TDT4136 Intro to AI - Final Exam Fall 2022 - Solution

Last version updated on 2023-01-03

Overview

- There are multiple correct answers in many of the questions, the solutions presented here show only one possible answer.
- Some tasks are asking for partial solution to the problem. For the sake of completeness, solutions presented here solve the complete problems, which is not expected from students.
- For the sake of correctness, truth tables are computed automatically.

Task 1: Theoretical questions

1.1) Why might we want to use a factored state representation instead of an atomic state representation?

If we use an atomic representation, we only know that two states are distinct. By using a factored representation we can get features of the state and use them to assess, compare, generalize states when using heuristics or evaluation functions. Factored representations also allow better handling of uncertainty and incomplete information, assigning it to specific features of the state, e.g. position and orientation of an object.

1.2) What is the benefit of having a model in a model-base agent?

By having a model, we have a state in order to better keep track of the uncertainty of the unobserved part of the environment. It allows us to make predictions about the future state of the environment, which allows us to select actions that achieve its goals.

1.3) Why is consistency a desired property of a heuristic?

Because when used in A, nodes are expanded only once - simplifying the algorithm and making it more efficient. It guarantees that once node is expanded, the cost of reaching it is the lowest possible. Otherwise, there might be another path to this node with lower cost so that we might need to expand it again to reach an optimal solution. It is also an indication that our heuristic is calculating appropriately because it's admissible, and hence required to reach an optimal solution.*

1.4) What is the difference between incremental and local search?

Incremental search algorithms start with a partial solution and then gradually build on that solution until a complete solution is found.

1.5) Why do we have mutations in genetic algorithms?

To explore the neighbourhood of some solutions by making small alterations to their genotype we expect improvements in their fitness, as opposed to crossover which is a bigger alteration to exploit the "good" solutions.

1.6) How can we apply minimax to a 3-player game?

Use vectors of values instead of single values for each node. For terminal states, this vector gives utility of the state from each player's perspective. For nonterminal state, the value is the highest values of the consequent states from the perspective of the player making a choice - playing its turn.

1.7) Why is Monte Carlo tree search less sensitive to evaluation function errors than alpha-beta search?

Monte Carlo runs multiple simulations to evaluate the potential value of each move, rather than relying on a pre-defined evaluation function.

1.8) Explain why the ignore-delete-list heuristic is an admissible heuristic for planning

By removing the constraints and ignoring the actions that undo progress, we can generate a heuristic that "can only get better" (and hence is admissible), so the length of the plan is the best way to know how good is it as any search method would solve it without the need for backtracking.

1.9) Entailment ($\models$) is not part of the logical representation language, nor a connective in any logic, Then, what is entailment? Entailment can be implemented by resolution/refutation or forward/backward chaining in propositional logic. Which inference rule is used in resolution refutation and which one is used in forward/backward chaining?

Entailment usually means: what is on the right can be derived from the left by using elements from the same logic system. It's not only "derivation" but also carries a semantic meaning, since what is on the right is a tautology given what is on the left. It can be obtained through model checking, inference rules or any other logic proof method. In resolution we use the resolution inference rule, disjunctive syllogism or unit resolution. In forward/backward chaining we use modus ponens.

1.10) What is the main disadvantage of propositional logic? What are the four new elements introduced in the syntax of first order logic to address this issue?

The main disadvantage of propositional logic is its limited expressive power, as it can only address what is true or not about a single object. It is hard or impossible to encode the statements about similar objects, relations, and groups of objects. An extension to this was First Order Logic (FOL), which introduces variables, quantifiers, predicates and functions, making it easy to express truth values of categories and classes, as well as relationships between such objects.

Task 2: Intelligent Agents - Grocery Store Robot

Grocery store robots can navigate autonomously through the store to scan shelves and report on inventory. In other words, it reports the quantity of products currently available on the shelves. The robot is equipped with cameras, an ultrasonic sensor (measures distances to obstacles) and a collision sensor.

2.1) Specify:

- performance measure
- environment properties
- actuators and sensors

Explain your specification, including the answer for the following questions:

- Why this performance measure makes sense for this environment?
- With respect to each of the properties, why this environment can be characterized in this way?

Solution.

The **PEAS** description could be similar to this:

