GENERATION AND AUTHORING OF BELIEF-BASED ACTION FAILURE IN NARRATIVE PLANNING

by

Rushit Sanghrajka

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computing

School of Computing

The University of Utah

December 2023

The University of Utah Graduate School

STATEMENT OF DISSERTATION APPROVAL

The dissertation of **Rushit Sanghrajka**
has been approved by the following supervisory committee members:

Robert Michael Young,	Chair(s)	**11/08/2023** Date Approved
Rogelio E. Cardona-Rivera,	Member	**11/01/2023** Date Approved
Tucker Hermans,	Member	**11/01/2023** Date Approved
Corrinne R. Lewis,	Member	**11/08/2023** Date Approved
Chris Martens,	Member	**11/02/2023** Date Approved
Ellen Riloff,	Member	**11/09/2023** Date Approved

by **Mary W. Hall**, Chair/Dean of
the Department/College/School of **Computing**
and by **Darryl P. Butt**, Dean of The Graduate School.

ABSTRACT

Narrative planning techniques have shown great potential in their ability to automatically generate plots that meet human authorial goals for their fictional world and characters. However, stories produced by existing techniques are limited in expressivity related to belief-based failed actions in stories, which is pervasive in various media ranging from *Star Wars* to *Looney Tunes*.

This dissertation outlines a principled means to generate storylines with failed actions and advance a broader goal of automatically creating more expressive, natural, and compelling narratives. It builds upon existing work in narrative generation using artificial intelligence planning algorithms. While traditional planning techniques rely on producing sound, causally coherent plans that avoid or recover from failure, our approach explicitly plans for action failure in the generated plans.

The approach described by this dissertation involves (1) defining a knowledge representation and an associated plan generation algorithm for intentionally generating stories where characters attempt an action and fail and (2) extending current methods for the specification of narrative planning problems included in this expanded representation involving desired actions, action failures, and the intent dynamics that surround them. We evaluated our approach to gauge human comprehension of the computationally generated plots and the capability of novice storywriters to use the enhanced representation to guide the planner towards desired stories consisting of failed actions.

For the love of learning, and the journey worth pursuing.

CONTENTS

LIST OF TABLES

ACKNOWLEDGEMENTS

I consider myself exceptionally fortunate to have had the opportunity to pursue a Ph.D., and I acknowledge that both luck and privilege played pivotal roles in guiding me towards a graduate degree in a field that ignites my passion. Gratitude fills my heart as I extend thanks to the diverse entities without whom this profound journey would not have unfolded.

I would like to commence by expressing heartfelt appreciation for my advisor and mentor, R. Michael Young. I vividly recall the awe-inspiring moment when your response prompted me to board a plane to Salt Lake City the very next day. Taking that leap of faith marked the beginning of a hero's journey of sorts, and I am honored to be a *Youngling*, guided by your mentorship.

I also wish to express gratitude to my esteemed committee members, each contributing uniquely to my path. Dr. Ellen Riloff, your contagious enthusiasm for advancements in natural language processing illuminated our meetings, and your insights were eagerly anticipated. Corrinne Lewis, your unwavering support and encouragement, coupled with the valuable perspective of an author and one who really cares about stories, enriched my research. Dr. Tucker Hermans, your mentorship in acclimating to the life of a Ph.D. student and profound insights into the core aspects of artificial intelligence were invaluable. Dr. Chris Martens, your guidance on framing devices and your illuminating insights into the user study significantly enhanced my dissertation. Dr. Rogelio Cardona-Rivera, your profound insights and engaging discussions have been integral to my journey from the outset.

I would like to thank supportive individuals at Rutgers and University of Utah for helping me. Dr. Mubbasir Kapadia, you introduced me to the domain of computational models of narrative. You saw something in me and challenged me to pursue it further, mentoring and guiding me as I took my first steps into the world of research and using computers to help authors. I am forever grateful to you for your guidance and mentorship.

To my graduate advisors at the University of Utah, including Robert Barber, Leslie Wallwork, Lauren Down, Jill Wilson, Allen Hill, and Sydnee Sartor, your guidance through administrative complexities was indispensable. I would also like to thank my therapist Josh Newbury at the University of Utah Counseling Center. Your impact on me has been crucial: I am proud of who I am, and thank you for helping me become this person.

I would like to thank my colleagues and friends who directly contributed in the research presented in this dissertation. Shreya Kotadia, your contribution to the front end of the plot construction tool and unwavering support were indispensable. To all friends and colleagues who participated in the pilot study, including Prapti, Michael Gardone, Michael Clemens, Shubham, Nancy, and Daniel, your crucial feedback refined both the tool and the user interface.

My colleagues in the Liquid Narrative/QED lab: thank you for always being around to support me: David Winer, Eric Lang, Michael Gardone, and, Nancy Blackburn. David, thank you for showing me the ropes when I started out, and have always been there to help me out. Michael Clemens, thank you for supporting me as a friend and colleague: your positivity was a force that really pushed me to continue when I needed it. My colleagues in academia: Stephen Ware, Rachel Farrell, Chris Martens, Sasha Azad, Brent Harrison, and many others. You all help me enjoy academia and I look to all of you for inspiration and respect you all for your ideas and contributions.

In the midst of the occasional burdens of research, teaching consistently emerged as a perpetual source of joy throughout the course of my graduate life. I would like to thank all my teaching mentors over the years. Dr. Erin Parker, your mentorship stands as a monumental force in my teaching career. The impact of your guidance is enduring, as I carry with me the profound lessons learned under your tutelage. Your support in navigating the intricacies of running a class has been transformative. Dr. Mary Emenike, thank you for showing up to the study group one day and helping me learn the ropes on running a study group- your mentorship during the Learning Assistant program shaped my journey to being an educator. In addition to these cherished mentors, I acknowledge the numerous professors and teachers who have served as inspiring role models. Ellen Riloff, Tucker Hermans, Rogelio Cardona-Rivera, Corrinne Lewis, Mubbasir Kapadia, David Cash, Alexander Schliep, Dina Vira, Vipul Shah, Seema Lamba, Neelam, and my very first teacher, Ishrat — each of you has contributed to the mosaic of influences shaping my understanding of education and teaching. The collective impact of your wisdom has been instrumental in defining my approach to education and has left an enduring imprint on my journey.

I have been truly fortunate to be surrounded by a multitude of vibrant social circles, each brimming with friends who embody qualities of unwavering support, genuine curiosity, and steadfast reliability, all of whom have been cheering me on throughout my journey. To Yash and Mallika, your steadfast presence has been a modern-day testament to what true friendship means—always there, unwavering, and filled with genuine warmth. Ronish and Rujul, you have been pillars of joy in my life, bringing light and laughter into every shared moment. Hena, Miti, and Yash Shah, your friendship has been a harmonious blend of companionship and thoughtful guidance, serving as beacons of reason during moments of reflection. Hong, thank you for your kinship as we both sailed the waters of graduate life together. I express gratitude to the friends in Salt Lake City— Hong, Trisha, Daniel, Tanvi, Sumanth, Yuan, Jermy, Manish, Ilya, and the many others. I want to acknowledge the reliability and support you've offered. Your presence has been a source of strength, and I am truly grateful for the comfort of having you all to rely on. To my school siblings from AVM, the family-like friends from JJA, the food enthusiasts from VJTI, the friends/students from Rutgers, the Rudravengers (special shoutout to Alex), and the housemates from my time at Disney, among many others, I want to express a heartfelt acknowledgment. Each of you is truly awesome, and your impact on my life is immeasurable.

I extend heartfelt thanks to my family, including my loyal companion Joey. Joey, thank you for showering me with boundless love. Prapti, thank you for being there and being kind, patient, and supportive. Your unwavering love and understanding created a sanctuary of support during the highs and lows of this scholarly expedition. Thank you to my parents, my two grandmothers, my aunt and my extended family: you have all been a source of strength that has propelled me forward during moments of uncertainty. To each family member, your presence in my life has been a source of solace and inspiration, and I am grateful for the shared journey we've navigated together.

Lastly, I want to express my gratitude to Salt Lake City, UT, a city that became more than a backdrop to my academic endeavors. The numerous cafes and libraries scattered throughout this inspiring city provided not only a physical space but also a sanctuary for intellectual exploration and dissertation crafting. In this environment, I found the support I needed to foster a productive journey.

CHAPTER 1

INTRODUCTION

Stories are everywhere: each of us tells and experiences stories every day. In addition to entertaining or conveying information, stories are a method by which essential values and traditions are communicated to others, including the next generation [53, 56, 111]. Storytelling is powerful: the art and craft of constructing powerful stories allows the story to convey its message to the audience, whether for entertainment, knowledge, or any of the many goals for telling stories.

Stories often include failure on purpose: stories that describe experiences where one learns from mistakes share inspiration with audiences. For example, consider the experience shared by medical professionals about their mistakes and how they became learning experiences for their staff [111]. Stories can also use failure for climactic moments to create tension and build emotional moments for audiences. For example, consider the moment in *Avengers: Endgame* [80], when Thanos the Mad Titan attempts to remove half of life in the universe by snapping his fingers. He (along with the audience) is surprised when his finger snap has no effect: he realizes too late that the Infinity Stones, which he had assumed were in place along the back of the gauntlet he's wearing, were missing, removing the gauntlet's powers. These are stories where failed actions are integral to the experience and leave an impact on the audience.

Interactive storytelling systems tell stories where the audience (a player) gets to experience the narrative while having some agency in the narrative [16]. Artificial intelligence systems have significantly advanced in telling engaging stories through interactive film and TV [49] and visual novels and videogames. The field has grown and produced tools that allow authors to craft experiences in interactive fiction [46, 48, 81]. Those authoring tools run the gamut of expressivity and support for authorial intent. On one side of the spectrum, authors craft branches of their stories and then use the tools to create experiences for audiences. On the other side of this spectrum, authoring of every possible story branch can quickly get complex: *Baldur's Gate 3*, for instance, has 17,000 endings [102, 103]. While there may be minor variations between some of these endings, managing complex branches during authoring and keeping track of the narrative experience for the audience is an exciting challenge to address.

Computational modeling of narratives is a growing and significant body of work that aims to produce stories with the help of computers. One major push in computational models of narrative is narrative planning [3]. Narrative planning approaches are goal-directed approaches extending classical planning algorithms to generate world simulations and produce causally coherent plans that meet character or authorial goals. The field has evolved to develop systems and algorithms that produce plans with features intrinsic to stories and have contributed to interactive narratives and computer-assisted authoring of stories [59, 78]. These systems aim to assist in the authoring process for plot construction– constructing plots based on specifications by the author, and can often support multiple stories in the same narrative space.

Computational approaches in narrative generation have representational limitations that impact the expressive range of these systems [3]. For example, one significant limitation is that most story generation systems cannot produce stories with failures. In this case, this limitation arises from the beneficial property of *soundness* – the guarantee that every plan produced by a planner will be executable – held by most planning systems. In many of the contexts where planning systems might find use, soundness is a key feature. In narrative generation systems, however, action failure within stories is common and desirable (at least by authors).

This dissertation considers computational modeling of narratives in the context of failed actions specifically: it considers the space of stories where characters like Thanos can fail when they attempt actions because their beliefs contradict the actual world conditions required to perform an action successfully. This dissertation considers how computational narrative generation approaches can create plots with such properties and still maintain the qualities that make it possible for people to track and comprehend such plots accurately. We also consider how authors can use the algorithms to express constraints regarding failed actions: how they can direct the algorithms to generate stories with these properties.

1.1 Motivation

Consider a story based in a *Star Wars*-like fictional world with two spaceships: one belonging to the Empire Troopers and one belonging to the Rebel fighters. The Rebel fighters want to escape the Empire Troopers by jumping through hyperspace to another part of the galaxy. In contrast, the Empire troopers want to capture the Rebel fighters by shooting their laser cannons at the Rebel ship and disabling it from jumping into hyperspace. This simplified story world has the potential for many different stories with the conflicting goals of the two teams. One possible story that can be produced is similar to a scene in *The Empire Strikes Back [55]* when Han Solo is racing to escape the Imperial fleet attacking Hoth. In the story, his goal aligns with the Rebel fighter, and the Imperial fleet's goals are similar to the Empire troopers. In the scene, Han pulls a lever to make the jump to hyperspace, but an equipment malfunction in the *Millennium Falcon* spaceship's hyperdrive causes the action to fail both immediately and dangerously.

In examples like this, the failures of character actions aren't simply emergent properties of a complex environment and the limitations of agents operating within it. Rather, the attempts and failures are designed intentionally by authors for narrative effect, e.g., to build tension, to prolong efforts around goal achievement, or to highlight to a reader the disparities of knowledge and ability between characters within the unfolding story world. These roles and others played by action failure within stories are central to many narrative functions. Consequently, developing principled means to generate storylines with failed actions advances the broader goal of automatically creating more natural and compelling narratives.

Authoring such a story without computational tools requires authors to consider aspects of the world that go beyond more straightforward reasoning about cause and effect in the story world. Specifically, authors crafting stories where characters attempt actions that fail must also reason about character beliefs in addition to the state of the world. Stories containing actions that fail due to characters' mistaken beliefs require additional factors to consider compared to stories that do not involve failed actions. This makes creating stories cumbersome and could possibly introduce inconsistencies, leaving audiences confused.

In recent years, work on automated story generation has shown success in developing planning-based generative methods (e.g., [23, 69, 113]). Planning-based methods for story generation offer several attractive features, including guarantees of soundness and completeness and the natural representational fit between plan structures and the goal-directed activity that characters undertake inside narratives. However, researchers have increasingly identified limited expressive capabilities in previously developed plan representations when used to characterize storyline structure. Knowledge representations adequate to produce plans that control robot execution fall short in their characterization of a range of features commonly found in stories. Much work that has gone into plan-based story generation (e.g., [6, 97, 108]) has sought to retain as many of the benefits of classical planning as possible while also increasing the expressive range of narrative generators.

One limitation of planning approaches for storyline creation arises from their inability to generate plans containing actions that fail, which is common in stories [61]. If we were to model the *Star Wars* example described above, existing computational approaches would be unable to generate a story where Han's false beliefs about the condition of the hyperspace drive of the *Milennium Falcon* cause him to fail when attempting to jump to hyperspace. As planning algorithms continue to improve to build more sophisticated stories, there is a need for them to also model stories that consist of failed actions.

Human control over the artificial intelligence technology advances in narrative generation is, at present, limited. There is a need for human authors to interact with the technologies to produce narrative content effectively. As automated planning approaches increase their expressive range and produce stories with complex properties, such as ones where characters reason about beliefs and intention [8], there is a greater need for (a) expansion to the affordances available to human authors to provide specifications to the system, and (b) understanding how human authors can work with these newer systems to produce stories that they desire.

1.2 Problem

The central problem this dissertation focuses on is how a planning system can generate plots with the properties described above. Typical AI planning approaches ensure that plans are sound and focus on satisfying goals in the world. Narratives, however, often explicitly include failed actions for the reasons described above. We address this challenge by distilling the task of using narrative planning to create stories consisting of failed actions into two subproblems.

The **first** problem is the ability of narrative planning approaches to produce plans with belief-based failed actions. To our knowledge, no other narrative planning system is capable of producing stories with these properties, and we consider how we can expand on the expressive range of narrative planning systems by creating stories where characters fail when attempting actions. Moreover, this problem also considers the comprehension aspect of produced stories: it is important for humans experiencing these stories to understand the belief and intent dynamics surrounding the computationally generated stories.

The **second** problem is the ability of authors to work with narrative planning systems to produce stories with desired properties surrounding failed actions effectively. Current narrative planning approaches are limited in the amount of expressivity for authors to specify constraints about the trajectory of the desired story: typical approaches are limited to considering the beginning and ending, but not the specific actions. We focus on the task of adding expressivity available to authors to provide more specifications to narrative planning systems: constraints surrounding actions, failed actions, and intent dynamics for action failure. Moreover, we also consider the impact of the added expressivity on the authoring process when working with a narrative planning system, to gain an understanding of how useful enhanced authorial expressivity is for producing stories with failed actions.

1.3 Approach

The work we describe here seeks to outline a principled means to generate storylines with failed actions and advance a broader goal of automatically creating more expressive, natural, and compelling narratives. We develop narrative planning techniques to support the generation of narratives where characters can attempt actions that potentially fail due to incorrect character beliefs, and support authorial capability to generate these narratives. Moreover, in order for the generated stories with action failures to be believable, the developed planning techniques should reason about the mental attitudes of characters to support the narrative structures around action failure. Specifically, to maintain narrative believability, characters can have mistaken beliefs about the world, observe the world as it changes, and update their beliefs accordingly. Characters must also maintain a set of commitments to action gained in their beliefs, so they can respond to action failure in believable ways. Finally, authorial control of narrative planning technologies must also be extended to support constraints surrounding these properties. The approach described by this dissertation involves (1) defining a knowledge representation and an associated plan generation algorithm for intentionally generating stories where characters attempt an action and fail, and (2) extending current methods for the specification of narrative planning problems included in this expanded representation involving desired actions, action failures and the intent dynamics that surround them.

We provide the design of an algorithm for story generation that explicitly plans for character actions that fail. The algorithm uses a knowledge representation that provides context for this failure based on the limitations of characters' beliefs about the story world around them (e.g., Han's false belief that the *Millennium Falcon's* hyperdrive was operational and could make the jump to lightspeed). The algorithm is capable of considering authorial goals (e.g., the story to still end with Han Solo escaping from the Empire). The stories produced also consider character intent dynamics (e.g., Han will only perform actions if he believes they will get him closer to his goal of jumping to hyperspace successfully) and observability (e.g., only characters present in the cockpit of the *Millennium Falcon* observe the effects of Han's failed attempt). Moreover, authors using the planner can specify constraints related to the failed actions (e.g., only produce stories where Han fails to jump to hyperspace) or intent dynamics surrounding failed actions (e.g., only produce stories where Han gives up on using the hyperspace engine upon failing). We call the pairing of the algorithm and its representation defined here HEADSPACE. The HEADSPACE algorithm produces story structure with many advantageous properties found in other plan-based approaches and is more parsimonious than previous approaches to story generation that also address character belief dynamics.

The following thesis captures our approach:

A planning system that utilizes character belief and intent dynamics to generate stories with failed actions can empower human authors to create plots with these distinctive qualities effectively.

The dissertation addresses the subproblems described via the following research questions:

Research Question 1. *How can a narrative planner be used to generate stories with characters exhibiting expressive behavior surrounding failed actions?*

Research Question 2. *How does adding expressivity for the specification of authorial constraints surrounding desired actions, failed actions, and intent dynamics surrounding failed actions impact the ability to create stories using a computational narrative planner?*

The first research question considers how automated planning algorithms can be used to generate stories consisting of failed actions. We propose a novel algorithm and representation to reason about character beliefs and then intentionally generate stories with failed actions. We then evaluate this research question by considering the accuracy with which humans can comprehend these automatically generated stories. Using the algorithm, we generated a few stories that consisted of failed actions. We then used a cognitive model of question answering to gauge how well readers could understand the belief and intent dynamics surrounding the failed actions in the stories.

The second research question addresses how the added expressivity impacts authors' ability to use automated planning tools: does providing greater control over story structure benefit the authorial process? We propose a novel representation for planning problems to capture constraints surrounding failed actions and intents, and demonstrate its ability to guide the planner to create stories with failed actions. The evaluation for this research question uses an ablative study design to compare authorial effort for different levels of expressivity for the creating of plots with failed actions. The results show that the additional expressivity reduces authorial effort required to use a narrative planner to produce stories with failed actions.

1.4 Contributions

This dissertation focuses on extending narrative planning approaches to support the specification of authorial constraints surrounding failed actions, and for generating plot with failed actions as output of the narrative planning algorithms.

This dissertation makes the following contributions:

- the ability to intentionally generate stories with failed actions: we extend narrative planning approaches to create stories where characters *by design* fail when attempting an action due to incorrect beliefs about the world [86].

- the ability to generate stories with specific intention dynamics surrounding failed actions: we draw on existing notions of intent in computational narratives [76] to determine how character intent may change when a character attempts a failed action, and how the change in intent could influence agents' response and plan of action in the world, leading to nuanced behavior.

- the ability to support authors as they specify constraints surrounding failed actions: we build the capability for authors to direct planning algorithms that reason about intent dynamics surrounding failed actions to specifically target authorial constraints surrounding the failed action and its consequences. In our approach, authors can specify whether a desired failed action in the resulting plan should lead to plan repair, a different intention plan, or dropping of the intent for the character attempting the failed action.

- the extension of existing evaluation methods that assess whether readers recognize and comprehend plot generated by the proposed approach accurately, to generate models from plans consisting of failed actions [83] and the development of evaluation methods for stories consisting of failed actions [85].

- an increased understanding of human authors' engagement with constraint-focused authoring tools used to drive automated narrative planners: an increase in our understanding of the requirements for authors to direct story generators by using an ablative study to evaluate challenges for authors with different levels of input (i.e., story structure in addition to domain structure).

1.5 Reader's Guide

Figure 1.1 provides an overview of the approach described in this dissertation. The organization of the remainder of this dissertation is as follows. Chapter 2 summarizes the related work in planning, narrative generation, cognitive models supporting human comprehension of stories, and authorial support tools.

In Chapter 3, we describe our contributions to generating and authoring narratives consisting of belief-based failed actions. First, we focus on the representation and algorithm for the generation of such narratives with the focus on meeting authorial goals in Section 3.1 (boxes 1a and 1b in Fig 1.1). Second, we describe the approach in the context of intention and observability dynamics, with an enriched representation and updated algorithm, which considers character intent dynamics alongside authorial goals (box 2b in Fig 1.1). This is described in Section 3.2. Third, Section 3.3 describes additions to authorial expressivity to allow for specification for constraints related to actions, failed actions, and character intent surrounding failed actions (boxes 3a and 3b in Fig 1.1). Finally, Section 3.4 concludes the chapter by proposing extensions to work in creating QUEST representations from narrative plans to support narrative plans that consist of belief-based failed actions.

Chapter 4 describes the evaluations conducted to evaluate the contributions of this dissertation. Section 4.1 provides a brief analytic evaluation discussing the claim of expressive range (box 1c in Fig 1.1). Section 4.2 describes an empirical evaluation that explores Research Question 1 through an evaluation of comprehension of the stories produced by HEADSPACE (box 2c in Fig 1.1). Section 4.3 describes a second experiment that explores Research Question 2, studying the influence of the enhanced authorial expressivity on producing stories with failed actions using the HEADSPACE planner through a user interface (boxes 4b and 3c in Fig 1.1).

Chapter 5 concludes the dissertation, summarizing the contributions and discussing future work.

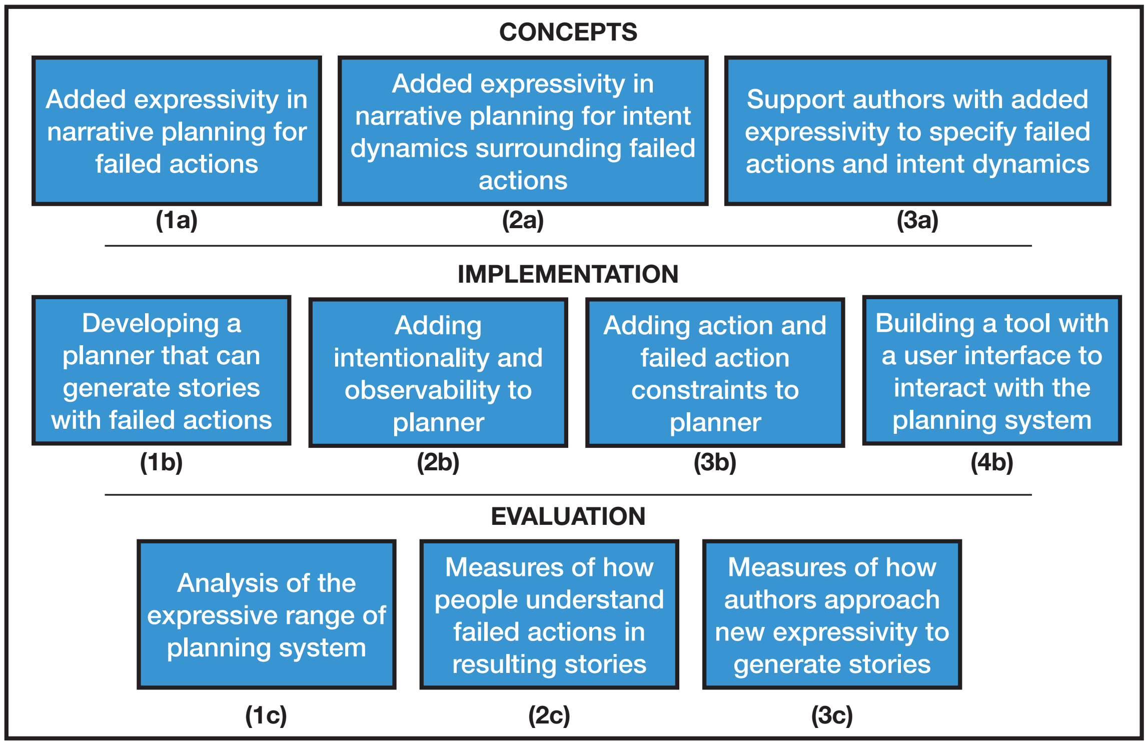

CONCEPTS
Added expressivity in narrative planning for failed actions
(1a)
Added expressivity in narrative planning for intent dynamics surrounding failed actions
(2a)
Support authors with added expressivity to specify failed actions and intent dynamics
(3a)
IMPLEMENTATION
Developing a planner that can generate stories with failed actions
(1b)
Adding intentionality and observability to planner
(2b)
Adding action and failed action constraints to planner
(3b)
Building a tool with a user interface to interact with the planning system
(4b)
EVALUATION
Analysis of the expressive range of planning system
(1c)
Measures of how people understand failed actions in resulting stories
(2c)
Measures of how authors approach new expressivity to generate stories
(3c)

Figure 1.1. Dissertation overview.

CHAPTER 2

RELATED WORK

This section provides an overview of the various domains related to the dissertation work. There are, broadly, four research areas that serve as background to this work. Section 2.1 introduces the reader to AI planning algorithms. The section provides a brief run-through of planning algorithms and the representation used to specify planning problems. The section also discusses planning with additional constraints and lays the groundwork relevant for building a narrative planner with added expressivity. Section 2.2 specifically provides background on narrative planning work. The section situates other narrative planning systems that reason about intentionality and disparate knowledge models for characters and how they relate to the approach in this work. In Section 2.3, we present the most closely related work in cognitive models on comprehension of narratives and failed actions. Finally, Section 2.4 presents computational approaches to support the human authoring process for stories. This section illustrates how computational narrative understanding and narrative generation technologies have been employed to assist in authoring and evaluation approaches.

2.1 Planning

Applications of planning algorithms are impacting technical disciplines and industries worldwide, ranging from protein folding to spacecraft mission sequencing [33, 44, 94]. Planning involves searching for a sequence of formally defined actions that transforms an initial world state into a desired goal state. Planners are concerned with causal reasoning and sequential decision-making– certain actions cannot happen before others (for example, one cannot paint the walls before building them). Within a typical planning knowledge representation, each action is represented by a set of **preconditions**: conditions that must be held in the world for the action to be executed, and **effects**: conditions that describe the changes in the world after the action is executed. The world state itself is described using predicate logic, indicating a set of facts that are true or not true in that state. The initial state represents the state of the world at the beginning of the plan. The goal conditions describe a set of authorial goals– a world state that meets the goal conditions is considered a goal state.

2.1.1 The Planning Domain Definition Language (PDDL)

The Planning Domain Definition Language (PDDL) [38] is a family of languages invented to describe planning problems in STRIPS-style. It was intended to express the "physics" of a domain. Over the years, PDDL has been used extensively to represent planning problems for the AI Planning and Scheduling Conference (AIPS) Planning Competitions. The following example shows a domain described in PDDL from [38]:

```
(define (domain briefcase-world)
    ...
    (:constants (briefcase - physob))
    (:types location physob)
    (:predicates (at ?x - physob ?l - location)
                 (in ?x - physob))
    ...
```

In this representation, there are two predicates: *at* for representing the location of an object, and *in* for representing the object inside the briefcase. The layout of the world in the inital and goal state can be described using these predicates in the following manner:

```
    ...
    (:init (place home) (place office)
          (object paycheck) (object documents)
          (object briefcase)
           (at briefcase home) (at paycheck home)
           (at documents home) (in paycheck))
    (:goal (and (at briefcase office) (at documents office)
    (at paycheck home))))
```

The above representation specifies that, in the beginning, the briefcase, paycheck and the documents are at home, and the paycheck is stored in the briefcase. The goal state indicates that in the end of the story, the briefcase and the documents should be at the office and the paycheck should be at home. Below, we also provide an example of PDDL used to define an operator `take-out ?x` that represents the action of taking x out of a briefcase:

```
(:action take-out)
    :parameters (?x - physob)
    :precondition (not (= ?x briefcase))
    : effect (not (in ?x)) )
```

Newer versions of PDDL have grown in expressivity to allow for more specifications on desired plans. PDDL 3.0 introduced the concept of state trajectory constraints, allowing for higher expressivity to specify constraints desired plans must meet [36]. The added expressivity provides for description of intermediate states and orderings between them through modal operators: *always, sometime, at most once, sometime after, sometime before* among others. For example, in a domain for truck deliveries, constraints can specify that all trucks must not only end in a particular location (a goal state constraint) but that some trucks must have visited a particular location at some time in the plan (a trajectory constraint). These additions in PDDL are relevant to this dissertation as we later develop additional expressivity to support narrative authoring.

2.1.2 General Planning Algorithms

The planners discussed in this work are based on the STRIPS-style language of specifying input to the planner [31]. The STRIPS language represents the world as a set of predicates. These predicates are used to describe the world as well as define actions in the world. Actions are defined by specifying the predicates that must hold for the action to be executed in the world and the predicates that are changed as an outcome of the action.

2.1.2.1 State-Space Search Planners

As the name implies, state-space search planners explore possible plans in a search space where each node is a viable state. Each edge is an action that can be executed at the current state, with the resulting state as the next node. Since the action definitions specify both preconditions and effects of an action, it is possible to search in either direction: forward from the initial state or backward from the goal state. Forward-directed state-space search planners use the initial state of the problem to consider possible sequences of actions until a sequence that leads to the goal state has been found [10].

2.1.2.2 POCL-Based Planners

State-space planners typically produce *totally ordered* plans, i.e., they explore strict linear sequences of actions. On the other hand, Partial Order Causal Link (POCL) planners such as SNLP and UCPOP [63, 67] attempt to solve planning problems using partially ordered plans as solutions. These planning algorithms attempt to find all necessary conditions that must hold at various points in a plan (*causal links*). The planners then attempt to resolve threats to existing causal links created by recent plan modifications, resulting in a solution that includes partial ordering between actions. Planning approaches built upon POCL use the concept of causal links to ensure that conditions hold in the world at necessary points, and when any of these conditions are threatened, the plan is modified to resolve the threat. Other planners, such as DPOCL [112], use similar approaches and construct partial-order-plan solutions to planning problems.

2.1.2.3 FF Planner and Heuristic

FastForward (FF) [45] is one particular forward-directed state space search planner. The FF planner was especially notable because of its simplified planning graph heuristic that combines forward and local search in a novel way. To determine the next state node to explore, FF uses a heuristic that considers a relaxed solution to provide an approximation of how likely a potential state can lead to a goal state. The relaxed solution involves only considering additive effects and ignoring the delete effects of an action. The heuristic is based on the HSP planner's max-level heuristic h_{max} and additive heuristic h_{add} [11], and considers the number of helpful actions in the relaxed plan graph.

2.1.3 Planners with Additional Constraints

As AI Planners have evolved alongside the languages they use, planners offer different levels of PDDL support and different approaches to solving a planning problem. Two of these approaches that are relevant to work in this dissertation are described here.

2.1.3.1 Planning with State-Trajectory Constraints in $SGPlan_5$

The $SGPlan_5$ Planner is capable of planning with state trajectory constraints [47]. The planner employs several strategies to solve a planning problem with hard and soft intermediate state constraints. First, an input planning problem is converted from a STRIPS representation to a multi-valued domain function (MDF) representation based on the SAS+ formulation [5]. This conversion allows for literals dependent on each other to get grouped into variables. For example, in a domain where a truck can be only at one location at one time, the literals `(at truck warehouse)` and `(at truck factory)` get reduced to a single multivalue variable that represents the location of the truck and can take one of two values: warehouse or factory. Second, $SGPlan_5$ analyzes the newly formed MDF representation to understand the domain further. The analysis involves finding guidance and bottleneck variables to identify localities in the domain. Their approach then applies a partitioning approach to divide the problem into a number of sub-problems, each with its subgoals. $SGPlan_5$ then uses a modified version of the Metric-FF planner to solve the planning problem. This approach was shown to perform competitively for IPC (International Planning Competition) benchmarks.

2.1.3.2 Planning with Action Constraints in PAC-C

Action constraints refer to the ability to provide specifications of actions in the planning problem that are desired in the final plan [9]. Action constraints can be helpful when plans need to have specific properties beyond initial and goal state specifications. For example, in the delivery domain, an action constraint could involve restricting the use of a particular truck to certain routes or requiring that a specific van deliver a particular package. In a murder mystery narrative domain, authors might want generated plots to strictly include a murder. Action constraints expand the expressivity available when specifying constraints in a planning problem.

Action constraints are not yet supported in current PDDL standards. However, some approaches have considered action constraints and preferences in planning algorithms. Recent work by Bonassi et al. has considered action constraints in planning and implemented a translation algorithm to precompile the action constraints into state constraints that meet PDDL 3.0 specifications [9]. Their approach considers a problem specification similar to PDDL 3.0 but also includes action constraints. In their proposed approach, Bonassi et al. use a precompilation strategy to compile action constraints into state constraints. Their precompilation algorithm, PAC-C, is central to our approach in allowing users to specify additional constraints over characteristics of generated stories. We provide a brief overview on PAC-C in this section.

PAC-C requires action formulae to be specified in disjunctive non-negative form, i.e., any specifications about action constraints must be in an OR-relationship without any negations.

Definition 1. *Let A be a set of actions of a planning problem. Given a plan π, an action formula ϕ defined over A is true at time t in π, i.e., $\pi(t)$ satisfies ϕ, iff:*

- *If $\phi = a$ then $\pi(t) = a$.*
- *If $\phi = \neg a$ then $\pi(t) \neq a$.*
- *If $\phi = \psi_1 \wedge \psi_2$ with ψ_1 and ψ_2 action formulae over A, then $\pi(t)$ satisfies ψ_1 and $\pi(t)$ satisfies ψ_2.*
- *If $\phi = \psi_1 \vee \psi_2$ ψ_1 and ψ_2 action formulae over A, then $\pi(t)$ satisfies ψ_1 or $\pi(t)$ satisfies ψ_2.*

Based on modal operators for state trajectory constraints specified in PDDL 3.0, Bonassi et al. introduce modal operators for the action constraints. The modal operators for action constraints are similar to those for state trajectory constraints, with the addition of *always next* and *pattern* modal operators.

Definition 2. *Given a plan $\pi = \langle a_1, a_2, \dots, a_n \rangle$, the following rules define when an action constraint is satisfied by π:*

- *π satisfies (always ϕ) iff $\forall t : 1 \leq t \leq |\pi| \cdot \pi(t) \in \phi$.*
- *π satisfies (sometime ϕ) iff $\exists t : 1 \leq t \leq |\pi| \cdot \pi(t) \in \phi$.*
- *π satisfies (at most once ϕ) iff $\forall t_1 : 1 \leq t_1 \leq |\pi| \cdot$ if $\pi(t_1) \in \phi$ then $\forall t_2 : t_1 < t_2 \leq |\pi| \cdot \pi(t_2) \notin \phi$.*
- *π satisfies (sometime after $\phi\,\psi$) iff $\forall t_1 : 1 \leq t_1 \leq |\pi|$ if $\pi(t_1) \in \phi$ then $\exists t_2 : t_1 \leq t_2 \leq |\pi|\ \pi(t_2) \in \psi$.*
- *π satisfies (sometime before $\phi\,\psi$) iff $\forall t_1 : 1 \leq t_1 \leq |\pi|$ if $\pi(t_1) \in \phi$ then $\exists t_2 : 1 \leq t_2 < t_1\ \pi(t_2) \in \psi$.*

- *π satisfies (always next $\phi\,\psi$) iff $\forall t : 1 \leq t < |\pi|$ if $\pi(t) \in \phi$ then $\pi(t+1) \in \psi$ and $\pi(|\pi|) \notin \phi$.*

- *π satisfies (pattern $\phi_1 \ldots \phi_k$) iff $\exists$ a sequence of actions $\langle a_1, \ldots, a_k \rangle$ from π that are ordered as in π, such that $\forall i \in \{1, \ldots, k\}$ $a_i \in \phi_i$.*

A planning problem with these action constraints is then passed into the PAC-C algorithm. The PAC-C algorithm compiles this planning problem with action constraints into an equivalent classical planning problem. It adds atoms to the planning problem to request or prevent actions in the resultant plans. For every (*at most once* ϕ) and (*sometime before*$\phi\,\psi$), the atoms $done_\phi$ and $done_\psi$ are used to record whether ϕ or ψ are ever held. For every (*always next* $\phi\,\psi$), atom $request_\phi$ is used to signal that the formula ϕ is satisfied at a plan step t, and the planner has to schedule an action $a \in \psi$ immediately after t. Similarly, for every (*sometime* ϕ) and (*sometime after* $\phi\,\psi$), atoms got_ϕ and $got_{\phi,\psi}$ are used to record whether or not the constraint is satisfied by the prefix plan. For every (*pattern* $\phi_1 \ldots \phi_k$), a set of atoms to track the progress of the stage in the pattern is added. The algorithm, presented in Algorithm 1, consists of three steps: (1) setting up the initial state of the world with the necessary atoms, (2) changing preconditions and effects of the actions based on the constraints, and finally, (3) adding the constraint atoms to the goal state of the planning problem.

2.2 Narrative Generation

Narrative planning approaches have extended classical planning approaches to produce stories as plans [76]. Approaches in this space consider a planning problem in a narrative context: the initial state is the beginning of the story, and the goal state is typically conditions in the story world that must hold at the end. The planning algorithm then attempts to find a sequence of actions that lead from the initial state to a possible goal state, and this path forms the plot for the generated story. Numerous systems have been developed that generate stories. Given the vast number of story generation approaches, it is beyond the scope of this dissertation to review all of them. The narrative generation approaches described in this section are most relevant to the research on story generation presented in this work. There has been a significant amount of work on narrative genera-

```
**PAC-C**$\langle\langle F, A, I, G\rangle, C\rangle$
// Part 1
$F' \leftarrow F \cup$ *prevent-atoms* $\cup$ *request-atoms*
$I' \leftarrow I \cup \bigcup_{sometime\ after\ \phi,\psi} got_{\phi,\psi}$
// Part 2
$A' \leftarrow \{a | a \in A$ and for each $A_\phi \in C, a \in \phi\}$
**for all** $a \in A'$ **do**
  **for all** $c \in prevent\text{-}atoms(C)$ **do**
    **if** $c = at\ most\ once\ \phi$ and $a \in \phi$ **then**
      Pre(a) = Pre(a) $\wedge \neg done_\phi$
      Eff(a) = Eff(a) $\cup \{done_\phi\}$
    **end if**
    **if** $c = sometime\ before\ \phi, \psi$ **then**
      **if** $a \in \phi$ **then** Pre(a) = Pre(a) $\wedge\ done_\psi$
      **if** $a \in \psi$ **then** Eff(a) = Eff(a) $\cup \{done_\phi\}$
    **end if**
    **if** $c = always\ next\ \phi, \psi$ **then**
      **if** $a \in \phi$ **then** Eff(a) = Eff(a) $\cup \{request_\psi\}$
      **else if** $a \in \psi$ **then** Eff(a) = Eff(a) $\cup \{\neg request_\psi\}$
      **if** $a \notin \psi$ **then** Pre(a) = Pre(a) $\wedge \neg request_\psi$
    **end if**
  **end for**
  **for all** $c \in request\text{-}atoms(C)$ **do**
    **if** $c = sometime\ \phi$ **and** $a \in \phi$ **then** Eff(a) = Eff(a) $\cup \{got_\phi\}$
    **if** $c = sometime\ after\ \phi, \psi$ **then**
      **if** $a \in \psi$ **then** Eff(a) = Eff(a) $\cup \{got_{\phi,\psi}\}$
      **if** $a \in \phi$ **and** $a \notin \psi$ **then** Eff(a) = Eff(a) $\cup \{\neg got_{\phi,\psi}\}$
    **end if**
    **if** $C = pattern\ \phi_1, ..., \phi_k$ **then**
      **for all** $\phi_i \in \langle\phi_1, ..., \phi_k\rangle$ **do**
        **if** $i > 1$ **then** Eff(a) = Eff(a) $\cup \{stage_c^{i-1} \rhd stage_c^i\}$
        **else** Eff(a) = Eff(a) $\cup \{stage_c^1\}$
      **end for**
    **end if**
  **end for**
**end for**
// Part 3
$G' = G \wedge \bigwedge_{sometime\ after\ \phi,\psi \in C} got_{\phi,\psi} \wedge \bigwedge_{somteime\ \phi \in C} got_\phi \wedge \bigwedge_{always\ next\ \phi,\psi \in C} \neg request_\psi$
  $\wedge \bigwedge_{c = pattern\ \phi_1 ... \phi_k \in C} stage_c^k$
**return** $\langle F', A', I', G'\rangle$
```

Algorithm 1: Algorithm for converting PAC problem into classical planning problem as proposed by Bonassi et al. [9]

tion that uses AI planning algorithms with extended knowledge representations intended to increase the expressive range [93] of these generative systems relative to conventional planners. The approaches accomplish this by considering aspects such as tension [24], suspense [17, 18, 66], character personality and affect [7, 25], intentionality [76, 97] and nested belief [89] representations in the narrative planning process.

