Minimax and Alpha-Beta Pruning

Traditional Game Playing [5.1]

- Two players
- Turn-taking (I move, you move, ...)
  - excludes: rock-paper-scissors
- Zero-sum (I win = you lose)
  - excludes cooperation, prisoners dilemma
- Perfect information
  - state is visible (opponent's strategy is not)
  - excludes: poker
- Deterministic
  - excludes: dice (backgammon), cards

Traditional Game Playing

State space:

- Two kinds of nodes
  - Player one moves
  - Player two moves
- States have values associated with them:
  - High = good for player one
  - Low = good for player two
- I maximize, you minimize

The minimax algorithm [5.2.1]

A simple game:

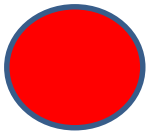
- Player 1: Wins if result > 0. Can 'play' (use without replacement) {1,0,-1}
- Player 2: Wins if result < 0. Can 'play' (use without replacement) {+,*}

Players play their possible moves and the resulting formula is calculated (iteratively) to obtain the result:

Eg. {1} {*} {0} {+} {-1} = -1 – Player Two wins!

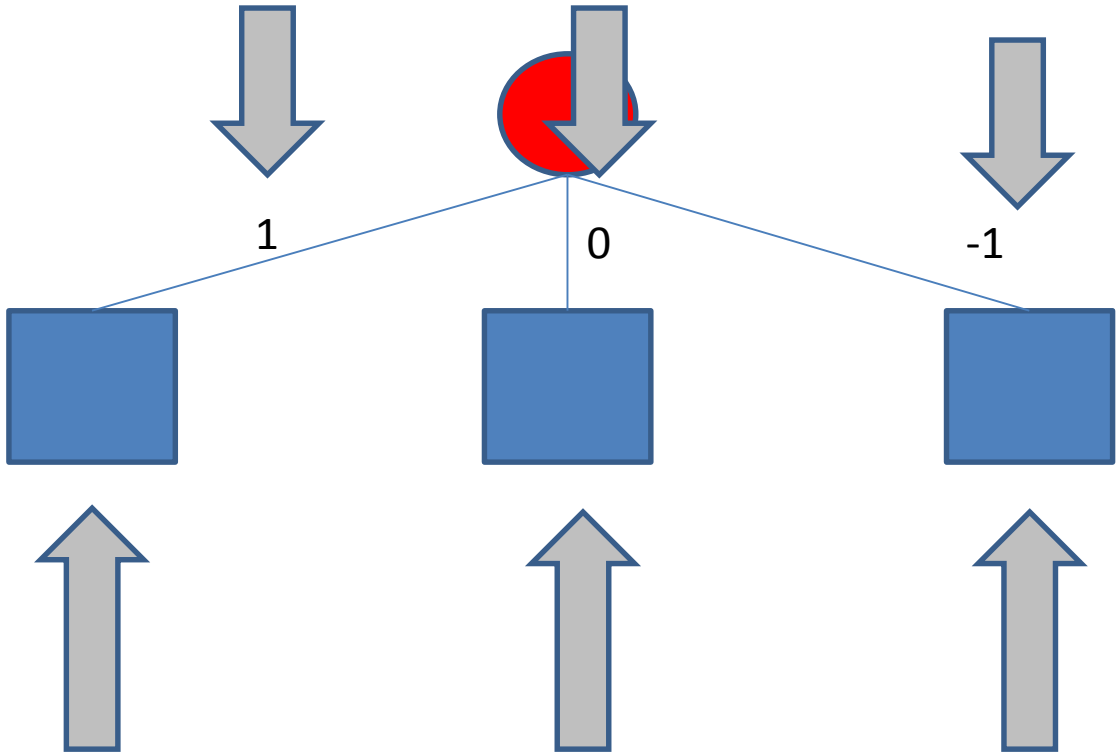
The minimax algorithm [5.2.1]

- Performs a simple recursive computation of the best moves for each player given successor states.

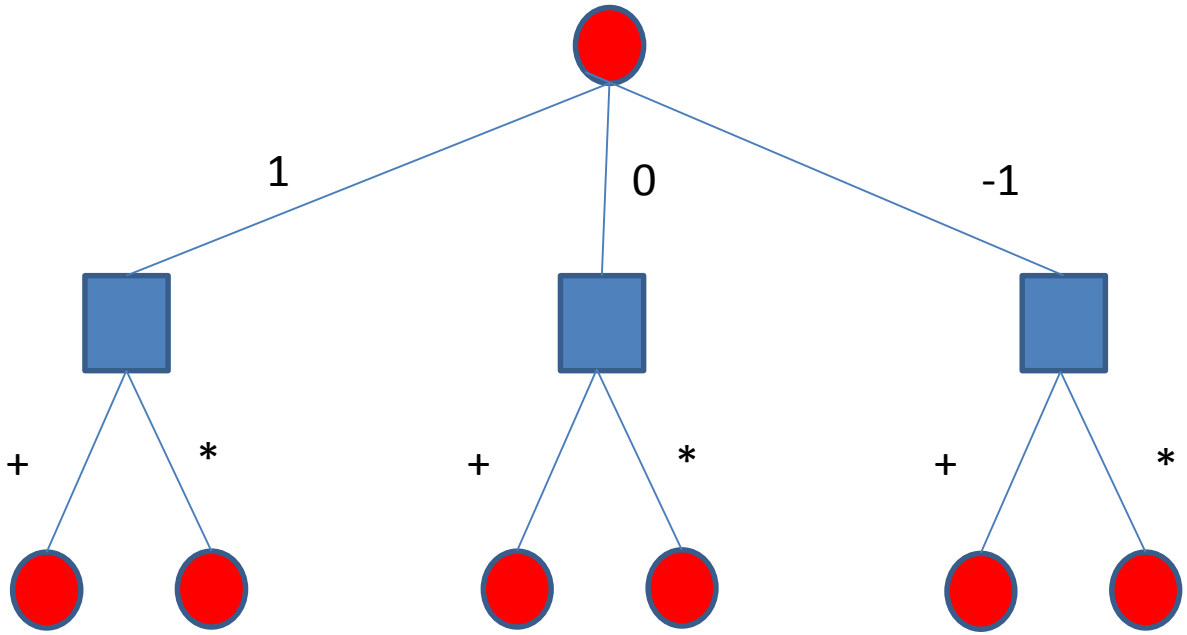


Player one chooses a first move.

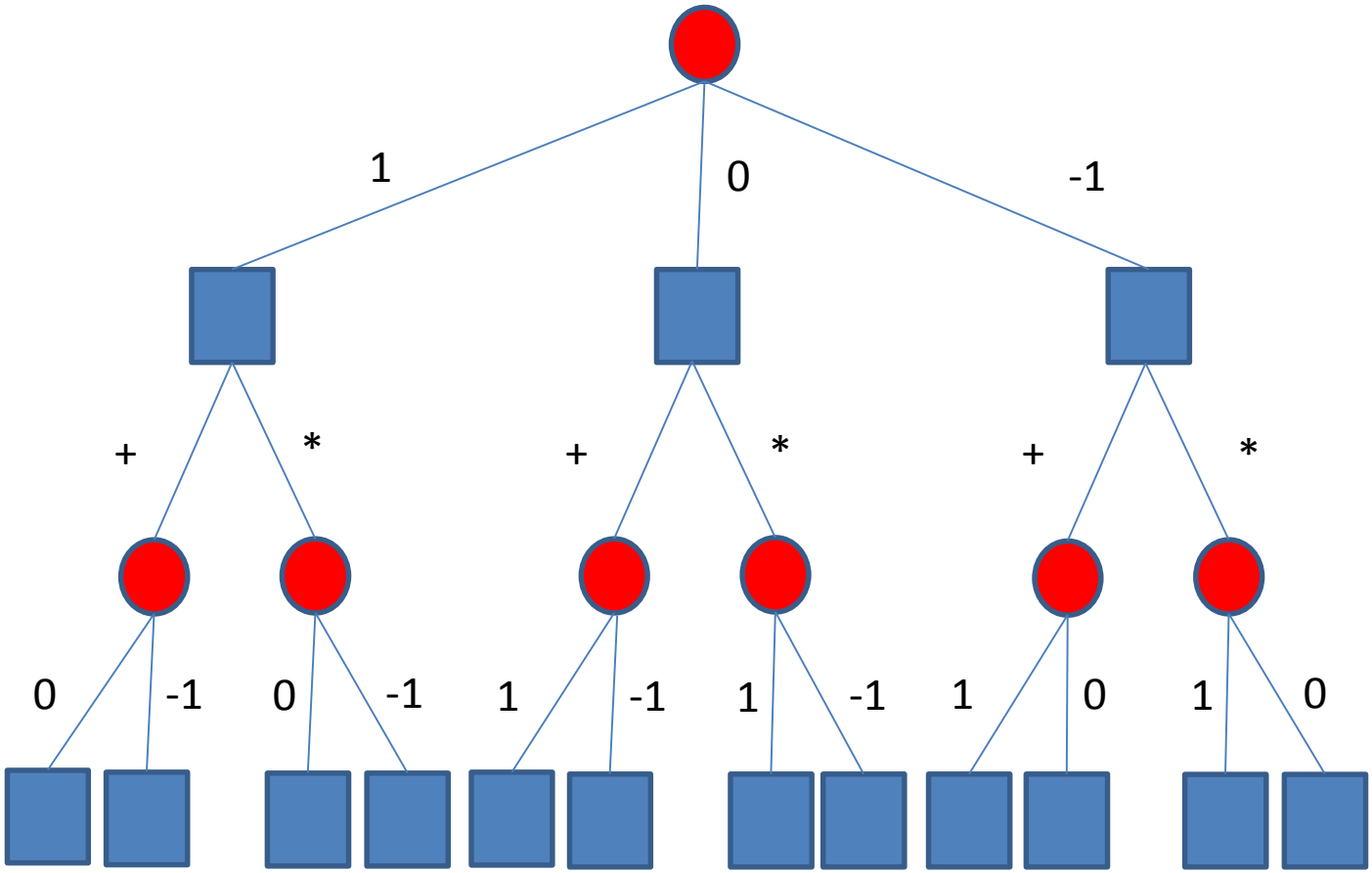
Depending on the move player one makes...



