**Dynamic Epistemic Logic**

B. Nebel, R. Mattmüller, T. Engesser
Winter Semester 2016/2017

University of Freiburg
Department of Computer Science

# Exercise Sheet P3

**Due: February 6th, 2017, 20:00**

We now want to extend our model checker to be a fully-fledged epistemic planner that can find standard and implicitly-coordinated (sequential) epistemic plans. For this, we first have to again slightly adapt the JSON input format specification. Actions receive another field `owner` specifying their *owner* agent. E.g., consider the following action library:

```
{"a1tella2whetherp": {"owner": 1,
                      "domain": ["e1", "e2"],
                      "indist": [[1, "e1"], [1, "e2"],
                                 [2, "e1"], [2, "e2"]],
                      "pre": {"e1": "K1p",
                              "e2": "K1~p"},
                      "eff": {"e1": "T",
                              "e2": "T"},
                      "designated": ["e1", "e2"]},
 "a2setq": {"owner": 2,
            "domain": ["e1"],
            "indist": [[1, "e1"], [2, "e1"]],
            "pre": {"e1": "p"},
            "eff": {"e1": "q"},
            "designated": ["e1"]}}
```

Instead of formulas that we want the model checker to check, we allow a *goal formula* to be specified via the field `goal` in the main JSON object. In the following example, the ellipsis again has to be replaced with the action library from above:

```
{"model": {"domain": ["w1", "w2", "w3"],
           "indist": [[1, "w1", "w2"], [2, "w2", "w3"]],
           "val": {"p": ["w1", "w2"], "q": []}},
 "designated": ["w1", "w2"],
 "actionlib": ...,
 "goal": "q"}
```

Depending on whether or not a goal is specified, your program should run in planning or model checking mode. For planning mode, the specification of one or more designated worlds is required. For planning mode, your program should also accept two optional command line arguments. The first argument specifies the type of the plan to be searched for (`standard|implicit`), defaulting to `standard`. The second argument specifies the agent that does the planning. If no such planning agent is given, the initial state of the planning task is simply specified by the fields `model` and `designated`. Otherwise, its *associated local state* for the planning agent is to be used. While your planner is supposed to only return subjectively optimal (= shortest) plans, actions of the planning agent are to be preferred over other agents' actions whenever possible. The resulting plan should then be outputted as list of actions, e.g. `["a1tella2whetherp", "a2setq"]`.

**Exercise P3.1** (Implementation, 8 points)

Extend your program with the functionality as described above. In particular, your planner should be able to find (subjectively) optimal standard and implicitly-coordinated (sequential) plans, and conform to the own-action-preference of the planning agent, given one is specified. In the case where no appropriate plan does exist, your program is not required to terminate.

**Exercise P3.2** (Examples, 6 points)

(a) Generate JSON specifications for the examples 5 and 6 from the paper `http://gki.informatik.uni-freiburg.de/papers/bolander-etal-dmap2016.pdf`. Generate for each of these examples one instance where the chess piece starts in the center field and one where the chess piece starts one field left or right of the center. Use them to check that in the implicitly-coordinated mode, your planner indeed returns subjectively optimal plans.

(b) Devise another interesting planning task for which the length of the shortest implicitly-coordinated plan is greater than the length of the shortest standard epistemic plan.

**Exercise P3.3** (Documentation, 4 points)

Make a short presentation (maximally three or four slides for a five minute talk) describing both your implementation (discussing its advantages/disadvantages) and your example. Be prepared to present it in the exercise sessions.