This section briefly introduces some of the earlier work in narrative planning. Section 2.2.2 discusses advances in narrative planning focused on character intent dynamics. In Section 2.2.3, narrative planning approaches that consider character belief are described. Finally, this section ends with a review of planners that allow for added narrative control with additional constraints in Section 2.2.4.

2.2.1 Classical Narrative Generation Approaches

Early work in narrative planning considered two distinct approaches to narrative planning: one guided by authorial constraints, often called author-centric narrative planning, and one guided by characters and character goals, referred to as character-centric narrative planning [75, 78].

Tale-Spin [64] is a system that generates stories like those from Aesop's Fables by modeling characters in a story world. Tale-Spin is capable of character-centric narrative planning, allowing for generation of stories where characters exhibit behavior that contributes towards their goals. The Universe system [60] uses an author-centric narrative planning approach to generate soap opera-like stories that meet authorial goals. Unlike Tale-spin, Universe strictly considers the author's goals when finding stories. Facade [62] is a narrative generation system that takes an author-centric approach to narrative generation. In Facade, autonomous characters may form goals that are not relevant to the unfolding of the story and perform actions to achieve them. Facade also uses a drama manager that works at the scene level and sequences plot elements (called beats) into a balanced narrative experience.

2.2.2 Intentionality-Based Planners

As narrative generation systems dived deeper into making believable stories, approaches started considering character intent in addition to character goals [76, 97]. While a typical author-centric narrative planner would consider adding any action to the story as long as it leads to the goal state, approaches in intentionality focus on whether the planner selects actions that match up with the intent of characters in stories. In particular, it allows characters only to perform actions that contribute to their own personal goals. These approaches consider character goals in conjunction with authorial goals– characters have their own goals, which may or may not match up with authorial goals. Moreover, not all character goals may be achieved in the story, but the planner must ensure that all actions performed in the produced stories are in pursuit of the characters' goals.

The proposed work in this dissertation uses intentionality and the concept of *intention plans*, plans that characters come up with to achieve their desires. In this model, authors specify character goals as *desires*, a framing based on belief-desire-intention cognitive reasoning [12, 22]. Desires include a (a) partial description of the world state that the character prefers, and (b) context for when the desire could become an intention, called *motivations*. When a character believes all motivations for a desire hold in the current world, the character considers possible plans to fulfill the desire. If the character can find a possible plan to achieve the desire, they commit to this plan, called an *intention plan*. Intention plans are partial plans that are from the characters' perspective and aimed at achieving the specific goal defined in the desire for the character. Steps in the intention plan may have *causal links*: characteristics about the world that must hold between steps for the plan to be successful. When a causal link is violated, i.e., something in the world changes that invalidates an action in their intention plan, characters are prompted to re-evaluate and update their current intention plans.

The Intent-based Partial Order Causal Link (IPOCL) planner by Riedl and Young [76] was the first narrative planning causal link planner focused on character intent. IPOCL's focus on intent was to link every action a character chooses to the relevant character goal in the resulting story, i.e., every action was explainable. The IPOCL planner also experimentally verified that the audience could successfully track the world state and infer the intentionality of the characters via frames of commitment. IPOCL added an explicit representation of *intention frames*, groupings of actions performed by a character in furtherance of a single goal. Each frame is established by a motivating action whose sole effect is the intended condition, and all character actions in plans produced by IPOCL belong to at least one intention frame. While all character actions in IPOCL plans can be seen as motivated towards a character's goals, characters in IPOCL plans never drop or adapt their plans, and the rationale for the addition of motivating actions is not based on character beliefs or desires.

The IMPRACTical (Intentional Multi-agent Planning with Relevant ACTions) system was proposed by Teutenberg and Porteous [97] as a way of addressing run-time challenges with planning systems such as IPOCL, especially when used in the context of *interactive* storytelling. The IMPRACTical system delegates the task of reasoning about intentions and narrative action relevance to narrative agents: the narrative agents are responsible for determining actions that are relevant to the agent based on intentionality. Then, a single narrative planner is used to generate narratives that satisfy the constraints obtained from the relevant actions provided by the narrative agents. The IMPRACTical system is capable of producing intentional plans where characters can cooperate, issue chains of command, and predict other characters' actions. The IMPRACTical planner is a forward-directed state space search planner. It uses a modified version of the FastForward-based heuristic– in which the relaxed plan graph also has arcs between relevant command actions and relevant actions to represent causality from intent accurately.

The IRIS system uses intention revision in the narrative planning approach to create narratives with suspense [30]. In this approach, characters are capable of updating as well as dropping intentions if they violate *working assumptions* for the character's intent. Our model of intent is similar to the one adopted by the IRIS system. However, while the IRIS system primarily focused on using intent revision in narrative planning to create suspense, we use intentionality to further guide the behavior of agents upon action failure.

2.2.3 Planners that Reason About Character Belief

Narrative planning research has incorporated additional constructs into the planning process to expand the expressive range of plan representations to support aspects of character decision-making beyond intent. Extending IPOCL, Ware and Young's CPOCL [106] adds a representation of conflict in story plans where one character might take actions that interfere with or thwart the execution of actions taken by other characters. In the Mask planning system, Bahamón and his collaborators [6, 7] incorporate a model of character personality that is used to drive character choice for action that expresses character personality traits.

These extensions to the classical planning approach have assumed two things. First, they make no distinction between the knowledge held by characters and that held by the planning system. Second, the algorithms are based on a long tradition of planning research outside of narrative planning, where the soundness of planning algorithms is a requirement. This is a highly desirable property when producing plans for robot execution, for example, running robots executing tasks on a factory floor. The focus of classical planning approaches on the representation of physical properties of the world and on requirements that any plan produced by a planner must be sound limits the production of plans to drive characters bumbling (by authorial design) through a story world.

The initial work addressing disparities of knowledge between agents in a planning context was done by Pollack in her work on the Spirit plan inference system [68]. This work motivated Geib's [35] approach to formalizing intention in a plan generation system. As part of the resulting ItPlanS planner, Geib and Webber [34] distinguish an action's preconditions and other conditions that are necessary for an action's execution (but are not established by the planner should they not hold). Geib also considers the importance of reasoning about action failure in plan generation. Cavazza and his collaborators [15] describe an approach to the generation of story sequences where characters are unaware of some aspects of the world around them, including the harmful consequences of some of their own actions.

Shirvani and their collaborators [89] propose an extension to state space planning models that represents character beliefs as well. In their model, agents also have beliefs about the world, and their beliefs about the world may be different from the actual state of the world. This model can also produce situations where agents act on incorrect beliefs. The authors discuss the concept of "surprise actions," where an agent is surprised by the outcome of an action because they have different beliefs. However, their approach doesn't leverage these beliefs to extend the space of narrative plans that can be produced to include ones where characters perform actions that can fail due to their flawed beliefs about the world.

Recent work by Eger and Martens [26] discusses how character beliefs play a central role in many narratives, but are often not represented in planning-based author-centered narrative generation systems, or are only represented in ad-hoc ways. The authors propose using Dynamic Epistemic Logic to represent actions that affect a story world and its characters' beliefs, in the context of narrative generation. This ensures that characters' beliefs are maintained in a logically consistent way. Eger, Martens, and Córdoba [27] also use this epistemic logic in an innovative approach to developing artificial intelligence control for the cooperative card game Hanabi.

An extension to IMPRACTical [98] enables balancing global plot-level planning with local character intent-driven planning. The extended work also allows for separate belief models for each agent in a story world. Using a combination of observation axioms and operator annotations, their system can create disparities between the belief models of the agents and the world state. Plan generation is directed based on actions supported by beliefs of the enacting agent. This enables *deceptive social actions* involving manipulation of the belief state of one agent by actions of another. Subsequently, the manipulated agent can be induced to act against its interests because of its incorrect beliefs. However, for an action to be included in a plan, the preconditions of the action itself must both be believed to be true by the performing agent and must hold in the world when the action is attempted. In situations where false beliefs of an agent suggest that an action be undertaken whose preconditions do not hold in the world, the suggested action is disregarded. Their planning model prevents executable actions from occurring when characters lack the belief in the actions' preconditions. In contrast, the planning model described in this dissertation enables attempts at actions where the actions are not executable, but characters believe that they are.

2.2.4 Narrative Control with Additional Constraints

Work in this space is growing to produce stories with qualities that human authors may find interesting. Character belief has been studied by various approaches, for instance, those taking into account character beliefs and theory of mind [26, 105]. The previously described Mask planning system allows for the specification of personality-related constraints to create stories where characters exhibit behavior that reflects their personalities [6]. Recent work by Shirvani and Ware allows for planning with consideration for character emotions [87].

Work in narrative planning has considered the possibility of additional constraints to provide more authorial control over narrative planners. State trajectory constraints have been used in narrative planning to specify intermediate points in the desired story. Riedl used *islands* as a way for the author to guide the narrative generation process and force the planner to look for more complex plans. [77] In their approach, islands were state constraints that authors could provide to guide the planner, requiring that every story plan produced should progress at some point through states that correspond to island descriptions. This approach helps when authoring stories where beginning and ending constraints are unable to capture specific properties of stories. For example, if the author wants to guide a narrative planner through a story in the *Little Red Riding Hood* world where the wolf eats the grandmother at some point in the story but the grandmother is saved and alive at the end of the story, specification of intermediate world states would guide the planner to produce stories with such plots.

Porteous and Cavazza also use state constraints in narrative planning, with operators such as *sometime before* and *sometime* to specify temporal constraints around states occurring in stories produced in an interactive storytelling system [70]. These approaches have successfully demonstrated the usefulness of allowing authors to specify more constraints beyond initial and goal conditions to have more control over the narrative produced. To our knowledge, no approaches in narrative planning have considered action trajectory constraints.

Recent work in narrative generation has considered some aspects of plan failure. Porteous and Lindsay [71] have looked at non-cooperative narrative planning where characters act to sabotage each others' plans. In their model, plan sabotage results in characters having to reassess the world and devise ways to achieve goals in a competitive environment. Their approach considers intent revision in the face of new information, but does not specifically address the potential failure of actions.

2.3 Cognitive Models Supporting Human Comprehension of Stories

Cognitive scientists have studied how humans understand stories, and this understanding of human cognition has also influenced the field of computational narratives [13].

Computational approaches have attempted to create stories that are not only believable but also perceived by human cognition in a manner consistent with human stories. For this reason, there are several cognitive models and theories that serve as background for understanding and evaluating computational narrative generation systems.

There is considerable evidence in cognitive psychology on how humans comprehend narrative. Human cognition plays an active role when experiencing narrative, and humans actively track intentionality and reasoning about the narrative world when experiencing narrative [12]. Humans recollect various aspects of events differently depending on event indices that are shared between a current event that they are exposed to and past events in the story [115, 116]. Readers are problem-solving and finding solutions in the story world– a cognitive activity that leads to the human experience of suspense [37].

Stein and Albro study how children tell and understand stories and propose the Goal-Action-Outcome framework as an effective model to track comprehension of goal-directed narrative [95]. They specifically study the extent to which children process both successful and failed outcomes, and find that children as young as 3-year-olds can track unsuccessful outcomes for characters in stories. Humans can track intent in narratives when a failed outcome occurs, by searching for obstacles to goals, and also tracking intent dynamics upon failure: reinstatement of goals, generation of new related goals, and unrelated goals. Trabasso and Wiley [101], in their review on cognitive models for inferences during comprehension of narrative, provide an overview of models of comprehension of events. They examine the role that causal inferences play in understanding narratives where characters succeed or fail at achieving goals.

This section will now describe one specific narrative comprehension model designed specifically to examine readers' knowledge of human goals through question-answers, called QUEST.

2.3.1 QUEST: A Model of Question-Answering

QUEST [41, 42] is a model of human question-answering in the context of stories. The QUEST model relies on graphical representations of knowledge known as QUEST knowledge structures (QKSs). These QKSs are built from story structure and have been experimentally evaluated to correlate with aspects of human narrative comprehension. QUEST knowledge structures use nodes for events, goals and states. Edges represent causal, intentional, temporal or other types of relationships between these nodes. Questions about event nodes (e.g., why did an event happen, how did an event happen, what were the consequences of an event happening) can potentially be answered by other nodes in the graph (e.g., From ”What is the consequence of event *A* happening?” to ”State *S* is a consequence of event *A*.”) Question-answer pairs can be generated from the knowledge graph by selecting two nodes of the appropriate types: one node to serve as the source of the question and another node to serve as the source of its potential answer. The QUEST model defines how the arc-distance between the question and answer node is calculated from the graph, starting from the question node as the source and finding possible target nodes as answers. This arc distance acts as a predictor of how strongly a human reader will rate the answer node as an effective answer to the question. The arc search procedures between question nodes and answer nodes vary depending on question type, and are primarily sensitive to arc direction and arc label.

As mentioned above, the QUEST knowledge structure consists of a directed graph with labeled nodes and arcs. QUEST has four types of nodes: event, state, goal and style. Nodes represent proposition-like expressions which can contain a predicate (i.e., verb or adjective) and one or more arguments (nouns, embedded propositions). A state node represents an ongoing characteristic which remains unchanged within the timeframe it is presupposed. An event is a state change that occurs within the timeframe. A goal refers to a state or an event that the agent desires. This work looks at QUEST graph structures primarily in the context of goal hierarchies as defined by [41, 42] and hence style nodes are not considered. There are also multiple types of arcs in QKSs: Table 2.1 provides a concise definition of some of the arcs that are present within QKSs and are important in the context of this work.

2.3.1.1 Arc Search Procedures in QUEST Knowledge Structures

For goal-oriented substructures, Graesser [42] defines the arc search procedures for ”Why,” ”How”, and ”What are the consequences of” questions. We describe the arc search procedures for these questions and then use the arc search procedures on the proposed algorithm to show that the QKS created is effective at finding possible answers to questions, and the arc search procedure rates them in a way consistent with human comprehension.

The ″Why″ questions have four sets of nodes produced as answers when probed on a goal node G. They are (1) superordinate goals via paths of forward reason and backward manner arcs, (2) sibling goal nodes via paths of forward before-arcs, (3) goal initiators connected to G or G's superordinate goals by a backward initiate arc, and (4) causal antecedents to each goal initiator which initiate from the goal initiator via backward consequence arcs, implies arcs, backward outcome arcs, and backward initiate arcs.

The ″How″ questions have likely answers on nodes that are found by traversing through backward reason arcs and forward manner arcs.

″What are the consequences of″ questions can be answered in two ways. The first way involves traversing through forward Reason arcs and backward Manner arcs to find achieved superordinate goals. The second involves finding causal consequences of the queried node and the superordinate goals by following forward consequence, implies, outcome, and initiate arcs.

2.3.1.2 Leveraging QUEST to Gauge Narrative Comprehension

QUEST has been extensively used to gauge narrative comprehension for plans produced by computational narrative generation approaches. Christian and Young [20] proposed the first approach for translating a narrative plan generated by an AI planner into a QUEST knowledge graph. Their process was automatic and was evaluated to show that the approach was successful at predicting user comprehension in a story. Additionally, Riedl and Young used QUEST as the cognitive model for evaluating the IPOCL planner [76]. Cardona-Rivera and their collaborators [14] studied the QUEST model with respect to intention-based partial order causal link (POCL) plans, summarizing the approaches by Christian and Young [20] as well as Riedl and Young [76]. They proposed another algorithm to generate QUEST knowledge structures based on previous approaches. However, existing approaches that translate the output of narrative planning systems into QKS structures are limited. Their source plans do not contain failed actions, and so their translation processes do not address elements that are increasingly present in more expressive narrative plan generators (e.g., those planners described in Section 2.2 above). As we explain below, QUEST already has a methodology for representing failed actions;

our work aims to provide a way to translate narrative planners into QUEST knowledge graphs that can represent failed actions.

In order to incorporate failed actions in QUEST knowledge structures in a way that can answer goal-related questions, we must ensure that the structure can appropriately answer all three questions above: ”how”, ”why”, and ”what are the consequences”.

Christian and Young [20] defined the initial approach to generate QUEST knowledge structures automatically from plans generated by narrative planners and used that method to empirically evaluate viewers’ comprehension of a cinematic narrative. One limitation of Christian and Young’s algorithm is that it did not consider failed actions and, therefore would not produce a sensible QKS for a plan incorporating failed actions. Essentially, the reason arcs from goals corresponding to failed actions within a QKS generated by their algorithm would cause incorrect question answering for the failed action’s goal. For example, if a story plan involved a character attempting to do action A and failing, with the character proceeding to perform action B, the goal structure would read, ”the character wanted to do A in order to do B”. If a character tried to open a door without knowing it was locked, failed in its attempt, realized the door was locked, and then unlocked it, for example, this would read as ”the character wanted to open the door in order to unlock the door.” Instead, the question of ”why did the character try to open the door?” is better asked through character intent, not by what happens next in the finished plan.

2.3.1.3 Failed Actions in QUEST Knowledge Structures

In this section, we describe those portions of the QUEST knowledge structure that are related to failed actions and unachieved goals – both elements that have not been previously used in QUEST-based evaluations of user experience in narrative generation systems. Full definitions for the QUEST knowledge structures are provided in [41] along with examples and definitions for the components of the graph structure that models knowledge built from a human’s reading of a narrative text. In brief, an event is a primitive that represents the change in state; an action is composed of an event paired with a goal node using an outcome arc.

- **Failed Events.** Event nodes are one of the three types of nodes in the conceptual graph structures defined by Graesser and Clark. These nodes are used to represent two types of actions. First and most commonly, they represent events that take place in the world and cause the world to change in state. Second, the representation can also be used to represent failed events in the QUEST graph structure. Failed events represent an event where there was an unsuccessful attempt to change the world.

- **Unachieved Goals.** Unachieved goal nodes are used in QUEST to represent goals that are not achieved in a story. We divide these goals into two categories. In the first, goals could be unachieved because the characters never attempted to execute actions that would have led to their achievement in the story segment. These unachieved goals are represented as goal nodes with no outcome edges coming out of them. In our second category, goals can be unachieved because, through the character's actions, they realize that it is impossible to achieve them (either by observation or through a failed action). Graesser and Clark show one such example of this case in a knowledge structure for *REWARDING*. This is shown in Figure 2.1. In this figure, Node 2 is an example of a goal node that was unachieved because the event sequence does not show any actions that would have led to its achievement. Node 2 is a goal node for ″X wants to get a reward″, and the graph structure does not specify whether it was achieved due to the outcome not being determinate. On the other hand, Node 10 is an example of an unachieved goal node by attempting it. Node 10 is a subordinate goal node that represents X's goal of ″informing Z to stop threatening Y″. The goal node has an outcome edge directed to a state node which depicts that, due to the outcome node 11 (″It is impossible to inform Z to stop threatening Y″), X's goal of informing Z to stop threatening Y failed. In this particular example, upon failing to achieve their sub-goal, X resorts to another plan to achieve their superordinate goal (Goal 9: X wants to stop Z from threatening Y), which can be observed in goal node 12 ″X wants to kill Z″.

- **Failed Actions.** There is no explicit definition for failed actions provided by Graesser and Clark. Actions performed by an agent could fail due to many reasons. An action could fail simply because of how it is performed (such as making a three-point shot in basketball), fail because it has a probabilistic chance of success (such as rolling a six on a dice), or fail because the actor had incorrect beliefs about the world (such as attempting to open a door when it is locked). **Intentional actions** are defined as goal nodes linked to event nodes with a positive outcome arc. Based on this, we determined that failed actions must be goal nodes connected to an event node with a negative outcome arc, i.e., a character performed an event with a goal that did not go as they intended it to.

- **Negative Outcomes.** Graesser and Clark define Negative Outcome arcs explicitly as occurring when the outcome node directly clashes with the goal node. Outcome arcs can link to both event and state nodes, so negative outcomes can lead to either events or states. Graesser and Clark provide an example of a negative outcome to a state node shown in the arc between Nodes 10 and 11 in Figure 2.1.

2.4 Authorial Support Tools

One of the goals of narrative planning technologies is to help human authors create plots. AI can play the role of a co-creative agent- a human author can guide the AI by providing them with a story world and authorial requirements about the narrative experience and the AI can produce several stories for humans to experience that meet the desired constraints. There has been a steady interest in integrating intelligent narrative generation technologies into tools that authors can use to create stories [21, 54, 64, 110]. This section is interested in reviewing them to understand how computational narrative systems allow for the expression of authorial expressivity in authorial support tools, and how they have been evaluated.

One direct attempt at providing authorial support during the storywriting process is LISA, the Lexically Intelligent Story Assistant [82]. LISA uses a knowledge representation and reasoning approach to stories: the system compiles story events into subject-verb-object triples and adds them to a logical programming database. LISA's interface allowed authors to ask questions to the system about the story, as well as for the system to point out any rule-based logical inconsistencies within the text-editor itself. This work was further expanded to apply to screenplays, as the strong textual structure of screenplays allowed for reasoning about knowledge specific to characters in story [84]. The interfaces developed for these tools were designed to work with the discourse (text of the story) rather than the plot itself and were focused on allowing authors to interact with the story rather than get assistance with generating plot.

Storium, an online storytelling platform, uses a game-like approach to creating stories. The Storium interface involves turn-taking among collaborators, where every collaborator contributes to the story through a card-like interface that represents character actions and outcomes of events. Figure 2.2 shows part of the user interface for Storium. Interestingly, recent research studies have used this platform for story generation, studying how users work with the generated stories and edit them to create final versions [1]. Figure 2.3 shows an example outline of how the approach by Akoury et al. produces natural language story excerpts for Storium using information from various segments of the story as input.

Goldfarb-Tarrant et al. use a web-based interface for their interactive story generation system [39]. They use language models implemented using LSTM techniques to generate stories and assist users, and their interface allows for interactivity where users could work with the agent to create stories.

Loose Ends is a storytelling play experience that also features an interface for authors to work with an AI system to build stories [58]. This system aimed to improve authors' capability to keep track of multiple storylines and allow for suggestions of goals for characters as well as actions that contribute towards the proposed goals. Loose Ends has undergone preliminary evaluation using an approach where experts in creative writing and research on intelligent narratives used the system and then responded to a survey with both qualitative measures and quantitative metrics.

Wide Ruled is a story-generation tool where authors can use an interface to construct a story domain and narrative generation problem [92]. Their interface allows an author to construct a domain with characters, objects, environments, and more, along with a specification of goals. The evaluation for Wide Ruled involved human participants using the tool to create story worlds and plots. The experimental evaluation results found that participants responded positively to the editor abilities for character and environment editing in the user interface, but had trouble using complex aspects of the tool: operator editing, plot fragment editing, and story structure components. We used these assessments as guidance to design the user interface for the tool in Section 4.3: we attempted to use an editing interface similar in design to Wide-Ruled, while minimizing editing of complex features that did not directly contribute to our hypotheses, such as editing of operators or desires.

There is a body of tools that is focused on providing authors with creative ideas, such as Creative Help [79]. The goal of the AI in Creative Help is that of a co-creative agent, which when prompted, can collaborate on the story and act as a creative stimulant for the user. In this scenario, the user interface is minimal– the author can call on the AI to suggest a sentence with a simple text command. The evaluation for Creative Help involved studying how much the AI's contributed suggestions were edited, to gauge the tool's helpfulness. The evaluation outcome found statistical significance, albeit marginal, for users preferring the AI's suggestions over random suggestions.

The Plan-Write-Revise system by Goldfarb-Tarrant et al. [39] is one recent approach to understanding how humans can work with AI systems on story construction tasks. In their work, the authors built various story construction models using LSTMs. The authors also built a web interface for humans to interact with the models. In their intra-model interface, users can select a model, provide a topic for generating stories, change the diversity of content, collaborate on the planning for the story, and collaborate on the story sentences. A range of participants on MTurk used the system. The study found that all the human-computer collaborative models produced better-quality stories than the computer-only versions. Moreover, the study found that allowing users to return to a story to revise and improve it led to higher engagement. These insights informed out design and construction of the tool described in Section 4.3.

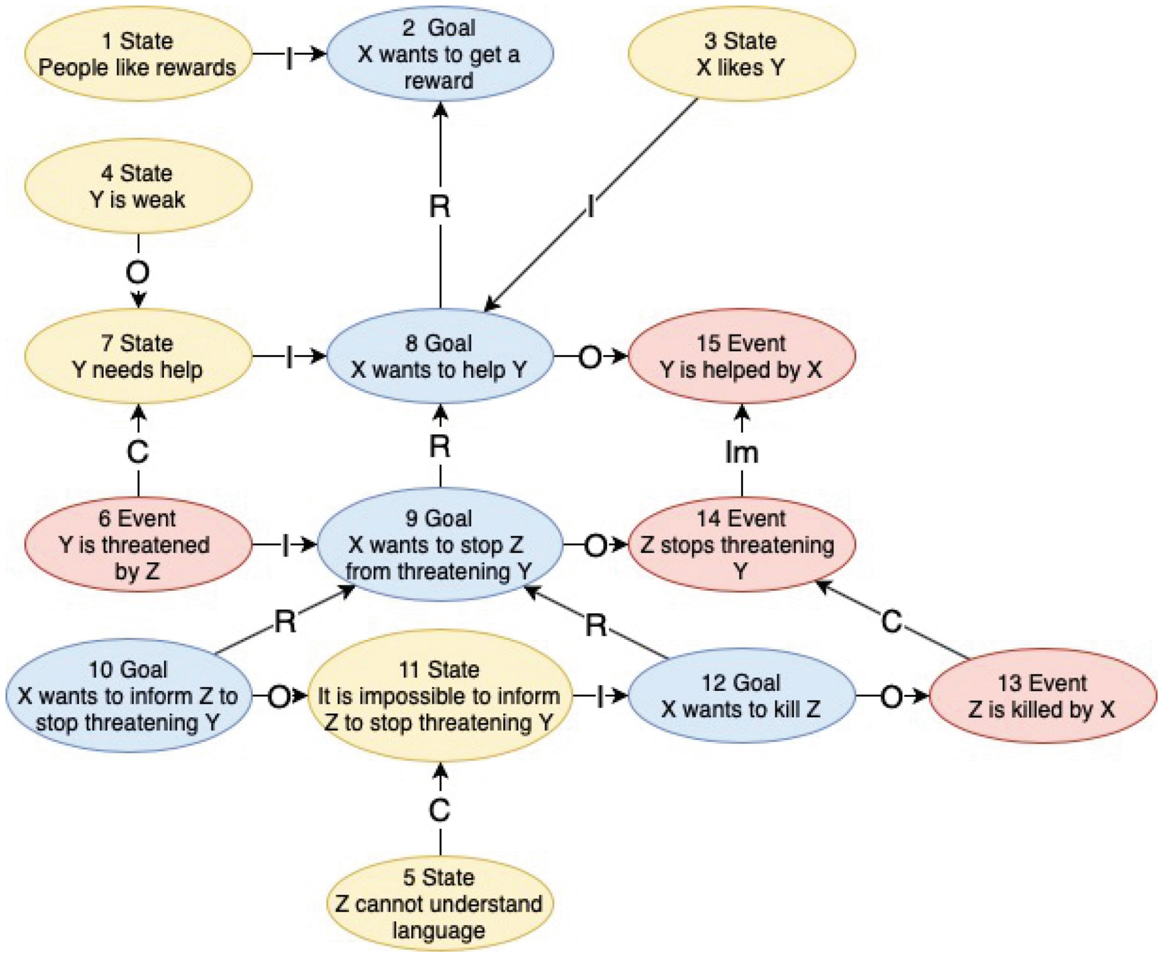
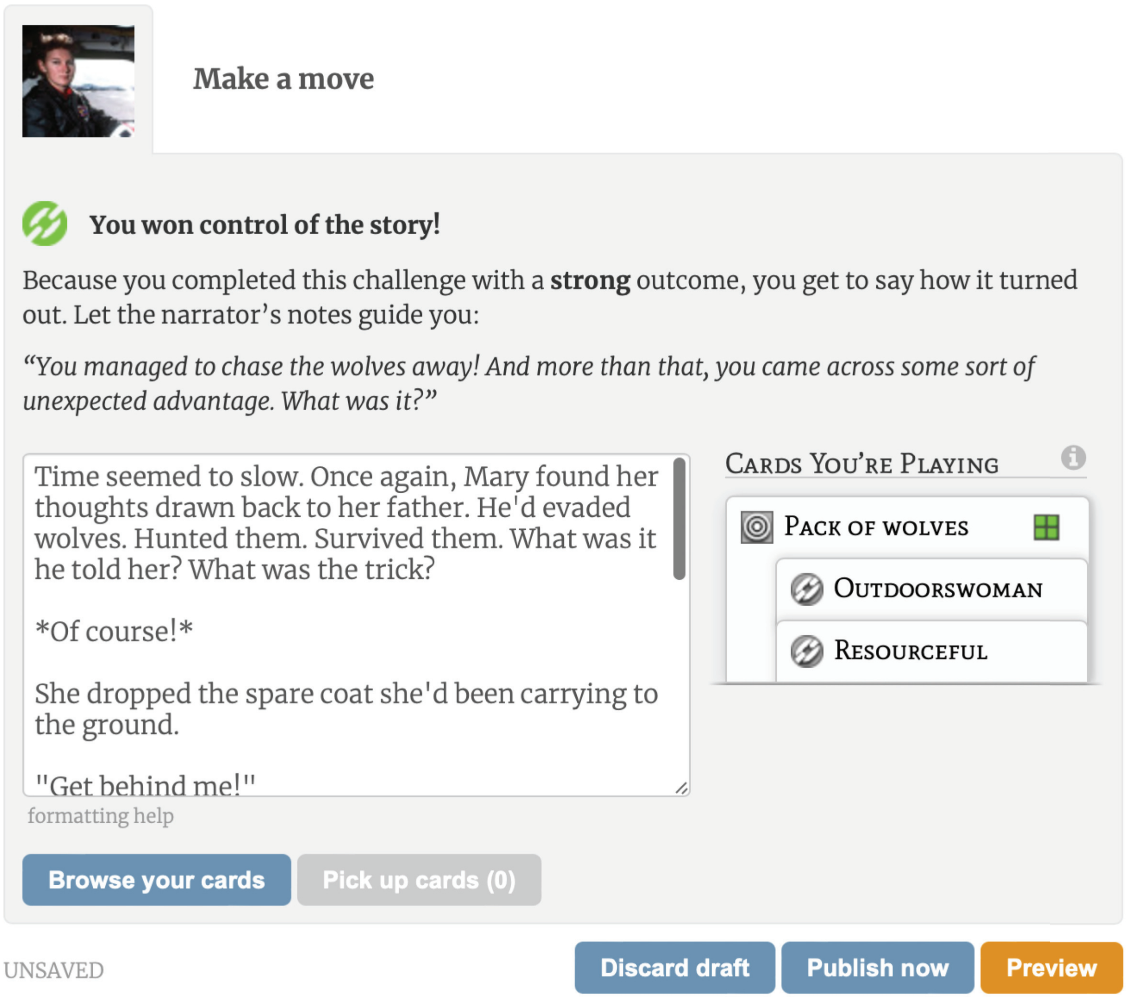


Figure 2.1. The conceptual graph structure for REWARDING, as depicted by Graesser and Clark. Nodes 2 and 10 are examples of unachieved goal nodes. Arcs labeled with "R" are reason arcs, "O" are outcome arcs, "C" are consequence arcs, and "I" is the initiate arc. Yellow nodes denote states, blue nodes denote goals, and red nodes denote events.



Make a move

You won control of the story!

Because you completed this challenge with a **strong** outcome, you get to say how it turned out. Let the narrator's notes guide you:

"You managed to chase the wolves away! And more than that, you came across some sort of unexpected advantage. What was it?"

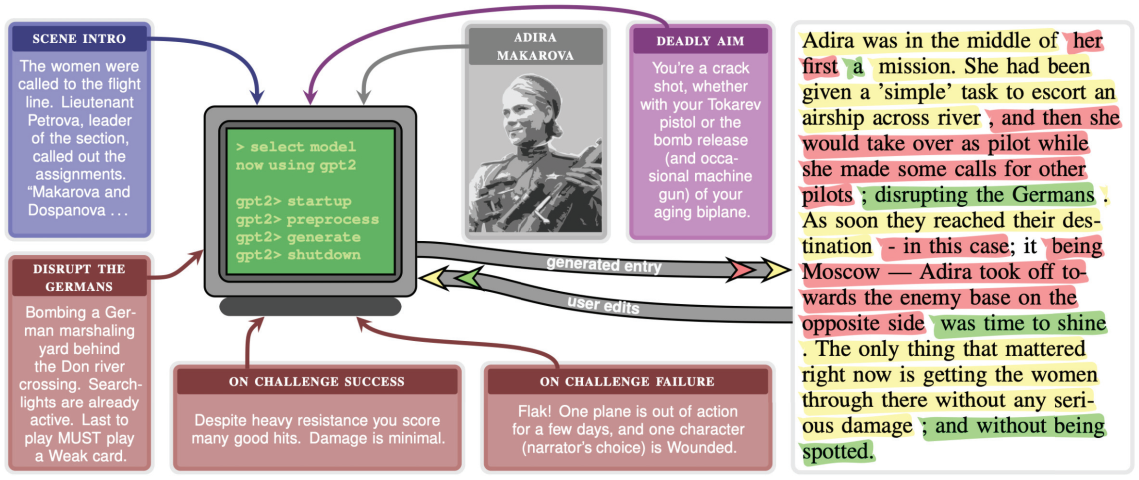
Time seemed to slow. Once again, Mary found her thoughts drawn back to her father. He'd evaded wolves. Hunted them. Survived them. What was it he told her? What was the trick?

Of course!

She dropped the spare coat she'd been carrying to the ground.

"Get behind me!"

Figure 2.2. The Storium interface. Players use a card-like interface to "play" action cards to change the story.



The women were called to the flight line. Lieutenant Petrova, leader of the section, called out the assignments. "Makarova and Dospanova . . .

You're a crack shot, whether with your Tokarev pistol or the bomb release (and occasional machine gun) of your aging biplane.

Bombing a German marshaling yard behind the Don river crossing. Searchlights are already active. Last to play MUST play a Weak card.

Despite heavy resistance you score many good hits. Damage is minimal.

Flak! One plane is out of action for a few days, and one character (narrator's choice) is Wounded.

```
> select model
now using gpt2

gpt2> startup
gpt2> preprocess
gpt2> generate
gpt2> shutdown
```

Adira was in the middle of her first a mission. She had been given a 'simple' task to escort an airship across river , and then she would take over as pilot while she made some calls for other pilots ; disrupting the Germans . As soon they reached their destination - in this case; it being Moscow — Adira took off towards the enemy base on the opposite side was time to shine . The only thing that mattered right now is getting the women through there without any serious damage ; and without being spotted.

Figure 2.3. An example outline of the approach by Akoury et al. Annotated Storium cards such as character cards (shown in gray), strength cards (shown in purple), and challenge cards (shown in red), are passed into the system along with other annotations by the player (shown in blue). The system then produces natural language text (shown in the right pane). Players may then edit the model output, by adding (green highlighted text) or deleting text (red highlighted text), before publishing the entry. Source: Page 2 [1]

Table 2.1: Selected set of arcs within the QUEST Knowledge structures as defined by Graesser et al. [41]

Arc Type	**Definition**	**Composition Rule**
Consequence (C)	A causes or enables B A precedes B in time	Event $\mid$ State -$C\rightarrow$ Event $\mid$ State
Reason (R)	B is a reason or motive for A B is a superordinate goal for A A is achieved before B is achieved	Goal -$R\rightarrow$ Goal
Outcome (O)	B specifies whether or not the goal A is achieved	Goal -$O\rightarrow$ Event $\mid$ State
Initiate (I)	A initiates or triggers the goal in B A precedes B in time	Event $\mid$ State -$I\rightarrow$ Goal

CHAPTER 3

GENERATION OF NARRATIVES CONSISTING OF FAILED ACTIONS

Building on the review of literature in which this research is contextualized, this chapter describes advances in generation of narratives consisting of failed actions. We introduce the HEADSPACE planner, a narrative heuristic search planner [45] capable of producing plot-lines consisting of failed actions due to incorrect beliefs [86]. The mechanics surrounding the action failure are grounded in character belief, leading to generation of stories where the nature of the failed action is understandable to humans. This planner can also construct plots with character behavior based on complex intention dynamics, incorporating existing work in narrative planning on character intent [76]. HEADSPACE also models observability, allowing characters to observe complete or partial effects of actions. This is based on a representation capable of reasoning about observability based on rules using first-order logic. HEADSPACE allows authors to specify additional authorial constraints about desired actions (successful or failed attempts) in the resulting plot, and desired character behavior upon action failure. Finally, this chapter describes work in translating HEADSPACE generated stories into a representation that can be used for evaluating comprehension of produced stories.

Figure 3.1 provides an overview of the HEADSPACE planner. The system operates by first taking the input planning problem (with the enhanced authorial constraints) and pre-compiling the specification using the Enhanced Authorial Constraints Precompiler, described in Section 3.3.2. Then, the transformed problem specification is provided to the planner: the top-level planner works with the character-level planner to produce intention plans (representing the commitments to action for each character) and integrate them into a single, global plan, which is provided as output.

This chapter is structured in a manner that reviews the key features of the planning algorithm incrementally. Section 3.1 focuses strictly on the representation and algorithm within HEADSPACE that produces failed actions due to incorrect character beliefs. Definitions for HEADSPACE plans and plans with failed actions are provided to explain how a failed action occurs in resulting plans, with the focus limited to authorial goals. Section 3.2 introduces intentionality and observability and describes the representation used for desires, intentions, and observability in HEADSPACE. Section 3.3 introduces enhanced authorial expressivity to HEADSPACE. Finally, Section 3.4 proposes a translation algorithm capable of converting a plan produced by HEADSPACE into a graph-like model that can be used to gauge comprehension of produced plots.

3.1 Generating Narratives with Failed Actions

In stories, characters commonly attempt to perform actions that fail [61]. For example, when Thanos the Mad Titan attempts to remove half of the life in the universe by snapping his fingers in *Avengers: Endgame* [80], he's surprised when his finger snap has no effect. He realizes too late that the Infinity Stones, which he had assumed were in place along the back of the gauntlet he's wearing, were missing, removing the gauntlet's powers.

When authors include actions that fail in their stories, it is not simply due to emergent properties of complex story worlds. Quite often, characters' action failures are designed intentionally by authors for narrative effect. They build tension, prolong efforts around goal achievement, or highlight the disparities between the knowledge states of a story's characters. These functions played by action failure are central to many plot-level narrative constructs. The work we describe here seeks to outline a principled means to generate storylines with failed actions and advance a broader goal of automatically creating more expressive, natural, and compelling narratives.