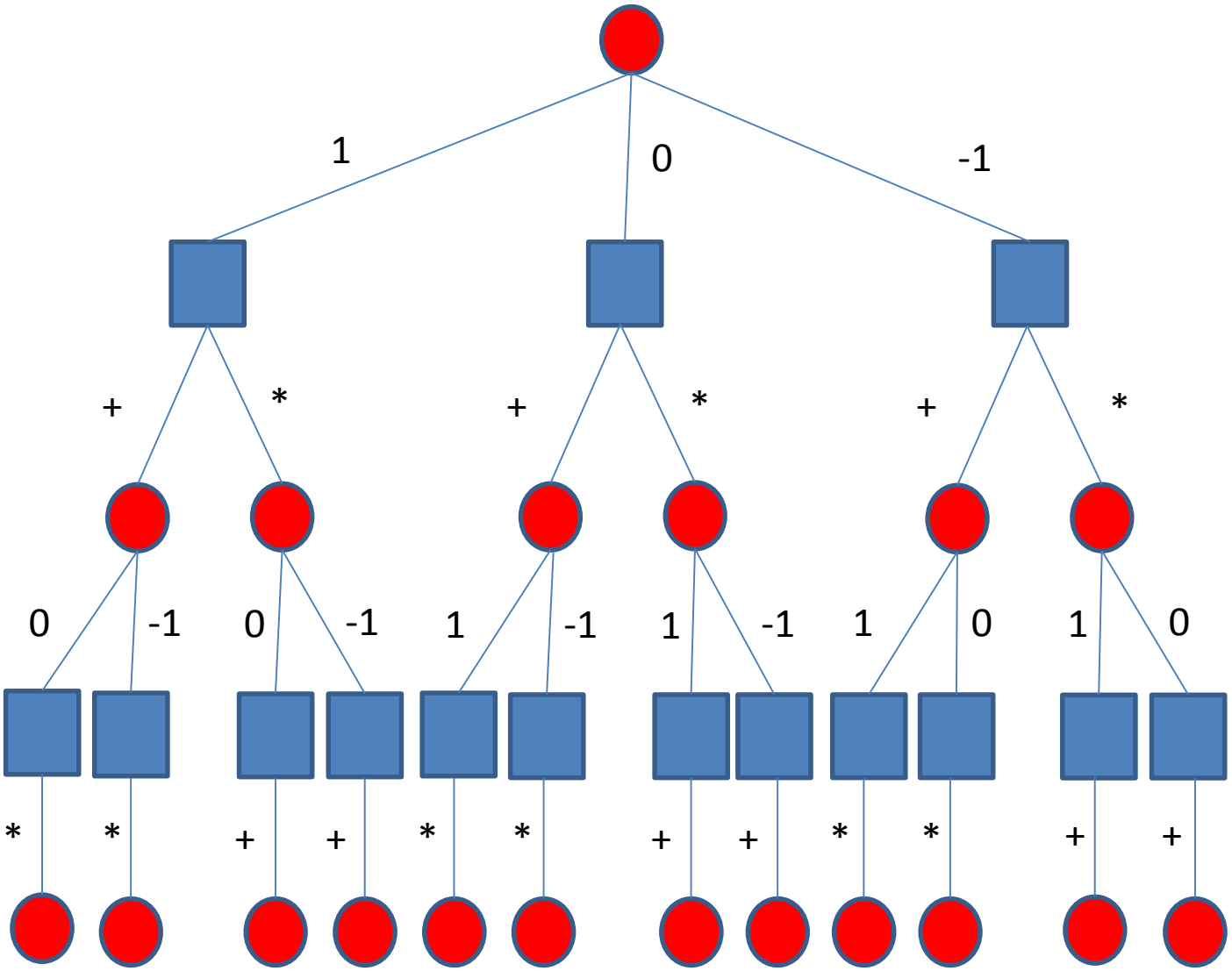
... the game will enter one of three states, at which point it will be player two's turn to make a move.



In each state there are two moves player two can make, so this leads to six possible states at player one's next turn...

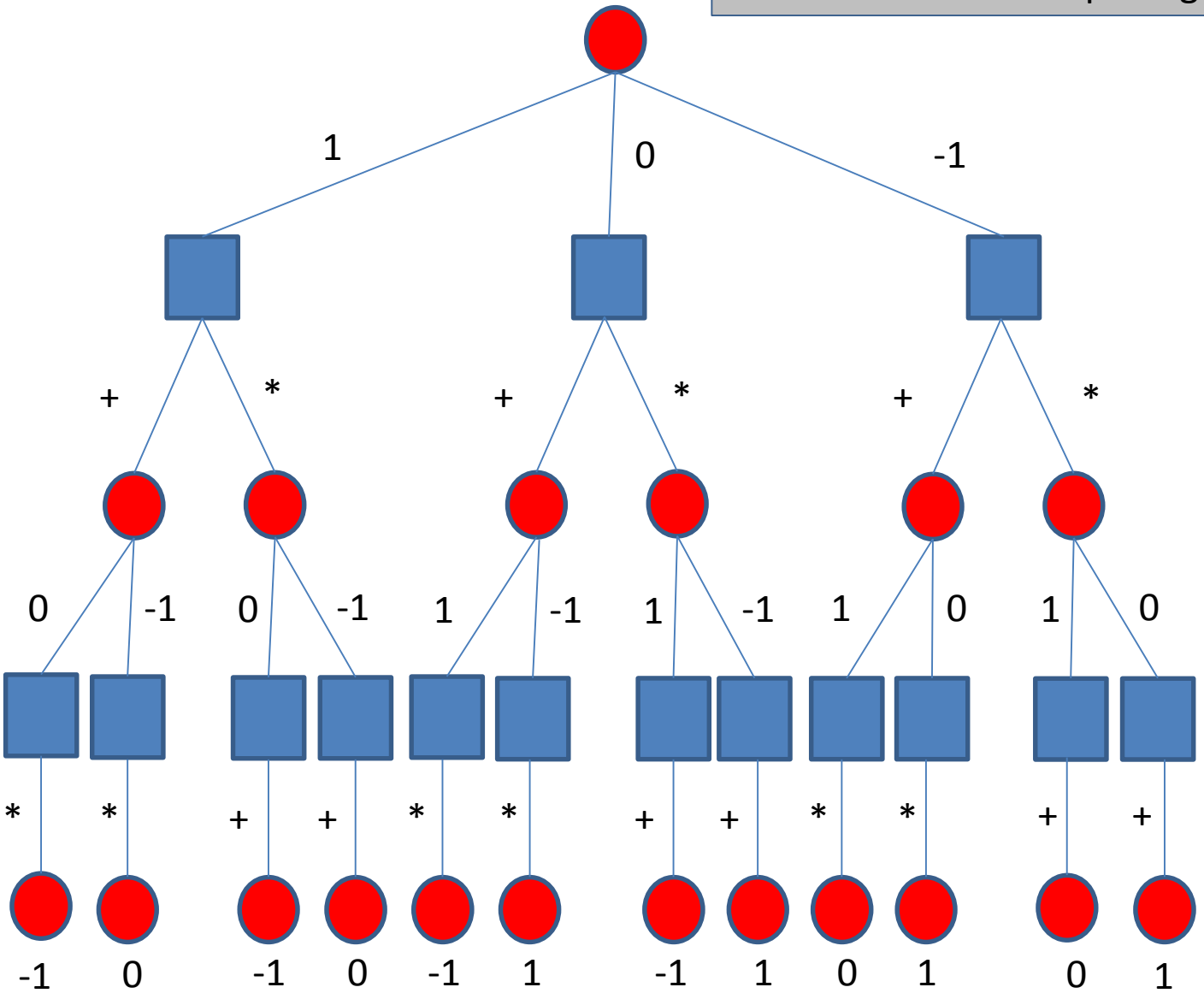


Player one can make two moves at each of these states.
Which moves these are depend on his first move.

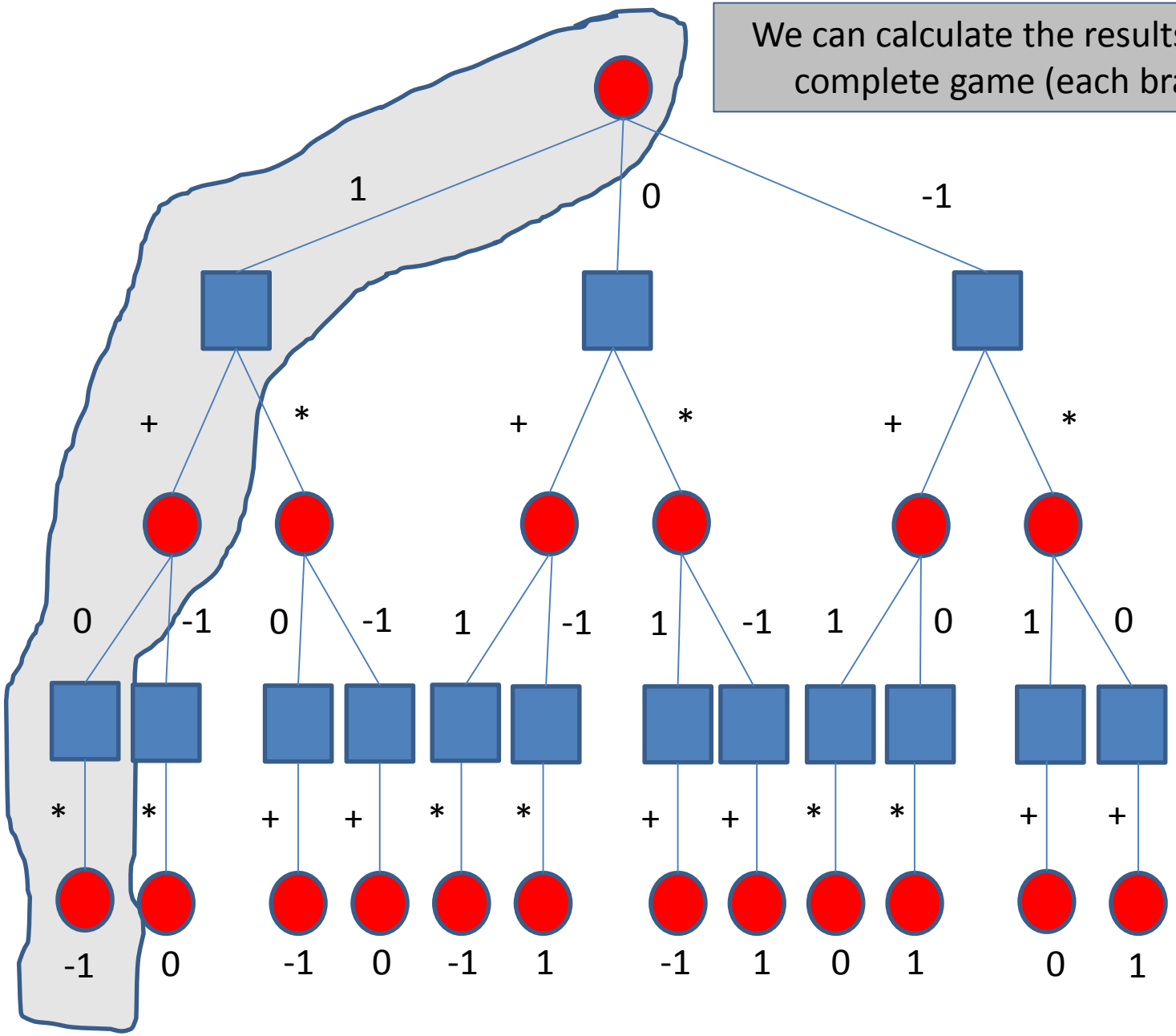


There are still 12 possible states, and it is player one's move. He also as only one move to play.

