The AIIDE-17 Workshop on
Intelligent Narrative Technologies
WS-17-20

# Character Beliefs in Story Generation

**Markus Eger, Chris Martens**
meger@ncsu.edu, crmarten@ncsu.edu
Principles of Expressive Machines Lab
NC State University
Raleigh, NC, USA

**Abstract**

Character beliefs play a central role in many narratives, but are often not represented in planning-based author-centered narrative generation systems, or only represented in an ad hoc way. In this paper we will discuss how actions in Dynamic Epistemic Logic that affect the story world as well as the characters' beliefs, can be used in the context of narrative generation. By using this logical foundation we can ensure that the actors' beliefs are maintained in a logically consistent way, but we will also discuss how our system supports writing these complex logical operations in a concise way. We will also show how our system, which has limited planning capabilities, can come up with simple stories that cause characters to have beliefs desired by the author, and how our approach can be integrated with other work in the field.

## Introduction

**Darth Vader:** Obi-Wan never told you what happened to your father.
**Luke:** He told me enough! He told me you killed him!
**Darth Vader:** No, I am your father.
**Luke:** No. No! That's not true! That's impossible!

Disparities in character beliefs are an essential part of many narratives across many genres, as exemplified by the iconic scene from *Star Wars Episode V: The Empire Strikes Back* (Lucasfilm 1980) presented above. In this particular instance, Luke held the belief that his father was killed by Darth Vader, because this was conveyed to him earlier by his mentor, Obi-Wan Kenobi. However, as is revealed in this scene, Darth Vader actually is his father, causing a surprised reaction by Luke (and the audience at the time). Of course, characters believing something only to be later informed that their beliefs were incomplete or wrong is not unique to Star Wars or the genre of Science Fiction, but does occur in virtually every narrative genre. Some genres, like detective stories, are even entirely built around the notion of a detective character obtaining information over time and forming the proper beliefs while the perpetrator of the crime tries to mislead them. In this case, it is common to have multiple nested levels of beliefs, e.g. the criminal believing that the detective believes that a third party is guilty. In fact, TV Tropes lists such deeper levels of nested beliefs as its own trope.[1]

Since beliefs and beliefs about beliefs often play a central role in narratives, it is be beneficial to be able to represent and reason about them in computational systems that generate narratives as defined by an author. In this paper we present how our work on practical Dynamic Epistemic Logic can be used to model character beliefs in narratives and used to generate (simple) narratives that fulfill an author's stated goals for the characters. We will also discuss how this approach could be integrated with existing techniques used in the field.

## Related Work

Narrative Generation, or more precisely, fabula generation is the problem of algorithmically creating stories. One approach that has been applied successfully is to view it as a problem-solving task in which an algorithm is tasked with finding a sequence of actions in the story world that fulfill the author's goals (Dehn 1981). It has been noted that AI planning can be used for this process, often with guidance by the human author to provide a general trajectory for the story in the form of landmarks that the resulting plan has to pass through (Porteous and Cavazza 2009). However, because the planner *only* works towards the author's goal, this approaches may result in undesirable stories, because the characters don't always act in their own best interest. To prevent this from happening, Riedl and Young (2010) extended the standard planning formalism with a representation of intentionality, which was further extended by Ware and Young (2011) to enable actors to drop intentions in the face of conflict. However, Cohen and Levesque (1990) have argued that the notion of intention is tightly linked with belief and can not be viewed independently. In the context of narrative generation, Eger et al. (2015) have also argued that a richer representation of narratives based on modal logic may be beneficial in certain scenarios. While their argument is that discourse generation, i.e. generating how a story is told, may need a representation of characters' beliefs to be able to tell the story from their point of view, we argue that having this belief representation benefits fabula generation as well. In fact, Teutenberg and Porteus (2015) have already