One of the strengths of recent planning-based narrative generation methods (e.g., [6, 23, 69, 97, 108, 113]) is that they retain many of the benefits of classical planning while also increasing the expressive range [93] of narrative generators. One limitation of planning approaches for storyline creation is their inability to generate plans containing actions that fail. In this section, we provide the design of an algorithm for story generation that explicitly plans for character actions that fail. The algorithm uses a knowledge representation that gives context for this failure based on the limitations of characters' beliefs about the story world around them (e.g., Thanos' false belief that all the Infinity Stones were in place in his gauntlet and he had the resulting power to change the universe). The algorithm, called HEADSPACE, produces story structure with many advantageous properties found in other plan-based approaches and is more parsimonious than previous approaches to story generation that also address character belief dynamics.

As we describe below, the HEADSPACE narrative planning algorithm can generate stories where (a) agents may operate under mistaken beliefs that lead them to attempt actions that fail, and these attempted actions do not produce the expected effects; (b) agents performing actions observe the success or failure of their actions' execution; (c) agents revise their belief states in response to an observed failure as well as both passive and active sensing actions.

3.1.1 Knowledge Representation

HEADSPACE uses a PDDL-like [38] syntax for representing schematized action types in which actions are characterized in terms of *preconditions* – conditions that must obtain in the world state for the action to execute – and *effects* – conditions in the world state that change upon the action's successful execution. For efficiency, we follow the approach of Nebel and Hoffmann [65] and others and pre-compile schematized operators for a given domain into a set of ground operators representing every valid ground instantiation of a domain's act-types. We further differentiate the knowledge representation by describing both preconditions and effects related to the physical world and others that obtain in the beliefs of the character performing the action.

In the HEADSPACE knowledge representation, the set O contains all the object constants for a given domain. There is a distinguished type of object symbols called *character*, character symbols are included in a set C where $C \subseteq O$. Characters are distinguished from other objects by their ability to take action. The environment can also be represented as a special character for environmental actions.

In HEADSPACE, a *world frame* captures the sets of ground literals that can be used to characterize the world, as well as the set of symbols used to name the characters capable of taking action in the world.

Definition 3 (World Frame)**.** *A world frame is a tuple* $W = \langle GL, C \rangle$ *where* GL *is a set of positive* *ground literals and C is a set of constants, each denoting a unique character.*

A *belief state* characterizes the ground literals that a character believes to be true and false, as well as those whose truth values are unknown to the character.

Definition 4 (Belief State)**.** *Given a world frame $W = \langle GL, C \rangle$, a belief state for some character $c \in C$ is a tuple $BS_c = \langle B_c^+, B_c^-, U_c \rangle$ such that B_c^+, B_c^- and U_c together form a partition of GL, whereB_c^+ designates all the ground literals that c believes to be true, B_c^- includes all the ground literals that c believes to be false and U_c designates all the ground literals that c does not believe to be true and does not believe to be false.*

A *world state* assigns truth values to every ground literal in a world frame and provides belief state specifications for every character in a world frame.

Definition 5 (World State)**.** *Given a world frame $W = \langle GL, C \rangle$, a world state is a tuple $w = \langle T_w, F_w, BS_{c_1}, ...BS_{c_n} \rangle$ where T_w and F_w together form a partition of GL, where T_w designates all the ground literals that are true at w, F_w includes all the ground literals that are false at w and each BS_{c_i} designates the belief state for character c_i at w, where $1 \leq i \leq |C|$.*

The *epistemic goal specification* for a character represents the beliefs for the characters that should hold at the end of the story, as desired by the author.

Definition 6 (Epistemic Goal Specification)**.** *Given a world frame $W = \langle GL, C \rangle$, an epistemic goal specification for some character $c \in C$ is a tuple $\mathcal{EG}_c = \langle B_c^+, B_c^-, U_c \rangle$ such that B_c^+, B_c^- and U_c contain only elements from GL and have no common elements, where B_c^+ designates all the ground literals that c should believe to be true, B_c^- includes all the ground literals that c should believe to be false and U_c designates all the ground literals that c should not believe to be true and should not believe to be false.*

Authors can specify the desired world and character beliefs at the end of the story in the *master goal specification*.

Definition 7 (Master Goal Specification)**.** *Given a world frame $W = \langle GL, C \rangle$, a master goal specification is a tuple $\mathcal{MGS} = \langle T_w, F_w, \mathcal{EG}_{c_1}, ...\mathcal{EG}_{c_n} \rangle$ where each element of the tuple is a set that contains only elements from GL, $T_w \cap F_w = \emptyset$, where T_w designates all the ground literals that* *must be true at some goal state, F_w includes all the ground literals that must be false at some goal state and each $\mathcal{EG}_{c_i}$ designates the epistemic goal specification that must hold for character c_i at the goal state, where* $1 \leq i \leq |C|$.

A *ground operator* is a complete specification of an action in terms of the character performing the action, the conditions that must be true or false in the world for the action to execute, what the character performing the action must believe about the world for them to take the action, and how the action, once successfully executed, changes the world and the beliefs of the performing character.

Definition 8 (Ground Operator)**.** *A ground operator GOP is a tuple* $GOP = \langle c$,PRE-T,PRE-F,PRE-B^+,PRE-B^-,PRE-U,EFF-T,EFF-F,EFF-B^+,EFF-B^-,EFF-U$\rangle$ *such that*

- PRE-T,PRE-F,PRE-B^+,PRE-B^-,PRE-U,EFF-T,EFF-F,
  EFF-B^+,EFF-B^-,EFF-U $\subseteq GL$
- PRE-T $\cap$ PRE-F=PRE-B^+ $\cap$ PRE-B^- $\cap$ PRE-U=EFF-T$\cap$ EFF-F $\cap$ EFF-B^+ $\cap$ EFF-B^- $\cap$ EFF-U$= \emptyset$
- $c \in C$.

Informally, c designates the character initiating (or performing) the ground operator, PRE-T and PRE-F indicate the conditions in the world that must be true or false in order for the operator to execute, PRE-B^+,PRE-B^- and PRE-U indicate the conditions that c must believe to be true, false or unknown in the world in order for c to consider the operator executable; EFF-T, EFF-F indicate the conditions in the world that become true or false upon the action's successful execution, and EFF-B^+, EFF-B^-, EFF-U indicate the conditions that c comes to believe are true, false or unknown in the world state resulting from the operator's successful execution.

- c designates the character initiating (or performing) the ground operator.
- PRE-T indicates the conditions in the world that must be true in order for the operator to execute.
- PRE-F indicates the conditions in the world that must be false in order for the operator to execute.

- PRE-B^+ indicates the conditions that c must believe to be true in the world in order for c to consider the operator executable.
- PRE-B^- indicates the conditions that c must believe to be false in the world in order for c to consider the operator executable
- PRE-U indicates the conditions that c must neither believe to be true or false in the world for c to consider the operator executable
- EFF-T indicates the conditions that become true in the world state resulting from the operator's successful execution
- EFF-F indicates the conditions that become false in the world state resulting from the operator's successful execution
- EFF-B^+ indicates the conditions that c believes become true in the world state resulting from the operator's successful execution
- EFF-B^- indicates the conditions that c believes become false in the world state resulting from the operator's successful execution
- EFF-U indicates the conditions that c neither believes are true nor are false in the world state resulting from the operator's successful execution

We call the preconditions and effects that refer to literals that are true or false in the physical world *material* and those that specify beliefs of the character *epistemic*. In HEADSPACE, beliefs are always held by a particular agent, and only about ground literals and their truth values. There are no nested beliefs, no existential or universal quantification over beliefs, and no implications defined over beliefs.

3.1.2 Algorithm for Generating Stories with Failed Actions

Typical planning representations include a set of schematized action operators characterizing the classes of actions that can occur in a domain. In our approach, we take a set of such operators and a set of object constants and generate a world frame and a set of ground operators from them. This pre-processing is comparable to typical grounding processes used by forward-state planning algorithms (e.g., those of Nebel and Hoffmann [65]).

A *planning problem*, then, is a tuple $\mathcal{PP} = \langle W, w_0, \mathcal{MGS}, GO \rangle$ including a world frame W describing all possible ground literals and characters in a domain, an initial world state w_0 characterizing the truth values of all literals and the beliefs of all characters, a goal specification $\mathcal{MGS}$ giving a partial description of a goal world and a set of ground operators GO available for characters to execute in the domain.

An action, represented by a ground operator, is *executable* in some state w just when all its material preconditions obtain in w. We say that a ground operator is *unexecutable* in state w just when it is not executable in w. An action is *apparently executable* in some state w for a character c just when c's belief state in w supports all of the action's epistemic preconditions in w. We say that a ground operator is *apparently unexecutable* for c in state w just when it is not apparently executable for c in w.

A plan for some planning problem $\mathcal{PP} = \langle W, w_0, \mathcal{MGS}, GO\rangle$ is an ordered sequence of tuples where, for each tuple $\langle a_i, w_i\rangle$, a_i indicates the ith action in the plan (attempted in world state w_{i-1}) and w_i the state that obtains after a_i was attempted. A solution for some planning problem $\mathcal{PP} = \langle WF, w_0, \mathcal{MGS}, GO\rangle$ is a plan $\mathcal{P}$ for $\mathcal{PP}$ where, for every tuple $\langle a_i, w_i\rangle$ in $\mathcal{P}$, a_i is apparently executable in w_{i-1}, w_i is the world resulting from attempting a_i in w_{i-1}, and for a plan of length k, w_k supports $\mathcal{MGS}$.

The HEADSPACE algorithm shown in Algorithm 2 is an author-centric version of the planning algorithm. Search starts at a given initial state, and the transition from a given state to its successor states is made through the ground operators that are apparently executable by the characters in the given state. Given a world frame $W = \langle GL, C\rangle$ and a world state $w_i = \langle T_{w_i}, F_{w_i}, BS_{c_1}, ...BS_{c_n}\rangle$, the planner generates successor states for w_i in the following manner. First, the planner generates the set of all apparently executable ground operators at w_i, designated AE_{w_i}, by taking the union of all actions that appear executable in w_i to each character c_k, $1 \leq k \leq |C|$.

For every executable action in AE_{w_i}, the algorithm finds the resulting world state from executing the action at w_i. The resultant world state is computed by applying all the effects of the action. Both material effects (i.e., EFF-T,EFF-F) as well as the epistemic effects (i.e., EFF-B^+,EFF-B^-,EFF-U), with the latter applied to the belief state of the character performing the action.

For two adjacent tuples $\langle a_i, w_i\rangle$ and $\langle a_{i+1}, w_{i+1}\rangle$ in a plan P, when a_{i+1} is an executable action, we say that a_{i+1} was *executed* by c_{i+1} in w_i, resulting in w_{i+1}. For two adjacent tuples $\langle a_i, w_i\rangle$ and $\langle a_{i+1}, w_{i+1}\rangle$ in a plan P, when a_{i+1} is an unexecutable action, we say that a_{i+1}

was *attempted* by c_{i+1} in w_i, resulting in w_{i+1}.

When unexecutable actions are attempted by a character, the actions fail. We call the manner in which the planner manages this kind of action failure the planner's *failure policy*. In the current work, we define a relatively straightforward failure policy. First, with regard to *action occurrence*, none of the attempted action's effects obtain, and no material conditions in the world change. In effect, the action does not execute. Second, with respect to *failure detection*, the character executing the failed action immediately detects that it fails. Third, with respect to *local attribution*, the character executing the failed action assumes that the failure was due neither to an execution error nor to an error in the definition of the ground operator. Rather, the character assumes that the failure was due to one or more of the action's epistemic preconditions not holding in the action's world state.

Formally, an *epistemic update* occurs when an action is attempted but fails. The epistemic update creates a new world state where the material state is unchanged, but the belief states are modified: the character that attempted the action is now uncertain about the preconditions of the attempted action holding, i.e., literals in PRE-B^+,PRE-B^-,PRE-U of the attempted action are all added to the U_c belief state of the character. The character does not attribute the cause of failure to the world, but rather, to their own beliefs being inconsistent with the true state of the world. The character then becomes certain about the conditions that can be verified by passive sensing (for example, that the character is still holding the gun), but the character does not need to perform the active sensing actions immediately. The character chooses to construct a plan with their current knowledge of how they can achieve the goal. The steps that form the plan may include active sensing actions. However, it is also important to note that these steps form a plan to achieve the character's goals, and the character already has an optimal result of the sensing action along with the rest of the plan's steps in their head.

HEADSPACE has a belief update policy upon action failure, specifying that upon failing to attempt an action, the character becomes uncertain of all the preconditions of the action they attempted. Through sensing actions, characters may then ascertain truth values for these uncertain beliefs. In HEADSPACE, there are two types of sensing actions- **passive sensing** actions update beliefs immediately, whereas **active sensing** requires intentional commitment. The knowledge about the world that can be learned via passive sensing

```
    HEADSPACE($\langle GL, C \rangle$,H,Plans,MGS,GO)
 2: Using heuristic ranking function H, rank all plans in Plans. Let $P$ be the
    highest-ranked plan in Plans.
    if $P$ is a solution then
 4:   Return $P$
    else
 6:   Let $w$ be $w_k$, the world state in $kth$ (final) tuple in the plan $P$
      Let AE = $\emptyset$
 8:   for all $c \in C$ do
        Let $AE = AE \cup$ all apparently executable actions for $c$ in $w_k$
10:   end for
      for all $a \in AE$ do
12:     if $a$ is executable by $c$ in $w_k$ then
          Let $w'$ be the world state resulting from $c$ executing action $a$ in world state $w$
14:     else
          Let $w'$ be the world state resulting from $c$ attempting action $a$ in world state $w$.
16:     end if
        Append $\langle a, w' \rangle$ to the end of $P$
18:     Let $Plans = Plans \cup P$
      end for
20:   Call HEADSPACE$(WF, H, Plans, G, GO)$
    end if
```

Algorithm 2: HEADSPACE author-centric narrative planning algorithm. For Planning Problem $PP = \langle WF, w_0, G, GO \rangle$ and plan heuristic ranking function H, call HEADSPACE$(WF, H, \langle\langle \perp, w_0 \rangle\rangle, G, GO)$.

is immediately updated, whereas beliefs that need to be verified by actively sensing the world around them need the character to act in order to determine their beliefs. These additions to the narrative planning algorithm allow for producing stories where characters can potentially fail in their attempts to perform actions due to incorrect beliefs.

3.1.3 Adding a Heuristic to the HEADSPACE Planner

On line 2 of Algorithm 2, we mention using a heuristic ranking function. The heuristic allows selecting the candidate plan from a list of potential candidate plans. A planning algorithm requires a heuristic to guide its search process– approaches such as breadth-first-search or depth-first-search are inefficient and cannot effectively guide the algorithm to the solution in a large domain. We define below a heuristic for determining the next step, which extends the FastForward heuristic by Hoffman and Nebel [45], taking into account the context in HEADSPACE afforded by multiple characters, each holding distinct beliefs about the state of the world. We developed a new heuristic for the planner because of the knowledge representation that HEADSPACE uses and the ability for characters to fail or succeed when performing an action.

The FF heuristic is a well-known heuristic used in AI planning and has also been used in narrative planning [104] to get an estimated plan distance to the goal conditions specified by the author. It is similar to the HSP heuristic [10], where a relaxed plan graph approximates the distance to the goal. A search graph is a graph consisting of alternating state and action layers and is used to compute a heuristic of how close the goal is given an action choice. A relaxed plan graph is a search graph that only factors in the addition effects of an action and ignores the delete effects. This leads the search space to consider state layers that get larger with each level in the search process. HEADSPACE, like the FF planner, uses *enforced hill climbing* on top of the relaxed search process to ensure that the search is moving closer to the goal.

It is important to note that the heuristic described has previously been used in the context of executable actions, i.e., actions that can be executed in the world based on the state layer in the search process. Using the heuristic in the context of HEADSPACE directly poses a challenge due to two reasons: (1) as described, the HEADSPACE representation consists of separate individual *spaces* that represent each character's current beliefs, along with the true state of the world, and (2) existing construction processes for relaxed graph plan do not take into account the possibility of actions failing.

To address the challenges, we used the FF heuristic as a starting point and modified the relaxed plan graph construction and search process to align with the qualities of HeadSpace. The algorithm for relaxed plan graph construction is provided in Algorithm 2. There are two significant additions to the FastForward heuristic calculation. While the algorithm constructs layers similar to the original FastForward algorithm, it stores the world states and character states separately at each time step t. This allows for the relaxed plan graph to track not only the changes in the world state and compare it with the authorial goals for the world but also allows for tracking changes in character beliefs and continuing the plan construction process until the authorial goals for material conditions in the world, and the epistemic goal specifications for all characters are met (as seen in line 6). The second addition is that for any action that can possibly fail, the epistemic update due to attempting to perform the action (and failing) is also added to the relaxed plan graph (line 16). In other words, the heuristic allows a character to consider the possible effects of failing and succeeding while performing an action.

This extended relaxed plan graph construction allows for constructing a graph that extends the world state towards the authorial goals for the world, and the various character beliefs toward the authorial goals for character beliefs simultaneously, and also accounts for the possible effects of actions failing at any point. Once this relaxed plan graph is constructed, we calculate the depth of this graph. This heuristic (the **max level heuristic**) is admissible and is provided to the HeadSpace algorithm to determine the best action from the possible steps. We call the heuristic described in this section the **global heuristic**, as it considers authorial goals for the world and character beliefs.

3.1.4 Running Example: Drink Refill

To demonstrate the range of belief dynamics and the interaction between belief and execution in HeadSpace, we define a simple story domain we call the Drink Refill domain. The Drink Refill domain is used as an example domain throughout this chapter and is also one of the domains used for evaluation in Chapter 4. We describe seven operators that are part of the domain:

```
    ComputeRelaxedPlanGraph($GO, w_0, w_G, C_0, C_G, IP$)
2:  Let $l$ be the Layers to be constructed.
    Let $t = 0$
4:  $l.F_w.add(t, w_0)$
    $l.F_C.add(t, C_0)$
6:  while $L.F_C(t) \neq C_G and L.F_w(t) \neq w_G$ do
        t = t+1
8:      for all $o \in IP$ do
            if $o$ is apparently executable then
10:             $l.A.add(t, o)$
            end if
12:     end for
        $l.F_w.add(t, l.F_w(t-1))$
14:     $l.F_C.add(t, l.F_C(t-1))$
        for all $o$ in $l.A(t)$ do
16:         Let $w'$ be the relaxed world state resulting from attempting and failing to
            execute action $o$ in world state $w$.
            Let $w''$ be the relaxed world state resulting from executing action $o$
            successfully in world state $w$
18:         $l.f(t) = l.f(t) \cup w' \cup w''$
        end for
20: end while
    returnl
```

Algorithm 3: Algorithm for relaxed plan graph construction for the FastForward Algorithm. As input we accept the initial state of the world w_0, the initial belief states of the characters $C_0 = c1_0, c2_0..., cn_0$, the goal conditions for the world w_G,goals for various characters as defined by the author $C_G = c1_G, ..., cn_G$, a set of ground operators GO and the intention plans for characters $IP = IP^c1_1, IP^c1_2....$, call $ComputeRelaxedPlanGraph(GO, w_0, w_G, C_0, C_G, IP)$.

1. HOLD, where a character previously holding nothing holds an object,
2. POUR-DRINK, where a character holding a bottle can pour a drink from it,
3. CHECK-BOTTLE-EMPTY, where a character can take a closer look at the bottle that they are holding in order to determine if the bottle is empty (an active sensing action),
4. PLACE-DOWN, where a character places an object down,
5. SERVE-DRINK, where a character serves a filled drink to a customer,
6. OBSERVE-LOCAL+, where a character passively observes the location of an object that's in the same location as the character, and

7. OBSERVE-HOLDING+, where a character passively observes what they are currently holding in their hand.

The definitions for the operators are shown in Figure 3.2. While we only show the seven operators here, these are the operators part of the particular plan we examine. Section 4.2 demonstrates that this example is part of a bigger domain that can produce other plots.

For example, consider a planning problem in this domain involving a bartender refilling a thirsty customer's drink glass. An informal sketch of the planning problem's initial state sets a character, Teddy, to serve as a bartender. The goal for the story is for the drink to be refilled and served to the customer. The plan for the story is shown in Figure 3.3. In the story plan, Teddy means to hold the bottle that he believes is filled with the drink, pour a refill, and then serve it back to the customer. The plan in Figure 3.3 shows the actual executed story actions. In world state w_0, Teddy believes that Bottle 1 and 2 are not empty, the drink needs to be refilled, and he is not holding anything. His beliefs at w_0 are correct except for the fact that Bottle 1 is actually empty. Teddy first holds Bottle 1, then attempts to pour liquid into the glass, thus refilling the drink. However, because the bottle is empty, the action fails. At this point, he realizes that the action failed, and becomes uncertain about the beliefs involved in the failed action's preconditions.

Thus, in the resulting state, w_2, all epistemic preconditions for Teddy's execution of Action 2 (the first POUR-DRINK action) have been asserted as unknown in his belief model. Teddy then passively senses his location (Action 3), the location of Bottle 1 (Action 4), and the glass (Action 5). He then passively senses that he's holding Bottle 1 in his hand (Action 6). He then actively seeks new beliefs about Bottle 1 by checking for liquid in the bottle (Action 7). As a result of Action 7, Teddy comes to believe in w_7 that Bottle 1 is empty. In Action 8, he places Bottle 1 on the counter, and in Action 9, he picks up Bottle 2. In Action 10, he attempts to pour the drink from Bottle 2. Succeeding this time, the glass is now refilled. Since Teddy believes correctly in w_{10} that the drink is refilled, he serves the drink (Action 10).

3.2 Intentionality and Observability in HEADSPACE

The previous section introduced the HEADSPACE planning algorithm, a forward-directed state space search algorithm that tracks both the world state and character beliefs to produce stories that may have failed actions due to incorrect character beliefs. The algorithm presented in Algorithm 2 is author-centric: only authorial goals are considered when finding a plan. This section introduces intentionality to HEADSPACE and describes how the planner considers character desires and character intent while still producing plots that meet authorial goals. This section also introduces observability, which enables the planner to create inconsistent beliefs for characters naturally.

3.2.1 Representing Beliefs, Desires, and Intentions

An enriched representation can be used to specify character desires and how they are motivated. When a character believes that the motivations for a desire are met, they can add it to their intent. The planning algorithm allows characters to construct intention plans to achieve said desires. We connect HEADSPACE's previously described capability to manage character beliefs with this extended representation to manage dynamic intention formation and revision driven by character beliefs. The resulting representation and the algorithm that uses it are provided below.

In this work, a character's desires describe how they want the world to be. Each desire is defined relative to a given context – when character c believes that condition p holds, then c will translate their desire into an intention to make p obtain. Formally, HEADSPACE represents a desire as (a) a partial description of the world state that the character prefers, along with (b) a contextual annotation indicating when the desire could become an intention. Desires are defined in the belief space of a character. That is, the desired conditions are expressed as desired beliefs, and the context for adopting a desire as a goal is also relative to the characters' beliefs rather than material descriptions of a world state.

Definition 9 (Desire)**.** *A desire is a tuple $D = \langle c, M, \varphi(g) \rangle$ where c is a character, M contains a set of mutually consistent epistemic literals called the* motivations *for $\varphi(g)$ and $\varphi(g)$ is a single epistemic literal of the form $B_c^+(g)$, $B_c^-(g)$, or $U_c(g)$.*

HEADSPACE doesn't require that a character's desires are consistent. A character may have inconsistent motivations for the same condition and/or may desire contradictory conditions. That is, a character may have a desire $\langle c, M_1, B_c^+(g) \rangle$ and another desire $\langle c, M_2, B_c^+(g) \rangle$, where $p \in M_1$ and $\neg p \in M_2$. Further, a character may hold both the desire $\langle c, M_1, B_c^+(g) \rangle$ and the desire $\langle c, M_2, B_c^-(g) \rangle$.

Intentions are translations of desires into specific goals and plans to achieve them.

Definition 10 (Intention)**.** *An intention is a tuple* $I_{\varphi(g)} = \langle c, w, M, \varphi(g), P_{\varphi(g)} \rangle$ *where* c *is a character,* w *is the world state where the intention is adopted,* M *is a set of conditions that motivated the adoption of* $I_{\varphi(g)}$*,* $\varphi(g)$ *is a single epistemic literal of the form* $B_c^+(g)$*,* $B_c^-(g)$ *or* $U_c(g)$ *and* $P_{\varphi(g)}$ *is an intention plan for achieving* $\varphi(g)$ *that* c *believes is executable from* w*.*

While the structure of the plans held by characters as part of an intention could take many forms, we adopt a model where these plans are structured as ground partial-order, causal link (POCL) data structures comparable to those produced by typical POCL planners (e.g., [109]). We call these plans **intention plans**, as they are specific to a character and their intent.

Each intention plan has a set of steps, with an initial step s_0 whose effects are just c's belief state at the world state where the intention was adopted, and a final step s_n whose preconditions are just $\varphi_c(g)$. Each intention plan has a set of ordering constraints defining a partial order on its steps and a set of causal links of the form

Definition 11 (Intention Plan)**.** *An intention plan is a tuple* $P_{\varphi(g)} = \langle c, S, CL, O \rangle$ *where* c *is a character,* S *is set of steps* $s_1, ..., s_n$*,* CL *is list of causal links of the form* $s_i \xrightarrow{\varphi(p)} s_j$*, where* $\varphi(p)$ *indicates that* c *believes that* s_i *establishes condition* $\varphi(p)$ *as an effect needed by* s_j *as a precondition, and* O *is list of pairwise orderings between the steps.*

Each of those steps' epistemic preconditions is satisfied by causal links within the plan labeled with the epistemic effects on c's beliefs contributed by steps in the plan. Each plan P is an *apparent solution* for g by c. That is, c believes in the initial state of the plan that all the literals that act as sources for causal links in P obtain, and for every step in the plan, c's epistemic preconditions for the step are satisfied by some causal link coming from either the initial state or from a condition that c believes is an effect of some prior step in P, linked to the precondition by a causal link. Finally, c believes that P establishes $\varphi(g)$ as the precondition to its final step via some casual link $s_i \xrightarrow{\varphi(p)} s_n$.

Finally, because each intention plan P marks a plan that its character may have partially enacted, each step in P is labeled as either executed or pending. For any given world state w where the plan is intended, zero or more pending actions in P are apparently executable for c in w. We call these steps the *apparent frontier* of the intention plan.

As we discuss below, intentions are always held by characters in particular world states. Just as a belief state holds a character's beliefs, an intention state is an element of a world state used in HEADSPACE to collect a character's intentions.

Definition 12 (Intention State). *An intention state IS_c for some character c is a set of intentions of the form $I_{\varphi(g)} = \langle c, w, M, \varphi(g), P_g \rangle$ such that, for each $I_\varphi(g)$, M and $\varphi(g)$ are drawn from a motivating desire $\langle c, M, \varphi(g) \rangle$ and $P_{\varphi(g)}$ is an intention plan for c with $\varphi(g)$ as its only goal.*

In the previous section, the characterization of a world state specifies the truth value of all material literals and belief states for each character. We update the definition of world state to also include, for each character, the intention state characterizing their current goals and plans to achieve them.

Definition 13 (World State). *Given a world frame $W = \langle GL, C \rangle$, a world state is a tuple $w = \langle T_w, F_w, BS_{c_1}, ...BS_{c_n}, IS_{c_1}, ...IS_{c_n} \rangle$ where T_w and F_w together form a partition of GL, where T_w designates all the ground literals that are true at w, F_w includes all the ground literals that are false at w, each BS_{c_i} designates the belief state for character c_i at w, and each IS_{c_i} designates the intention state for character c_i at w, where $1 \leq i \leq |C|$.*

Following Grosz and Sidner's [43] terminology, when $\langle c, w, M, \varphi(g), P_{\varphi(g)} \rangle$ is in the intention state for c at world state w, we say that c *intends that* $\varphi(g)$, and that c *intends to* perform each action $a \in P_{\varphi(g)}$. For any effect e of any action a, $a \in P_{\varphi(g)}$, when e is a label on some causal link in $P_{\varphi(g)}$, we say that at w, c *intends that* e. Otherwise, we say that c considers e a *side-effect* of a.

We require a simple notion of logical consistency for characters' intentions. Specifically, for any character c and material literal g, at most one of $I_{B_c^+(g)}$, $I_{B_c^-(g)}$ or $I_{U_c(g)}$ can be in an intention state IS for c. Further, in any given intention state for c, all of c's character plans must be internally consistent. Specifically, if P_I is a plan that merges all of c current character plans by (a) creating an initial state based on c's beliefs in the intention state IS at w, (b) creating a final step containing the union of all the goals of all the plans in IS and

(c) creating causal links, orderings and steps from the corresponding components in each of the plans in IS, then P_I must be apparently executable for c at w.

3.2.1.1 Story Generation with Intention Maintenance

The plan generation algorithm for HEADSPACE shown in Algorithm 2 needs to be revised to generate intention plans. For generation of intention plans, we make the following additions and changes to HEADSPACE:

- **Specifying Characters' Desires**. Desires in HEADSPACE are a part of a planning problem specification. They're used when adopting intentions, as described in the next bullet item.

- **Having Characters Adopt Intentions**. Following the work of Riedl and Young [76], intentions in HEADSPACE are adopted as the result of a *motivating action*, a step in the plan that represents the mental act of adopting a character goal and the plan to achieve it. In HEADSPACE, we extend the definition of enabled actions to include apparently executable actions and motivating actions that mark the translation of a desire into an intention. At the point where HEADSPACE considers all enabled actions for inclusion as the next step in a plan under construction, all desires for each character are checked. For each desire whose motivations are obtained in the current world state, the planner considers the desire *motivated*.

  For each motivated desire, the planner creates a new planning problem for a micro-planner responsible for creating a plan to achieve the desired outcome. This work limits a micro-planner to constructing plans where the character is the sole agent performing actions. However, a more general approach would have the character constructing plans that contract out parts or rely on anticipated actions by other characters. The planning problem starts with a world state based on the character's current beliefs and commitment to the execution of all of their character plans. The goal state of the planning problem is just the desired outcome. If the micro-planner cannot find a solution plan for this problem, then the motivating action is not considered enabled. If a solution plan is found, then the motivating action is created and considered enabled, along with any physical actions that characters believe they can execute in the current state.

- **Having Characters Respond to Achieving Their Intentions**. Whenever a character *c* changes its beliefs in a new world state, the new belief state is compared to the goal of each intention held by *c*. If the new belief state supports the goal of an intention, then that intention is dropped (removed from the world state).

- **Restricting Characters to Act only in Pursuit of Their Goal**. As mentioned above, in HEADSPACE, any actions apparently executable by a character are considered potential actions. In HEADSPACE, we modify the process to include just those apparently executable actions at the apparent frontier of one or more of a character's plans present in the current intention state.

- **Having Characters Drop Unmotivated Intentions**. Whenever a character *c* changes its beliefs in a new world state, the new belief state is compared to the motivations for each of *c*'s intentions. If the new belief state doesn't support an intention's motivations, then that intention is dropped from the world state.

- **Having Characters Revise Their Plans When They Realize They Are Broken**. Whenever a character *c* changes its beliefs in a new world state, each of *c*'s character plans are checked. If a plan has a causal link that spans the current world state (i.e., a link whose source step has already executed and whose destination step hasn't) *and* the link's label is inconsistent with the new belief state, that indicates that the character believes the link is threatened, making the step at the destination of the link apparently unexecutable. At this point, the system uses the character's micro-planner to create a plan that achieves the intention but contains no threatened causal links. If no such plan can be found, the system drops the intention from the world state.

3.2.2 Observability

The example described in Section 3.1 demonstrates a story where Teddy fails initially to pour a drink from the bottle. In this example story, Teddy's character has an incorrect belief that the bottle was filled, whereas the world truth was that the bottle was empty. This was part of the initial state of the problem, i.e., the story starts with Teddy having inconsistent beliefs. However, in stories, we often observe that characters may also have correct beliefs that become inconsistent over the course of the narrative. For example, a character may have initially correctly believed that they had money in their bag but missed observing that another character took the money out of the bag, causing them to fail at an action in the future.

In an environment where multiple characters act toward achieving their goals, it is essential to add observability to allow characters to update their beliefs by observing the effects of actions from other agents. Moreover, an action may not have all its effects observable by other agents. For example, if a person shoots another person, only the people in the room observe the victim being shot, but it is possible for other agents in the vicinity to hear the gunshot and update *some* beliefs. To support observability, we added the ability to add observation effects in the operator definition to each effect. The implemented observability model supports direct specification of whether an effect is publicly observable, locally observable, or private. The implemented model also supports predefined axioms for observability, such as only Jedi characters can observe all shifts in the Force.

3.2.3 Algorithm

Now that we have established a representation consisting of intentions and intention plans, we describe how HEADSPACE integrates character intent with previously described authorial goals. The HEADSPACE planner performs planning on two levels:

- a top-level planner manages the world, performs intention management, tracks authorial goals, and tracks the solution to the planning problem; and
- a character-level micro-planner performs planning to compute an intention plan for a single character.

The algorithm for the top-level planner is presented as a flow diagram in Figure 3.4. The recursive algorithm accepts, as input, the planning problem. The algorithm first checks if the goals are satisfied. Then, the algorithm checks desires to see if any of them have motivations that are met in the world currently in the *Extract Arising Desires* step (Box A). If they are met, they get added to the list of intentions to track. The algorithm then updates these intentions with intentional plans in the *Adopt Intentions* step (Box B). Once all intentions are updated with relevant intention plans, the planner compiles possible actions from the apparent frontier and executes one of them in the *Select And Execute Action* step (Box C). Finally, the algorithm checks if the resulting plan was a dead end (no possible actions), an endless plan (ended up in a duplicate world state), or previously generated (by referring to previously generated plans for the same planning problem). This is done by tracking worlds visited, two fringes for other possible actions, and plans previously generated, which are all initialized to empty lists. If any of these are true, the algorithm attempts to recover to an alternate path using the fringe. There are two fringes: one at the planning problem level and one at the intention-plan level. The top-level fringe tracks other possible actions across all the intention plans, and the intention-plan level fringe tracks alternate approaches for the specific intention plan. The algorithm will recover the planning problem and recursively call the top-level algorithm with the recovered planning problem. The algorithm returns failure if the fringes are empty or no alternate plan was found with the recursive call. If the resulting plan was not a dead end or endless plan or a previously generated plan, the algorithm recursively calls the top-level planning algorithm with the plan updated with the newly added action. While the rest of the algorithm is straightforward, we describe three particular steps from Figure 3.4 in detail below.

3.2.3.1 Extraction of Arising Desires

The algorithm for extracting any desires that arise at a particular world state is provided in Algorithm 4. In this algorithm, we check that, for each possible desire, whether the corresponding character believes that the motivations are satisfied. If true, then we update the intention states by either creating a new intention, or adding the motivation to an existing intention state if one already exists with the same goal (to support narrative scenarios may multiple motivations exist for one goal).

3.2.3.2 Adoption of Intentions

The algorithm for the AdoptIntentions function is provided in Algorithm 5. In this step, for every intention, the algorithm invokes the HEADSPACE algorithm defined in Section 3.1 to find a possible intention plan. If a plan is found, the intention plan is updated in the world state. If it is not found, the intention is dropped from consideration since the character could not find a plan to achieve the desire even though the motivations were

```
1: ExtractArisingDesires($w$, desires)
2: intentions $\leftarrow \{\}$
3: for all $d \in$ desires do
4:    $character - beliefs \leftarrow$ all beliefs of character $d.character$ in$w$
5:    if $d.motivations \in character - beliefs$ then
6:       for all intention $\in$ intentions do
7:          if $d \in$ intention then
8:             intention.motivations $\leftarrow$ intention.motivations $\cup$ $d$.motivations
9:          else
10:            intentions $\leftarrow$ intentions $\cup \langle d.character, w, d.motivations, d.goals, \phi \rangle$
11:         end if
12:      end for
13:   end if
14: end for
15: return intentions
```

Algorithm 4: Algorithm that extracts any desires that arise at a worldstate w given a list of desires *desires*.

satisfied.

```
1: AdoptIntentions(plan,goal, groundedoperators,w, intentions)
2: for all $i \in intentions$ do
3:    $pp \leftarrow$ Create PlanningProblem with $w, i.goals, i.character, groundedoperators$
4:    Use character-level planner on $pp$ to find Plan $P_{\varphi(g)}$
5:    if $i \in w.intentions$ and $p \neq \emptyset$ then
6:       update the corresponding intention in $w$ to have $p$ as the plan
7:    else if $i \in w.intentions$ and $p = \emptyset$ then
8:       $w.intentions \leftarrow w.intentions - i$
9:    else if $i \notin w.intentions$ and $p \neq \emptyset$ then
10:      $w.intentions \leftarrow w.intentions \bigcup i$
11:   end if
12: end for
```

Algorithm 5: The algorithm for AdoptIntentions. It takes, as input, the plan so far, goals, grounded operator, current world state, and a list of intentions that arise at the current world state.

3.2.3.3 Select and Execute Action

The algorithm for action selection and execution is provided in Algorithm 6. Once the intention plans have been generated, the planner must compile the possible actions. This is done by adding the first unexecuted step of each intention plan to a list of possible steps. Then, the SelectAStep function decides a step to execute. Note that the current approach does not have a heuristic at the top level on which action to pick among different intention plans. The current implementation for SelectAStep arbitrarily chooses a step. Future work can consider using an authorial goal-focused decision mechanism. Once a step is selected, the planner checks to see if that action was actually executable: if the action is unexecutable, then the corresponding failed action is added to the plan. If there are any intent constraints for the failed action, they are updated as well- Section 3.3 delves deeper into this. The observability updates occur whenever the next world state is computed for an action (such as line 18 in Algorithm 6).

```
1: SelectAndExecuteAction(plan, groundedoperators, w):
2: Let possibleSteps = $\emptyset$
3: for all Intention $I_{\varphi(g)} = \langle c, M, \varphi(g), P_{\varphi(g)} \rangle$ in $w$ do
4:     Let possibleSteps $\leftarrow$ possibleSteps $\cup$ $nextSteps(c, w, P_{\varphi(g)})$.
5: end for
6: if possibleSteps $\neq \emptyset$ then
7:     Let $step$ be an action from possibleSteps
8:     for all $a \in possibleSteps$ that is not $step$ do
9:         fringe.Enqueue(current problem state $a$)
10:    end for
11:    if $step$ is executable by $c$ in $w$ then
12:        Let $w'$ be the world state resulting from $c$ executing $step$ in world state $w$
13:    else
14:        Let $w'$ be the world state resulting from $c$ attempting but failing to
15: perform $step$ in world state $w$.
16:    end if
17:    Append $\langle step, w' \rangle$ to the end of plan $\mathcal{P}$
18: end if
```

Algorithm 6: The algorithm for SelectAndExecuteAction. This step selects an action from the intention plans and adds the successful or failed version of the action to the plan.

3.2.4 A Brief Example of Intention Dynamics

To demonstrate the range of intention dynamics and the interaction between belief, intention and execution in HEADSPACE, we sketch a plan construction process for an abstracted plan in Figure 3.5. In this example, the planning problem for HEADSPACE includes an intention for character c to believe that $g1$ obtains. The intention is shown in the initial state and the character plan for the intention is given in the breakout labeled P_{g1} to the right. In subfigure a, the plan is shown with its first step (the box labeled #1) and world state (the box labeled #2) already added to the plan. In subfigure b, the second step (step

#3) from plan P_{g1} has been added to the plan. This step has the effect intended by c for P_{g1}, and also unintentionally adds beliefs to c's belief state so that a new desire is motivated (all of the desire's motivation conditions are supported in c's belief state in world state #4). As a result, an intention (for c to achieve $B_c^+(g2)$) is created; world state #4 then shows both intentions for c: intending $B_c^+(g1)$ and $B_c^+(g2)$. In subfigure c, step #5 has been added to the plan. Here, step #5 is a step performed by some character other than c. Step #5's execution changes c's beliefs in such a way that the motivations for c's intention to achieve $B_c^+(g1)$ is no longer supported. As a result, that intention is removed and no longer appears in the resulting world state #6. In subfigure d, step #7 is added to the plan, also a step executed by some other character. Step #7's execution changes c's beliefs so that c believes their plan to achieve $B_c^+(g2)$ is no longer valid. Specifically, an effect of step #7 asserts $B_c^+(x)$ threatening a causal link in P_{g2}. As a result, P_{g2} is no longer apparently executable for c in world state #7.

As a result, the micro-planner for c searches for and finds a new plan $P'_{g2}$to achieve $B_c^+(g2)$ from world state #7. This new plan is part of a replacement intention for c at world state #7 to achieve $B_c^+(g2)$.

