . yeah.
184 You'd think it would be the same every time you refresh it, but it's not. OK, well that's
185 another bug to look at. OK, so now you've added branches, go to task 5, which is combining
186 tropes and visualising them. So to do that, go to the "arrange" tab in the Story Builder. So
187 that's next to. . . yeah, there you go. So you have your "Task 2" there. . . maybe select a
188 simpler one, like one of the examples. And then click "+", and then you have another trope.

189 **Participant:** Example 2, I guess.

190 **Myself:** Yeah, you could try that. And then. . . just wait a while because this takes a lot
191 of time. Sometimes. There we go.

192 **Participant:** Oh my God!

193 **Myself:** So, because we have these two tropes, it's like it takes the first event from one
194 trope, and then the next one from the other trope, or the other way around, then does that
195 for every single event. So when you have these two tropes together, you can visualise all the
196 possible paths when both those things are happening in the story. So yeah, you can see why
197 I wanted you to do it with just the simple example first, but you could try changing one of
198 those tropes to your complicated one, and it will just get more and more complicated. And
199 you might have to wait a little while, with this puny server that I'm using.

200 **Participant:** It takes the first of... oh, hang on, wait a minute.

201 **Myself:** It's like all the different possible combinations of events, based on the tropes that
202 you've given it.

203 **Participant:** OK... so it kind of like splices them together or something? It doesn't line
204 them up one after another?

205 **Myself:** It splices them together, yeah.

206 **Participant:** OK, OK. So first event on the first trope, first event on the second trope,
207 second event on the first trope, second event on the second trope, sort of line them up like
208 that?

209 **Myself:** Yeah, something like that. Oh it is taking a long time. I guess I should be asking
210 you some more open-ended questions while we're waiting for this. So, I think if you try to
211 author a story with lots of paths through it, how... do you think this could be a useful tool
212 for kind of visualising the paths through the story? I know it's kind of a preliminary example
213 of it, and this isn't a fully-featured thing, but do you think that something like this would be
214 useful?

215 **Participant:** I mean, I'm not familiar... OK, so I didn't create the trope "Example 1",
216 I mean I'm not actually seeing my own trope yet.

217 **Myself:** No, it's taking a long time.

218 **Participant:** But I imagine if I did know a story, the individual tropes inside out, just
219 to be able to like... oh gosh, there it is... just to be able to line them up and see what the
220 various different paths are, and... I guess for example, if these are two different characters,
221 maybe just looking at them occurring simultaneously. I don't know, it's quite big. Oh, there
222 we are. I'm trying to work out what this is showing me.

223 **Myself:** So if you look at the connections between the nodes, it's labelled with the name
224 of the trope there, so you can say if the red arrows are following your trope, task 2, it's saying
225 "this can happen, but then after that we can follow the other trope, the example 1 trope, and
226 the Hero can go home".

227 **Participant:** Oh, OK. So it's like merging two together.

228 **Myself:** Yeah.

229 **Participant:** I mean, I can't think... OK, personally I can't think of an application,
230 I don't know, maybe it will come to light. I can't, um, I don't know. I'm having trouble
231 understanding it from the outset. I get what's going on here, you're taking steps from each
232 of the tropes and slicing them together and producing paths and showing what happens. Oh

Usability:
Visualisation

233 wait no, I see, I see. It's like introducing the tropes at each individual... what happens when
234 you introduce the second trope at each... stage of the first trope. And comparing them that
235 way, I guess? Yeah. Oh so like, you can take a diversion. So like there can be one trope, and
236 that would go one way, and then a second trope would go another way. But if you're half way
237 through the first trope, and then decide to go on the second trope, and then return to the first
238 trope. I think that's what it's showing me, anyway.

239 **Myself:** Yes, something like that. So do you remember from the intelligent agents lectures,
240 I think Charlie gave a lecture on institutions, which is like permissions and obligations, and
241 social norms which act on intelligent agents. So the point of the system is that if you have
242 intelligent agents which act out the characters in a story, these are kind of like the constraints
243 which act on these agents. So this language actually compiles to like, a constraint language,
244 which describes the permissions and obligations that act on these agents. I don't know if
245 Charlie mentioned it in the lecture, it's called "InstAL". And then that compiles to this
246 Answer Set programming language, which is a kind-of-like Prolog-like language, which we can
247 use to generate all the different possibilities of these permissions and obligations. And that's
248 kind of what we're visualising here. So if you're... this describes what agents are permitted,
249 what they're allowed to do at any point in the story. And then given a certain sequence of
250 events, they're allowed to do something else.

251 **Participant:** OK, OK, OK. Alright. This is like a sort of... OK, how I'm thinking about
252 this being used is someone trying to write a story, and they're trying to look at how... I guess
253 I'm looking at it from more of a literary point of view. If this is applied in programming when
254 someone is creating a game or something, and they're trying to create a narrative, give the AI
255 some instructions. Is that the kind of thing you mean?

256 **Myself:** Yes, something like that. You're kind of constraining what the AI can do. Like
257 if you have these intelligent agents, and you've given them plans and rules to follow, they can
258 just go off and do their own thing. But with this you're kind of constraining their actions
259 so that it fits with some shape of a story. And then you're composing the story out of these
260 tropes.

261 **Participant:** They fit with the behaviour of other agents as well? So it's like, kind of like
262 a social aspect to it.

263 **Myself:** Yes, that's right. Yes.

264 **Participant:** OK, OK.

265 **Myself:** But also the other thing is actually kind of composing the story out of these
266 components, these tropes, which... let's see... so the next task is kind of asking you to
267 create a new trope with a subtrope inside of it. So if you go back to the other document,
268 and it says "Task 6: Tropes within tropes". I'd like you to create a new trope. Well actually,
269 you'd have to create... oh no, yeah. Create a new trope which uses one of the tropes that
270 have already been created previously.

271 **Participant:** OK, alright. Um, so create a new trope.

272 **Myself:** And unfortunately there's another limitation here, which is that the trope has to
273 happen at the end of this trope. Again, I found a bug where it can't happen during the trope,

274 it has to happen at the end.

275 **Participant:** Oh, the subtrope has to be at the end?

276 **Myself:** Yeah.

277 **Participant:** OK, OK. I'll just call it, I don't know... "Use Sub Trope". So OK, question
278 about this then. So the declarations I make in this trope which is using a subtrope. Do I need
279 to make... so for example, suppose the characters in the trope I'm creating now, suppose I
280 have a Hero, a Villain and a Mentor in this trope.

281 **Myself:** So, automatically, by default they kind of refer to the same characters as in the
282 other trope. Again, in a previous version I had it so that you could specify that they could
283 be different characters, but that turned out to be too complicated for this. So yeah, any
284 declarations you create here will kind of merge with the other ones. They refer to the same
285 character.

286 **Participant:** OK, so I can say: "The Hero is a role", and it will refer to the same Hero
287 that's in the subtrope?

288 **Myself:** Yes, that's right.

289 **Participant:** "Or the Villain"... no, I'll just keep those two there. And then...

290 **Myself:** "Training is a place"... OK.

291 **Participant:** Not the training... "The Training place is a place".

292 **Myself:** Yep, sounds good.

293 **Participant:** "The Training Place is a place". Um... and "The Fighting Stick is an
294 object".

295 **Myself:** Cool.

296 **Participant:** "The Hero goes to the Training Place, Then the Hero meets the Mentor,
297 The Mentor trains". Wait, can I do "picks up", can it use that?

298 **Myself:** Yes, you can use that, yeah. I think so. Yeah, yeah. Ah yeah, that should work.

299 **Participant:** Ah, and then to include a subtrope, I... what is it?

300 **Myself:** So "Then the" ...and then the name of the subtrope ... "happens".

301 **Participant:** "Then the"... what was the trope called? Task 2. "'Task 2" trope happens'.

302 **Myself:** Yes, that's right, yes. So save that trope. Hopefully that will compile. So click
303 on edit, and if it appears... use subtrope. Yeah. So select that and click refresh. Hopefully.
304 I have a feeling this one's going to take a long time as well, so while that's churning away, I'll
305 ask you... So at the moment, because of this bug it can only happen at the end of a trope.
306 But do you think this is a good way to kind of include these abstractions? Do you think that
307 this method of composing tropes together into a story is a good way to create stories?

308 **Participant:** It's really interesting, because it's like you're subdividing the story into
309 these different modules, and you're developing them independently. I could see there being a
310 placeholder, so... I don't know. I imagine it would be quite cool. Yeah, OK. Just like kind
311 of modularising the story itself. Like into different acts, for example.

312 **Myself:** Yes, exactly.

313 **Participant:** I don't know. It's pretty cool. I think it's pretty helpful.

314 **Myself:** OK, cool. Have you heard of a trope before? Do you know what tropes are?

Limitation:
Subtrope at
end

Usability:
Tropes

315 **Participant:** Er, tropes are common, erm, I don't know, like symbols or... not symbols...
316 patterns of a story and they're used commonly across different stories, they're quite... I can't
317 quite think how to explain it. I know what they are in my head.

318 **Myself:** Yeah, that's pretty much what it is. Kind of like commonly occurring themes or
319 patterns in stories. So what's happened here is that because of that five event limit, it's done
320 the four events of the first trope, then it's got as far as the first event of the second trope. So
321 maybe if you edit the... maybe edit this one so it's a bit shorter.

322 **Participant:** OK. "The Mentor trains the Hero". Hold on, let's do that.

323 **Myself:** Yeah. Again, it's taking an awful lot of time. So, you're familiar with tropes...
324 oh, did that refresh? No, you're just moving the browser. So yeah, as you said, tropes are
325 kind of like acts, can be like acts in a play and it seems to be kind of a useful way to divide a
326 story up.

327 **Participant:** Sure, yeah. Because I thought tropes were just like, erm, it could be across
328 the entire story, right, it could be a thread throughout the story?

329 **Myself:** That's right. Even a three act structure of a play, where you always have three
330 acts, that's a trope in itself. It's kind of a pattern which you see in lots of stories. Did that
331 refresh? Oh, no.

332 **Participant:** It's like a cultural... ooh. That's weird.

333 **Myself:** OK, cool. So that's it. The final task is just to... free story creation. Create
334 your own story, again there's that five-event limitation, sadly, otherwise we'd be sitting here
335 all day waiting for it. So just, I don't know, if you click on "arrange", maybe just using the
336 existing tropes, and put some tropes together and see what happens.

337 **Participant:** And this still stays within the five event limit, right?

338 **Myself:** Yes it will, yeah. But I think because it's just merging them, and they're all five
339 events long, it will be OK in this case.

340 **Participant:** OK. Tell you what, I'll see what happens if I... so this is a way of building
341 up stories?

342 **Myself:** Yeah.

343 **Participant:** Oh, I see... branching. Now I see... you're exploring all the different
344 options you can have in as story. Right. I see. And this is where you build your... OK, OK.
345 Now I understand.

346 **Myself:** OK, cool.

347 **Participant:** That took a while. I've used a really big one. (reads the events)

348 **Myself:** It really is painfully slow. And it's frustrating because it's something which looks
349 really simple. I feel like there'd be a simple way to program this which would be way quicker,
350 but it seems like it's part of a PhD that you always have to use these complicated technologies
351 to do something which is research-y, which ends up being a bit janky, like this. But it's still
352 kind of interesting to explore, using these answer set technologies. But yeah, it's incredibly
353 slow because it's doing this really exhaustive search over all these different possibilities.

354 **Participant:** This isn't using something like Prolog? It's using something like Prolog?

355 **Myself:** So AnsProlog, which is like Answer Set programming. So with Answer Set

356 Programming, you have to ground all your variables first, so you know that the variable is
357 always assigned to some value, so that when it does this search, it's always searching for
358 things that have actual values, so obviously you can't guarantee it will terminate, but you're
359 pretty sure it will. And yeah, it's useful for generating lots of different possibilities based on
360 constraints that you've described.

361 **Participant:** OK, OK, OK.

362 **Myself:** I'm going to have to move this to a better server as well, because I've literally
363 used one of the cheapest servers I could, which is, I think it's got an ARM processor, like a
364 4-core ARM processor. It shouldn't be too bad, but still, I think it's quite slow.

