Week 15 (A): Game Theory (Part1: Adversarial Search)

1. Games

1.1. Type of Games

1.1.1. By information

Perfect Information:

deterministic, fully observable environments

- All players know the game structure.
- Each player, when making any decision, is **perfectly informed** of all the events that have **previously** occurred.
- or to say: players have the entire state of the game available to them to use and strategise

Imperfect Information:

Not perfect information

Attention: perfect information ≠ complete information

- **complete information means** common knowledge of each player's utility functions, payoffs, strategies and "types".
- A game with perfect information may or may not have complete information.

1.1.2. By Deterministic

Deterministic Game

- you give an input and always have the same output.
- a particular move always leads to the same result

> **e.g.** In **chess**, if you move your piece, both you and your opponent **know where it's going to end up**. It doesn't randomly fail to move, it doesn't randomly end up elsewhere, it doesn't cause any surprise at all.

Non-Deterministic Game (chance game)

You choose to do an action and something entirely different may happen

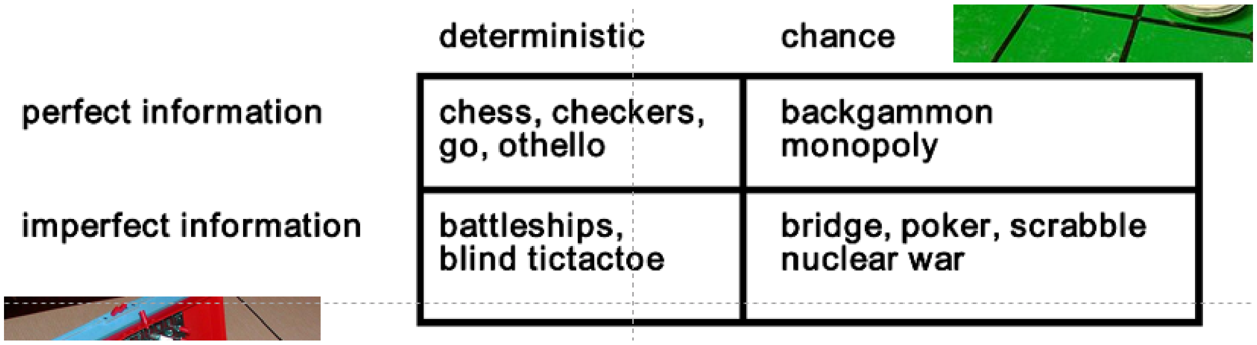
1.1.3. By output

Constant-Sum game

situation in which each participant's gain or loss of utility is exactly balanced by the losses or gains of the utility of the other participants

Coordination

1.1.4. Some examples



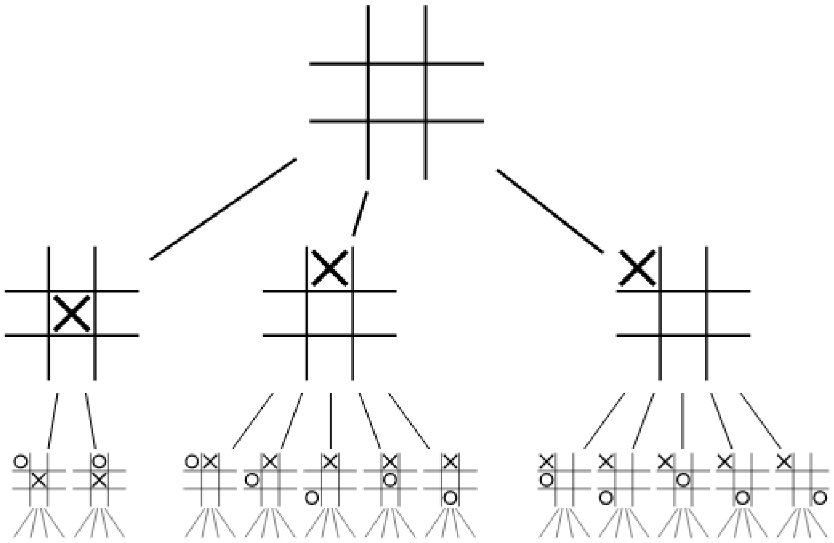
	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information	battleships, blind tictactoe	bridge, poker, scrabble nuclear war