3.2.4.1 Example Using the Drink Refill Domain

Figure 3.6 shows the Drink Refill story from Section 3.1 along with the intention plans. Passive sensing actions are not shown for brevity purposes. As mentioned earlier, the goal for the story is for Teddy, the bartender, to refill the customer's drink. We also add a desire for Teddy to have the customer's drink filled if it is empty.

In the story plan, Teddy intends to pick up Bottle 1, pour the drink, and serve it to the customer (Intention Plan 1). In the world state before the start of the plan, Teddy believes that Bottle 1 is not empty. His belief is incorrect: Bottle 1 is actually empty. Teddy picks up Bottle 1 (Action 1), then attempts to pour the drink into the customer's glass, expecting it to succeed. Because Bottle 1 is actually empty, the action fails (Action 2). At this point, Teddy realizes the action has failed.

In the world state resulting from Action 2, all epistemic preconditions for Teddy's execution of Action 2 have been asserted as unknown in his belief model. In HEADSPACE, when a character attempts an action but fails, the character immediately updates the truth values of literals that can be passively sensed. Teddy verifies that he is indeed holding the bottle and the glass is empty. Teddy now has an unknown belief about the precondition that needs to be actively sensed: whether Bottle 1 is empty.

Teddy detects that these new beliefs are inconsistent with his intention plan IP1, causing him to drop IP1 and form a new intention plan first to confirm that Bottle 1 is filled. Teddy then actively seeks new beliefs about Bottle 1's status by checking the bottle (Action 3). In the world state resulting from Action 3, Teddy finds the bottle empty. Having expected the bottle to be filled, Teddy detects that his belief is inconsistent with intention plan IP2, causing him to drop IP2 and form a new intention plan to place Bottle 1 down and use Bottle 2 instead. In Action 4, Teddy places Bottle 1 down, and in Action 5, he picks up Bottle 2. In Action 6, Teddy attempts to pour the drink again, this time from Bottle 2. Succeeding this time, the glass is now refilled. Since Teddy believes correctly that the glass has been refilled, he serves the drink to the customer (Action 7).

3.3 Enhanced Authorial Expressivity for Narrative Planning

In this chapter, we have proposed novel knowledge representation and algorithms to support the generation of plots that can consist of situations where characters fail when attempting an action when they have incorrect beliefs about the world. This increases the expressiveness and expands the space of narratives produced with computational narrative generation approaches. However, with the added expressivity in generative approaches, there is a need for enhancements in authorial expressivity to the narrative planner. With the current representation of a planning problem, authors cannot specify directly to the planner to create a story with failed actions.

To allow authors to direct the narrative planning technology to specifically construct plans with desired properties, such as failed actions and behavior of characters with intent when a failed action occurs, this section introduces enhanced authorial constraints in the form of action-related and intent constraints. Section 3.3.1 further illustrates the need for additional authorial constraints with an example. Once the motivation for this work has been set, Section 3.3.2 introduces the constraints proposed and formally defines them.

3.3.1 Motivating Example

As a motivating example, we revisit the Drink Refill domain provided in Section 3.1. In the Drink Refill domain, Teddy is a bartender who intends to refill a customer's drink using a bottle. In the specific planning problem described earlier, Teddy believes that Bottle 1 is not empty and can be used to refill the drink. However, Bottle 1 is empty. In the resulting solution plan for the specific planning problem, Teddy attempts to pour the drink from Bottle 1 and fails. This failed action leads to Teddy exhibiting behavior where he performs plan repair. He checks Bottle 1, hoping to find it filled and retry his earlier attempt. When he realizes that Bottle 1 is actually empty, he substitutes his initial strategy with a new plan, now involving Bottle 2. He finally uses Bottle 2 to pour the refill for the customer.

Let us extend this domain to allow for multiple agents, observation dynamics, and intents. We introduce an additional character, Joey, who intends to refill a different customer's drink. Consider that the action of the drink being poured cannot be publicly observed, as the bar is dimly lit. In this extended domain, Teddy and Joey are bartenders going around refilling drinks using the same set of bottles. The story's beginning has both Bottles 1 and 2 filled, and everyone has correct beliefs.

In this extended Drink Refill domain, multiple stories can be generated, shown in Figure 3.7. As observed in the figure, one possible plot is that both Teddy and Joey successfully refill their respective customers' drinks without any failed actions using different bottles **(Story A)**. However, another possible story can involve Teddy using Bottle 1 to pour his customer's drink, with Joey not observing the action or its effects. Joey could also choose Bottle 1 to pour his customer's drink, failing in this attempt **(Story B)**. Conversely, Joey could succeed and Teddy could fail instead if Teddy picked the same bottle that Joey did **(Story C)**. Both the stories with failed actions are different, and the author may want a specific character to fail or succeed.

With the current expressivity available to authors, they have limited control over the stories the planner produces. The HEADSPACE planner uses an FF-based heuristic, which prefers shorter plans. Story A would be the most optimal story and be the first one to be generated. Authors may be interested in longer, more exciting stories, with specific characters failing– the jump to hyperspace in a spaceship in *Star Wars*, or a crucial climactic moment where the villain attempts a critical act but fails [80]. They either need to keep generating stories until the planner produces the desired one or use filtering techniques to sift through the produced stories to find ones with the desired actions. Current narrative planning techniques are unable to accept specifications about desired actions.

With the current expressivity available to authors, they do not possess direct control over the intent dynamics exhibited by characters in the stories produced by the planner. Consider the situation where an author wants a character to persistently try repairing a failed action, doggedly trying different ways to capture a mouse (such as in popular cartoon shorts [72, 74]). The tenacious behavior exhibited in those stories can differ from intentional behavior, where a character gives up immediately after failing. When working with automated narrative planners, authors are unable to express such requirements to the planner directly and can only rely on specifying them indirectly through goal state constraints. Moreover, current planning technologies do not reason about character personality through intent dynamics upon action failure, though there is related work on exhibiting character personality traits and emotions [7, 88].

3.3.2 Enhanced Authorial Constraints

With the need to support additional authorial expressivity justified, we now consider the kind of expressivity to supply authors with. In this section, we outline an extension to action-related constraints proposed by Bonassi et al. [9], as well as intent-related constraints for authors to specify intent dynamics upon action failure.

3.3.2.1 Action Constraints from PAC-C

Recently, Bonassi et al.[9] proposed a PDDL-like representation to represent action constraints to a planning problem. Their compiler, PAC-C, is capable of compiling action-related constraints with operators such as *always, sometime, at most once, sometime after, sometime before, always next*, and *pattern* into a new planning problem with updated world frame, initial state, operators and goals. We use the PAC-C approach to expand the constraints for authoring further.

We extended the PAC-C algorithm for precompilation with belief-based precompilations to be compatible with the knowledge representation used by the HEADSPACE planner. The precompilation approach implemented introduces the action-constraint atoms to character beliefs in addition to true and false world knowledge. An example precompilation is demonstrated in Figure 3.8.

3.3.2.2 Failed Action Constraints

We advanced the approach by Bonassi et al. to also consider failed action constraints. The author can specify a failed action in combination with any of the action constraints mentioned above. The approach by Bonassi et al. strictly requires action formulae to be in a disjunctive non-negative form (NNF). To comply with the requirements for PAC-C, the HEADSPACE algorithm compiles a failed action specification into an *or* sequence of all the possible combinations in which an action can fail.

Definition 14 (Failed Action Constraint)**.** *A failed action constraint over an action a, where the set of failed actions $A_F = a_{F1}, ..., a_{Fn}$ represent all the possible ways the action a can fail, is compiled into an action constraint over a disjunction of actions in A_F, i.e., an action constraint over $a_{F1} \vee ... \vee a_{Fn}$.*

3.3.2.3 Intentionality-Related Constraints

As mentioned earlier, in addition to specifying these failed actions, authors often may want to specify the underlying intention plan the character is attempting. Authors may have desires for how the failed action impacts the character's intent – should the Big Bad Wolf keep pursuing the Three Little Pigs even though it keeps on failing, or should it give up upon failing the first time [50]? Should the villain keep pursuing the McGuffin to achieve their intention, or should they give up on the McGuffin and find another approach to achieve their goal instead?

Such intent dynamics are interesting for human authoring from an expressivity perspective. We enable authors to specify the intent dynamics when specifying a failed action–describing how the failed action should influence character intent. There are three possible ways to handle intent upon a failed action:

- plan repair, where the character performs plan repair to fix the reason for the failed action, and continues with the rest of the intention plan (termed in short as `PERSIST`),
- accommodation, where the character finds another plan to achieve their intention (termed as `SUBSTITUTE`),
- abandoning the intention entirely, where the character gives up the intention (termed

as DROP).

Each of the three is formally defined below.

Definition 15 (PERSIST Intent)**.** *A persistent intent on a failed action constraint for a failed action s_F defines that, for an intention plan $P_{\varphi(g)} = \langle c, S, CL, O \rangle$ where S is set of steps $s_1, ..., s_i, ..., s_n$ and s_i is the successful version of s_F, when character c encounters the failed action s_F, they strictly consider updated intention plans such that the set of steps S' for the updated intention plan includes all actions from s_i to s_n, i.e., $\{s_i, ..., s_n\} \subseteq S'$.*

Definition 16 (SUBSTITUTE Intent)**.** *A substitute intent on a failed action constraint for a failed action s_F defines that, for an intention plan $P_{\varphi(g)} = \langle c, S, CL, O \rangle$ where S is a set of steps $s_1, ..., s_i, ..., s_n$ and s_i is the successful version of s_F, when character c encounters the failed action s_F, they strictly consider updated intention plans where the set of steps S' for the updated plan does not include s_i, i.e $s_i \notin S'$.*

Definition 17 (DROP Intent)**.** *A drop intent on a failed action constraint for a failed action s_F defines that, for an intention plan $P_{\varphi(g)} = \langle c, S, CL, O \rangle$ where S is a set of steps $s_1, ..., s_i, ..., s_n$ and s_i is the successful version of s_F, when character c encounters the failed action s_F, they no longer work towards achieving the corresponding desire $\varphi(g)$ for the intent.*

The above-mentioned intention constraints only get invoked when a failed action occurs in the plan. In Algorithm 6, on line 19, the action constraints get updated when a failed action gets added to the plan. The algorithm for the *UpdateFailedActionConstraints* function is provided below, in Algorithm 7. The algorithm first checks if a failed action that occurred was part of a constraint. It then retrieves the particular constraint and checks if it has any intent constraints attached to it. The algorithm then tags the respective desire and intention plan with the intent constraint, which can be used by the plan generation mechanism in Algorithm 2 to limit the plan search accordingly. The DROP constraint is specifically added to the respective desire to prevent the desire from activating once again. Once a character gives up on an intent, HEADSPACE ignores the desire and prevents them from pursuing the desire again.

Definition 18 (Planning Problem with Enhanced Authorial Constraints)**.** *A planning prob-*

lem with enhanced authorial constraints is a tuple $\mathcal{PP} = \langle W, w_0, \mathcal{MGS}, GO, D, AC \rangle$ including a world frame W describing all possible ground literals and characters in a domain, an initial world state w_0 characterizing the truth values of all literals and the beliefs of all characters, a master goal specification $\mathcal{MGS}$, a set of ground operators GO available for characters to execute in the domain, a set of desires D that describe character desires, and a set of authorial constraints AC describing any action-related or intent constraints.

HEADSPACE compiles away the enhanced authorial constraints of the planning problem with enhanced authorial constraints. It generates new constraint atoms for the action-related constraints. These literals are added to the world frame. The literals are also updated in the initial state and goal state depending on the type of constraint. Finally, the ground operators are updated: the literals are added to the preconditions and effects (both world and epistemic) so that the actions can only be executed if the conditions are satisfied. Intent constraint tags are attached to desires or failed action constraints to be used as reference when the failed action constraint is satisfied.

3.3.3 Updating the HEADSPACE Algorithm to Consider Action Constraints and Character Intent

The HEADSPACE planning algorithm provided earlier needs to be updated with its intention plan mechanism to work with the enhanced authorial expressivity. These challenges present interesting problems to consider, which are discussed below.

The algorithm previously defined in Algorithm 2 relied solely on the global heuristic: i.e., characters determined their next action based on an FF problem that guided them to authorial goals. Conversely, a heuristic can be constructed that strictly estimates the distance to the character's desire goal ($\phi(g)$)– we define this heuristic as the **character heuristic**. The character heuristic allows for behavior guided by character desires but does not factor in the authorial goals. Moreover, the character heuristic does not guarantee the satisfaction of action constraints. Characters may commit to an action that may make the action constraint impossible to satisfy. For example, a character who desires to kill someone may pursue it by shooting them, making an action constraint that specifies that the character uses poison impossible to achieve. These reasons make it critical to consider a heuristic based on (a) global world truth to satisfy authorial constraints, (b) character

```
UpdateFailedActionConstraints(res, failedop, constraints, desires)
actionLiteral $\leftarrow \emptyset$
for all c $\in$ constraints do
    actionLiteral $\leftarrow$ actionLiteral $\vee$ c.GetMatchingActionLiteralIfExists(failedop)
end for
if actionLiteral $\neq \emptyset$ and actionLiteral.Intents.Count $> 0$ then
    if actionLiteral.Intent = PERSIST then
        for all intention $\in$ res.Item2.intentions do
            if step.Item1 $\in$ intention.plan.steps then
                Update intention.constraint to PERSIST
            end if
        end for
    else if actionLiteral.Intent = SUB then
        for all intention $\in$ res.Item2.intentions do
            if step.Item1 $\in$ intention.plan.steps then
                Update intention.constraint to SUB
            end if
        end for
    else
        for all intention $\in$ res.Item2.intentions do
            if step.Item1 $\in$ intention.plan.steps then
                Update intention.constraint to DROP
                for all d $\in$ desires do
                    if d.character = (intention.character) and d.goals = intention.goals then
                        Add the DROP constraint to d
                    end if
                end for
            end if
        end for
    end if
end if
```

Algorithm 7: UpdateFailedActionConstraints

beliefs and desires to generate plans where characters behave in accordance with their intents and knowledge, and (c) the action constraints specified in the planning problem.

The current implementation, when deciding which action to select, assigns actions priority in the following order:

- An action is part of an action constraint that has not yet been satisfied,
- If no actions are part of an action constraint, a version of the A^* heuristic function is used: $f = g + (h_C + h_G)/2$, where g is the cost so far, h_C is the character max-level FF heuristic, and h_G is the global max-level FF heuristic,
- If the heuristic function values are equal, then the action with a lower cost g is preferred,
- If there are multiple actions with the same g as well, the action with a lower character heuristic h_C is preferred.

The above decision mechanism ensures that action constraints are prioritized, followed by a heuristic that factors in weighting all the heuristics. If the weighting is equal, shorter plans are preferred, followed by a preference for what a character might consider the most reasonable action to perform.

The reasoning behind prioritizing an action with an action constraint is that, in the current approach, action constraints are not considered in relation to desires. Without prioritization, actions with action constraints are not prioritized if other candidate actions have better heuristics: the planner saves the action constraints for later. At a later stage, the character may have already achieved the desire and may no longer want to pursue the specific action even though it is only X steps away. To overcome this challenge, the current algorithm for HEADSPACE prefers adding actions with action constraints to the plan. For a discussion on this limitation with an example, see Chapter 5.

Differences between character and global heuristics may lead to interesting behavior where a character may decide to "change their mind" and perform actions later to undo them– this happens when the global heuristic takes preference and prevents *dead ends* from occurring: a character may perform a sequence of actions that are guided by the character heuristic until the global heuristic prevails and leads to a change of plan.

The updated HEADSPACE algorithm, which considers action constraints for intention plan generation and prefers actions in the above-described manner, is provided in Algorithm 8.

```
1: FFApparentSolution(w, G, $C_G$, GO, mode)
2: if w = G then
3:    return null
4: end if
5: while $solutionPlan$ is null do
6:    $tmp \leftarrow 0$
7:    if $best\_candidate$ == null then
8:       $best\_candidate \leftarrow$ null
9:       $best\_candidate\_character\_heuristic \leftarrow \infty$
10:      $best\_candidate\_global\_heuristic \leftarrow \infty$
11:      $best\_candidate\_cost \leftarrow \infty$
12:      $n \leftarrow$ all apparently executable next actions and resulting states in $w$
13:      for all $op$, $next\_state \in n$ do
14:         $current\_node\_character\_heuristic \leftarrow$ compute RPG using $next\_state$ and $C_G$ and calculate the FFmax heuristic
15:         if $op$ is executable at $w$ then
16:            $current\_node\_global\_heuristic \leftarrow$ compute RPG using $next\_state$ and $G$ and calculate the FFmax heuristic
17:         else
18:            $failedop \leftarrow$ appropriate failed version of $op$ in $GO$
19:            $next\_state \leftarrow$ resulting worldstate on attempting $failedop$ at $w \cup$ resulting character beliefs on executing $op$ at $w$
20:            $current\_node\_global\_heuristic \leftarrow$ compute RPG using $next\_state$ and $G$ and calculate the FFmax heuristic
21:         end if
22:         if $current\_node\_character\_heuristic \neq \infty$ then
23:            Compare $best\_candidate$ and $next\_state$ and select action
24:            if $next\_state$ is selected then
25:               Create a $fringe\_element$ as a tuple containing $best\_candidate$
26:               Enqueue $fringe\_element$ into $character\_fringe$ with a priority tuple of ($best\_candidate\_character\_heuristic$, $best\_candidate\_global\_heuristic$, $best\_candidate\_cost$)
27:               $best\_candidate \leftarrow next\_state$
28:               $best\_candidate\_global\_heuristic \leftarrow current\_node\_global\_heuristic$
29:               $best\_candidate\_character\_heuristic \leftarrow current\_node\_character\_heuristic$
30:               $best\_candidate\_cost \leftarrow$ current cost
31:            end if
32:         end if
33:      end for
34:   end if
```

Algorithm 8: FFApparentSolution Function

```
35:   if dead end, endless plan, or plan previously generated then
36:     recover from character_fringe and continue to the next iteration
37:   end if
38:   Add $\langle op, best\_candidate \rangle$ to intention_plan
39:   best_candidate $\leftarrow$ null
40:   if best_candidate = G then
41:     if (mode = PERSIST || mode = SUBSTITUTE) then
42:       Check actions of intention_plan to see if they match the respective intent
        constraint conditions
43:       Recover from fringe if plan does not meet constraints
44:     end if
45:     if (mode equals REGENERATE) then
46:       Check plan to ensure new plan is different from previous plans
47:     else
48:       Recover from fringe
49:     end if
50:     solutionPlan $\leftarrow$ current
51:     return solutionPlan
52:   end if
53: end while
54: return solutionPlan
```

3.4 Translating Stories with Failed Actions into QUEST Knowledge Structures

In this chapter, so far we have proposed a narrative planning process that can generate stories where characters can fail when attempting an action if they have incorrect beliefs. The narrative planning process has been integrated into a model that also considers character intent, observability, and enhanced authorial constraints when generating plots.

Before we consider how the added expressivity provided by HEADSPACE can be used by authors to produce stories, it is important to evaluate human comprehension of the stories produced themselves. Cognitive model evaluations in the narrative planning community have been widely used to gauge comprehension of computationally generated narratives [14, 51]. These evaluation approaches usually involve humans watching, playing through, or reading a narrative, with the hypotheses focused on comparing the generated story with the mental model of the story story built by the study participants.

One such cognitive model, called QUEST [41, 42], has been developed and validated as a cognitive model of human question answering in the context of stories. The model uses a graph-form knowledge representation – called a QUEST Knowledge Structure, or QKS – to characterize the relationships between events, goals and intentions that underlie a story. In the QUEST model, the arc distance between any two nodes in a QKS predicts how well the second node serves as an answer for *why*, *how*, or *in-order-to* questions about the first node. The model has been used by the narrative planning community to evaluate their planning approaches [6, 20, 75, 106]. In these methods, plan data structures are translated into the data structures representing a cognitive model of narrative comprehension. Experiments compare the way humans process the generated stories to the predictions made by the cognitive models.

One limitation of existing translation algorithms used to generate QUEST graph structures from narrative planning output is that they have only focused on positive actions in the plan [14, 20] (e.g., actions that are attempted by characters and execute correctly in the story world). New algorithms, such as HEADSPACE, that create narratives with greater expressive range, have prompted consideration of adapting or extending evaluation methods, including those centered around user experience. In particular, a need has arisen because of increased expressivity around failed actions and mistaken beliefs [71, 100], two narrative elements not accounted for in prior evaluative methods. Evaluation methods that adopt a cognitive stance require new knowledge representations that include failed actions in their model.

While not used by previous approaches of evaluation, the original QUEST Knowledge Structure (QKS) theories consider the possibility of failed events and unachieved goals appearing in the story discourse. To date, however, no evaluative approaches that leverage QKSs have taken advantage of these elements to gauge the effectiveness of stories with failed actions. This section builds upon the character of QKSs containing failed actions and their semantics as defined by the original QUEST work, which were described in Section 2.3.1. We present an automatic translation algorithm to translate from a modified Planning Domain Definition Language (PDDL) representation [32] into QKS that can be applied in evaluation strategies. We also provide a walkthrough using an example translation of a narrative plan to its corresponding QKS.

3.4.1 Towards the Automatic Creation of QUEST Knowledge Structures from Plans that Include Failed Actions

Our proposed algorithm, shown in Algorithm 9, addresses the above issues and allows for the generation of new types of QKS structures that are both faithful to Graesser and Clark's definitions and also incorporate failed actions effectively.[1] When generating a QKS with failed events, the proposed algorithm requires a knowledge representation that includes both the story plan and various intention plans for each character.

Our translation algorithm incorporates the intention plan of the character as a chain of goals with reason arcs so that forward goal relationships are maintained from failed actions. The intention plan incorporated in this algorithm does not stop when a goal is not achieved because of a failed action. This is consistent with Graesser's system, as questions such as "what are the consequences of" solely look at achieved goals, not unachieved goals, while questions like "why" can traverse reason arcs of both achieved and unachieved goals.

Examples of how candidate answers for specific questions can be computed using the output from our algorithm can be found in Section 3.4.2.2.

3.4.2 Example Translation: Escape Scenario

The escape scenario is a simple domain and problem constructed to help describe how the QKS generation algorithm works. In the escape scenario, there is an agent in a room with two doors: the west door and the east door. The agent is on the west side of the room. The agent believes that both doors are unlocked, but in reality only the east door is unlocked. The agent is trying to escape from the room and either door is suitable for the escape. The agent only has three actions: moving from one side of the room to the other, opening unlocked doors, and escaping through an open exit door. The full story plan and all intention plans are shown in Figure 3.9.

When the planner constructs the agent's first intention plan, it creates the simple plan of opening the west door and then escaping through the door. This is consistent with the agent's (mistaken) beliefs indicating the west door is unlocked. Steps from this plan

[1]Our worked example shows a plan where characters' actions fail due to incorrect beliefs.In this plan, the characters' beliefs are automatically updated when they observe their own action failures. This is consistent with plans produced by HEADSPACE, but is not a required aspect of the translation algorithm we present here.

```
Data: A plan P $(s, B, \prec, L, I)$
Result: A QUEST Knowledge Graph
Create a total ordering for all the steps in P;
for every step $s_i$ in $P$ do
    Create event node $\varepsilon_i$ for $s_i$;
    for each effect $e_j$ in $s_i$ do
        Create a State node $\sigma_{e_j}$ ;
        Link it to $\varepsilon_i$ with a Consequence arc $\varepsilon_i \xrightarrow{c} \sigma_j$;
    end
end
$k \leftarrow -1$;
for each Intention Plan $\varphi_i$ in $I$ do
    $active \leftarrow true$;
    for each step $s_j$ in $\varphi_i$ do
        Create a goal node $\Upsilon_{ij}$;
        if $active$ then
            Link $\Upsilon_{ij}$ to event node $\varepsilon_k$ using an Outcome arc $\Upsilon_{ij} \xrightarrow{O} \varepsilon_k$;
            if $\varepsilon_k$ was a failed event then
                $active \leftarrow false$;
            end
            $k \leftarrow k + 1$;
        end
    end
    for each causal link $s_l \xrightarrow{p} s_m$ in $\varphi_i$ do
        Link the corresponding goal nodes with a Reason arc $\Upsilon_{il} \xrightarrow{R} \Upsilon_{im}$;
    end
end
for each failed event $\varepsilon_i$ do
    if there exists a causal link $s_i \xrightarrow{p} s_j$ then
        Connect the state node for p $\sigma_p$ to the goal node connected to $\varepsilon_j$ with an
          Initiate arc $\sigma_p \xrightarrow{I} \Upsilon_{xj}$;
    end
    else
        Connect $\varepsilon_i$ to the event node $\varepsilon_j$ for the agent's next step based on the total
          ordering with an Initiate arc $\varepsilon_i \xrightarrow{I} \varepsilon_j$;
    end
end
For each causal link $s_i \xrightarrow{p} s_j$ in P, link the corresponding state node $\sigma_p$ to the event
  node $\varepsilon_j$ with a Consequence arc $\sigma_p \xrightarrow{c} \varepsilon_j$;
```

Algorithm 9: Translation algorithm for creating a QKS from a Plan

are added to the story plan up to the point where the first action fails (in this case, at the agent's first action, Step 8). At this point, the agent's mistaken beliefs are updated and its inconsistent intention plan is discarded in favor of a new one consistent with its updated beliefs. Because that intention plan has no failed actions, all its steps are added to the story plan. The final plan has four steps: the agent attempts to open the west door and fails, the agent moves to the east location, the agent opens the east door, and then the agent escapes.

3.4.2.1 Constructing a QKS

To create a QKS corresponding to the plan in Figure 3.9, the translation algorithm begins by creating an event node for every step in the story plan. This results in four events corresponding to the four steps described above (line 3). The next part of the algorithm (lines 4–7) generates a state node for every effect of each event along with a consequence link from the event to the generated state node. For example, the event of the agent moving to the east side is linked with a consequence arc to a new state node for `At(agent, east-loc)`. Similarly, the event for failing to open the west door causes a belief update of `Locked(west-door)` for the agent, so a state node is created for that belief update and it is linked from the event node via a consequence arc.

Once all events are added and their effects are represented by state nodes connected from consequence arcs, the algorithm iterates through every intention plan in order that was used to generate the story plan (line 10) while considering each event node that it had added on line 3. The first intention plan was to open the west door and then escape. The algorithm creates a goal for opening the west door (line 13). It then links this goal with an outcome to the event of failing to open the west door (line 15). This intention plan is then flagged as having a failing outcome (line 17), so the remaining steps in the intention plan (in this case, just the escape through the west door) are only added as goals. Once all goals for the intention plan are added, causal links between the steps in the intention plan are translated as reason arcs between goals (line 23). In this case, the ”Open West Door” goal links with a reason arc to the ”Escape West” goal.

The algorithm proceeds to the next intention plan: to move to the east door, open the east door, and then escape through the east door. For each of these steps it creates a goal and then links the goal with an outcome link to the corresponding event in the final plan.

Afterward, it translates any causal links in the intention plan into reason arcs for the goal nodes.

The next step of the translation is to create initiate arcs from failed actions to the first goal of the next intention plan (lines 26–33). These initiate arcs make it clear that the next goal was a product of what happened previously (the observation by a character of a failed action), not a necessary step based on the previous goal, which is why the new intention plan's goal does not have a reason arc from the previous goal corresponding to the failed action. Instead, if the failed step has a causal link to the next step, an initiate arc is created from the failed step's condition that the causal link depends on to the goal of the next step. If no such causal link exists, the goal of the failed event points to the goal of the next step with an initiate arc. In this example, failing to open the west door initiates the goal of moving to the east, as there is no causal link from the failing step to the next step in the plan.

Finally, all causal links in the final plan are represented by adding consequence arcs from the effect of the event that was derived from the source step of the causal link to the event that was derived from the sink step of the causal link (line 34). In our example, one instance of this is the consequence arc from the state `At(agent, east-loc)` to the event corresponding to opening the east door.

The final QKS for this example is presented in Figure 3.10.

3.4.2.2 Escape Scenario Goal Questions

As described in Section 3.4.1, The QUEST model is capable of taking a QKS and a node from that QKS and returning sets of logically appropriate nodes that can serve as answers to ”Why,” ”How,” or ”What are the consequences of” questions about the input node. It can also provide relative rankings of goodness of answer within each set. We provide short descriptions below for the processes used in QUEST to determine these sets, including for QKSs that contain failed actions.

- **Why:** ”Why” questions about goals start with a goal and traverse reason arcs forward, manner arcs backward, sibling nodes from forward before-arcs, goal initiator arcs backwards, and consequence arcs from the start of those initiator arcs backwards. Among these arcs, we solely include reason arcs, initiator arcs, and consequence arcs; manner arcs do not exist in our system. Our QKS can provide goodness of answer ratings for ″why″ questions such as:

  – *Why does the character want to open the west door?* The character wants to open the west door in order to escape to the west.
  – *Why does the character move to the east door?* The character moves to the east door because they failed to open the west door. The character failed to open the west door because they believed the west door was unlocked.

- **How:** ″How″ questions traverse backward through reason arcs and forward through manner arcs using **achieved** goals. Since our translation algorithm doesn't generate manner arcs, we solely rely on reason arcs. The main benefit of our translation algorithm is in avoiding invalid answers to ″How″ questions. Namely, by ensuring there is no reason arc from the ″Open West Door″ goal to the ″Move to East Door″ goal, we avoid including an answer like ″The character wants to open the west door in order to move to the east door″ for a question like *How does the character want to move to the east door?* Such a ″how″ question response would be included if, for example, we used Christian and Young's [20] algorithm on a plan with failed actions.
- **What are the consequences of:** The arc search procedure for these types of questions can only progress through *achieved* goals and are answered by traversing through forward reason arcs, backward manner arcs, forward consequence arcs, forward implies arcs, forward outcome arcs, and forward initiate arcs. In our system, the only arcs of relevance are the forward reason, consequence, outcome, and initiate arcs. This allows the QKS to generate sets of answers for questions like:
  – *What are the consequences of the character failing to open the west door?* The character wants to move to the east door because they fail to open the west door.

As the expressive range of narrative generation systems increases, existing evaluation methods can be expanded to account for novel plot/plan structures with minimal impact on previous well-founded experimental designs. This section highlights previously unused aspects of the QUEST cognitive model that align with novel features of stories (i.e., failed actions) being generated in recently developed planning systems. We define a translation algorithm that takes as input a story plan containing failed actions and creates a knowledge structure consistent with QUEST's QKS definitions. These knowledge structures can then be integrated into human subjects experiments comparing user experience during plan comprehension with the plan structures service as the plot line's source. This translation algorithm is used as part of the experimental evaluation described in Section 4.2.

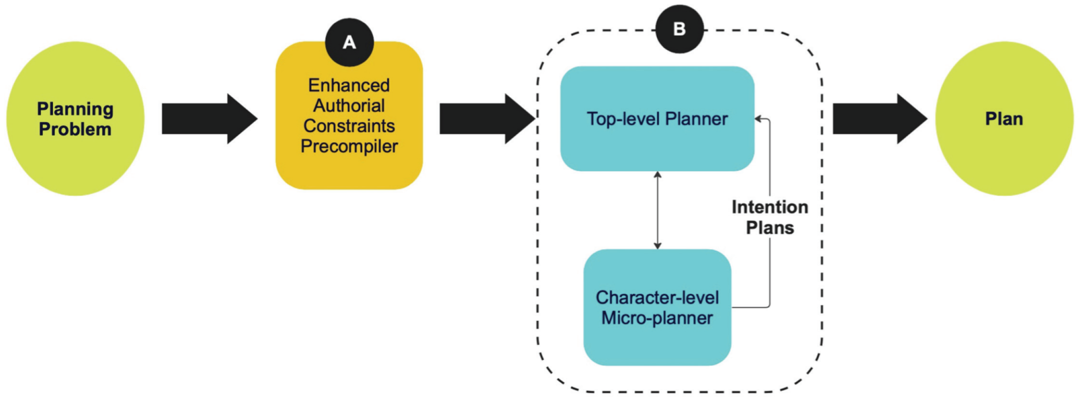

Planning Problem
A
Enhanced Authorial Constraints Precompiler
B
Top-level Planner
Intention Plans
Character-level Micro-planner
Plan

Figure 3.1. An overview of the HEADSPACE planner. Box A, the Enhanced Authorial Constraints Precompiler, is described in Section 3.3.2. Box B, which includes the top-level planner and the character-level planner, is described in Section 3.2.

Hold(T, B1)	
PRE-F	holding(T, B1)
	holding(T, B2)
PRE-B-	holding(T, B1)
	holding(T, B2)
EFF-T	holding(T, B1)
EFF-B+	holding(T, B1)

Place-Down(T, B1)	
PRE-T	holding(T, B1)
PRE-B+	holding(T, B1)
EFF-F	holding(T, B1)
PRE-B-	holding(T, B1

*Observe-Local+(T, B1, B)	
PRE-T	at(T, B)
	at(B1, B)
PRE-B+	at(T, B)
PRE-U	at(B1, B)
EFF-B+	at(B1, B)

Serve-Drink(T, G)	
PRE-F	empty(G)
	served(G)
PRE-B-	empty(G)
	served(G)
EFF-T	served(G)
EFF-B+	served(G)

*Observe-Local+(T, G, B)	
PRE-T	at(T, B)
	at(G, B)
PRE-B+	at(T, B)
PRE-U	at(G, B)
EFF-B+	at(G, B)

*Observe-Local+(T, T, B)	
PRE-T	at(T, B)
PRE-U	at(T, B)
EFF-B+	at(T, B)

Pour-Drink(T, B1)	
PRE-T	holding(T, B1)
	empty(G)
PRE-F	empty(B1)
PRE-B+	holding(T, B1)
	empty(G)
PRE-B-	empty(B1)
EFF-T	empty(B1)
EFF-F	empty(G)
EFF-B+	empty(B1)
EFF-B-	empty(G)

Hold(T, B2)	
PRE-F	holding(T, B1)
	holding(T, B2)
PRE-B-	holding(T, B1)
	holding(T, B2)
EFF-T	holding(T, B2)
EFF-B+	holding(T, B2)

Pour-Drink(T, B2)	
PRE-T	holding(T, B2)
	empty(G)
PRE-F	empty(B2)
PRE-B+	holding(T, B2)
	empty(G)
PRE-B-	empty(B2)
EFF-T	empty(B2)
EFF-F	empty(G)
EFF-B+	empty(B2)
EFF-B-	empty(G)

Check-Bottle-Empty(T, B1)	
PRE-T	holding(T, B1)
PRE-F	empty(B1)
PRE-B+	holding(T, B1)
PRE-U	empty(B1)
EFF-B-	empty(B1)

Figure 3.2. Ground operators used in the Drink Refill domain. Here, object constants have been abbreviated to preserve space. Throughout, we use T for Teddy, B for Bar, B1 for Bottle 1, B2 for Bottle 2, and G for the glass.

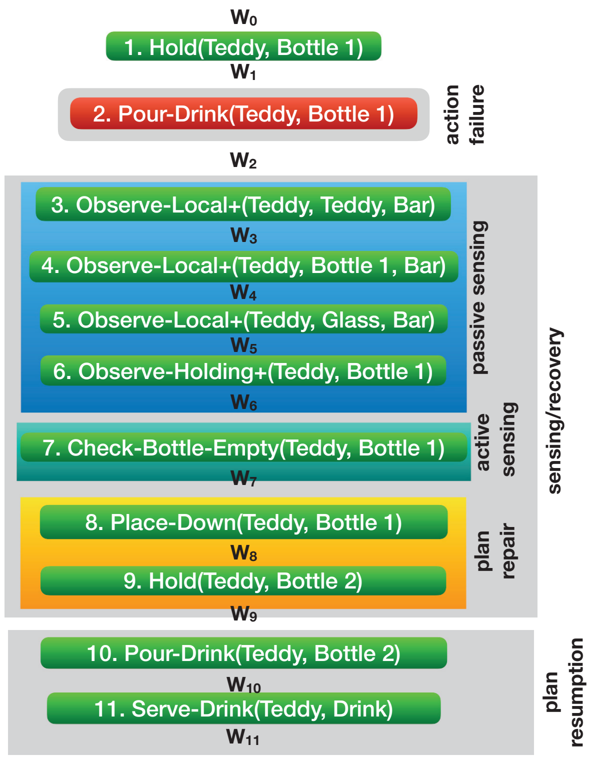

W0
1. Hold(Teddy, Bottle 1)
W1
2. Pour-Drink(Teddy, Bottle 1)
action failure
W2
3. Observe-Local+(Teddy, Teddy, Bar)
W3
4. Observe-Local+(Teddy, Bottle 1, Bar)
W4
5. Observe-Local+(Teddy, Glass, Bar)
W5
6. Observe-Holding+(Teddy, Bottle 1)
W6
passive sensing
7. Check-Bottle-Empty(Teddy, Bottle 1)
W7
active sensing
8. Place-Down(Teddy, Bottle 1)
W8
9. Hold(Teddy, Bottle 2)
W9
plan repair
sensing/recovery
10. Pour-Drink(Teddy, Bottle 2)
W10
11. Serve-Drink(Teddy, Drink)
W11
plan resumption

Figure 3.3. A solution plan for the Drink Refill domain's planning problem. Green actions are successfully performed actions. Red actions are ones that are attempted but that fail because their material preconditions are not all met in the world state where they are attempted.

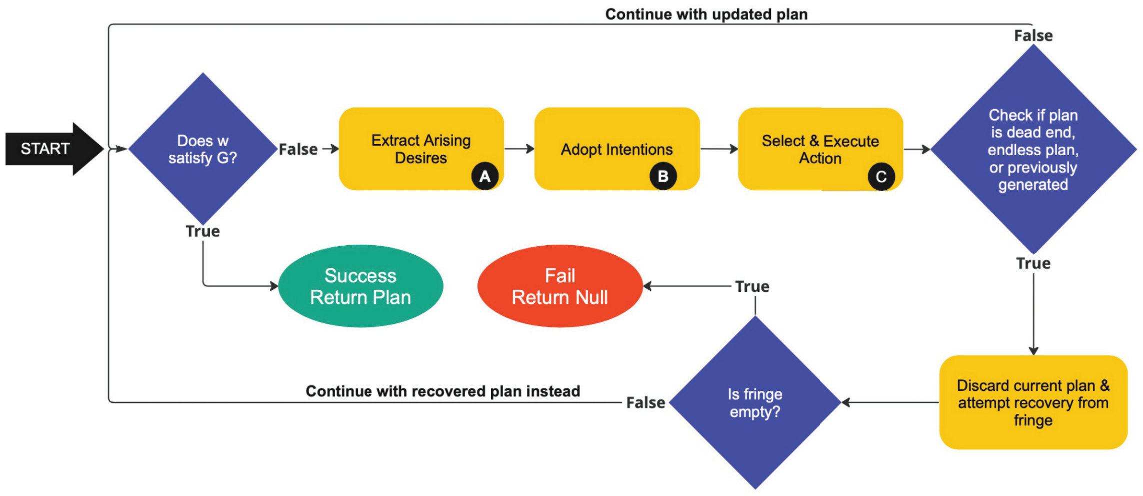
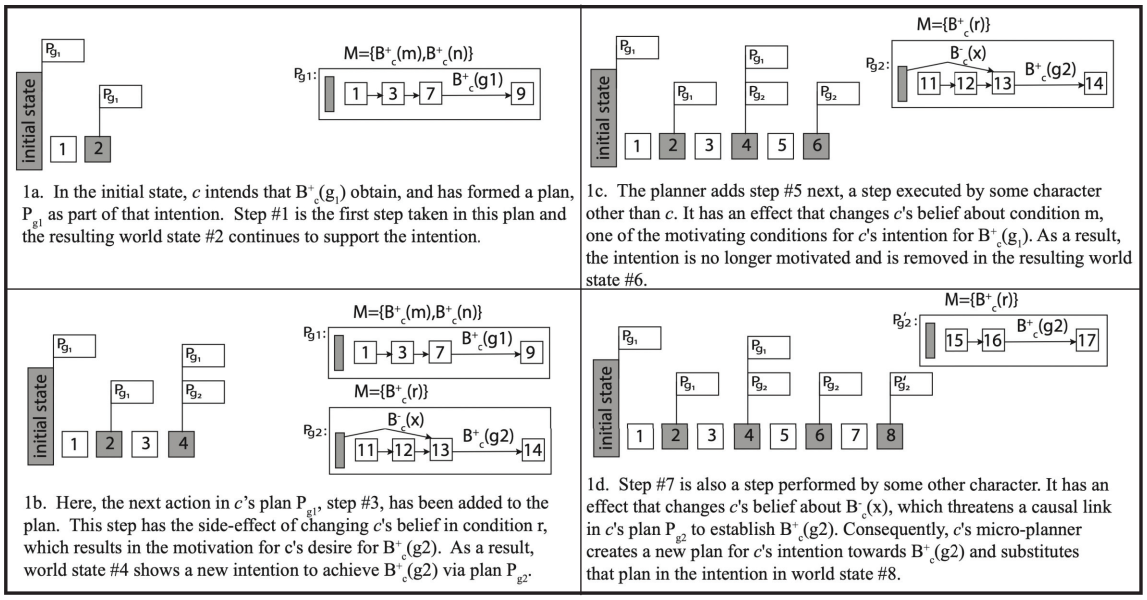

Continue with updated plan
False
START
Does w satisfy G?
False
Extract Arising Desires
A
Adopt Intentions
B
Select & Execute Action
C
Check if plan is dead end, endless plan, or previously generated
True
Success Return Plan
Fail Return Null
True
True
Discard current plan & attempt recovery from fringe
Is fringe empty?
False
Continue with recovered plan instead

Figure 3.4. The algorithm for the HEADSPACE planner that integrates intentionality and observability.