365 **Participant:** Is this at the University of Bath, or is this somewhere else you're hosting
366 this?

367 **Myself:** No, I'm using a service called "Scaleway", which I'm using because it gives you
368 lots of space, but sadly not very beefy processors. But yeah, I could move it to a server at
369 Bath.

370 **Participant:** OK, because I feel like that might be faster or something.

371 **Myself:** I think you're right.

372 **Participant:** Did I do this right?

373 **Myself:** I think so.

374 **Participant:** I'll change it to Task 2... oh, I just cut it short. Why did I do that?

375 **Myself:** I think it's still going to try to compute that. So it still might be working on the
376 previous one. Unless... if you refresh the page, and then do the same thing again, I think
377 we'll find out.

378 **Participant:** Alright.

379 **Myself:** Yeah, it might still be working on the previous one.

380 **Participant:** Oh, here we are.

381 **Myself:** Hey, great. So I guess it got stuck somewhere. OK, cool.

382 **Participant:** Wait, are these just task 2? There's no example 1.

383 **Myself:** Oh, OK. I guess it did task 2, and now... you might have to wait for it to do
384 example 1. Unless... oh no, it's because task 2 is five events long. That's why.

385 **Participant:** There we are. OK, so I just need to get a different one?

386 **Myself:** Yeah. Maybe that was the case with the previous thing. We were waiting for it
387 for ages, but it was just because it was five events long.

388 **Participant:** ...I did that last time, that's boring. Um. "Item Search", what happens
389 there?

390 **Myself:** Hmm. I don't think it's going to change... ah yeah, there you go. Oh. I think
391 it's done what you selected previously. It's working through all the different things you've been
392 selecting, and doing them one after another, because it's kind of queued them up, I think.
393 Ah, there you go. That's... I think that's what you've selected. "Item Search", and then
394 "Example 1". Yeah, that looks good. But I think... I'd say you've done all the tasks now,
395 so that looks good. Is there any kind of general feedback you'd like to give? Obviously this is
396 a kind of... there's lots of bugs, it's really slow, it's kind of a very preliminary version. With

397 that in mind, do you think... how do you think this could be useful?

398 **Participant:** I feel like Aiden would probably give a better answer to this. I don't use
399 Blueprints as like, guides, for narratives anyway, I'm not sure what he uses, I don't know how
400 it works. Anyway...

401 **Myself:** He didn't actually mention Blueprint. He said he was working on games, but...

402 **Participant:** There was some.. I have no idea, I don't do any of this stuff. He just said
403 he was using a thing called... I don't know what it does. But this ability to... maybe if I
404 got more accustomed to the software, I'd understand a bit better. I don't know!

405 **Myself:** So OK, but the idea of tropes and composing stories out of tropes, do you think
406 that's a useful way to create these kind of stories? So you can visualise the branches?

407 **Participant:** Yeah, I think... I imagine the process before using this software would be
408 to write out the story itself and translate that into some computer code with tropes. I mean,
409 OK, so I've written one short story in my lifetime.

Usability:
Tropes

410 **Myself:** OK, yeah.

411 **Participant:** I've never written anything for guiding AI. But dividing up a story into
412 tropes. I guess there would have to be a translation process when the story's written to the
413 actual thing you're building on the computer. But then I guess that's quite cool because you
414 get to play with your own ideas, and it's like an extension of your own cognition. I don't know.
415 I don't think I have much else to say on that.

Usability:
Tropes

416 **Myself:** That's cool, I think I've got plenty of useful information from you. At the very
417 least, I definitely know about some more bugs that I need to squash.

G.6 Participant F

Dialogue starts at 14:52.

1 **Myself:** So now I'd like you to do some tasks, some simple tasks to do with this language.
2 So task 1 is to take an existing trope from that list of tropes, and just edit it in some way.
3 Change anything you want. So find a trope you like and change something. Something simple
4 at first.

5 **Participant:** (uses the tool for a while)

6 **Myself:** So what are you thinking?

7 **Participant:** Yeah, it's like the variables can have a type. Any type I define, right?

Feature:
Other types
of entities

8 **Myself:** No, they can only be role, or object, or place. But you can change the Hero to
9 something else, for example.

10 **Participant:** Role, place or object. OK. Do I have to use "the" and "an / a"?

Syntax: Use
of "the"

11 **Myself:** In theory you don't have to, but you should because it sometimes breaks if you
12 don't.

13 **Participant:** For example?

14 **Myself:** Yes, that's fine.

15 **Participant:** Is it case sensitive?

16 **Myself:** Yes, it is. So one thing I didn't mention is that the variables have to start with
17 a capital letter. So what you've done there is correct, that will work.

18 **Participant:** Yeah. So if I did, for example: "The prince" now, it wouldn't work?

19 **Myself:** That wouldn't work, no, it has to be a capital "P".

Syntax:
Capitalisation

20 **Participant:** (uses tool some more). What verbs?

21 **Myself:** Any verb is fine. You can use any verb. Any verb at all. You have to wait a
22 little bit after you click "Save Trope". So click "save trope", and then click refresh. Ah yeah,
23 so that's worked. OK, cool. So we can go to task 2. So for task 2, I want you to create a new
24 trope from the beginning. In that task, it says use those character roles, but actually I think
25 just use any character roles you want. But at first just have just one thing after another, so
26 at first don't put any branches in. So go to... go back to the tool. And click on the "new"
27 button. Very good. First, type in the name of the trope in that box, which says "trope name".
28 "Task 2" - OK, cool. Right, so first define your character roles.

Syntax: Using any verb

29 **Participant:** Is...?

30 **Myself:** A role. Did you mean to type "Hero". "The Her"? Oh, "The Hero", yeah, yeah.
31 So you don't have to use all of those roles. In fact, you can use any ones that you want to
32 make up.

33 **Participant:** So if I declare a role, but I will not use it, will it somehow affect the trope?

34 **Myself:** No, that will be fine. It won't appear, but it won't break it either.

35 **Participant:** OK. Places. (types) Should I create three places?

36 **Myself:** Ah no, just create however many you want. Actually no, I think that's fine. So
37 just use those for now. And then work out... have maybe two or three events using those
38 things.

39 **Participant:** (consults the documentation)

40 **Myself:** Oh. So if you put "The Mentor goes to the Castle", and make sure the Castle is
41 a capital "C".

Syntax:
Capitalisation

42 **Participant:** So I have to use, like, (types it in)

43 **Myself:** Yeah. So click on "save trope". I think it will help if you... yeah, click on
44 "Save trope", and then go to "edit" and select your trope. Then you'll be able to visualise
45 your trope as you are editing it. Select your trope from the... what did you call it? "Task
46 2". Yeah. So click on "refresh". Yeah: "The Mentor goes to the Castle". Ah yeah. Good.
47 Maybe have one more event. OK, very good. So task three was adding objects to your trope,
48 but you've already done that, so there's no need to do that. If we go to task 4, that's to add
49 some branches to your trope. So, using the "Or" keyword - and don't forget to put two spaces
50 before it - add some kind of branches to that.

51 **Participant:** (uses the tool some more)

52 **Myself:** So you have to put "Or": "Or the Hero goes to the Castle". Yeah, very good.
53 Ah, save that and see if... so every time you type a line, just to be safe, save it and refresh
54 it, because sometimes it won't work and this will help us work out what's broken it. Ah, but
55 yeah, this one works, so it's OK.

56 **Participant:** OK, so if I want to go further, for example to make a trope go for example,

57 you know, the way that "goes Castle". You know like the second step where the Hero is
58 already at the castle.

59 **Myself:** Ah yeah, so actually for this study, this is a very simple version of the language,
60 so I haven't implemented that. I can only have, at the moment, branches with one event,
61 unfortunately. So all the branches have to merge back to the story after one event.

62 **Participant:** Oh, alright.

63 **Myself:** So, something I want to ask you though, is: if you did want to do that, how would
64 you design the language so that you could continue that branch? What would you expect to
65 type?

66 **Participant:** I think that I would expect this to be... for example, I would add here
67 another branch.

68 **Myself:** OK.

69 **Participant:** "Or the Hero goes to the"... "Villain goes to the Garden". So for now, it's
70 like we have three options.

71 **Myself:** Yeah.

72 **Participant:** And I think really useful would be adding some variable that, another
73 variable that remembers the state. So let me represent this here, something like... (types
74 into text box) So instead of making the first option with this syntax without spaces, I would
75 make something like this. For example, first is this, second is this. So later on, you can like
76 refer the number.

77 **Myself:** So you can say "go back to scene 1", or "go back to event 1", and then it can
78 repeat. Yeah, that's a good idea.

79 **Participant:** Or easily you can just keep writing more spaces after this one.

80 **Myself:** Ah I see. I see what you mean. I think.

81 **Participant:** Yeah, so for example... I mean, for now it's simple, so you can just keep...
82 like this and then that there. "The Hero has the Apple". You kind of make the story go
83 further.

84 **Myself:** I think that makes a lot of sense. Indenting it another level would extend the
85 branch. That's kind of what I was thinking. I was thinking along those lines as well. OK,
86 that's good that that's an intuitive way to do it.

87 **Participant:** But at some point, when you add like ten levels, you know the spacing can
88 go really crazy.

89 **Myself:** Yeah, that is a problem. And that's actually why we have this ability to put the
90 tropes inside the tropes. So once it gets really crazy, you can just refer to another trope that
91 already has those...

92 **Participant:** Like functions...

93 **Myself:** Yeah, exactly. OK, cool. What's the next one. So we have the branches. So
94 task 5 is to combine two tropes into a story. So far we've been using that "edit" tab, there's
95 also a tab called "arrange" on the top left. If you click on that. So this allows us to combine
96 multiple tropes together. So you'd say: "This trope is happening in the story, and this trope
97 is happening", and it will visualise the different possibilities for those combined tropes. So

Limitation:
Branch
length limit

Feature:
State man-
agement

Usability:
Tropes

98 if you leave that on task 2, click the "+" symbol, and then click on another trope, maybe a
99 simple one at first, because sometimes this takes a long time. So example 1, for example. And
100 sometimes it takes a little time to generate all the permutations. But it should appear on the
101 right in a little while. It does take some time.

102 **Participant:** Ah, there is something.

103 **Myself:** It should replace that... there we go. So it's generated all the different combi-
104 nations of events that can happen with those two tropes.

105 **Participant:** Oh yeah.

106 **Myself:** So you can zoom in, if you hover your mouse in it and scroll up, you can zoom
107 in to see the events. So you can see the red arrows follow the "Task 2" trope, and the blue
108 arrows follow the "Example 1" trope.

109 **Participant:** And it mixes everything together.

110 **Myself:** Yes, exactly. OK, cool. So yeah, you can imagine, you could have more than two
111 tropes, but it takes a long time to generate this.

112 **Participant:** So if I created a really long trope, and then combined with another really
113 long trope, it would create a much bigger scheme than now.

114 **Myself:** Yes, exactly. So I have to work out how to optimise it so it generates it a lot
115 more quickly.

116 **Participant:** Yeah, because sometimes not all the... I think that sometimes some options
117 might be really boring. You know, generates, I don't know... You could make a system that
118 detects for example, and create a trope... when you combine two tropes, then you could have
119 a, like a system that for example first will always put an event like going somewhere. Then,
120 for example, events like meeting, like two people meet together.

121 **Myself:** So it would kind of prioritise the more interesting events. Yeah, that's a good
122 idea. OK.

123 **Participant:** Yeah, in some way. Or yeah, depending on what the user wants to create.

124 **Myself:** That's a really good idea actually, I hadn't considered that. OK, yeah, very
125 good. OK, so task 6. So go back to the "edit" tab. For this one, I want you to take the trope
126 that you have made, which is called "task 2", and embed that inside a new trope. So create
127 a new trope and put task 2... just put task 2 at the end of it, because that's the only thing
128 that works right now. So yeah, have a basic trope with some events, and then put "task 2" at
129 the end.

130 **Participant:** Hold on. And now if I... task 2?