And here is the complete game tree

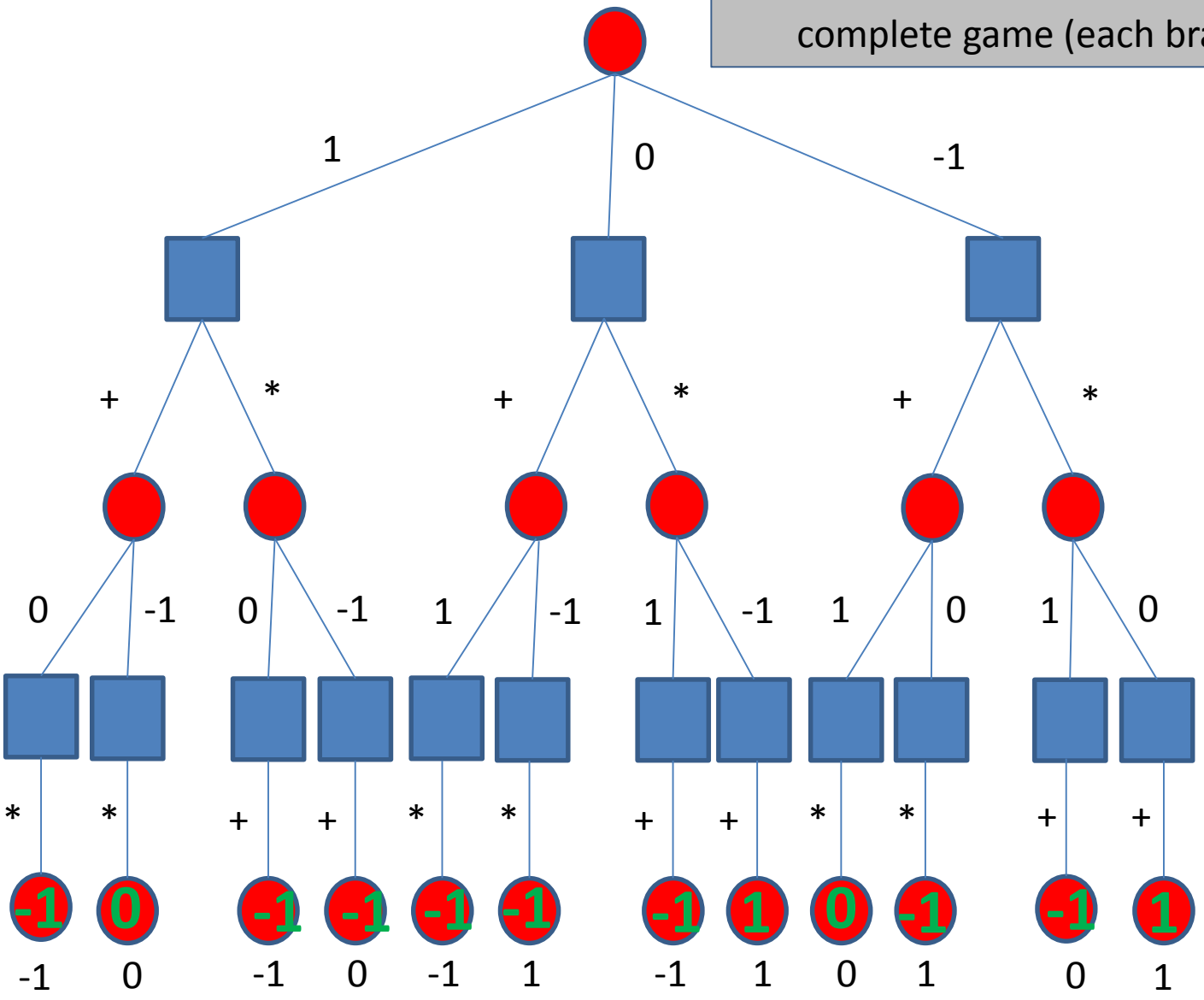


We can calculate the results of each complete game (each branch).

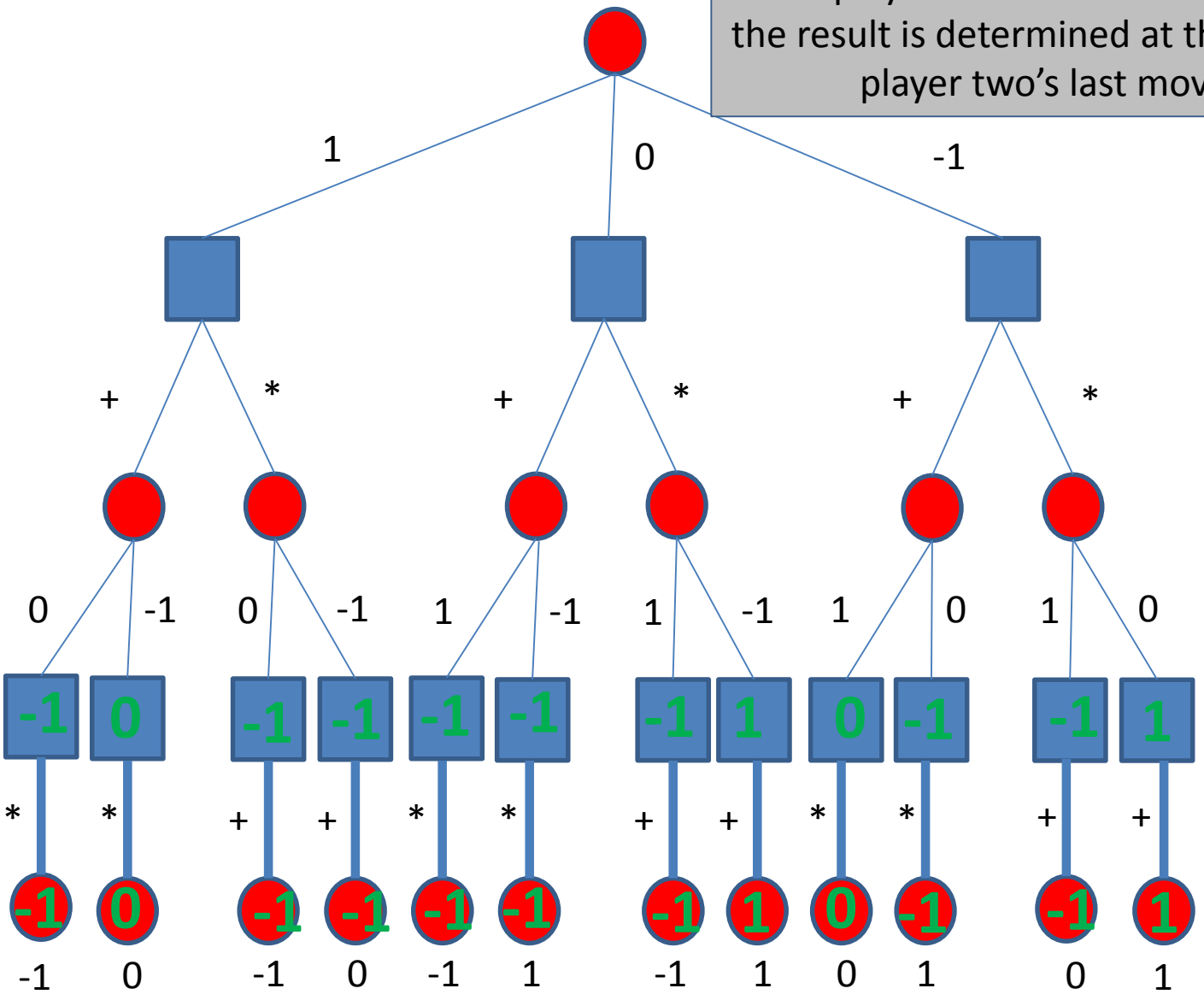


Result=1+0*(-1)=-1

We can calculate the results of each complete game (each branch).



Since player one's last move is forced, the result is determined at the time of player two's last move.