[1] http://tvtropes.org/main/IKnowYouKnowIKnow

described how to incorporate character beliefs into intentional planning, and the benefits that come with it. They use the story of Romeo and Juliet as an example where characters' (wrong) beliefs are central to the story. While their approach allows the representation of beliefs about beliefs, and is applicable to a wide range of narratives, it does not come without limitations. For example, a deceptive action that they show, feign death, is deceptive by virtue of the fact that it has two effects: One to indicate the perception of the agent, namely that the target is not alive, and one to indicate that the act was deceptive, which in this case is called "feigning death". Every other action that refers to actors being alive or not then also has to be modified to take the case of feigning death into account. When multiple such deceptive acts can occur for the same fact, this can become unmanageable for domain authors. Our approach avoids such problems by using a logical formulation that can naturally represent deception and mistaken beliefs.

Epistemic Logic is the logic of knowledge and beliefs (Hintikka 1962) (for a more current and accessible description see the Stanford Encyclopedia of Philosophy Archive (Hendricks and Symons 2015)). It can be used to reason about the state of the world, as well as what actors believe about the world, believe about other actors' beliefs, etc. The semantics of such logics are often given by a possible worlds model (Kripke 1963), where one world is marked as the actual world, which describes the state of the physical world, and several worlds that are in relation to the actual world describing which worlds an actor considers possible in the actual world. This relation is called the (belief) *accessibility relation*, and the set of worlds that are belief accessible by an actor represent that actor's beliefs and uncertainty about the world. Each of these worlds may in turn have other worlds that are belief-accessible by the actor themselves or other actors, representing beliefs about beliefs. We will call the set of worlds $V$ that is belief-accessible to an actor $a$ from a world $w$ the *appearance* of that world $w$ to the actor $a$, and say that the world $w$ *looks like* any of the worlds in $V$. An actor *believes* some fact $\phi$ if it holds in all worlds that are belief-accessible from the actual world. Such a belief will be written as $\Box_a \phi$.

While Epistemic Logic is concerned with the representation of a fixed state of the world and the beliefs of the actors therein, Dynamic Epistemic Logic (DEL) (Van Ditmarsch, van Der Hoek, and Kooi 2007) provides the means to describe how the world and beliefs change over time. While there are several different flavors of DEL, the classical approach is to view actions similar to states: An action has an *actual action* that describes how it changes the physical world, as well as alternative actions, some of which are accessible to each actor via a relation similar to the accessibility relation above. These alternative actions describe the appearance of the actual action to the different actors. This makes it possible to model, for example, an action that changes the value of a variable $x$ to 3, while it appears to one actor that the value of the variable is changed to 2 and another actor may believe that is changed to either 1 or 5. Actions are applied to states by applying the actual action to the actual state, and the appearance of an action to an actor to every world in the appearance of the actual world to that actor. This way, the worlds an actor considers possible at any point are the result of what the actions that happened looked like to the agent. A particular flavor of DEL, presented by Baltag (2002), also allows actors to merely *suspect* that an action happens, even if it doesn't in reality. In this model, actions have preconditions and can only be applied to worlds where these preconditions hold. If an actor suspects that an action that requires $p$ to be true to happen, an action that appears to them as having $p$ as a precondition will be executed. This appearance of the action is applied to every world in the appearance of the actual world to that actor, but will only apply to the worlds in which $p$ holds. The result is that the actor will discard any worlds that they previously considered possible in which $p$ does not hold, causing them to believe $p$. In other words, *suspecting* something is the process of adopting a belief without having evidence for the fact, i.e. the belief may or may not be mistaken. On the other hand, the process of acquiring a belief based on the objective reality of the world is called *learning* and works analogous, with the difference that the actual action also has $p$ as a precondition, i.e. if $p$ does not hold, the action does not apply.

