Using Domain Compilation to add Belief to Narrative Planners

Matthew Christensen
University of Utah

UUCS-20-006

School of Computing
University of Utah
Salt Lake City, UT 84112 USA

1 May 2020

Abstract

Using domain compilation, we present a narrative planning system that is capable of creating narrative plans that use both character intention and character beliefs. We introduce a model capable of representing character beliefs in PDDL domains. This model allows characers to fail at actions when their beliefs about the world differ from the actual world state. Domains of this type can be compiled into purely intentional domains, and fed as input to intentional planners. The resulting stories feature characters that pursue their own intentions based on their own knowledge of the world, learn from mistakes to update their beliefs, and communicate information to each other. These types of stories are not possible with purely intentional domains.

Using Domain Compilation to add Belief to Narrative Planners

by
Matthew Christensen

A Senior Honors Thesis Submitted to the Faculty of
The University of Utah
In Partial Fulfillment of the Requirements for the

Honors Degree in Bachelor of Science

In

The School of Computing

Approved:

Rogelio E. Cardona Rivera Thesis Faculty Supervisor	Ross Whitaker Director, School of Computing
Erin Parker Honors Faculty Advisor	Sylvia D. Torti Dean, Honors College

April 2020

ABSTRACT

Using domain compilation, we present a narrative planning system that is capable of creating narrative plans that use both character intention and character beliefs. We introduce a model capable of representing character beliefs in PDDL domains. This model allows characers to fail at actions when their beliefs about the world differ from the actual world state. Domains of this type can be compiled into purely intentional domains, and fed as input to intentional planners. The resulting stories feature characters that pursue their own intentions based on their own knowledge of the world, learn from mistakes to update their beliefs, and communicate information to each other. These types of stories are not possible with purely intentional domains. [1]

[1] Our implementation can be found at `https://github.com/qed-lab/belief-intention-compilation`

TABLE OF CONTENTS

1 Introduction

Narrative planning is the problem of automatically creating compelling narrative structures using planning systems. It is primarily concerned with creating the plot structure (or *fabula*) of stories, rather than the prose used to tell them. One of the challenges of narrative planning is developing tools to allow planning systems to reason about conflict[13] and character[8] when crafting stories. Modern narrative planners such as Glaive can use models of intention that enable characters to pursue, achieve, and change their goals[14]. However, there does not yet exist a narrative planning system that allows for characters to have their own knowledge states about the world[10].

In stories, characters often have differing beliefs about what is true in the world, and those beliefs play a role in determining what actions those characters take. A villain, for instance, could not steal a valuable artifact unless they knew where that artifact was hidden. Likewise, a hero would not attempt to open a door if they already believed it was locked. Not only do these beliefs affect what choices a character makes, but they also can cause characters to take actions that fail. Perhaps the hero initially believed the door was unlocked, but upon trying to open it, it would not budge. These failures occur when a character's beliefs are different from the actual state of the world.

We present a model that is capable of representing character beliefs in PDDL[5] domains. We will then demonstrate how this model can be compiled to a domain which only uses intentionality. This compilation can be used in conjunction with an intentional planner to produce narrative plans which reason about both character beliefs and intentions. This system expands the space of stories that can be generated at a cost of greater domain size during the planning phase.

2 Background

2.1 Planning Background

This paper relies on the formulation of **classical planning**. Classical planning is a model of problem solving, wherein agent actions are fully observable and deterministic. Classical problems are typically represented in the STRIPS formalism[2]. A STRIPS planning problem is a tuple $P{=}\langle F, I, A, G\rangle$ where F is the set of fluents, $I{\subseteq}F$ is the initial state, $G{\subseteq}F$ is the set of goal conditions, and A is a set of actions. Each action is a triple $a{=}\langle \text{PRE}(a), \text{ADD}(a), \text{DEL}(a)\rangle$ that represents the precondition, add, and delete lists respectively, all of which are subsets of F. A state is a conjunction of fluents. An action a is applicable in a state s if $\text{PRE}(a){\subseteq}s$; applying said applicable action in the state results in a new state $s'{=}(s{\setminus}\text{DEL}(a)){\cup}\text{ADD}(a)$.

The solution to a planning problem P is a plan $\pi{=}[a_1, ..., a_m]$, a sequence of actions $a_i{\in}A$ that transforms the problem's initial state I to a state s_m that satisfies the goal; *i.e.,* $G{\subseteq}s_m$.

Planning is an attractive avenue to pursue storytelling. Most stories follow a logical chain of events that carry the plot from an initial state to an end state. Since a plan is simply a logical ordering of actions, a planner can be used to generate that chain of events, referred to as the *fabula*. This is distinct from the *discourse*, which interprets the *fabula* to tell the story through some medium (text, images, sounds, etc.).

The compilation process we describe is designed to read and output planning problems written using PDDL[5], one of the most common languages used to write planning domains and problems.

2.2 Intentional Planning

Classical planners are built to solve problems with efficient plans. This is a desirable property, but often the most efficient way to solve a problem is the least interesting from a narrative perspective. Narrative planners combat this by introducing preferences and restrictions on the types of plans that can be generated.

Our system builds on the methods established by intentional planners. In an intentional domain, an action $a \in A$ can have one or more consenting agents (i.e., *actors*). Intentional planners add the following restriction: action a is only applicable if all of it's preconditions are met (as described above) **and** a is a step towards achieving a goal of each of its consenting agents[8, 12, 14].

An agent's goals are represented using the modal *intends* literal. For instance, a villain may intend to have the artifact, and that intention is represented in PDDL as `(intends villain (has villain artifact))`.