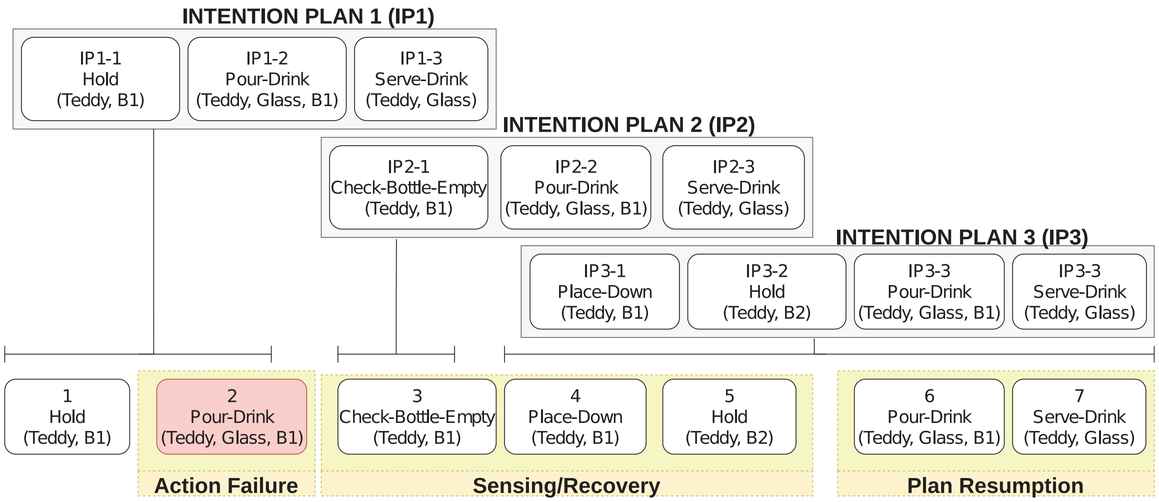
1a. In the initial state, *c* intends that $B^+_c(g_1)$ obtain, and has formed a plan, P_{g1} as part of that intention. Step #1 is the first step taken in this plan and the resulting world state #2 continues to support the intention.

1b. Here, the next action in *c*'s plan P_{g1}, step #3, has been added to the plan. This step has the side-effect of changing *c*'s belief in condition r, which results in the motivation for c's desire for $B^+_c(g2)$. As a result, world state #4 shows a new intention to achieve $B^+_c(g2)$ via plan P_{g2}.

1c. The planner adds step #5 next, a step executed by some character other than *c*. It has an effect that changes *c*'s belief about condition m, one of the motivating conditions for *c*'s intention for $B^+_c(g_1)$. As a result, the intention is no longer motivated and is removed in the resulting world state #6.

1d. Step #7 is also a step performed by some other character. It has an effect that changes *c*'s belief about $B^-_c(x)$, which threatens a causal link in *c*'s plan P_{g2} to establish $B^+_c(g2)$. Consequently, *c*'s micro-planner creates a new plan for *c*'s intention towards $B^+_c(g2)$ and substitutes that plan in the intention in world state #8.

Figure 3.5. A schematic representation of the planning process for a problem in HEADSPACE. In the figures above, squares with white backgrounds indicate actions in the plan, while squares with gray backgrounds indicate world states in the plan. Rectangles indicate character plans that are held as part of intentions in some world states. Insets show additional plan structure for character plans labeled P_{g1}, P_{g2}, and P'_{g2}



INTENTION PLAN 1 (IP1)
IP1-1 Hold (Teddy, B1)
IP1-2 Pour-Drink (Teddy, Glass, B1)
IP1-3 Serve-Drink (Teddy, Glass)
INTENTION PLAN 2 (IP2)
IP2-1 Check-Bottle-Empty (Teddy, B1)
IP2-2 Pour-Drink (Teddy, Glass, B1)
IP2-3 Serve-Drink (Teddy, Glass)
INTENTION PLAN 3 (IP3)
IP3-1 Place-Down (Teddy, B1)
IP3-2 Hold (Teddy, B2)
IP3-3 Pour-Drink (Teddy, Glass, B1)
IP3-3 Serve-Drink (Teddy, Glass)
1 Hold (Teddy, B1)
2 Pour-Drink (Teddy, Glass, B1)
3 Check-Bottle-Empty (Teddy, B1)
4 Place-Down (Teddy, B1)
5 Hold (Teddy, B2)
6 Pour-Drink (Teddy, Glass, B1)
7 Serve-Drink (Teddy, Glass)
Action Failure
Sensing/Recovery
Plan Resumption

Figure 3.6. The Drink Refill plan with Teddy's intention plans shown. The red action was attempted but failed because the non-belief preconditions are not all met in the world state where they are attempted. Passive sensing actions are not shown for brevity.

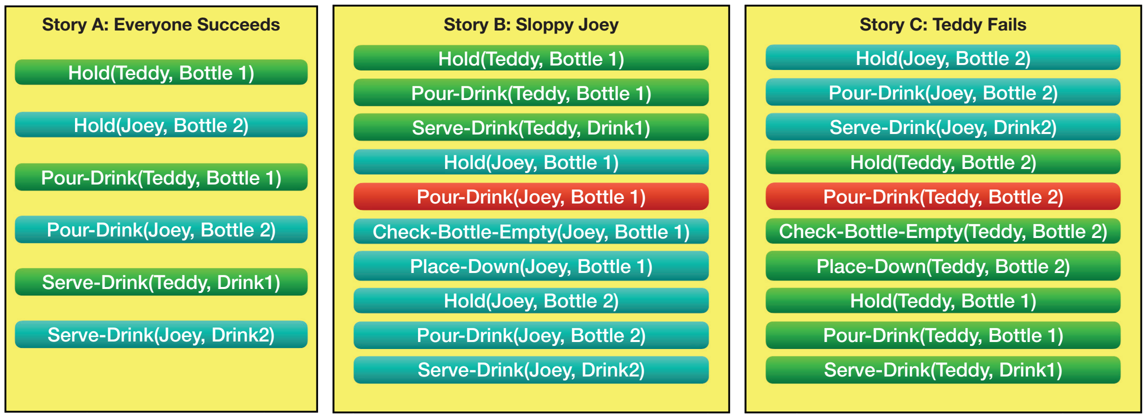

Story A: Everyone Succeeds
Hold(Teddy, Bottle 1)
Hold(Joey, Bottle 2)
Pour-Drink(Teddy, Bottle 1)
Pour-Drink(Joey, Bottle 2)
Serve-Drink(Teddy, Drink1)
Serve-Drink(Joey, Drink2)
Story B: Sloppy Joey
Hold(Teddy, Bottle 1)
Pour-Drink(Teddy, Bottle 1)
Serve-Drink(Teddy, Drink1)
Hold(Joey, Bottle 1)
Pour-Drink(Joey, Bottle 1)
Check-Bottle-Empty(Joey, Bottle 1)
Place-Down(Joey, Bottle 1)
Hold(Joey, Bottle 2)
Pour-Drink(Joey, Bottle 2)
Serve-Drink(Joey, Drink2)
Story C: Teddy Fails
Hold(Joey, Bottle 2)
Pour-Drink(Joey, Bottle 2)
Serve-Drink(Joey, Drink2)
Hold(Teddy, Bottle 2)
Pour-Drink(Teddy, Bottle 2)
Check-Bottle-Empty(Teddy, Bottle 2)
Place-Down(Teddy, Bottle 2)
Hold(Teddy, Bottle 1)
Pour-Drink(Teddy, Bottle 1)
Serve-Drink(Teddy, Drink1)

Figure 3.7. The various stories that satisfy the planning problem for the extended Drink Refill domain. Blue actions are executed by Joey; green actions are executed by Teddy. Actions in red are failed actions. Story A is the shortest story that can be produced, Story B has Joey failing to pour from Bottle 1, and Story C demonstrates Teddy failing to pour from Bottle 2.

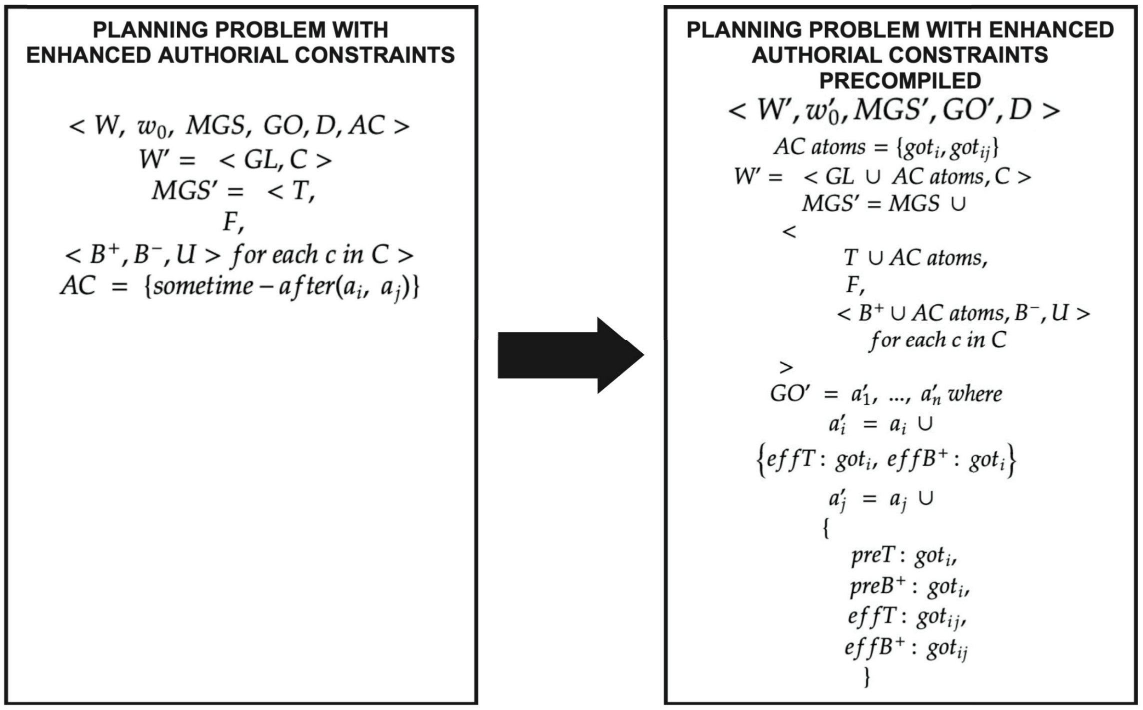

PLANNING PROBLEM WITH ENHANCED AUTHORIAL CONSTRAINTS
$< W, w_0, MGS, GO, D, AC >$
$W' = \ < GL, C >$
$MGS' = \ < T,$
$F,$
$< B^+, B^-, U > for\ each\ c\ in\ C >$
$AC = \{sometime-after(a_i,\ a_j)\}$
PLANNING PROBLEM WITH ENHANCED AUTHORIAL CONSTRAINTS PRECOMPILED
$< W', w'_0, MGS', GO', D >$
$AC\ atoms = \{got_i, got_{ij}\}$
$W' = \ < GL \cup AC\ atoms, C >$
$MGS' = MGS \cup$
$<$
$T \cup AC\ atoms,$
$F,$
$< B^+ \cup AC\ atoms, B^-, U >$
$for\ each\ c\ in\ C$
$>$
$GO' = a'_1, ..., a'_n\ where$
$a'_i = a_i \cup$
$\{effT: got_i,\ effB^+: got_i\}$
$a'_j = a_j \cup$
$\{$
$preT: got_i,$
$preB^+: got_i,$
$effT: got_{ij},$
$effB^+: got_{ij}$
$\}$

Figure 3.8. An example compilation for a planning problem with the "sometime after" action constraint, based on the approach proposed by Bonassi et al. [9]. In this example, the "sometime-after" constraint gets compiled away with updated operators to include new literals as preconditions and effects, and updated goal to also include the new literals.

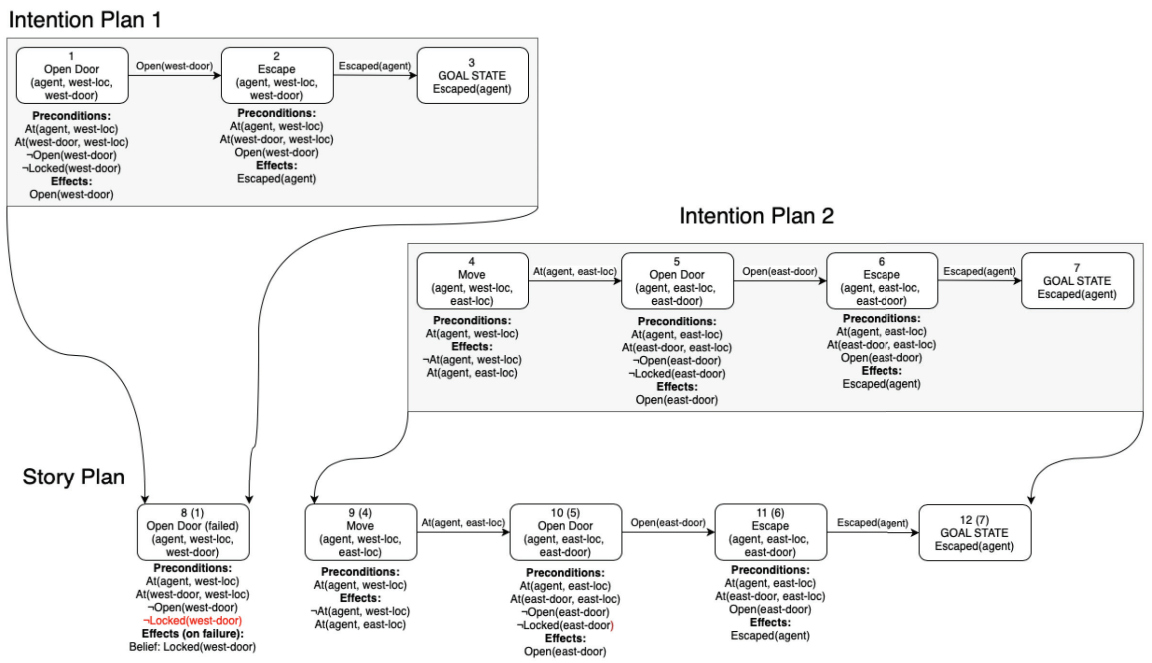

Intention Plan 1
1 Open Door (agent, west-loc, west-door)
Preconditions: At(agent, west-loc) At(west-door, west-loc) ¬Open(west-door) ¬Locked(west-door) Effects: Open(west-door)
Open(west-door)
2 Escape (agent, west-loc, west-door)
Preconditions: At(agent, west-loc) At(west-door, west-loc) Open(west-door) Effects: Escaped(agent)
Escaped(agent)
3 GOAL STATE Escaped(agent)
Intention Plan 2
4 Move (agent, west-loc, east-loc)
Preconditions: At(agent, west-loc) Effects: ¬At(agent, west-loc) At(agent, east-loc)
At(agent, east-loc)
5 Open Door (agent, east-loc, east-door)
Preconditions: At(agent, east-loc) At(east-door, east-loc) ¬Open(east-door) ¬Locked(east-door) Effects: Open(east-door)
Open(east-door)
6 Escape (agent, east-loc, east-coor)
Preconditions: At(agent, east-loc) At(east-door, east-loc) Open(east-door) Effects: Escaped(agent)
Escaped(agent)
7 GOAL STATE Escaped(agent)
Story Plan
8 (1) Open Door (failed) (agent, west-loc, west-door)
Preconditions: At(agent, west-loc) At(west-door, west-loc) ¬Open(west-door) ¬Locked(west-door) Effects (on failure): Belief: Locked(west-door)
9 (4) Move (agent, west-loc, east-loc)
Preconditions: At(agent, west-loc) Effects: ¬At(agent, west-loc) At(agent, east-loc)
At(agent, east-loc)
10 (5) Open Door (agent, east-loc, east-door)
Preconditions: At(agent, east-loc) At(east-door, east-loc) ¬Open(east-door) ¬Locked(east-door) Effects: Open(east-door)
Open(east-door)
11 (6) Escape (agent, east-loc, east-door)
Preconditions: At(agent, east-loc) At(east-door, east-loc) Open(east-door) Effects: Escaped(agent)
Escaped(agent)
12 (7) GOAL STATE Escaped(agent)

Figure 3.9. The character's intention plans and the final story plan for the "Escape" scenario. Steps are indicated using rounded rectangles and arcs between steps indicate causal links, with labels on those links indicating the condition connecting the source and sink steps. Preconditions for each action are showing immediately below the action, and preconditions that match a character's mistaken beliefs are shown in red. Intention plans, those plans formed by characters indicating their intended courses of action, are shown above with gray backgrounds. The story plan is shown below. Arrows from the boundaries of the intention plans into the story plan indicate the span of the story during which the Agent character holds the intention plans. Numbers in each step are for reference, and story plan steps contain both reference numbers and parenthetical numbers linking each step to its origin in an intention plan. Once Step 8 in the story plan fails, the character's beliefs about the West Door being locked are revised, the character's intentions are replanned, and Intention Plan 2 is adopted and added to the story plan.

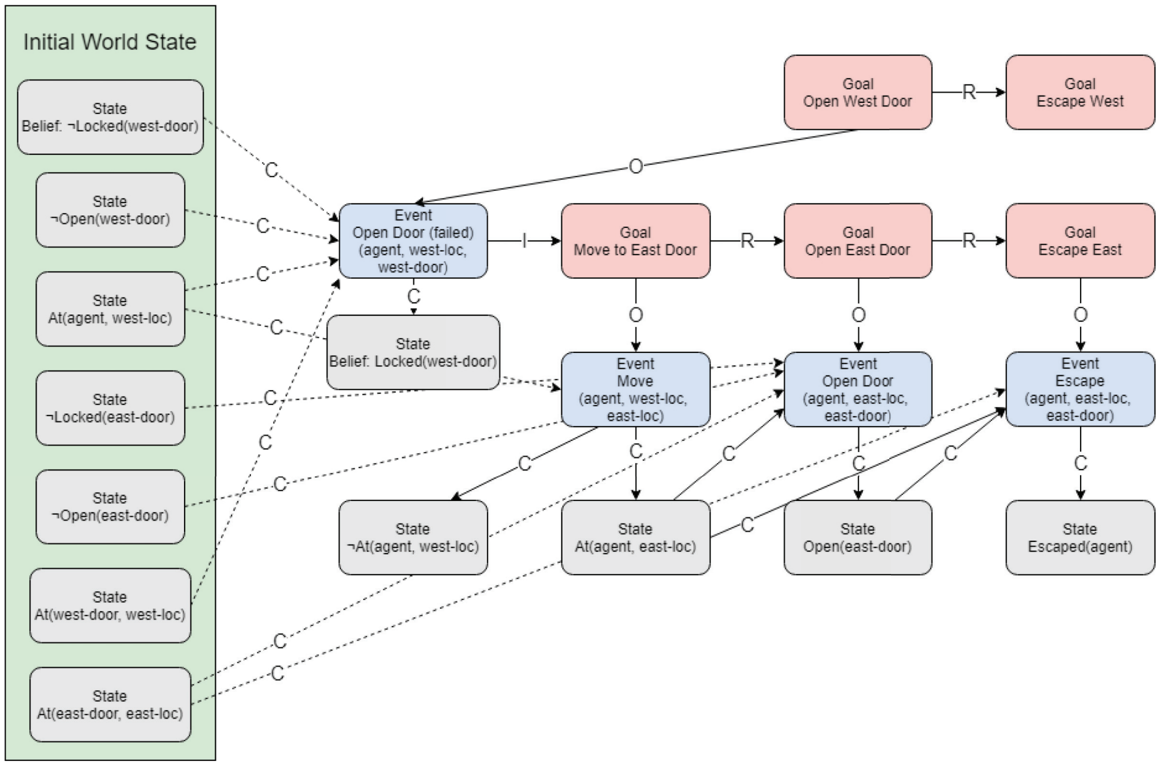


Figure 3.10. The QKS generated for the escape scenario. As in Figure 2.1, arcs labeled with ”R” are reason arcs, ”O” are outcome arcs, ”C” are consequence arcs, and ”I” is the initiate arc. The initial states are included to provide consequence arcs for causal links relying on the initial state of the domain and are shown in dotted lines solely to aid in making the diagram easier to read.

CHAPTER 4

EVALUATION

In the previous chapter, we described the HEADSPACE planner, which can produce plots where characters fail when attempting an action with an incorrect belief. The planner's representation uses the physical world state and the character's beliefs to produce stories that account for intentionality and observability. The HEADSPACE planner can also accept a planning problem with additional constraints about actions, failed actions, and intent and leverage them to further constrain the stories generated.

In this chapter, we provide an evaluation of the work proposed. We first consider an analytic evaluation in Section 4.1. In this section, we briefly provide an analytical assessment of the claim that HEADSPACE produces stories with failed actions and increases the expressive range of narrative planning algorithms. Sections 4.2 and 4.3 describe two empirical evaluations, which are more comprehensive and are intended to address our research questions more directly.

The first empirical evaluation considers the comprehension aspect of stories produced by HEADSPACE. It concerns the research question: how can a narrative planner generate stories where characters exhibit expressive behavior surrounding failed actions, and how do humans comprehend them? It focuses on the plots produced by HEADSPACE and a reader's ability to understand the intent dynamics surrounding failed actions correctly. This evaluation relies on the QUEST cognitive model to evaluate comprehension of the stories produced by HEADSPACEand is described in Section 4.2.

The second empirical evaluation considers how the enhanced authorial expressivity impacts the authoring process. It is centered around the research question: how does the added expressivity impact the ability of authors to create plots using narrative planning technologies? This evaluation approach studies the authorial effort required to create stories with failed actions using an ablative study design: comparing how participants with access to different levels of authorial expressivity perform on a task to construct plots with specific properties. This evaluation has been described in Section 4.3.

4.1 Analytical Evaluation

In this dissertation, we propose an approach for narrative planning capable of producing plot-level structures that current narrative planners do not generate. Specifically, HEADSPACE can generate plans that allow for characters' actions to fail, and those failures can prompt belief revision as characters seek new ways to achieve their goals.

While there have been narrative planning approaches that use metrics drawn from conventional planning evaluation (e.g., minimum plan length and plan cost [98]), we argue that such metrics, used alone or as the over-riding indicator of the strength of an algorithm, would be inappropriate. Plot lines and the plan-based structures that can be used to characterize them are commonly not optimal or cost efficient, partly due to the limitations of the characters performing the plot's actions. HEADSPACE features plans that incorporate failed actions, where characters have incorrect beliefs that might lead them to perform actions that are not optimal for their goals. This could lead to narratives that are longer than ones where the least amount of actions lead to the goal state being achieved. Moreover, in scenes such as the example from *Avengers: Endgame*, metrics such as plan length and minimum cost would not lead to such narratives. Hence, while plan efficiency metrics are important contributors to the design of appropriate story plan generation methods, there is value in narrative planning approaches that generate plans with properties that would not conventionally be considered ideal.

A significant contribution of HEADSPACE is its ability to produce plans that no other planning approach currently generates: plans where actions fail because of incorrect beliefs and characters subsequently correct those incorrect beliefs where possible and carry on in pursuit of their goals. Our work differs from related efforts in several ways.

Teutenberg and Porteous [98] propose an approach that focuses on characters holding distinct sets of beliefs to support deception. In their work, however, their planner cannot produce plans containing action failure. Similarly, the Glaive planner [107] supports characters holding distinct sets of beliefs about the world, and Glaive does form plans where characters pursue subplans that, if fully executed, would not succeed (due to unanticipated conflict with other characters' subplans). However, Glaive plans also contain no failed actions. Plans produced by VST [96] contain subplans built using mistaken character beliefs. Like Glaive, however, characters in VST plans always detect their mistaken beliefs prior to attempting actions that would otherwise fail, and so they update their plans to avoid action failure. Much like the plans produced by HEADSPACE, the plans produced by the method defined by Christensen, Nelson, and Cardona-Rivera [19] contain failed actions. However, their approach cannot produce plans where a character's belief about a proposition changes to ignorance (as happens in HEADSPACE when an action fails). As a result, their method cannot produce plans where information-seeking behavior arises because of action failure and subsequent actions take that plan repair into account.

While these other systems address aspects of character belief and plan/action failure, none of them provide an integrated model producing narratives that involve characters that have incorrect beliefs about the world, plan to take actions that they perceive to be executable, attempt but fail at those actions, and as a result, become ignorant about conditions in the world related to their failed action's execution. Furthermore, while our approach demonstrates this increased expressivity, it also generates narratives where characters do not have distinct sets of beliefs (e.g., where all characters have complete knowledge of the world) and, as a result, actions do not fail. This is achieved in our approach by assigning all characters full, correct belief in the initial state and restricting the syntax of our operators to have identical epistemic and material preconditions and effects. In this way, a character would never have incorrect beliefs and would never include in their plan an action whose preconditions were not materially satisfied at execution.

4.2 Empirical Evaluation 1

This section sets out an experimental methodology for evaluating plan-based story generation that includes failed actions, and uses that methodology to evaluate a specific narrative planning knowledge representation, particularly the plans produced by the HEADSPACE planning algorithm. In this evaluation, plan data structures are translated into text examples and cognitive models of stories defined by cognitive psychologists. The cognitive models support the generation of story-related question-answer pairs with predictions on how well humans will correlate the answer and question statements. Human participants are asked to rate pairs of questions and answers about stories based on their comprehension, and their ratings are compared to predictions made by the cognitive models to gauge the participants' understanding of the underlying story structure. The experiment demonstrates that the descriptions produced from HEADSPACE plans are understandable and effectively convey stories' structure to readers when actions fail due to characters' mistaken beliefs.

To gauge reader comprehension of plan-based stories containing failed actions, we designed an experiment that followed prior approaches used to evaluate reader comprehension of stories generated by planning systems [6, 75, 106]. As described above, the experimental design borrows from cognitive psychologists' approach to modeling narrative comprehension using QUEST [40]. The QUEST cognitive model uses a representation called QUEST knowledge structures. QUEST knowledge structures (QKS) are graph structures consisting of nodes and edges. Nodes can represent either a goal, an event, or a state. Generated direct edges are one of four types: outcome, initiate, reason and consequence. The QUEST cognitive model describes an evaluation approach based on the arc distance between nodes in the graph. An arc search procedure measures the arc distance between two nodes. The arc distance represents the degree to which people would connect the two nodes to be related. The evaluation approach involves forming a question-answer pair between the two nodes. Evaluation of comprehension is done by asking study participants to rate the question-answer pair presented based on how good of an answer the answer statement is to the question presented. The *Goodness of Answer*(GOA) rating for that QA pair is calculated based on these responses, and user comprehension is evaluated by correlating the GOA ratings and the arc distance between the QA pairs.

For the evaluation, the following hypothesis is being tested:

Hypothesis 1. *If readers can comprehend the role of failed actions in stories generated by a planner, they will rate correct answers higher than incorrect answers for comprehension questions about the belief and intent dynamics behind failed actions in the story.*

Specifically, we hypothesized that if readers were able to comprehend the role of failed actions in stories generated by our approach, then, for questions related to those failed actions, readers' mean GOA ratings for Question-Answer pairs identified by QUEST as "better" would be higher than their mean GOA ratings for Question-Answer pairs identified by QUEST as "worse."

4.2.1 Example Plan and Corresponding QKS and Story Text

To demonstrate the character of belief dynamics and failed actions in HEADSPACE and to show the connection between a HEADSPACE plan, a QKS data structure, and a textual representation of the story they define, we used two stories in the experiment– both stories drawn from the same Western-themed planning domain. The first problem is the Drink Refill problem introduced in Chapter 3. The second problem, which we call the Breakout problem, is an updated version of the example domain first described by Thorne and Young [100]. In this section, we describe the Breakout problem: the plan we describe is shown in Figure 4.1, the corresponding QKS is shown in Figure 4.2, and the text based on the plan is shown in Example Text 1.

The Breakout example makes use of the following operators:

1. PICKUP, where a character picks up an object from the floor,

2. SHOOT-AT-DOOR, where a character fires a gun they're holding at the lock of a door to break (and unlock) the door's lock,

3. CHECK-GUN-LOADED-F/CHECK-GUN-LOADED-T, where a character opens the cylinder of a revolver that they're holding in their hand, then learns whether the revolver is unloaded,

4. LOAD-GUN, where a character takes bullets that they're holding and loads a gun that's in their hand,

5. ESCAPE, where a character opens an unlocked jail cell door and walks through it to freedom.

For the example below, an informal sketch of the planning problem's initial state sets a character, Dolores, locked in a jail cell with only one exit: a locked door. Fortunately, a deputy has foolishly left a gun and some bullets on the floor just outside Dolores' cell, within her reach. The goal for the story is for Dolores to be in the hallway outside her cell.

Dolores desires to escape and be free, with the motivation for the desire being that she is in the jail cell.

The plan for the story is shown in Figure 4.1. In the story plan, Dolores intends to shoot the door's lock to damage it, then open the door and escape her cell. The plan in Figure 4.1 shows the actual executed story actions, the actions that are attempted but fail, and the actions that were intended by Dolores throughout the plan's execution. In the world state before the start of the plan, Dolores believes that a gun is loaded and on the floor within her reach, bullets are also next to the gun, and the door to her jail cell is locked and closed. Her beliefs at that time are correct except for the fact that the gun is actually unloaded. Dolores first picks up the gun (Action 1), then pulls the trigger, intending to shoot the door's lock, thus unlocking it (Action 2). Because the gun isn't loaded, the action fails. At this point, Dolores realizes that the action failed and becomes uncertain about the beliefs involved in the failed action's preconditions.

In the world state after Action 2, all of the epistemic preconditions for Dolores' execution of Action 2 have been asserted as unknown in her belief model. In HEADSPACE, when a character attempts an action but fails, the character automatically comes to doubt its belief in each of the failed action's preconditions. Through sensing actions, Dolores passively verifies that she is indeed in the jail cell, holding the gun, and the jail door is in front of her. However, she becomes uncertain whether the gun is loaded. Dolores detects that these new beliefs are inconsistent with her intention plan IP1, causing her to drop IP1 and form a new intention plan to first confirm that her gun is loaded and then proceed to use it to escape. Dolores then actively seeks new beliefs about the gun's ammo status by checking the gun's cylinder (Action 3). In the world state resulting from Action 3, Dolores believes that her gun is unloaded. Having expected the gun to be loaded, Dolores detects that her new belief is inconsistent with intention plan IP2, causing her to drop IP2 and form a new intention plan to load her gun and then use it to escape. In Action 4, she takes bullets from the floor; in Action 5, she uses them to load her gun. In Action 6, she fires at the lock again. Succeeding this time, the door is now unlocked. Since Dolores believes correctly at that point that the door is unlocked, she opens the door (Action 7) and escapes her cell.

The example plan was used to generate a simple text-based story using a semi-automated process. For each action of the plan, we created a simple sentence based on a pre-defined template for that action's type and filled references in the template with proper names of characters and objects used in the action. Boilerplate text describing the initial state and the goal of the character was added to the beginning of the story. The automatically generated story elements were then edited by hand to replace proper nouns with pronoun references and combine independent clauses using conjunctions to increase readability. The text in Text Example 1 shows the example story sequence generated for the Breakout story.

Example Text 1. *Dolores is imprisoned inside a jail cell. She wants to escape from the jail. Just outside of the jail cell, there lies a revolver and some bullets within her reach. Dolores walks over and picks up the gun. She walks over to the jail door. Dolores attempts to shoot at the jail door lock but fails. Dolores checks the revolver to see if it is loaded, and finds it unloaded. She walks over and picks up the bullets. Dolores loads the bullets into the gun and fires again, this time successfully breaking open the jail door. She then walks out of the jail cell and exits the jail.*

In addition to generating text from the plan, the plan data structures are also used to create a QUEST knowledge structure (QKS) for the story. The translation algorithm proposed in Section 3.4 was used to accomplish this. We extended the approach by Christian and Young [20], adapted to HEADSPACE plans by adding goal nodes characterizing the elements of unfulfilled intention plans and the addition of failed outcome arcs for failed events. The generated QKS for the plan in Figure 4.1 is depicted in Figure 4.2.

4.2.2 Study Design

The design of our experiment is similar to previous QUEST-based evaluation methods but made use of HEADSPACE plans and related stories that contained action failures. In our experiment, we first generated HEADSPACE plans that contained action failures. We then used the translation approach described in Section 3.4 to translate the plans into QKSes. We also converted the plans into text-based stories using a template-based approach. Using QUEST's arc search procedure, question-answer pairs for “Why”, “How,” and “What are the consequences of” were generated for pairs of nodes in the QKSs. Human participants read the stories and then answered questions requiring them to provide GOA ratings on a 4-point Likert Scale for the question-answer pairs. For this study, we focused on just those

Why questions that involved a failed action as either the question node or the answer node in the QKS. An example question-answer pair is provided in Example Text 2.

Example Text 2. *Why did Dolores want to check the revolver? Because she failed to shoot at the jail door lock.*

We compared participant GOA ratings for nodes that QUEST rated as better answers to participant GOA ratings for nodes that QUEST rated as worse answers. If participants understood the relationships between the events in the stories, we would expect to see higher participant GOA ratings for the better answers than for the worse answers.

We generated plans, stories, and QUEST representations in two different planning problems drawn from a common Western-themed domain: the Drink Refill and the Breakout problems. Figure 4.3 shows the ground operators used in the Breakout or Drink Refill plans. Both problems and resulting solution plans were designed to be similar in structure, with an identical number of actions, the second action failing, an intermediate set of sensing/recovery actions, and then successful plan completion. Participants were randomly assigned to one of the two generated story contexts.

4.2.3 Recruitment

Participants for this experiment were recruited using Amazon Mechanical Turk. Out of 42 participants who completed the survey, we discarded responses from 33 participants because they did not pass simple pre-defined comprehension check questions (provided in Appendix B). Data from the nine remaining participants was used for this study. Mechanical Turk is known for producing noisy results, as observed from other studies that have recruited participants using MTurk for story comprehension-related tasks (e.g., [29]). For this study, we were able to report significant results even with a relatively small number of participants with high confidence, which was verified by a power analysis done before running the study.

4.2.4 Results

A standard one-tailed t-test was used to compare the mean GOA rating of question-answer pairs with an arc distance of 1 (identified as the better answers) to the mean GOA rating of question-answer pairs with an arc distance of 3 (identified as the worse answers).

The t-test result with 11 degrees of freedom yields $t = 1.7959$ with a p-value of 0.0065 ($p < 0.01$). The responses to question-answer pairs with an arc distance of 1 ("better") were rated significantly higher than responses to question-answer pairs with an arc distance of 3 ("worse").

To provide a more fine-grained analysis, a standard one-tailed t-test was used to compare mean GOA ratings for question-answer pairs with an arc distance of 1 or 2 (identified as "better" answers) with the mean GOA rating of question-answer pairs with an arc distance of 3 (identified as "worse" answers). The t-test result with 10 degrees of freedom yields $t = 1.8124$ with a p-value of 0.0092 ($p < 0.01$). The responses to question-answer pairs with an arc distance of 1 or 2 ("better") were rated significantly higher than responses to question-answer pairs with an arc distance of 3 ("worse"). The means and variances for the conditions are reported in Table 4.1. T-tests performed to compare responses for an arc distance of 2 against an arc distance of 3 and an arc distance of 1 against an arc distance of 2 or 3 also found some evidence ($p < 0.05$) of significance.

4.2.5 Discussion

The above-mentioned experiment focused on measuring readers' comprehension of underlying HEADSPACE plan structure. Specifically, we focused on comparing readers' GOA ratings with QUEST ratings specifically for *Why* questions relating to node pairs involving failed actions.[1] The analysis clearly shows that readers rated node pairs closer together in the QKS as better answers and node pairs farther apart as worse answers – as predicted by the QUEST arc search procedure. These results provide strong support for the claim that readers could understand the role of failed actions in the unfolding events of each story, and that the corresponding HEADSPACE plan structures served as the basis for readers' comprehension. The results of this experiment support Hypothesis 1.

One limitation of the current study is that it only considers stories where actions fail and then characters modify the world state to allow the failed action to be performed successfully. In many stories with failed actions, characters drop their intentions when actions fail and adopt new courses of action rather than repairing the world to re-attempt

[1]Because there is already a long history of work leveraging QUEST to evaluate plan-based plot generation, we focused our evaluation specifically on comprehension indicators for HEADSPACE's novel contribution: failed actions.

the failed action. Work by Amos-Binks [4] has shown that readers engage more in plan-based stories where characters change their intentions. In future work, we will evaluate the connection between reader comprehension of story structure, intention revision, mistaken beliefs and failed actions.

Future work could also consider similarities and differences in user comprehension in stories with no failed actions versus stories with failed actions. This could be done by having the same planner produce two similar stories, except that one has a character achieve their goal immediately, and one has them fail at some point, correct their failure and then achieve the goal. If the mean QUEST goodness of answer ratings for both cases are similar, results of such a study would strongly support that people track intentionality in stories with failed actions in a similar fashion as they perceive stories without failed actions.

4.3 Empirical Evaluation 2

The second research question of this dissertation focuses on understanding how effective it is for authors to create stories with failed actions. Specifically, it is imperative to evaluate whether the added expressivity resulting from this new representational capacity helps authors generate stories with failed actions when using a system that leverages the HEADSPACE narrative planner.

This section describes an experiment that was conducted to understand the impact of added expressivity on the authoring of computational narratives. The following hypothesis is being tested:

Hypothesis 2. *If a narrative planner that creates stories consisting of failed actions can support authorial expressivity related to failed action constraints, then authors who use the enhanced authorial expressivity can create desired stories with failed actions with lesser authorial effort than authors who do not.*

4.3.1 Example Domain: The Kitchen Domain

For this experiment, we created a new planning problem domain that people can use for authoring multiple stories in the world. The previous domains (the Drink Refill and the Breakout problems) were limited in the narrative choices available to a character to achieve their desires. We identified the following desired characteristics for the new domain:

- presence of two characters, each with a different desire,
- the two characters' desires and respective intention plans may conflict with one another,
- it is possible for characters to miss out on observing certain effects of actions, leading to the formation of incorrect beliefs during the plot (in contrast to being specified in the planning problem directly),
- characters can achieve their goal through multiple possible paths: stick to the initial sequence, or try a different approach,
- the domain is easily understandable to participants (in contrast to the Breakout domain, which was Western-themed and could create ambiguities due to cultural unfamiliarity),
- the reason for the action failure is easily understandable to participants due to the failure being something potentially common in real-life situations.

We designed a narrative planning domain that meets the above desiderata, which we call the Kitchen domain. This domain centers around an apartment with two rooms: a kitchen and a living room. Characters may traverse freely between the two rooms. The kitchen has typical appliances such as a stove, refrigerator, toaster, and microwave. However, the toaster and microwave share a power outlet, and only one can simultaneously connect to power. Two characters, Teddy and Poppy, share this space. Teddy's desire is to heat up and eat soup, which can be heated up using the microwave or the stove. Poppy's desire is to heat up and eat bread, which can only be heated using the toaster.

