Input model algorithm

Generated model

The following model was generated for the example ontology from the paper.

```
Input parameters:
    RobotEnvironment.1.RobotCommand.cmd,
    RobotEnvironment.1.RobotCommand.1.WheelSurface.surface
    RobotEnvironment.1.RobotCommand.2.WheelSurface.surface
    RobotEnvironment.2.RobotCommand.cmd
    RobotEnvironment.2.RobotCommand.1.WheelSurface.surface
    RobotEnvironment.2.RobotCommand.2.WheelSurface.surface
    RobotEnvironment.3.RobotCommand.cmd
    RobotEnvironment.3.RobotCommand.1.WheelSurface.surface
    RobotEnvironment.3.RobotCommand.2.WheelSurface.surface

Domains:
    RobotEnvironment.3.RobotCommand.cmd=[left,straight,right,ε]
    RobotEnvironment.1.RobotCommand.cmd=[left,straight,right,ε]
    RobotEnvironment.2.RobotCommand.cmd=[left,straight,right,ε]
    RobotEnvironment.2.RobotCommand.2.WheelSurface.surface=[normal,slippery,ε]
    RobotEnvironment.1.RobotCommand.1.WheelSurface.surface=[normal,slippery,ε]
    RobotEnvironment.3.RobotCommand.1.WheelSurface.surface=[normal,slippery,ε]
    RobotEnvironment.3.RobotCommand.2.WheelSurface.surface=[normal,slippery,ε]
    RobotEnvironment.1.RobotCommand.2.WheelSurface.surface=[normal,slippery,ε]
    RobotEnvironment.2.RobotCommand.1.WheelSurface.surface=[normal,slippery,ε]

Constraints:
    RobotEnvironment.1.RobotCommand.1.WheelSurface.surface ≠ ε ∧ RobotEnvironment.1.RobotCommand.2.WheelSurface.surface ≠ ε
    RobotEnvironment.2.RobotCommand.1.WheelSurface.surface ≠ ε ∧ RobotEnvironment.2.RobotCommand.2.WheelSurface.surface ≠ ε
    RobotEnvironment.3.RobotCommand.1.WheelSurface.surface ≠ ε ∧ RobotEnvironment.3.RobotCommand.2.WheelSurface.surface ≠ ε
    RobotEnvironment.1.RobotCommand.cmd ≠ ε ∨ RobotEnvironment.2.RobotCommand.cmd ≠ ε ∨ RobotEnvironment.3.RobotCommand.cmd ≠ ε
    (RobotEnvironment.1.RobotCommand.1.WheelSurface.surface ≠ ε ∨ RobotEnvironment.2.RobotCommand.1.WheelSurface.surface ≠ ε ∨
    RobotEnvironment.3.RobotCommand.1.WheelSurface.surface ≠ ε)
    (RobotEnvironment.1.RobotCommand.2.WheelSurface.surface ≠ ε ∨ RobotEnvironment.2.RobotCommand.2.WheelSurface.surface ≠ ε ∨
    RobotEnvironment.3.RobotCommand.2.WheelSurface.surface ≠ ε)
```

Additional Constraints

The model is equivalent to the one provided in the paper, except for the two additional constraints that were created:

```
(RobotEnvironment.1.RobotCommand.1.WheelSurface.surface ≠ ε ∨ RobotEnvironment.2.RobotCommand.1.WheelSurface.surface ≠ ε ∨
RobotEnvironment.3.RobotCommand.1.WheelSurface.surface ≠ ε)
(RobotEnvironment.1.RobotCommand.2.WheelSurface.surface ≠ ε ∨ RobotEnvironment.2.RobotCommand.2.WheelSurface.surface ≠ ε ∨
RobotEnvironment.3.RobotCommand.2.WheelSurface.surface ≠ ε)
```

They have been created in the loop at lines 31 to 35 in the pseudocode due to the relation RobotEnvironment-RobotCommand with cardinality 1-3, next to the constraint concerning `RobotCommand.cmd`.

This happened because previously (in lines 28 to 30, but when processing the relation RobotCommand-WheelSurface), the two WheelSurface attributes were added to the RobotCommand entity.

First possible issue

The first possible issue I am not sure about is concerning the following constraints, that have been created to ensure that the 2-2 relation between RobotCommand and WheelSurface is correct:

```
RobotEnvironment.1.RobotCommand.1.WheelSurface.surface ≠ ε ∧ RobotEnvironment.1.RobotCommand.2.WheelSurface.surface ≠ ε
RobotEnvironment.2.RobotCommand.1.WheelSurface.surface ≠ ε ∧ RobotEnvironment.2.RobotCommand.2.WheelSurface.surface ≠ ε
RobotEnvironment.3.RobotCommand.1.WheelSurface.surface ≠ ε ∧ RobotEnvironment.3.RobotCommand.2.WheelSurface.surface ≠ ε
```

However, this now means that **all 3 x 2** Wheel surfaces need to be set (≠ ε), although RobotCommand is in a 1-3 relationship with RobotEnvironment, meaning that from my understanding it would be enough if 1 of the 3 constraints would be fulfilled.

I am not sure if this is irrelevant for further processing (if surfaces are set that are not required).

In case it is relevant, the next part shows what I thought about as a possible solution.
My biggest concern is that this means that there will be a lot less constraints in total, but they will be way more complex.