It is important to make a distinction between a character's goals, and goals in a classical planning sense. A character's goals are not necessarily the same as the goal state of a planning problem. A problem's goal state can be thought of as the author's goals for the end of the story. Intentional planners will search for a sequence of events that results in the author's goals, all the while ensuring that every action taken by a character is a step toward that character's goals.

We target the intentional planner Glaive[14] as the final step of our belief-intention planning system, but the output of our compilation is compatible with any intentional planner.

3 Previous Work

The compilation we describe is based in part on a compilation by Haslum[3] designed to compile intentional PDDL into a form that could be processed by a classical planner. Haslum's compilation was originally designed to operate on domains built for the IPOCL[8] planning system, and offered significant speedup thanks to fast, generalized heuristics for classical planners[4]. Since that time, intentional planners have incorporated those heuristics, and added their own[14]. Initially, we had planned on combining our compilation process with Haslum's to skip intentional planning altogether, but concerns about domain size made it a less appealing option.

Haslum's compilation drew influence from Palacios and Geffner's work on compiling conformant planning problems[6]. Conformant planning problems present the world state with a degree of uncertainty; meaning a planner may have an imperfect representation of the state of the world when making decisions. An important distinction to make between their work and the compilation we present here is that in Palacios and Geffner's work, it was the planner itself with imperfect knowledge. In ours, it is the characters that may be mistaken about the world. The planner will always have perfect knowledge of the world state.

One of the motivations for this work is to make a step towards the BDI (Beliefs, Desires, and Intentions) agent model[7] for narrative planners. Thus far, narrative planners have focused on intention by giving characters the ability to make and pursue goals. This work addresses belief by representing each character's belief about the world and what actions are available to them. Modelling desire in narrative plans is still relatively unexplored, though some models have been proposed[11].

Recent work by Shirvani et al. demonstrated the importance of reasoning about belief in addition to intention when it comes to generating believable stories[10, 9]. Their paper

proposed another model for character beliefs, and found that plans that followed this model were generally perceived to be more believable by readers. The model we present here draws from the one put forward by Shirvani et al.[9], but has significant differences such as prohibiting beliefs about beliefs, and adding the potential for failed actions.

4 Model of Belief

The term *belief* as it relates to narrative and planning has a specific meaning. In the BDI model, a character's belief state is their current knowledge of the world state[7]. As such, it is sensible that a character shouldn't act against their best knowledge. To a reader, it is jarring when a character in any story behaves in a way that contradicts what they know.

In general, this knowledge is not limited to simple facts of the world: a character can have beliefs about another character's intentions or beliefs. This can quickly lead to very complex situations, as can be seen in the work by Shirvani et al., where characters can have beliefs about others' beliefs to arbitrary depth[9]. For the sake of clarity and feasibility, we will restrict our model to only allow for beliefs of simple statements about the world (non-modal literals). While this places a limitation on the complexity of narratives that we can generate with this system, we will demonstrate that this model is expressive enough to capture many stories that could not be created using a purely intentional planning system.

Of particular interest to us are the situations in which a character attempts to take an action which they believe is possible, but in reality is impossible. Such a situation can only arise when an agent has at least one belief about the world state that isn't true. In classical and intentional planning, if an action is impossible (i.e., when its preconditions aren't met), it is deemed non-applicable, and thus cannot be taken. However, it should be possible for a character to attempt an action they cannot complete. Such an attempt results

in failure. Failing any action has consequences that can affect both the state of the world and the beliefs of the character.

In a compelling story, it is often necessary for a character to fail in order for them to learn something important about the world. Or perhaps the consequences of their failure will later open up new opportunities for them to achieve their (or the author's) goals (see the *hubris* domain below in Figure 5).

In general, characters do not act unless they believe their action will succeed. Our model of belief encodes this by requiring characters to believe all preconditions of an action before they attempt to take it.

Lastly, any model of character knowledge has to deal with changes to a character's beliefs. There are any number of microtheories that could be applied to update character beliefs when an action succeeds or fails. One such microtheory may be that a character knows all consequences of the actions that they perform or fail. Another microtheory may be that a character knows about the effects of all actions that occur in the same location they are in. As a matter of flexibility, we choose not to commit to any microtheory of knowledge, and instead leave it to domain authors to explicitly list out what changes need to happen to character beliefs for each action individually.

5 Belief Domain Formulation

In order to satisfy our model, we must create a mathematical definition that meets the following requirements:

- A character can have beliefs about the world that differ from the actual world state.
- A character may only have beliefs about simple statements about the world.

- An action can only be taken if all of its consenting agents **believe** that all of its preconditions are met.
- An action with consenting agents can be taken both if its preconditions are met, and if its preconditions aren't met.
- If an action is taken when its preconditions are not met, the action is said to fail, and the consequences of that failure will be applied to the world.

We present our model first in a mathematical STRIPS-like format, then more practically in PDDL.

