Among Us: Modelling movement in Among Us with epistemic logic

Group 19:
Barbera de Mol (s3374610)
Jeroen Muller (s2590182)
Max Valk (s3246922)

June 25, 2021

Background

One kind of reasoning about the game Among Us involves the way players move from room to room. For example, consider a scenario where player c learns that b was killed in a specific room and tries to figure out who could have been the killer. He might for example think as follows: *"I was close to player a while b was killed at the other side of the map. This means a cannot be the killer."* A good player does not just consider the information that he observed directly, but also what can be extrapolated from it by considering the rules of the game. For example, a might have been out of sight of c for a short time, but c can use his knowledge of the map to deduce that there was insufficient time for a to move all the way to the place where b was killed. In this case c should still remove a from his list of suspects.

We can capture this kind of reasoning by constructing a Kripke model that describes what agents know about where everyone is at any time. To keep things simple we will study a highly simplified version of the game in which we only consider the way agents move from room to room. Movement and observations will be described by using Dynamic Epistemic Logic action models [2] with post-conditions[1]. This approach could in principle be extended to a more complex model, where we also consider things like other types of actions and agent roles, although in that case the number of states would probably become very large even when considering modest numbers of agents and rooms. In what follows we will first describe the Kripke model and the action models used to update it. We will then describe how the model was implemented in Python, describe two optimizations that are used in the implementation, and show its application to a simple example. Our source code is available on Github and can also be executed interactively using a web-based notebook on Binder without the need to install any software.

Kripke model

Let $\mathcal{A}$ denote the set of agents and $\mathcal{R}$ the set of rooms. At any time, each agent is in one of a fixed set of rooms. We describe the location of agents with the set of atomic propositions $P_{\mathcal{M},\mathcal{R}} = \{a_r | a \in \mathcal{A}, r \in \mathcal{R}\}$, where a_r denotes that a is in room r. The way agents can move from room to room is described by the relation $\rightsquigarrow$ on the rooms, such that $r_1 \rightsquigarrow r_2$ if room r_2 can be reached from room r_1 in one movement step. Since we will always allow agents to move in both directions between rooms and to remain in the same room, the relation $\rightsquigarrow$ will be reflexive and symmetric.

When $|\mathcal{A}| = m$, at any given time the knowledge in our model is described by a $\mathcal{S}5_{(\mathsf{m})}$-world $(\mathbb{M}, s)$, where $\mathbb{M} = \langle S, \pi, R_1, \ldots, R_m \rangle$ is a Kripke structure and the distinguished state $s \in S$ describes the real world. The valuation function $\pi : S \to (P_{\mathcal{M},\mathcal{R}} \to \{\mathbf{t}, \mathbf{f}\})$ assigns a truth value to all atomic propositions in each state. Because agents are always in exactly one room, the valuation function must satisfy the following constraint for every $a \in \mathcal{A}$ in order to describe a consistent state of the game:

$$[\pi(s)(a_i) = \mathbf{t}] \quad \text{iff} \quad [\pi(s)(a_j) = \mathbf{f}](j \neq i)$$

Action models

The Kripke model is updated by executing an action model with preconditions and postconditions. An action model describes a set of possible actions, of which one will be executed. The definitions we use are similar to those given in [1], but the preconditions are restricted to being conjunctions of possibly negated atomic propositions, which is sufficient for our prupose. Briefly, an action model with postconditions is very similar to a Kripke structure and has the following structure: $\mathsf{M} = \langle \mathsf{S}, \mathsf{pre}, \mathsf{pos}, R_1, \ldots, R_m \rangle$. Here S is the set of actions that are possible in the action model. $\mathsf{pre} : \mathsf{S} \to \mathcal{L}$ assigns a precondition sentence to each action (something like $p_1 \wedge p_2 \wedge \neg p_3$), and $\mathsf{post} : \mathsf{S} \to (\{p_1, \ldots\ p_j\}, \{\neg q_1, \ldots, \neg q_k\})$ assigns to each state a set of propositions that will become *true* after the corresponding action is executed and a set of propositions that will become *false*. The relations R_m relate states that are indistinguishable for agent m.

The result of executing an action (M, j) in action model $(\mathbb{M}, s)$ is the Kripke world $(\mathbb{M} \otimes \mathsf{M}, (s, \mathsf{j}))$. We will not repeat the definition of the operation, but the states (s, j) of $\mathbb{M} \otimes \mathsf{M}$ are those states in the Caresian product $S \times \mathsf{S}$ where $(mathbbM, s) \vDash pre(\mathsf{j})$, with the substitutions in the postcondition carried out. Two states in the product model are related for agend a if the corresponding states and actions are related for that agent in *both* the Kripke model and the action model (thus agents can distinguish resulting states when they can distinguish either the previous state or the action).

