Repair example

For a model with compounds a,b,c, suppose a has an inconsistent function.

To render it consistent, we will be using the results obtained from a single experiment performed in the real world system, presented in the observations below:

timestep 0	timestep 1	timestep 2	timestep 3		timestep n-1	timestep n
obsv(0,a,0).	obsv(1,a,1).	obsv(2,a,1).	obsv(3,a,0).	…	obsv(n-1,a,1).	obsv(n,a,1).
obsv(0,b,1).	obsv(1,b,1).	obsv(2,b,0).	obsv(3,b,0).		obsv(n-1,b,1).	obsv(n,b,0).
obsv(0,c,1).	obsv(1,c,0).	obsv(2,c,1).	obsv(3,c,1).		obsv(n-1,c,0).	obsv(n,c,1).

timestep 0	timestep 1	timestep 2	timestep 3		timestep n-1	timestep n
obsv(0,a,0).	obsv(1,a,1).	obsv(2,a,1).	obsv(3,a,0).	…	obsv(n-1,a,1).	obsv(n,a,1).
obsv(0,b,1).	obsv(1,b,1).	obsv(2,b,0).	obsv(3,b,0).		obsv(n-1,b,1).	obsv(n,b,0).
obsv(0,c,1).	obsv(1,c,0).	obsv(2,c,1).	obsv(3,c,1).		obsv(n-1,c,0).	obsv(n,c,1).

Our function must be able to replicate the observations, i.e. produce 0s when 0s are observed, and 1s when 1s are observed.

The function will be built using nodes, which are combinations of compounds. E.g. {ab}, {c}, {abc}, etc (the final function will look something like Fa = ab V c).

How are we ensuring that the 0s are being produced by our function?

- We look at the negative observations, and ensure that no node is ever positive when they happen

How are we ensuring that the 1s are being produced by our function?

- We look at the positive observations, and ensure that at least one node is positive when they happen

Q: How are we creating these nodes?

```
1 {node_regulator(1..MN,C) : maximum_node_number(MN), compound(C)}.
```

A: We let clingo decide which compounds belong to which nodes, using the node_regulator(N,C) predicate. N is the node ID, C is the compound inside that node.

Q: How is this not inefficient?

A: Because of the restrictions placed in the maximum number of nodes used, as well as the behavior that is expected from those nodes.

```
1 {node_regulator(1..MN,C) : maximum_node_number(MN), compound(C)}.
```

Q: How are we restricting the node behavior?

A:
```
:- timestep_positive_node(E,T-1,N), curated_observation(E,T,compound,0), T > 0.
```

If a negative observation exists in some experiment E and timestep T, we can have no positive nodes (i.e. output 1) in timestep T-1 (ensures negative observations are never violated)

```
:- unique_positive_observation(E,T), not node_positive_observation_coverage(_,E,T).
```

If a positive observation exists in some experiment E and timestep T, we must have some positive node that covers that observation, i.e. is positive in timestep T-1 (ensures positive observations are never violated)

```
1 {node_regulator(1..MN,C) : maximum_node_number(MN), compound(C)}.
```

timestep 0	timestep 1	timestep 2	timestep 3		timestep n-1	timestep n
obsv(0,a,0).	obsv(1,a,1).	obsv(2,a,1).	obsv(3,a,0).		obsv(n-1,a,1).	obsv(n,a,1).
obsv(0,b,1).	obsv(1,b,1).	obsv(2,b,0).	obsv(3,b,0).	…	obsv(n-1,b,1).	obsv(n,b,0).
obsv(0,c,1).	obsv(1,c,0).	obsv(2,c,1).	obsv(3,c,1).		obsv(n-1,c,0).	obsv(n,c,1).

Q: How does this work in practice?
A: Clingo generates some nodes. Due to the integrity constraints, we know that:

1. All nodes will be negative (i.e. output 0) before a negative observation – so, all negative observations are replicated.
2. At least some node will be positive before a positive observation – ensuring that all positive observation are replicated, too.