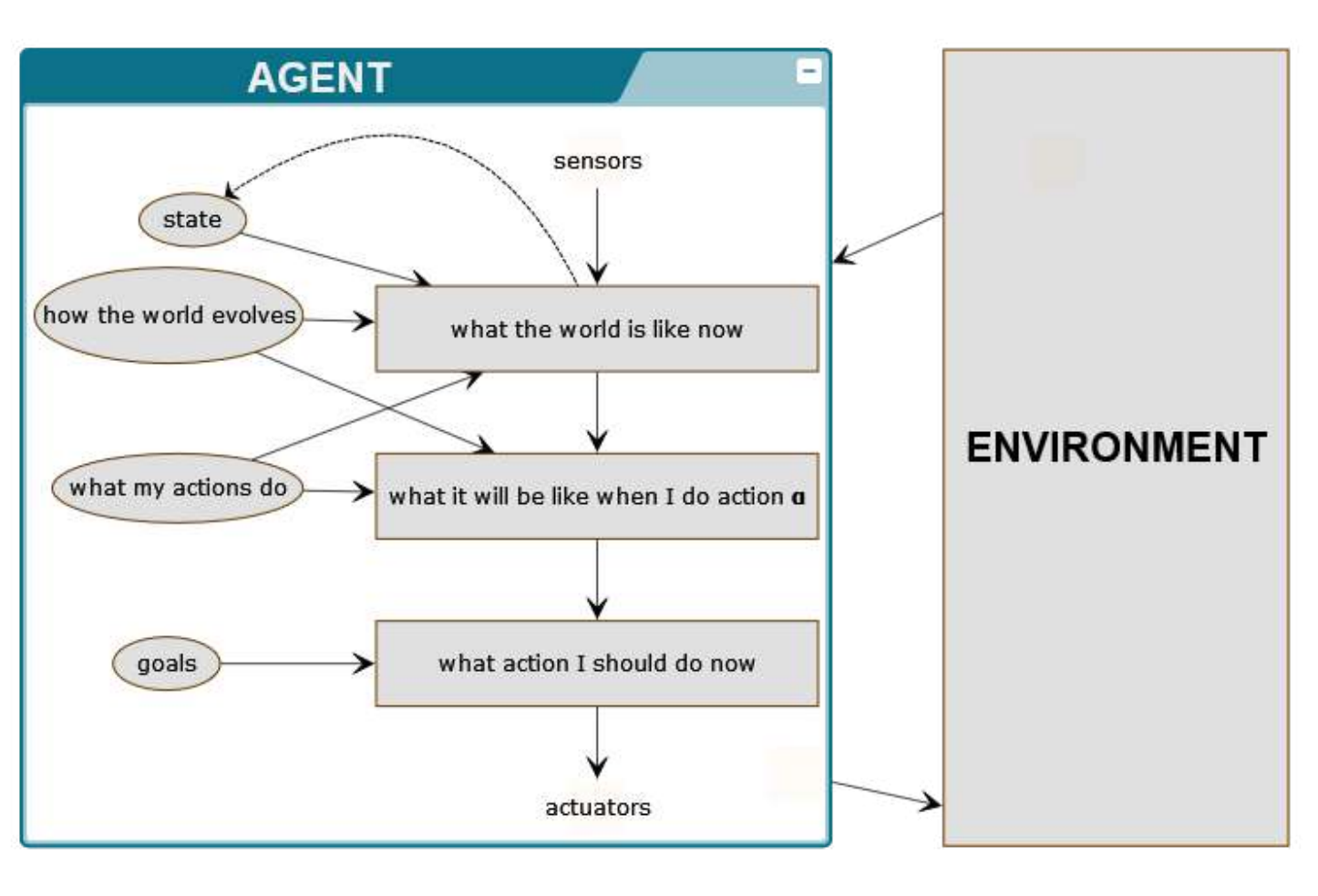
- **P**erformance measure: *Safe, fast, maximize scanned products, minimize collisions on other aisle users.*
- **E**nvironment properties: *Partially observable, Single agent, Deterministic, Sequential, Dynamic, Continuous*
- **A**ctuators and **S**ensors: *Camera, scanner, ultrasonic sensor, collision sensor, display, wheels.*

We want the robot to scan all products while not hitting anyone nor anything in the process. The agent needs to be aware of its surroundings and identify aisles where it can move through, and objects and people that should avoid. It should also distinguish between the types of products available so that it can notify/classify when a product is running out. The cameras, scanners and sensors will be used to attain these goals as well as for navigation. A display might be a good idea for debugging and configuration. Wheels are necessary for movement.

The environment is partially observable since there are limitations on what the sensors can perceive. It is single agent, as the customers and rest of the staff can be modelled as part of the environment and do not affect the decisions the robot should make. It is Deterministic as there are no stochasticity in the actions or state transitions. It is sequential, since the actions of the future are dependent on decisions made in the past. Lastly, it is continuous as it is intended for use in the real world, with continuous mobility and perception.

2.2) What agent type will work best for this environment? Explain why. Make a visualization showing the main components for this type of agent. Briefly describe each component and interaction between them.

Solution. A store robot could have the following architecture, which is that of a **model-based agent**:



The **model-based** architecture was chosen for the agent because of the partial observability of the environment (as the agent can only see a few meters away, and not behind aisles and shelves.).

The **sensors** feed the agent on the **state of the environment** (which is dependent on **its actions** and on **how the environment evolves**). Then, the agent can **update** the model of the world (**what it will be after acting**) depending on **what its actions do**. The agent will finally choose an action inspired by **its goals**, which will tell the **actuators** what action to perform and how to affect the environment.