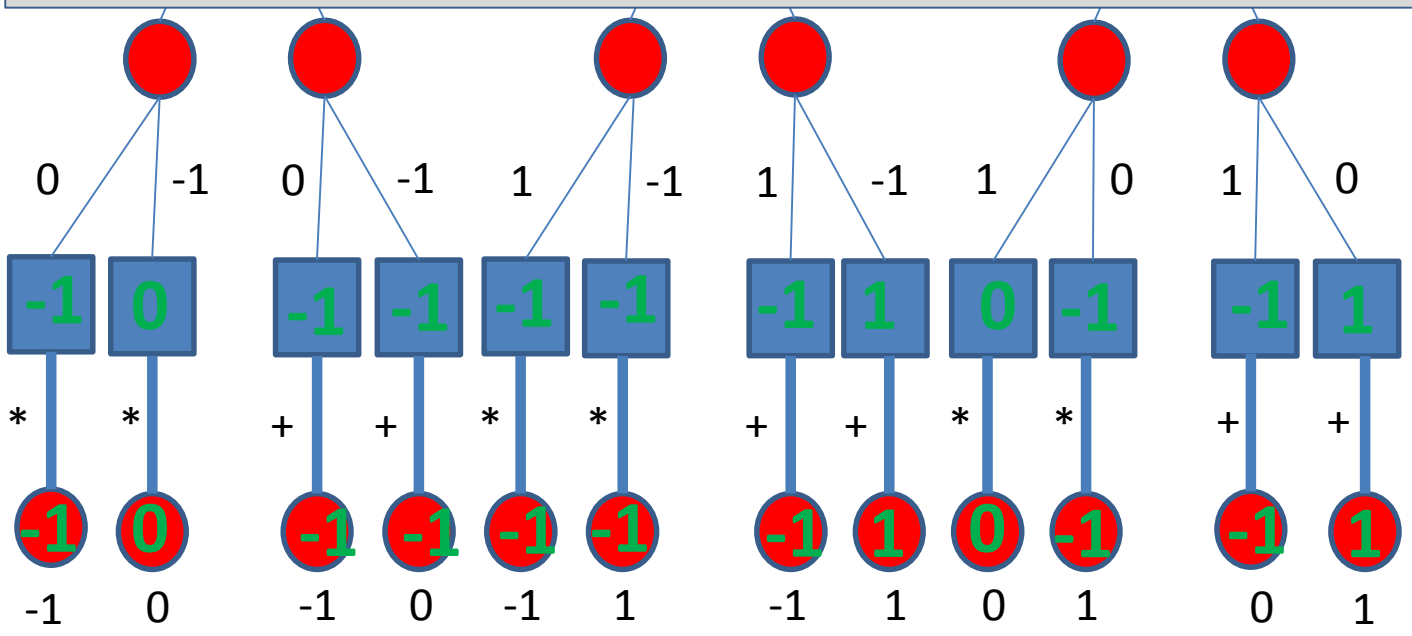


Now things get interesting! Player one has a choice of second moves.

He can see from the game tree what the result of the game will be depending on the move he makes. He will seek to maximize the final result.

We record which move player one chooses, and set the expected result at the state accordingly.

We'll break ties arbitrarily.

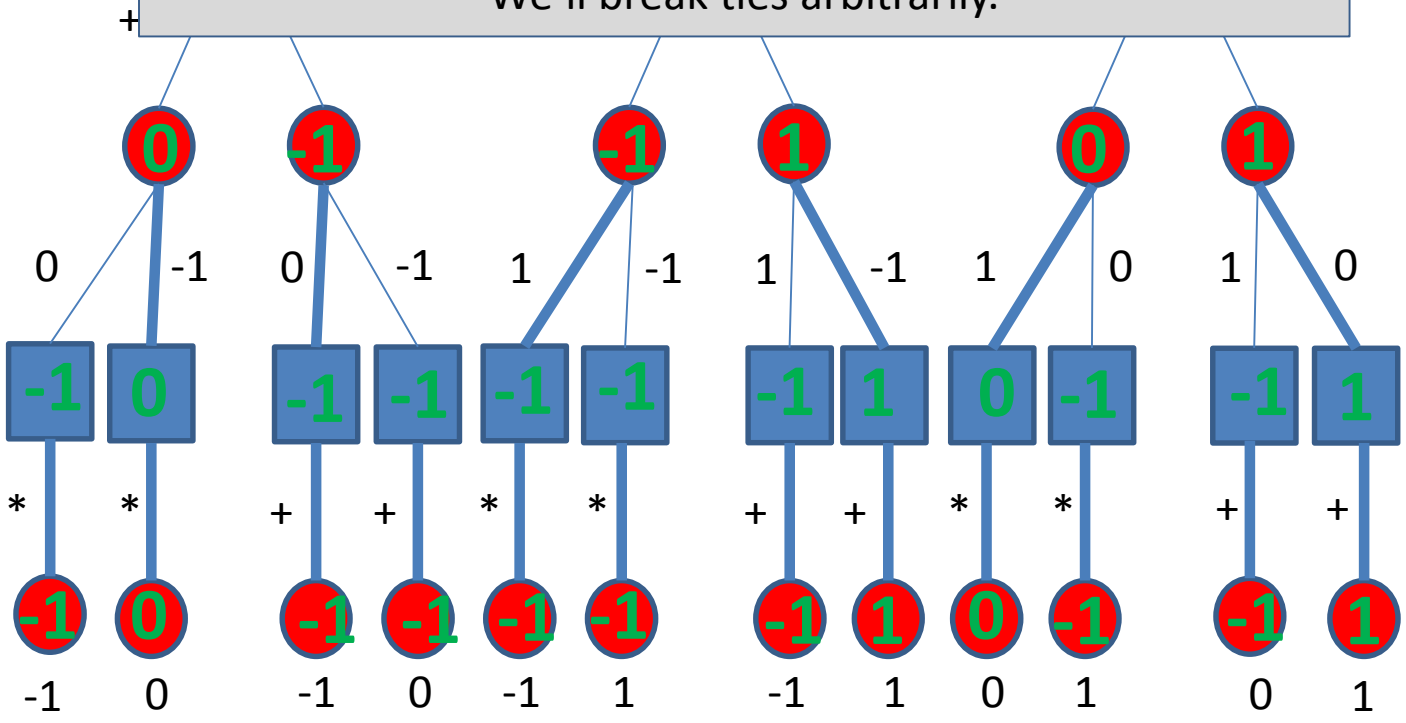


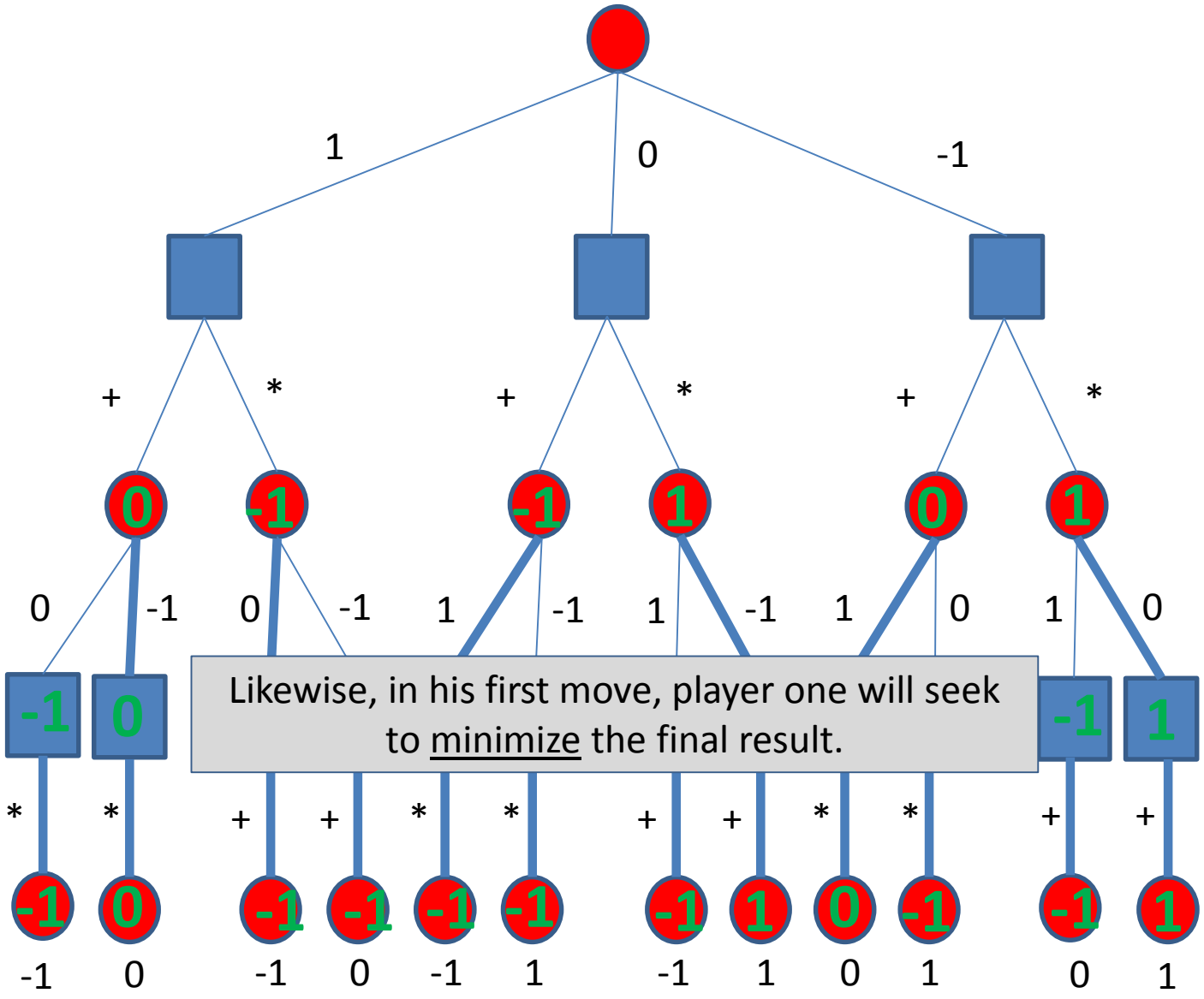
Now things get interesting!

