Al Planning: Planning with Dynamic Epistemic Logic

Markus Eger

2019-11-15

Outline

- Motivation
 - One Night Ultimate Werewolf
- 2 Review: Dynamic Epistemic Logic
- Open Planning with Dynamic Epistemic Logic
- Story Generation

Outline

- Motivation
 - One Night Ultimate Werewolf
- 2 Review: Dynamic Epistemic Logic
- Opening with Dynamic Epistemic Logic
- 4 Story Generation

Review: The Planning Problem

A planning problem consists of three parts:

- A definition of the current state of the world
- A definition of a desired state of the world
- A definition of the actions the agent can take

Outline

- Motivation
 - One Night Ultimate Werewolf
- 2 Review: Dynamic Epistemic Logic
- Opening with Dynamic Epistemic Logic
- Story Generation

One Night Ultimate Werewolf



(Source: whatsericplaying.com)

One Night Ultimate Werewolf

- Competitive
- Players are secretly assigned roles
- Factions: Werewolves, Villagers
- Goal of the Villagers: Find the Werewolves
- Goal of the Werewolves: Avoid detection
- Night phase for actions, day phase for communication
- Players can lie
- Players' roles can change without their knowledge

• Seer: Look at another player's card



- Seer: Look at another player's card
- Robber: Exchange cards with another player, look at the new card



- Seer: Look at another player's card
- Robber: Exchange cards with another player, look at the new card
- Rascal: May exchange your two neighbors' cards



- Seer: Look at another player's card
- Robber: Exchange cards with another player, look at the new card
- Rascal: May exchange your two neighbors' cards
- Insomniac: Look at your own card



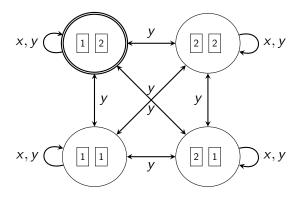
One Night Ultimate Werewolf - Challenges

- Hidden information
- Deception
- Suspicion
- Good game play requires a model of belief

Outline

- Motivation
 - One Night Ultimate Werewolf
- 2 Review: Dynamic Epistemic Logic
- Open Planning with Dynamic Epistemic Logic
- 4 Story Generation

Formal Model: Possible Worlds



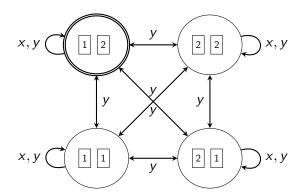
Formal Model: Dynamic Epistemic Logic

Baltag's variant of Dynamic Epistemic Logic

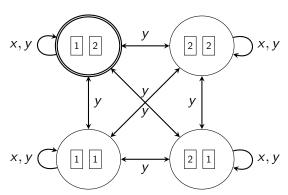
• Possible worlds represent agents' beliefs

 Actions can add worlds (increase uncertainty) or remove worlds (decrease uncertainty)

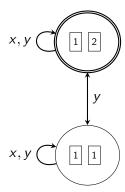
learn (y): left(x) == 1



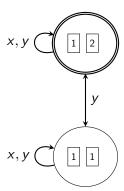
learn (y): left(x) == 1 $(? left(x, 1))^{*y}$



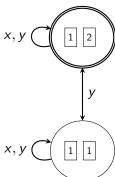
learn (y): left(x) == 1 $(? left(x, 1))^{*y}$



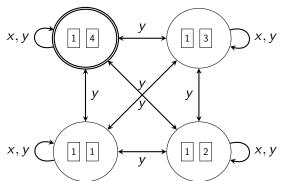
```
drawright(p: Players, c(p): Cards)
    right(p) = c
```



```
\begin{split} & \text{drawright}\,(p\colon \, \text{Players}\,,\,\, c(p)\colon \, \text{Cards}) \\ & \text{right}\,(p) \,=\, c \\ & (\text{flip right}(x,2)\cdot \text{flip right}(x,4))^{*x} \cdot \\ & (\text{flip right}(x,2)\cdot (\text{flip right}(x,1)+\text{flip right}(x,2)+\text{flip right}(x,3)+\text{flip right}(x,4)))^y \end{split}
```



```
\begin{split} & \text{drawright}\,(p\colon \, \text{Players}\,,\,\, c(p)\colon \, \text{Cards}) \\ & \text{right}\,(p) \,=\, c \\ & (\text{flip right}(x,2)\cdot \text{flip right}(x,4))^{*x} \cdot \\ & (\text{flip right}(x,2)\cdot (\text{flip right}(x,1)+\text{flip right}(x,2)+\text{flip right}(x,3)+\text{flip right}(x,4)))^y \end{split}
```