1.2. Challenge

- writing effective game-playing agents for multi-plaer games with **probabilistic events**
- writing effective game-playing agents for multi-plaer games with **incomplete information**
- deterministic games with perfect information can theoretically **be fully solved** if the entire state space is memorized

1.3. Game Tree

- a **game tree** is a directed graph whose nodes are positions in a game and whose edges are moves.
- The **complete game tree** for a game is the game tree starting at the initial position and containing all possible moves from each position
- Game tree **size** is the total number of different games that can be played. It is **the number of leaf nodes** in the game tree.



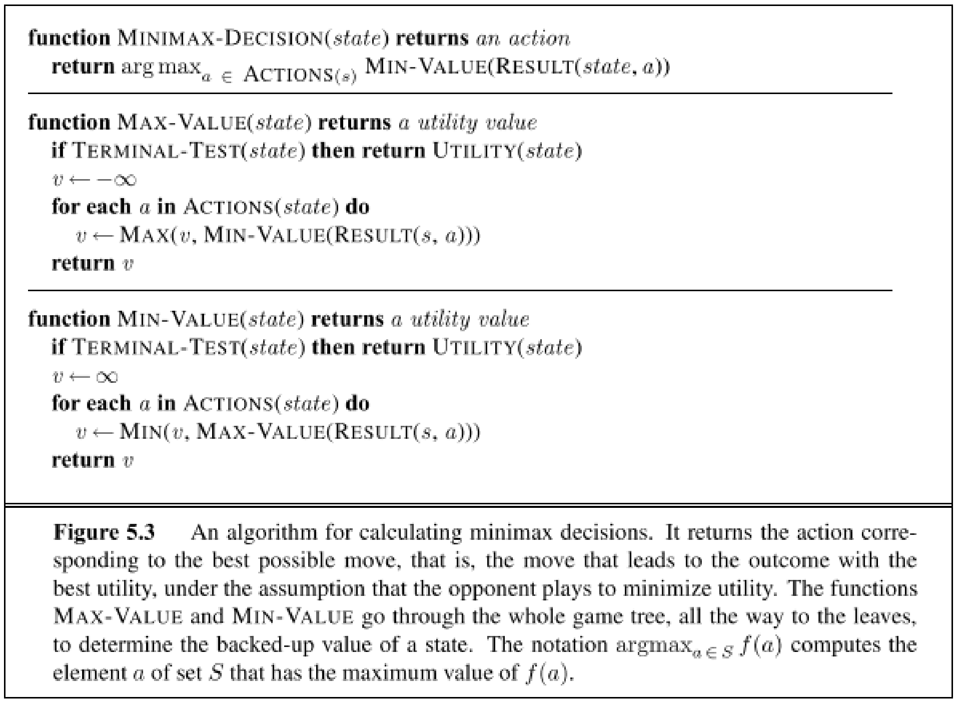
1.4. State Space

State Space is the number of legal unique game positions that can be achieved from the starting position

2. Optimal Decisions

2.1. Minimax Algorithm

Minimax is a perfect play for deterministic games



```
function MINIMAX-DECISION($state$) returns $an\ action$
  return $\arg\max_{a\ \in\ \text{ACTIONS}(s)}$ MIN-VALUE(RESULT($state$, $a$))

function MAX-VALUE($state$) returns $a\ utility\ value$
  if TERMINAL-TEST($state$) then return UTILITY($state$)
  $v \leftarrow -\infty$
  for each $a$ in ACTIONS($state$) do
    $v \leftarrow$ MAX($v$, MIN-VALUE(RESULT($s$, $a$)))
  return $v$

function MIN-VALUE($state$) returns $a\ utility\ value$
  if TERMINAL-TEST($state$) then return UTILITY($state$)
  $v \leftarrow \infty$
  for each $a$ in ACTIONS($state$) do
    $v \leftarrow$ MIN($v$, MAX-VALUE(RESULT($s$, $a$)))
  return $v$
```