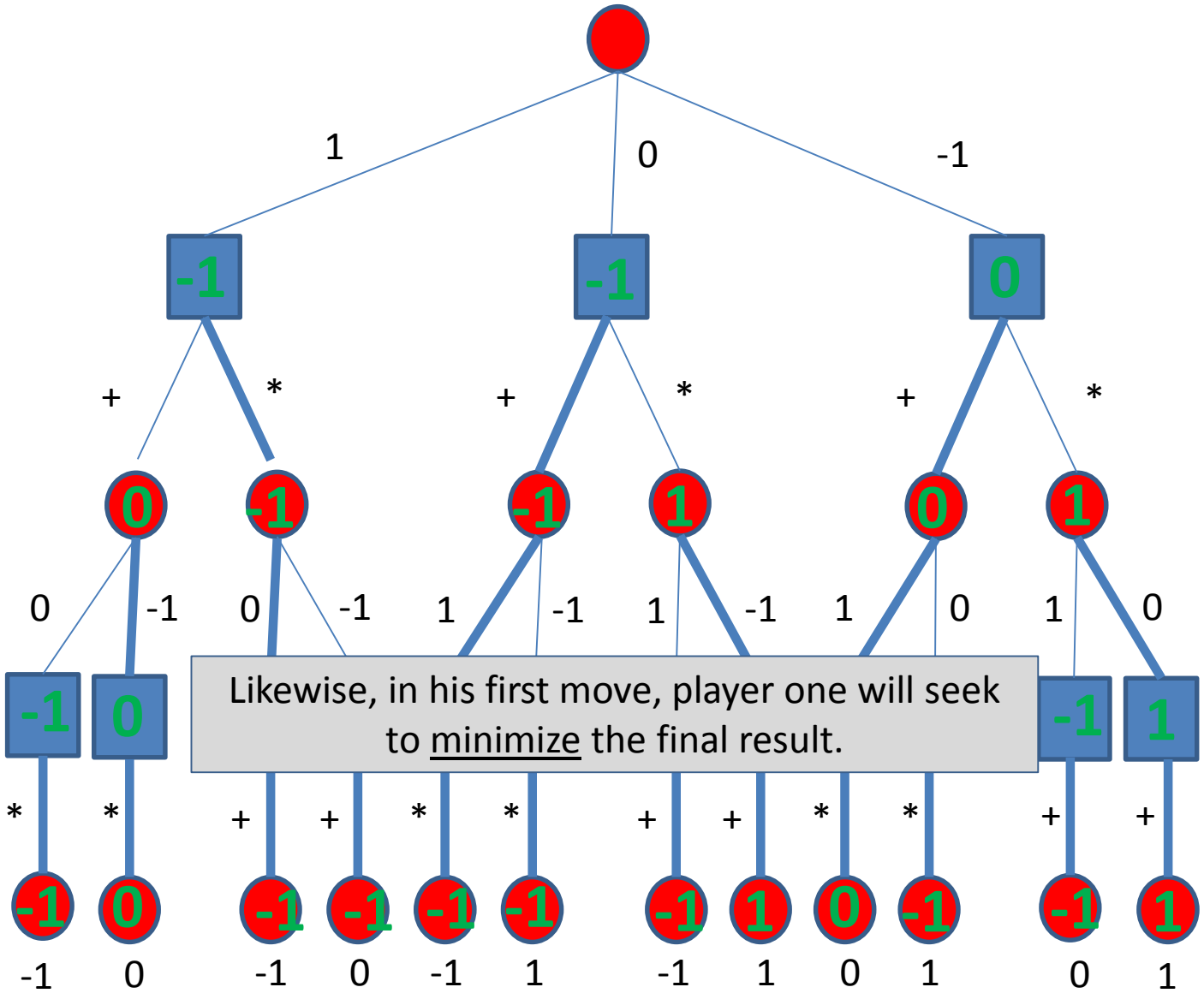
Since he has a choice of moves at his second move, player one will seek to maximize the final result.

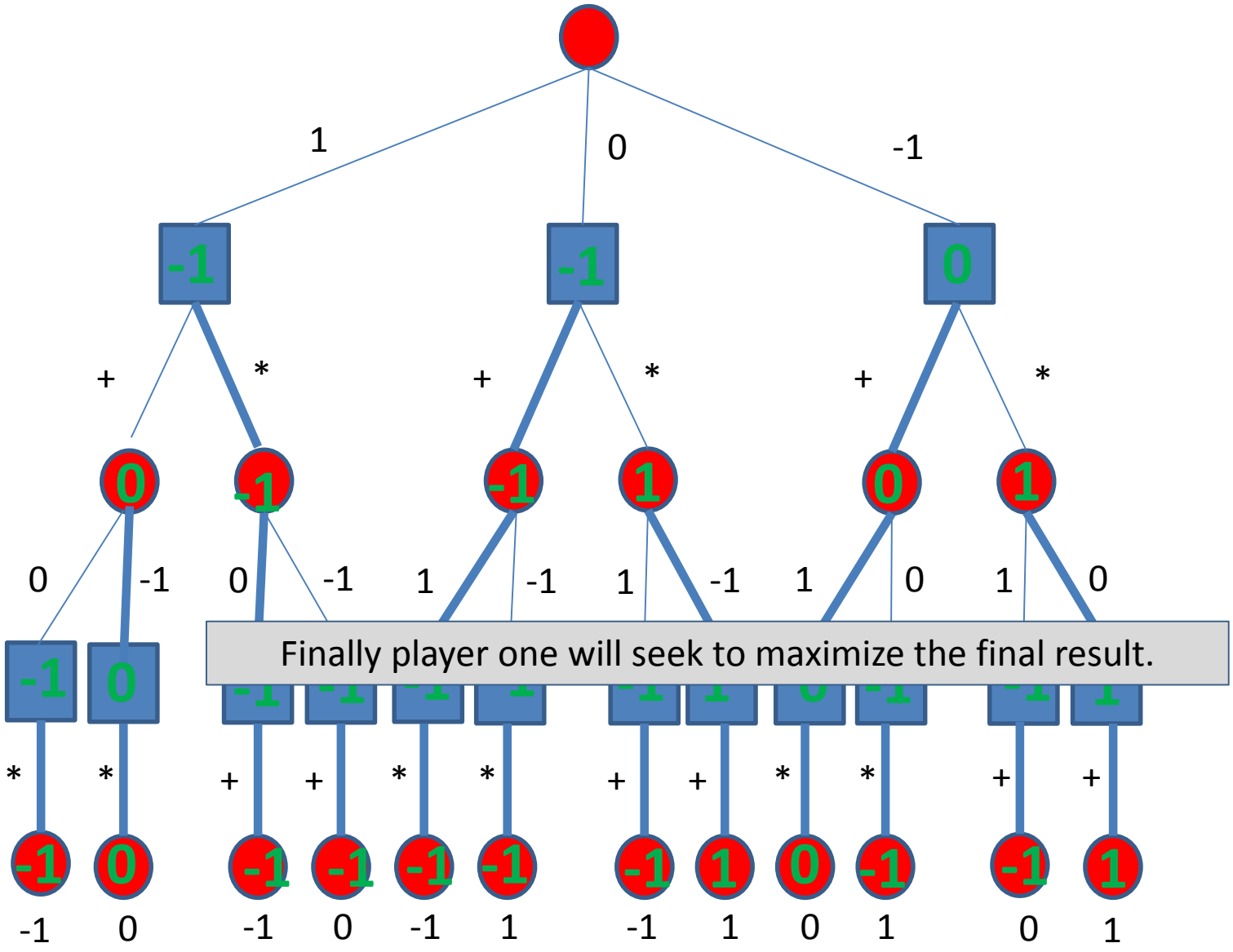
We record which move player one chooses, and set the expected result at the state accordingly.

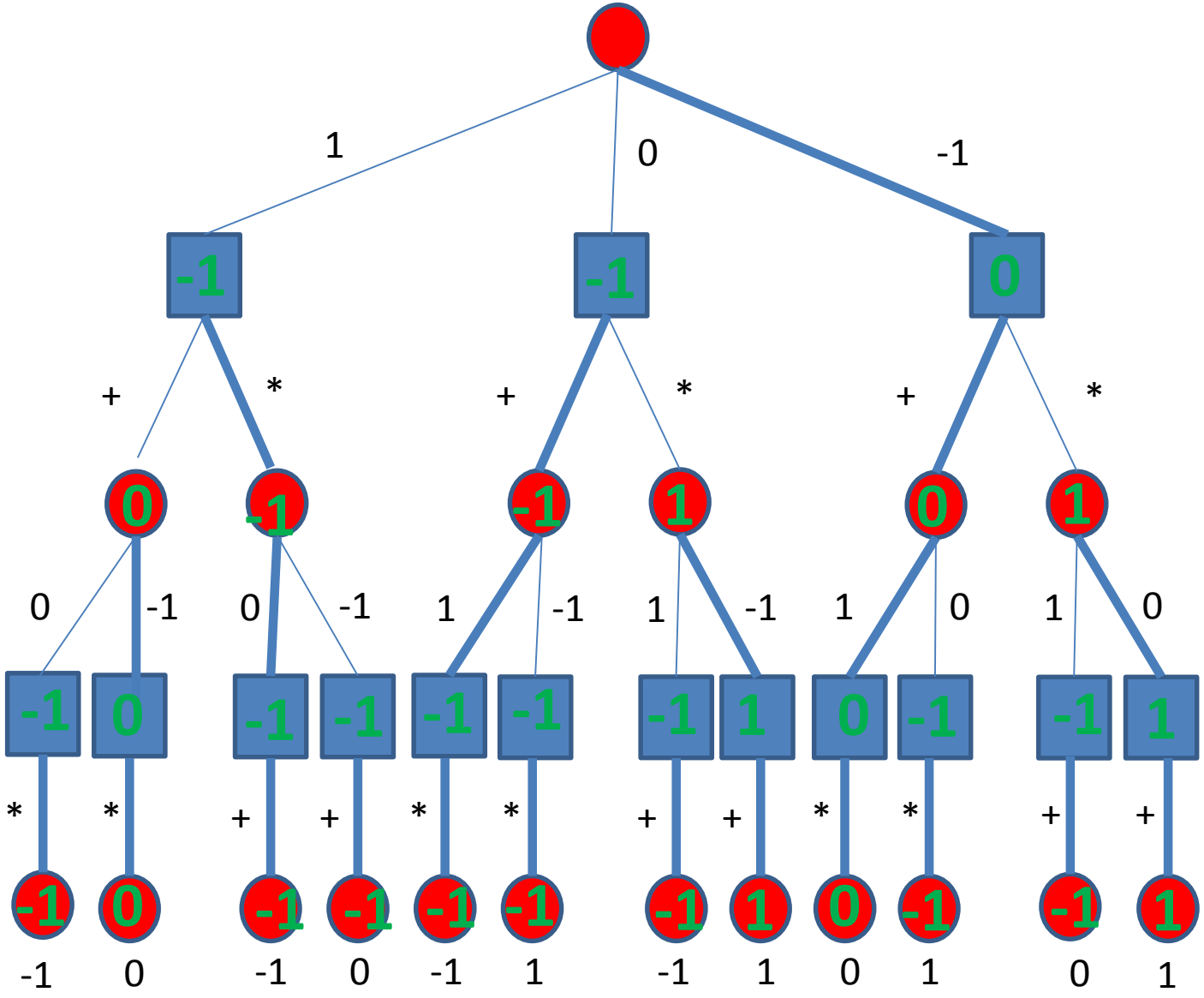
We'll break ties arbitrarily.