5.1 Formal Definition

Definition 1. *A belief-intention planning problem is a tuple* $P{=}\langle F, S, A, G, C\rangle$*, where* F *is the set of fluents,* S *is the initial state,* $G{\subseteq}F$ *is the set of goal conditions,* C *is the set of characters, and* A *is the set of actions.*

Definition 2. *A state in a belief-intention planning problem is a tuple* $s{=}\langle W, B, I\rangle$ *where* $W{\subseteq}F$ *represents the state of the world,* B *is a set of belief states* $b_c{\subseteq}F$ *for each character* $c \in C$*, and I is a set of intentions* $i_c{\subseteq}F$ *for each character* $c \in C$*.*

Each belief state b_c represents the set of all fluents that character c believes in the current state. An action $a{\in}A$ is now represented as a 5-tuple:

$$a = \langle \text{PRE}(a), \text{ADD}_s(a), \text{DEL}_s(a), \text{ADD}_f(a), \text{DEL}_f(a)\rangle$$

Here, $\text{PRE}(a)$ is a set of fluents like in classical planning. The add and delete lists of a are triples $\left\langle W_{\text{ADD/DEL}}, B_{\text{ADD/DEL}}, I_{\text{ADD/DEL}}\right\rangle$, that represent changes to be made to the world

state, character belief states, and character intentions respectively. In order to apply add and delete lists to state s to obtain s', apply the respective lists as in classical planning (e.g. $W'=(W \setminus W_{\text{DEL}}) \cup W_{\text{ADD}}$)

Action a is applicable in a state s in two cases:

- If a has no consenting characters, then $\text{PRE}(a) \subseteq W$
- If a has at least one consenting character, then for each consenting character c, $\text{PRE}(a) \subseteq b_c$

Actions that meet the first condition are actions taken only by the planning agent, and not by any of the characters. It is common to refer to such actions as acts of fate in the story. Such actions can only be taken if the world state meets their preconditions. The second condition does not require the preconditions of a to be met in the state of the world. This means that if a character believes the preconditions for an action are met, that action is applicable whether or not it is "possible."

The effect of applying a to some state s is as follows:

- If a has no consenting characters, then $\text{ADD}_s(a)$ and $\text{DEL}_s(a)$ are both applied to state s to obtain s'.
- If a has consenting characters, and $\text{PRE}(a) \subseteq W$, then $\text{ADD}_s(a)$ and $\text{DEL}_s(a)$ are applied to s.
- If a has consenting characters, but $\text{PRE}(a) \not\subseteq W$, then $\text{ADD}_f(a)$ and $\text{DEL}_f(a)$ are applied to s.

The third case represents an action failing. All consenting characters believed that the preconditions were met, but those beliefs were incorrect. $\text{ADD}_f(a)$ and $\text{DEL}_f(a)$ represent

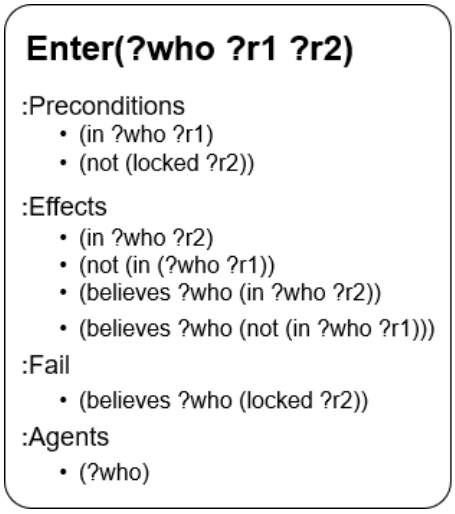

Figure 1: A simplified sample action from a belief domain

the consequences that get applied to the world and belief states because of that failure.

5.2 PDDL Encoding

To represent belief domains and problems in PDDL, we introduce the requirement *belief-preconditions*. This requirement allows us to use the modal `(believes c l)` predicate, where c is a character and l is some non-modal literal. The belief state for a character c is represented by all *believes* predicates in the current state with c as the first parameter.

Figure 1 demonstrates an example of an action in the *rooms* belief domain. This action represents a character attempting to leave one room and enter another. In addition to the normal fields, there is a `:fail` heading. This contains the effects that should take place when the action fails (in this case, the character now believes the room is locked).

Besides failure, actions are written in much the same way as they would be in an intentional or classical domain. However, each action with consenting agents will only be applicable if each of the consenting agents *believes* the preconditions are met (i.e., there should be a literal `(believes c p)` for each consenting character c and each precon-

dition p). In the example above, the consenting agent would have to believe both that they are in room *r1* and that room *r2* is not locked.

5.3 Negative Preconditions, Belief, and the Closed World Assumption

In our implementation, we allowed for actions to have negative preconditions. Under the closed world assumption of classical planning[2], preconditions of the form (*not p*) for some predicate p are considered met if p is not in the current state. However, in belief planning, such a precondition would only be met if there is a literal of the form `(believes c (not p))` for each consenting character c. This statement is different from `(not (believes c p))`. The former explicitly means that c believes that p is false. The latter implies that c **does not have a belief** that p is true. In other words, c could be ambivalent as to whether or not p is true.

Thus, while the closed world assumption still applies for the planner itself, the same is not true for the characters in the plans it creates. For instance, in the *rooms* domain (Figure 3), the main character does not have any belief about the key. We believe this more accurately reflects how characters learn and behave in stories.