The domain uses ten operators, with the definitions provided in Figure 4.4.:

- WALK, where a character walks from one room to another,
- DISCONNECT, where a character disconnects either the microwave or the toaster from the power outlet,
- CONNECT, where a character connects either the microwave or the toaster to the power outlet,

- TOAST, where a character uses the toaster to heat the bread,
- HEAT-WITH-MICROWAVE, where a character uses the microwave to heat the soup,
- HEAT-WITH-STOVE, where a character uses the stove to heat the soup,
- CHECK-OUTLET, where a character checks the status of the power outlet to see what is plugged in,
- EAT, where a character eats the heated food,
- SWITCH-CONTAINER, where a character transfers the soup between a microwave-safe bowl and a cooking pot, and
- REMOVE-FROM-FRIDGE, where a character can remove the food item from the refrigerator if it is stored in the refrigerator.

In this domain, the characters can passively sense everything about the world except for the appliance being powered by the power outlet, which needs to be actively sensed by performing the CHECK-OUTLET action. Moreover, characters can develop incorrect beliefs about the appliance connected to the power outlet if they are not in the room when a different character connects/disconnects an appliance during the story. Due to this, it is possible to craft narratives in this world, where, for example, while both Teddy and Poppy have correct beliefs initially, if one of them is not in the kitchen when the other changes the appliance plugged into the power outlet, and then one of them can fail in their attempt to heat the food through the appliance of their preference due to their incorrect beliefs. There are multiple failed actions possible in the planning problem, and in this experiment, we allow users to modify parts of the planning problem to play with the constraints and generate different stories.

4.3.2 Plot Construction Tool

The HEADSPACE planner described in this dissertation uses a PDDL-like syntax for problem specification. To allow participants to use the planner easily, there was a need for an interface that allowed them to easily express their constraints, run the planner, and view the generated plans without working with the technical PDDL-like syntax. For the purpose of the experiment described here, we built a plot construction tool.

This plot construction tool was built as a React app [28], allowing people to interact with the problem domain and the HEADSPACE planner in a web browser using a simplistic yet familiar design. This tool lets users specify beginning, ending and enhanced authorial constraints through tabs and various dropdowns. To streamline the tutorial and focus on evaluating the hypothesis, we decided to limit users' expressivity over the domain. Users of the tool cannot modify the world frame, operators, or character desires of the planning problem. Users can specify the initial state and authorial goals. They also have access to the specification of beliefs for characters. They can also specify any enhanced authorial constraints. Users can click a button to send the constraints to the planner– the planner uses this new planning problem to generate a plan, which is displayed in a list form in the tool. Users can ask the system to generate more stories for the same planning problem or change the constraints to restart the planning process. Since the experiment uses an ablative study design, users may see different levels of functionality available to them depending on the study group they are assigned to.

The plot construction tool also guides users on the tasks during the experiment. The task description can include tips on new features and several task objectives with true/-false feedback on whether they satisfied each objective of a task.

Based on an informal pilot study of 8 participants, we made many changes to the tool iteratively to tweak elements of the interface and the tasks for clarity to the users. Some of these changes are described below:

- We reduced the number of dropdowns available to users by combining logically connected literals [5]. For example, instead of having users work with two separate literals "at Teddy Kitchen" and "at Teddy LivingRoom," which could lead to inconsistencies, we condensed them into one dropdown representing Teddy's location.

- Some users unfamiliar with working with computational logic found the True/ False specification difficult. Based on feedback, we simplified the terminology to Yes/No. Similarly, we changed the vocabulary used throughout the tool for simplification (e.g., changing wording from "Generate Story" to "Create Story," and more minor changes).

- Users tended to skip or gloss over reading the task description and focus on the task objectives. Based on feedback from the pilot study, we introduced tips in the tasks when necessary, with visual cues to guide their attention to new features they may have glossed over.

- We limited the action constraints to *sometime* and *sometime after* to streamline the process of allowing users to specify constraints without overloading them with too many choices. We also renamed *sometime* to *at some point* based on feedback.

Figure 4.5 shows the final version of the user interface of the plot construction tool. Figure 4.6 demonstrates the process of specifying an action constraint in the tool.

4.3.3 Study Design

The study design uses various quantitative measures to capture aspects of "authorial effort" in a specific plot construction task. Participants used the narrative planner to create a story where characters start with correct beliefs and fail when attempting an action. In this experiment, participants were provided with a user interface tool for plan creation that had the planning algorithm in the back end. Participants were trained to work with a narrative planner to create stories. After teaching them about the domain, the user interface, and all the features available, they were asked to use the tool in specific ways to create stories with particular properties. The complete study process is described below.

At the beginning of the study, participants were assigned to one of three groups, with each group representing a level of expressivity available to them:

- **Highly limited expressivity:** Participants will only be able to specify beginning and ending constraints, and ask the planner to regenerate different plans

- **Limited expressivity:** Participants have the expressivity in the above condition, with the added ability to specify actions they would like to see in the final plan, and pairwise orderings for the actions

- **Full expressivity:** Participants have access to all features from the above conditions, but also added the ability to specify failed actions and intent dynamics surrounding failed actions

Once participants are assigned a group, they watch a 3-minute video that provides an overview of the model of narratives used in narrative planning and a description of how to use the tool. The participants then read a domain description and perform training tasks. Each task involves participants creating stories with certain beginnings, endings, or actions during the story– similar to plot construction tasks in Storyteller [8]. Participants are allotted up to five minutes per training task and can ask for assistance at any time if they are confused. Moreover, to keep them on track to progress, for any training task that took the participant more than three minutes, the PI would give them a hint or if needed, guide them with the knowledge that the particular task was trying to convey.

The learning outcomes that were the focus of the various training tasks are briefly summarized in Table 4.2. The purpose of the training tasks was to ensure that all participants' familiarity with the user interface and the planning domain were on the same baseline. The study design for the evaluation assumes that participants have some degree of familiarity with the story domain and working with a narrative planning tool, which is done using the training tasks. An example training task is provided below for reference:

Example Text 3. ***(Task 1)*** *Stories have a beginning and ending. Find a story where Teddy starts in LivingRoom, goes to the Kitchen, heats up the soup, and eats the soup.*
Task Objectives:

- *Teddy starts in the Living Room.*
- *Soup is not heated up initially.*
- *Teddy heats up the soup in the created story.*
- *Teddy eats the soup in the created story.*

After completing all the training tasks in their study group, participants were instructed to perform the final task and informed that they had up to 15 minutes to complete it. The description for the task, along with the task objectives, are provided below (the same text was used across all three groups):

Example Text 4. ***(Task 11)*** *Use all you have learnt to create a story with both Poppy and Teddy achieving their goals, and Teddy failing to use the Microwave at some point. Teddy and Poppy cannot start with incorrect beliefs. Specifically, create a story where Teddy enters the Kitchen after*

Poppy disconnects the microwave and fails to use the microwave initially because he still believed the microwave was plugged in.

- *Everyone* ***correctly*** *believes the Microwave is powered initially.*
- *Both Soup and Bread are not heated up initially.*
- *Teddy is* ***not*** *in the Kitchen in the beginning of the story.*
- *Poppy eats the Bread in the story.*
- *In the story, Teddy (initially) attempts and fails to heat the Soup using the Microwave.*
- *Teddy finally eats the soup in the story.*

For the final task, we controlled PI intervention to a fixed statement, reminding them to consider all the elements of the interface and informing them about how much time they had.

Upon finishing the final task, participants answered a set of survey questions provided in Table 4.3. The wording of the text is based on the evaluation surveys used in related work [57, 58]. The first question, Q1, is an open-ended question, allowing participants to reflect on the tool and express their thoughts qualitatively. Questions Q2 through Q5 are designed to gauge their experience with the tool. These questions are not part of the research question being studied but allow participants to specifically focus on their experience during the final task when answering questions Q6 through Q8. We consider questions Q6 through Q8 as questions aimed at understanding authorial effort: Q6 considers the perceived difficulty of the authoring task, Q7 considers perceived effort, and Q8 considers the perceived level of authorial control over the planner. Finally, questions Q9 and Q10 are demographic questions to understand further the population used for this experiment.

4.3.3.1 Recruitment

The hypothesis focuses on narrative authoring tasks. Hence, we considered novice storytellers as a criterion for inclusion. Participants were recruited from a pool of students at the University of Utah enrolled in a class titled “Storycrafting for Games,” which had a class size of roughly 60 students. All students had some exposure to working with stories as part of the course (although all participants did not have experience with creating stories specifically). Students participated in the study for credit in the course. However, students had a choice between participating in the research study or performing another task equivalent in terms of time spent to receive credit. The study was conducted in-person: students could sign up to participate over three weeks. A preliminary power analysis led us to aim to recruit at least thirty subjects for the study: we received thirty-six participants. However, we discarded data from four participants due to language barriers and learning impairments we could not support during the experiment.

4.3.4 Results

For this study, the independent variable was the condition that users were assigned to, i.e., the degree of expressivity authors had when using a computational narrative planning approach. The dependent variables were metrics that measure authorial effort in performing the task. To capture "authorial effort," we decided to use multiple measures that can capture aspects of what can be considered as authorial effort in a narrative planning task:

1. We used task completion time as one possible metric of gauging efficiency in an authoring task [2].

2. We also tracked participant actions in the tool during the task, and considered the number of story generation requests made as another measure of effort: if an author needs to make more edits or requests to get to a story with specific properties, there is more authorial effort.

3. Finally, we also considered self-reporting measures: participants' perception of how much effort it took to produce the story in the task.

4.3.4.1 Task Completion Data

Table 4.4 provides an overview of the participant numbers. In the Highly Limited Expressivity group, 4 out of 12 participants could not complete the task within the time limit, and their data was not used in the analyses. We believe it is challenging to produce stories that meet the objectives when it is not possible to specify constraints about the actions desired in the stories. Moreover, we designed this task specifically so that the story was not the first-generated plan produced by the planner unless the action constraints guided the planner to do so. In other words, participants in this group could only generate the desired story by asking the planner for more stories with the same specifications using the “Create Another Story” button. The qualitative data suggests that participants tended to think they had incorrect specifications rather than consider that the system failed to produce a story of their choice on its first attempt.

In the Limited Expressivity group, 2 out of 10 participants could not complete the task. Upon looking at the data, there are multiple possible factors in play. First, without the ability to specify failed actions directly, participants in the Limited Expressivity group still need to explicitly specify actions in an order that leads to inconsistent beliefs and the resulting failed action. Participants need to mentally track how they will produce the inconsistent belief, which could be challenging depending on the complexity of the domain. Second, the most direct way to produce the story involves using the *sometime after* action constraint. Participants in this group only had one training task that introduced action constraints to them, and they were not required to use the *sometime after* constraint specifically in that task. The task design could have contributed to them being less comfortable with the action constraints than the participants in the Full Expressivity Group. Finally, when enhanced authorial constraints are provided, the planner takes significant time to compute a plan. Looking at the computation time shows multiple instances where, cumulatively, the planner spent 6 minutes computing plans. With the inability to directly specify the failed action constraint, the participants in this constraint had to make more requests to the planner for a plan, and had less time to work on the task given the significant time spent waiting for a story to generate.

For the participants that completed the final task, we now perform analyses on various measures to find if there was a statistically significant correlation between the level of authorial expressivity and the authorial effort to produce plots with failed actions. Table 4.5 provides an overview of the measures considered, the mean values for each measure by group, and the statistical tests used. We define each measure and describe the results below.

4.3.4.2 Task Completion Time

The task completion time was computed using telemetry data. This is the time the participant took to perform the final task, from the point they clicked on the task button to the point that they clicked on the "Create Story" button that resulted in a story that met the task objectives.

A one-way ANOVA was performed to compare the effect of authorial expressivity on the task completion time. It revealed a statistically significant difference in the task completion time between at least two groups (F(2, 23) = 6.599, p = 0.005).

Tukey-Kramer's Test for multiple comparisons found that the mean values of task completion time were significantly different between Highly Limited Expressivity and Full Expressivity (p = 0.008, 95% C.I. = [8.339E-4,5.887E-3]), and Limited Expressivity and Full Expressivity (p=0.0277, 95% C.I. = [2.769E-3,5.33E-3]).

There was no statistically significant difference between Highly Limited Expressivity and Limited Expressivity (p=0.8605).

4.3.4.3 Task Completion Time Without Computation

The above metric for task completion time includes computation time for stories that did not meet task objectives. During these time segments, participants can still change the constraints or think about the task, but cannot request a story with the updated constraints until the planner completes computing a story. Hence, while the above metric accounts for time that participants may have spent thinking about the task, it overlooks that the computation time was not something the participants had control over.

We consider an additional metric, which we call Task Completion Time Without Computation. This metric deducts all computation time from the task completion time using the telemetry data.

A one-way ANOVA was performed to compare the effect of authorial expressivity on the task completion time without computation. It revealed a statistically significant difference in the task completion time between at least two groups (F(2, 23) = 8.378, p = 0.00184).

Tukey-Kramer's Test for multiple comparisons found that the mean value of task completion time was significantly different between Highly Limited Expressivity and Full

Expressivity (p = 0.00126, 95% C.I. = [9.911E-4, 4.122E-3]).

There were no statistically significant differences between Highly Limited Expressivity and Limited Expressivity (p=0.078), and Limited and Full Expressivity (p=0.236).

4.3.4.4 Number of Story Generation Attempts

In addition to capturing time for the task, a measure to understand authorial effort can also be gauged from the number of interactions they make with the planner. We consider the number of times participants clicked on the "Create Story" button as another measure of gauging effort to produce a desired plot.

As alluded to earlier, participants in the Highly Limited Expressivity group cannot generate the desired story in one attempt due to the design of the task. Participants in the other groups are capable of generating the desired story in one attempt, but may not necessarily be successful if they are unable to use the enhanced expressivity effectively. While one may argue that measuring the difference in the number of story generation attempts is tautological, we attempt to experimentally verify this by performing a test for statistical significance.

The variances for the three groups for this measure were not equal, violating the assumption of equal variance made by the ANOVA test. Due to this, we performed a Kruskal-Wallis test, a non-parametric version of the one-way ANOVA test. The Kruskal-Wallis test is robust to different sample sizes and unequal variances.

The Kruskal-Wallis Test revealed that the number of story generation attempts was statistically different among the three groups(H=14.632, p = 5.062E-4, df = 2).

Nemenyi's Test for multiple comparisons found that the mean value of the number of story generation attempts significantly differed between Highly Limited Expressivity and Full Expressivity (p = 4.786E-4).

4.3.4.5 Self-Reported Measures

Participants self-reported their perceived difficulty, effort, and level of control for the final task in a survey after completing it. We report results for the three survey questions in Table 4.5. For all these measures, since the survey responses were Likert-scale responses, we use a ranked ANOVA test to check for statistical significance, which is recommended when working with ordinal data.

- **Perceived difficulty.** This measure relates to how strongly participants agree with the statement "I found it easy to create the story for the final task". We used the ranked ANOVA test with this measure because the data is ordinal.

  A one-way ranked ANOVA test revealed no statistically significant differences among the three groups on how they rated the difficulty of the final task (p = 0.189).

- **Perceived authorial effort.** This measure relates to how strongly participants agree with the statement "I was able to create stories for the final task without unnecessary effort."

  A one-way ranked ANOVA test revealed statistically significant differences among the perceived authorial effort for participants between at least two groups (p = 0.036).

  A ranked pairwise test found that the mean value of perceived authorial effort significantly differed between Highly Limited Expressivity and Full Expressivity (p = 0.03).

  There were no statistically significant differences between Highly Limited Expressivity and Limited Expressivity (p=0.87), and Limited and Full Expressivity (p=0.22).

- **Perceived authorial control.** This measure relates to how strongly participants agree with the statement "I felt a sense of control over the stories created for the final task."

  A one-way ranked ANOVA test revealed no statistically significant differences among the three groups on how they rated the difficulty of the final task (p = 0.464).

4.3.4.6 Additional Data Collected

We asked the participants questions about their experience using the tool for two primary reasons: (1) to allow them to distinguish between their impressions of the tool and their experience with plot construction in the final task, and (2) to ensure that the general experience with the tool was positive enough for the tool to not cause any bias in the results. We present the means for their responses on Q2 through Q5 in Table 4.6. It is important to note that the means for all the questions across all groups are more than 3, which corresponds to the Neutral option in the Likert scale. This data shows that the plot construction tool did not hamper the participants' experience in using the narrative planner.

As part of the survey, we also collected demographic information from the participants to gain a deeper level of understanding about their experience with storywriting and their interests. We present a report of their experience with creating stories in Figure 4.7. The participants were all registered for the *Storycrafting for Games* course, a class taught in the Games department of the university. We also asked them about their interests in game development roles, which has been reported in Table 4.7.

4.3.5 Discussion

Table 4.8 summarizes the statistically significant results for each pairwise comparison. From the data collected, evidence strongly suggests that participants with access to the complete set of expressivity features could complete the final plot construction task with less authorial effort than participants with limited access to the expressivity features. This evidence suggests that, if we provide the users of narrative planning algorithms with enhanced authorial expressivity abilities, they are much more effective at using the HEADSPACE planning system to create stories with failed actions. This supports Hypothesis 2.

We believe that the effect size for the Limited Expressivity could have been significant with more training for participants. The current study design yielded statistically significant differences surrounding the Limited Expressivity group in only one measure: task completion time. Upon looking at the telemetry data and the qualitative responses to Q1, we observe that the Limited Expressivity group participants did not have enough time to acquaint themselves well enough with the action constraint abilities available to them. They only had one tutorial task that allowed them to access the enhanced authorial constraints, compared to three tutorial tasks for the Full Expressivity group. Moreover, the tutorial tasks did not explicitly force them to learn all the action constraints ("At some point" and "Sometime after"). However, one of the logical approaches to the final task for this group involved using the "Sometime after" constraint to force the planner to find stories where Teddy enters the kitchen after Poppy disconnects the microwave from the power outlet. The participants in this group did not have the ability to directly specify the planner to find a story with the failed action using the "At some point" constraint. We believe it is possible to have seen a more significant effect size if we had designed more tutorial tasks to train participants to use the "Sometime after" action constraint separately.

One limitation of this experiment is that, while the Full Expressivity group had access to the expression of intent-related constraints on action failure, the final task did not explicitly require them to use them. Upon looking at the telemetry data for the final task, we observed that 2 participants used intent constraints in the final task. We believe that a comprehensive evaluation of intent-related constraints might be needed to understand how they can be used effectively to guide automated planning techniques to produce stories where characters exhibit specific behaviors upon failing an action. Moreover, we believe that the tutorial tasks were unable to communicate the mechanics of intent constraints to participants elaborately. A significant proportion of participants in the Full Expressivity condition had questions about the intent constraints, especially their presence only concerning failed actions. We believe that intent constraints might require a deeper understanding of the narrative model used by computational models of narrative.

4.4 Summary

In this chapter, we evaluated the dissertation's contributions in three ways. First, we provided a brief analytic evaluation showcasing how HEADSPACE expands upon the expressive range of stories that current narrative planning technologies can generate. Second, we studied user comprehension of the produced stories with failed actions to evaluate whether readers can comprehend the intentionality behind the failed actions accurately. The study found that people accurately track the beliefs and intentions of characters in computationally produced stories and can effectively comprehend the belief and intent dynamics surrounding failed actions. Finally, we evaluated whether the enhanced authorial expressivity contributions of the HEADSPACE planner influence the authoring process for creating plots with failed actions. The results of this experiment validated that novice storytellers are able to use action constraints effectively and failed action constraints to guide the HEADSPACE planner to create stories with failed actions.

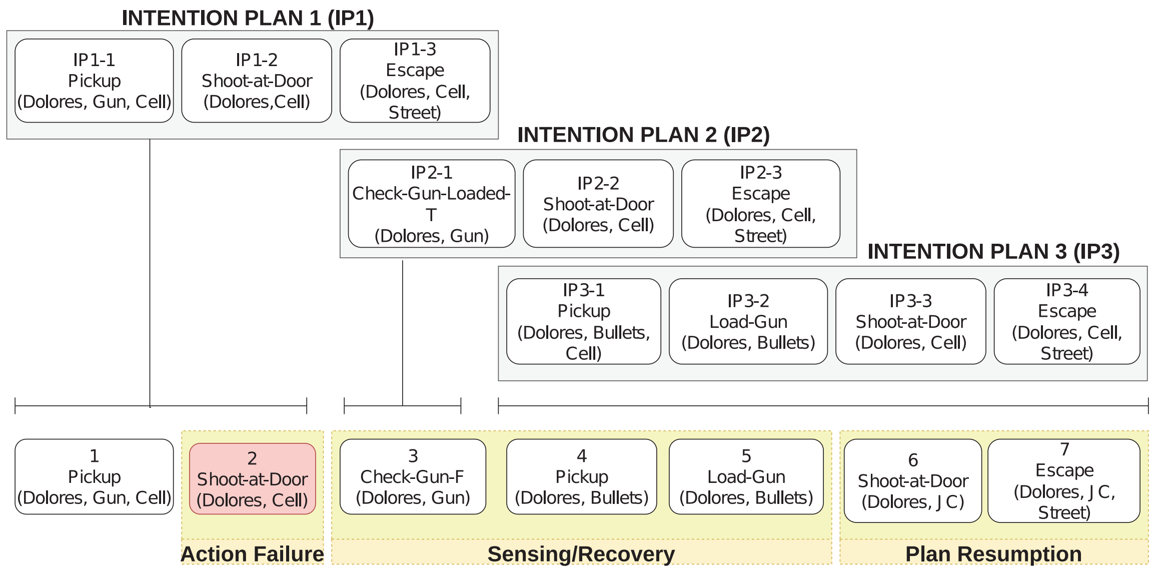

INTENTION PLAN 1 (IP1)
IP1-1 Pickup (Dolores, Gun, Cell)
IP1-2 Shoot-at-Door (Dolores,Cell)
IP1-3 Escape (Dolores, Cell, Street)
INTENTION PLAN 2 (IP2)
IP2-1 Check-Gun-Loaded-T (Dolores, Gun)
IP2-2 Shoot-at-Door (Dolores, Cell)
IP2-3 Escape (Dolores, Cell, Street)
INTENTION PLAN 3 (IP3)
IP3-1 Pickup (Dolores, Bullets, Cell)
IP3-2 Load-Gun (Dolores, Bullets)
IP3-3 Shoot-at-Door (Dolores, Cell)
IP3-4 Escape (Dolores, Cell, Street)
1 Pickup (Dolores, Gun, Cell)
2 Shoot-at-Door (Dolores, Cell)
3 Check-Gun-F (Dolores, Gun)
4 Pickup (Dolores, Bullets)
5 Load-Gun (Dolores, Bullets)
6 Shoot-at-Door (Dolores, J C)
7 Escape (Dolores, J C, Street)
Action Failure
Sensing/Recovery
Plan Resumption

Figure 4.1. The HEADSPACE plan from the Breakout domain. In this figure, time flows from left to right. Actions attempted by the character in the story are shown across the bottom of the figure using rounded rectangles and numbered 1 to 7. For preconditions and effects of each action, refer to Figure 4.3. Action 2, shown in red, is attempted but fails because its non-belief preconditions are not all met in the world state where it is attempted. The three intention plans held by the character during the course of the story are shown in gray rectangles above the plan's actions, along with an indicator showing the interval of story actions during which they are held.

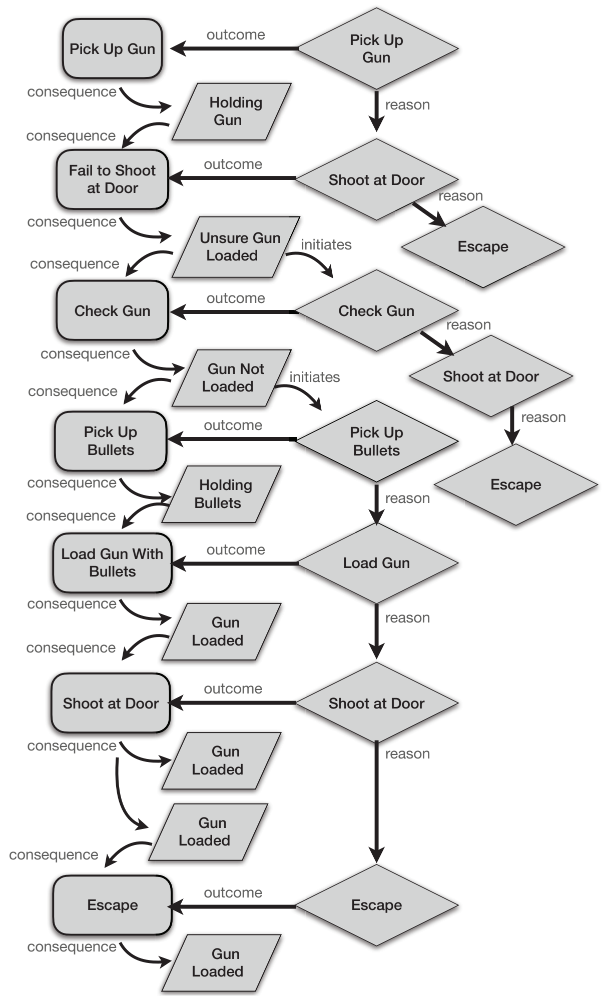


Figure 4.2. The QKS corresponding to the plan in Figure 4.1. Rounded rectangle indicate event nodes. Skewed rectangles indicate state nodes. Diamonds indicate goal nodes. Arcs from one node to another are labeled with one of the QUEST relationship types showing in Table 2.1.

Pickup(?character, ?item, ?loc)	
PRE-T	at(?character, ?loc)
	in(?item, ?loc)
PRE-F	has(?character, ?item)
PRE-B+	at(?character, ?loc)
	in(?item, ?loc)
PRE-B-	has(?character, ?item)
EFF-T	has(?character, ?item)
EFF-F	in(?item, ?loc)
EFF-B+	has(?character, ?item)
EFF-B-	in(?item, ?loc)

Load-Gun(?character, bullets)	
PRE-T	has(?character, gun)
	has(?character, bullets)
PRE-F	loaded(gun)
PRE-B+	has(?character, gun)
	has(?character, bullets)
PRE-B-	loaded(gun)
EFF-T	loaded(gun)
EFF-B+	loaded(gun)

Check-Gun-Loaded-T(?character)	
PRE-T	has(?character, gun)
	loaded(gun)
PRE-B+	has(?character, gun)
PRE-U	loaded(gun)
EFF-B+	loaded(gun)

Place-Down(?character, ?item, ?loc)	
PRE-T	holding(?character, ?item)
	at(?character, ?loc)
PRE-F	in(?item, ?loc)
PRE-B+	holding(?character, ?item)
	at(?character, ?loc)
PRE-B-	in(?item, ?loc)
EFF-T	in(?item, ?loc)
EFF-F	holding(?character, *)
EFF-B+	in(?item, ?loc)
EFF-B-	holding(?character, *)

Hold(?character, ?item, ?loc)	
PRE-T	at(?character, ?loc)
	in(?item, ?loc)
PRE-F	holding(?character, *)
PRE-B+	at(?character, ?loc)
	in(?item, ?loc)
PRE-B-	holding(?character, *)
EFF-T	holding(?character, ?item)
EFF-B+	holding(?character, ?item)

Check-Gun-Loaded-F(?character)	
PRE-T	has(?character, gun)
PRE-F	loaded(gun)
PRE-B+	has(?character, gun)
PRE-U	loaded(gun)
EFF-B-	loaded(gun)

Check-Bottle-Empty-T(?character, ?bottle)	
PRE-T	holding(?character, ?bottle)
	empty(?bottle)
PRE-B+	holding(?character, ?bottle)
PRE-U	empty(?bottle)
EFF-B+	empty(?bottle)

Check-Bottle-Empty-F(?character, ?bottle)	
PRE-T	holding(?character, ?bottle)
PRE-F	empty(?bottle)
PRE-B+	holding(?character, ?bottle)
PRE-U	empty(?bottle)
EFF-B-	empty(?bottle)

Pour-Drink(?character, ?bottle, ?glass)	
PRE-T	holding(?character, ?bottle)
	empty(?glass)
PRE-F	empty(?bottle)
PRE-B+	holding(?character, ?bottle)
	empty(?glass)
PRE-B-	empty(?bottle)
EFF-T	empty(?bottle)
EFF-F	empty(?glass)
EFF-B+	empty(?bottle)
EFF-B-	empty(?glass)

Escape(?character, cell, street)	
PRE-T	at(?char, cell)
PRE-F	locked(door)
PRE-B+	at(?char, cell)
PRE-B-	locked(door)
EFF-T	at(?char, street)
EFF-F	at(?char, cell)
EFF-B+	at(?char, street)
EFF-B-	at(?char, cell)

Shoot-at-Door(?character, ?loc)	
PRE-T	at(?character, ?loc)
	has(?character, gun)
	loaded(gun)
	in(door, ?loc)
PRE-B+	at(?character, ?loc)
	has(?character, gun)
	loaded(gun)
	in(door, ?loc)
EFF-F	locked(door)
EFF-B-	locked(door)

Figure 4.3. Operators that were part of the Western Domain. These operators form the plan for the Breakout and Drink Refill problems within the domain.

Walk(?character, ?from, ?to)	
PRE-T	at(?character, ?from)
PRE-F	at(?character, ?to)
PRE-B+	at(?character, ?from)
PRE-B-	at(?character, ?to)
EFF-T	at(?character, ?to)
EFF-F	at(?character, ?from)
EFF-B+	at(?character, ?to)
EFF-B-	at(?character, ?from)

HeatWithMicrowave(?character, ?bowl)	
PRE-T	at(?character, Kitchen)
	plugged-in(Microwave)
	contained(Soup, ?bowl)
PRE-F	heated(Soup)
	eaten(Soup, *)
	in-fridge(Soup)
PRE-B+	at(?character, Kitchen)
	plugged-in(Microwave)
	contained(Soup, ?bowl)
PRE-B-	heated(Soup)
	eaten(Soup, *)
	in-fridge(Soup)
EFF-T	heated(Soup)
EFF-B+	heated(Soup)

Eat(?character, ?food)	
PRE-T	heated(?food)
PRE-F	eaten(?food, *)
	in-fridge(?food)
PRE-B+	heated(?food)
PRE-B-	eaten(?food, *)
	in-fridge(?food)
EFF-T	eaten(?food, ?character)
EFF-B+	eaten(?food, ?character)

Toast(?character)	
PRE-T	at(?character, Kitchen)
	plugged-in(Toaster)
PRE-F	heated(Bread)
	eaten(Bread,*)
	in-fridge(Bread)
PRE-B+	at(?character, Kitchen)
	plugged-in(Toaster)
PRE-B-	heated(Bread)
	eaten(Bread,*)
	in-fridge(Bread)
EFF-T	heated(Bread)
EFF-B+	heated(Bread)

CheckOutletMicrowave-T(?character)	
PRE-T	plugged-in(Microwave)
PRE-U	plugged-in(Microwave)
EFF-B+	plugged-in(Microwave)
EFF-B-	plugged-in(Toaster)
	outlet-empty

CheckOutletMicrowave-F1(?character)	
PRE-T	plugged-in(Toaster)
PRE-U	plugged-in(Microwave)
EFF-B+	plugged-in(Toaster)
EFF-B-	plugged-in(Microwave)
	outlet-empty

CheckOutletMicrowave-F2(?character)	
PRE-T	outlet-empty
PRE-U	plugged-in(Microwave)
EFF-B+	outlet-empty
EFF-B-	plugged-in(Microwave)
	plugged-in(Toaster)

Disconnect(?character, ?appliance)	
PRE-T	at(?character, Kitchen)
	plugged-in(?appliance)
PRE-F	outlet-empty
PRE-B+	at(?character, Kitchen)
	plugged-in(?appliance)
PRE-B-	outlet-empty
EFF-T	outlet-empty
EFF-F	plugged-in(?appliance)
EFF-B+	outlet-empty
EFF-B-	plugged-in(?appliance)

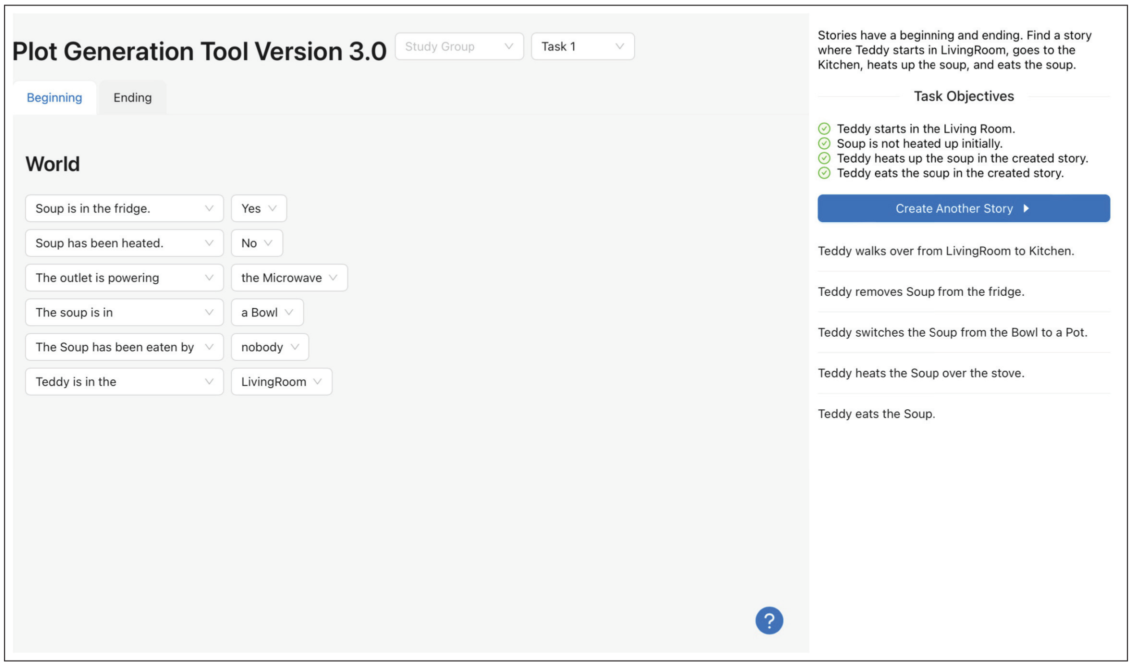
SwitchContainer(?character, ?container1, ?container2)	
PRE-T	contained(Soup, ?container1)
PRE-F	contained(Soup, ?container2)
	eaten(Soup,*)
PRE-B+	contained(Soup, ?container1)
PRE-B-	contained(Soup, ?container2)
	eaten(Soup,*)
EFF-T	contained(Soup, ?container2)
EFF-F	contained(Soup, ?container1)
EFF-B+	contained(Soup, ?container2)
EFF-B-	contained(Soup, ?container1)

RemoveFromFridge(?character, ?food)	
PRE-T	at(?character, Kitchen)
	in-fridge(?food)
PRE-B+	at(?character, Kitchen)
	in-fridge(?food)
EFF-F	in-fridge(?food)
EFF-B+	in-fridge(?food)

Connect(?character, ?appliance)	
PRE-T	at(?character, Kitchen)
	outlet-empty
PRE-F	plugged-in(Microwave)
	plugged-in(Toaster)
PRE-B+	at(?character, Kitchen)
	outlet-empty
PRE-B-	plugged-in(Microwave)
	plugged-in(Toaster)
EFF-T	plugged-in(?appliance)
EFF-F	outlet-empty
EFF-B+	plugged-in(?appliance)
EFF-B-	outlet-empty

HeatWithStove(?character, ?pot)	
PRE-T	at(?character, Kitchen)
	contained(Soup, ?pot)
PRE-F	heated(Soup)
	eaten(Soup, *)
	in-fridge(Soup)
PRE-B+	at(?character, Kitchen)
	contained(Soup, ?pot)
PRE-B-	heated(Soup)
	eaten(Soup, *)
	in-fridge(Soup)
EFF-T	heated(Soup)
EFF-B+	heated(Soup)

Figure 4.4. The operators that are defined as part of the Kitchen domain. Note that some literals have been condensed: for example, the literal *eaten(Soup, *)* uses the * as a wildcard to signify all possible values in its place. Also note that observability and failed actions are not displayed in the operator definitions presented above.



Soup is in the fridge. — Yes
Soup has been heated. — No
The outlet is powering — the Microwave
The soup is in — a Bowl
The Soup has been eaten by — nobody
Teddy is in the — LivingRoom

Stories have a beginning and ending. Find a story where Teddy starts in LivingRoom, goes to the Kitchen, heats up the soup, and eats the soup.

Task Objectives

- ✓ Teddy starts in the Living Room.
- ✓ Soup is not heated up initially.
- ✓ Teddy heats up the soup in the created story.
- ✓ Teddy eats the scup in the created story.

Teddy walks over from LivingRoom to Kitchen.

Teddy removes Soup from the fridge.

Teddy switches the Soup from the Bowl to a Pot.

Teddy heats the Soup over the stove.

Teddy eats the Soup.

Figure 4.5. A screenshot of the plot construction tool. Task objectives are visible in the top right. Users can specify constraints in the left pane with Beginning/Actions/Ending tabs. They can click on the blue Create Story button and the generated plot is visible as a list in the right pane.

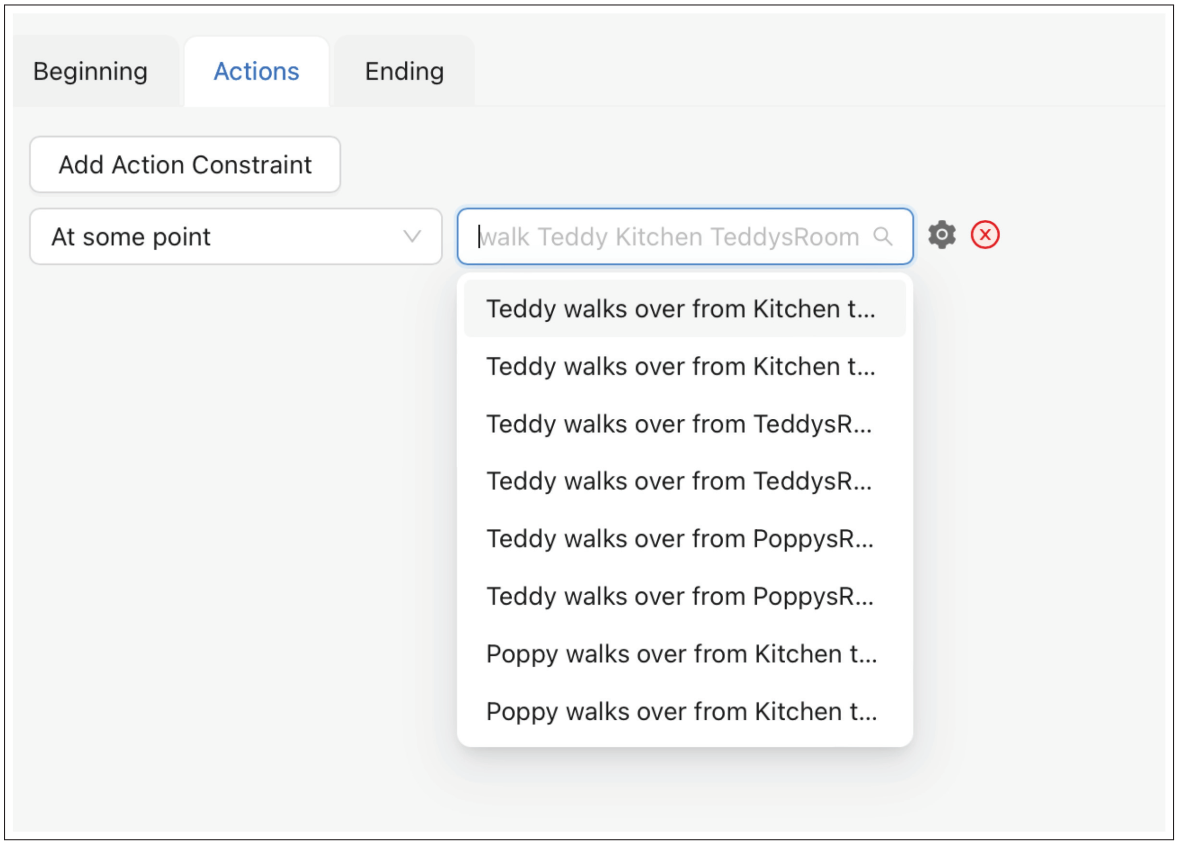

Beginning
Actions
Ending
Add Action Constraint
At some point
walk Teddy Kitchen TeddysRoom
Teddy walks over from Kitchen t...
Teddy walks over from Kitchen t...
Teddy walks over from TeddysR...
Teddy walks over from TeddysR...
Teddy walks over from PoppysR...
Teddy walks over from PoppysR...
Poppy walks over from Kitchen t...
Poppy walks over from Kitchen t...