Dynamic Epistemic Logic - Limitations

- Basic operation flip changes one bit at a time
- Game state has to be encoded in one bit units
- Actions have to change single bits at a time
- Interesting actions are cumbersome to write

Example: Swap two players' roles

```
(?role(b,Rascal) . ?phase(game,TMPhase) .
(?role(tmp,Werewolf), flip role(tmp,Werewolf) + ?role(tmp,Rascal), flip role(tmp,Rascal) +
?role(tmp,Villager) . flip role(tmp,Villager) + ?!role(tmp,Werewolf) .
?!role(tmp.Rascal) . ?!role(tmp.Villager)) . (?role(a.Werewolf) . flip role(tmp.Werewolf) +
?role(a,Rascal) . flip role(tmp,Rascal) + ?role(a,Villager) . flip role(tmp,Villager)) .
(?role(a, Werewolf) . flip role(a, Werewolf) + ?role(a, Rascal) . flip role(a, Rascal) +
?role(a,Villager) . flip role(a,Villager) + ?!role(a,Werewolf) . ?!role(a,Rascal) . ?!role(a,Villager)) .
(?role(c.Werewolf) , flip role(a.Werewolf) + ?role(c.Rascal) , flip role(a.Rascal) +
?role(c,Villager) . flip role(a,Villager)) . (?role(c,Werewolf) . flip role(c,Werewolf) + ?role(c,Rascal) .
flip role(c,Rascal) + ?role(c,Villager) . flip role(c,Villager) + ?!role(c,Werewolf) . ?!role(c,Rascal) .
?!role(c.Villager)) . (?role(tmp.Werewolf) . flip role(c.Werewolf) + ?role(tmp.Rascal) .
flip role(c,Rascal) + ?role(tmp,Villager) . flip role(c,Villager)) . (?role(b,Rascal) .
?phase(game,TMPhase) . (?role(tmp,Werewolf) . flip role(tmp,Werewolf) + ?role(tmp,Rascal) .
flip role(tmp,Rascal) + ?role(tmp,Villager) . flip role(tmp,Villager) + ?!role(tmp,Werewolf) .
?!role(tmp.Rascal) . ?!role(tmp.Villager)) . (?role(a.Werewolf) . flip role(tmp.Werewolf) +
?role(a,Rascal) . flip role(tmp,Rascal) + ?role(a,Villager) . flip role(tmp,Villager)) .
(?role(a, Werewolf) . flip role(a, Werewolf) + ?role(a, Rascal) . flip role(a, Rascal) +
?role(a,Villager) . flip role(a,Villager) + ?!role(a,Werewolf) . ?!role(a,Rascal) . ?!role(a,Villager)) .
(?role(a,Werewolf) . flip role(a,Werewolf) + ?role(a,Rascal) . flip role(a,Rascal) + ?role(a,Villager) .
flip role(a, Villager)) . (?role(a, Werewolf) . flip role(a, Werewolf) + ?role(a, Rascal) . flip role(a, Rascal) +
?role(a,Villager) . flip role(a,Villager) + ?!role(a,Werewolf) . ?!role(a,Rascal) . ?!role(a,Villager)) .
(?role(tmp,Werewolf) , flip role(a,Werewolf) + ?role(tmp,Rascal) , flip role(a,Rascal) +
?role(tmp,Villager) . flip role(a,Villager)) + ?role(b,Rascal) . ?phase(game,TMPhase) .
(?role(tmp,Werewolf), flip role(tmp,Werewolf) + ?role(tmp,Rascal), flip role(tmp,Rascal) +
?role(tmp,Villager) . flip role(tmp,Villager) + ?!role(tmp,Werewolf) . ?!role(tmp,Rascal) . ?!role(tmp,Village
(?role(a, Werewolf) . flip role(tmp, Werewolf) + ?role(a, Rascal) . flip role(tmp, Rascal) +
?role(a,Villager) , flip role(tmp,Villager)) , (?role(a,Werewolf) , flip role(a,Werewolf) + ?role(a,Rascal) ,
flip role(a.Rascal) + ?role(a.Villager) , flip role(a.Villager) + ?!role(a.Werewolf) , ?!role(a.Rascal) ,
?!role(a,Villager)) . (?role(b,Werewolf) . flip role(a,Werewolf) + ?role(b,Rascal) . flip role(a,Rascal) +
?role(b,Villager) . flip role(a,Villager)) . (?role(b,Werewolf) . flip role(b,Werewolf) + ?role(b,Rascal) .
flip role(b,Rascal) + ?role(b,Villager) . flip role(b,Villager) + ?!role(b,Werewolf) . ?!role(b,Rascal)
```

