**Agent 4 Design**

The basic principle upon which the design of Agent 4 is based is the Minesweeper example that was done during lecture in which we treated unconfirmed cells like variables and the mines sensed to be neighboring that cell as the RHS of the equation. Only here, they’re not mines, they’re blocks. So, where Agent 3 can look at surrounding neighbors and infer (possibly propagate outward from there), Agent 4 attempts to synthesize information and observations across multiple cells and build new “equations” that may lead to something new that can be inferred.

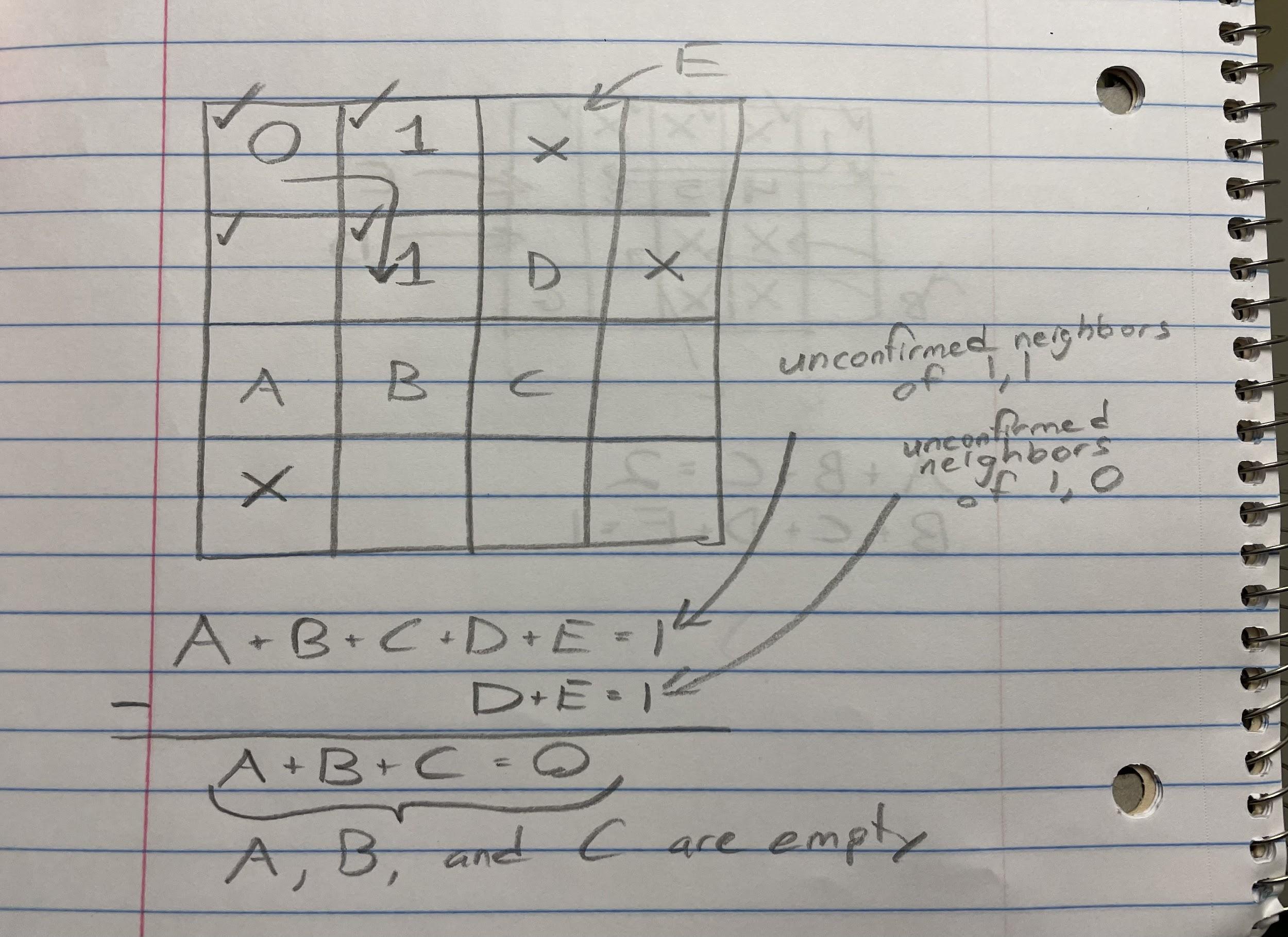
There were five main inference rules that we decided were deterministic about the status of 1 or more cells with an equation.

First of all, if an equation has only one term, then the block status of the cell is easily found. For example, if you had -2A = -2, you know for sure that A = 1, and that would mean A is blocked.

All remaining inference rules make use of counting up the sum of the factors (i.e. -2A + 2B + 1C would have a sum of factors equal to 1 because -2 + 2 + 1 = 1), the number of positive factors (in the prior example, that would be 2), and the sum of the positive factors (which would’ve been 3 in the prior example.