## Approach

Our approach provides the means to write actions in Dynamic Epistemic Logic that can be used in a story world. The advantage of using a logical representation like this is that it comes with sound reasoning capabilities that ensure that the results of actions are logically consistent. We will show an application of this in the example section later, but first we will detail the capabilities of our system. It is based on Baltag's variant of Dynamic Epistemic Logic to represent epistemic actions, but supports a more concise syntax that is then compiled to the logical notation itself. The details of this compilation process are discussed in more detail in an upcoming publication (Eger and Martens 2017). For this present paper, we will just briefly describe the syntax we support. Then we will detail how these actions can be executed and how this execution process can be used for story generation, similar to a planning process.

### Action Syntax

Actions, in our system, are written in a syntax inspired by imperative programming languages of the C-family.

```
pickup(a: Actors, i(a): Item)
{
    precondition at(a) == at(i);
    if (holdable(i) == True)
    {
        public (a) holder(i) = a;
    }
    else
    {
        learn (a) holdable(i) == False;
    }
}
```

Listing 1: An example action in our language

Listing 1 shows an example of an action, in which an actor attempts to pick up an object. If the object can be held, they will then be holder of that object, otherwise they will learn that it is not holdable. As this example illustrates, actions in our language can have the following components:

- **Parameters**, each with a type, which is a set of objects the parameter can be bound to. Note that some parameters may be *secret* to most actors, and only known to some. This is written by placing the actors the parameter is known to in parenthesis after the parameter name. In the example action, the $i$ parameter, which is the item that is picked up, is only known to $a$, the actor performing the action, and thus is written `i(a): Item`. The effect of this is that other actors will know that an object has been picked up by the actor $a$, but not which, i.e. the appearance of the action to other actors is such that the $i$ parameter is bound to any valid value.
- **Assignment statements** describe the actual changes that are made to the world. Our language allows the definition of arbitrary *properties* of objects, to which values can be assigned.
- **Preconditions** are conditions that must be true for the action to be executable.
- **If/Else Conditions** allow the conditional execution of statements.
- **Public statements** modify how changes to the world are shown to the actors. By default, statements are transparent to the actors, i.e. they know that an action is happening, but not what that action does. By making a statement *public* to a set of actors, these actors will truthfully believe that the change of the property value happens.
- **Learn/Suspect statements** allow the explicit communication of conditions to actors. A `learn` statement will truthfully communicate to the given actor(s) that a condition holds, while the suspect statement that follows the same syntax with the keyword `learn` replaced with `suspect` may communicate things that are not true.

Note that conditions in our system may contain quantifiers. This makes it possible, for example, to tell an actor that there exists an object that satisfies a certain condition, which we write as `learn (a) Exists o in Objects: p(o) == x`. There are also two special pseudo-quantifiers for the `learn` and `suspect` statements:

- **Which** behaves similar to `Exists`, but in addition to telling the actor that an object fulfills the condition, the actor is also told which object that is. If there are multiple such objects, this will result in a multiple result states, one for each object. The main use for this pseudo-quantifier is to tell an actor the value of a property.
- **Each** will tell the actor the exact subset of objects for which a condition holds. In other words, for each object in the given set, it will tell the actor if that object satisfies the condition. This pseudo-quantifier is mainly useful to have characters partition a set of objects based on some criterion.

Additionally, because epistemic actions have an effect on the actors' beliefs, it is also possible to query those beliefs in a condition. Analogous to the $\Box_a$-notation from epistemic logic, this is written using as `[] (a) c`, which represents that actor $a$ believes the condition $c$.