Example: Swap two players' roles

Gig role(trp, Nerwoil) + ?role(trp, Troblemsker) . Gig role(trp, Troblemsker) + ?role(trp, Troblemsker) + ?role(trp, Nerwoil) + ?rol Trois (a. Fromblemaker) - Frois (a. Fromblem Trunis (a, . flip role (c, Villager) + Firole (c, Warewolf) . Firole (c, Troublemaker) . Firole (c, Troublemaker) . Firole (c, Villager)) . (Frole (tmp, Warewolf) . flip role (c, Warewolf) . Troublemaker) . flip role (c, Troublemaker) + Frole (tmp, Villager) (Penis (by, Troublemaker) - Trois (top, Trois (top (Prole(a, Marwoolf) . file role(um, Marwoolf) + Prole(a, Trophemaker) . file role(um, Marwoolf) . file role(um, Marwoolf) + Prole(a, Trophemaker) . file role(um, Marwoolf) . file r | 14000 | | Testing | Test Tele Lindbarder (Signish Irredicated) Pain Nillager (Signish Irredicated) Pain Nill role(s. Topoleumbar) + Frole(s. Villages) . City role(s. Villages) + Frole(s. Villages) + Frole(s. Villages) . Tity role(s. Villages) . Tity role((Peria in, Nerword) City pris (h, Nerword) Peria (pris (h, Nerword) Peria (h, Nerword) Peria (pris (h, Nerword) Peria (flip role(b, Troublemaker) + Prole(tmp, Villager) - flip role(b, Villager) + Prole(bp, Villager) + Prole(tmp, Troublemaker) + Pro ?!role(top, Nerworld) . ?!role(top, Troublemaker) . ?!role(top, Troublemaker) . ?!role(top, Troublemaker) . (ig role(top, Trou Trois is . Trophiensker . file rois Color pink (A. Tribings) (Princip, National) (Report) (top real principal ("real principal ("re rols [b, Varietic] + Trols [b, Troublemaker] - Clip rols [b, Troublemaker] + Trols [b, Villager] + Trols [b, V Total as, Villager) . (No role to, Vince to, V To the type record of the control of role(up, reconstances) = recoe(up, vininger) : in role (up, vininger) + recoe(up, vininger) + recoe(up, vininger) : recoe(up, vining Project, Productatory (Sip roles), Tronblemaker | Arriva | Project, National | Project rices (n., remains) - And the first state of the stat (Projecting, Personal) (Projecting, Personal) (Projecting, Projecting, Projecting, Personal) (Projecting, Personal Treat-risk, Terminologies, Ter mich_indexest * heish_illager (in prints/illager * Frieds/hillager (in prints/illager * Frieds/hillager (in prints/illager) (i role(c, Villager) + 7:role(c, Verevolf) . 7:role(c, Troublemaker) . 7:role(c, Villager)) . (Youle (s, Marevolf) . flip sole (s, Marevolf) + Teole (s, Touchierakes) . flip sole (s, Touchierakes) + Toule (s, Villages) . flip sole (s, Villages)) . (Youle (s, Marevolf) ?role(c, Trophiensker) . (Sig role(c, Trophiensker) + ?role(c, Warwoold) - sign frank (mining membra) in the sign of the sign o rest top, feedbasser : "rest top; to the state of the sta Gio relata, Translatation : Prointing, National Control of the Con The Control of the Co Trois (tro. Trois) into Trois (trois (tro Tipolatum, Topolatum, Tipolatum, Total (A) Total TROIS (C, VILLAGES) . THIS POLS (K, VILLAGES) (TROIS (C), MARMONIT) . THIS POLS (K, VILLAGES) . THIS POLS (K, VILLAGES) . THIS POLS (K, VILLAGES) . THOSE (C, VILLAGES) . THIS POLS (K, VILLAGES) . THOSE (C, VILLAGES) . TH Trols (tmp, Villager) . filip rols (tmp, Villager) + Frols (tmp, Villager) + Frols (tmp, Villager) - F Proceeding the control of the contro ning first, reconnection - related, variables - related, variables - related, variables - related variable policy (months) Troub (m. Arrows) (months) (mont Technical Science - (Ege reich Armidianester - Ferich Armidianester) - (Ege reich Armidianester - Ferich Armidianester) - (Ege reich Armidiane role (up, frontienter) + Trole (b, Villager) - fits role (p, Villager) - fits role (b, Villager) rais (top, frontienness + front in Willager). His print (top, Willager). (Frontien house of the frontier fronti (Medicing, Nameshi) . (In status, Nameshi) * Fedicing, Fedicinates . (In status, Nameshi) * Fedicing, Fedicinates . (In status, Nameshi) * Fedicing, Fedicinates . (In status, Nameshi) * Fedicinates . (In status, Fedicinates . Totals (p. Position and provided by the control of . Clip role(a, Villager) + 7/role(a, Merevolf) . 7/role(a, Troublemaker) . 7/role(a, Troublemaker) . 7/role(a, Villager)) . (Trole(a, Villager) + 7role(trp, Troublemaker) . 7/role(trp, Villager) . (Trole(a, Villager)) . (Trole(a, Villager)) * Totales, Totalescalary : Tylans (pass, 1975-mag). (Total (up., Narrout) : Flip or tal (up., Narrout) : Total (u role(c, Villeger) + Trole(c, Nerword) - Trole(c, Troublestater) - Trole(c, Villeger) - (Trole(c, Nerword) - Trole(c, Nerword) - Trole(c, Troublestater) + Trole(c, Troublestater) + Trole(c, Villeger) - (Trole(c, Villeger) - (Trole(c, Nerword) - Trole(c, Nerword) - Trole(c, Villeger) - (Trole(c, Nerword) - Trole(c, Nerword) - role(c_Trophiemier) + Trole(c_Villager) - (fip role(c_Villager) + Trole(c_Villager) + (fip role(c_Villager) + Trole(c_Villager) + Trole(c_Villager) + (fip role(c_Villager) + (fip role(c_ control, and control of the control flip role(c, Troublemaker) + Trole(trp, Villager) . flip role(c, Villager))) "c) ""b