* + The second inference rule we used was very niche but still quite effective. If the sum of the factors = 0 and there are 2 factors overall and the total number of blocks is equal to the absolute value of both terms’ factors, then we know that one is blocked and one is empty. For example, if we had the equation 1A + -1B = 1, then we can see that 1A = 1 + 1B. Since A can only be zero or one, we can conclude that A must be 1 (blocked) and B must be 0 (empty).
  + The third inference rule we used was if the sum of the positive factors was equal to the total number of blocks, then we can conclude that every term with a positive factor must be blocked and every term with a negative factor must be empty. This follows from the following proposition while considering the equation 1A + 1B + -1C = 2: say that any one of these terms with a positive factor were actually empty and evaluated to zero. The equation is no longer satisfiable. We cannot possibly reach the total number of blocks without this term being equal to the factor \* 1. On the other hand, consider if a cell with a negative factor was actually blocked. Again, this is no longer satisfiable because we can no longer reach the magnitude of the RHS of the equation.
  + The fourth inference rule is essentially the same idea but switching roles. Take -1A + -1B + 1C = -2. It follows that every cell with a negative factor must be blocked and every cell with a positive factor must be empty (refer to above).
  + The fifth and final inference rule we decided to employ was this idea: if all terms were either positive or negative and the RHS (total number of blocks between them) was equal to 0, then we know that every cell in this equation must be empty. Imagine 1A + 2B + 3C + 1D = 0. If any one of these terms evaluated to anything but zero, there would be no way to reduce the RHS back down to zero. Thus, every term must be zero and every cell must be empty.

Let’s see this in practice. Consider the following example:



The check marks indicate the cells that have been confirmed by the agent, and the numbers in the middle of the cells explored so far indicate the number of blocks sensed around that cell. We’ve assigned variables to unconfirmed cells for simplicity’s sake, and let’s say we try to derive a new equation from cells 1,1, and 1,0. On their own, they each know they have a block somewhere around them, but they don’t necessarily know where it is. However, if we compile their information together, then we find that the agent can determine at least some information about the neighboring cells. Mainly, we determine that A, B and C are all empty thanks to the fifth rule described above.

So, you can see that Agent 4 just builds on Agent 3, taking in the same information and also having a similar inference phase. Agent 4 just extends that idea of inference into a database of extra facts as well. You can actually think of Agent 4 as just an extension of Agent 3.

The biggest challenge in designing this particular approach is having a computer decide which equations were best to pair together and match up to reduce the amount of terms we are looking at. The simpler the equation, the less variables there are, the better. But often as humans, we’re making judgements that are very complex and not necessarily intuitive algorithmically. With that in mind, if you don’t synthesize / reduce your equations in just the right way, the agent may not be inferring every possible thing that it could possibly infer for the maze at a particular state. But how much time would it take for the agent to work out exactly what equations to match up? And would it be worth it? That’s a fair question. So, when implementing this was something we had to take into account.

**Agent 4 Implementation**

Information Representation

For this implementation, this is essentially using Agent 3 as a framework and building on from that. The biggest addition (refer to the appendix) is the Association class. This is how we devised to store these so-called “equations” in our program. Each Association instance had an integer called totalBlocks which acted as the RHS of an equation (also known as the total blocks sensed between all terms) and an array of type Association.Unit that stored the terms of the LHS. The Unit inner class defined an object type that stored a cell (CellInfo type) and an integer that simulates the factor of a given term (i.e. with 2A, 2 is the factor).

As opposed to Agent 3, we’ve now added an Association field to each cell that stores all of its unconfirmed neighbors. This is set up when the maze is built and then each subsequent “discovery” allows us to update the necessary neighbors of this change in status. More specifically, if a cell is stored in an Association instance, its status has not yet been confirmed to us, either by inference or observation. Whenever a cell’s status is confirmed, we update our knowledge base immediately so that we’re not trying to infer things that have already been determined (and also possibly shielding us from more information).

Our Agent 4 has a field that is essentially a database of extra facts that we’ve been able to determine from the cells in the maze. This is an array of Association objects, and it is also updated upon each subsequent discovery. The agent looks to this database at certain times throughout the traversal of the maze (described in Workflow below).

Workflow

The workflow is almost identical to Agent 3 except for a few key distinctions that come about during the inference phase that we’ve improved upon / extended. Once we’re done inferring everything we can about the cell that the agent is currently in and its neighbors (Agent 3 implementation), we look to the next cell in our path. Treating this almost like the expert systems idea from class, we try to narrow our focus in this extension of the inference phase so we don’t get stuck looking at the entire maze, potentially wasting time (refer to “Computational Issues” below). It directs our attention and computational power to something that truly matters for us, the next step in our path.

The first thing we do is we try to synthesize a new Association based on the information provided by the current cell and its parent cell (the next cell back in the path). The reason we are doing this is because Association objects require a “number of neighbors sensed to be blocked” value to be a valid equation, so we can’t look to any neighbors we haven’t already visited, and the safest bet is the cell we just came from. The other added bonus of synthesizing the information from these two cells in particular is they will have shared neighbors, so they’re bound to have some terms cancel out. The hope is that they provide concrete information as a result of this, but that’s an issue for the processing part (paragraph below). The last note I’ll make is that if the equation actually has all terms cancel, then it’s not very helpful to us. This situation would occur if the only unconfirmed neighbors were the ones that the two cells shared.

After this synthesis is done and there has been a new Association (potentially) added to the knowledge base, we feed this next cell into the database of extra facts we’ve compiled over time, we check for any and all equations that may contain this cell, and we attempt to process these facts. This is essentially inference part 2. This is where we take all of those inference rules from the design plan above and we check for them here. If one of them gets a hit and we can confirm some of the cells in the equation, we send those back to the Agent and that tells it to infer (part 1 / Agent 3 stuff) on those cells again, just in case we learn even more. And of course, our knowledge base is updated as soon as possible. A small change from a method that was in Agent 3 was that checkSurroundings() also updates the Association object attached to the cell and gets rid of any cells that have since been confirmed.