## Initial State

To use the actions defined using the syntax described above, our system must be provided with an initial state to which the actions can then be applied. The way the initial state is defined in our system mirrors the possible worlds model from Epistemic Logic: The user defines one actual world, and any number of alternative possible worlds and which actors can access which other worlds from each world. Listing 2 show an example definition of an initial state. It defines that the actor Sherlock is at the location Baker Street in the initial state, and the appearance of that world to Sherlock is only the initial world itself, while the appearance of that world to Moriarty includes an alternative world, *Alt1*, in which Sherlock is at Scotland Yard. These two worlds represent Moriarty's uncertainty about the location of Sherlock. Note that even the alternative world *Alt1* has an accessibility defined for Sherlock, which only includes *Alt1*. This represents, that while Moriarty does not know whether Sherlock is at Baker Street or at Scotland Yard, she[2] knows that he knows where he is in either case. Likewise, Sherlock knows that Moriarty does not know for sure where he is, but considers these two options possible.

```
Initial:
at(Sherlock) = BakerStreet
looks like (Sherlock): Initial
looks like (Moriarty): Initial, Alt1

Alt1:
at(Sherlock) = ScotlandYard
looks like(Sherlock): Alt1
looks like (Moriarty): Initial, Alt1
```

Listing 2: An example for an initial state

## Action Execution

To execute actions, a user provides an *execution trace*, which consists of actions to be executed, print statements, queries to the state and goals to be planned towards. The syntax to execute an action is the same as used for a function call in C-like languages, i.e. consisting of the name of the action followed by the values its parameters should be bound to in parenthesis, separated by commas. For example, to execute the `pickup` action from above, one would write `pickup(Sherlock, clue)`. The first action in the execution trace is applied to the initial state, and produces a new state to which the next action is then applied, and so forth. Rather than actually executing an action, it is also possible to have an actor merely suspect that it happens, for example by writing `Moriarty suspects pickup(Sherlock, clue)`. This will not change the

[2]To distinguish between them we will refer to Moriarty with female pronouns and to Sherlock with male pronouns.

actual world at all, but only have an effect on the suspecting actor's mental state. At any point in the execution trace, a print statement may be used to show the current actual world, with or without its appearance to the different actors. This output contains all facts that hold in the actual world or its appearances, and may be hard to parse for humans. We therefore also provide the capability to query the state using the same syntax that is available for conditions in if-statements, including the ability to query actors' beliefs. Finally, and for the purpose of this paper most importantly, it is not necessary for the user to provide a concrete list of actions that they want to be executed, but can simply provide a condition that they want to hold as a goal. The system will then search for and execute a sequence of actions such that that condition holds. This sequence of actions may contain the actual execution of actions, or just a suspicion on part of a character that an action happened. We will describe how this search process works in more detail in the next section.

## Goal-Directed Search

When the execution trace contains a goal definition, which is indicated by the keyword `goal`, followed by an arbitrary condition, for example `goal: holder(clue) = Sherlock`, our system will try to find a sequence of actions such that after the execution of these actions starting at the current state, the goal condition is fulfilled, similar to the definition of the classical planning problem (Hendler, Tate, and Drummond 1990). When our system encounters a goal definition, it will insert all possible combinations of parameters into the action definitions, and additionally add versions of each of these action instances that are merely suspected by each actor. This set of actions and suspected actions is then used in a breadth-first search (Bundy and Wallen 1984) until a state is reached in which the goal is satisfied. In this case, the number of possible ways to assign parameters to all actions, multiplied with one plus the number of actors (for the suspected actions) determines the branching factor, and can be prohibitively large. We will first show a short example that is still tractable using this approach and then discuss how it could be integrated with more efficient planning approaches.

## Example

To demonstrate how epistemic actions defined in our language and the limited goal-finding capabilities of our system can be used in narrative generation, consider a detective story, in which the murderer wants to frame another character by hiding the gun in their garden. Listing 3 shows how we can define actions for this domain. The `take` and `put` actions allow any suspect to pick up or put down a gun, while the `kill` action requires that they own the gun used for the murder.[3] Note that all three of these actions have a secret parameter `m` that is only known to `m` themselves, i.e. only the person performing the action knows that they were the ones doing it, everyone else will consider any person to be the possible actor for the action.