131 **Myself:** So put some other event first, so say "The Prince goes to the Room, Then 'Task
132 2' happens". So if you write "then"... actually, no, "Then the 'Task 2' trope happens". And
133 "Task 2" has to be inside double quotes. Actually, that might work. Save that. I'm not sure
134 if I... so usually you have to put "Then the"... and then the name of the trope, and then
135 ... "trope happens". But that might work, so see. "Task 6"... oh, there's two of them. OK.
136 Click refresh and see if that one works. That one won't work. OK. So you have to put "Then
137 the 'Task 2' trope happens".

138 **Participant:** Here? Or inside the box?

139 **Myself:** That's right, that's right. "Trope happens". OK, then save the trope, then
140 click "refresh". It might not like it because you have a blank line at the end. OK, try that.
141 "Prince"... OK. "The Prince is a role, The Room is a place". Ah! It worked. Great. It was
142 just a little bit slow. Cool. Excellent. Alright, so for task 7, I want you to just play with the
143 tool and try to come up with some new ideas. Maybe combine some other tropes together
144 with the "arrange" tab to make a new story. So one limitation I have to explain is that at the
145 moment the story can only be five events long, because in the background it's using this really
146 complicated permutation generation thing. And if you have, like, too many possibilities, it
147 just takes far too long for anything longer than five events.

148 **Participant:** Alright. Five events in one trope?

149 **Myself:** Yep, that's right. So, yeah, try changing some tropes and combining them
150 together.

151 **Participant:** (uses the tool)

152 **Myself:** So, I'm not sure... save that, but I have a feeling it might not actually work,
153 because it has...

154 **Participant:** (indistinct)

155 **Myself:** So rather than something altering the state directly by saying "something is
156 somewhere else", it only works with verbs. But there is a small chance it will work, so save
157 it and see what it says. No, it won't work because: "The Child is in the Home, The Woman
158 is in the Home" is just altering the state. Unfortunately at the moment, it only works with
159 verbs like "The Child goes to the Home", or "The Woman goes to the Home". But you can
160 use any verb you want, but it has to be...

161 **Participant:** Alright, because I see that "is" is used for declaring the variables.

162 **Myself:** Ah yes, I think that's why it gets a bit confused. So you have to use "is" just
163 for the definitions at the top. Hmm, I'm not sure what's wrong. Ah, there you go. OK. So,
164 do you want to combine that with another trope? So go to the "arrange" tab and see. Or do
165 you want to put some branches in or something?

166 **Participant:** No, it's fine. Um, if I combine... Hang on, I will combine this with... can
167 I get rid of some tropes.

168 **Myself:** Yeah, if you go back to "edit", you can delete some of those.

169 **Participant:** It is annoying to (indistinct)

170 **Myself:** Ah yeah, so click "delete". Yeah. "Task 6"... if "Task 6" was at all complicated,
171 it might take a little while to generate. So while waiting for that, I can ask you some questions.
172 So leave it running. Do you think that using tropes like this, combining them to form a story,
173 is a good way to create non-linear stories. Do you think it's a good component for stories?

174 **Participant:** Uh, I think it is a good idea. However, it's like, at this point when you are
175 combining them, you can't really decide where you put which trope, or...

176 **Myself:** So you always have to just combine them, you can't put them one after another.
177 Yeah, that's a good point.

178 **Participant:** Yeah, so for example the user would like to choose to start only from the
179 trope, for example, "Meeting", where two people meet, and that's the beginning of the story,

180 because maybe they have an idea for this. And I think it would save time for generating all
181 the possibilities, because you could cut out half of the solutions, right?

Feature:
Story structure

182 **Myself:** Yes, exactly, that's a very good point. So rather than always having the trope
183 merging all the events together, so that it's always one trope, then another trope, then another
184 trope, instead of all those permutations. Ah, there we go. It took a long time to do that, for
185 some reason. Ah yeah, so constraining it in some way so that it generates a subset of those
186 permutations would be a good idea, and then it would take less time to generate all of these.

187 **Participant:** Yeah, for example the user will choose to start with the "Trope 1", and then
188 for example, he can mix all the other tropes later on. Because at this point, I see that the
189 tropes, they are...

190 **Myself:** In this case, in "Task 6", you... I guess this was kind of a way of ordering it,
191 because "Task 2" was embedded inside "Task 6" as the last event. So that was kind of a way
192 of making sure that "task 2" happens at the end of "task 6". So there's kind of a way of
193 ordering tropes there, but it's not ideal. As you said, it would be good to specify it with this
194 interface, rather than merging it all together.

195 **Participant:** I think if the user had some kind of possibility to decide what to put where,
196 and then generate, that would be useful. But at this point, for sure, it generates lots of
197 ideas that you can go through and just search for the ones that you like. I'm just trying to
198 understand...

199 **Myself:** I think this... it looks like this one has gone slightly wrong as well, because I
200 can't see the story trope anywhere. At the beginning it says "task 2" rather than "story", so
201 I'm thinking that that's some kind of mistake. Because you selected the "story" trope, but
202 that isn't anywhere. It's quite strange.

203 **Participant:** I'll try to do this again. If I refresh, then will I lose?

204 **Myself:** You won't lose it, no. It's all stored on a server. So yeah: select "story", then
205 "task 6".

206 **Participant:** (indistinct)

207 **Myself:** So again, it might take some time, so I'll ask you some more questions. So in
208 terms of this programming language, do you think that if you didn't have any background in
209 programming, would it be easy to learn, do you think?

210 **Participant:** Well the syntax is not really difficult. I mean, it's difficult to say because I
211 know how programming works.

Usability:
Ease of use

212 **Myself:** So do you think that having a real... having it look like a proper programming
213 language, with proper variables and everything, so you have more power, would be better than
214 having a simplified language like this? So, it'd be more effort for the user to learn it. Sorry?

215 **Participant:** I mean, if you use much more complicated language in your program, I
216 think less users will be able to use it. They will just give up at the beginning because they will
217 not understand. That's the thing. I think the simplicity of the language is fine. But before
218 starting something like this, I think it would be really useful to create some kind of a tutorial.
219 I mean, not a tutorial, but some kind of a table or chart with the syntax that you can use,
220 with examples of "this is correct" and "this is incorrect". Because if the user will sit and start

221 using it, he will get only compile errors. He will not have a person to ask them. I mean, not
222 many people have the owner of the application there to help them use the program.

223 **Myself:** No, good point.

224 **Participant:** Again, (indistinct)

225 **Myself:** No, I'm not sure what's happened there. That's a shame. So instead of "task 6",
226 try using something else, like "example 2" or "example 1". So this one should be... this one
227 shouldn't take so long because it's a lot simpler. I'm wondering why it's taking a long time.
228 Oh, there you go.

229 **Participant:** It might be my laptop. Oh, now it took the "story" and "example 1".

230 **Myself:** OK, so yeah obviously there's some kind of bug.

231 **Participant:** I see that the "story", it happens more often now. I mean this trope.
232 Because when you had "task 2" and "task 6", it was only "task 6" here, and the rest was
233 "task 2".

234 **Myself:** Yeah, I don't think that "story" appeared at all in the other time you used it.
235 So it's strange.

236 **Participant:** No, there wasn't a "story", yeah.

237 **Myself:** OK. Well, I think that's everything. Unless you have any other general comments,
238 we can finish there.

239 **Participant:** Um. It's fine for me. If I had any thoughts, I will send an email. I think
240 that you could change actually the way, like making, like when a person sees, the person thinks
241 there are two options, when in fact there are three options.

242 **Myself:** Yeah, that is tricky. I was thinking maybe... the other way I was thinking
243 of expressing it was something like "Or:", and then everything underneath that were the
244 possibilities.

245 **Participant:** Yeah, like with numbers or something.

246 **Myself:** Yeah.

247 **Participant:** For example, first branch... that would actually be nice. Let me show
248 you...

249 **Myself:** Hmm, numbers.

250 **Participant:** Let's say that - I'll just copy from here - the "Then" is starting the branching,
251 right?

252 **Myself:** That's right, that's the first possibility.

253 **Participant:** OK. So I thought that you could make like an event, for example, "The
254 Hero goes Home", and a number, and that would be the first branch, and you could write a
255 whole story in there. Even with the spaces... (types into box) So for example, that would be
256 the first branch, and that would be all in the evens, like following the first branch. And when
257 you want to make another, you just bring the second branch. You make like the second. I
258 think it would be clearer without implementing the thing like "Then". Or maybe before, you
259 could make the comment "new branch", at this point the user would know that's it's like, now
260 he can make it clearly.

261 **Myself:** Yeah, I see what you mean. I think that makes it more like a proper programming

262 language in some sense, because there's more to learn than just typing, but it's a lot clearer
263 once you have typed that way.

264 **Participant:** I think for the user, you could make it simpler to find a way to not call it
265 like a programming language. Or maybe not call it a branch, but "new"... I mean, I have an
266 idea, I will have to sit and think about it. I could write you an email later.

267 **Myself:** Yeah, please do.

268 **Participant:** I'll make a sketch or something, so.

269 **Myself:** I'll include your email in the thesis when I do the write-up, because I think you
270 have got some really good ideas for how to improve the language, so I think this has been
271 a really good session because I've got some really good feedback from you. So thanks very
272 much.

273 **Participant:** That's fine. I'm happy that I could help, actually.

G.7 Participant G

Dialogue starts at 12:16.

1 **Myself:** Now we can kind of start doing some tasks with it. So, the first task is to just take
2 one of the existing tropes and edit it. I should explain that there's a couple of limitations.
3 So because this uses what's called an "Answer Set" solver, which kind of generates all the
4 different possibilities, I've limited each trope to just five events long. And the reason for that
5 is because when you combine multiple tropes together, it can take a long time to generate all
6 the possibilities.

7 **Participant:** Sure, you get a combinatorial explosion.

8 **Myself:** Exactly, so yeah, bear that in mind.

9 **Participant:** OK. Alright, well I have some questions, but let me pull up one here.

10 **Myself:** OK.

11 **Participant:** So you want me just to edit this, huh?

12 **Myself:** Yeah. And you can use any verb, that should be fine.

13 **Participant:** OK, so I will add. Let's see here. So that was one of my questions, was...
14 so down here we have things like "goes" and "finds", and so on. So you can use basically any
15 verb in there?

16 **Myself:** Yes, that's right.

17 **Participant:** Although it knew to turn "goes" into "go", and "finds" into "find", so you've
18 got some smarts in there somewhere about that.

19 **Myself:** That's right, it uses Wordnet to work out the basic form of the...

20 **Participant:** OK, so it tries to find it as a conjugation of it.

21 **Myself:** Yes, exactly.

22 **Participant:** OK.

23 **Myself:** Oh, sorry, you have to click "save trope", and that sends it to the... because
24 this is actually running on a server, so it sends it to a server and puts it in a database.

25 **Participant:** Do I need to make... do "new" or something...?

Syntax: Using any verb

26 **Myself:** No, just editing this one is fine. So click "save trope", then click "refresh", and
27 that should update it. There you go.

28 **Participant:** OK.

29 **Myself:** Cool, so that's enough of that first task. The second task is to create a new
30 trope. So in that description it says you can use those character roles, but ignore that. Just
31 create any trope you like. But just for now, don't add any branches to your trope. Just have
32 a sequence of events.

33 **Participant:** Just a straightforward sequence.

34 **Myself:** Exactly. So click on the "new" tab. Then first type the trope name in that box
35 underneath. And just in case, please remove the apostrophe. Do both if you want, yeah.

36 **Participant:** Do I need to save, or?

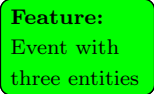
37 **Myself:** Um, no, don't save it yet. Put some stuff in there first.

38 **Participant:** OK. So (types into tool). I've gone off the rails here. Let's say: "The Witch
39 is a role". I can do these in any order, I assume?

40 **Myself:** Yeah, yeah. As long as they are at the beginning, it's fine.

41 **Participant:** Yeah, OK. Let's say... (types into tool). So we can start with just anything,
42 so I can just start with, let's say... (more typing).

43 **Myself:** OK, cool. Now save your trope. If you go to the edit tab, you can kind of visualise
44 it, just in case there's some kind of error or something. Which is actually quite likely. So click
45 "refresh", let's see if it's managed that one. Oh, yep.