In our simplified version of Among Us, every turn consists of all agents simulataneously taking a movement step, followed by them simultaneously observing the state of the new room they are in. We can neatly define this set of operations as the execution of a set of action models. We will define a movement action model MOV_a describing agent a taking one step, and an action model OBS for the effect of all observations. The relation between the Kripke models of two consecutive time steps is then as follows:

$$(\mathbb{M}_{t+1}, s_{t+1}) = (\mathbb{M}_t, s_t) \otimes (\mathsf{MOV}_a, \mathsf{mov}_{a,t}) \otimes \cdots \otimes (\mathsf{MOV}_m, \mathsf{mov}_{m,t}) \otimes (\mathsf{OBS}, \mathsf{obs}_t)$$

Single agent movement

The single agent movement action model $\mathsf{MOV_a}$ contains one state for every connected pair of rooms $(r, s) \in \rightsquigarrow$. It has precondition $\mathsf{pre}(r, s) = a_r$, and postcondition $\mathsf{post}(r, s)(\{a_s\}, \{\neg a_r\})$ (so the only change is the position of agent a). All states are distinguishable for a and indistinguishable for all other agents, so R_x includes only the reflexive relations if $x = a$ and is the full relation on the states of $\mathsf{MOV_a}$ for all other agents.

Observation

The observation action model contains one action for every possible distribution of agents over rooms, so the total number of states is $|\mathcal{R}|^{|\mathcal{A}|}$. For the state for a distribution $(a_i, b_j, \ldots)$, the precondition will be the conjunction $a_i \wedge b_j \wedge \ldots$, so each action picks out states in the Kripke model that match that distribution of agents. Two states in the observation action model are equivalent for agent a if the set of agents in the room where a is is the same. The postconditions of the observation action

model is empty. Thus, applying this action model leaves the size of the Kripke model unchanged but removes relations where agents can distinguish the corresponding states based on what they can see in the room they are in.

Implementation

The Kripke models and action models were implemented from scratch in Python using the `NetworkX` graph manipulation library[4]. As we only consider fairly simple sentences here, all logical sentences are evaluated using the built-in set operations of Python. To be able to handle more complex sentences (like higher-order knowledge) the Kripke models could be translated into `mlsolver` models like the other part of our project, but for the sake of simplicity we did not pursue this.

Optimization 1: Removing irrelevant states

Because we only evaluate sentences that contain a single K_a-operator in the distinguished state s and this operator only depends on the states that are reachable from s in one step, the ourcomes we report do not depend on the states in the Kripke-model that are not related to s for any agent. We can use this fact by removing all states that are not neighbours of s after every application of an action model. It is easy to see that this simplification is safe as long as the preconditions and postconditions in the action model don't use the K-operator (which is currently not possible in our simulation). This optimization can be enabled or disabled in the simulation, and was used during the generation of the results below.

Optimization 2: Replacing nodes with their bisimulation class

After the execution of a sequence of action models, one state will be created for each sequence of actions that is allowed by the preconditions. This means the number of states tends to grow as the simulation proceeds. Many of these states can be removed by the first optimization, but an orthagonal problem is the creation of states that are equivalent for the evaluation of all sentences. This type of equivalence is called a bisimulation, and we can deal with it by replacing all states in the Kripke-model with their bisimulation class. A full explanation is given in [3], in addition to an algorithm that can be used to perform the optimization. We implemented this algorithm and saw a simplification of the Kripke model in many cases. For example, when the simulation is started with two agent in the same room that are then moved apart and back together, both agents know the full state of the game, but the Kripke model will consist of a number of indistinguishable states referring to each other. The application of the bisimulation algorithm correctly reduces these to a single state. In our example below, this step was performed after the removal of irrelevant sates every time an action model was executed. In the implementation of the bisimulation algorithm an existing recipe for partitioning sets into equivalence classes based on a binary predicate was used [5] together with the graph algorithms built into `NetworX`.

Example

For the example, we consider a game where two agents a and b are moving around in a map of four room that are connected in a linear fashion ($r_1 \leftrightsquigarrow r_2 \leftrightsquigarrow r_3 \leftrightsquigarrow r_4$). In the first time step, both agents are in room 1. Every consecutive step, agent b moves one step to the right while agent a remains in room 1. Becuase the Kripke models are not very informative to look at directly, we have at each time step evaluated propositions of form $a_i, \neg a_i, K_{a'} a_i, K_{a'} \neg a_i$ in the distinguished state s (the real world) for every combination of $a \in \{a, b\}, a' \in \{a, b\}, i \in \{1, 2, 3, 4\}$. The code includes a function for

automatically evaluating such a set of propositions and constructing a LaTeXtable that can be used to easily check the state of the Kripke model.