[3]To shorten the example, we have conflated owning a gun with carrying it.

```
take(m(m): Suspects, g: Guns)
{
    gunowner(g) = m
}
put(m(m): Suspects, g: Guns, l: Locations)
{
    precondition gunowner(g) == m;
    gunowner(g) = Null;
    at(g) = l
}
kill(m(m): Suspects, v: Victims, g: Guns)
{
    precondition gunowner(g) == m;
    murderer(v) = m
}
find(d: Detectives, g: Guns, l: Locations)
{
    precondition at(g) == l;
    suspect (d): gunowner(g) == owner(l)
}
```

Listing 3: Some actions for the detective domain

The `find` action, finally, causes the detective that performs it to suspect that the person who owns the location where the gun was found is also the owner of the gun. This means that `find` has a purely epistemic effect, which causes the detective to believe that they found the gun owner, whether or not they are right.

```
Initial:
owner(Manor) = Moriarty
owner(Shed) = Watson
looks like (Moriarty,Sherlock,
            Watson,Victim): Initial

Execute:
take(Moriarty,pistol)

goal: [] (Moriarty): [] (Sherlock):
        murderer(Victim) == Watson

print: state
query: [] (Sherlock):
      murderer(Victim) == Watson
```

Listing 4: The initial state and execution trace to generate part of the detective story

To generate a narrative, we can execute a trace like the one shown in listing 4, starting from the initial state also shown there. Note that, in this example, every character knows exactly what the state of the world is initially. We then explicitly execute the `take` action to have Moriarty take the pistol. Because this action has a secret parameter, in particular the person taking the gun is only known to that person themselves, other actors will only know that the action was performed, but not what the actual value of that parameter was. For example, the detective will now consider multiple worlds possible, one for every possible suspect having taken the pistol, as shown in figure 1. Note that, as a logical consequence of how the action is defined, Watson also knows that Moriarty took the gun, since she is the only other suspect, and Watson knows that he does not have the gun. This is an

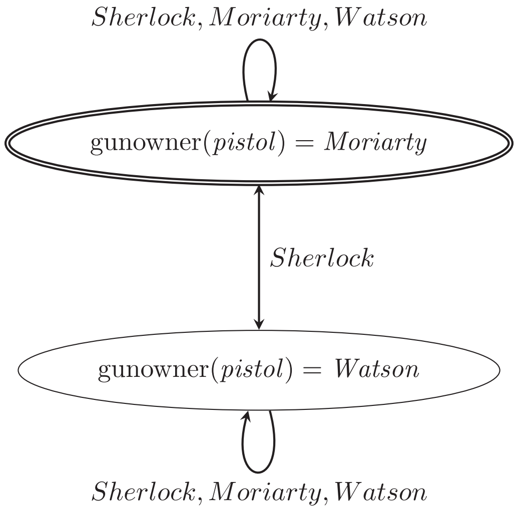
$Sherlock, Moriarty, Watson$
gunowner($pistol$) = $Moriarty$
$Sherlock$
gunowner($pistol$) = $Watson$
$Sherlock, Moriarty, Watson$

Figure 1: The epistemic state after Moriarty takes the gun. Each node represents one possible world with the actual state on top marked with a double outline. The directed dges represent the accessibility relation, i.e. in each world an actor considers all worlds possible that can be reached with an edge labeled with that actor's name. The interpretation of this particular state is that Sherlock does not know which of the two suspects owns the gun, but he knows that whoever took the gun, knows that they took the gun, and they know that Sherlock does not know.

example for how the logical foundation we are using keeps the actors' mental states logically consistent.