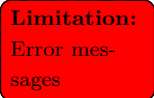
46 **Participant:** Yeah, OK, so (indistinct). OK (types some more). So we don't have objects
47 in the verbs, huh? So I can't say: "The Witch gives the Apple to Sleeping Beauty"? 

48 **Myself:** Yeah, you can. You can.

49 **Participant:** OK.

50 **Myself:** Ah, but yeah, save it and refresh. Just in case.

51 **Participant:** Now, or before I do that? Here, let me take this out and save. I think so
52 far I'm OK. So, "The Witch"...


53 **Myself:** No... a very un-useful error. 

54 **Participant:** I'll take that out.

55 **Myself:** OK, I wonder why...

56 **Participant:** Let's just... that's cool. We'll do it this way (typing).

57 **Myself:** Oh, it's not appeared. Oh, there we go.

58 **Participant:** There it is. Ah... (typing). Like this? 

59 **Myself:** Yeah, that should be good.

60 **Participant:** Oh, you wanna bet?

61 **Myself:** Yeah.

62 **Participant:** No problem.

63 **Myself:** Getting a bit overconfident at this point.

64 **Participant:** Right, yeah. (more typing)

65 **Myself:** Remember, it can only be five events long.

66 **Participant:** Oh, OK. So I'm about at my limit here.

67 **Myself:** So, you can add some branches to it later.

68 **Participant:** Right, let's fix this, then. Take out some stuff. We got 1, 2, 3... got one
69 more. Let's see (types).

70 **Myself:** Cool.

71 **Participant:** Alright. OK.

72 **Myself:** So the next task...

73 **Participant:** Yes, next task.

74 **Myself:** Task 3 was adding objects to your trope, but you've done that, so that's good.
75 So task 4 is to add some branches to your trope, using the "Or" keyword. And remember to
76 indent with two spaces.

77 **Participant:** OK. So... now I'm getting off... Sleeping Beauty's pretty sequential, I
78 think there aren't any "ors" to it, but I'll come up with something. So with the branches, am
79 I still trying to keep within five actions, or five on each branch, or...?

80 **Myself:** No, actually, in this case it doesn't matter. So you can have as many branches
81 as you want.

82 **Participant:** OK. So, let's do let's do it here, and I gotta go back and refresh myself on
83 how the "or" works here. So yeah, OK. (indistinct) ...in the subplot there. So the "or" can
84 just be one... right now it can just be one action, basically?

85 **Myself:** Yeah, that's right. It can only be one thing.

86 **Participant:** OK. I'll change it into a subtrope in a minute. I'm sure that's here some-
87 where. I'll have a subtrope where the helper helps Sleeping Beauty, then. Let's see... OK.
88 Alright, so I got an "or" over here. Alrighty, is that enough, or?

89 **Myself:** Yeah, that's enough. So before you can do your subtrope, there's one thing you
90 can look at is if you click on the "arrange" tab.

91 **Participant:** Uh-huh.

92 **Myself:** You can combine two tropes together. I would advise... so start with two simple
93 tropes, so combine example one and maybe example two or something.

94 **Participant:** Oh interesting, OK.

95 **Myself:** And this will generate all the possibilities... yeah, of... if you have those two
96 tropes happening at the same time. Or it should do. That doesn't look quite right to me.
97 Hang on. It's doing one, then the other.

98 **Participant:** It's sort of showing you picking up the second... oh. So it actually put
99 example two first, but then it tacks one on to the end of...

100 **Myself:** Yeah, that shouldn't be right. Hmm.

101 **Participant:** Can I... is there...

102 **Myself:** Delete that third one, yeah. Try... let me have a look.

103 **Participant:** I don't have a refresh button here, so try again here.

104 **Myself:** It's still doing that. Let me see...

105 **Participant:** So let's put two first.

106 **Myself:** Oh, that's interesting. Ah OK, some strange bug has happened where it's over-
107 written Example One with... no, it's overwritten example two.

Limitation:
Branch
length limit

108 **Participant:** Oh, it's because I saved... because I didn't save example one? Is that the
109 problem?

110 **Myself:** Maybe. No, it's OK. I can edit it from my end and put it back in.

111 **Participant:** OK. You do that.

112 **Myself:** OK. So example two was the branches. But maybe we should avoid... so go to
113 example three and maybe combine that with example one.

114 **Participant:** With example one?

115 **Myself:** Yeah. It might take a little while.

116 **Participant:** (indistinct)

117 **Myself:** Yeah, something strange has happened. Could you just refresh the page in your
118 browser?

119 **Participant:** Sure, let's do that.

120 **Myself:** OK, now go to arrange, and select example...

121 **Participant:** Let me just pull up example one and see... so that looks OK.

122 **Myself:** Yeah, that looks OK.

123 **Participant:** Alright, so example one first, and then example two?

124 **Myself:** Yeah.

125 **Participant:** OK, there's example one that's pulled in.

126 **Myself:** Hmm, it's still doing that.

127 **Participant:** It's still flipping it around. Well, y'know, it's beta software.

128 **Myself:** Well, yeah, it's a bug. OK, try... instead of example one and example two, try
129 example three and example one.

130 **Participant:** So example three first?

131 **Myself:** Yeah.

132 **Participant:** OK. So let's pull in one first.

133 **Myself:** So I'm just trying it on my end, and I...

134 **Participant:** Of course, it's working fine?

135 **Myself:** This was working before. Let's see. It usually takes a while to combine two more
136 complicated tropes, but it doesn't usually take this long. So I'm wondering why... Oh, it's
137 done it on... OK, so it's just worked for me. I think it's quite finicky about the order in
138 which you do it. So refresh the page again. And then click on "arrange". Then do trope three
139 first. Then wait for it to appear.

140 **Participant:** OK.

141 **Myself:** There we go.

142 **Participant:** Ah, there we go.

143 **Myself:** And then, yeah, example one. And it takes a little bit of time. Even so, hmm.
144 There we go.

145 **Participant:** Ah, there we go. Wow.

146 **Myself:** So you can zoom in on that by hovering over it and scrolling... yeah, that's
147 right. So in this case, it hasn't done what it did with the other one, it's actually kind of
148 combined them. So if you look at the first event, and the branches it takes from example one

149 and example two. Oh no, it's doing example three first. Oh, this definitely wasn't happening
150 before.

151 **Participant:** Except here, it's doing...

152 **Myself:** Yeah, this is...

153 **Participant:** So after every step, it's offering to jump into example one, huh?

154 **Myself:** Yeah. Although strangely, it's the first... yeah, the first event is only example
155 three. But this definitely wasn't the case before, so I'm kind of wondering what's going on
156 here. It's a shame, because this is... I've done seven...

157 **Participant:** So it's supposed to do all the combinations between the two?

158 **Myself:** Yeah, that's right. This is the seventh time I've done this study, and I think this
159 might be the last one, so of course it's going to go wrong this time. Yeah, you can see what
160 it's supposed to do. The solver will run it for all of the, yeah, all the different combinations
161 of them there.

162 **Participant:** Uh-huh. Interesting. OK. Well, I get what it's supposed to do.

163 **Myself:** OK, cool.

164 **Participant:** But I see why you get... you really get an explosion if you have complicated,
165 long tropes.

166 **Myself:** Yes, definitely. OK, so the task... next task is to put a trope inside a trope. So
167 take the trope you already created and put that... either put a new trope at the end of it,
168 or put this at the end of a new trope. Again, this only works if you put it at the end at the
169 moment, if you put it somewhere in the middle, it's not going to work.

170 **Participant:** Oh, it's not going to work. So, uh, (types). So, did the roles have to match
171 up if I stick it at the end? I guess it won't... /featureDifferent characters in different tropes

172 **Myself:** It doesn't matter...

173 **Participant:** It doesn't really care, does it?

174 **Myself:** No it doesn't, it doesn't care.

175 **Participant:** Ah! (indistinct). Happily ever after. Actually, let's do this. So let's change
176 this. Let's make (typing). I'm having rather too much fun with this.

177 **Myself:** Good, that's all good. Oh, OK. Alright. Save that, let's see if this works, yeah.
178 OK, go to "edit", and then select it. And just see... there we go. That looks OK.

179 **Participant:** At least we didn't... oh, OK. It did it, no "compile error", that's good.
180 OK.

181 **Myself:** So you wanna put that at the end of the...

182 **Participant:** So let me save that, and I'll put that at the end of the other one.

183 **Myself:** That's right.

184 **Participant:** And I'm doing it as a subtrope, so I do "Then the 'Item Search' trope
185 happens", OK. So... (types). Do I have to put some quotes?

186 **Myself:** Yeah, you have to put it in quotes.

187 **Participant:** Alright.

188 **Myself:** Yeah, so save it and click "refresh". But... so the five event limit includes the
189 trope you put at the end, so click on...

Limitation:
Five event
limit

190 **Participant:** Oh really, so I might have too much?

191 **Myself:** Yeah. Click on "refresh" and see... first trope... so yeah, it's already getting
192 to five events there. So delete some of the ones at the beginning.

193 **Participant:** So I'll cut out some things here. Ah, let's see. I don't need this. That's all
194 syntactic sugar there. Let's see. OK, there we go.

195 **Myself:** Cool. That's the first event of the happy ending. But you can see what was
196 gonna happen.

197 **Participant:** So we ran out of... yeah, I'm still running out of events, but yeah, right.
198 Here's the... OK. Yeah.

199 **Myself:** Alright, cool. I think that's... oh no, the last one is just. It says "free story
200 creation", you can create your own thing. But you've pretty much done that, I think. You've
201 explored all the possibilities. Cool. Alright. So I just kind of want to ask you some open-
202 ended questions about that. So, you can imagine like, obviously you have a programming
203 background, but if you were someone with a non-programming background who's just a story
204 author used to writing, do you think that this would be an easy kind of language to learn and
205 pick up?

206 **Participant:** Yeah, I think so. I think it's very self-evident what's going on, and how to
207 use it, so I don't think that anybody would have a problem.

208 **Myself:** And do you think that these tropes would be a good formalism... well, a good
209 way to model the story? So when you're describing these components of a story and combining
210 them together, do you think that is something that would be intuitive to an author?

211 **Participant:** My only concern I guess would be about the level of abstraction. So, you
212 know I think probably all of us who work in this area fancy ourselves authors to some extent
213 or another, and still I wonder... so I mean at the level I wrote it I think is a little bit too...
214 well... I guess Hero and Heroine, um, that's OK. I guess a lot depends on the verbs, I guess.
215 I think if I can use the right kind of verbs, yeah, I think I can express a lot of tropes. I'd
216 have to give some thought as to... you know, I'm sure there are limitations. But I think at
217 a certain level, this would be fine.

218 **Myself:** OK, cool. Right. I think I've gotten everything I want to... all the information
219 I need. Do you have any other general comments you'd like to make?

220 **Participant:** Yeah, so... one of the things I tried to do in MINSTREL... I guess one
221 of the metatheories in MINSTREL was that creativity or something about creativity comes
222 from getting lots of combinations of things, and maybe even unexpected combinations, right?
223 So there's a sort of this tension between... you want this combinatorial explosion kind of
224 thing going on, but you also somehow want to limit it. Or I guess in MISTREL's case, what
225 I was trying to do was sort of intentionally find paths in the combinatorial explosion which
226 hadn't been explored a lot, right? So MINSTREL had this notion that... it had this episodic
227 memory of a bunch of stories, right? So if it saw something, like if the Prince going to the
228 woods happened in lots of things, and it was already in its episodic memory, it would say:
229 "Well, that's not very creative", and maybe try to avoid it, or at least say "Well, if I'm telling
230 a story, and I put that in it, I probably haven't made the story very creative", right?

Usability:
Ease of use

Feature:
Story structure

231 **Myself:** OK, right.