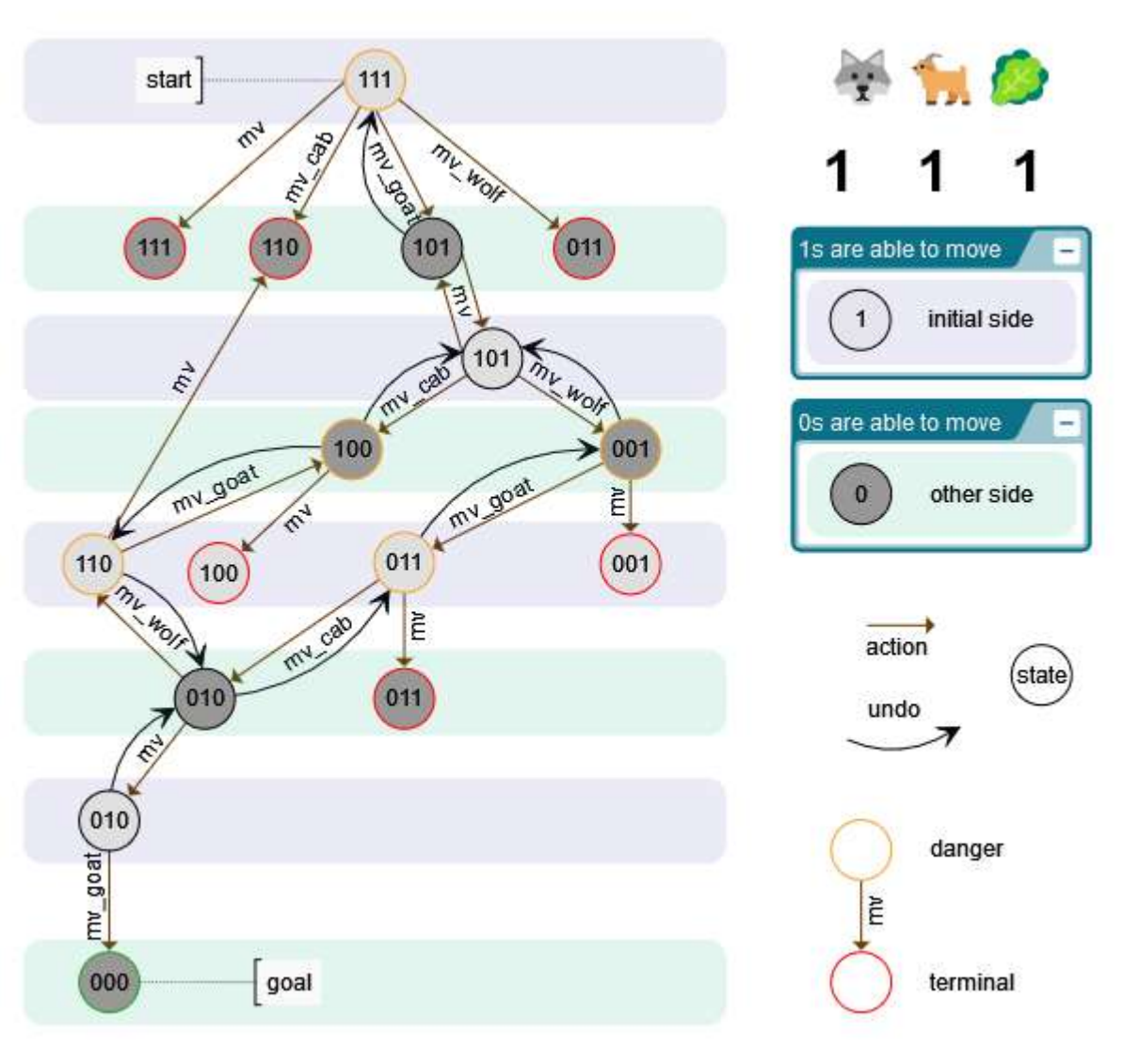
Task 3: Solving problems by searching: Wolf, goat and cabbage

A farmer went to the market and purchased a wolf, a goat, and a cabbage. On his way home, the farmer came to the bank of a river and rented a boat. But to cross the river by boat, the farmer could carry only himself and a single one of his purchases: the wolf, the goat, or the cabbage. If left unattended together, the wolf would eat the goat, or the goat would eat the cabbage. The farmer's challenge is to carry himself and his purchases to the far bank of the river, leaving each purchase intact.

3.1 Draw part of a state-space graph for this problem, with nodes that are 3 or less edges away from the start node. Explain what nodes and edges represent. Mark start and end nodes.

Solution.

