Logics Workbench: Multi-Agent Systems

Hans van Ditmarsch

1 Multi-agent systems

We present one of the theories describing gard games from this LWB multi-agent modelling archive. This theory 2/3 is identical to the file `2-3.lwb` in the archive. Below, we reprint that program. Immediately below a clause from the program, we extensively comment on the invisible choices that eventually led to it, in order to help you to formulate similar theories for similar problems.

```
load(kn);
```

We will prove our desiderata in multimodal $\mathbf{K}$ ($\mathbf{K_{(m)}}$) and not in multimodal $\mathbf{S5}$ ($\mathbf{S5_{(m)}}$). Of course we assume that our agents have $\mathbf{S5_{(m)}}$ reasoning abilities. There are two reasons for proving in $\mathbf{K_{(m)}}$. The first is, as was already mentioned before, that LWB does not have an $\mathbf{S5_{(m)}}$ module (although it does have e. g. an $\mathbf{S4_{(m)}}$ module, so there's no need to fall back 'all the way' upon `kn`). The second is 'parsimony' or 'applying Occam's razor': if we can prove what we want in a less powerful proof system, why not do so? We show that a 'stupid' $\mathbf{K_{(m)}}$ player 1 can already win the game, without having to be a smart introspective $\mathbf{S5_{(m)}}$ personality.

```
cards_are_unique :=
        [(white & ~white1 & ~white2) v
        (~white & white1 & ~white2) v
        (~white & ~white1 & white2),
        (green & ~green1 & ~green2) v
        (~green & green1 & ~green2) v
        (~green & ~green1 & green2),
        (red & ~red1 & ~red2) v
        (~red & red1 & ~red2) v
        (~red & ~red1 & red2)];
```

The theorem `cards_are_unique` states that a card can be held by only one person, or in other words, that 'card ownership' is a function from cards to players. In a predicate logic theory for this problem, we would have therefore have defined this as

the type of a function, and that would have been all we had to do. An actual dealing of cards over players then corresponds to a specific function of that type.

As we have chosen a propositional language to model our information, instead, we have to state all this explicitly: `white & ~white1 & ~white2` says that the white card can be assigned to one person only, etc. An actual dealing of cards then gives the actual assignment, in this program `red & green1 & white2` (see below).

```
has_cards01 :=
        red v green v white;
has_cards11 :=
        red1 v green1 v white1;
has_cards21 :=
        red2 v green2 v white2;
```

This states that there is at least one card on the table, at least one held by player 1, and at least one held by player 2. Of course, we actually want to say that there is exactly one on the table and held by both players. Exclusive disjunction is not an operator in LWB. Observe that `cards_are_unique` and the three `has_cards` theorems together also guarantee 'exactly one'.

Exercise 1 Prove the forementioned observation about 'exactly one' in propositional logic.

The agents do not know the entire system description but have limited access to it: they *only* know their own cards:

```
agent_access_to_world :=
        [red1 -> box1 red1, green1 -> box1 green1, white1 -> box1 white1,
        red2 -> box2 red2, green2 -> box2 green2, white2 -> box2 white2];
```

Again, we have been somewhat parsimonious here: if you do not have a card, you also know that you don't have it, e. g. `~red1 -> box1 ~red1`. Observe (by checking it in LWB) that this is *not* derivable in the present theory! Apparently we do not use it in order to derive that 1 knows the card on the table, as below.

```
cluedo := concat(
        [has_cards01, has_cards11, has_cards21],
        agent_access_to_world,
        cards_are_unique        );
```

The concatenation of all this information is the current theory `cluedo`, that is known to all.

```
situation := red & green1 & white2;
```

The current situation or actual dealing of cards, is that the red card is on the table, player 1 holds the green card, and player 2 holds the white card.

We now proceed with some proofs in the theory `cluedo`.

```
consistent(cluedo);
        # true
```

LWB replies to the request `consistent(cluedo)` with: `true`. The line `# true` in the program is just a comments line, to remind ourselves that we have actually tested the program and received that answer (# is the LWB comments character). Therefore the theory `cluedo` is consistent. This request/true combination means the same as the answer `false` to the request `provable(false, cluedo)` , expressing 'the theory `cluedo` is not inconsistent'.

Exercise 2 Consider the next expression:

```
provable(~(situation & dia1 red2), cluedo);
        # false
```

This expresses that it is *consistent* with the current situation (where red is on the table) and with what is known about the game, that player 1 can imagine that player 2 holds the red card. Show the relation between consistency and the LWB request `provable(~(situation & dia1 red2), cluedo)`.

Take a look at the next expression:

```
        # game: ask(1, red), nonshow(2), accuse(1, red), success
cluedo_1red := concat([box1 ~red2], cluedo);
provable(situation -> box1 red, cluedo_1red);
        # true
provable(situation -> box2 red, cluedo_1red);
        # false
```

The two `provable` commands represent the following information. Given player 1's request "do you have the red card", player 2 replies with "no". Player 1 now deduces that the red card is on the table, whereas player 2 cannot do so.

We use an informal notation for playing knowledge games. `ask(1, red)` means that player 1 asked for the red card. `nonshow(2)` means that player 2 answered this request by not being able to show a card. `accuse(1, red)` means that player 1 accuses red, which is, in a way, identical to publicizing his knowledge that the red card

is on the table. `success` is confirmation of that knowledge, interesting in an actual game situation where players can be mistaken about their accusations but uninteresting for our purposes, where players are perfect logicians.

Although we have introduced a 'new' theory `cluedo_1red` , it would have sufficed to reassign the name 'cluedo' to this extended theory. In other words, we could have defined as well: `cluedo := concat([box1 ~red2], cluedo)`.

```
        # game: ask(1, white), show(2, white), accuse(1, red), success
cluedo_1white := concat([box1 white2], cluedo);
provable(situation -> box1 red, cluedo_1white);
        # true
provable(situation -> box2 red, cluedo_1white);
        # false
```

A different game is where 1 asked for the white card instead of the red card. Now player 2 can show his card. Player 1, obviously, still wins.

```
        # player1 knows which cards she doesn't hold
provable(situation -> box1 ~red1, cluedo);
        # true
```

Although we haven't stated explicitly that players know the cards they do not hold, i.e. we haven't added theorems such as `~red1 -> box1 ~red1` to the theory `cluedo`, fortunately we can still derive that knowledge!

Exercise 3 How is $\Box_1 \neg \mathsf{red}_1$ proved axiomatically in the current theory? So, prove $\neg \mathsf{red}_1 \rightarrow \Box_1 \neg \mathsf{red}_1$ in propositional logic.