Example: Swap two players' roles

```
causeRuckus(p(p): Players, t(p): Bool)
{
    precondition initrole(p) == Rascal;
    if (eqt(t) == True)
    {
        role(tmp) = role(left(p));
        role(left(p)) = role(right(p));
        role(right(p)) = role(tmp);
    }
    else
    }
```

Outline

- Motivation
 - One Night Ultimate Werewolf
- 2 Review: Dynamic Epistemic Logic
- Open Planning with Dynamic Epistemic Logic
- 4 Story Generation

Planning with Dynamic Epistemic Logic

- Define initial epistemic state
- Define epistemic actions
- Define a goal condition
- Task: Find a sequence of actions such that the goal is satisfied

Planning with Dynamic Epistemic Logic

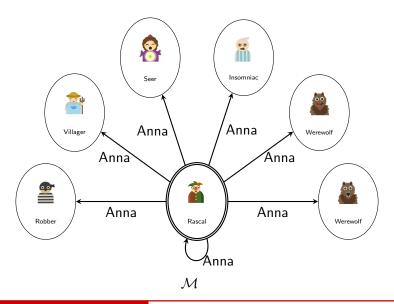
- Define initial epistemic state
- Define epistemic actions
- Define a goal condition
- Task: Find a sequence of actions such that the goal is satisfied(-ish)