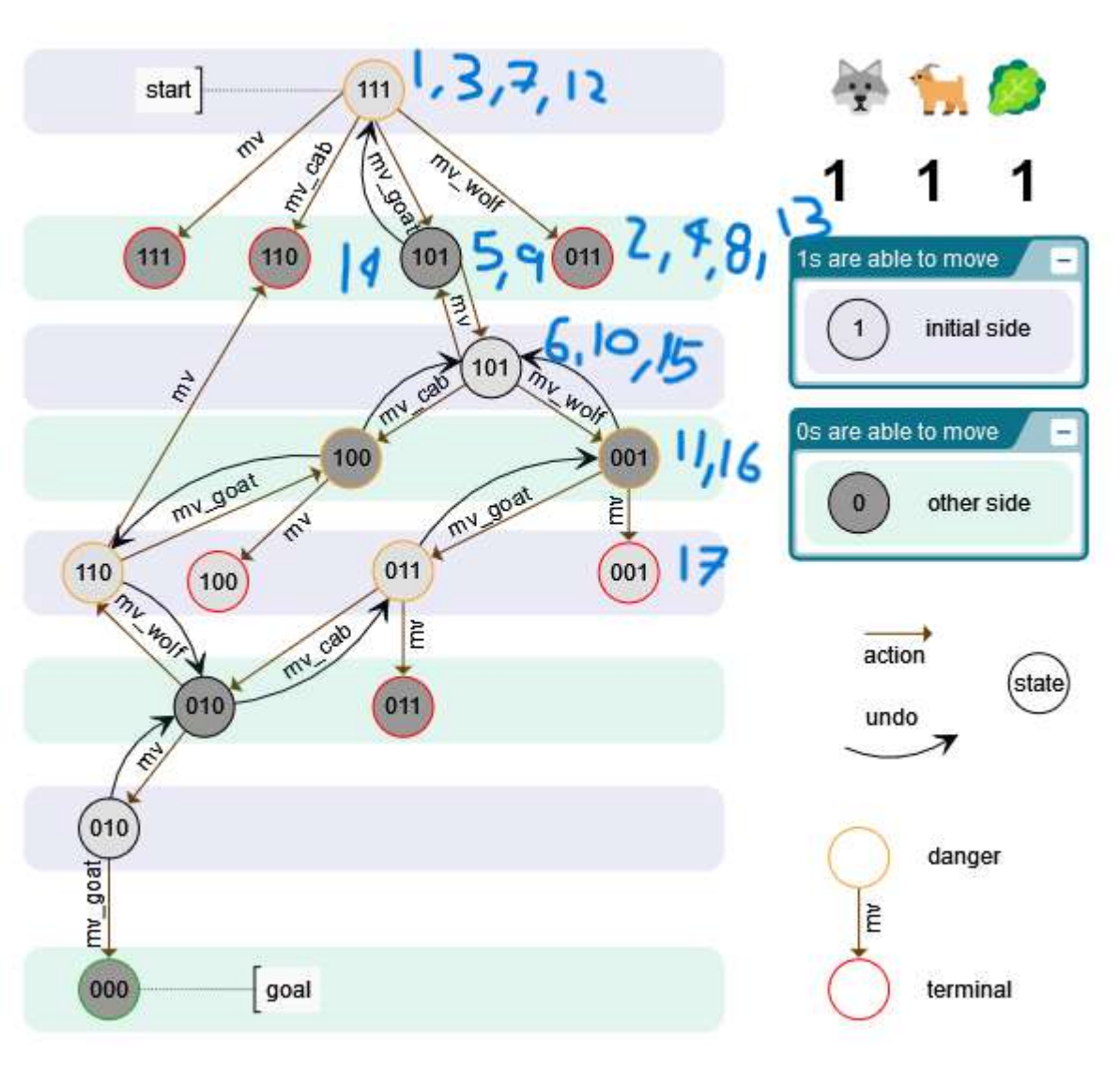
Here is a possible complete graph. Students only need to show the first 4 levels.



3.2) Apply iterative deepening search with depth limit of 3. Enumerate nodes in the order they are expanded.

Solution. Nodes are marked depending if they are on the (**i**)nitial or the (**o**)ther side. We explore always the smaller number first (alphabetical order):

- depth=0
  1. check 111-i at level=0, not goal, add children to stack: [011-o, 101-o, 110-o, 111-o]. Choose 011-o.
  2. check 011-o at level=1, cutoff: increase depth.
- depth=1
  1. check 111-i at level=0, not goal, add children to stack: [011-o, 101-o, 110-o, 111-o]. Choose 011-o.
  2. check 011-o at level=1, not goal, no children, go back, choose 101-o.
  3. check 101-o at level=1, not goal, add children to stack: [101-i]. Choose 101-i.
  4. check 101-i at level=2, cutoff: increase depth.
- depth=2
  1. check 111-i at level=0, not goal, add children to stack: [011-o, 101-o, 110-o, 111-o]. Choose 011-o.
  2. check 011-o at level=1, not goal, no children, go back, choose 101-o.
  3. check 101-o at level=1, not goal, add children to stack: [101-i]. Choose 101-i.
  4. check 101-i at level=2, not goal, add children to stack: [001-o, 100-o]. Choose 001-o.
  5. check 001-o at level=3, cutoff: increase depth.
- depth=3
  1. check 111-i at level=0, not goal, add children to stack: [011-o, 101-o, 110-o, 111-o]. Choose 011-o.
  2. check 011-o at level=1, not goal, no children, go back, choose 101-o.
  3. check 101-o at level=1, not goal, add children to stack: [101-i]. Choose 101-i.
  4. check 101-i at level=2, not goal, add children to stack: [001-o, 100-o]. Choose 001-o.
  5. check 001-o at level=3, not goal, add children to stack: [001-i, 011-i]. Choose 001-i.
  6. check 001-i, at level=4, cutoff: that's it.