Possible approach in case this is relevant

To be completely correct my approach would be to have the single constraint

```
((RobotEnvironment.1.RobotCommand.1.WheelSurface.surface ≠ ε ∧ RobotEnvironment.1.RobotCommand.2.WheelSurface.surface ≠ ε) ∨
(RobotEnvironment.2.RobotCommand.1.WheelSurface.surface ≠ ε ∧ RobotEnvironment.2.RobotCommand.2.WheelSurface.surface ≠ ε) ∨
(RobotEnvironment.3.RobotCommand.1.WheelSurface.surface ≠ ε ∧ RobotEnvironment.3.RobotCommand.2.WheelSurface.surface ≠ ε))
```

instead of the 3 separate ons for the 1:3 relation in combination with the 3:3 relation.

To achieve this result dynamically generated by the algorithm, the lines 28:30 in the pseudocode have to be changed.

Do not add all constraints of all m InputModels, because only n of them need to be present (similar how attributes are handled in lines 31:35)

Instead, for each constraint set c1 to cm add only a subset of n (exactly line 33 in the algorithm, but for the constraints.)

With this process the example of the paper would end up with the result shown above for a 1:3 relation between RobotEnvironment and RobotCommand.

For a 2:3 relation between RobotEnvironment and RobotCommand it would generate the following (unfortunately rather complex) constraint:

```
(((RobotEnvironment.1.RobotCommand.1.WheelSurface.surface ≠ ε ∧ RobotEnvironment.1.RobotCommand.2.WheelSurface.surface ≠ ε) ∧
(RobotEnvironment.2.RobotCommand.1.WheelSurface.surface ≠ ε ∧ RobotEnvironment.2.RobotCommand.2.WheelSurface.surface ≠ ε)) ∨
((RobotEnvironment.1.RobotCommand.1.WheelSurface.surface ≠ ε ∧ RobotEnvironment.1.RobotCommand.2.WheelSurface.surface ≠ ε) ∧
(RobotEnvironment.3.RobotCommand.1.WheelSurface.surface ≠ ε ∧ RobotEnvironment.3.RobotCommand.2.WheelSurface.surface ≠ ε)) ∨
((RobotEnvironment.2.RobotCommand.1.WheelSurface.surface ≠ ε ∧ RobotEnvironment.2.RobotCommand.2.WheelSurface.surface ≠ ε) ∧
(RobotEnvironment.3.RobotCommand.1.WheelSurface.surface ≠ ε ∧ RobotEnvironment.3.RobotCommand.2.WheelSurface.surface ≠ ε)))
```

For a 3:3 relation the following would be generated:

```
((RobotEnvironment.1.RobotCommand.1.WheelSurface.surface ≠ ε ∧ RobotEnvironment.1.RobotCommand.2.WheelSurface.surface ≠ ε) ∧
(RobotEnvironment.2.RobotCommand.1.WheelSurface.surface ≠ ε ∧ RobotEnvironment.2.RobotCommand.2.WheelSurface.surface ≠ ε) ∧
(RobotEnvironment.3.RobotCommand.1.WheelSurface.surface ≠ ε ∧ RobotEnvironment.3.RobotCommand.2.WheelSurface.surface ≠ ε))
```

Second possible issue

For the example of the paper this issue occurs because of the two extra constraints added by the algorithm.

It concerns the constraints created for the 1:3 relation between RobotEnvironment and RobotCommand.

```
RobotEnvironment.1.RobotCommand.cmd ≠ ε ∨ RobotEnvironment.2.RobotCommand.cmd ≠ ε ∨ RobotEnvironment.3.RobotCommand.cmd ≠ ε
(RobotEnvironment.1.RobotCommand.1.WheelSurface.surface ≠ ε ∨ RobotEnvironment.2.RobotCommand.1.WheelSurface.surface ≠ ε ∨
RobotEnvironment.3.RobotCommand.1.WheelSurface.surface ≠ ε)
(RobotEnvironment.1.RobotCommand.2.WheelSurface.surface ≠ ε ∨ RobotEnvironment.2.RobotCommand.2.WheelSurface.surface ≠ ε ∨
RobotEnvironment.3.RobotCommand.2.WheelSurface.surface ≠ ε)
```

While these constraints should ensure that the minimum of 1 RobotCommand must exist, the attributes of RobotCommand are all handled independently.

This means that attributes could be present for different RobotCommands, although there isn't a single RobotCommand that contains all attributes.

Important Note: This is not only because of the 2 additional constraints. For example. if I add an extra attribute (cmd2) to RobotCommand (next to cmd), this are the resulting constraints for this relation.
So even if ignoring the 2 additional constraints (for easier showing), the issue would be that for example RobotCommand 1 could have a cmd but no cmd2, while another RobotCommand has a cmd2.

```
RobotEnvironment.1.RobotCommand.cmd ≠ ε ∨ RobotEnvironment.2.RobotCommand.cmd ≠ ε ∨ RobotEnvironment.3.RobotCommand.cmd ≠ ε
RobotEnvironment.1.RobotCommand.cmd2 ≠ ε ∨ RobotEnvironment.2.RobotCommand.cmd2 ≠ ε ∨ RobotEnvironment.3.RobotCommand.cmd2 ≠ ε
```

Possible approach in case this is relevant

The problem is because the loop in lines 31:35 goes through all attributes and adds the constraints for each attribute separately.

Instead of processing them separately all parts concering the same index would have to be ANDed resulting in a single constraint (again only using the example from above for easier showing):

```
((RobotEnvironment.1.RobotCommand.cmd ≠ ε ∧ RobotEnvironment.1.RobotCommand.cmd2 ≠ ε) ∨
(RobotEnvironment.2.RobotCommand.cmd ≠ ε ∧ RobotEnvironment.2.RobotCommand.cmd2 ≠ ε) ∨
(RobotEnvironment.3.RobotCommand.cmd ≠ ε ∧ RobotEnvironment.3.RobotCommand.cmd2 ≠ ε))
```

Implementation wise everything should be fixable, but the main concern is that it will probably always end up with a single extremely complex constraint.