Figure 5.3 An algorithm for calculating minimax decisions. It returns the action corresponding to the best possible move, that is, the move that leads to the outcome with the best utility, under the assumption that the opponent plays to minimize utility. The functions MAX-VALUE and MIN-VALUE go through the whole game tree, all the way to the leaves, to determine the backed-up value of a state. The notation $\text{argmax}_{a \in S} f(a)$ computes the element a of set S that has the maximum value of $f(a)$.

Understanding

- From the bottom of game tree and propogate upward.
- The root is max, then next level min, then max ...
- Nodes in Min level find the minimum value of its children
- Nodes in Max level find the maximum value in its children

Properties

- **Complete** if tree is **finite**
- The solution is **Optimal**
- **Time complexity** $O(b^m)$, like DFS
    - **b** is branching factor of the game-tree, and **m** is the maximum depth of the tree
- **Space complexity** $O(bm)$, like DFS

Drawbacks

- minimax algorithm's time and space complexity is too large
- The correct answer is not displayed for **Written Response type questions.**

Multiplayer situation

1. First replace the single value for each node with a **vector of values**
2. The backed-up value of a node n is always the utility vector of the successor state with the **highest value** for the player choosing at n

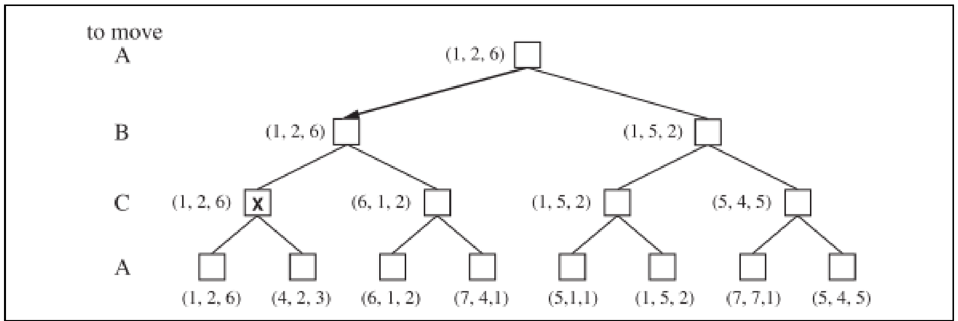
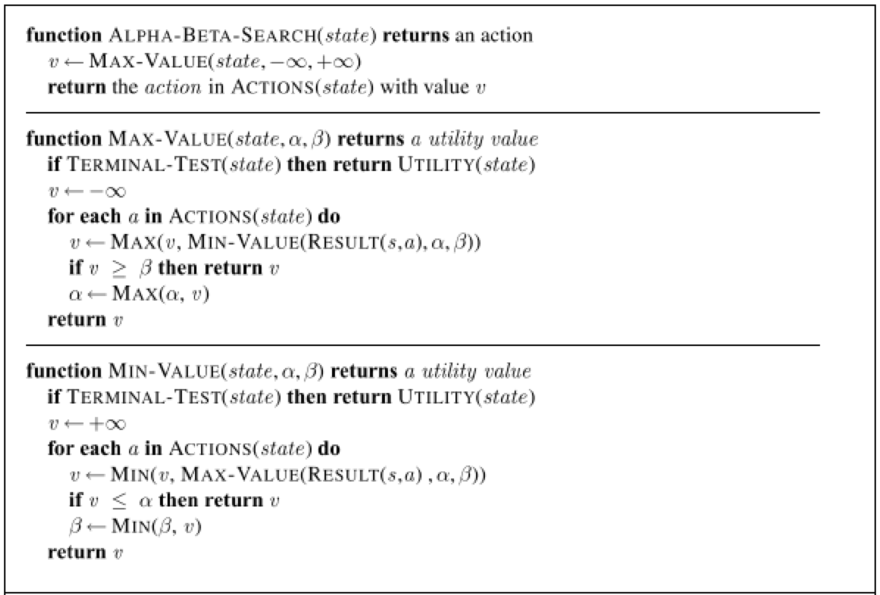