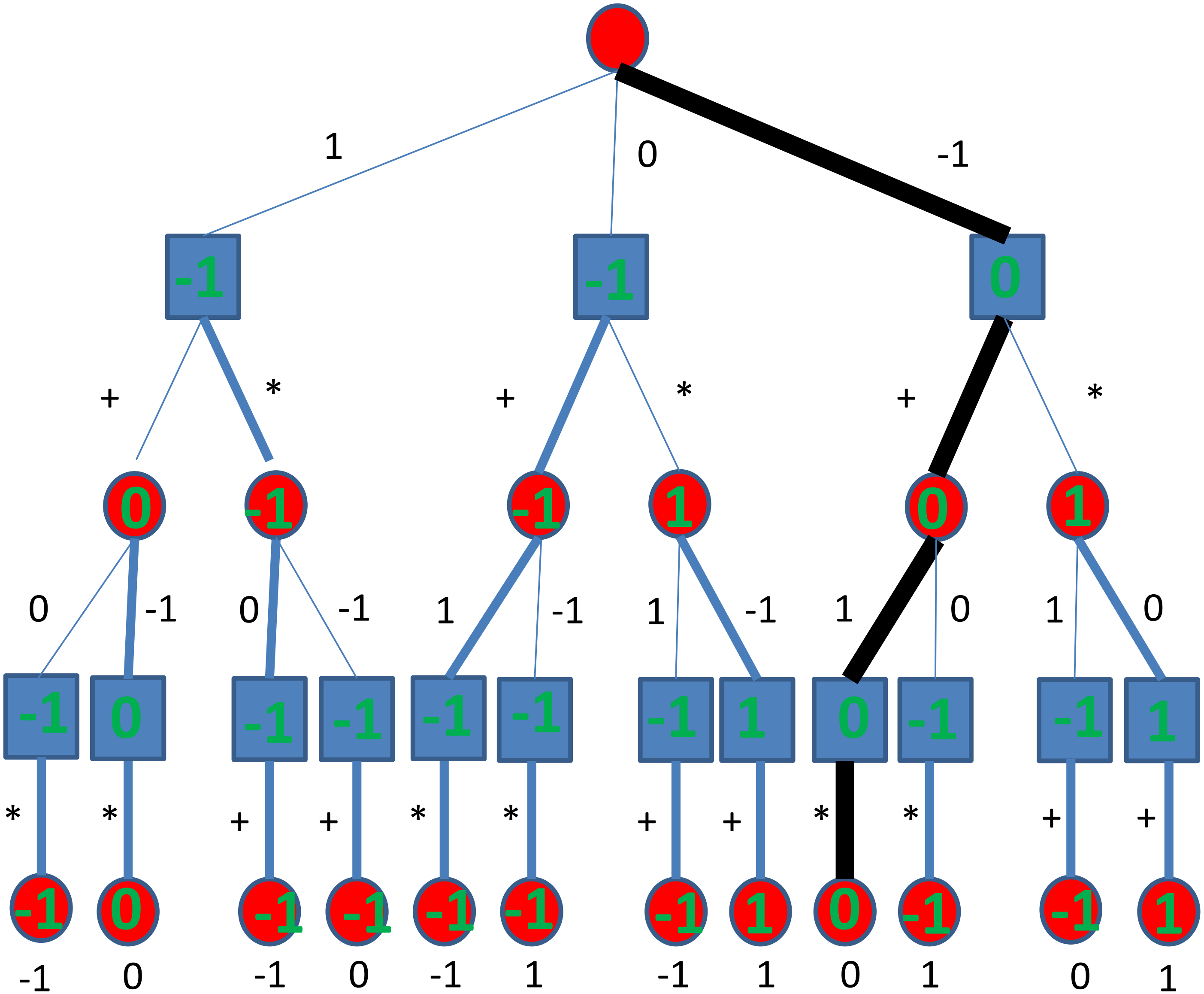



1
0
-1
+
*
0
-1
-1
1
0
1
0
-1
0
-1
1
-1
1
-1
1
0
1
0
-1
1
*
*
+
+
*
*
+
+
*
*
+
+
-1
0
-1
-1
-1
-1
-1
1
0
-1
-1
1
-1
0
-1
1
-1
1
0
1
Likewise, in his first move, player one will seek to minimize the final result.


1
0
-1
-1
-1
0
+
*
+
*
+
*
0
-1
-1
1
0
1
0
-1
0
-1
1
-1
1
-1
1
0
1
0
-1
0
-1
1
* * + + * * + + * * + +
-1 0 -1 -1 -1 -1 -1 1 0 -1 -1 1
-1 0 -1 0 -1 1 -1 1 0 1 0 1
Likewise, in his first move, player one will seek to minimize the final result.


1
0
-1
-1
-1
0
+
*
+
*
+
*
0
-1
-1
1
0
1
0
-1
0
-1
1
-1
1
-1
1
0
1
0
-1
0
-1
-1
0
-1
1
0
-1
1
-1
0
-1
1
-1
1
0
1
Finally player one will seek to maximize the final result.


1
0
-1
-1
-1
0
+
*
+
*
+
*
0
-1
-1
1
0
1
0
-1
0
-1
1
-1
1
-1
1
0
1
0
-1
0
-1
1
-1
1
-1
1
0
1
0
*
*
+
+
*
*
+
+
*
*
+
+
-1
0
-1
-1
-1
-1
-1
1
0
-1
-1
1
-1
0
-1
1
-1
0
-1
1



We see that if both players play optimally, this is the series of moves that will occur, and the game will result in a draw.

Minimax Algorithm

In this way we can find optimal strategies for each player.

We can also analyze the game:

- If both players perform optimal choices, we see the expected series of moves and that the game will result in a draw.

Minimax

In general:

- the number of nodes involved grows exponentially with depth
- the search space is too large to perform the complete algorithm.

So search is normally performed to some depth and a heuristic analysis of the state of the game is given at that depth.

The values obtained from this heuristic analysis is then the basis for the upwards choice of moves by minization/maximization.

- Heuristic:
  - Positive if state believed good for MAX
  - Negative if state believed good for MIN
  - Higher absolute values according to how good the state is for the relevant player.
- Heuristic domain dependent. In chess it might be the weighted value of the remaining pieces.

Alpha Beta Pruning

We can optimize the algorithm: At each state, only moves within a particular interval [alpha,beta] are interesting.

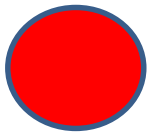
From MAX's point of view:

- Values below alpha: He has already found a better move.
- Values above beta: His opponent will never let the game reach the state required to play it.

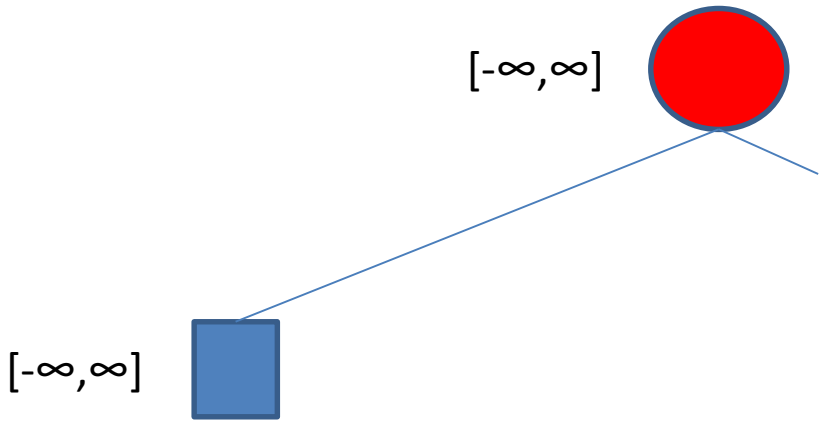
From MIN's point of view: Vice versa.

R code for alphabeta pruning

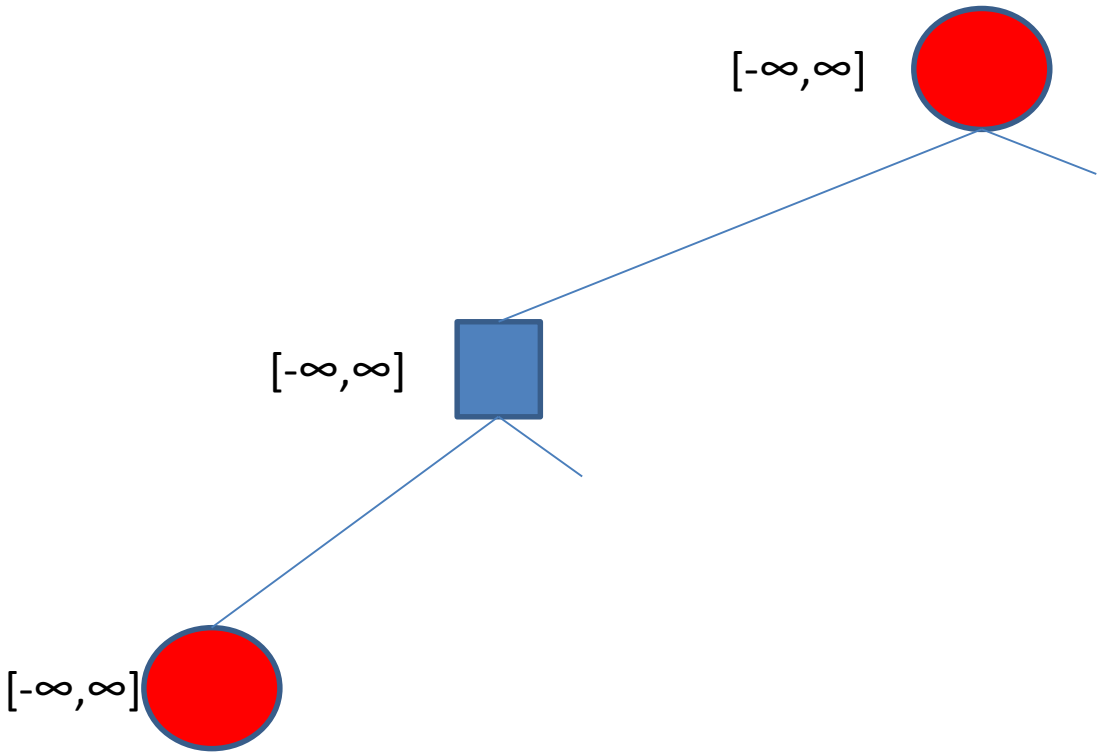
```
alphabeta = function (node, depth, alpha, beta) {
        if (leaf(node) || depth==0) return (evaluate(n))
        if (max(node)) {
                for (child in children(node)) {
                        v=alphabeta(child,d-1,alpha,beta)
                        if (v>alpha) alpha=v
                        if (alpha>=beta) return (alpha)
                }
                return (alpha)
        }
        if (min(node)) {
                for (child in children(node)) {
                        v=alphabeta(child,d-1,alpha,beta)
                        if (v<beta)) beta=v
                        if (alpha>=beta) return (beta)
                }
                return (beta)
        }
}
```