3.3) Design an admissible heuristic for this problem. Describe the heuristic and explain why it is admissible.

Solution. Again, the solution depends on what they wrote and the convention they are using. Following what we presented, a nice admissible heuristic is to **count the number of zeros**, as each zero means that one of the purchases is already on the other side.

It is an admissible heuristic because it never overestimates the cost of getting to the goal. It is also consistent due to actions closer to the goal having more 0s than actions closer to the start.

Task 4: Constraint Satisfaction Problems - Four digit number

In this problem, you are trying to find a four-digit number satisfying the following conditions:

1. The number is odd
2. The number only contains the digits 1, 2, 3, 4 and 5
3. Each digit (except for the leftmost) is strictly larger than the digit to its left

4.1) Formulate this problem as a CSP: 4.1.1) Visualize this problem as a constraint graph. Explain what nodes and edges represent. 4.1.2) Apply unary constraints (condition 1) and 3) and show domains.

Solution. A CSP can be defined using a triple like $CSP = (X, D, C)$, where

- Variables: $X = \{x_1, x_2, x_3, x_4\}$
- Domains: $D = \{d_i \mid d_i = \{1:5\}, i \in [1:4]\}$
- Constraints $C = \{x_2 > x_1, x_3 > x_2, x_4 > x_3\} \cup \{(x_1 \# x_2 \# x_3 \# x_4 \mod 2 = 1)\}$

Notes:

- *# is the concatenate operator*
- *The constraint «... is odd» can also be expressed in many other ways.*

Each node represents a variable $x_i \in X$ while each edge represents a binary constraint $c \in C$. Arrows are important since the "greater than" relation is not bidirectional.

4.2) Solve this problem using backtracking with the following heuristics:

- Use **Minimum remaining values** for variable selection
- Use **Least constraining value** for value selection

Show each step with updated domains of the variables:

Solution.

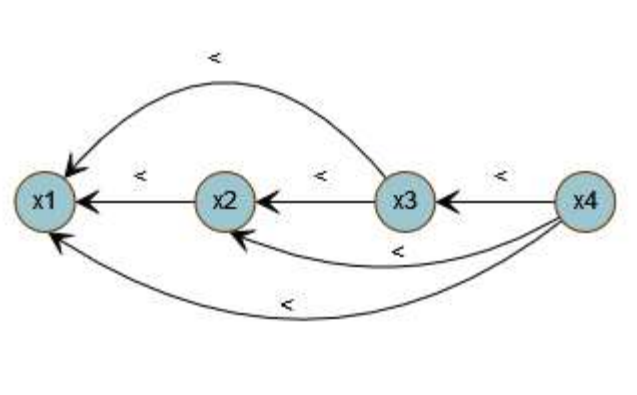
First of all, here's how each heuristic works:

- Minimum remaining values (also known as Max-conflicts) always selects the most constrained variable first.
- Least constraining values (Min-conflicts) always selects the value with the minimum conflicts on the neighbors.

Second, I want to point out some remarks of this specific problem (instance) to make it *easier* to solve (given the limited time and that we are reasoning minds, unlike computers):

All the binary constraints in the problem form what is known as an *strict order* (in this case, *total*), so the problem has some additional *implicit constraints*:

- Since x_4 is greater than x_3, and x_3 is greater than all the rest, then x_4 is greater than all the rest.
- This means that for all x_i, $x_i > x_1, \ldots x_i$ and for all $i > 1$. Which means that x_4 is dependent on three variables, and so on. The final problem ends up looking something similar to this:



Which affects the variable selection process.

Regarding the selected values, in order for the concatenation of all digits x_i to be odd, the value of x_4 needs to be odd, therefore we have an additional *implicit* constraint:

- $C_u = \{x_4 \mod 2 = 1\}$

We can also simplify d_4 (since all values need to be odd) as:

- $d_{4_u} = \{1, 3, 5\}$
- $D_u = D \setminus d_4 \cup d_{4_u}$

So the final instance is $CSP = (X, D_u, C \cup C_u)$.