Figure 5.4 The first three plies of a game tree with three players (A, B, C). Each node is labeled with values from the viewpoint of each player. The best move is marked at the root.

2.2. ALPHA-BETA PRUNING



```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

Figure 5.7 The alpha–beta search algorithm. Notice that these routines are the same as the MINIMAX functions in Figure 5.3, except for the two lines in each of MIN-VALUE and MAX-VALUE that maintain α and β (and the bookkeeping to pass these parameters along).

Understanding

- Each time, when doing a extreme computation, see one step upward, and use the result to pruning
    - for min, when a previous node has larger lower-bound, then abort current nodes
    - for max, when a previous node has smaller upper-bound, then abort current nodes

Properties

- Pruning does note affect final result
- Good move ordering improves effectiveness of pruning
- with "perfect ordering", **time complexity** = $O(b^{m/2})$

> The alpha-beta pruning algorithm seeks to reduce the number of nodes that are evaluated in the minimax tree. It stops evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. Such moves need not be evaluated further. When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decisi

Examle

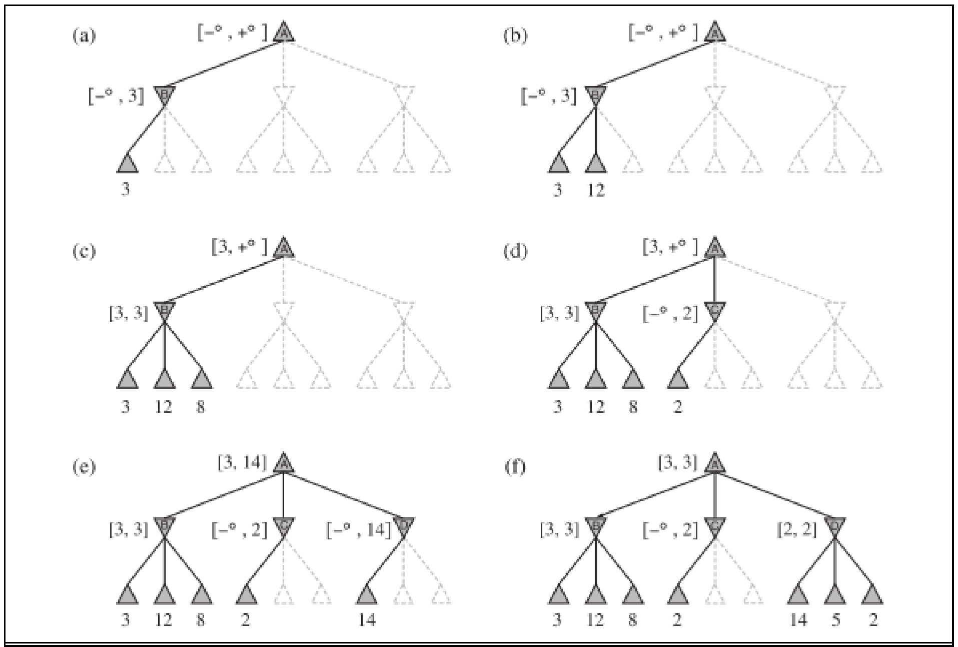


Figure 5.5 Stages in the calculation of the optimal decision for the game tree in Figure 5.2. At each point, we show the range of possible values for each node. (a) The first leaf below B has the value 3. Hence, B, which is a MIN node, has a value of *at most* 3. (b) The second leaf below B has a value of 12; MIN would avoid this move, so the value of B is still at most 3. (c) The third leaf below B has a value of 8; we have seen all B's successor states, so the value of B is exactly 3. Now, we can infer that