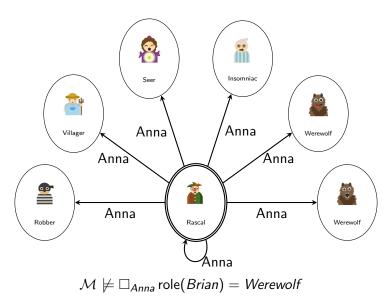
Planning with Dynamic Epistemic Logic

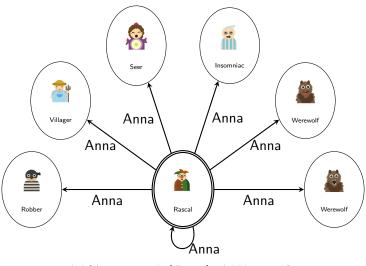
- Define initial epistemic state
- Define epistemic actions
- Define a goal condition
- Task: Find a sequence of actions such that the goal is satisfied(-ish)
- What if the goal can not be reached?

A simple scenario

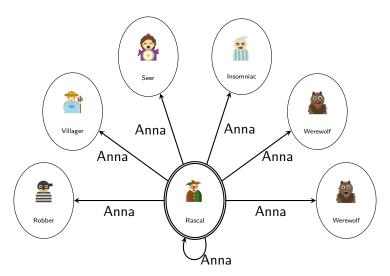
- Anna and Brian play One Night Ultimate Werewolf
- Anna got a Villager card
- Brian got the Rascal card
- But Anna does not know which card Brian has



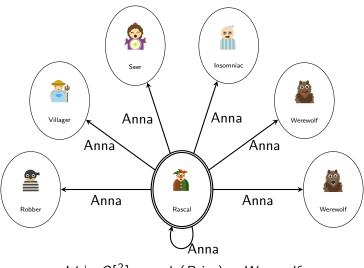




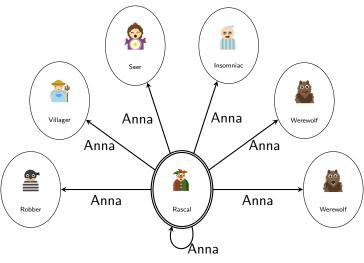
 $\mathcal{M} \not\models \Box_{Anna} \operatorname{role}(Brian) \neq Werewolf$



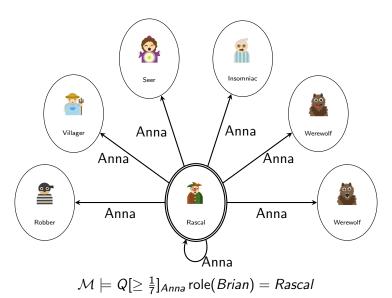
Brian is a Werewolf in 2 out of 7 worlds!



 $\mathcal{M} \models \textit{Q}[\tfrac{2}{7}]_{\textit{Anna}} \, \mathsf{role}(\textit{Brian}) = \textit{Werewolf}$



 $\mathcal{M} \models Q[\geq \frac{2}{7}]_{Anna} \operatorname{role}(Brian) = Werewolf$

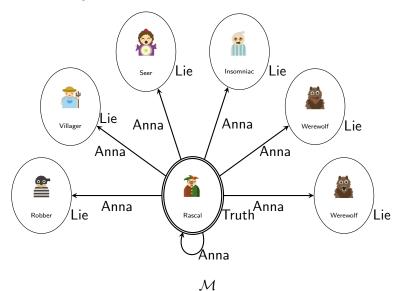


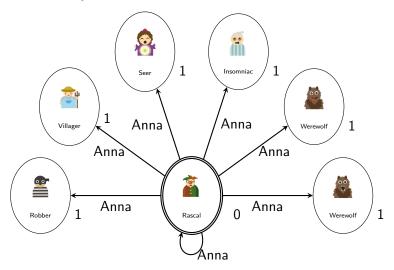
What about communication?

- Brian says (truthfully) that he is the Rascal
- What is Anna supposed to do with this information?
- Anna does not know that Brian is telling the truth

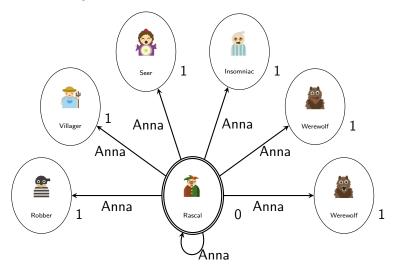
What about communication?

- Brian says (truthfully) that he is the Rascal
- What is Anna supposed to do with this information?
- Anna does not know that Brian is telling the truth
- But what if he is?