After the execution of the `take` action, we use the goal-directed search to find an action trace to satisfy a condition, in this case that Moriarty believes that Sherlock believes that the murderer is Watson, i.e. we want a sequence of actions such that Moriarty believes that she framed Watson for the murder of the victim. When executing this line, our system will respond with the sequence of actions `kill(Moriarty, Victim, pistol)`, `put(Moriarty, pistol, Shed)` and `Moriarty suspects find(Sherlock, pistol, Shed)`. This last action demonstrates how our implementation utilizes Baltag's logic to represent that actions can just be suspected to happen, even if they are not actually executed. The story fragment we created thus has Moriarty first pick up the pistol, kill the victim with it, and then place it in the shed, which is owned by Watson. Moriarty then thinks that Sherlock will find the gun in the shed, causing him to suspect Watson of the murder. This last step may seem surprising, since the only effect of the `find` action is that Sherlock believes that the gun owner is the owner of the location where he found the gun. However, Sherlock also knows that a kill action happened previously and will automatically eliminate previously established worlds that have turned out to be inconsistent, such as the world in which the `kill` action was performed by someone other than the gun owner. After the system has found an action sequence that satisfies the goal condition, it will also execute these actions, resulting in a new state. We can then execute more actions, or perform further goal-directed searches, or use any of the query or print-capabilities of our system. In this example, we first print the state of the world. The parts of the output of this operation that pertain to Sherlock's and Moriarty's mental state can be seen in figure 2. As can be seen, Moriarty only considers one world possible, in which she is the murderer (to keep the output more concise, the system does not display the appearances of this world, i.e. her beliefs about beliefs can not be seen in this output.) We can also see that Sherlock still considers two worlds possible: One in which Moriarty is the murderer and one in which it is Watson. This is because the action of Sherlock finding the pistol was only suspected by Moriarty but has not actually happened (yet). In a story generation setting, it would be up to the author to now have Sherlock actually find the gun (perhaps by formulating a goal that states that Sherlock should suspect Watson), or perform a completely different action/pursue a different goal. Finally, because parsing the state printed in this way can get really cumbersome, we can also just query it. The query shown at the end of listing 4 asks whether or not Sherlock believes that Watson is the murderer. As we just showed, Sherlock considers that possible in one world, but not in all worlds (because he still considers the possibility that Moriarty is the murderer), and therefore the query will be answered with False.

While this example was kept short to demonstrate the capabilities of our system, there are several ways in which it could be expanded upon. While we only provided 4 different actions, each of which had a clear purpose in our example, the modular nature of this approach allows domain authors to define a whole range of possibilities for deception, such as Moriarty luring Watson to the scene of the murder, or planting false DNA at the scene. The goal of "Moriarty believes that Sherlock believes that Watson is the murderer" then merely serves as a landmark that all detective stories must contain, but the exact way of achieving it is up to the system.

```
compare(d: Detective, s: Suspect, v: Victim)
{
  public (d) if (fingerprints(location(v))
      == s)
  {
     suspect (d) murderer(v) == s
  }
  else
  {
     suspect (d) murderer(v) != s
  }
}
```

Listing 5: Comparing fingerprints with a suspect's as an epistemic action

Furthermore, other popular investigative actions, like dusting for fingerprints can be modeled in such a way that Sherlock can rule out suspects once he manages to compare their fingerprints to the ones at the scene, for example by using the `compare` action shown in Figure 5. In this example,

```
The world is now:
Truth:
  ["at(pistol,Shed)",
   "gunowner(pistol,Moriarty)",
   "murderer(Victim,Moriarty)",
   "owner(Manor,Moriarty)",
   "owner(Shed,Watson)"]

looks like:
  Moriarty:
     [["at(pistol,Shed)",
       "gunowner(pistol,Moriarty)",
       "murderer(Victim,Moriarty)",
       "owner(Manor,Moriarty)",
       "owner(Shed,Watson)"]]
  Sherlock:
    [["at(pistol,Shed)",
      "gunowner(pistol,Moriarty)",
      "murderer(Victim,Moriarty)",
      "owner(Manor,Moriarty)",
      "owner(Shed,Watson)"],
     ["at(pistol,Shed)",
      "gunowner(pistol,Watson)",
      "murderer(Victim,Watson)",
      "owner(Manor,Moriarty)",
      "owner(Shed,Watson)"]]
  ...
```

Figure 2: The epistemic state after the system found and executed an action sequence such that the provided condition is satisfied. Note that only the appearance of the actual world is printed, and not the appearance of that appearance etc.

the `if` statement is public to the detective, which means they will know which branch is taken, i.e. after performing the action they will know whether or not the fingerprints they found belong to the suspect, and will subsequently believe that they are the murderer in case of a match, or believe that they are not the murderer otherwise.