6 Belief-Intention Compilation

6.1 Overview

In order to create a plan for a belief-intention domain, we first compile it into an intentional domain. From there, the compiled domain is supplied as an input to an intentional planner (See Figure 2). This paper uses Glaive[14], although any system that can create plans from

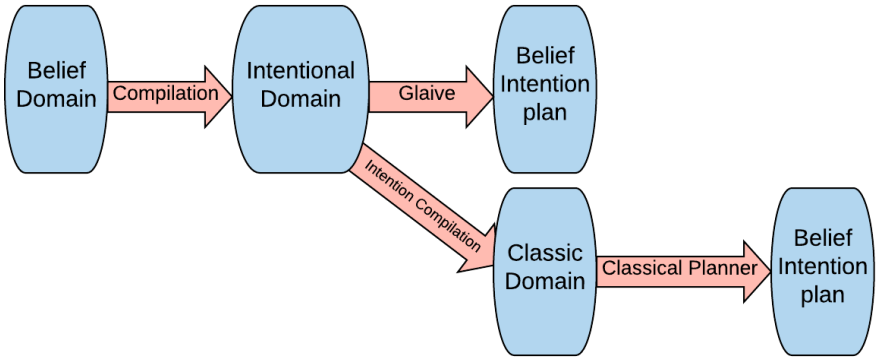

Figure 2: An overview of the belief planning process.

intentional domains could be used (e.g. Haslum's intention compilation[3] followed by a classical planner).

6.2 Compilation

The PDDL representation of a belief domain can be represented by its predicates and operators, $Dom_b{=}\langle P, O\rangle$. A problem is represented by its initial state, and goal state, $Prob_b{=}\langle I, G\rangle$. These are the only fields that we alter to obtain an intentional domain $Dom_i{=}\langle P', O'\rangle$ and problem $Prob_i{=}\langle I', G'\rangle$.

Other PDDL fields, such as `types:` and `objects:` are not changed. The `requirements:` field in the Domain is updated to remove *belief-preconditions*, and add *negative-preconditions, disjunctive-preconditions* and *intentionality*. The domain/problem names are also changed to avoid conflict with the original domain and problem files.

6.2.1 State Compilation

The initial state of the problem is comprised of grounded literals. For each literal $l{\in}I$:

- If l is of the form `(believes c (m x1 x2 ...))` for some character c and

some non-modal predicate m, we create a new flattened literal `(believes_m c x1 x2 ...)` and add it to I'.

- If l is of the form `(believes c (not (m x1 x2 ...)))` for some character c and some non-modal predicate m, we create a new flattened literal `(believes_not_m c x1 x2 ...)` and add it to I'.
- Otherwise, we add l to I'.

The first two conditions have the effect that all *believes* literals are flattened into non-modal literals, with the believing character appended as the first parameter. For example, `(believes villain (not(at artifact cave)))` would be compiled to `(believes_not_at villain artifact cave)`. In essence, each character belief is treated as being part of the world state itself.

In PDDL, G is not a set, but a logical sentence. We replace every belief term in G with its flattened version as above to obtain G'.

6.2.2 Predicate Compilation

Every predicate $p \in P$ is a non-modal [2] statement `(p ?x1 - t1 ?x2 - t2 ...)`. For every p we create a new predicate p_b of the form `(believes_p ?c - character ?x1 - t1 ?x2 - t2 ...)` and another predicate $p_{\neg b}$ of the form `(believes_not_p ?c - character ?x1 - t1 ?x2 - t2 ...)`. We add each p, p_b, and $p_{\neg b}$ to P'.

[2] The *believes* and *intends* predicates are implicit in belief and intentional domains respectively.

6.2.3 Action Compilation

In Belief Domains, an Action $a \in O$ is represented as a tuple

$$a = \langle Param, Pre, Eff, Fail, Agents \rangle$$

These are the parameters, preconditions, effects, failure effects, and consenting agents respectively. Preconditions, effects, and failure effects are each logical sentences.

If a has no consenting agents, we create a single new action a'. The parameters, preconditions, effects and agents fields are identical to those found in a. If any of the preconditions or effects of a are *believes* predicates, we flatten them into their non-modal *believes_p* (or *believes_not_p*) forms as in the goal compilation. We add a' to O'.

If a has consenting agents, we create two actions a_s and a_f.

- The parameters and agents of a_s and a_f are the same as those in a.

$$Param_{s/f} = Param$$

$$Agents_{s/f} = Agents$$

- The effects of a_s and a_f are the effects and failure effects of a respectively (where each term is flattened using the goal compilation as before).

$$Eff_s = Eff$$

$$Eff_f = Fail$$

- For each consenting agent $?c \in Agents$, we create a new logical sentence Pre_c^B,

which is a simplified [3] copy of Pre. We replace each term $p \in Pre_c^B$, with a new predicate p_c^B of the form `(believes_p ?c ?x1 ?x2 ...)` (or `(believes_not_p)` if it is negated). If p is already a *believes* predicate, we replace it with its flattened version.

The preconditions for a_s are:

$$Pre_s = Pre \wedge \left[\bigwedge_{c \in Agents} Pre_c^B \right]$$

In other words, the preconditions for a_s are all of the preconditions of a, plus every consenting character must believe those preconditions are met.

$$Pre_f = (\neg Pre) \wedge \left[\bigwedge_{c \in Agents} Pre_c^B \right]$$

The preconditions for a_f are similar: Every consenting character must believe the preconditions for a are met, but the negation of the preconditions for a must be true [4].

After which, both a_s and a_f are added to O'. In our implementation, we differentiate between these actions from the original action by appending the suffixes `_success` and `_fail` to the action names.

An example of a compiled action is given in Appendix A.

[3] To simplify a logical sentence, we use DeMorgan's laws to push negations as far inward as possible. At the end of the simplification, every *not* statement should contain a single term (i.e., a *not* cannot contain an *and* or an *or*). This is so negative preconditions are treated as *believes not* as opposed to *not believes* statements, preserving the open world for character belief states.

[4] During simplification, the negation of any *and* statements become *or* statements of the negative terms. This is why the belief compilation adds the requirement for both disjunctive and negative preconditions.