232 **Participant:** So when I think of these tropes, what I want to do when I combine them,
233 or when I'm thinking of combining them, I think it would be really interesting if... so like,
234 you have the roles, right? So "The Prince is a role, Sleeping Beauty is a role", and so on, so
235 it'd be kind of interesting when you're doing this, when you're adding two tropes together, to
236 sort of be able to cross-match across those roles, right? So maybe... so I have my generic
237 trope about somebody saving somebody, right? And I have my other generic trope about,
238 you know, people falling in love and getting married, or whatever. And then so I can do these
239 sort of combinations across these two tropes, right? So I get the typical story: "The Hero
240 saves the Princess, and she falls in love with him, and then they live happily ever after", but
241 I could also get: "The Princess saves the Prince", right? And, you know, and so... it would
242 be really interesting in this trope language you have, where if you could... if the language
243 would support sort of matching up those roles and figuring out even objects too. So you have
244 "The Apple is the object that causes Sleeping Beauty to go to sleep" but maybe have objects
245 in other tropes, and when you cross them over: "Oh, this object", you know, King Arthur
246 pulls Excalibur out of the stone, and then later on that becomes the object in the "Sleeping
247 Beauty" trope that makes her go to sleep, for some reason, right, or something.

248 **Myself:** Hmm.

249 **Participant:** So some kind of support for that sort of... combinations of different things
250 across the different tropes, and again maybe doing some kind of constraint-based thing to figure
251 out "maybe this makes sense, or this doesn't make sense", you know. Well, this object is not
252 the kind of object, it's not a consumable for instance, right? So maybe in the "Sleeping Beauty"
253 trope, it has to be a consumable that puts her to sleep, and Excalibur's not a consumable, so
254 that doesn't fit in there. But this potion that you found when you were exploring the dungeon,
255 that could be a consumable, or something along those lines.

256 **Myself:** So yeah, you have to have some information about the type of role that you put
257 in, if it's compatible with other roles in other tropes, and yeah also the type of object, as you
258 said, it could be a consumable, it could be a weapon, for example. And if that can be swapped
259 with another object in another trope to serve the same function.

260 **Participant:** And MINSTREL did some of that. Of course the problem with that is you
261 get this sort of bootstrapping problem that you'd really like, I mean, sort of the first thing
262 you'd like when you start writing a storytelling program of some sort is complete common-
263 sense knowledge about the world, you know? So just as a starting point, right? So you'd like
264 to know an apple is a consumable, and all these different kinds of things. So there's a big
265 problem there, because how do you get all that knowledge? But yeah, once you have it, once
266 you do some constraint stuff that I think is kind of interesting... The other thing I think
267 would be an interesting... I mean, I'm just rambling, so...

268 **Myself:** No, this is all interesting stuff, this is all good feedback.

269 **Participant:** Yeah, so I mean, I'm just taking the opportunity to discuss things with you,
270 you know. But when I think about this notion of having... so your notion is that the trope
271 act as constraints, and you're gonna have some agents acting, and somehow being constrained,

272 right? So one of the interesting things about creativity is when you throw the constraint off,
273 right? So the classic example is kind of modern art, right, where some artists decided: "Well,
274 you know, there's sort of a constraint where paintings should look realistic, but what if we get
275 rid of that?", you know? And we just, we paint stuff that doesn't look realistic. Or you can
276 imagine, if you're doing fairytales, right? You have a constraint that the Hero is a man. But
277 it would be really interesting to have a... some kind of meta-reasoning or something in the
278 program that would say, or even just experiment really without knowing anything by saying:
279 "Well, I'm going to throw this constraint out the door", right? Or even if I read a trope, and
280 I say, you know, "The Woods is a place, and it happens here", maybe you'd have something
281 which says: "Well, I'm going to try using this trope, but I'm gonna throw out that constraint,
282 I'm gonna leave it out of the trope."

283 **Myself:** Yeah, I think as you said, it would be nice to invert or subvert a trope by reversing
284 the roles of some characters so as you said before, the princess rescues the hero, or whatever, in
285 a trope. To have that kind of mechanism to add some kind of element of unexpected creativity
286 to it, I think that would be a really good idea. What was I going to say? It's really late here.
287 It's so late, I'm not really able to think straight. What you were saying before about the.. so
288 adding some elements of creativity to it. So what did you just say about that?

289 **Participant:** Well, so, I can talk forever about creativity as you can imagine... so I
290 think one thing is this notion of you know, you've got a big solution space in some sense, you
291 know, whether it's combinations of your tropes and your actions, or whatever it is. And some
292 parts of it are well-trodden, and some parts aren't, right? And so the ones that aren't are sort
293 of intrinsically more interesting, let's say, or I would say that they are. On the other hand,
294 they have to make sense, and they have to, well, they have to make sense not only in terms
295 of the real world, but also in the sense of the storytelling, right? So you can't just publish a
296 book which is 10,000 words that have never been used in this combination before. Well that's
297 certainly new terrain, but it's not interesting, right? So there's this tension between structure
298 and living within the constraints and being interesting by being in new spaces or selectively
299 throwing out constraints and seeing where that gets you.

300 **Myself:** Yeah, so what I was going to say, which I forgot to say, which is: the way that
301 this hooks into the multi-agent system, it's not just constraints, so the characters, the agents,
302 see the story as a set of social norms which govern their behaviour. So, as they're social norms,
303 they're kind of suggested behaviour, but they can actually change to break that, break these
304 rules, and obviously there are certain consequences, there would be some penalty for breaking
305 these rules. But if you give your characters, for example, emotional models, so that if they are
306 able to attain their goals, they're happy, but if they're unable to, then they get angry. Then
307 the angrier and angrier they get, the more likely they are just to say "The hell with it", and
308 break away from these constraints, right?

309 **Participant:** That's interesting, yeah.

310 **Myself:** Yeah, so that's kind of what all this compiles to, a set of what we call social
311 institutions, rather than just constraints.

312 **Participant:** So is the intention that the agents... so I write: "The Prince goes to the

313 Woods", so is the intention that the Prince knows something about how to go to the woods,
314 Or. . .

315 **Myself:** Yeah, the Prince has a. . . you'd write the Prince with a plan involving going to
316 the Woods.

317 **Participant:** Gotcha, OK. And he has, presumably his choices about, you know, he knows
318 he can ride his horse there, he can walk there, something along those lines.

319 **Myself:** Yeah, exactly.

320 **Participant:** Gotcha. Hmm. Yeah, that's interesting. So another thing I always thought
321 was kind of interesting or sort of fruitful area is: so supposing you get a whole boatload of
322 these tropes. So one of the things MINSTREL did, right, so I said it had an episodic memory,
323 right, so it put, you know, as it "read" stories, which I really just handcrafted and added to
324 the memory, but you know, as it "read" stories they get added to the episodic memory. And
325 the episodic memory does a couple of things. So one of the things is it notices generalisations,
326 right? So "Oh, the Prince rode a horse in this story, and the Prince rode a horse in this story,
327 and the Prince rode a horse in this story" and one of the things that comes out of that, like
328 I that, like I said, is: well, you notice I've seen this a bunch of times, it's not very creative,
329 you know. On the other hand, you do know "Oh, well 'Prince riding a horse' happens a lot of
330 times, so I know that's a standard thing I can do", right?

331 **Myself:** So that's kind of like a trope, it identifies tropes in these stories.

332 **Participant:** Well, except it's not really capturing the literary aspect of the trope, right?
333 So it's really just, it's more like learning to plan from the episodic memory. So if you see
334 a bunch of stories where the Prince is in location A, and he ends up in location B, and in
335 between it says he rode his horse, you can sort of learn from that: "Oh, to get from location
336 A to location B, I can ride my horse", right? And then the other. . . the episodic memory also
337 can do some generalisations, see if he walks from A to B, it can sort of figure out: "Oh, to get
338 from A to B, I can walk or I can ride my horse", right?

339 **Myself:** Ah right, OK.

340 **Participant:** Right. So and then the other thing that MINSTREL's episodic memory
341 did is it had these things which I called "TRANS", which, I don't know, stood for something.
342 But basically the idea was that you could have some rules about how you could transform
343 things. So for instance one of those things was "use an agent". So the transform "use an agent
344 transform" says: "well, if you need to do something, you need to get from A to B, you can
345 do it, or you can get an agent to do it for you". So that was kind of a general thing, but it
346 would sort of figure out: "Oh, I can apply that to different things", right? So the story I tell
347 in my dissertation is that when my little niece was like 8 years old or something, spilled her
348 milk in the kitchen, and she knew she was going to be in trouble for spilling it, right? And
349 normally grandma cleaned it up by getting paper towels. Well she couldn't reach the paper
350 towels, right? I'm in the other room watching this. And so she leaves the apartment, and
351 she goes next door, and she comes back with one of the kittens that had been born next door
352 previously, and she puts it on the table to lick up the milk. So she's figured out: "Well, if
353 I can't clean it up, I can get some agent to do it for me.". So you have these memories in

354 there, it's kind of a silly example, but let's say you know that the Prince can get to the woods
355 by walking there, but that's the only thing you know, but maybe through this you can figure
356 out: "Well, I can get an agent to walk me there, instead of me walking there. Horses can
357 walk. Oh - I can ride a horse there", right? On the other hand, if you pick the Princess to
358 be your agent, "I have the Princess carry me to the woods", that doesn't make much sense,
359 right? But you can sort of figure things out like that. So that's sort of level one, although
360 I never really totally got this working, but the idea was: I could pull the story structures
361 into episodic memory and let the same thing happen, right? So if I had what you would call
362 a trope where the Prince goes somewhere. I might be able to automatically learn a variant
363 of that trope, where instead of the Prince going somewhere, the Prince gets an agent to go
364 somewhere for him and act on his behalf. So instead of the Prince going to the woods and
365 kissing Sleeping Beauty, he gets somebody else, you know, the Jester, the Court Jester, to
366 go and do it for him. Now that's a weird kind of thing, right, but in a way, that's actually
367 kind of an interesting story, where does it go from there? So, I mean, again it comes back to
368 the thing I was talking about, you've got different ways to generate different combinations of
369 things, and somehow you want to constrain those to be interesting somehow. And in different
370 ways. I forgot how I got started on that. Looking at what you've written, I can see writing
371 sort of tropes that apply to tropes, right? I read a book once about writing, and about writing
372 novels, and the author was like: "Basically, a novel is: somebody has a problem to solve, and
373 this is how he's going to solve it.". He starts off doing it, and then there's a problem in how
374 he's doing it, and then he solves that problem, but then there's a problem in how he does
375 that, and there's this ripple of problems that eventually comes back and he solves the first
376 problem, and everything's great. But that's kind of a meta-trope, right? So any trope where
377 I'm going and trying to do something, in the middle of there I can pick one of the steps and I
378 can say: "I'm going to break that step", and then I'll insert, as you have, a sort of sub-tropes
379 thing. So I'll put another trope in there, and that will lead me off: I'm doing this, and then
380 I'll do it again, and you can see how that sort of thing can cascade into ending up in a long
381 story of fixing this, and fixing this, and fixing this, and eventually getting around to whatever
382 you were initially trying to do.

383 **Myself:** Yeah, it sounds almost kind of like a recursive process. Yeah.

384 **Participant:** Yeah, but I can see in the framework you've set up there, I can see how
385 that could work really easily. By extending the language, you can get this kind of really neat
386 kind of fertile ecosystem of things you can combine in different ways. And one of the things
387 that I never explored was sort of this notion of being an author assistant, right? So one of the
388 problems when I talk about that is, you know, as they say, yeah, it's pretty cool when I say
389 the Prince gets the horse to be his agent to take him to the woods. Oh, it's figured out how to
390 ride a horse to the woods. It makes a lot less sense if the Princess is the one that carries him
391 to the woods, right? But if you had a human author involved in it, you could imagine a thing
392 where as the human author is typing in his tropes and everything, the system is popping up
393 and saying "Look, you have a thing where the Prince is going to the woods. I have a trope
394 where a guy tries to go somewhere, and he runs into a troll under the bridge, and the troll

395 asks him to get some item. Maybe we could insert that here and make this trip to the woods
396 more interesting.". But then if it suggests something completely stupid, it says: "Hey, the
397 Prince goes to the woods, I have this trope where the Princess carries him to the woods", you
398 can say: "Yeah, no, I think that's stupid. But I liked the idea about the troll".

399 **Myself:** Yeah, so I guess it's kind of like an IDE that suggests different courses of events.

400 **Participant:** Since you're more into the area of interactive fiction than I was, you would
401 expect to have an author using this and building things with this, so that could be a cool
402 thing. If you need another six years of research for your...

403 **Myself:** That's the thing, yeah.

404 **Participant:** Exactly. Believe me, I was in the graduate school for ten years.

