DEL-based Epistemic Planning for Human-Robot Collaboration: Theory and Implementation —Supplementary Material—

Thomas Bolander, Lasse Dissing, Nicolai Herrmann

A Additional Proof Details

Proof of Proposition 1. $\Rightarrow$: Suppose Z is a bisimulation between $s = (W, R, L, W_d)$ and $t = (V, Q, K, V_d)$ and that $s \models \phi$. Then prove $t \models \phi$ which is equivalent to proving that $(V, Q, K, v) \models \phi$ for every $v \in V_d$. Choose $v \in V_d$ arbitrarily. By the condition [designated], there exists $w \in W_d$ with $(w, v) \in Z$. Now note that Z is a standard bisimulation between (W, R, L, w) and (V, Q, K, v) (it satisfies [atom], [back] and [forth] and $(w, v) \in Z$), and it hence follows that the two states are modally equivalent (Blackburn, de Rijke, and Venema 2001). From $s \models \phi$ and $w \in W_d$, we now immediately get $(W, R, L, w) \models \phi$, and hence $(V, Q, K, v) \models \phi$ by modal equivalence.

$\Leftarrow$: Suppose two states $s = (W, R, L, W_d)$ and $t = (V, Q, K, V_d)$ are modally equivalent. We need to find a bisimulation Z between them. From s we build a new model $s' = (W', R', L', w_d)$ with $W' = W \cup \{w_d\}$, $L(w_d) = \emptyset$, $R'_i = R_i$ for all $i \in \mathcal{A}$, and we add a fresh agent j with $R'_j = \{(w_d, w) \mid w \in W_d\}$. We construct t' from t in the same way, using v_d as the designated world and using the same fresh agent j. By induction on the structure of the formula, we can now prove s' and t' to be modally equivalent: All cases except $K_j\phi$ and $C\phi$ are trivial. The case of $K_j\phi$ holds due to the modal equivalence of s and t ($s' \models K_j\phi$ iff $s \models \phi$ iff $t \models \phi$ iff $t' \models K_j\phi$). The case of $C\phi$ follows from the cases of $K_i\phi$, $i \in \mathcal{A} \cup \{j\}$, since $C\phi$ is true iff $K_{i_1} \cdots K_{i_n}\phi$ is true for all $n \geq 0$ and agents $i_1, \ldots, i_n$.

We have now proved s' and t' to be modally equivalent. Since they are standard single-pointed epistemic models with actual worlds w_d and v_d, respectively, there must exist a bisimulation Z' between them with $(w_d, v_d) \in Z'$ (Blackburn, de Rijke, and Venema 2001) (note that our models are finite and hence trivially image-finite). Let $Z = Z' \cap (W \times W')$. Since Z' is a bisimulation relation on $(W \cup \{w_d\}) \times (V \cup \{v_d\})$, Z must satisfy the conditions [atom], [forth] and [back] for the states s and t. It now suffices to prove that Z also satisfies [designated]. So let $w \in W_d$. We need to find a $v \in V_d$ with $(w, v) \in Z$. By $w \in W_d$, we get $(w_d, w) \in R_j$, and since $(w_d, v_d) \in Z'$ and Z' is a bisimulation, from [forth] we get the existence of a world v with $(w, v) \in Z$ and $(v_d, v) \in R_j$. From $(v_d, v) \in R_j$ we can infer $v \in V_d$, as required. The other direction is symmetric $\square$

B Coin-flip Domain

The coin-flip domain is a small domain causing an exponential blowup unless full bisimulation contraction is applied to the states following each product update. The intuition behind the domain is as follows: Two agents take turns flipping a coin in such a way that the outcome is only visible to the acting agent, i.e., both outcomes are indistinguishable to the other agent. This causes the product update to create two copies of the current epistemic state and thus an exponential growth in the number of worlds. However since the coin can at any point only be in one of two configurations (either head or tails up), it is always possible to contract this down to a bisimilar state consisting of only two worlds. More formally, the domain is represented as follows: Let $\mathcal{A} = \{agt_1, agt_2\}$ be a set of agents. We then represent the current state of the coin as a proposition p and use propositions C_n, $n \in \mathbb{N}$ to count the number turns taken. The coin flip action is then defined by the epistemic action of Figure 1, where *turn*(i, C_n) is a special proposition which is true iff $(i = agt_1 \wedge n \bmod 2 = 0) \vee (i = agt_2 \wedge n \bmod 2 = 1)$, i.e., it ensures that the agents take turns flipping the coin. The domain can then be made into a planning problem by specifying an initial state s_0 consisting of a single world where C_0 holds and a goal C_n for some specific n equal to the desired search depth. This leads to a linear policy with the action sequence $agt_1{:}flip(C_0), agt_2{:}flip(C_1), agt_1{:}flip(C_2), \ldots$, which when applied to the initial state s_0 results in the sequence of states as shown in Figure 2.