6.2.4 Expression Parameters

The requirement `:expression-parameters` allows for PDDL actions to be defined with predicates as parameters. Such actions may be useful for adding variable beliefs or intentions to world state. For example, Figure 3 features an action `read`, where one of the parameters is an expression representing the info present in a letter.

To compile an action a with an expression parameter $?exp$, it must first be grounded such that no more expression parameters appear. For each possible and relevant [5] predicate p create a new action a_p, with all of the parameters of a as well as the parameters of p.

$$Params(a_p) = Params(a) \backslash ?exp \cup Params(p)$$

In addition, all occurrences of $?exp$ in the effects and preconditions of a should be replaced with p. A full PDDL example of this process can be found in Appendix A.

Since grounding an expression parameter requires creating a new action for every predicate, this process will quickly increase the size of the search space. Including too many expression parameters in a domain can result in compiled problems that take hours to plan.

6.3 Decompilation

A plan produced using a compiled belief domain consists of compiled actions, which may have the *success* or *fail* suffixes. If an action in the plan ends with one of these suffixes, we trim off the suffix to obtain the name of the action in the original domain. An action without either suffix does not require any further changing, as it already appeared in the domain.

[5] This terminology comes from a similar process in Haslum's compilation[3]. We take *possible and relevant* to mean every predicate that appears as a literal in the initial state or in the effects of an action.

When Glaive outputs an intentional plan, it also reports some actions as being *not-taken*. These actions were part of an intentional frame for some character, but weren't part of the final plan. These actions are also decompiled to obtain what characters *intended* to do, but didn't have a chance during the story.

7 Evaluation

7.1 Domain size

The primary metric we use to analyze the cost of this compilation process is the size of the compiled domain relative to the uncompiled domain. It is difficult to precisely quantify the effect that the size of a domain will have on the running time of the planning process, but a larger search space will generally increase the time it takes to plan[1].

For every predicate in a belief problem, the compilation adds two more predicates representing the *believes* and *believes_not* versions of it. Therefore, the compilation multiplies the number of predicates in the domain by a factor of 3.

Actions in the domain with consenting agents are split into their success and failure versions. This effectively doubles the number of actions in the domain. Discounting the exponential growth caused by actions with many expression variables, this results in linear growth of the domain with respect to the size of the input domain.

7.2 Sample Domains

We created three domains to demonstrate the types of stories that are enabled by our model of belief.

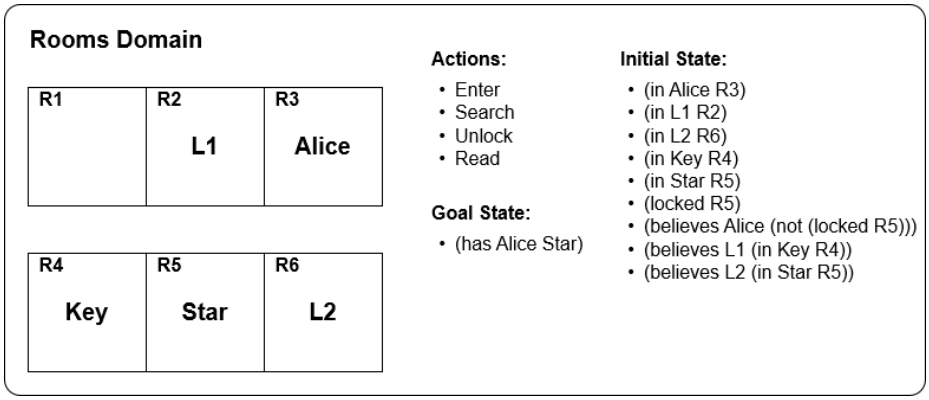

Figure 3: The rooms domain created with our belief model. (See Appendix A for full domain.)

7.2.1 Rooms

The *rooms* domain in Figure 3 is built to demonstrate how belief can facilitate the spread of information. The main character, Alice, wants to obtain the star. However, she doesn't know which room the star is in, nor does she know that the room is locked. There are letters in rooms 2 and 6 that have information about where the key and star are.

In an intentional domain, the optimal plan solution would have Alice first enter the room where the key is, then use it to unlock the door and obtain the star. However, in the belief domain, it doesn't make sense for her to take those actions because she wouldn't even know she needs to search for a key yet. First, she must learn that the door is locked by attempting to open it and failing. Then, by traveling to other rooms, she can find letters that tell her where the key is. (See Figure 4.)

7.2.2 Hubris

The *hubris* domain (Figure 5) was created to demonstrate how failure can add drama to a situation. Here, a villain has obtained the ultimate artifact, and intends to use it to destroy the hero and take over the world. However, they lack the arcane knowledge necessary to