405 **Myself:** Oh, wow.

406 **Participant:** Don't do what I did! That was probably a bad call.

407 **Myself:** I'm getting there, but, I only have a few more months to write up, actually.

408 **Participant:** Are you wrapping things up?

409 **Myself:** Yeah.

410 **Participant:** So you have a date to defend and everything already?

411 **Myself:** Almost. We're sending off for the external examiner to get the dates to defend.
412 But it will be before October, hopefully.

413 **Participant:** Good luck.

414 **Myself:** Thanks very much.

G.8 Participant H

Dialogue starts at 12:23.

1 **Myself:** So, that's the explanation of the language. Have you got any questions, initially?

2 **Participant:** Um, yes. Do any of these rules, I guess, phrases, have special meaning, such
3 as, where's it gone? "The Hero dies". Does anything special happen after that?

4 **Myself:** No. It just gives the Hero permission to die. That's literally it. There's nothing
5 really clever going on there.

6 **Participant:** But it doesn't forbid the Hero from doing anything after they're dead?

7 **Myself:** No. It would have been too much work for me to actually give some kind of
8 semantics to all the verbs. That is future work, I guess. This is just a case of compiling a
9 controlled natural language to some kind of institution. So giving it the semantics where,
10 yeah, saying "to die", and then the Hero dies, and having that actually have some kind of
11 consequence, like the Hero not being able to perform in the game, would be difficult. Because,
12 I'd have to do that with all the synonyms of dying as well, and all that kind of stuff. So yeah,
13 that's not part of it.

14 **Participant:** Well, if you're talking about synonyms, there's two things I'd flag up. One
15 is that there's nothing to stop you having keywords, such as "dies".

16 **Myself:** True, true.

17 **Participant:** I don't know if "kills" is also a keyword.

Feature:
Event semantics

18 **Myself:** No, it's not. Actually, the first iteration of the language, I did have key words.
19 But I decided not to do that, I kind of abandoned it. It ended up being too much work.
20 Actually, I think one thing you might be about to suggest is using WordNet, or something
21 like that, and using synonyms through that. That's probably the way forward, because I'm
22 actually using WordNet to get the stems, not the stems, the root form of these verbs, so you
23 can use any verb and it will kind of, you know, "goes" will become "go", "kills" will become
24 "kill". But using WordNet, I could also get the synonyms of those as well. So that might be
25 a way forward. That's why I'm doing this study, is to actually talk about these things, get
26 these ideas. That's something I hadn't really considered before.

27 **Participant:** Sure. The other thing, which is something I'll have to share with... no...
28 I'm not signed into Twitter on this account. Something I can send to you later is a very recent
29 series of tweets by Liz England about how they created ScribbleNauts. Familiar with that
30 game?

31 **Myself:** No.

32 **Participant:** OK, ScribbleNauts is a game where you, a small 2D cartoon character, have
33 a magic pen, which you can write the name of something with, and that thing will appear
34 in front of you and help you to solve your problem. If your problem is that there's a river
35 that you have to get across, you can write "bridge", and you'll have a bridge. But you could
36 also write "crocodile", and run across the back of the crocodile. Or "bucket full of sun", and
37 evaporate all the water.

38 **Myself:** Wow, so how smart is it? Does it work really well?

39 **Participant:** It's interesting. I will send you the full... I haven't actually played it,
40 but the series of tweets was about how that wasn't anything like systematic, or Wordnet or
41 anything clever, that was hand-annotated for huge amounts of content.

42 **Myself:** Wow, OK.

43 **Participant:** You should definitely read the thread.

44 **Myself:** Sure. Cool, that sounds really relevant and interesting. Thank you. OK, so I'll
45 try to remind you to send me that after.

46 **Participant:** Yeah.

47 **Myself:** OK, so back to this. So, going onto the tasks. So yeah, you said you wanted to edit
48 a trope. So take one of these examples and edit it. Some things to bear in mind, actually. So
49 I've limited it to five events long, because once you start combining tropes together, things...
50 the answer set solver basically dies. So I've limited everything to just five events. What else?
51 You'll discover all the problems. Just edit it and see what happens.

52 **Participant:** (uses the tool)

53 **Myself:** So... "die Hero", yeah. So you just added "die Hero" as another possibility.

54 **Participant:** An alternative to finding a sword, killing the Villain, or going to the Land
55 of Adventure.

56 **Myself:** Ah yeah, that's right. So after the "go_hero_home" nodes, you've got the four
57 branches from that. So I think that does... did that do what you expected it to do?

58 **Participant:** Um, once we had talked about the fact that "die" doesn't have any special

59 semantics, yes, that is pretty much what I expected.

60 **Myself:** Because of course he can't meet the Mentor or go home after he dies. OK.

61 **Participant:** He can in this situation, it's just the Mentor will be very disappointed, and
62 so will the people at his home.

63 **Myself:** Oh, OK. OK, cool. So you've edited that. Task 2 is to create a trope from
64 scratch. So if you click on "new", next to "edit". And type in trope name in there. Ah yeah,
65 good idea.

66 **Participant:** Ah.

67 **Myself:** Hmm?

68 **Participant:** OK, so at this point, best to save the trope. So save it, and then click on
69 "edit". And then select your trope. And that way you'll be able to visualise it as you're going
70 along.

71 **Myself:** Betrayal?

72 **Participant:** Yeah, I'm just thinking... "meets the Mentor". Ah, I have "betrayal"
73 twice.

74 **Myself:** Ah, that's interesting. I wonder why that happened. Actually, probably best
75 to delete one of them in case it confuses things. So the reason it says "compile error"...
76 So, whenever there's zero answer sets, it'll just say "compile error", because there's no error
77 message I can really give about that. So in this case, there's zero answer sets because nothing
78 is happening. So set up a first event.

79 **Participant:** You can always say "nothing is happening".

80 **Myself:** That's probably a better message to have. You don't have to use all of those
81 roles and places, etc. Just any of those. Also, try to avoid having any blank lines in there,
82 because that can trip it up. It's very, very brittle, as you would imagine.

83 **Participant:** I had wondered before why there wasn't a distinction between definitions
84 and actions.

85 **Myself:** Yeah, it's something I should get around to adding at some point, but there's
86 lots of things like that. So, as you're typing each line, it's probably best to save and refresh,
87 in case there's a problem. OK, so you're fine.

88 **Participant:** This is the one that I'm expecting to trip it up.

89 **Myself:** Ah, no, it's not going to like that. "The Mentor"... No, that's something I
90 would have to add specifically.

91 **Participant:** Can I try it, or is that unwise?

92 **Myself:** It won't work. Try it, it won't work. It'll probably just say "compile error".
93 What?! OK, I wasn't expecting that. It's even using "be" rather than "is". Hang on.

94 **Participant:** I like that. I'm happy with this.

95 **Myself:** I will settle for that. Wow.

96 **Participant:** (types)

97 **Myself:** Um... oh, you've mistyped "villain" on line 9. Missing an "i". OK. I think you
98 might have reached the length of how many events you can have now. So it's only up to five.
99 But OK, that's proved to be OK. Oh yeah, so you can put in branches. That's actually the

100 next task, so yeah, we'll just go straight onto the next task.

101 **Participant:** That's alright.

102 **Myself:** I think...

103 **Participant:** OK, I've skipped straight through that, but I'm doing it now.

104 **Myself:** This is what everyone's doing. People get carried away, so I'm letting people do
105 whatever they want. That's fine. Um, did you save and then refresh?

106 **Participant:** I may not have hit "save" properly. There we go.

107 **Myself:** OK, cool.

108 **Participant:** Ah. Is... oh yeah, "kill_hero_villain", that's... yeah.

109 **Myself:** But it does sometimes get those word orders wrong. So it can sometimes say
110 "kill_villain_hero" when it's the other way round. But it seems to be OK in this case.

111 **Participant:** Yeah, that makes sense to me as a computer programmer. "kill_hero_
112 villain", it's like a tuple.

113 **Myself:** It's like a prefix notation, really.

114 **Participant:** Yeah, that's right. But I can see that confusing other people.

115 **Myself:** I figured you'd get it. I have had to explain it to other people.

116 **Participant:** Oh actually, if that's the word order, then "be_villain_mentor" is poten-
117 tially the wrong way round.

118 **Myself:** "The Villain"... yes, you're right, yes. So this is a bug that has confused me,
119 because I'm not sure why it's been doing that, and I've been trying... I don't know. I
120 eventually gave up. So I just explain it when it happens.

121 **Participant:** Anyway, I have an object, though it's not one of those. Does it matter that
122 it's not one of those?

123 **Myself:** Ah yeah, it's fine. Don't worry.

124 **Participant:** Ah, I've added another branch.

125 **Myself:** OK, that's cool. So task number five: put two tropes in a story. So, so far you've
126 been in the "edit" tab. Go up... so scroll up and go to the "arrange" tab. So you've got your
127 betrayal trope there. If you click "+", you can add another trope. I would suggest adding a
128 simple one first, because sometimes it takes a while. So example 1 was pretty simple.

129 **Participant:** I can't remember what's in there, but...

130 **Myself:** So I've got the solver running on Mist, but even that - the Mist server at Bath -
131 even that takes a while sometimes to generate these answer sets, because it's using two InstAL
132 institutions and then generating all the possible... so there you go. If you hover your mouse
133 over it and scroll up, it will zoom in. So you can see the blue lines relate to the "betrayal"
134 trope, and the red lines are the "Example 1" trope, and so it's saying like: "if you follow the
135 Example 1 trope, it will take you... you can choose this action, and then it will lead on to
136 this next thing, etc".

137 **Participant:** I got it.

138 **Myself:** So yeah, once you've looked at that, you can try combining it with some other
139 tropes. You could even try a third trope, but I think we could be sitting here for a while. It
140 works, but the... I need to talk to you about optimising the answer set generation, because

141 it's quite slow.

142 **Participant:** Um, do you know the work of Adam Smith?

143 **Myself:** Uh, the name rings a bell, but...

144 **Participant:** He does a lot of answer set programming work for game content, procedural
145 generation.

146 **Myself:** OK.

147 **Participant:** Um, he's also currently running a course at the University of Santa Cruz on
148 answer set programming for games.

149 **Myself:** Hmm, OK.

150 **Participant:** ... which I started following along with, and then I got busy with other
151 things, but have been meaning to finish. Last week I got a paper rejected, which I was pretty
152 much expecting.

153 **Myself:** From who?

154 **Participant:** From the procedural content generation workshop at FDG.

155 **Myself:** Oh, OK, right. That's a shame.

156 **Participant:** But one of the reviewers followed a pattern which I have pretty much
157 conclusively identified as his. And one of the things he pointed to was a few of the recent
158 lectures in his course, cover, what is it? Not focused optimisation. Targeted optimisation for
159 re-grounding. So I can send you those links as well.

160 **Myself:** Yeah, that could be useful. Although it might be too late at this point, because
161 I'm doing this study and then writing up. But it would be good to include in the thesis as
162 ways I could improve this, yeah. For sure. OK, so moving on then. So you've played with
163 that, so... oh yeah, task 6 is to use the trope that you created and embed it inside of another
164 trope. So, your trope was called "betrayal", so if you go back to the "edit" tab, and create
165 a new trope. Some trope which will contain a betrayal somewhere. Because this uses bridge
166 institutions, and it's a bit finicky, at the moment it will only work if your trope happens at
167 the end of the previous trope. Yep.

168 **Participant:** OK. That won't work.

169 **Myself:** What were you trying to do?

170 **Participant:** "Hero has trust issues, Hero goes Home, a betrayal happens, Hero takes
171 object".

172 **Myself:** Ah, OK. No, that won't.

173 **Participant:** So it's something that happens with a betrayal at the end.

174 **Myself:** Yeah. I like how the visualisation has gone over the text box there. I haven't
175 noticed that before, that's pretty cool. Oh wait, what happened?

176 **Participant:** I made the new thing, I tried to switch to the "arrange" tab to get this
177 visualisation to go away, but when I go back to "edit", it's still there.

178 **Myself:** Ah yeah, OK.

179 **Participant:** But if I load "Hero makes an enemy", or...