Figure 4.6. A screenshot demonstrating the interface for action constraints. Users can specify the kind of constraint (At some point or Sometime After), and then can select an action from the dropdown menu. Users can also type to search for a specific action. Users can click on the settings icon to add an intent constraint.

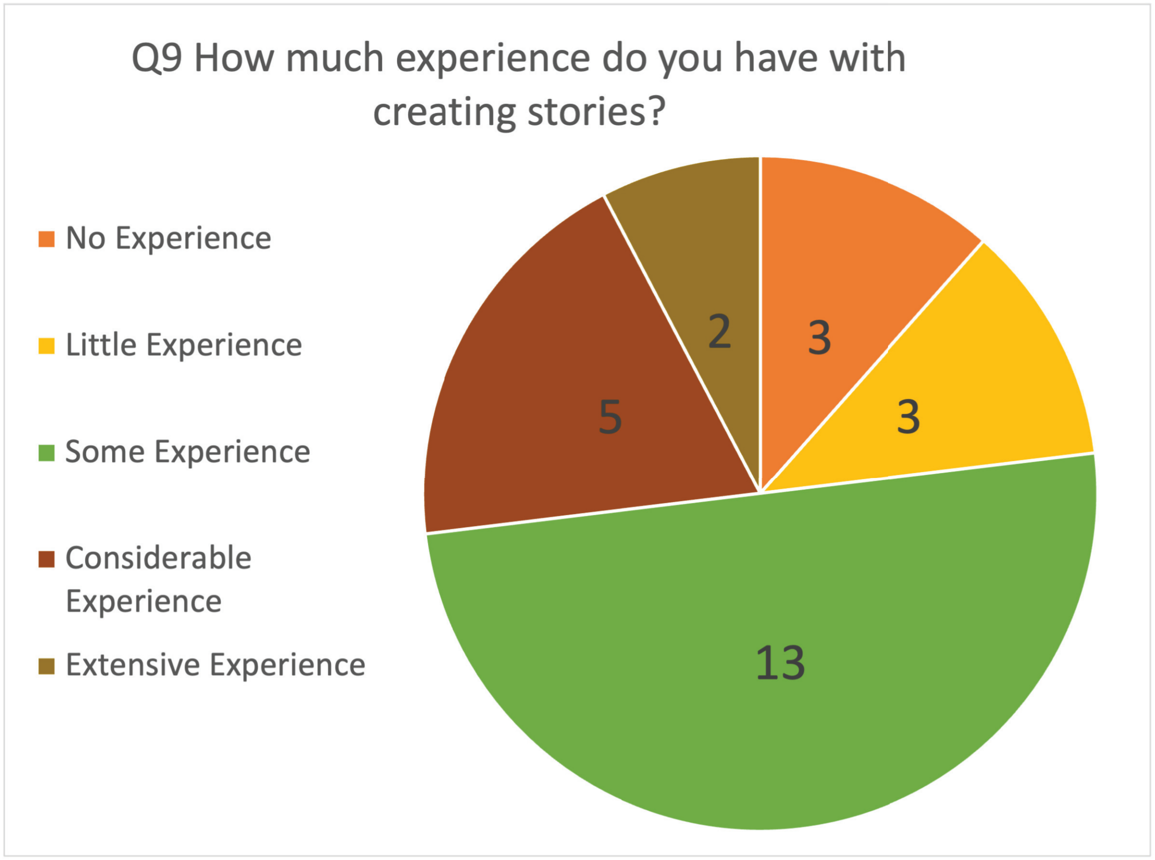


Figure 4.7. Distribution of self-reported demographic information about participants regarding their experience with creating stories.

Table 4.1: Mean Goodness of Answer (GOA) ratings (and standard deviations) for Question/Answer pairs containing action failures.

Responses with arc distance of 1	**Responses with arc distance of 1 or 2**	**Responses with arc distance of 3**
3.1111 (0.673)	3.0377 (0.6908)	2.1111 (0.861)

Table 4.2: Learning outcomes for the tutorial tasks that participants performed as part of training. Tasks 1-7 were completed by all groups. Task 8 was completed by the limited and full expressivity groups, since only these groups can specify action constraints. Tasks 9 and 10 were only completed by participants in the Full Expressivity group.

Task Number	**Learning Outcomes for Tutorial Task**	**Groups**
1	Beginning and Ending Constraints, Teddy's desires	All groups
2	Poppy's desires distinct from Teddy's desires	All groups
3	Alternate plans to achieve same goals, Generating multiple stories for same problem	All groups
4	Generating multiple stories for same problem	All groups
5	Specifying character beliefs, Generating stories with failed actions	All groups
6	Generating stories with different behaviors upon failed action (persist/substitute)	All groups
7	Changing ending constraints to create story where character does not pursue goal after failing an action (drop)	All groups
8	Specifying action constraints	Limited + Full Expressivity
9	Specifying failed action constraints	Full Expressivity
10	Specifying intent constraints	Full Expressivity

Table 4.3: Survey questions used for the experiment described in Section 4.3. Question Q1 is open-ended and questions Q2 through Q8 are 5-point Likert-Scale questions.

Code	Question Text
Q1	What was your overall impression of this tool?
Q2	I found this tool easy to use to create stories.
Q3	I was able to use this tool to create stories without unnecessary effort.
Q4	I felt a sense of control over the stories created using the tool.
Q5	I found the tool fun to use.
Q6	I found it easy to create the story for the final task.
Q7	I was able to create stories for the final task without unnecessary effort.
Q8	I felt a sense of control over the stories created for the final task.
Q9	How much experience do you have with creating stories?
Q10	What game development role(s) do you identify with?

Table 4.4: Participant task completion data by group.

Group	Total number of participants	Participants that completed the final task
Highly Limited Expressivity	12	8
Limited Expressivity	10	8
Full Expressivity	10	10

Table 4.5: Measures and tests used for gauging if authorial expressivity was correlated with authorial effort. The bolded measures represent that they produced statistically significant results.

Measure	**Means**			**Statistical Test Used**
	Highly limited expressivity (n=8)	**Limited expressivity (n=8)**	**Full Expressivity (n=10)**	
Task Completion Time (sec)	472	424	181	One-way ANOVA
Task Completion Time without all computation time (sec)	395	265	175	One-way ANOVA
Number of story generation attempts	10	5.124	1.4	Kruskal-Wallis Test
Perceived Difficulty (Q6)	3.43	3.78	4.3	Ranked ANOVA
Perceived Authorial Effort (Q7)	3.71	3.78	4.44	Ranked ANOVA
Perceived Authorial Control (Q8)	3.57	4	4.2	Ranked ANOVA

Table 4.6: Mean values by study group for the survey questions that were related to participants' experience with the plot construction tool.

	Highly Limited	Limited	Full Expressivity
Q2 I found this tool easy to use to create stories.	4	4.22	3.8
Q3 I was able to use this tool to create stories without unnecessary effort.	3.86	3.77	3.8
Q4 I felt a sense of control over the stories created using the tool.	4	4.22	3.6
Q5 I found the tool fun to use.	4.43	4	4

Table 4.7: Participant responses to Q10, a demographic question to capture participant's interests.

Q10 What game development role(s) do you identify with?	Number of participants (N=26)
Game Designer	15
Narrative Designer	11
Producer	11
Artist	8
Level Designer	7
Tech Artist	3
Animator	2
Engineer	2
Games User Researcher	2
Other	2

Table 4.8: Table summarizing the results of the statistical analyses for each pairwise combination of the three study groups.

	Highly Limited vs Limited	**Limited vs Full**	**Highly Limited vs Full**
Task Completion Time	No	Yes ($p < 0.05$)	Yes ($p < 0.01$)
Task Completion Time without Computation	No	No	Yes ($p < 0.01$)
Number of story generation attempts	No	No	Yes ($p < 0.01$)
Q6 Perceived Difficulty	No	No	No
Q7 Perceived Authorial Effort	No	No	Yes ($p < 0.05$)
Q8 Perceived Control	No	No	No

CHAPTER 5

CONCLUSION

This dissertation described work in computational narrative generation that focused on how narrative planning approaches can be used to produce stories consisting of belief-based failed actions. In this chapter, we summarize our contributions in Section 5.1, and then discuss limitations and possible directions for future work in Section 5.2

5.1 Contributions

In this dissertation, we contributed to narrative planning related to the construction of stories where characters fail at an action when they have incorrect beliefs about the world. First, we proposed a novel algorithm that operates on a knowledge representation consisting of a world model alongside character beliefs about that world. The HEADSPACE planning algorithm generates plots where a character-level planner ensures that characters act with intent on possibly incorrect beliefs, and a global-level planner is guided by a search process to meet authorial goals. We then contributed to using the QUEST cognitive model to create representations for narratives to also consider plans consisting of failed actions. We evaluated these contributions by performing a user study that supports the claim that readers can accurately track and comprehend the computationally generated plans consisting of failed actions. While the planning algorithm expands the expressive range of computational narrative generation approaches, we also considered the current limitations of authorial expressivity when working with narrative planners. We developed support for authors to express additional constraints about desired properties in stories: constraints about successful or failed attempts at actions and character intent behavior surrounding failed actions. This was evaluated by measuring the ability of novice storywriters to work with the enhanced authorial expressivity proposed by this work to guide the narrative planner to produce stories consisting of failed actions.

To contextualize our contributions with the research questions, methods, and hypotheses, Table 5.1 provides a Blumenfeld chart that summarizes the research questions and hypotheses this dissertation covers.

5.2 Limitations and Areas for Future Work

5.2.1 Narrative Planning with Machine Learning Approaches

Machine learning techniques have found their use in creative systems to produce stories and assist in authoring [21, 79]. Moreover, recent advances and the success of large language models like ChatGPT have shown to perform competitively against human-authored stories [114]. While ML approaches are not in scope of the work in this dissertation, future work can be directed to consider how ML approaches can benefit the work described in this dissertation.

There has been a body of work that focuses on using ML techniques with PDDL planning systems in many ways: building domains and creating action models, assisting with search control by creating domain-specific strategies and heuristics, all of which are complex tasks for humans [52]. Future work can consider how ML techniques can reason about stories with complex belief and intent dynamics for failed actions as well.

Large language models (LLMs) have also been used alongside planning approaches to construct stories. The Tattletale system [91], for example, uses a hybrid approach: it uses planning to create narrative plans, which are then used as input to an LLM to produce natural language stories. However, recent approaches have also warned that LLMs are not currently capable of solving planning problems independently but can solve some nontrivial problems [90]. However, the future of machine learning holds promise at tackling many of the bottlenecks that are part of approaches that only use planning.

5.2.2 Authorial Heuristics

The current implementation for the action selection mechanism when selecting the next action from multiple intention plans (line 7 in Algorithm 6) uses the plan's current cost as a heuristic, i.e., is effectively a breadth-first-search. While the algorithm uses a fringe to recover from undesired paths, it is an efficient approach. With the addition of action constraints, existing heuristics are unable to effectively determine the most "natural" location to insert an author-specified action into the plan. We believe that a planner must be capable to reason about determining the intended intention plan that an action constraint may be related to, and then use that also to guide the authorial heuristic appropriately. Future work will be directed to considering how heuristics can be improved to attend to authorial goals while considering the intent behind an enhanced authorial constraint specification.

5.2.3 Limitations on the Type of Action Failure

The work described in this dissertation focused on the production of narratives with a specific kind of failed action: an action that fails when the character attempting it has incorrect beliefs about the world. However, in human-generated plots, characters fail when attempting actions for a myriad of reasons, and belief-based failed actions are just one of many ways in which they can fail.

An action that fails due to chance are not modeled by HEADSPACE. For example, an action for winning the lottery has a non-zero probability of failing. Additionally, actions that fail because the character had incorrect beliefs about the conditions for successful action execution itself are outside of the scope of this work. Consider stories such as *Wizard of Oz*, where it is possible for characters to have completely beliefs about how an action can be executed, which could also lead to failure. HEADSPACE cannot produce failed actions of this nature- the knowledge representation does not currently support character's beliefs about how actions work.

5.2.4 Authorial Expressivity for Intent Constraints

We believe that further study is necessary about authorial expressivity concerning intent constraints. The current representation for intent constraints surrounding failed actions is limited in use: for example, what happens when an author might have two specific action constraints for the same action, but they want a character to persist in their goal for one instance but drop their goal in the other instance? Another consideration: should these constraints be associated with desires (since authors might think along the lines of *I want X to be rigid about pursuing goal A but be fickle when pursuing goal B*), or whether their current association with failed actions is more appropriate. Such boundary cases are out of the scope of the current representation used for representing intent constraints. While the algorithms described in this work are capable of using intent constraints to guide the planner, the representation currently used is limited in accessibility and robustness. Future work will further study how intent constraints can be effectively leveraged for computational narrative generation.

5.2.5 Model of Intent

Cohen and Levesque [22] describe principles on rational agent actions and describe properties for a model that characterizes intent. The plans generated by HEADSPACE align well with many of the properties put forward by Cohen and Levesque:

- When intentions arise, characters work to form plans to achieve them through the invocation of a character-specific micro-planner. The plans are formed in the belief space of the intending character, so that each character believes their intentions are achievable. When an intention is motivated but no character plan can be formed, that intention is never adopted.
- Character plans are formed in the context of the character's existing plans. Character plans formed for new intentions are a) consistent with the character's beliefs and intentions and b) can leverage existing commitments to action to achieve the new goals.
- If characters act at all, they work to follow the plans that they form.
- The conditions under which characters believe they will bring about their intentions are explicitly represented in their plans, through the motivating conditions for the intentions and the causal structure of the character plans.
- The distinction between conditions that characters intend and those that are side effects is made clear in the plan structures formed by characters' micro-planners.
- When characters detect that some aspect of their plans has failed, they review those plans and are able to either a) revise their plans in light of their new beliefs or b) drop the intention altogether.

Work by Rao and Georgeff [73] provided categories of agent commitment characterizing the contexts in which types of agents would drop intentions. They identify three types of commitment: blind, single-minded, and open-minded. In blind commitment, an agent (a) only considers new plans that are consistent with its current goals, (b) doesn't adopt new beliefs that conflict with its current intentions and (c) doesn't entertain any changes to goals that are inconsistent with its current intentions. In single-minded commitment, agents operate with comparable commitment to blind commitment agents except that they allow new beliefs that might invalidate the plans they already hold to achieve their goals. When adopting these new beliefs, single-minded agents drop their now inconsistent plans but maintain their old goals. In open-minded commitment, characters may also adopt new goals that are inconsistent with their old goals, causing them to drop both their old goals and the plans intended to achieve them. In HEADSPACE , characters currently use single-minded commitment. That is, they only consider new plans that are consistent with their current goals and don't adopt goals that are inconsistent with already adopted goals. However, characters in HEADSPACE always adopt new beliefs, overriding any inconsistent old beliefs and causing plan revision or goal dropping when characters new beliefs invalidate current plans. The single-mindedness of HEADSPACE characters is not meant to be the final and defining characterization for all characters in the stories it produces. This approach is the initial behavior defined here, but clearly stories see a wide range of goal and plan commitment in characters, and our approach will require extension to be able to support that range of behavior on a per character basis.

Additionally, intent dynamics upon action failure in HEADSPACE plans do not align with the dynamics proposed by the Goal-Action-Outcome cognitive model [95]. The Goal-Action-Outcome model proposes that characters may even drop the current goal to find adjacent goals that can be satisfied, and character may end up pursuing other desires upon a failed outcome of an action. The intent dynamics supported by HEADSPACE are limited compared to this model. While the Goal-Action-Outcome model is just one example of increasing expressivity around character plans, future work will consider this model and others reflecting more complex intent dynamics in addition to existing ones.

The structures produced by HEADSPACE have clear parallels to existing work [30, 76, 105, 108]. However, HEADSPACE structures lack certain complex intention-driven features present in other planners, such as theory of mind and characters acting based on other characters' beliefs and intents. Future work will seek to explore additions of those features in context with how they can work with the HEADSPACE model of belief-based action failure.

5.2.6 Evaluation of Intent Dynamics in HEADSPACE

The two empirical evaluations conducted in this dissertation do not comprehensively evaluate the intent dynamics that are possible in stories generated using the HEADSPACE planner. The evaluation for comprehension of stories with failed actions uses a story where the character performs plan repair and sticks to the initial plan upon action failure (`PERSIST`). Future work will consider a comprehensive evaluation of the possible intent dynamics surrounding action failure and how effectively readers can comprehend them in computationally generated stories. The final task used for the evaluation of enhanced authorial expressivity did not have any specific behaviors as part of the task: it did not evaluate the effectiveness of authors being able to specify intent constraints when working with narrative planning approaches. Future work will study these complex intent dynamics to understand better how people comprehend and use them.

Table 5.1: Blumenfeld Chart depicting the research questions and how they relate to the proposed work and evaluation. We have added a section to the Blumenfeld Chart that describes the approach used.

Research Question	Approach	Experiment	Analysis	Hypothesis
How can a narrative planner be used to generate stories with characters exhibiting expressive behavior surrounding failed actions?	- Build algorithms and knowledge representation that support failed actions, - Extend evaluation methods for story comprehension (QUEST) to support stories containing failed actions.	Generate stories containing failed actions and their respective cognitive models using QUEST. Generate question-answer pairs for people to rate as good or bad.	Standard one-tailed t-test comparing mean ratings for QA pairs with the arc distance calculated by QUEST.	If readers are able to comprehend the role of failed actions in stories generated by a planner, they will rate correct answers higher than incorrect answers for comprehension questions about the belief and intent dynamics behind failed actions in the story.
How does the addition of added expressivity for specification of authorial constraints surrounding actions, failed actions and intent dynamics surrounding failed actions impact the ability to create stories using a computational narrative planner?	- Build algorithm that support authors to specify actions, failed actions and intent dynamics surrounding failed actions in desired stories, - A user interface that allows authors to specify constraints and generate plans.	Provide authors with a task to generate stories with certain qualities in conditions where they have different levels of expressivity on specifications to the planner.	ANOVA tests comparing story generation effort over three conditions.	If a narrative planner that creates stories consisting of failed actions can support authorial expressivity related to failed action constraints, then authors who use the enhanced authorial expressivity can create desired stories with failed actions with lesser authorial effort than authors who do not.

APPENDIX A

THE HEADSPACE ALGORITHM

In this appendix chapter, we provide the pseudocode for other HEADSPACE functions. Algorithm 11 describes the algorithm used to compute the next worldstate given a world-state and a grounded operator. Algorithm 11 is the pseudocode version of the top-level algorithm, which was provided as a flow diagram in Figure 3.4.

```
getNextState(current, ground)
$newState \leftarrow current$
for all $lit \in ground$.effT do
  if $lit \in$ newState.F then
    newState.F $\leftarrow$ newState.F $- lit$
  end if
  if $lit \notin$ newState.T then
    newState.T $\leftarrow$ newState.T $\cup lit$
  end if
end for
for all $lit \in ground$.effF do
  if $lit \in$ newState.T then
    newState.T $\leftarrow$ newState.T $- lit$
  end if
  if $lit \notin$ newState.F then
    newState.F $\leftarrow$ newState.F $\cup lit$
  end if
end for
$c \leftarrow$ get character beliefs in newstate for $ground$.character
for all $lit \in ground$.effBPlus do
  $chars \leftarrow$ all characters that can observe $lit$
  for all $ch \in chars$ do
    $character \leftarrow$ get character beliefs in newstate for $ch$
    if $lit \in$ character.bMinus then
      character.bMinus $\leftarrow$ character.bMinus $-lit$
    end if
    if $lit \in$ character.unsure then
      character.unsure $\leftarrow$ character.unsure $-lit$
    end if
    if $lit \notin$ character.bPlus then
      character.bPlus $\leftarrow$ character.bPlus $\cup lit$
    end if
  end for
  $c.bPlus \leftarrow c.bPlus \cup lit$
end for
...............
// similarly, repeat for effBMinus and effUnsure
return $newState$
```

Algorithm 10: The getNextState function computes the resulting world state when an action *ground* is executed at worldstate *current*.

```
Data: desires, plan, goal, groundedoperators, fringe, worldsVisited
Result: A plan
1  $w \leftarrow$ Last WorldState in plan;
2  $intentions \leftarrow$ Extract Arising Desires$(w, desires)$;
3  if w is a Goal State for goal then
4      return plan;
5  end
6  Execute the AdoptIntentions step using plan, goal, grounded operators, w and
     intentions;
7  Execute the SelectAndExecuteAction step using plan, grounded operators, fringe,
     w, possible steps, worldsVisited, and desires;
8  if plan is Dead End, & Endless Plan, or Previously Generated Plan then
9      Attempt recovery from fringe if possible;
10 end
11 Recursively call this function using desires, the new plan, goal, grounded
     operators, fringe, and worldsVisited;
```

Algorithm 11: HeadSpace top-level planning algorithm.

APPENDIX B

AUXILIARY DATA FOR EVALUATION 1

In this appendix, we provide additional material that was used in Evaluation 1 described in Section 4.2.

As described, the experiments we ran made use of 2 distinct plans and corresponding QKSs and text realizations.

The plan in Figure B.1 was selected because, while the actions in it are distinct from those in the experimental plan shown in Figure 4.1, its structure is nearly identical. As a result, the two QKSs (shown in Figs 4.2 and B.2) are also similar. This similarity between the experimental materials was intentional to prevent any confounding variables between the two conditions. Both stories have the same number of actions, with the same structure. The character has three possible intention plans in both stories, with the intention plans being adopted and invalidated at the same points in the sequence of actions. This allows for keeping the conditions about the planning problem the same and reducing the bias from one of the "settings" of the stories on results.

B.1 Drink Refill Plan

Figure B.1 shows the second example plan used in our experiment, and Figure B.2 shows the corresponding QKS. Example Text 4 shows the corresponding story realized as text. Some of the QA pairs used in the Drink Refill problem are reported below. Texts 5-10 were used to filter out participants (5-7 are examples of good or very-good QA pairs, and texts 8-10 are examples of bad or very bad pairs). Example Texts 10-15 are QA pairs used in the analysis. Texts 11 through 13 have a calculated arc distance of 1, texts 14-15 have an arc distance of 2, and text 16 has an arc distance of 3.

Example Text 5. *Teddy is a bartender working in a bar. He wants to serve a customer their drink. There are bottles of beverage on the shelf. Teddy walks over to the shelf and picks up a bottle. He* *then attempts to pour a drink from it but fails. He checks the bottle and sees that the bottle is empty. Teddy then places the first bottle back on the shelf and picks up a new bottle. He attempts to pour a drink again and is successful. Teddy then serves the drink to the customer.*

Example Text 6. *What was the consequence of Teddy failing pouring the drink from the first bottle? He checked the first bottle and saw that it was empty.*

Example Text 7. *What was the consequence of Teddy failing pouring the drink from the first bottle? He wanted to check the first bottle.*

Example Text 8. *Why did Teddy want to pick up the first bottle? Because he wanted to pour a drink from the first bottle.*

Example Text 9. *What was the consequence of Teddy serving the drink to the customer? Teddy was unsure whether the first bottle was empty.*

Example Text 10. *What was the consequence of Teddy failing pouring the drink from the first bottle? He wanted to pick up the first bottle.*

Example Text 11. *What was the consequence of Teddy pouring the new drink successfully? He failed to pour the drink from the first bottle.*

Example Text 12. *Why did Teddy attempt to (unsuccessfully) pour the drink from the first bottle? Because the customer's glass was empty.*

Example Text 13. *Why did Teddy attempt to (unsuccessfully) pour the drink from the first bottle? Because Teddy was holding the first bottle.*

Example Text 14. *Why did Teddy attempt to (unsuccessfully) pour the drink from the first bottle? Because he wanted to pour a drink from the first bottle.*

Example Text 15. *Why did Teddy attempt to (unsuccessfully) pour the drink from the first bottle? Because he picked up the first bottle.*

Example Text 16. *Why did Teddy attempt to (unsuccessfully) pour the drink from the first bottle?In order to refill the customer's drink.*

Example Text 17. *Why did Teddy attempt to (unsuccessfully) pour the drink from the first bottle? Because he wanted to pick up the first bottle.*

B.2 Breakout Plan

This section provides the question-answer pairs used as part of the experiment with the Breakout plan. Example Texts 17 through 22 are QA pairs that were used as comprehension check questions (17 is a good QA pair and 18-22 are bad QA pairs). Example Texts 23 through 31 are used in the analysis. Texts 23 through 29 have a calculated arc distance of 1. Text 28 has a calculated arc distance of 2, and Text 29 has a calculated arc distance of 3.

Example Text 18. *What was the consequence of Dolores failing when shooting at the door lock? She checked the revolver and found that it was unloaded.*

Example Text 19. *What was the consequence of Dolores loading the revolver? She failed shooting at the door lock.*

Example Text 20. *What was the consequence of Dolores successfully shooting the jail door open? She failed shooting at the door lock.*

Example Text 21. *What was the consequence of Dolores picking up the bullets? She failed shooting at the door lock.*

Example Text 22. *What was the consequence of Dolores loading the revolver? She wanted to load the revolver.*

Example Text 23. *What was the consequence of Dolores checking the revolver and finding it unloaded? She failed shooting at the door lock.*

Example Text 24. *Why did Dolores try to (unsuccessfully) shoot at the door lock? Because Dolores believed that the revolver was loaded.*

Example Text 25. *Why did Dolores try to (unsuccessfully) shoot at the door lock? Because she wanted to shoot at the jail door lock.*

Example Text 26. *Why did Dolores' attempt to shoot fail? Because the revolver was not loaded.*

Example Text 27. *Why did Dolores try to (unsuccessfully) shoot at the door lock? Because Dolores had the gun.*

Example Text 28. *Why did Dolores try to (unsuccessfully) shoot at the door lock? Because the jail door was locked.*

Example Text 29. *Why did Dolores try to (unsuccessfully) shoot at the door lock? In order to escape from prison.*

Example Text 30. *Why did Dolores try to (unsuccessfully) shoot at the door lock? Because she wanted to pick up the revolver.*

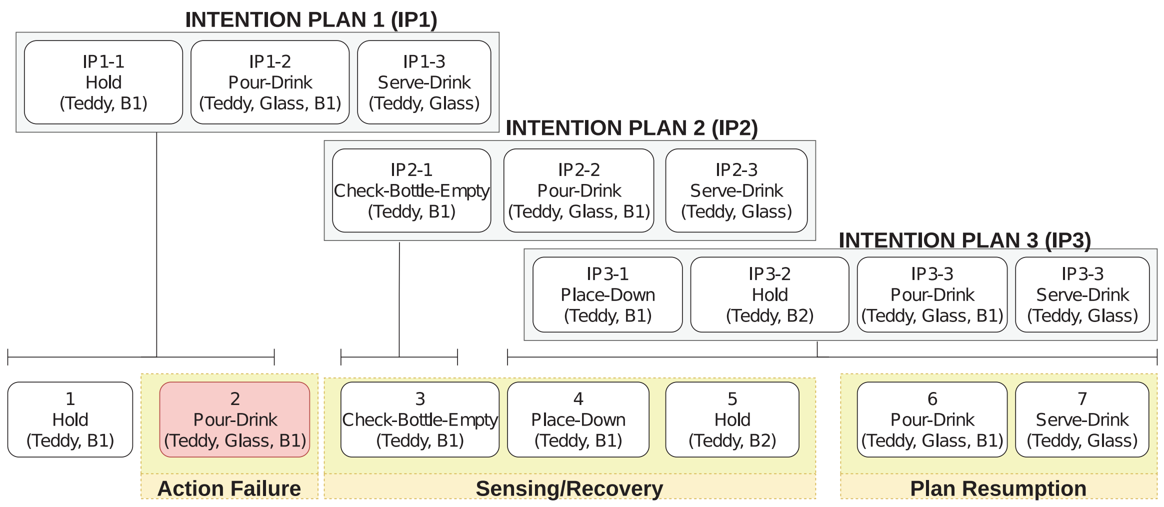

INTENTION PLAN 1 (IP1)
IP1-1 Hold (Teddy, B1)
IP1-2 Pour-Drink (Teddy, Glass, B1)
IP1-3 Serve-Drink (Teddy, Glass)
INTENTION PLAN 2 (IP2)
IP2-1 Check-Bottle-Empty (Teddy, B1)
IP2-2 Pour-Drink (Teddy, Glass, B1)
IP2-3 Serve-Drink (Teddy, Glass)
INTENTION PLAN 3 (IP3)
IP3-1 Place-Down (Teddy, B1)
IP3-2 Hold (Teddy, B2)
IP3-3 Pour-Drink (Teddy, Glass, B1)
IP3-3 Serve-Drink (Teddy, Glass)
1 Hold (Teddy, B1)
2 Pour-Drink (Teddy, Glass, B1)
3 Check-Bottle-Empty (Teddy, B1)
4 Place-Down (Teddy, B1)
5 Hold (Teddy, B2)
6 Pour-Drink (Teddy, Glass, B1)
7 Serve-Drink (Teddy, Glass)
Action Failure
Sensing/Recovery
Plan Resumption

Figure B.1. The plan used in the second experimental domain. The red action was attempted but failed because the non-belief preconditions are not all met in the world state where they are attempted.

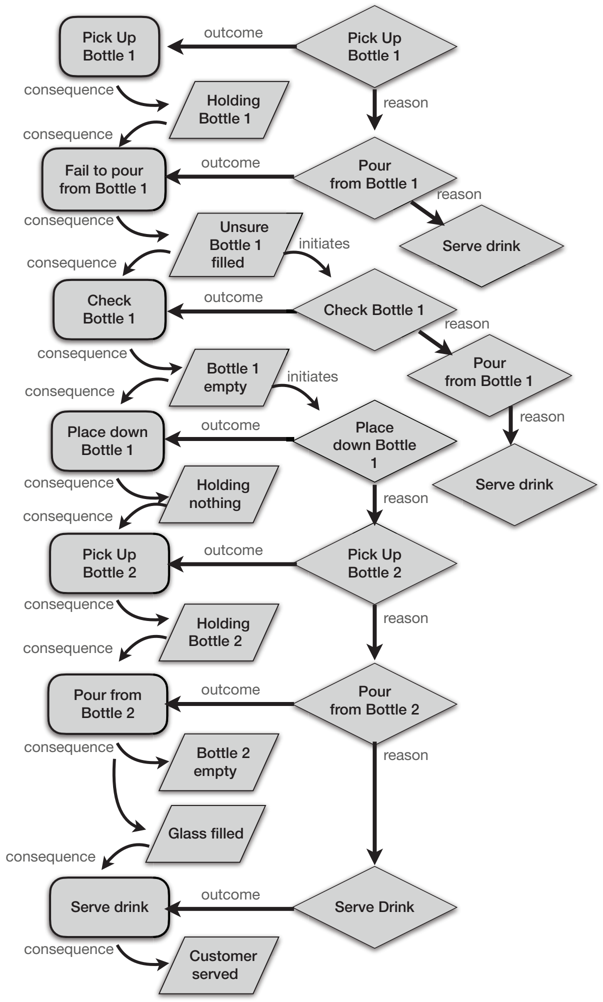


Figure B.2. The QKS corresponding to the plan from Figure B.1. Rounded rectangle indicate event nodes. Skewed rectangles indicate state nodes. Diamonds indicate goal nodes. Arcs from one node to another are labeled with one of the QUEST relationship types showing in Table 2.1.

APPENDIX C

AUXILIARY DATA FOR EVALUATION 2

In Listing C.1, we provide a PDDL-like JSON representation of the Kitchen domain. We describe the typed heirarchies in the `types`, all the objects in the domain in `instances`, the character desires in `desires`, and all the operator definitions in `operators`.

We provide the text used in each task during the study. Tasks 1 through 7 were performed by all groups, Task 8 was performed by participants in the Limited Expressivity and Full Expressivity groups, and Tasks 9 and 10 were performed by participants in the Full Expressivity group. Task 11 is the final task that all groups were measured on.

Example Text 31. ***(Task 1)*** *Stories have a beginning and ending. Find a story where Teddy starts in LivingRoom, goes to the Kitchen, heats up the soup, and eats the soup.*
Task Objectives:

- *Teddy starts in the Living Room.*
- *Soup is not heated up initially.*
- *Teddy heats up the soup in the created story.*
- *Teddy eats the soup in the created story.*

Example Text 32. ***(Task 2)*** *Different characters have different desires. While Teddy will always have soup, Poppy will always have bread. Bread needs to be heated using the Toaster. Find a story where Poppy starts in the LivingRoom, goes to the Kitchem, heats the Bread and then eats it.*
Task Objectives:

- *Poppy starts in the Living Room.*
- *Bread is not heated up initially.*
- *Poppy heats up the Bread in the created story.*

- *Poppy eats the Bread in the created story.*

Example Text 33. ***(Task 3)*** *Now back to Teddy. Teddy can take different plans to achieve his goal. Try creating a story where Teddy uses an alternate approach.*

- *You can click the Create Story button again to create other stories with the same beginning and ending.*

Task Objectives:

- *Soup is not heated up initially.*
- *Teddy heats up the soup without using the microwave in the created story.*

Example Text 34. ***(Task 4) For this task, do not change details in the Beginning/Ending tabs except the ones specified.*** *Keep generating different stories until you find one where Teddy disconnects the Microwave and uses the stove to heat the soup.*

- *You can click the Create Story button again to create other stories with the same beginning and ending.*

Task Objectives:

- *Beginning Tab: Soup has been heated is set to No.*
- *Ending Tab: The Soup has been eaten by Teddy.*
- *No other changes made in the Beginning/Ending tabs.*
- *Teddy disconnects the Microwave in the story.*
- *Teddy heats up the soup without using the microwave in the story.*
- *Teddy eats the soup in the story.*

Example Text 35. ***(Task 5)*** *In this story world, characters can have beliefs about what is powered by the power outlet, which can be different from what is actually being powered. Characters can fail when they attempt an action with incorrect beliefs. If Teddy believes the Microwave is plugged in* *but it isn't plugged in, he will fail. Try creating a story where he fails.*

- *To fail an action, the character must have incorrect belief related to the action!*

Task Objectives:

- *In the beginning, the outlet is not powering the Microwave.*
- *In the beginning, Teddy believes the Microwave is plugged in.*
- *In the story, Teddy (initially) attempts and fails to heat the Soup using the Microwave.*
- *Teddy eventually heats up the soup in the story.*
- *Teddy still eats the soup in the story.*

Example Text 36. ***(Task 6)*** *Upon failing, characters can choose an alternate plan, or repair the initial plan. Try creating multiple stories where Teddy fails to use the Microwave initially to explore different outcomes.*

- *Remember, you can click on Create Story multiple times to find more stories.*

Task Objectives:

- *In the story, Teddy (initially) attempts and fails to heat the Soup using the Microwave.*
- *Create a story where Teddy heats up the soup over the stove.*
- *Create a story where Teddy heats up the soup using the microwave.*

Example Text 37. ***(Task 7)*** *By changing the goals, you can explore different endings of a story when a character fails. For example, create a story where Teddy ends up not eating the Soup.*

- *You can also add beliefs as goals in the Ending tab.*

Task Objectives:

- *In the story, Teddy attempts and fails to heat the Soup using the Microwave.*

- *The soup has not been eaten in the end of the story.*

Example Text 38. ***(Task 8)*** *Using action constraints in the Actions tab, you can directly specify what actions you'd like to see in the story. Try using action constraints to specifically guide Teddy to use the stovetop to heat up the Soup.*

- *Action constraints help you specify what actions you would like to see in the story.*

Task Objectives:

- *Soup is not heated up initially.*
- *An action constraint specifies that Teddy heats up the soup using the stove.*
- *Teddy eats the soup in the story.*

Example Text 39. ***(Task 9)*** *You can also specify failed actions in action constraints. Create (any) story where Teddy fails to use the microwave by specifying the failed attempt to use the microwave.*

- *Action constraints can also include a failed action attempt.*

Task Objectives:

- *In the story, Teddy attempts and fails to heat the Soup using the Microwave.*
- *An action constraint specifies that Teddy attempts and fails to use the Microwave.*

Example Text 40. ***(Task 10)****In an action constraint, you can also specify how you want the character to behave when they fail an action. Characters can either persist with their initial plan, substitute it, or drop their goal entirely.*

- *Specify an intent constraint with an action constraint by clicking on the settings icon.*

Task Objectives:

- *An action constraint specifies that Teddy attempts and fails to use the Microwave.*
- *An action constraint also has an intent specified.*