the value of the root is *at least* 3, because MAX has a choice worth 3 at the root. (d) The first leaf below C has the value 2. Hence, C, which is a MIN node, has a value of *at most* 2. But we know that B is worth 3, so MAX would never choose C. Therefore, there is no point in looking at the other successor states of C. This is an example of alpha–beta pruning. (e) The first leaf below D has the value 14, so D is worth *at most* 14. This is still higher than MAX's best alternative (i.e., 3), so we need to keep exploring D's successor states. Notice also that we now have bounds on all of the successors of the root, so the root's value is also at most 14. (f) The second successor of D is worth 5, so again we need to keep exploring. The third successor is worth 2, so now D is worth exactly 2. MAX's decision at the root is to move to B, giving a value of 3.

3. Imperfect Real-Time Decisions

3.1. Background

$\alpha - \beta$ pruning searches all the way to terminal nodes

3.2. Limited game tree search

there are two standard approach:

1. **Cutoff test**

for example depth limit

2. **Heuristic evaluation function**

   for example estimated esirablity of position

3.3. Evaluation functions

An **evaluation function** returns an estimate of the **expected utility** of the game from a **given position**

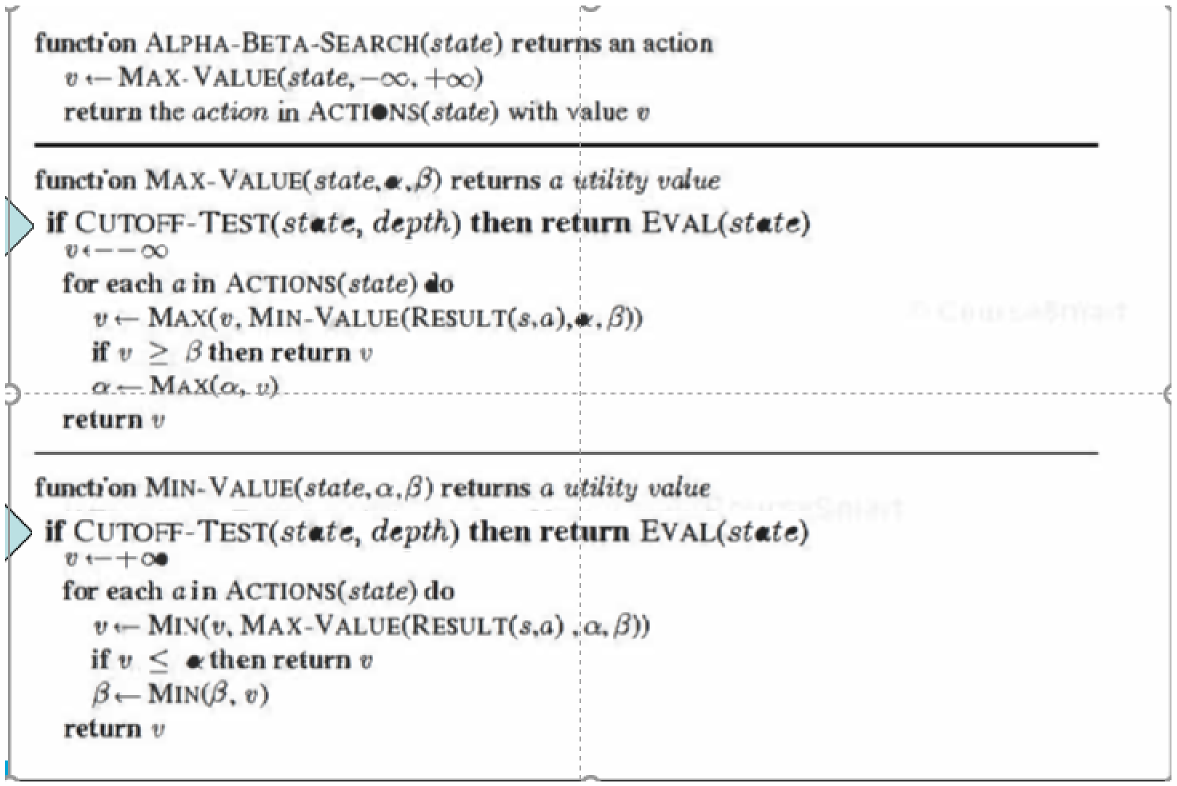
Standard:

- the evaluation function should order the terminal states in the same way **as the true utility function**: states that are wins must evaluate better than draws, which in turn must be better than losses
- Most evaluation functions work by calculating various features of the state: for example number of black chess and white chess

Weighted Linear Function

$$\mathrm{EvAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s)$$

3.4. Cutting off search