PropositionalLet U be the set of unassigned variables. For simplicity, consistency will only be checked using C and not $C \cup C_u$. The assignment would then be:

1. U: $\{x_1, x_2, x_3, x_4\}$
   - MaxConflicts: x_4
   - d_4: $\{1, 3, 5\}$
   - MinConflicts = 5
2. Domain updates:
   - $d_1 = \{1:4\}$
   - $d_2 = \{1:4\}$
   - $d_3 = \{1:4\}$
3. U: $\{x_1, x_2, x_3\}$
   - MaxConflicts: x_3
   - d_3: $\{1, 2, 3, 4\}$
   - MinConflicts = 4
   - Consistency: 5 > 4, check.
4. Domain updates:
   - $d_1 = \{1:3\}$
   - $d_2 = \{1:3\}$
5. U: $\{x_1, x_2\}$
   - MaxConflicts: x_2
   - d_2: $\{1, 2, 3\}$
   - MinConflicts: 2 (2 and 3 have the same number of conflicts, but 2 is first chronologically)
   - Consistency: 5 > 4 > 2, check.
6. Domain updates:
   - $d_1 = \{1\}$
7. U: $\{x_1\}$
   - MaxConflicts: x_1
   - d_1: $\{1\}$
   - MinConflicts: 1
   - Consistency: 5 > 4 > 2 > 1, check.
8. Global constraint check: $(x_1 \# x_2 \# x_3 \# x_4 \mod 2 = 1)$
   - $1245 \mod 2 = 1$, check.
9. Success.

Therefore, assignment is $\{x_1 = 1, x_2 = 2, x_3 = 4, x_4 = 5\}$

For graders: Students might have used AC3 or forward checking to simplify the problem. These should be considered acceptable answers, no points deducted from the score.

Task 5: Propositional logic - Boxes with gold

Three boxes are presented to you. One contains gold, the other two are empty. Each box has imprinted on it a clue as to its contents. The clues are:

- Box 1: *The gold is not here*
- Box 2: *The gold is not here*
- Box 3: *The gold is in Box 2*

Only one message is true, the other two are false. Which box has the gold? Solve the puzzle using propositional logic by completing the following steps:

5.1) Let B_i stand for the gold is in the i-th box, where $i \in [1:3]$. Formalize the following statements of the problem in propositional logic:

- One box contains gold, the other two are empty
- Only one message is true, the other two are false. Do the logical equivalent if needed.

Solution. This means we have 3 variables, $B_i, i \in [1:3]$. The statements S are the following:

- S_1: one box contains gold, the other two are empty.
- S_2: Only one message is true, the other two are false.

Since we don't have quantifiers in Propositional logic, we need the full specification: either one of the boxes contain it but the others don't, so:

$$S_1 = (B_1 \land \neg B_2 \land \neg B_3) \lor (\neg B_1 \land B_2 \land \neg B_3) \lor (\neg B_1 \land \neg B_2 \land B_3)$$

Notice how each clause (enclosed in parentheses) is and AND and as it will always be true.

For S_2 we need the information about the messages. Let M_i be the i-th message:

- B_1 says "not me", therefore $M_1 = \neg B_1$
- B_2 says "not me", therefore $M_2 = \neg B_2$
- B_3 says "Box 2", therefore $M_3 = B_2$

Then we can formalize S_2:

$$S_2 = (M_1 \land \neg M_2 \land \neg M_3) \lor (\neg M_1 \land M_2 \land \neg M_3) \lor (\neg M_1 \land \neg M_2 \land M_3)$$

And now we substitute with the actual values of the messages:

$$S_2 = (\neg B_1 \land \neg\neg B_2 \land \neg B_2) \lor (\neg\neg B_1 \land \neg B_2 \land \neg B_2) \lor (\neg\neg B_1 \land \neg\neg B_2 \land B_2)$$

And simplify:

$$\begin{aligned} S_2 &= (\neg B_1 \land B_2 \land \neg B_2) \lor (B_1 \land \neg B_2 \land \neg B_2) \lor (B_1 \land B_2 \land B_2) && (5.1a)\\ &= (\neg B_1 \land F) \lor (B_1 \land \neg B_2) \lor (B_1 \land B_2) && (5.1b)\\ &= (B_1 \land \neg B_2) \lor (B_1 \land B_2) && (5.1c)\\ &= B_1 \land (\neg B_2 \lor B_2) && (5.1d)\\ &= B_1 \land T && (5.1e)\\ &= B_1 && (5.1f) \end{aligned}$$

(5.1a) is obtained replacing the **double negation** for the truth values. Then a contradiction is obtained in the first clause, while identical values can be grouped together (in the second and third clauses) to obtain (5.1.b). Finally, a contradiction always results in a false statement and hence cancelling the first clause in (5.1b). Expression (5.1c) shows the final expression S_2 .

You can further reduce the expression (from 5.1.c and onwards) by using the distributivity of conjunction over disjunction.

5.2) Compute the truth table for expressions S_1 and *simplified* S_2, which we call S_3, and make an inference about which box has the gold.

Solution. The solution for this is using the Python package `ttg` which stands for truth table generator.

In [5]:
```
import ttg
```

$$S_1 = (B_1 \land \neg B_2 \land \neg B_3) \lor (\neg B_1 \land B_2 \land \neg B_3) \lor (\neg B_1 \land \neg B_2 \land B_3)$$

We use the object `Truths` and instantiate it using a list of symbols (`vars`), and then a list of strings representing the conditions we want to test.