180 **Myself:** Ah, oh. OK, there's a bug where if it doesn't compile, it won't add it there. So
181 if you go back to "new", if you don't put anything in there, it won't compile properly, so it

Limitation:
Subtrope at
end

182 won't add it, even with that tick mark, which is misleading. The tick mark just means it's
183 uploaded to the server. So I think for it to compile, you need two roles at least. Yeah.

184 **Participant:** Fair enough.

185 **Myself:** Or yeah, two lines, I think, at least of things. And then I think it will compile.

186 **Participant:** Um... is the apostrophe going to mess it up?

187 **Myself:** I would avoid it when you're adding a trope here. But add it in when you're
188 editing it, just to see if it works. So don't have it now, but add it later. I have a feeling I
189 accounted for apostrophes, but change it later. So... oh yeah, so select your... so put it in
190 now and see if it will compile.

191 **Participant:** There we go.

192 **Myself:** So yeah, zero answer sets because there's no event. So, yes, put in... yeah, put
193 in a basic event. "Hero goes to friend's house"... see if that works. I'm not sure if it will.
194 I honestly can't remember if I took that into account. Ah, OK, try that. So save that and
195 refresh. Because I'm not sure if "friend's house" with the apostrophe is going to work. I don'
196 think it will. Get rid of the apostrophe. Hmm. OK, I wonder what's going wrong here. Could
197 you put "the"s in front of everything? Because it should be optional, but sometimes it seems
198 to get tripped up if you don't have "the" in front of certain words.

199 **Participant:** Yes, "the".

200 **Myself:** And then: "The Hero goes to the Friend's House". Yeah, try that. OK. I think
201 you have to "save", then refresh. So, with the "betrayal" trope... 1, 2, 3, 4... OK, it's
202 five events long, so that's the problem. That's why the "betrayal" trope has been cut off. So
203 maybe get rid of one of your events there. I'll have to extend this, I think, five events is too
204 short. Trouble is, though, the answer sets... you saw how long it took to generate answer sets
205 sometimes. So maybe delete line five. Oh, hang on, you have. Oh, "betrayal" is five events
206 long, isn't it? So it's always going to be too long for this. So go back and edit the "betrayal"
207 trope. Uh, yeah. If you... hang on... 1, 2, 3, 4. It should be OK now, so if you try that.
208 Hello?

209 **Participant:** Yeah.

210 **Myself:** Ah cool, so that seems to have worked OK. So, the final task is I don't know, just
211 kind of mess around with it. Try arranging things with the "arrange" tab, try some things
212 that you haven't got the chance to try yet.

213 **Participant:** I'm trying to remember other tropes. (uses tool). Um, so this is one where
214 somebody gets the treasure.

215 **Myself:** OK. Oh, I wonder why that's gone wrong. That seems quite straightforward.

216 **Participant:** I can guess, actually. Oh no, it's basically that one. I was gonna say: where
217 does the root... what's the root of the tree?

218 **Myself:** Uh, yeah, no it should be OK, because it's the same as the other one, as you
219 said. Because it creates a "Start" node as the root.

220 **Participant:** Oh, does it?

221 **Myself:** Yeah. Oh, that's interesting. "The Hero takes the Treasure, Or the Villain takes
222 the Treasure". That should be OK. I wonder why it's not. (reads the trope). Hmm.

Limitation:
Five event
limit

223 **Participant:** (reads the trope)

224 **Myself:** I can't see any obvious problems with that. "Hero goes to the cave"... try
225 deleting the "Or" statement, and see if that's tripping it up. Just, yeah, go line by line. How
226 many spaces do you have there, just two?

227 **Participant:** Two.

228 **Myself:** Yeah, so it should be OK. Hmm, OK. Try deleting that line, and save and refresh.
229 That's weird. "... is a role, Villain is a role".

230 **Participant:** That one works fine.

231 **Myself:** Hmm. I wonder... Ah, I know what it is, yeah. There's a bug that I forgot
232 about. If the trope name is the same as something in the trope, so you've got Treasure as the
233 name of the trope. Ah, so this trips up the ASP because the name of the institution ends up
234 being the same as the name of something in the institution, which it doesn't like.

235 **Participant:** Cool. Ah, OK, so we have that. (types into tool)

236 **Myself:** Uh oh, what was it called?

237 **Participant:** Find the map.

238 **Myself:** If you click on the last blank one, it's probably that, if you want to copy and
239 paste it.

240 **Participant:** Pardon?

241 **Myself:** If you click on one of the blank ones, like the one at the bottom, it's probably
242 that one, so you can copy and paste the text. Yeah, there you go.

243 **Participant:** Why...?

244 **Myself:** I don't know. Right, (reads the trope). Ah yeah, that's it.

245 **Participant:** There we go. "The Hero goes to the Market, The Hero finds the Map".
246 There you go.

247 **Myself:** Nice! Cool.

248 **Participant:** And then...

249 **Myself:** I wonder... try changing "overhears" for something else. It should be OK. What
250 it does is: "overhears" will go into Wordnet, and it'll look up any word that matches that and
251 find the root verb of it. I should imagine "overhears" is in there. But just in case... oh, so
252 it's not that. Oh.

253 **Participant:** Oh no, "betray" was there, it was just taking a long time.

254 **Myself:** Uh, what next?

255 **Participant:** It did show up, momentarily. "betray_hero_spy", good.

256 **Myself:** So "overhears" wasn't there, then? Oh yeah, so arrange them, yeah. Yeah, yeah,
257 wait for it before you add a new one.

258 **Participant:** Yeah, I think that's all I need, because the Hero finds a Map, no the Hero
259 goes to the Market, finds a Map, the Hero's going to the Cave, but they get betrayed by the
260 Spy, who the Villain pays, and then the Villain takes the Treasure, maybe. Ah, that may be
261 too long.

262 **Myself:** Ah, maybe.

263 **Participant:** It's going to be too long, isn't it? "Hero goes to Market, Hero finds the
264 Map, is betrayed and paid"... ah. You can see my screen, right?

265 **Myself:** Yeah, just about. Could you zoom in?

266 **Participant:** Yeah, sorry. So what I was thinking is: "Hero goes to Market, finds the
267 Map, is betrayed, the Villain pays them off (though I can possibly get rid of that one), the
268 Hero goes to the Cave, (which I would expect to come off here), but the Villain takes the
269 Gold".

270 **Myself:** Oh yeah, so "The Villain takes the Gold" is... yeah, yeah.

271 **Participant:** But there's no branch after "pay_villain_spy".

272 **Myself:** Ah, OK. So another thing that it might be is that sometimes it doesn't generate
273 all the nodes that it should, because I limited it to 100, and there are some...

274 **Participant:** I think I can fix that, because I don't need to worry about the thing, and
275 then finding the map, I don't need to worry about going to the market. "The Hero finds the
276 Map"... yeah. So then in "arrange", "find the map", and over here. Yeah, there we go. That
277 top one: "The Hero finds the Map, is betrayed by the Spy, Hero goes to the cave, but the
278 Villain takes the Gold"

279 **Myself:** Oh, OK. Cool. Are there any other interesting paths through that that would
280 make a good story? "Hero finds the Map, goes to the Cave, Then the Spy betrays him".

281 **Participant:** Well yeah, goes to the Cave, takes the Gold, and then the Spy betrays him.
282 Gets to the Cave, the Hero takes the Gold AND the Spy betrays him makes makes a little less
283 sense, maybe.

284 **Myself:** Hmm, yeah. He kind of betrays him by taking the gold, you'd say. But OK.

285 **Participant:** Um, well yeah, this next one is similar except the spy doesn't betray him
286 until after they're at the cave, and then the Villain takes the gold, so that one's cool. And
287 again, that one makes... here the Spy betrays him before he's found the map, even.

288 **Myself:** Ah, right.

289 **Participant:** But that's not the end of the world, it could be like... for some reason all
290 I can think of is Hansel and Gretl, by leaving a trail of breadcrumbs.

291 **Myself:** Ah yeah, I see what you mean.

292 **Participant:** And then the Villain gets the gold. I'm not very kind to my Hero.

293 **Myself:** He just ends up being betrayed all the time. OK, cool. So, that's great. That's
294 everything in the study. So, what do you think of this use of tropes to assemble the story?
295 Do you think that tropes are a suitable component, are they a kind of good abstraction?

296 **Participant:** Yes, I do. I think that just looking at this image... do you want me to
297 capture this image in some way?

298 **Myself:** Only if you want to, I mean I'm recording this whole thing.

299 **Participant:** I can send you that as well, for reference. Um... what was I saying?
300 Yes, looking at this image, one thing that occurs to me is that it might be useful to apply
301 preconditions of some kind. I don't know how feasible that is, but specifically looking at this
302 later one, not allowing the "betray" trope to happen before the Hero has something worth
303 betraying.

304 **Myself:** Yeah, I'd agree that that's a good idea. So I had an extra bit of syntax, which
305 was "when", so you'd say: "When the Hero finds the Map, Then the Spy betrays the Hero".
306 So when some kind of event happens, then something happens. Or when a sequence of events
307 have happened. But of course you could also say "if". So you'd say... I wasn't sure which
308 was the best way to do it. So saying: "When event X, event Y and event Z have happened,
309 then event 2 can happen". Or: "Event A happens if Event B, Event C and Event D have
310 happened". I think it's not important which one... I dunno. But it was all kind of event-
311 based, because there's no kind of sense of state in this at the moment. So you could say "The
312 Hero takes the Sword", and then "the Hero goes to the Land of Adventure if the Hero has the
313 Sword". But, yeah, that's the kind of thing I think would be needed.

314 **Participant:** Yeah. How much state do you track? So here, where I say: "take_villain_
315 gold"...

316 **Myself:** So, it's literally just the events that have happened. So if you say: "take_
317 villain_gold", there's no fluent that appears that says that the Villain has the gold. That's
318 something that I have to add for it.

319 **Participant:** What was the rest of the question?

320 **Myself:** I've forgotten. I think that that's everything that I wanted to say. Have you got
321 any other general comments?

322 **Participant:** Um... the limitation of five events made things a little trickier.

323 **Myself:** Yeah, that's for sure.

324 **Participant:** More so on this screen than the others, really.

325 **Myself:** Yeah. Maybe... I think I'll try extending it to ten events and see how long it'll
326 take to generate those answer sets, because it was taking a really long time, but I kind of
327 changed the constraints I was using, etc, and it might be better now. So that's something I
328 should tweak before the next study.

329 **Participant:** Then just with this information, this line I assume is part of the language?
330 ("_" is a trope where:)

331 **Myself:** Yes, so the way you're editing it now it puts that in for you, yeah. I'm not sure
332 why I decided to do that. But if you were using text files, you'd write it as it is in that example
333 rather than as it is there. You'd actually put that bit at the top as well. And everything else
334 is indented one level.

335 **Participant:** Yeah. Well, if this were just in the same font as the rest of this, then it
336 would be more clear that that was happening.

337 **Myself:** Ah, OK, sure.

338 **Participant:** But that's just so incredibly minor. Yeah, I think the color-coding of the
339 different tropes is very handy. I dunno how you've got "find map" there, instead of "find the
340 map".

341 **Myself:** I just delete "the"s wherever there's a "the", I tell it to ignore it in the parser.

342 **Participant:** Yeah. Nothing else comes to mind at the moment.

343 **Myself:** OK, cool.