## Limitations and Future Work

While the logic we use as the basis for our system can express a wide range of relations between character's beliefs, it is not without limitations. One such limitation is illustrated by The Star Wars example from the introduction. In this scenario, Luke believed that his father was dead, i.e. in all worlds that he considered possible his father was dead. However, when Darth Vader revealed that he is Luke's father, Luke then believed that his father was dead, Darth Vader was his father and Darth Vader was alive, which caused a contradiction. When we use a naive encoding of the narrative, this would leave him with no worlds that he would consider possible (and indeed, he uses the phrase "That's impossible"). This is not technically a limitation of the logic, but something that must be taken into consideration when modeling narratives in it. One way to avoid it is to model the epistemic effect that happened as consisting of two parts: First, Luke increased his uncertainty by adding new worlds in which his father was indeed alive and then he discards the worlds in which his father is not Darth Vader. This approach has the drawback that it requires a certain level of awareness of this kind of accidental contradictions on part of the domain author. The other approach would be to explicitely model the process of learning the identity of an unknown person such that the properties of the unknown person are overwritten with those of the identity, or more concretely, when Luke learns that Darth Vader is his father, he also learns that his father has all properties that Darth Vader has, including the fact that he is alive, without regard for what he previously believed about him.

The main limitations of our current system are all a consequence of using breadth-first search. The story fragment presented above can be found in less than one minute on modern hardware, but longer stories, or stories with more available actions will take significantly longer. Adding just one more necessary step to the story extends the planning process to several minutes. We also limited ourselves to one level of suspicion to limit the branching factor accordingly. However, actions defined in our system have well-defined preconditions and effects because they compile to actions as defined by Baltag (2002), for which a precondition function is defined in his paper. This function is actually a necessity for the execution of the actions as well, and is therefore already present in our implementation. Least-commitment planning as described by Weld (1994) as an abstract process requires atomic, deterministic actions, an omniscient planner [4], and that no change of the world is external to the plan. Our actions fulfill these requirements if care is taken to avoid non-determinism, but while our implementation uses a (simplistic) search through state-space, using a least-commitment plan-space search approach as described in Weld's article would make the search more efficient. The main difference to the typical formulation of this approach is that causal links can then also contain beliefs of characters. Consider the detective example from above: The goal state consists of a belief of an actor about another actor's belief. To find an action to fulfill this particular goal, the planner needs to choose an action that has a belief about a belief as an effect, and add that action's preconditions to the agenda. Furthermore, the idea of variable bindings as discussed in the article also has a direct parallel in our language with an action's parameters and could make the planning process even more efficient. The feasibility of this general idea has already been demonstrated by Herzig et al. who show how to use a different epistemic logic in a planning process (Herzig, Lang, and Marquis 2003). The logic they are using does not increase the theoretical complexity of the planning process, which is already quite high in theory, while our approach is more concerned with its application in practice.

Using such a planning formalism would bring our approach closer to current research in the area, since many contemporary story generation systems are based on various flavors of planning, as described earlier. It would then be possible to combine a narrative planner, even a state-space

[4]Note that Weld calls this an omniscient agent, but the agent in this case is the author or planner that performs the story generation

one like Glaive (Ware and Young 2014), with our action definition language, and use epistemic actions rather than plain PDDL actions. The benefit would be that many advances made in classical planning, such as better heuristics, and work on narrative planning in particular, such as work on narrative diversity (Amos-Binks, Roberts, and Young 2016) would be directly applicable. Additionally, this approach would also enable a more detailed model of intentionality, that is based on character beliefs about the achievability of their goals.