First, I am making a table comparing both the original statement S_2 as well as the simplified version, S_3, to demonstrate that they are equivalent:

In [6]:
```
vars = ["B1", "B2", "B3"]

s11 = "(B1 and (~B2) and (~B3))"
s12 = "((~B1) and B2 and (~B3))"
s13 = "((~B1) and (~B2) and B3)"

s21 = "(~(B1) and B2 and ~(B2))"
s22 = "(B1 and (~B2) and (~B2))"
s23 = "(B1 and B2 and B2)"

s1 = f"{s11} or {s12} or {s13}"
s2 = f"{s21} or {s22} or {s23}"
s3 = "(B1 and ~(B2)) or (B1 and B2)"

ttg.Truths(vars, [s2, s3]).as_pandas()
```

Out[6]:

	B1	B2	B3	(~(B1) and B2 and ~(B2)) or (B1 and (~B2) and (~B2)) or (B1 and B2 and B2)	(B1 and ~(B2)) or (B1 and B2)
1	1	1	1	1	1
2	1	1	0	1	1
3	1	0	1	1	1
4	1	0	0	1	1
5	0	1	1	0	0
6	0	1	0	0	0
7	0	0	1	0	0
8	0	0	0	0	0

The table shows how both original `s2` and simplified `s3` are equivalent as both columns are identical, which means our simplification was correct.

Now using the tables for both S_1 and S_3 we get:

In [7]:
```
ttg.Truths(vars, [s1, s3, f"({s1}) and ({s3})"]).as_pandas()
```

Out[7]:

	B1	B2	B3	(B1 and (~B2) and (~B3)) or ((~B1) and B2 and (~B3)) or ((~B1) and (~B2) and B3)	(B1 and ~(B2)) or (B1 and B2)	((B1 and (~B2) and (~B3)) or ((~B1) and B2 and (~B3)) or ((~B1) and (~B2) and B3)) and ((B1 and ~(B2)) or (B1 and B2))
1	1	1	1	0	1	0
2	1	1	0	0	1	0
3	1	0	1	0	1	0
4	1	0	0	1	1	1
5	0	1	1	0	0	0
6	0	1	0	1	0	0
7	0	0	1	1	0	0
8	0	0	0	0	0	0

Considering that both S_1 AND S_3 are true statements, then only the rows evaluated to 1 on both columns are our viable options. I went ahead and made an AND between them which is presented in the last column. This last column has a single true row, which is **row 4**. If we look at that interpretation, we get:

$$B_1 = T, B_2 = F, B_3 = F$$

which complies with the information: only one box says the truth, and the other two lie. Hence we know that box B_1 has the gold.

5.3) Instead, if resolution refutation should be used for inference, how would you define the knowledge base KB and query α? And what should be achieved to prove the entailment $KB \models \alpha$?

Solution. Since we know both statements S_1 and S_3, we can use them as the rules for our Knowledge Base (KB). Both rules are already in Conjunctive Normal Form, so we don't need to do anything.

Our queries could as follows. However, in order to use refutation we need to **negate** the query. This would generate a contradiction when using the KB, and hence means that our assumption (the opposite of our query) was wrong, from where it follows that the original query is true.

- Query 1: B_1 has the gold, so α is B_1
- Query 2: B_2 has the gold, so α is B_2
- Query 3: B_3 has the gold, so α is B_3

This means that the KB would consist of S_1 and either S_2 or S_3. Then we could add $\neg B_1$ to KB and the resolution process starts, by turns $\neg B_2$ and $\neg B_3$ as well.

Task 6: Planning - Monkey and bananas

A monkey in a laboratory wants to get some bananas which are hanging on the ceiling out of reach. However, a box that could enable the monkey to reach the bananas (if the monkey were to climb onto it) is available nearby. Consider the following:

- Initially, the monkey is at A, the bananas at B and the box at C.
- Both the monkey and the box have height Low, but if the monkey climbs onto the box he will have height $High$, which is the same as the bananas.
- The actions available to the monkey include:
  - Go from one place to another
  - $Push$ an object from one place to another
  - $ClimbUp$ onto on an object
  - $ClimbDown$ from an object *(NB! This one is no longer used in the exam. Left up for completeness purposes.)*
  - $Grab$ an object (only if the object and the monkey are in the same place and at the same height)
  - $Release$ an object *(NB! This one is no longer used in the exam. Left up for completeness purposes.)*

7 .1) Specify the problem using Planning Domain Definition Language (PDDL). Include initial and goal state descriptions as well as actions schemas.

Solution.

Let's start with the constants: M for monkey, $High$ and Low are heights. Additionally, some *helper* function symbols are introduced for clarity: $On(x, y)$ and $Holding(x, y)$ in case x is on top of y, or x is holding y, respectively. This makes it easier for us to declare the state of the world for such cases.