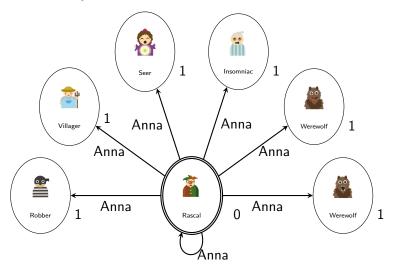




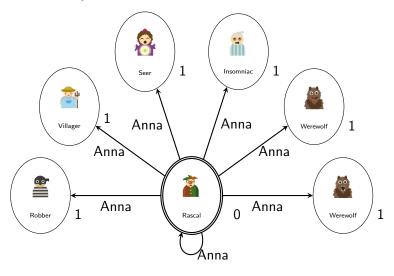
Higher numbers mean more lies, less likely to be true.



Instead of counting each world equally, use factor $\frac{1}{1+w}$.



$$\mathcal{M} \models W[\geq \frac{1}{4}]_{Anna} \operatorname{role}(Brian) = Werewolf$$



$$\mathcal{M} \models W[\geq \frac{1}{4}]_{Anna} \operatorname{role}(Brian) = Rascal$$

What does this have to do with planning?

- Brian says (truthfully) that he is the Rascal
- The outcome is not that Anna now believes him
- Anna thinks it is more likely that he is the Rascal, though

What does this have to do with planning?

- Brian says (truthfully) that he is the Rascal
- The outcome is not that Anna now believes him
- Anna thinks it is more likely that he is the Rascal, though
- Instead of planning to reach a goal: plan to maximize weighted quality

Encoding: Communicative actions

```
claimInitRole(p: Players, r: Roles, r1(p): Roles)
{
    precondition initrole(p) == r1;
    if (initrole(p) != r)
    {
        addWeight(1)
    }
    else
    {
    }
}
```

Goals

```
x = any(GoodRole)
    in
```

Goals

```
x = any(GoodRole)
    in

W [>$cert] (self): role(self) == Werewolf:
```

Goals

```
x = any(GoodRole)
   in

W [>$cert] (self): role(self) == Werewolf:

Forall p in Players: W [>0.7] (p): (role(self) == x);
```

Application: One Night Ultimate Werewolf



Outline

- Motivation
 - One Night Ultimate Werewolf
- 2 Review: Dynamic Epistemic Logic
- Opening with Dynamic Epistemic Logic
- 4 Story Generation

Detective story

Moriarty kills the victim

 Goal: Moriarty believes that Sherlock believes that Watson is the murderer

```
kill(m(m): Suspects, v: Victims, g: Guns)
{
    precondition gunowner(g) == m;
    murderer(v) = m
}
```

```
take(m(m): Suspects, g: Guns)
{
    gunowner(g) = m
}
```

```
take(m(m): Suspects, g: Guns)
{
    gunowner(g) = m
}

put(m(m): Suspects, g: Guns, 1: Locations)
{
    precondition gunowner(g) == m;
    gunowner(g) = Null;
    at(g) = 1
}
```

```
take(m(m): Suspects, g: Guns)
{
    gunowner(g) = m
put(m(m): Suspects, g: Guns, 1: Locations)
    precondition gunowner(g) == m;
    gunowner(g) = Null;
    at(g) = 1
find(d: Detectives, g: Guns, 1: Locations)
{
    precondition at(g) == 1;
    suspect (d): gunowner(g) == owner(l)
```

Story generation example: Resulting Story

```
kill(Moriarty, Victim, pistol)
put(Moriarty, pistol, Shed)
Moriarty suspects find(Sherlock, pistol, Shed)
```

References

Thank you for your attention

Strategies for a Social Deduction Game. AIIDE 2018

Markus Eger and Chris Martens. Keeping the Story Straight: A Comparison of Commitment

- Markus Eger and Chris Martens. Practical Specification of Belief Manipulation in Games. AIIDE 2017
- Henry Mohr, Markus Eger, and Chris Martens. Eliminating the Impossible: A Procedurally Generated Murder Mystery. EXAG 2018
- Markus Eger and Chris Martens. Character Beliefs in Story Generation. INT 2017
- Markus Eger and Chris Martens. Programming with Epistemic Logic. OBT 2017