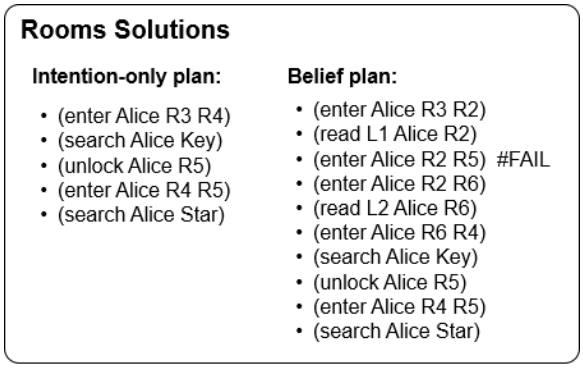

Figure 4: Comparison of solutions to the rooms domain

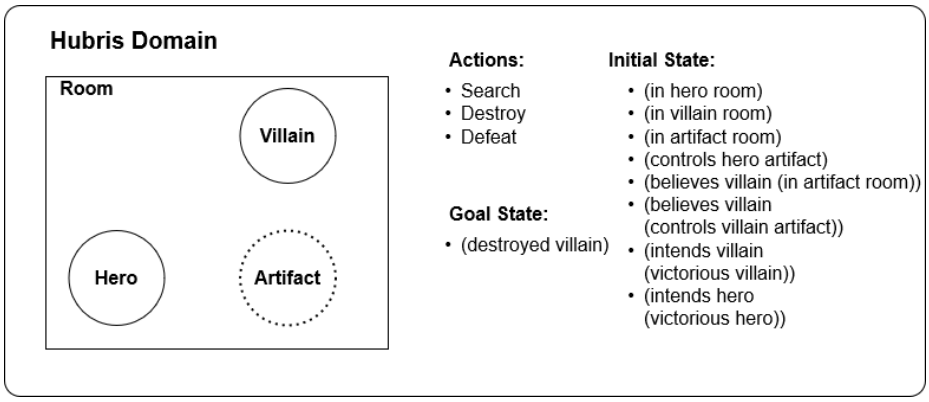

Figure 5: The hubris domain created to demonstrate importance of failure.

wield the artifact. Thus, when they attempt to use it, they fail, alerting the hero in the process. Our hero, who can wield the artifact, takes this opportunity to steal it from the villain and use it against them. In an intentional domain, the villain wouldn't attempt to use the artifact, because they could not do so successfully. It is only the villain's mistaken belief that allows the story to progress to the author's goal.

7.2.3 Journey

We created the *journey* domain (Figure 6) to show how belief can add obstacles to a character's journey. Here, we've given Alice the power to turn away a calamity facing her village. However, she doesn't yet know that she has this power. She must take a journey to a far off land before she learns that she had the power the whole time. Like the *rooms* domain,

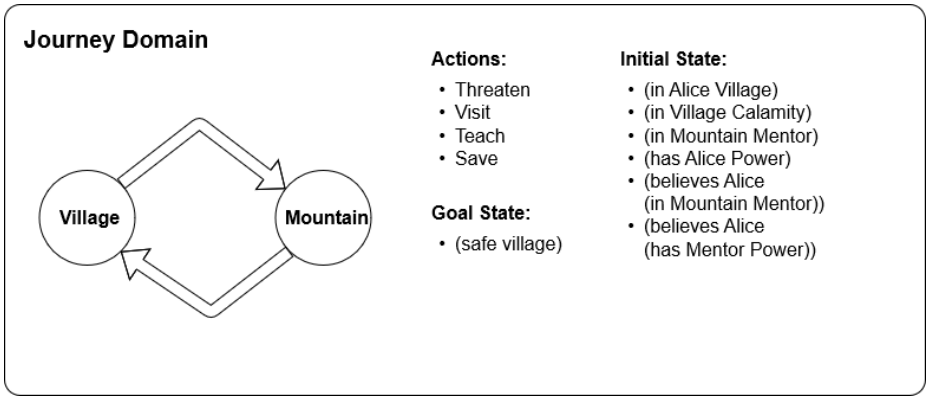

Figure 6: The journey domain created to demonstrate how knowledge can add interest and length to a story.

the optimal intentional plan for the story is much shorter, but allowing characters to have incorrect or incomplete beliefs adds interest to the story.

8 Future Work

The restriction that *believes* literals may not contain other modal literals like *intends* or *believes* places a limit on the complexity of stories that can be achieved with this model. Using an approach similar to the one in this paper, it is possible to compile beliefs of beliefs. However, as the depth of the model grows (beliefs of beliefs of ...), the number of predicates in the compiled domain would grow exponentially, to say nothing of the difficulty of manually authoring such a domain. However, comprehensible stories likely only need to reason about beliefs of depth two or three.

A next potential step is finding a similar compilation for a simplified model of desire. Combining belief and desire compilations with intentional planners could lead to good approximations of BDI agents[7] for narrative planning.

While we chose not to commit to a particular microtheory of how to update beliefs, our model requires all consenting agents to *explicitly* believe all preconditions to perform an

action. However, as was noted earlier, agents are allowed to be ambivalent about facts in the world. Therefore, a different model of belief might restrict characters not to the actions that they believe are possible, but rather, the actions that they don't believe are impossible. It is an open question as to how such a restriction would be represented.

Future work can be done to better understand the interaction of belief and intention. For instance, what does it mean to intend to believe something? How should a character's belief about another character's intents affect the way they behave?

Our model and compilation lay a groundwork to pursue each of these questions. We hope to continue building on this work in these directions.

9 Conclusion

Using our model of belief, it is possible to compile belief domains into intentional domains, with a linear increase to domain size. Intentional planners can then create stories where characters have different knowledge states and act according to their beliefs. These characters can fail, learn from their mistakes, and share information with each other. In general, they act more closely to how a reader would expect them to behave. We exemplify this with three domains, each of which rely on belief. With this, narrative planning has one more tool to create more interesting and believable stories.

References

[1] Tom Bylander. Complexity results for planning. In *IJCAI*, volume 10, pages 274–279, 1991.

[2] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

[3] Patrik Haslum. Narrative planning: Compilations to classical planning. *Journal of Artificial Intelligence Research*, 44:383–395, 2012.

[4] Jörg Hoffmann. Ff: The fast-forward planning system. *AI magazine*, 22(3):57–57, 2001.

[5] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. Pddl-the planning domain definition language, 1998.