Listing C.1. The Kitchen Domain

```
{
    "types": [
      {
        "name": "Character",
        "type": "Object"
      },
      {
        "name": "Location",
        "type": "Object"
      },
      {
        "name": "Appliance",
        "type": "Object"
      },
      {
        "name": "Container",
        "type": "Object"
      },
      {
        "name": "CookableContainer",
        "type": "Container"
      },
      {
        "name": "MicrowaveSafeContainer",
        "type": "Container"
      },
      {
        "name": "Food",
        "type": "Object"
      },
      {
        "name": "ToastableFood",
        "type": "Food"
      },
      {
        "name": "CookableFood",
        "type": "Food"
      }
    ],
    "instances": [
      {
        "name": "Teddy",
        "type": "Character"
      },
      {
        "name": "Poppy",
        "type": "Character"
      },
      {
        "name": "Microwave",
        "type": "Appliance"
      },
      {
```

```
      "name": "Toaster",
      "type": "Appliance"
    },
    {
      "name": "Kitchen",
      "type": "Location"
    },
    {
      "name": "LivingRoom",
      "type": "Location"
    },
    {
      "name": "Bread",
      "type": "ToastableFood"
    },
    {
      "name": "Soup",
      "type": "CookableFood"
    },
    {
      "name": "Pot",
      "type": "CookableContainer"
    },
    {
      "name": "Bowl",
      "type": "MicrowaveSafeContainer"
    }
  ],
  "desires": [
      {
          "character": "Teddy",
          "motivations": {
            "bplus": [
            ],
            "bminus": [
            ],
            "unknown": []
          },
          "goal": {
            "bplus": [
              "(eaten Soup Teddy)"
            ],
            "bminus": [],
            "unknown": []
          }
        },
        {
          "character": "Poppy",
          "motivations": {
            "bplus": [
            ],
            "bminus": [
            ],
            "unknown": []
            },
            "goal": {
              "bplus": [
                "(eaten Bread Poppy)"
              ],
              "bminus": [],
              "unknown": []
            }
          }
    ],
  "operators": [
    {
      "name": "walk",
      "args": [
        {
          "name": "?char",
          "type": "Character"
        },
        {
          "name": "?fromLoc",
          "type": "Location"
        },
        {
          "name": "?toLoc",
          "type": "Location"
        }
      ],
      "char": "?char",
      "loc": "?fromLoc",
      "pre-t": [
        "(at ?char ?fromLoc)"
      ],
      "pre-f": [
        "(at ?char ?toLoc)"
      ],
      "eff-t": [
        "(at ?char ?toLoc)"
      ],
      "eff-f": [
        "(at ?char ?fromLoc)"
      ],
      "pre-bplus": [
        "(at ?char ?fromLoc)"
      ],
      "pre-bminus": [
        "(at ?char ?toLoc)"
      ],
      "pre-u": [],
      "eff-bplus": [
        {
          "effect": "(at ?char ?toLoc)",
          "observability": [
            "localObs(?toLoc)",
            "localObs(?fromLoc)"
        ]
      }
    ],
    "eff-bminus": [
      {
        "effect": "(at ?char ?fromLoc)",
        "observability": [
          "localObs(?toLoc)",
          "localObs(?fromLoc)"
        ]
      }
    ],
    "eff-u": []
  },
  {
    "name": "connect-true",
    "args": [
      {
        "name": "?appliance",
        "type": "Appliance"
      },
      {
        "name": "?char",
        "type": "Character"
      }
    ],
    "char": "?char",
    "loc": "Kitchen",
    "pre-t": [
      "(at ?char Kitchen)",
      "(outlet-empty)"
    ],
    "pre-f": [
      "(plugged ?appliance)"
    ],
    "eff-t": [
      "(plugged ?appliance)"
    ],
    "eff-f": [
      "(outlet-empty)"
    ],
    "pre-bplus": [
      "(at ?char Kitchen)",
      "(outlet-empty)"
    ],
    "pre-bminus": [
      "(plugged ?appliance)"
    ],
    "pre-u": [],
    "eff-bplus": [
      {
        "effect": "(plugged ?appliance)",
        "observability": [
          "localObs(Kitchen)"
        ]
      }
    ],
    "eff-bminus": [
      {
        "effect": "(outlet-empty)",
        "observability": [
          "localObs(Kitchen)"
        ]
      }
    ],
    "eff-u": []
  },
  {
    "name": "connect-false1",
    "args": [
      {
        "name": "?appliance",
        "type": "Appliance"
      },
      {
        "name": "?char",
        "type": "Character"
      }
    ],
    "char": "?char",
    "loc": "Kitchen",
    "pre-t": [
      "(at ?char Kitchen)"
    ],
    "pre-f": [
      "(plugged ?appliance)",
      "(outlet-empty)"
    ],
    "eff-t": [
    ],
    "eff-f": [
    ],
    "pre-bplus": [
      "(at ?char Kitchen)",
      "(outlet-empty)"
    ],
    "pre-bminus": [
      "(plugged ?appliance)"
    ],
    "pre-u": [],
    "eff-bplus": [
    ],
    "eff-bminus": [
      {
        "effect": "(outlet-empty)",
        "observability": [
          "localObs(Kitchen)"
        ]
      }
    ],
    "eff-u": [
      {
        "effect": "(plugged Toaster)",
        "observability": [
          "localObs(Kitchen)"
        ]
      },
      {
        "effect": "(plugged Microwave)",
        "observability": [
          "localObs(Kitchen)"
        ]
      }
    ]
  },
  {
    "name": "connect-false2",
    "args": [
      {
        "name": "?appliance",
        "type": "Appliance"
      },
      {
        "name": "?char",
        "type": "Character"
      }
    ],
    "char": "?char",
    "loc": "Kitchen",
    "pre-t": [
      "(at ?char Kitchen)",
      "(plugged ?appliance)"
    ],
    "pre-f": [
      "(outlet-empty)"
    ],
    "eff-t": [
    ],
    "eff-f": [
    ],
    "pre-bplus": [
      "(at ?char Kitchen)",
      "(outlet-empty)"
    ],
    "pre-bminus": [
      "(plugged ?appliance)"
    ],
    "pre-u": [],
    "eff-bplus": [
      {
        "effect": "(plugged ?appliance)",
        "observability": [
        "localObs(Kitchen)"
      ]
    }
  ],
  "eff-bminus": [
    {
      "effect": "(outlet-empty)",
      "observability": [
        "localObs(Kitchen)"
      ]
    }
  ],
  "eff-u": [
  ]
},
{
  "name": "disconnect-true",
  "args": [
    {
      "name": "?appliance",
      "type": "Appliance"
    },
    {
      "name": "?char",
      "type": "Character"
    }
  ],
  "char": "?char",
  "loc": "Kitchen",
  "pre-t": [
    "(at ?char Kitchen)",
    "(plugged ?appliance)"
  ],
  "pre-f": [
    "(outlet-empty)"
  ],
  "eff-t": [
    "(outlet-empty)"
  ],
  "eff-f": [
    "(plugged ?appliance)"
  ],
  "pre-bplus": [
    "(at ?char Kitchen)",
    "(plugged ?appliance)"
  ],
  "pre-bminus": [
    "(outlet-empty)"
  ],
  "pre-u": [],
  "eff-bplus": [
    {
      "effect": "(outlet-empty)",
      "observability": [
        "localObs(Kitchen)"
      ]
    }
  ],
  "eff-bminus": [
    {
      "effect": "(plugged Toaster)",
      "observability": [
        "localObs(Kitchen)"
      ]
    },
    {
      "effect": "(plugged Microwave)",
      "observability": [
        "localObs(Kitchen)"
      ]
    }
  ],
  "eff-u": []
},
{
  "name": "disconnect-false1",
  "args": [
    {
      "name": "?appliance",
      "type": "Appliance"
    },
    {
      "name": "?char",
      "type": "Character"
    }
  ],
  "char": "?char",
  "loc": "Kitchen",
  "pre-t": [
    "(at ?char Kitchen)",
    "(outlet-empty)"
  ],
  "pre-f": [
    "(plugged ?appliance)"
  ],
  "eff-t": [
  ],
  "eff-f": [
  ],
  "pre-bplus": [
    "(at ?char Kitchen)",
    "(plugged ?appliance)"
  ],
  "pre-bminus": [
    "(outlet-empty)"
  ],
  "pre-u": [],
  "eff-bplus": [
    {
      "effect": "(outlet-empty)",
      "observability": [
        "localObs(Kitchen)"
      ]
    }
  ],
  "eff-bminus": [
    {
      "effect": "(plugged ?appliance)",
      "observability": [
        "localObs(Kitchen)"
      ]
    }
  ],
  "eff-u": []
},
{
  "name": "disconnect-false2",
  "args": [
    {
      "name": "?appliance",
      "type": "Appliance"
    },
    {
      "name": "?char",
      "type": "Character"
    }
  ],
  "char": "?char",
  "loc": "Kitchen",
  "pre-t": [
    "(at ?char Kitchen)",
    "(plugged Microwave)"
  ],
  "pre-f": [
    "(plugged ?appliance)",
    "(plugged Toaster)",
    "(outlet-empty)"
  ],
  "eff-t": [
  ],
  "eff-f": [
  ],
  "pre-bplus": [
    "(at ?char Kitchen)",
    "(plugged ?appliance)"
  ],
  "pre-bminus": [
    "(outlet-empty)"
  ],
  "pre-u": [],
  "eff-bplus": [
    {
        "effect": "(plugged Microwave)",
        "observability": [
          "localObs(Kitchen)"
        ]
      }
    ],
    "eff-bminus": [
      {
        "effect": "(plugged ?appliance)",
        "observability": [
          "localObs(Kitchen)"
        ]
      },
      {
        "effect": "(outlet-empty)",
        "observability": [
          "localObs(Kitchen)"
        ]
      }
    ],
    "eff-u": []
  },
  {
    "name": "disconnect-false3",
    "args": [
      {
        "name": "?appliance",
        "type": "Appliance"
      },
      {
        "name": "?char",
        "type": "Character"
      }
    ],
    "char": "?char",
    "loc": "Kitchen",
    "pre-t": [
      "(at ?char Kitchen)",
      "(plugged Toaster)"
    ],
    "pre-f": [
      "(plugged ?appliance)",
      "(plugged Microwave)",
      "(outlet-empty)"
    ],
    "eff-t": [
    ],
    "eff-f": [
    ],
    "pre-bplus": [
      "(at ?char Kitchen)",
      "(plugged ?appliance)"
    ],
    "pre-bminus": [
    "(outlet-empty)"
  ],
  "pre-u": [],
  "eff-bplus": [
    {
      "effect": "(plugged Toaster)",
      "observability": [
        "localObs(Kitchen)"
      ]
    }
  ],
  "eff-bminus": [
    {
      "effect": "(plugged ?appliance)",
      "observability": [
        "localObs(Kitchen)"
      ]
    },
    {
      "effect": "(outlet-empty)",
      "observability": [
        "localObs(Kitchen)"
      ]
    }
  ],
  "eff-u": []
},
{
  "name": "toast-true",
  "args": [
    {
      "name": "?food",
      "type": "ToastableFood"
    },
    {
      "name": "?char",
      "type": "Character"
    }
  ],
  "char": "?char",
  "loc": "Kitchen",
  "pre-t": [
    "(at ?char Kitchen)",
    "(plugged Toaster)"
  ],
  "pre-f": [
    "(eaten ?food Teddy)",
    "(eaten ?food Poppy)",
    "(heated ?food)",
    "(infridge ?food)"
  ],
  "eff-t": [
    "(heated ?food)"
  ],
    "eff-f": [
    ],
    "pre-bplus": [
      "(at ?char Kitchen)",
      "(plugged Toaster)"
    ],
    "pre-bminus": [
      "(eaten ?food Teddy)",
      "(eaten ?food Poppy)",
      "(heated ?food)",
      "(infridge ?food)"
    ],
    "pre-u": [],
    "eff-bplus": [
      {
        "effect": "(heated ?food)",
        "observability": [
          "publicObs(Kitchen)"
        ]
      }
    ],
    "eff-bminus": [
    ],
    "eff-u": []
  },
  {
    "name": "toast-false",
    "args": [
      {
        "name": "?food",
        "type": "ToastableFood"
      },
      {
        "name": "?char",
        "type": "Character"
      }
    ],
    "char": "?char",
    "loc": "Kitchen",
    "pre-t": [
      "(at ?char Kitchen)"
    ],
    "pre-f": [
      "(eaten ?food Teddy)",
      "(eaten ?food Poppy)",
      "(heated ?food)",
      "(infridge ?food)",
      "(plugged Toaster)"
    ],
    "eff-t": [
    ],
    "eff-f": [
    ],
    "pre-bplus": [
      "(at ?char Kitchen)",
      "(plugged Toaster)"
    ],
    "pre-bminus": [
      "(eaten ?food Teddy)",
      "(eaten ?food Poppy)",
      "(heated ?food)",
      "(infridge ?food)"
    ],
    "pre-u": [],
    "eff-bplus": [
    ],
    "eff-bminus": [
    ],
    "eff-u": [
      {
        "effect": "(plugged Toaster)",
        "observability": [
          "localObs(Kitchen)"
        ]
      }
    ]
  },
  {
    "name": "heatWithMicrowave-true",
    "args": [
      {
        "name": "?food",
        "type": "CookableFood"
      },
      {
        "name": "?container",
        "type": "MicrowaveSafeContainer"
      },
      {
        "name": "?char",
        "type": "Character"
      }
    ],
    "char": "?char",
    "loc": "Kitchen",
    "pre-t": [
      "(at ?char Kitchen)",
      "(plugged Microwave)",
      "(contained-in ?food ?container)"
    ],
    "pre-f": [
      "(eaten ?food Teddy)",
      "(eaten ?food Poppy)",
      "(heated ?food)",
      "(infridge ?food)"
    ],
    "eff-t": [
      "(heated ?food)"
    ],
    "eff-f": [
    ],
    "pre-bplus": [
      "(at ?char Kitchen)",
      "(plugged Microwave)",
      "(contained-in ?food ?container)"
    ],
    "pre-bminus": [
      "(eaten ?food Teddy)",
      "(eaten ?food Poppy)",
      "(heated ?food)",
      "(infridge ?food)"
    ],
    "pre-u": [],
    "eff-bplus": [
      {
        "effect": "(heated ?food)",
        "observability": [
          "publicObs(Kitchen)"
        ]
      }
    ],
    "eff-bminus": [
    ],
    "eff-u": []
  },
  {
    "name": "heatWithMicrowave-false",
    "args": [
      {
        "name": "?food",
        "type": "CookableFood"
      },
      {
        "name": "?container",
        "type": "MicrowaveSafeContainer"
      },
      {
        "name": "?char",
        "type": "Character"
      }
    ],
    "char": "?char",
    "loc": "Kitchen",
    "pre-t": [
      "(at ?char Kitchen)",
      "(contained-in ?food ?container)"
    ],
    "pre-f": [
      "(eaten ?food Teddy)",
      "(eaten ?food Poppy)",
      "(heated ?food)",
      "(infridge ?food)",
      "(plugged Microwave)"
    ],
    "eff-t": [
    ],
    "eff-f": [
    ],
    "pre-bplus": [
      "(at ?char Kitchen)",
      "(plugged Microwave)",
      "(contained-in ?food ?container)"
    ],
    "pre-bminus": [
      "(eaten ?food Teddy)",
      "(eaten ?food Poppy)",
      "(heated ?food)",
      "(infridge ?food)"
    ],
    "pre-u": [],
    "eff-bplus": [
    ],
    "eff-bminus": [
    ],
    "eff-u": [
      {
        "effect": "(plugged Microwave)",
        "observability": [
          "localObs(Kitchen)"
        ]
      }
    ]
  },
  {
    "name": "heatWithStove",
    "args": [
      {
        "name": "?food",
        "type": "CookableFood"
      },
      {
        "name": "?container",
        "type": "CookableContainer"
      },
      {
        "name": "?char",
        "type": "Character"
      }
    ],
    "char": "?char",
    "loc": "Kitchen",
    "pre-t": [
      "(at ?char Kitchen)",
      "(contained-in ?food ?container)"
    ],
    "pre-f": [
      "(eaten ?food Teddy)",
      "(eaten ?food Poppy)",
      "(heated ?food)",
      "(infridge ?food)"
    ],
    "eff-t": [
      "(heated ?food)"
    ],
    "eff-f": [
    ],
    "pre-bplus": [
      "(at ?char Kitchen)",
      "(contained-in ?food ?container)"
    ],
    "pre-bminus": [
      "(eaten ?food Teddy)",
      "(eaten ?food Poppy)",
      "(heated ?food)",
      "(infridge ?food)"
    ],
    "pre-u": [],
    "eff-bplus": [
      {
        "effect": "(heated ?food)",
        "observability": [
          "publicObs(Kitchen)"
        ]
      }
    ],
    "eff-bminus": [
    ],
    "eff-u": []
  },
  {
    "name": "switchContainer",
    "args": [
      {
        "name": "?food",
        "type": "CookableFood"
      },
      {
        "name": "?fromContainer",
        "type": "Container"
      },
      {
        "name": "?toContainer",
        "type": "Container"
      },
      {
        "name": "?char",
        "type": "Character"
      }
    ],
    "char": "?char",
    "loc": "Kitchen",
    "pre-t": [
      "(at ?char Kitchen)",
      "(contained-in ?food ?fromContainer)"
    ],
    "pre-f": [
      "(contained-in ?food ?toContainer)",
      "(eaten ?food Teddy)",
      "(eaten ?food Poppy)",
      "(infridge ?food)"
    ],
    "eff-t": [
      "(contained-in ?food ?toContainer)"
    ],
    "eff-f": [
      "(contained-in ?food ?fromContainer)"
    ],
    "pre-bplus": [
      "(at ?char Kitchen)",
      "(contained-in ?food ?fromContainer)"
    ],
    "pre-bminus": [
      "(contained-in ?food ?toContainer)",
      "(eaten ?food Teddy)",
      "(eaten ?food Poppy)",
      "(infridge ?food)"
    ],
    "pre-u": [],
    "eff-bplus": [
      {
        "effect": "(contained-in ?food ?toContainer)",
        "observability": [
          "publicObs(Kitchen)"
        ]
      }
    ],
    "eff-bminus": [
      {
        "effect": "(contained-in ?food ?fromContainer)",
        "observability": [
          "publicObs(Kitchen)"
        ]
      }
    ],
    "eff-u": []
  },
  {
    "name": "checkOutlet-true",
    "args": [
      {
        "name": "?char",
        "type": "Character"
      }
    ],
    "char": "?char",
    "loc": "Kitchen",
    "pre-t": [
      "(at ?char Kitchen)",
      "(plugged Microwave)"
    ],
    "pre-f": [],
    "eff-t": [],
    "eff-f": [],
    "pre-bplus": [
      "(at ?char Kitchen)"
    ],
    "pre-bminus": [],
    "pre-u": [
      "(plugged Microwave)"
    ],
    "eff-bplus": [
      {
        "effect": "(plugged Microwave)",
        "observability": [
          "localObs(Kitchen)"
        ]
      }
    ],
    "eff-bminus": [],
    "eff-u": []
  },
  {
    "name": "checkOutlet-false1",
    "args": [
      {
        "name": "?char",
        "type": "Character"
      }
    ],
    "char": "?char",
    "loc": "Kitchen",
    "pre-t": [
      "(at ?char Kitchen)",
      "(plugged Toaster)"
    ],
    "pre-f": [
      "(plugged Microwave)",
      "(outlet-empty)"
    ],
    "eff-t": [],
    "eff-f": [],
    "pre-bplus": [
      "(at ?char Kitchen)"
    ],
    "pre-bminus": [],
    "pre-u": [
      "(plugged Microwave)"
    ],
    "eff-bplus": [
      {
        "effect": "(plugged Toaster)",
        "observability": [
          "localObs(Kitchen)"
        ]
      }
    ],
    "eff-bminus": [
      {
        "effect": "(plugged Microwave)",
        "observability": [
          "localObs(Kitchen)"
        ]
      },
      {
        "effect": "(outlet-empty)",
        "observability": [
          "localObs(Kitchen)"
        ]
      }
    ],
    "eff-u": []
  },
  {
    "name": "checkOutlet-false2",
    "args": [
      {
        "name": "?char",
        "type": "Character"
      }
    ],
    "char": "?char",
    "loc": "Kitchen",
    "pre-t": [
      "(at ?char Kitchen)",
      "(outlet-empty)"
    ],
    "pre-f": [
      "(plugged Microwave)",
      "(plugged Toaster)"
    ],
    "eff-t": [],
    "eff-f": [],
    "pre-bplus": [
      "(at ?char Kitchen)"
    ],
    "pre-bminus": [],
    "pre-u": [
      "(plugged Microwave)"
    ],
    "eff-bplus": [
      {
        "effect": "(outlet-empty)",
          "observability": [
            "localObs(Kitchen)"
          ]
        }
      ],
      "eff-bminus": [
        {
          "effect": "(plugged Microwave)",
          "observability": [
            "localObs(Kitchen)"
          ]
        },
        {
          "effect": "(plugged Toaster)",
          "observability": [
            "localObs(Kitchen)"
          ]
        }
      ],
      "eff-u": []
    },
    {
      "name": "removeFromFridge",
      "args": [
        {
          "name": "?food",
          "type": "Food"
        },
        {
          "name": "?char",
          "type": "Character"
        }
      ],
      "char": "?char",
      "loc": "Kitchen",
      "pre-t": [
        "(at ?char Kitchen)",
        "(infridge ?food)"
      ],
      "pre-f": [
      ],
      "eff-t": [],
      "eff-f": [
        "(infridge ?food)"
      ],
      "pre-bplus": [
        "(at ?char Kitchen)",
        "(infridge ?food)"
      ],
      "pre-bminus": [
      ],
      "pre-u": [
      ],
      "eff-bplus": [],
    "eff-bminus": [
      {
        "effect": "(infridge ?food)",
        "observability": [
          "publicObs(Kitchen)"
        ]
      }
    ],
    "eff-u": []
  },
  {
    "name": "eat",
    "args": [
      {
        "name": "?food",
        "type": "Food"
      },
      {
        "name": "?char",
        "type": "Character"
      }
    ],
    "char": "?char",
    "loc": "Kitchen",
    "pre-t": [
      "(at ?char Kitchen)",
      "(heated ?food)"
    ],
    "pre-f": [
      "(eaten ?food Teddy)",
      "(eaten ?food Poppy)",
      "(infridge ?food)"
    ],
    "eff-t": [
      "(eaten ?food ?char)"
    ],
    "eff-f": [
    ],
    "pre-bplus": [
      "(at ?char Kitchen)",
      "(heated ?food)"
    ],
    "pre-bminus": [
      "(eaten ?food Teddy)",
      "(eaten ?food Poppy)",
      "(infridge ?food)"
    ],
    "pre-u": [],
    "eff-bplus": [
      {
        "effect": "(eaten ?food ?char)",
        "observability": [
          "publicObs(?loc)"
        ]
```

```
          }
        ],
        "eff-bminus": [
        ],
        "eff-u": []
      }
    ]
    ...
}
```

REFERENCES

[1] N. Akoury, S. Wang, J. Whiting, S. Hood, N. Peng, and M. Iyyer, *Storium: A dataset and evaluation platform for machine-in-the-loop story generation*, in Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2020, pp. 6470–6484.

[2] V. Aleven, B. M. McLaren, J. Sewall, and K. R. Koedinger, *The cognitive tutor authoring tools (ctat): Preliminary evaluation of efficiency gains*, in Intelligent Tutoring Systems: 8th International Conference, ITS 2006, Jhongli, Taiwan, June 26-30, 2006. Proceedings 8, Springer, 2006, pp. 61–70.

[3] A. I. Alhussain and A. M. Azmi, *Automatic story generation: a survey of approaches*, ACM Computing Surveys (CSUR), 54 (2021), pp. 1–38.

[4] A. A. Amos-Binks, *Intention Revision of Plan-Based Agents for Narrative Generation*, PhD thesis, North Carolina State University, 2018.

[5] C. Bäckström and B. Nebel, *Complexity results for sas+ planning*, Computational Intelligence, 11 (1995), pp. 625–655.

[6] J. C. Bahamón, C. Barot, and R. M. Young, *A goal-based model of personality for planning-based narrative generation.*, in AAAI, 2015, pp. 4142–4143.

[7] J. C. Bahamón and R. M. Young, *An empirical evaluation of a generative method for the expression of personality traits through action choice*, in Proceedings of the Annual Conference on Artificial Intelligence And Interactive Digital Entertainment, 2017.

[8] D. Benmergui, *Storyteller*. Videogame, 2023.

[9] L. Bonassi, A. E. Gerevini, and E. Scala, *Planning with qualitative action-trajectory constraints in pddl*, in Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, Vienna, Austria, Jul 2022, International Joint Conferences on Artificial Intelligence Organization, p. 4606–4613.

[10] B. Bonet and H. Geffner, *Planning with incomplete information as heuristic search in belief space*, in Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, 2000, pp. 52–61.

[11] ——, *Heuristic search planner 2.0*, AI Magazine, 22 (2001), pp. 77–77.

[12] M. Bratman, *Intentions, Plans, and Practical Reason*, Harvard University Press, Cambridge, MA, 1987.

[13] J. Bruner, *The narrative construction of reality*, Critical Inquiry, 18 (1991), pp. 1–21.

[14] R. E. Cardona-Rivera, T. Price, D. Winer, and R. M. Young, *Question answering in the context of stories generated by computers*, Advances in Cognitive Systems, 4 (2016), pp. 227–245.

[15] M. Cavazza, F. Charles, and S. Mead, *Intelligent virtual actors that plan... to fail*, in Smart Graphics, Springer, 2003, pp. 343–368.

[16] M. Cavazza, J.-L. Lugrin, D. Pizzi, and F. Charles, *Madame bovary on the holodeck: immersive interactive storytelling*, in Proceedings of the 15th ACM International Conference on Multimedia, 2007, pp. 651–660.

[17] Y. Cheong and R. Young, *A computational model of narrative generation for suspense*, in Working Notes of the AAAI 2006 Workshop on Computational Aesthetics, 2006, pp. 8–14.

[18] Y.-G. Cheong and R. M. Young, *Suspenser: A story generation system for suspense*, IEEE Transactions on Computational Intelligence and AI in Games, 7 (2014), pp. 39–52.

[19] M. Christensen, J. Nelson, and R. Cardona-Rivera, *Using domain compilation to add belief to narrative planners*, in Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, vol. 16, 2020, pp. 38–44.

[20] D. Christian and R. M. Young, *Comparing cognitive and computational models of narrative structure*, in Proceedings of the National Conference on Artificial Intelligence, Menlo Park, CA, 2004, American Association of Artificial Intelligence, AAAI, pp. 385–390.

[21] E. Clark, A. S. Ross, C. Tan, Y. Ji, and N. A. Smith, *Creative writing with a machine in the loop: Case studies on slogans and stories*, in 23rd International Conference on Intelligent User Interfaces, 2018, pp. 329–340.

[22] P. R. Cohen and H. Levesque, *Persistence, intention, and commitment*, in Intentions in Communication, P. R. Cohen, J. Morgan, and M. E. Pollack, eds., MIT Press, Cambridge, MA, 1990.

[23] A. Coman and H. Munoz-Avila, *Creating diverse storylines by reusing plan-based stories*, in Working Notes of the ICCBR-12 Workshop on TRUE and Story Cases: Traces for Reusing Users' Experiences - Cases, Episodes, and Stories, 2012.

[24] P. Delatorre, C. León, P. Gervás, and M. Palomo-Duarte, *A computational model of the cognitive impact of decorative elements on the perception of suspense*, Connection Science, 29 (2017), pp. 295–331.

[25] J. Dias and A. Paiva, *Feeling and reasoning: A computational model for emotional characters*, in Portuguese Conference on Artificial Intelligence, Springer, 2005, pp. 127–140.

[26] M. Eger and C. Martens, *Character beliefs in story generation*, in Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, vol. 13, 2017, pp. 184–190.

[27] M. Eger, C. Martens, and M. A. Córdoba, *An intentional ai for hanabi*, in 2017 IEEE Conference on Computational Intelligence and Games (CIG), IEEE, 2017, pp. 68–75.

[28] Facebook, *React*, 2013.

[29] R. Farrell, S. Robertson, and S. G. Ware, *Asking hypothetical questions about stories using quest*, in International Conference on Interactive Digital Storytelling, Springer, 2016, pp. 136–146.

[30] M. W. Fendt and R. M. Young, *Leveraging intention revision in narrative planning to create suspenseful stories*, IEEE Transactions on Computational Intelligence and AI in Games, (2016), pp. 381–392.

[31] R. E. Fikes and N. J. Nilsson, *STRIPS: A new approach to the application of theorem proving to problem solving*, Artificial Intelligence, 2 (1971), pp. 189–208.

[32] M. Fox and D. Long, *Pddl2. 1: An extension to pddl for expressing temporal planning domains*, Journal of Artificial Intelligence Research, 20 (2003), pp. 61–124.

[33] E. Galceran and M. Carreras, *A survey on coverage path planning for robotics*, Robotics and Autonomous systems, 61 (2013), pp. 1258–1276.

[34] C. Geib and B. Webber, *A consequence of incorporating intentions in means-end planning*, in Working Notes–AAAI Spring Symposium Series: Foundations of Automatic Planning: The Classical Approach and Beyond, 1993.

[35] C. W. Geib, *The intentional planning system: ItPlanS*, in Proceedings of the International Conference on AI Planning Systems, 1994, pp. 55–60.

[36] A. Gerevini and D. Long, *Preferences and soft constraints in pddl3*, in ICAPS Workshop on Planning with Preferences and Soft Constraints, 2006, pp. 46–53.

[37] R. J. Gerrig and A. B. Bernardo, *Readers as problem-solvers in the experience of suspense*, Poetics, 22 (1994), pp. 459–472.

[38] M. Ghallab, A. Howe, D. Christianson, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins, *Pddl—the planning domain definition language*, AIPS98 Planning Committee, 78 (1998), pp. 1–27.

[39] S. Goldfarb-Tarrant, H. Feng, and N. Peng, *Plan, write, and revise: An interactive system for open-domain story generation*, in Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations), 2019, pp. 89–97.

[40] A. Graesser, K. Lang, and R. Roberts, *Question answering in the context of stories*, Journal of Experimental Psychology: General, 120 (1991), pp. 254–277.

[41] A. C. Graesser and L. F. Clark, *Structures and Procedures of Implicit Knowledge*, vol. 17, Ablex Publishing Corporation, 1985.

[42] A. C. Graesser, S. E. Gordon, and L. E. Brainerd, *Quest: A model of question answering*, Computers & Mathematics with Applications, 23 (1992), pp. 733–745.

[43] B. Grosz and C. Sidner, *Plans for discourse*, in Intentions in Communication, P. Cohen, J. Morgan, and M. Pollack, eds., MIT Press, Cambdrige, MA, 1990, pp. 417 – 444.

[44] J. A. Hendler, A. Tate, and M. Drummond, *Ai planning: Systems and techniques*, AI Magazine, 11 (1990), pp. 61–61.

[45] J. Hoffmann and B. Nebel, *The ff planning system: Fast plan generation through heuristic search*, Journal of Artificial Intelligence Research, 14 (2001), pp. 253–302.

[46] K. T. Howard and R. Donley, *Using ink and interactive fiction to teach interactive design*, in Interactive Storytelling: 12th International Conference on Interactive Digital Storytelling, ICIDS 2019, Little Cottonwood Canyon, UT, USA, November 19–22, 2019, Proceedings 12, Springer, 2019, pp. 68–72.

[47] C.-W. Hsu, B. W. Wah, R. Huang, and Y. Chen, *Constraint partitioning for solving planning problems with trajectory constraints and goal preferences.*, in IJCAI, 2007, pp. 1924–1929.

[48] J. Ingold, *Designing for narrative momentum*, in Procedural Storytelling in Game Design, AK Peters/CRC Press, 2019, pp. 75–89.

[49] B. Ivars-Nicolas and F. J. Martinez-Cano, *Interactivity in fiction series as part of its transmedia universe: The case of black mirror: Bandersnatch*, Narrative Transmedia, (2019).

[50] J. Jacobs, *English Fairy Tales*, David Nutt, 1890.

[51] A. Jhala and R. M. Young, *Cinematic visual discourse: Representation, generation, and evaluation*, IEEE Transactions on Computational Intelligence and Artificial Intelligence in Games, 2 (2010), pp. 69–81.

[52] S. Jiménez, T. De La Rosa, S. Fernández, F. Fernández, and D. Borrajo, *A review of machine learning for automated planning*, The Knowledge Engineering Review, 27 (2012), pp. 433–467.

[53] B. Kaye and B. Jacobson, *True tales and tall tales: The power of organizational storytelling*, Training & Development, 53 (1999), pp. 44–51.

[54] C. Kelleher, R. Pausch, and S. Kiesler, *Storytelling alice motivates middle school girls to learn computer programming*, in Proceedings of the SIGCHI conference on Human Factors in Computing Systems, 2007, pp. 1455–1464.

[55] Kershner, I, *The Empire Strikes Back*, 20th Century Fox, Brackett, L. and Kasdan, L., 1980.

[56] W. K. Kirchner, *Corporate legends and lore: The power of storytelling as a management tool*, Personnel Psychology, 47 (1994), p. 404.

[57] M. Kreminski, M. Dickinson, J. Osborn, A. Summerville, M. Mateas, and N. Wardrip-Fruin, *Germinate: A mixed-initiative casual creator for rhetorical games*, in Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, vol. 16, 2020, pp. 102–108.

[58] M. Kreminski, M. Dickinson, N. Wardrip-Fruin, and M. Mateas, *Loose ends: A mixed-initiative creative interface for playful storytelling*, in Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, vol. 18, 2022, pp. 120–128.

[59] B. Kybartas and R. Bidarra, *A survey on story generation techniques for authoring computational narratives*, IEEE Transactions on Computational Intelligence and AI in Games, 9 (2016), pp. 239–253.

[60] M. Lebowitz, *Story-telling as planning and learning*, Poetics, 14 (1985), pp. 483–502.

[61] A. Lenhart, J. Kahne, E. Middaugh, A. R. Macgill, C. Evans, and J. Vitak, *Teens, video games, and civics: Teens' gaming experiences are diverse and include significant social interaction and civic engagement.*, Pew Internet & American Life Project, (2008).

[62] M. Mateas and A. Stern, *Façade: An experiment in building a fully-realized interactive drama*, in Game Developers Conference, vol. 2, 2003, pp. 4–8.

[63] D. McAllister and D. Rosenblitt, *Systematic nonlinear planning*, in Proceedings of the National Conference on Artificial Intelligence, 1991, pp. 634–639.

[64] J. MEEHAN, *Tale-spin, an interactive program that writes stories*, in Proceedings of the 5th International Joint Conference on Articial Intelligence, Aug. 1977, vol. 1, 1977, pp. 91–98.

[65] B. Nebel and J. Hoffmann, *The ff planning system: Fast plan generation through heuristic search*, Journal of Artificial Intelligence Research, 14 (2001), pp. 253–302.

[66] B. O'Neill and M. Riedl, *Dramatis: A computational model of suspense*, in Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, AAAI'14, AAAI Press, 2014, pp. 944–950.

[67] J. S. Penberthy and D. Weld, *UCPOP: A sound, complete partial order planner for ADL*, in Proceedings of the Third International Conference on Knowledge Representation and Reasoning, Cambridge, MA, 1991.

[68] M. Pollack, *A model of plan inference that distinguishes between actors and observers*, in Proceedings of the 24th annual meeting of the ACL, 1986.

[69] J. Porteous and M. Cavazza, *Controlling narrative generation with planning trajectories: The role of constraint*, in Proceedings of Second International Conference on Interactive Digital Storytelling (ICIDS), 2009.

[70] J. Porteous, M. Cavazza, and F. Charles, *Applying planning to interactive storytelling: Narrative control using state constraints*, ACM Transactions on Intelligent Systems and Technology (TIST), 1 (2010), pp. 1–21.

[71] J. Porteous and A. Lindsay, *Protagonist vs antagonist provant: Narrative generation as counter planning*, in Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, 2019, pp. 1069–1077.

[72] N. Prescott and L. Scheimer, *The Tom and Jerry Comedy Show*, CBS, 1980.

[73] A. S. Rao and M. P. Georgeff, *An abstract architecture for rational agents.*, KR, 92 (1992), pp. 439–449.

[74] S. Register, *Looney Tunes*, Cartoon Network, 2011.

[75] M. Riedl, *Narrative generation: Balancing plot and character (PhD thesis)*, North Carolina State University, Raleigh, NC, (2004).

[76] M. Riedl and R. M. Young, *Narrative planning: Balancing plot and character*, Journal of Artificial Intelligence Research, 39 (2010), pp. 217–268.

[77] M. O. Riedl, *Incorporating authorial intent into generative narrative systems.*, in AAAI Spring Symposium: Intelligent Narrative Technologies II, 2009, pp. 91–94.

[78] M. O. Riedl and V. Bulitko, *Interactive narrative: An intelligent systems approach*, Ai Magazine, 34 (2013), pp. 67–67.

[79] M. Roemmele and A. S. Gordon, *Creative help: A story writing assistant*, in Interactive Storytelling: 8th International Conference on Interactive Digital Storytelling, ICIDS 2015, Copenhagen, Denmark, November 30-December 4, 2015, Proceedings 8, Springer, 2015, pp. 81–92.

[80] Russo, A. and Russo, J., *Avengers: Endgame*, Marvel Studios, Marcus, C. and McFeely, S., 2019.

[81] A. Salter, *Building interactive stories*, The Routledge Companion to Media Studies and Digital Humanities, (2018), pp. 462–471.

[82] R. Sanghrajka, D. Hidalgo, P. Chen, and M. Kapadia, *Lisa: Lexically intelligent story assistant*, in Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, vol. 13, 2017, pp. 221–227.

[83] R. Sanghrajka, E. Lang, and R. M. Young, *Generating quest representations for narrative plans consisting of failed actions*, in The 16th International Conference on the Foundations of Digital Games (FDG) 2021, 2021, pp. 1–10.

[84] R. Sanghrajka, W. Witoń, S. Schriber, M. Gross, and M. Kapadia, *Computer-assisted authoring for natural language story scripts*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, 2018.

[85] R. Sanghrajka and R. M. Young, *Evaluating reader comprehension of plan-based stories containing failed actions*, in Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, vol. 18, 2022, pp. 179–188.

[86] R. Sanghrajka, R. M. Young, and B. Thorne, *Headspace: Incorporating action failure and character beliefs into narrative planning*, in Proceedings of the AAAI Conference on AI and Interactive Entertainment, Pomona, CA, 2022.

[87] A. Shirvani and S. Ware, *A formalization of emotional planning for strong-story systems*, in Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, vol. 16, 2020, pp. 116–122.

[88] A. Shirvani, S. G. Ware, and L. J. Baker, *Personality and emotion in strong-story narrative planning*, IEEE Transactions on Games, (2022).

[89] A. Shirvani, S. G. Ware, and R. Farrell, *A possible worlds model of belief for state-space narrative planning*, in Proceedings of the 13th AAAI International Conference on Artificial Intelligence and Interactive Digital Entertainment, 2017, pp. 101–107.

[90] T. Silver, V. Hariprasad, R. S. Shuttleworth, N. Kumar, T. Lozano-Pérez, and L. P. Kaelbling, *Pddl planning with pretrained large language models*, in NeurIPS 2022 Foundation Models for Decision Making Workshop, 2022.

[91] N. Simon and C. Muise, *Tattletale: Storytelling with planning and large language models*, in ICAPS Workshop on Scheduling and Planning Applications, 2022.

[92] J. Skorupski, L. Jayapalan, S. Marquez, and M. Mateas, *Wide ruled: A friendly interface to author-goal based story generation*, in Virtual Storytelling. Using Virtual Reality Technologies for Storytelling: 4th International Conference, ICVS 2007, Saint-Malo, France, December 5-7, 2007. Proceedings 4, Springer, 2007, pp. 26–37.

[93] G. Smith and J. Whitehead, *Analyzing the expressive range of a level generator*, in Proceedings of the 2010 Workshop on Procedural Content Generation in Games, 2010, pp. 1–7.

[94] G. Song and N. M. Amato, *Using motion planning to study protein folding pathways*, in Proceedings of the Fifth Annual International Conference on Computational Biology, 2001, pp. 287–296.

[95] E. R. Stein and N. L. Albro, *Building complexity and coherence: Children's use of goal-structured knowledge in telling stories*, in Narrative Development, Routledge, 2012, pp. 5–44.

[96] H. Ten Brinke, J. Linssen, and M. Theune, *Hide and sneak: Story generation with characters that perceive and assume*, in Tenth Artificial Intelligence and Interactive Digital Entertainment Conference, 2014.

[97] J. Teutenberg and J. Porteous, *Efficient intent-based narrative generation using multiple planning agents*, in Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems, International Foundation for Autonomous Agents and Multiagent Systems, 2013, pp. 603–610.

[98] ——, *Incorporating global and local knowledge in intentional narrative planning*, Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, (2015), pp. 1539–1546.

[99] J. Thomas and R. M. Young, *Annie: Automated generation of adaptive learner guidance for fun serious games*, IEEE Transactions on Learning Technologies, 3 (2010), pp. 329–343.

[100] B. Thorne and R. M. Young, *Generating stories that include failed actions by modeling false character beliefs*, in Working Notes of the AIIDE Workshop on Intelligent Narrative Technologies, Salt Lake City, UT, 2017.

[101] T. Trabasso and J. Wiley, *Goal plans of action and inferences during comprehension of narratives*, Discourse Processes, 39 (2005), pp. 129–164.

[102] C. Waldrop, *Does Baldur's Gate 3 Really Have 17,000 Endings? Explained*, https://twinfinite.net/guides/does-baldurs-gate-3-really-have-17000-endings-explained-bg3/, 2023.

[103] D. Walgrave, *Baldur's Gate 3*, Larian Studios, 2023.

[104] S. G. Ware, *The intentional fast-forward narrative planner*, in Eighth Artificial Intelligence and Interactive Digital Entertainment Conference, 2012.

[105] S. G. Ware and C. Siler, *Sabre: A narrative planner supporting intention and deep theory of mind*, in Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, vol. 17, 2021, pp. 99–106.

[106] S. G. Ware and R. M. Young, *CPOCL: A narrative planner supporting conflict.*, in Proceedings of the Conference on AI and Interactive Digital Entertainment, 2011.

[107] S. G. Ware and R. M. Young, *Glaive: a state-space narrative planner supporting intentionality and conflict*, in Tenth Artificial Intelligence and Interactive Digital Entertainment Conference, 2014.

[108] S. G. Ware, R. M. Young, B. Harrison, and D. L. Roberts, *A computational model of narrative conflict at the fabula level*, IEEE Transactions on Computational Intelligence and Artificial Intelligence in Games, 6 (2014), pp. 271–288.

[109] D. Weld, *An introduction to least committment planning*, AI Magazine, 15 (1994), pp. 27–61.

[110] R. P. y Pérez and M. Sharples, *Three computer-based models of storytelling: Brutus, minstrel and mexica*, Knowledge-Based Systems, 17 (2004), pp. 15–29.

[111] P. S. Yoder-Wise and K. Kowalski, *The power of storytelling*, Nursing Outlook, 51 (2003), pp. 37–42.

[112] R. M. Young, M. E. Pollack, and J. D. Moore, *Decomposition and causality in partial order planning*, in Proceedings of the Second International Conference on AI and Planning Systems, 1994.

[113] R. M. Young, S. G. Ware, B. A. Cassell, and J. Robertson, *Plans and planning in narrative generation: a review of plan-based approaches to the generation of story, discourse and interactivity in narratives*, Sprache und Datenverarbeitung, Special Issue on Formal and Computational Models of Narrative, 37 (2013), pp. 41–64.

[114] Z. Zhao, S. Song, B. Duah, J. Macbeth, S. Carter, M. P. Van, N. S. Bravo, M. Klenk, K. Sick, and A. L. Filipowicz, *More human than human: Llm-generated narratives outperform human-llm interleaved narratives*, in Proceedings of the 15th Conference on Creativity and Cognition, 2023, pp. 368–370.

[115] R. Zwaan and G. Radvansky, *Situation models in language comprehension and memory.*, Psychological Bulletin, 123 (1998), p. 162.

[116] R. A. Zwaan, M. C. Langston, and A. C. Graesser, *The construction of situation models in narrative comprehension: An event-indexing model*, Psychological Science, 6 (1995), pp. 292–297.