For example, from timestep 0 to 1, a becomes active. Considering that for this example we are dealing with activators only, the only nodes that would be able to uphold condition 2 would be {b}, {c} and {bc}. Yet, one of them would violate condition 1:

- At time step 3, we know that a should be inactive. But, if we consider {c} to be one of the function nodes, that observation would be violated. This is because, at time step 2, c is active, therefore causing that node {c} be active, and consequently that a be active in the next time step. So, only {b}, and {bc} could be a part of the final solution.

These two constraints ensure that the nodes obtained will respect all observations.

```
1 {node_regulator(1..MN,C) : maximum_node_number(MN), compound(C)}.
```

Q: How many nodes will we need?

A: Our strategy focuses on using positive observations to determine this.

In the universe of valid nodes, none of them will be positive before a negative observation. So, negative observations are “guaranteed”, no matter which nodes we select for our solution.

But, the same is not true for positive observations. For any given positive observation, only some nodes will be positive before that observation, ensuring that we are able to replicate it.

What this means is that, in our universe, nodes that don’t help in replicating any positive observations are useless to us. The only “essential” nodes are those that can replicate positive observations, as they supersede those that cannot.

Thus, at most, we will need one node per positive observation, to ensure that our function will cover all observations. In reality, because some nodes may cover more than one positive observation, we may end up using even less nodes.

```
1 {node_regulator(1..MN,C) : maximum_node_number(MN), compound(C)}.
```

In truth, we don't even need as many nodes as positive observations. We only need as many as unique positive observations.

timestep 0	timestep 1	timestep 2	timestep 3	…	timestep n-1	timestep n
obsv(0,a,0).	obsv(1,a,1).	obsv(2,a,1).	obsv(3,a,0).	…	obsv(n-1,a,1).	obsv(n,a,1).
obsv(0,b,1).	obsv(1,b,1).	obsv(2,b,0).	obsv(3,b,0).		obsv(n-1,b,1).	obsv(n,b,0).
obsv(0,c,1).	obsv(1,c,0).	obsv(2,c,1).	obsv(3,c,1).		obsv(n-1,c,0).	obsv(n,c,1).

If we look at timesteps 2 and n, both observations describe the same model state. Since they also come from observations that describe the same state (1 and n-1), if we can find a node to replicate observation at time step 2, that same node can replicate the observation at time step n (because they have the same inputs).

```
maximum_node_number(POSITIVE_OBSERVATIONS_COUNT) :- positive_observations,
 POSITIVE_OBSERVATIONS_COUNT = #count{E,T : unique_positive_observation(E,T)}.
```

Note: while timestep n is a duplicate of timestep 2, we do not know if n-1 is a duplicate of 1, since n-1's previous state is not known.

timestep 0	timestep 1	timestep 2	timestep 3		timestep n-1	timestep n
obsv(0,a,0).	obsv(1,a,1).	obsv(2,a,1).	obsv(3,a,0).	…	obsv(n-1,a,1).	obsv(n,a,1).
obsv(0,b,1).	obsv(1,b,1).	obsv(2,b,0).	obsv(3,b,0).		obsv(n-1,b,1).	obsv(n,b,0).
obsv(0,c,1).	obsv(1,c,0).	obsv(2,c,1).	obsv(3,c,1).		obsv(n-1,c,0).	obsv(n,c,1).

So what would be the chosen function for this example? (considering that, when they are present as regulators in a node, a,b,c will be activators)

Our possible nodes would be:

obsv(1,a,1): {b}, {bc}

obsv(2,a,1): {b}, {ab}

obsv(n-1,a,1) - cannot determine, but let's consider: {b}

Since we will be minimizing the number of nodes, the final function would be Fa = b

Note: if we consider the hypothetical scenario where our observations have the following possible nodes:

obsv(1,a,1): {a}

obsv(2,a,1): {b}, {bc}

obsv(n-1,a,1): {a}, {b}

Then, in this scenario, the final function could be one of these two:

- Fa = a V bc
- Fa = a V b

In this case, whichever function had more regulators in common with the original inconsistent function would be chosen.