[6] Héctor Palacios and Héctor Geffner. Compiling uncertainty away: Solving conformant planning problems using a classical planner (sometimes). In *AAAI*, pages 900–905, 2006.

[7] Anand S Rao, Michael P Georgeff, et al. Bdi agents: from theory to practice. In *ICMAS*, volume 95, pages 312–319, 1995.

[8] Mark O Riedl and Robert Michael Young. Narrative planning: Balancing plot and character. *Journal of Artificial Intelligence Research*, 39:217–268, 2010.

[9] Alireza Shirvani, Stephen G Ware, and Rachelyn Farrell. A possible worlds model of belief for state-space narrative planning. In *Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2017.

[10] Alireza Shirvani, Rachelyn Farrell, and Stephen G Ware. Combining intentionality and belief: Revisiting believable character plans. In *Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2018.

[11] Theo Wadsley and Malcolm Ryan. A belief-desire-intention model for narrative generation. In *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2013.

[12] Stephen G Ware. The intentional fast-forward narrative planner. In *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2012.

[13] Stephen G Ware and R Michael Young. Cpocl: A narrative planner supporting conflict. In *Seventh artificial intelligence and interactive digital entertainment conference*, 2011.

[14] Stephen G Ware and R Michael Young. Glaive: a state-space narrative planner supporting intentionality and conflict. In *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2014.

Appendix A Examples

A.1 Action Compilation Example

An example of the action compilation described in section 6.2.3

Uncompiled Action

```
;; A character moves from one room to another (via a hallway adjacent to
    all rooms)
(:action enter
 :parameters   (?character - character ?roomfrom - room ?roomto - room)
```

```
:precondition
 (and
  (not (locked ?roomto))
  (at ?character ?roomfrom)
 )
:effect
 (and
  (not (at ?character ?roomfrom))
  (at ?character ?roomto)
  (believes ?character (not (at ?character ?roomfrom)))
  (believes ?character (at ?character ?roomto))
  (not (believes ?character (at ?character ?roomfrom)))
  (not (believes ?character (not (at ?character ?roomto))))

  (forall (?char2 - character)
   (when (at ?char2 ?roomto)
    (and
     (believes ?character (at ?char2 ?roomto))
     (believes ?char2 (at ?character ?roomto))
     (not (believes ?character (not (at ?char2 ?roomto))))
     (not (believes ?char2 (not (at ?character ?roomto))))
    )
   )
  )
 )
:fail
 (when (and (locked ?roomto) (at ?character ?roomfrom) )
  (and
   (believes ?character (locked ?roomto))
   (not (believes ?character (not (locked ?roomto))))
  )
 )
:agents
```

```
  (?character)
)
```

Compiled Actions

```
(:action enter_success
 :parameters   ( ?character - character ?roomfrom - room ?roomto - room)
 :precondition
  (and
   (not
    (locked ?roomto)
   )
   (at ?character ?roomfrom)
   (believes_not_locked ?character ?roomto)
   (believes_at ?character ?character ?roomfrom)
  )
 :effect
  (and
   (not
    (at ?character ?roomfrom)
   )
   (at ?character ?roomto)
   (believes_not_at ?character ?character ?roomfrom)
   (believes_at ?character ?character ?roomto)
   (not
    (believes_at ?character ?character ?roomfrom)
   )
   (not
    (believes_not_at ?character ?character ?roomto)
   )
   (forall
    (?char2 - character)
    (when
```

```
     (at ?char2 ?roomto)
     (and
      (believes_at ?character ?char2 ?roomto)
      (believes_at ?char2 ?character ?roomto)
      (not
       (believes_not_at ?character ?char2 ?roomto)
      )
      (not
       (believes_not_at ?char2 ?character ?roomto)
      )
     )
    )
   )
  )
 :agents (?character)
)


(:action enter_fail
 :parameters   ( ?character - character ?roomfrom - room ?roomto - room)
 :precondition
  (and
   (or
    (locked ?roomto)
    (not
     (at ?character ?roomfrom)
    )
   )
   (believes_not_locked ?character ?roomto)
   (believes_at ?character ?character ?roomfrom)
  )
 :effect
  (when
```

```
    (and
     (locked ?roomto)
     (at ?character ?roomfrom)
    )
    (and
     (believes_locked ?character ?roomto)
     (not
      (believes_not_locked ?character ?roomto)
     )
    )
  )
 :agents (?character)
)
```

A.2 Expression Parameter Grounding

The *read* action in the *rooms* domain has an expression parameter called `?info`

```
(:action read
  :parameters (?letter - character ?info - expression ?informed -
    character ?room - room)
  :precondition (and
    ( at ?letter ?room )
    ( at ?informed ?room )
    ( believes ?letter ?info )
  )
  :effect (and
    ( believes ?informed ?info )
    ( not (believes ?informed (not ?info)))
  )
)
```

In order to remove the expression parameter from this action, a new action must be created for every predicate. We will demonstrate how to ground *read* with respect to the predicate *in*:

```
(:action read_in
  :parameters (?letter - character ?informed - character ?room - room ?
   thing - thing ?room - room)
  :precondition (and
    ( at ?letter ?room )
    ( at ?informed ?room )
    ( believes ?letter (in ?thing ?room) )
  )
  :effect (and
    ( believes ?informed (in ?thing ?room) )
    ( not (believes ?informed (not (in ?thing ?room))))
  )
)
```

In this example, the `?info` parameter has been replaced by the *in* predicate in the effects and preconditions. The parameters `?thing` and `?room` from *in* have been added as parameters of *read*. Lastly, the name has been changed to differentiate this new action from the other actions that will be created.