The actions are then:

- $Action(Go(from, to),$
  - $PRECOND$: $At(M, from)$
  - $EFFECT$: $\neg At(M, from) \land At(M, to))$
- $Action(Push(obj, from, to),$
  - $PRECOND$: $At(M, from) \land At(obj, from) \land Holding(M, obj)$
  - $EFFECT$: $\neg At(M, from) \land \neg At(obj, from) \land At(M, to) \land At(obj, to))$
- $Action(ClimbUp(obj, loc),$
  - $PRECOND$: $At(M, loc) \land At(obj, loc) \land Height(M, Low) \land Height(obj, Low)$
  - $EFFECT$: $\neg Height(M, Low) \land Height(M, High) \land On(M, obj))$
- $Action(ClimbDown(obj, loc),$
  - $PRECOND$: $At(M, loc) \land At(obj, loc) \land Height(M, High), On(M, obj)$
  - $EFFECT$: $\neg Height(M, High) \land \neg On(M, obj) \land Height(M, Low))$
- $Action(Grab(obj, loc, h),$
  - $PRECOND$: $At(M, loc) \land At(obj, loc) \land Height(M, h) \land Height(obj, h)$
  - $EFFECT$: $Holding(M, obj) \land \neg At(obj, loc) \land \neg Height(obj, h))$
- $Action(Release(obj, loc),$
  - $PRECOND$: $At(M, loc) \land Holding(M, obj)$
  - $EFFECT$: $At(obj, loc) \land Height(obj, Low) \land \neg Holding(M, obj))$ $\square$

The initial state is:

$$\begin{aligned} &Init(At(M, A) \land At(Bananas, B) \land (Box, C)\\ &\quad \land Height(M, Low) \land Height(Bananas, High) \land Height(Box, Low)) \square \end{aligned}$$

And the goal state is $Goal(Holding(M, Bananas))$

7.2) Solve the problem using backwards search (regression). Show the first 3 steps in one of the branches, including actions and states after/before each action.

NB! This question now asks for only the first three steps. This solution shows the whole plan for completeness, and describes how the state looks after every action in the plan.

Solution. Using **backwards search** what we do is start from the goal and continue searching back to the start. We list the plan P as well as the state S after every action:

Since $Goal(Holding(M, Bananas))$, then for M to be $Holding\ Bananas$, it had to $Grab$ them:

$$P = [Grab(M, Bananas)]$$
$$S = (Holding(M, Bananas))$$

To $Grab$ them, M should have been at the same height ($High$) and same place (B), which means it would have had to $ClimbUp$ the Box at B:

$$P = [ClimbUp(Box, B), Grab(M, Bananas)]$$
$$S = (At(M, B) \land At(Bananas, B) \land At(Box, B) \land\ Height(M, High) \land Height(Bananas, High) \land Height(Box, Low) \land On(M, Box))$$

To $ClimbUp$ the Box at location B, M should have Pushed the Box there, but for that it had to Grab it and then Release. Let's go one step at a time:

$$P = [Release(Box, B), ClimbUp(Box, B), Grab(M, Bananas)]$$
$$S = (At(M, B) \land At(Bananas, B) \land At(Box, B) \land\ Height(M, Low) \land Height(Bananas, High) \land Height(Box, Low))$$

To let go off the Box, it should have had to $Grab$ it first. However, since it was $Push$ed from the previous location, it was $Grab$bed on the previous location. Therefore it was $Push$ed to location B from its previous location, C:

$$P = [Push(Box, C, B), Release(Box, B), ClimbUp(Box, B), Grab(M, Bananas)]$$
$$S = (At(M, B) \land At(Bananas, B) \land At(Box, B) \land\ Height(M, Low) \land Height(Bananas, High) \land Height(Box, Low) \land Holding(M, Box))$$

To $Push$ the Box from C, M should have $Grab$bed it first:

$$P = [Grab(Box, C), Push(Box, C, B), Release(Box, B), ClimbUp(Box, B), Grab(M, Bananas)]$$
$$S = (At(M, C) \land At(Bananas, B) \land At(Box, C) \land\ Height(M, Low) \land Height(Bananas, High) \land Height(Box, Low) \land Holding(M, Box))$$

M should have been at C in order to $Grab$ the Box, and since its starting location was not C then it would have had to Go from its previous location A to the location of the Box, C:

$$P = [Go(A, C), Grab(Box, C), Push(Box, C, B), Release(Box, B), ClimbUp(Box, B), Grab(M, Bananas)]$$
$$S = (At(M, C) \land At(Bananas, B) \land At(Box, C) \land\ Height(M, Low) \land Height(Bananas, High) \land Height(Box, Low))$$

Then, we can compare the state S to the initial state:

$$Init(At(M, A) \land At(Bananas, B) \land (Box, C) \land\ Height(M, Low) \land Height(Bananas, High) \land Height(Box, Low))$$

The only difference is the position of M, $At(M, A)$, which is a $PRECOND$ for $Go(A, C)$. Hence, the final plan is:

$$P = [Go(A, C), Grab(Box, C), Push(Box, C, B), Release(Box, B), ClimbUp(Box, B), Grab(M, Bananas)]$$