As can be seen, the number of worlds in each state doubles for each applied action, thus resulting in an exponential blowup of the state size. However notice that in s_2 the worlds w_3 and w_4 are modally equivalent, i.e., the same formulas hold in the two worlds, and similarly for w_5 and w_6. We can therefore create the contracted state $\lfloor s_2 \rfloor$ of Figure 3 by applying ordered partition refinement on s_2 from Fig-

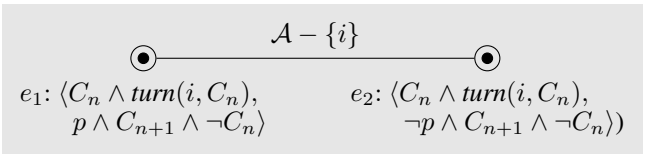

Figure 1: The coin flip action $i{:}flip(C_n)$ for agent $i \in \mathcal{A}$

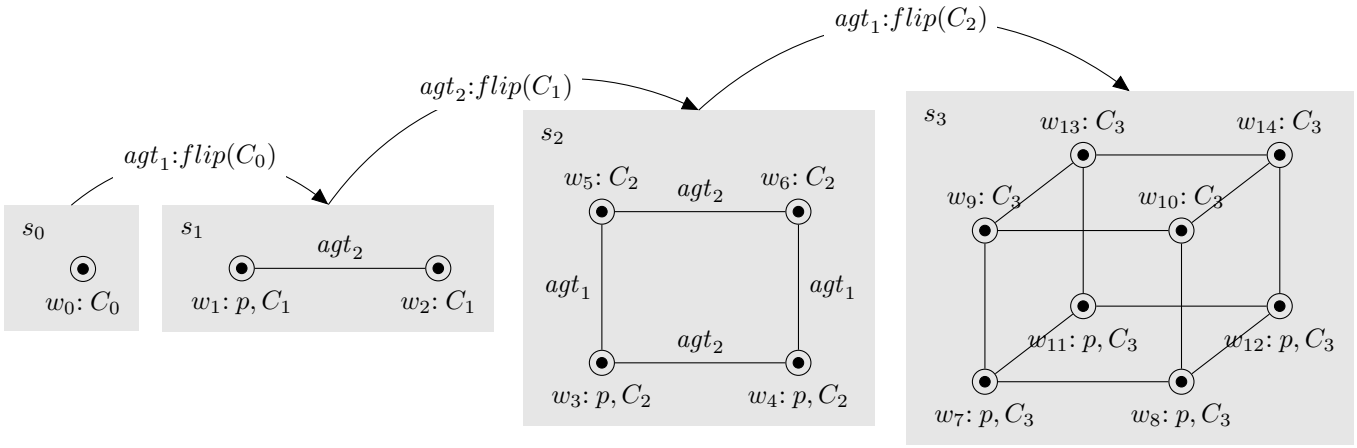

Figure 2: The evolution of the epistemic state after repeated coin flip actions without bisimulation contraction. The agent labels for the accessibility relations of state s_3 has been left out to reduce clutter, but they follow a pattern similar to the smaller states.

ure 2. Note that $\lfloor s_2 \rfloor$ only differs from s_1 by the names of the worlds, the agent which currently has uncertainty with regards to p and the counter value. Thus the number of worlds remains constant and the exponential blowup is avoided. Applying ordered partition refinement after each coin flip action therefore keeps the number of worlds at a constant number and therefore avoid the exponential blowup.

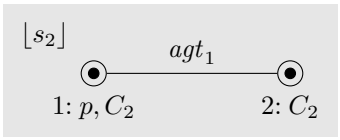

Figure 3: The result of contracting the state s_2 using ordered partition refinement.

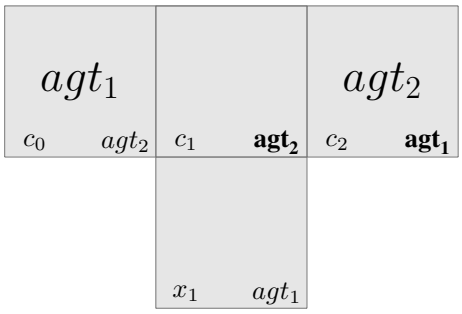

Figure 4: Initial state for MAPF/DU example. Left corner shows the cell name, right corner shows what agent may have the cell as goal (where bold font is the actual goal), and the center text is the initial position of the agents.