In order to complete the process, a new action *read_p* must be created for every possible and relevant predicate p (thus making the search space much larger).

A.3 Rooms Domain

Domain

```
(define (domain rooms)
```

```
(:requirements :adl :universal-preconditions :expression-variables :
  intentionality :belief)
(:types
 letter key star - thing
 character room thing
)
(:predicates
 (locked ?room - room)
 (at ?character - character ?room - room)
 (unlocked-by ?room - room ?key - key)
 (in ?thing - thing ?room - room)
 ;;(hidden ?thing - thing)
 (has ?character - character ?thing - thing)
)

;; A character moves from one room to another (via a hallway adjacent
  to all rooms)
(:action enter
 :parameters   (?character - character ?roomfrom - room ?roomto - room)
 :precondition
  (and
   (not (locked ?roomto))
   (at ?character ?roomfrom)
  )
 :effect
  (and
   (not (at ?character ?roomfrom))
   (at ?character ?roomto)
   (believes ?character (not (at ?character ?roomfrom)))
   (believes ?character (at ?character ?roomto))
   (not (believes ?character (at ?character ?roomfrom)))
   (not (believes ?character (not (at ?character ?roomto))))

```

```
    (forall (?char2 - character)
     (when (at ?char2 ?roomto)
      (and
       (believes ?character (at ?char2 ?roomto))
       (believes ?char2 (at ?character ?roomto))
       (not (believes ?character (not (at ?char2 ?roomto))))
       (not (believes ?char2 (not (at ?character ?roomto))))
      )
     )
    )
   )
  :fail
   (when (and (locked ?roomto) (at ?character ?roomfrom) )
    (and
     (believes ?character (locked ?roomto))
     (not (believes ?character (not (locked ?roomto))))
    )
   )
  :agents
   (?character)
)

; An letter tells the character something the letter believes.
(:action read-letter
 :parameters  (?letter - character ?info - expression ?informed -
   character ?room - room)
 :precondition
   (and
    (at ?letter ?room)
    (at ?informed ?room)
    (believes ?letter ?info)  ;; For the informant to do this action,
   need they believe that they believe?
   )
```

```
 :effect
  (and j
   (believes ?informed ?info)
   (not (believes ?informed (not ?info))) ;; Future versions may do
  this differently, allowing lies, trust/mistrust
  )

 :fail () ;; Possibly the informed disbelieves the info if the
  informant disbelieves it
 :agents (?informed)
)

;; A character searches the room for a specific thing
(:action search-for
 :parameters  (?character - character ?thing - thing ?room - room)
 :precondition
  (and
   (at ?character ?room)
   (in ?thing ?room)
  )
 :effect
  (and
   (has ?character ?thing)
   (not (in ?thing ?room))

   (believes ?character (has ?character ?thing))
   (believes ?character (not (in ?thing ?room)))
   (not (believes ?character (not (has ?character ?thing))))
   (not (believes ?character (in ?thing ?room)))
  )
 :fail
  (when (and (not (in ?thing ?room)) (at ?character ?room))
   (and
```

```
95      (not (believes ?character (in ?thing ?room)))
96      (believes ?character (not (in ?thing ?room)))
97     )
98    )
99   :agents (?character)
100  )
101
102  ;; A character unlocks a room with a key
103  (:action unlock
104   :parameters  (?character - character ?key - key ?room - room)
105   :precondition
106    (and
107     (locked ?room)
108     (unlocked-by ?room ?key)
109     (has ?character ?key)
110    )
111   :effect
112    (and
113     (not (locked ?room))
114     (believes ?character (not (locked ?room)))
115     (not (believes ?character (locked ?room)))
116    )
117   :fail
118    (when (and (locked ?room) (has ?character ?key) (not (unlocked-by ?
      room ?key)))
119     (and
120      (believes ?character (not (unlocked-by ?room ?key)))
121      (not (believes ?character (unlocked-by ?room ?key)))
122     )
123    )
124   :agents (?character)
125  )
126 )
```

Problem

```
(define (problem six-rooms)
 (:domain rooms)
 (:objects
  alice - character
  letter1 - character
  letter2 - character
  ;;;letter1 - letter
  ;;;letter2 - letter
  key - key
  star - star
  r1 - room
  r2 - room
  r3 - room
  r4 - room
  r5 - room
  r6 - room
 )
 (:init
  (at letter1 r2)
  (at letter2 r6)
  (at alice r3)
  (in key r4)
  (in star r5)

  (locked r5)
  (unlocked-by r5 key)

  ;; Alice (wrongly) believes no rooms are locked, and there are no keys
    .
  (believes alice (at alice r3))
  (believes alice (not (locked r1)))
```

```
  (believes alice (not (locked r2)))
  (believes alice (not (locked r3)))
  (believes alice (not (locked r4)))
  (believes alice (not (locked r5)))
  (believes alice (not (locked r6)))

  (intends alice (has alice star) )

  ;; The informants hold crucial knowledge of where the star and key are
  (believes letter1 (in star r5))
  (believes letter2 (in key r4))
  (believes letter2 (unlocked-by r5 key))
  (believes letter1 (at letter1 r2))
  (believes letter2  (at letter2 r6))
 )

 (:goal
  (and
   ;;(has alice star)
   ;;(at alice r6)
   ;;(at alice r4)
   (has alice star)
  )
 )
)
```

Name of Candidate: Matthew Christensen
Address: 130 S. 900 E. #207
Salt Lake City, Utah, 84102