## Conclusion

We have presented how our work in modeling epistemic actions can be applied to the problem of story generation. Our system enables the definition of Dynamic Epistemic Logic actions in a syntactically concise way, and can be used to apply a sequence of these actions to an initial state. We have also described how our system provides the means to search for a sequence of actions, or suspected actions, that satisfy a goal condition. This capability mirrors how planning is used for narrative generation, and we have showed a short example of parts of a detective story in which the murderer tries to frame another person for a murder. The goal in this example is that the murderer believes that the detective believes that someone else is the murderer. We have presented the execution trace that our system finds that includes actions performed by the murderer, as well as an action that she merely suspects to happen, resulting in the desired goal state. Finally, we have discussed how our system could be integrated with planning approaches to benefit from current research in the area.

## References

Amos-Binks, A.; Roberts, D. L.; and Young, R. M. 2016. Summarizing and comparing story plans. In *OASIcs-OpenAccess Series in Informatics*, volume 53. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Baltag, A. 2002. A logic for suspicious players: Epistemic actions and belief–updates in games. *Bulletin of Economic Research* 54(1):1–45.

Bundy, A., and Wallen, L. 1984. Breadth-first search. In *Catalogue of Artificial Intelligence Tools*. Springer. 13–13.

Cohen, P. R., and Levesque, H. J. 1990. Intention is choice with commitment. *Artificial intelligence* 42(2-3):213–261.

Dehn, N. 1981. Story generation after TALE-SPIN. In *IJCAI*, volume 81, 16–18.

Eger, M., and Martens, C. 2017. Practical specification of belief manipulation in games. In *Proceedings of the 13th AAAI International Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Eger, M.; Barot, C.; and Young, R. M. 2015. Merits of a temporal modal logic for narrative discourse generation. In *Eighth Intelligent Narrative Technologies Workshop*.

Hendler, J. A.; Tate, A.; and Drummond, M. 1990. AI planning: Systems and techniques. *AI magazine* 11(2):61.

Hendricks, V., and Symons, J. 2015. Epistemic logic. In Zalta, E. N., ed., *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, fall 2015 edition.

Herzig, A.; Lang, J.; and Marquis, P. 2003. Action representation and partially observable planning using epistemic logic. In *Proceedings of the 18th international joint conference on Artificial intelligence*, 1067–1072. Morgan Kaufmann Publishers Inc.

Hintikka, J. 1962. *Knowledge and belief: an introduction to the logic of the two notions*, volume 4. Cornell University Press Ithaca.

Kripke, S. A. 1963. Semantical analysis of modal logic i normal modal propositional calculi. *Mathematical Logic Quarterly* 9(5-6):67–96.

Lucasfilm. 1980. Star Wars Episode V: The Empire Strikes Back. [Movie].

Porteous, J., and Cavazza, M. 2009. Controlling narrative generation with planning trajectories: the role of constraints. In *Joint International Conference on Interactive Digital Storytelling*, 234–245. Springer.

Riedl, M. O., and Young, R. M. 2010. Narrative planning: Balancing plot and character. *Journal of Artificial Intelligence Research*.

Teutenberg, J., and Porteous, J. 2015. Incorporating global and local knowledge in intentional narrative planning. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, 1539–1546. International Foundation for Autonomous Agents and Multiagent Systems.

Van Ditmarsch, H.; van Der Hoek, W.; and Kooi, B. 2007. *Dynamic epistemic logic*, volume 337. Springer Science & Business Media.

Ware, S. G., and Young, R. M. 2011. CPOCL: A narrative planner supporting conflict. In *Proceedings of the 7th AAAI International Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Ware, S. G., and Young, R. M. 2014. Glaive: A state-space narrative planner supporting intentionality and conflict. In *AIIDE*.

Weld, D. S. 1994. An introduction to least commitment planning. *AI magazine* 15(4):27.