C MAPF/DU Example

To illustrate how the policies are created, we show the process for the concrete MAPF/DU example shown in the main report. Recall that MAPF/DU defines a grid world with agents having a start cell and a set of possible goal cells. Each agent knows their own goal cell, but has uncertainty regarding the goal cells of the other agents. The agents must navigate to their own goal cell and announce this, at which point they are no longer allowed to move. Figure 4 shows the initial state where agents 1 and 2 are located at the top left and top right respectively. For each cell, the lower left corner shows the name of the cell, and the lower right corner shows what agent may have the cell as goal (where bold font marks the actual goal). The initial state is formalised in Figure 5 where $g(agt_i, c_x)$ means agent agt_i has goal cell c_x, $at(agt_i, c_x)$ means agent agt_i is currently occupying cell c_x, $free(c_x)$ means cell c_x is not occupied, and $adj(c_x, c_y)$ means that cells c_x and c_y are adjacent to each other, i.e. it is possible to move from cell c_x to cell c_y. The domain has two actions. We will use the shorthand $move(agt_i, c_x, c_y)$ for the action $agt{:}ontic(at(agt_i, c_x) \wedge free(c_y) \wedge adj(c_x, c_y) \wedge \neg stopped(agt_i), \neg at(agt_i, c_x) \wedge \neg free(c_y) \wedge at(agt_i, c_y) \wedge free(c_x))$ which moves agent agt_i from c_x to c_y if the c_y is not occupied, and agt_i has not already stopped. Where $stopped(agt_i)$ means that the agent has reached its goal and is no longer allowed to move. Likewise we use the shorthand $stop(agt_i, c_x)$ for the action $agt{:}ontic(at(agt_i, c_x) \wedge g(agt_i, c_x) \wedge \neg stopped(agt_i), stopped(agt_i))$ which announces that the agent is at its goal and will stop moving. Note that the announcement action is technically an ontic action as it has both a precondition and a postcondition. However, it does not change any physical aspects of the world, but rather sets a support flag to more easily be able to check whether an agent has reached its goal. Actions are performed sequentially such that at every time step, one agent performs one action.

The part of the search graph which is used to extract a policy can be seen on Figure 6, where s_0 is given on Figure 5, s_{23} and s_{25} are goal states, and the descendants of s_{15} and s_{17} have been cut for brevity (the longest root to leaf path has 9 actions). All squares are and-nodes, and all circles are or-nodes. The children of an and-node are all the globals of the

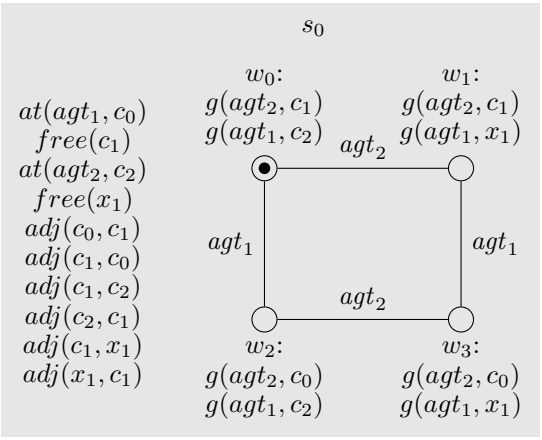

Figure 5: Initial state s_0 for MAPF/DU example. There is no uncertainty regarding the list of propositions on the left, and they have therefore been listed separately instead of at each world, for brevity.

state, and the child of an or-node is the product update with the chosen action. All or-nodes and their chosen action have been marked with a thick line, and they form the basis of the policy. The policy is extracted by, for each or-node, performing a perspective shift to the owner of the chosen action, then performing ordered partition refinement, and mapping them to their respective chosen action. Take node s_{13} for example, the policy is extended by $\lfloor s_{13}^{agt_1} \rfloor \rightarrow stop(agt_1, x_1)$

Note that the graph has been generated from the perspective of agent agt_1 which is why s_0 has two children. I.e. the initial perspective shift of s_0 causes both w_0 and w_2 to be designated, i.e. $W_d = \{w_0, w_2\}$. Interestingly, the first three actions of the policy are fixed, i.e. $a_0 = a_1$, $a_2 = a_3$, and $a_4 = a_5$. They specify that agt_1 moves to cell c_1 and x_1, where after agt_2 moves to cell c_1. We can realise that this is the only optimal solution since we are considering worst-case optimality. If instead agt_2 had made the first move, to cell c_1, it would block both of agt_1's goals. If agt_2 continued moving, to cell x_1, it would block one of agt_1's goals. Since our policies do not allow cycles, agt_2 would not be able to move back (reverse its actions). If the actual world is w_0, the worst case execution length would be 11, compared to the 9 when agt_1 moves first. After the first three actions (s_9 and s_{10}), if in worlds w_1 or w_3, agt_1 will announce that she has arrived at her goal x_1, else agt_2 will move out of the way to allow agt_1 to reach her goal at c_2.