[-∞,∞] 

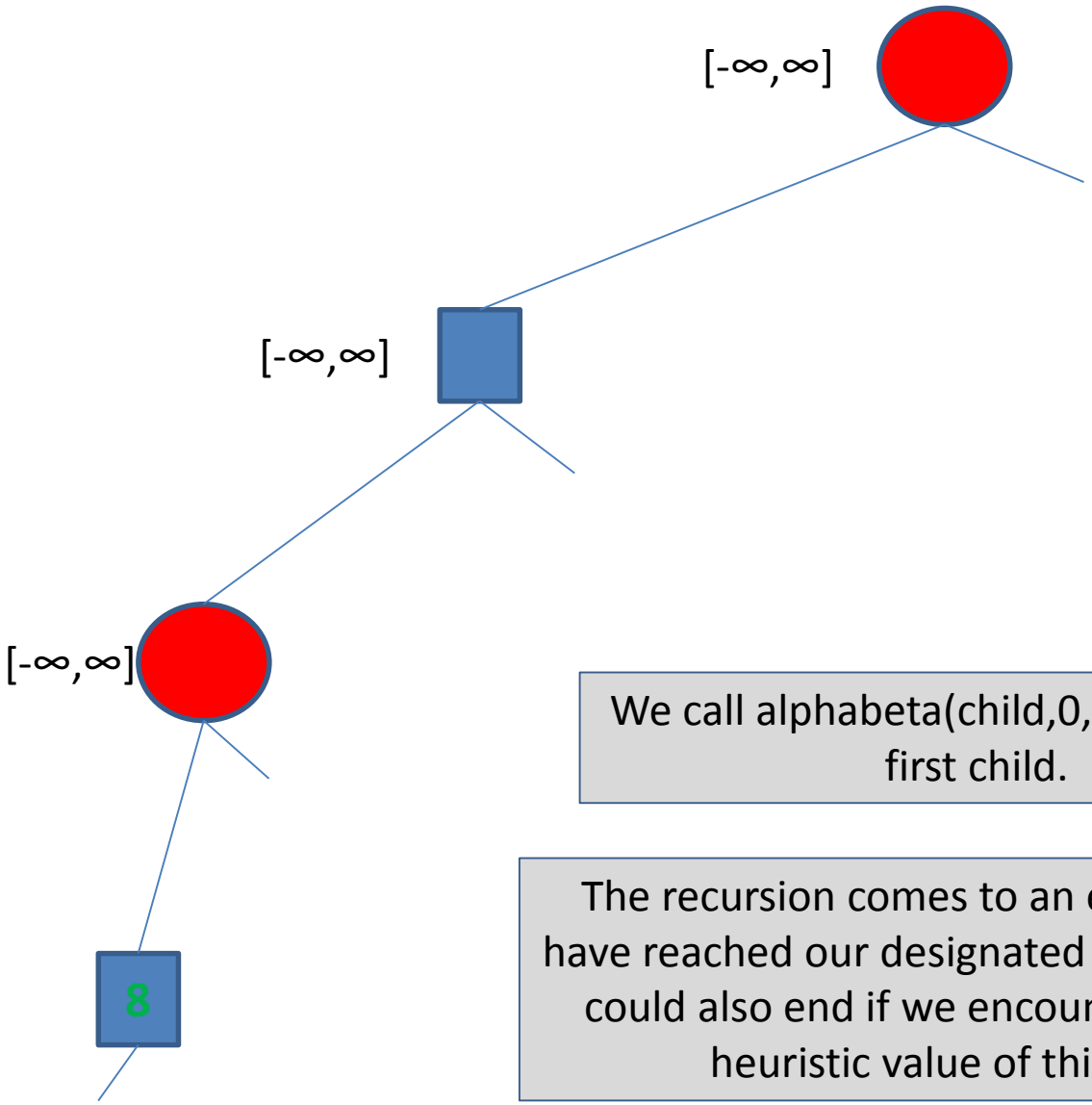
We call alphabeta(origin,3,-inf,inf)