Limitation:
Five event
limit

Syntax: Use
of "the"

Bibliography

- Aarne, A. and Thompson, S. (1987). *The types of the folktale: A classification and bibliography*. Suomalainen tiedeakatemia Helsinki.
- Aarseth, E. J. (1997). *Cybertext: perspectives on ergodic literature*. JHU Press.
- Ahn, J., Gobron, S., Garcia, D., Silvestre, Q., Thalmann, D., and Boulic, R. (2012). An NVC emotional model for conversational virtual humans in a 3D chatting environment. In *Articulated Motion and Deformable Objects*, pages 47–57. Springer.
- Armony, J. L., Servan-Schreiber, D., Cohen, J. D., and LeDoux, J. E. (1997). Computational modeling of emotion: Explorations through the anatomy and physiology of fear conditioning. *Trends in Cognitive Sciences*, 1(1):28–34.
- Artikis, A., Sergot, M., and Pitt, J. (2009). Specifying norm-governed computational societies. *ACM Transactions on Computational Logic (TOCL)*, 10(1):1.
- ASD-STE100, A. (2007). Simplified technical english. *ASD standard*. <http://www.asd-ste100.org/> [Accessed: 2017-03-30].
- Bal, M. (2009). *Narratology: Introduction to the theory of narrative*. University of Toronto Press.
- Barthes, R. and Duisit, L. (1975). An introduction to the structural analysis of narrative. *New Literary History*, 6(2):237–272.
- Booker, C. (2004). *The seven basic plots: Why we tell stories*. A&C Black.
- Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007). *Programming multi-agent systems in AgentSpeak using Jason*. John Wiley & Sons.
- Campbell, J. (2008). *The hero with a thousand faces*, volume 17. New World Library.
- Cardoso, H. L. and Oliveira, E. (2007). Institutional reality and norms: Specifying and monitoring agent organizations. *International Journal of Cooperative Information Systems*, 16(01):67–95.
- Cavazza, M., Charles, F., and Mead, S. J. (2002). Character-based interactive storytelling. *IEEE Intelligent systems*.

- Chomsky, N. and Halle, M. (1968). The sound pattern of english.
- Clarke, V. and Braun, V. (2014). Thematic analysis. In *Encyclopedia of critical psychology*, pages 1947–1952. Springer.
- Cliffe, O., De Vos, M., and Padget, J. (2007). Specifying and reasoning about multiple institutions. In Vazquez-Salceda, J. and Noriega, P., editors, *COIN 2006*, volume 4386 of *Lecture Notes in Computer Science*, pages 63–81. Springer. ISBN: 978-3-540-74457-3. Available via http://dx.doi.org/10.1007/978-3-540-74459-7_5.
- Clocksink, W. F. and Mellish, C. S. (2003). *Programming in PROLOG*. Springer Science & Business Media.
- Crawford, C. (2012). *Chris Crawford on interactive storytelling*. New Riders.
- Ekman, P. (1992). An argument for basic emotions. *Cognition & emotion*, 6(3-4):169–200.
- Engelberg, M. (2017). Instaparse, a clojure parser generator library. <https://github.com/Engelberg/instaparse> [Accessed: 2017-07-11].
- Evans, R. (2010). Introducing exclusion logic as a deontic logic. In *Deontic Logic in Computer Science*, pages 179–195. Springer.
- Evans, R. and Short, E. (2014). Versu—a simulationist storytelling system. *Computational Intelligence and AI in Games, IEEE Transactions on*, 6(2):113–130.
- Fornara, N., Viganò, F., and Colombetti, M. (2007). Agent communication and artificial institutions. *Autonomous Agents and Multi-Agent Systems*, 14(2):121–142.
- Friedman, M. (1992). Regulative and constitutive. *The Southern Journal of Philosophy*, 30(S1):73–102.
- Fuchs, N. (2017). ACE In a Nutshell, a short overview of the ace language. http://attempto.ifi.uzh.ch/site/docs/ace_nutshell.html [Accessed: 2017-03-31].
- Fuchs, N. E. and Schwitter, R. (1996). Attempto controlled english (ACE). *arXiv preprint cmp-lg/9603003*.
- Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., and Schneider, M. (2011). Potassco: The Potsdam answer set solving collection. *AI Communications*, 24(2):107–124.
- Gervás, P., Díaz-Agudo, B., Peinado, F., and Hervás, R. (2005). Story plot generation based on cbr. *Knowledge-Based Systems*, 18(4):235–242.
- Giunchiglia, E. et al. (2001). Causal laws and multi-valued fluents. In *In Proceedings of Workshop on Nonmonotonic Reasoning, Action and Change (NRAC)*. Citeseer.

- Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., and Turner, H. (2004). Nonmonotonic causal theories. *Artificial Intelligence*, 153(1-2):49–104.
- Grasbon, D. and Braun, N. (2001). A morphological approach to interactive storytelling. In *Proc. CAST01, Living in Mixed Realities. Special issue of Netzspannung. org/journal, the Magazine for Media Production and Inter-media Research*, pages 337–340. Citeseer.
- Gratch, J. and Marsella, S. (2004). A domain-independent framework for modeling emotion. *Cognitive Systems Research*, 5(4):269–306.
- Halliwell, S. (1986). *Aristotle’s poetics*. University of Chicago Press.
- Harris, J. (2017). The Periodic Table of Storytelling a visualisation of story tropes. <http://jamesharris.design/periodic/> [Accessed: 2017-02-15].
- Harris, W. F. (1959). *The basic patterns of plot*. University of Oklahoma Press.
- Hartmann, K., Hartmann, S., and Feustel, M. (2005). Motif definition and classification to structure non-linear plots and to control the narrative flow in interactive dramas. In *Virtual Storytelling. Using Virtual Reality Technologies for Storytelling*, pages 158–167. Springer.
- Hayes, N. (2000). *Doing psychological research*. Taylor & Francis Group Abingdon.
- Hickey, R. (2017a). Clojure, a lisp for the java virtual machine. <https://clojure.org/> [Accessed: 2017-07-11].
- Hickey, R. (2017b). ClojureScript, a compiler for clojure that targets javascript. <https://clojurescript.org/> [Accessed: 2017-07-11].
- Hodas, J. S. and Miller, D. (1991). Logic programming in a fragment of intuitionistic linear logic. In *Logic in Computer Science, 1991. LICS’91., Proceedings of Sixth Annual IEEE Symposium on*, pages 32–42. IEEE.
- Kambhampati, S., Knoblock, C. A., and Yang, Q. (1995). Planning as refinement search: A unified framework for evaluating design tradeoffs in partial-order planning. *Artificial Intelligence*, 76(1):167–238.
- Kowalski, R. and Sergot, M. (1986). A logic-based calculus of events. *New generation computing*, 4(1):67–95.
- Lang, R. (1999). A declarative model for simple narratives. In *Proceedings of the AAAI fall symposium on narrative intelligence*, pages 134–141.
- Lee, J., Baines, V., and Padget, J. (2013). Decoupling cognitive agents and virtual environments. In Dignum, F., Brom, C., Hindriks, K., Beer, M., and Richards, D., editors, *Cognitive Agents for Virtual Environments*, volume 7764 of *Lecture Notes in Computer Science*, pages 17–36. Springer Berlin Heidelberg.

- Lehnert, W. G. (1981). Plot units and narrative summarization. *Cognitive Science*, 5(4):293–331.
- Lévi-Strauss, C. (1963). *Structural anthropology*, volume 1. Basic Books.
- Li, T. (2014). *Normative Conflict Detection and Resolution in Cooperating Institutions*. PhD thesis, University of Bath.
- Liu, H. and Singh, P. (2004). Conceptnet—a practical commonsense reasoning tool-kit. *BT technology journal*, 22(4):211–226.
- Martens, C. (2015). Ceptre: A language for modeling generative interactive systems. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*.
- Mateas, M. (1999). *An Oz-centric review of interactive drama and believable agents*. Springer.
- Mateas, M. and Stern, A. (2003). Façade: An experiment in building a fully-realized interactive drama. In *Game Developers Conference*, volume 2.
- McKee, R. (1997). *Substance, Structure, Style, and the Principles of Screenwriting*. New York: HarperCollins.
- Meehan, J. R. (1977). Tale-spin, an interactive program that writes stories. In *IJCAI*, volume 77, pages 91–98.
- Mehrabian, A. and Russell, J. A. (1974). *An approach to environmental psychology*. the MIT Press.
- Miller, G. A. (1995). Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.
- Noriega, P. (1999). *Agent mediated auctions: the fishmarket metaphor*. Institut d’Investigació en Intel·ligència Artificial.
- Ogden, C. K. (1944). *Basic English: A general introduction with rules and grammar*. Number 29. K. Paul, Trench, Trubner.
- Pemberton, L. (1989). A modular approach to story generation. In *Proceedings of the fourth conference on European chapter of the Association for Computational Linguistics*, pages 217–224. Association for Computational Linguistics.
- Polti, G. (1921). *The thirty-six dramatic situations*. JK Reeve.
- Propp, V. (1968). Morphology of the folktale. 1928. *Trans. Svatava Pirkova-Jakobson*. 2nd ed. Austin: U of Texas P.
- Reed, A. (2010). *Creating Interactive Fiction with Inform 7*. Cengage Learning.

- Riedl, M., Saretto, C. J., and Young, R. M. (2003). Managing interaction between users and agents in a multi-agent storytelling environment. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 741–748. ACM.
- Riedl, M. O. and Young, R. M. (2010). Narrative planning: balancing plot and character. *Journal of Artificial Intelligence Research*, 39(1):217–268.
- Rumelhart, D. E. (1975). Notes on a schema for stories. *Representation and understanding: Studies in cognitive science*, 211:236.
- Russell, J. A. (1980). A circumplex model of affect. *Journal of personality and social psychology*, 39(6):1161.
- Schank, R. C. (1990). *Tell me a story: A new look at real and artificial memory*. Charles Scribner’s Sons.
- Schweizer, L. (2017). Logical modeling using answer set programming. <https://iccl.inf.tu-dresden.de/w/images/0/04/LM-ASP-2017.pdf> [Accessed: 2018-05-15].
- Shanahan, M. (1999). The event calculus explained. In *Artificial intelligence today*, pages 409–430. Springer.
- Shklovsky, V. (1991). *Theory of prose*. Dalkey Archive Press.
- Thomas, J. M. and Young, R. M. (2006). Author in the loop: Using mixed-initiative planning to improve interactive narrative. *ICAPS 2006*, page 21.
- Thompson, M. (2017). TropICAL, a domain-specific language for tropes. <https://github.com/cblop/tropic> [Accessed: 2018-02-28].
- Thompson, M., Battle, S., and Padget, J. (2015a). Telling non-linear stories with interval temporal logic. In *ICIDS*, pages 370–373.
- Thompson, M., Padget, J., and Battle, S. (2015b). An interactive, generative punch and judy show using institutions, asp and emotional agents. In *Proceedings of the 2015 International Conference on Coordination, Organizations, Institutions, and Norms in Agent Systems XI*, pages 396–417. Springer.
- Thompson, M., Padget, J., and Satoh, K. (2016). Describing legal policies as story tropes in normative systems. In *Legal Knowledge and Information Systems - JURIX 2016: The Twenty-Ninth Annual Conference*, pages 207–210.
- Turner, S. R. (1993). *Minstrel: a computer model of creativity and storytelling*. PhD thesis, University of California at Los Angeles.
- Turner, S. R. and Dyer, M. G. (1986). *Thematic knowledge, episodic memory and analogy in MINSTREL, a story invention system*. University of California, Computer Science Department.

- TV Tropes (2017a). TV Tropes an online wiki for media tropes. <http://tvtropes.org> [Accessed: 2017-02-15].
- TV Tropes (2017b). TV Tropes: Playing with a trope. <http://tvtropes.org/pmwiki/pmwiki.php/Main/PlayingWithATrope> [Accessed: 2017-02-15].
- TV Tropes (2017c). TV Tropes: Propp's functions of folktales. <http://tvtropes.org/pmwiki/pmwiki.php/Theatre/PunchAndJudy> [Accessed: 2017-09-18].
- TV Tropes (2017d). TV Tropes: Propp's functions of folktales. <http://tvtropes.org/pmwiki/pmwiki.php/Main/ProppsFunctionsOfFolktales> [Accessed: 2017-02-15].
- Von Wright, G. H. (1951). Deontic logic. *Mind*, 60(237):1–15.
- Vonnegut, K. (2009). *Palm Sunday: an autobiographical collage*. Dial Press.
- Young, R. M. (1999). Notes on the use of plan structures in the creation of interactive plot. In *AAAI Fall Symposium on Narrative Intelligence*, pages 164–167.
- Young, R. M., Pollack, M. E., and Moore, J. D. (1994). Decomposition and causality in partial-order planning. In *AIPS*, pages 188–194.
- Young, R. M., Riedl, M. O., Branly, M., Jhala, A., Martin, R., and Saretto, C. (2004). An architecture for integrating plan-based behavior generation with interactive game environments. *Journal of Game Development*, 1(1):51–70.