The results are shown in the tables below, where all propositions in the *truth* row should be read as $(\mathbb{M}_t, s_t) \vDash p$, in the *agent a* row as $(\mathbb{M}_t, s_t) \vDash K_a p$ and in the *agent b* row as $(\mathbb{M}_t, s_t) \vDash K_b p$ (so each table cell shows propositions describing that room that are *true* or that are *known to be true by that agent*).

$t = 1$	Room 1	Room 2	Room 3	Room 4
Truth	a_1, b_1	$\neg b_2, \neg a_2$	$\neg b_3, \neg a_3$	$\neg a_4, \neg b_4$
Agent a	a_1, b_1	$\neg b_2, \neg a_2$	$\neg b_3, \neg a_3$	$\neg a_4, \neg b_4$
Agent b	a_1, b_1	$\neg b_2, \neg a_2$	$\neg b_3, \neg a_3$	$\neg a_4, \neg b_4$

Initially, agent a and b can see each other, so both know where the other is, and consequently the entire state of the game.

$t = 2$	Room 1	Room 2	Room 3	Room 4
Truth	$a_1, \neg b_1$	$b_2, \neg a_2$	$\neg b_3, \neg a_3$	$\neg a_4, \neg b_4$
Agent a	$a_1, \neg b_1$	$b_2, \neg a_2$	$\neg b_3, \neg a_3$	$\neg a_4, \neg b_4$
Agent b	$a_1, \neg b_1$	$b_2, \neg a_2$	$\neg b_3, \neg a_3$	$\neg a_4, \neg b_4$

At time 2, b has moved one step to the right (as can be read from the *truth* row). After one step of b, agent a knows that b must be in room 2 because he is no longer in room 1 and this is the only room that was reachable in one step. Similarly, agent b sees that a is not in room 2. Because the only possible actions for a were to move to room 2 or to stay in room 1, he knows a must still be in room 1. Consequently, both agents still know the valuation of all atomic propositions.

$t = 3$	Room 1	Room 2	Room 3	Room 4
Truth	$a_1, \neg b_1$	$\neg b_2, \neg a_2$	$b_3, \neg a_3$	$\neg a_4, \neg b_4$
Agent a	$a_1, \neg b_1$	$\neg a_2$	$\neg a_3$	$\neg a_4, \neg b_4$
Agent b	$\neg b_1$	$\neg b_2$	$b_3, \neg a_3$	$\neg a_4, \neg b_4$

After another time step, a has no way to know if b stayed in room 2 or moved to room 3. Consequently, all he knows about b is that he is not in room 1 (or he would be seen by a) or in room 4 (because that room cannot be reached from room 1 in two steps). Agent b also doesn't know exactly where a is anymore, but he does know a cannot be in room 3 or 4.

$t = 4$	Room 1	Room 2	Room 3	Room 4
Truth	$a_1, \neg b_1$	$\neg b_2, \neg a_2$	$\neg b_3, \neg a_3$	$b_4, \neg a_4$
Agent a	$a_1, \neg b_1$	$\neg a_2$	$\neg a_3$	$\neg a_4$
Agent b	$\neg b_1$	$\neg b_2$	$\neg b_3$	$b_4, \neg a_4$

Finally, agent b moves to the last room. Both agents now only kow that the other is not in the same room as they are.

References

[1] Mario Benevides and Isaque Lima. "Action Models with Postconditions". In: *Computacion y Sistemas* 21 (Sept. 2017), pp. 401–406. DOI: 10.13053/CyS-21-3-2808.

[2] H. V. Ditmarsch, W. Hoek, and Barteld P. Kooi. "Dynamic Epistemic Logic". In: 2007.

[3] Jan van Eijck. *Lecture notes Logica voor AI: Bisimulations*. https://staff.fnwi.uva.nl/d.j.n.vaneijck2/courses/lai0506/LAI11.pdf. 2006.

[4] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. "Exploring Network Structure, Dynamics, and Function using NetworkX". In: *Proceedings of the 7th Python in Science Conference*. Ed. by Gaël Varoquaux, Travis Vaught, and Jarrod Millman. Pasadena, CA USA, 2008, pp. 11–15.

[5] John Reid. *Equivalence partition (Python recipe)*. `https://code.activestate.com/recipes/499354-equivalence-partition/`. 2007.