```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
  if CUTOFF-TEST(state, depth) then return EVAL(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) returns a utility value
  if CUTOFF-TEST(state, depth) then return EVAL(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a), α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

Set depth limitation

The most straightforward approach to controlling the amount of search is to **set a fixed depth limit** so that **CUTOFF-TEST(state, depth)** returns true for all depth greater than some fixed depth d.

- The depth d is chosen so that a move is selected **within the allocated time**.
- a more robust approach is to apply iterative deepening

4. Stochastic Games

In real life, many **unpredictable external events** can put us into unforeseen situations. Many games mirror this unpredictability by including a random element, such as the throwing of dice

We call these **stochastic games**

3.1. Stochastic Game Tree

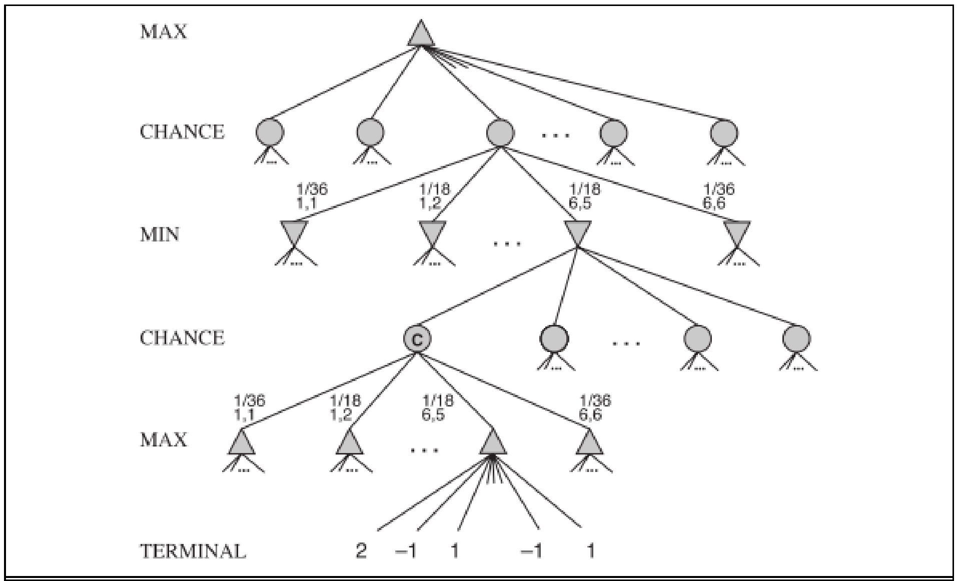


Figure 5.11 Schematic game tree for a backgammon position.

include **chance nodes** in addition to MAX and MIN nodes.

3.2. Algorithm for non-deterministic games

$$\text{EXPECTIMINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \\ \sum_r P(r)\text{EXPECTIMINIMAX}(\text{RESULT}(s,r)) & \text{if PLAYER}(s) = \text{CHANCE} \end{cases}$$