References

Blackburn, P.; de Rijke, M.; and Venema, Y. 2001. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge, UK: Cambridge University Press.

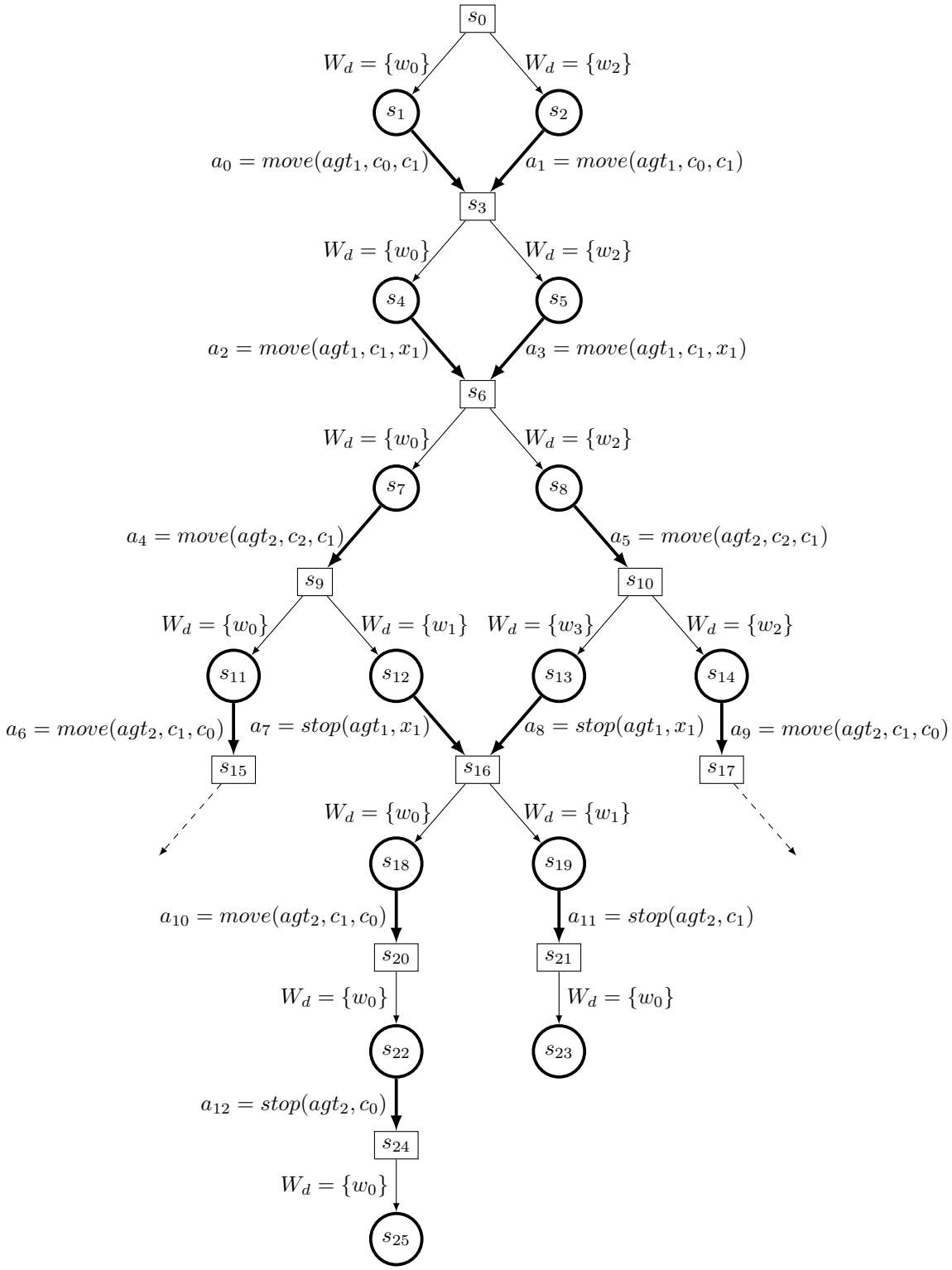

Figure 6: Partial search graph for MAPF/DU example, used for policy extraction. Circles are or-nodes and squares are and-nodes. Thick lines indicate state action pairs used for policy extraction.