We call alphabeta(child,2,-inf,inf) on the first child

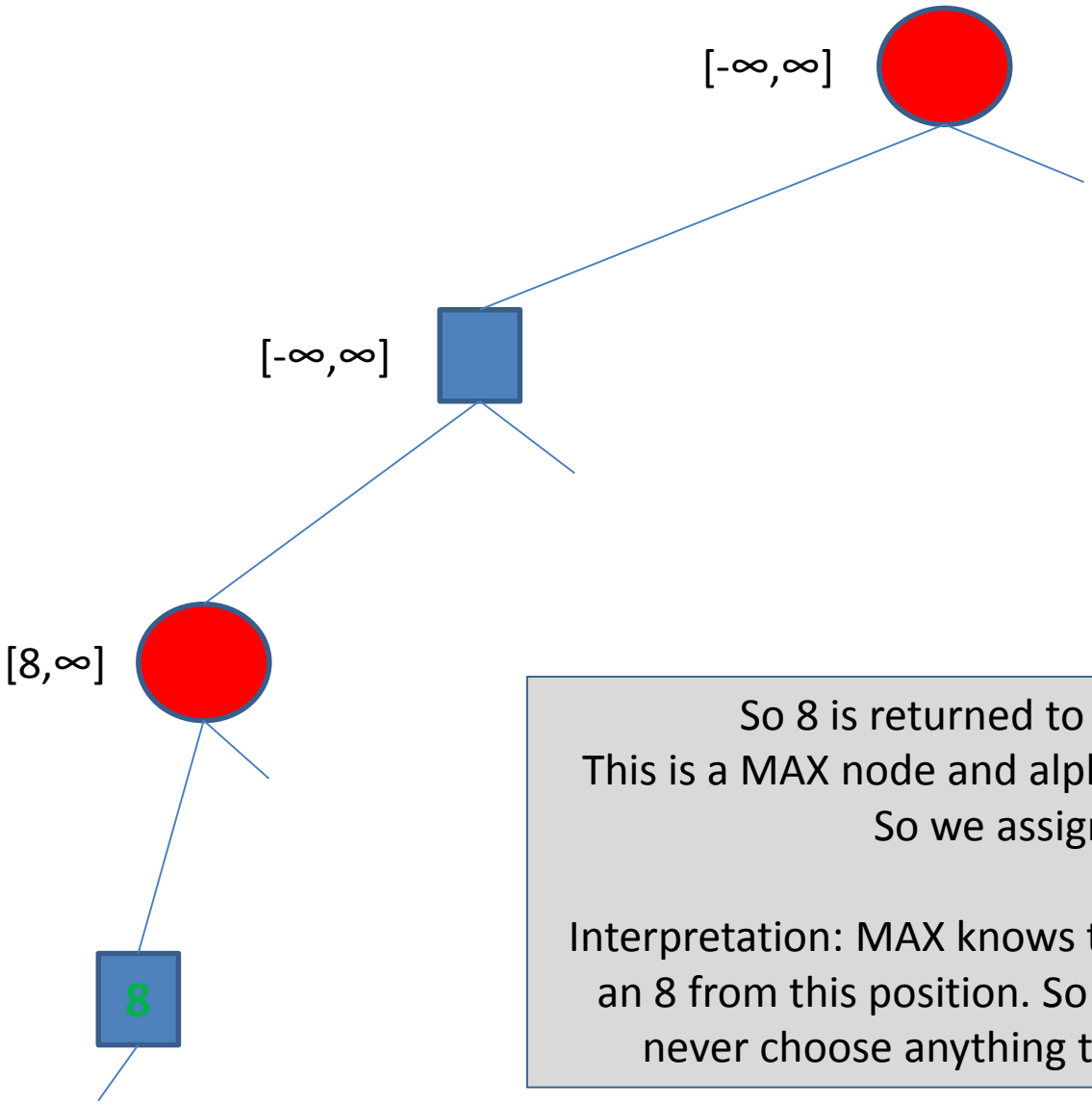


We call alphabeta(child,1,-inf,inf) on the first child's first child



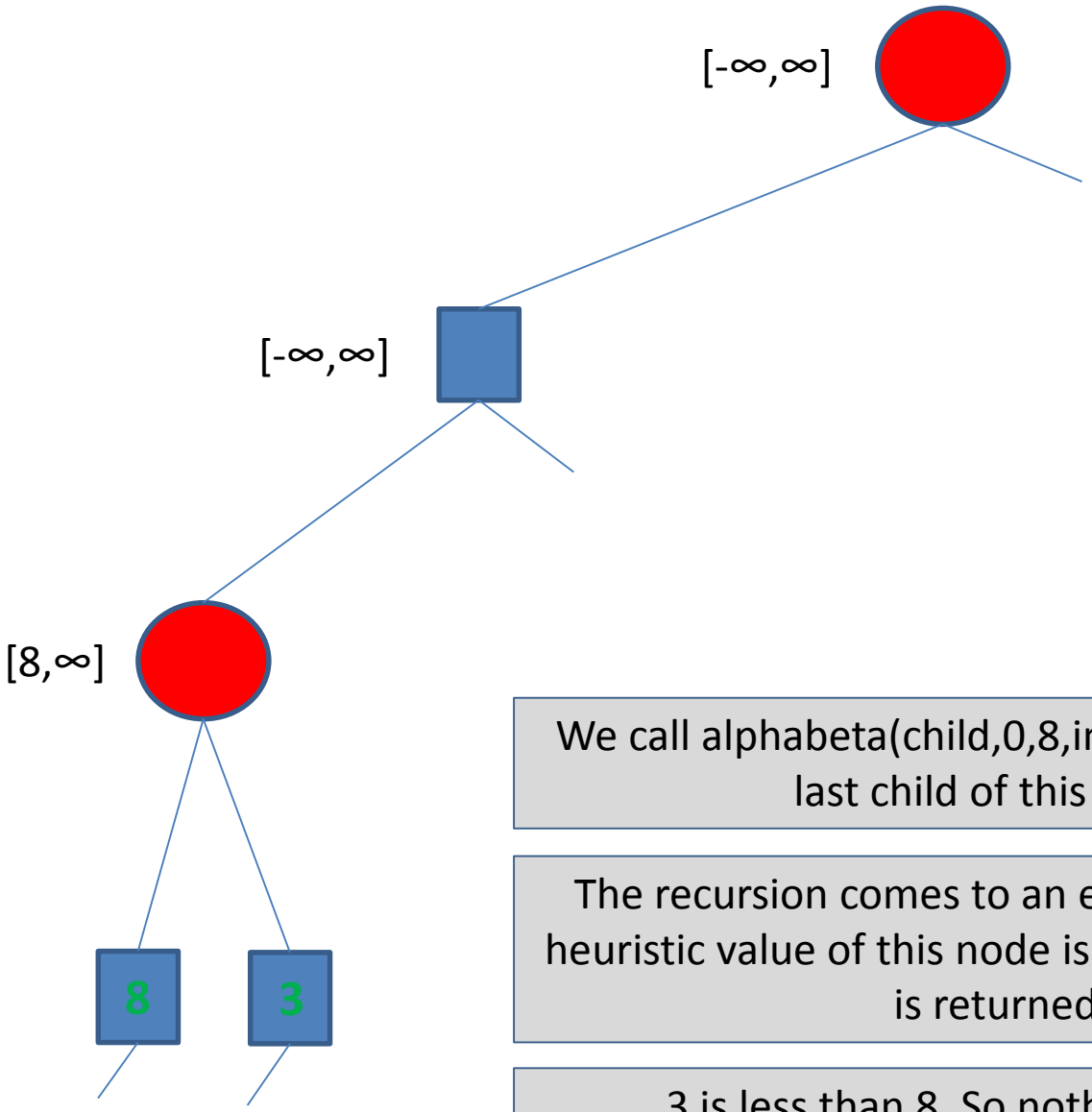
We call alphabeta(child,0,-inf,inf) on its first child.

The recursion comes to an end since d=0. We have reached our designated maximum depth. (It could also end if we encountered a leaf). The heuristic value of this node is 8.



So 8 is returned to the node's parent.
This is a MAX node and alpha is larger than alpha=-inf.
So we assign alpha = 8.

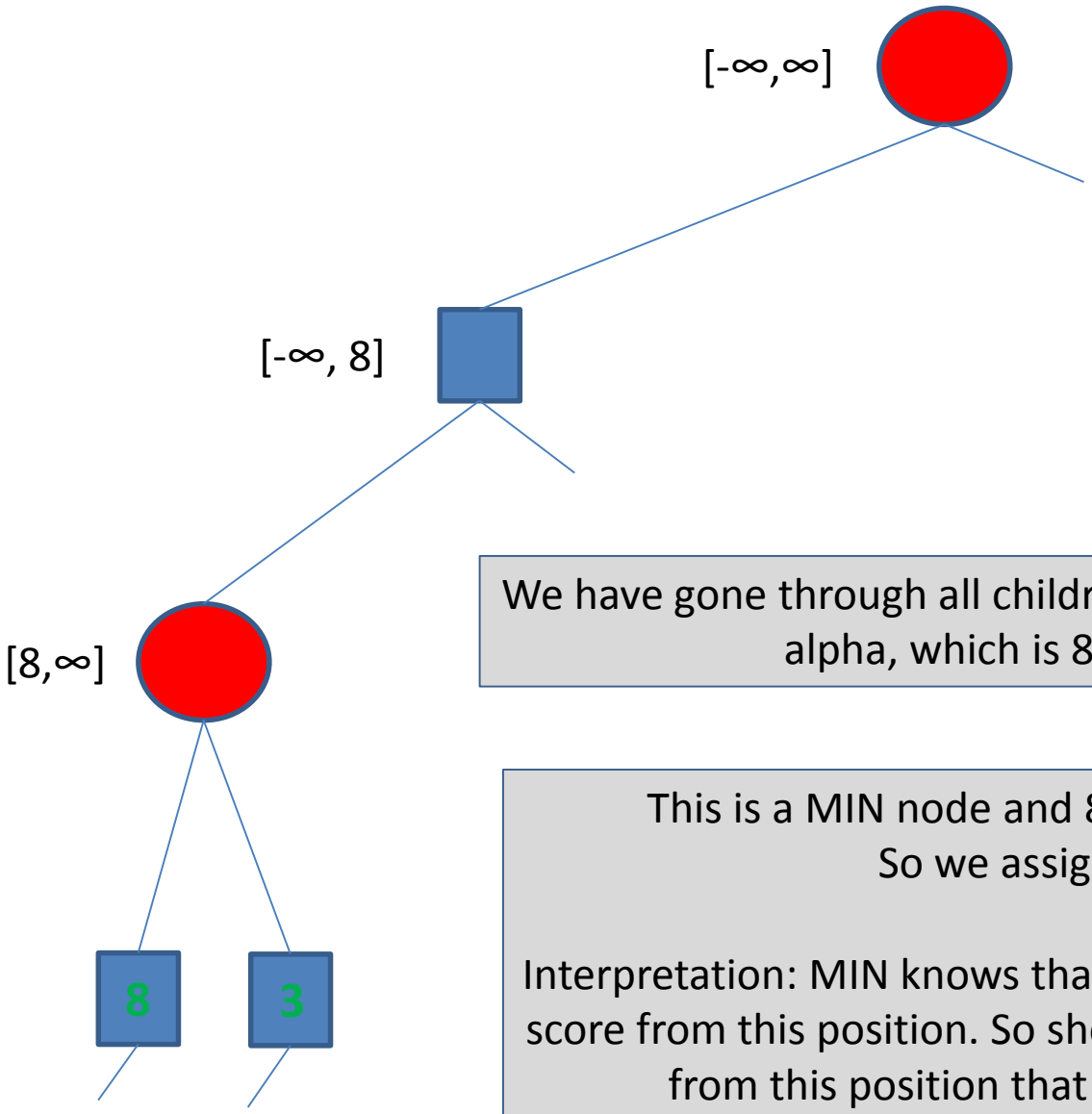
Interpretation: MAX knows that she can obtain at worst an 8 from this position. So from this position she will never choose anything that leads to less than 8.



We call alphabeta(child,0,8,inf) on the next and last child of this node.

The recursion comes to an end since d=0. The heuristic value of this node is 3, and so this value is returned.

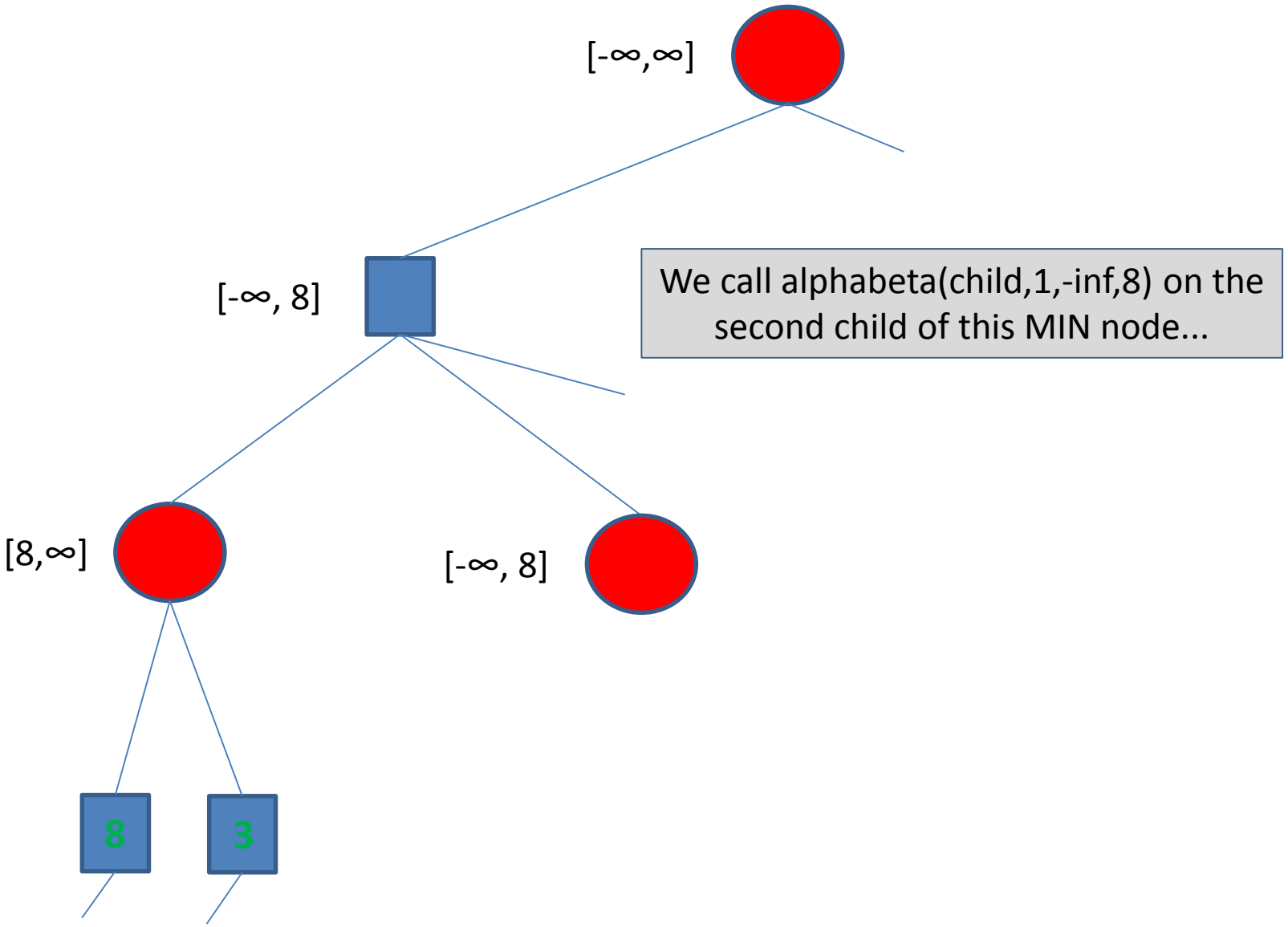
3 is less than 8. So nothing happens

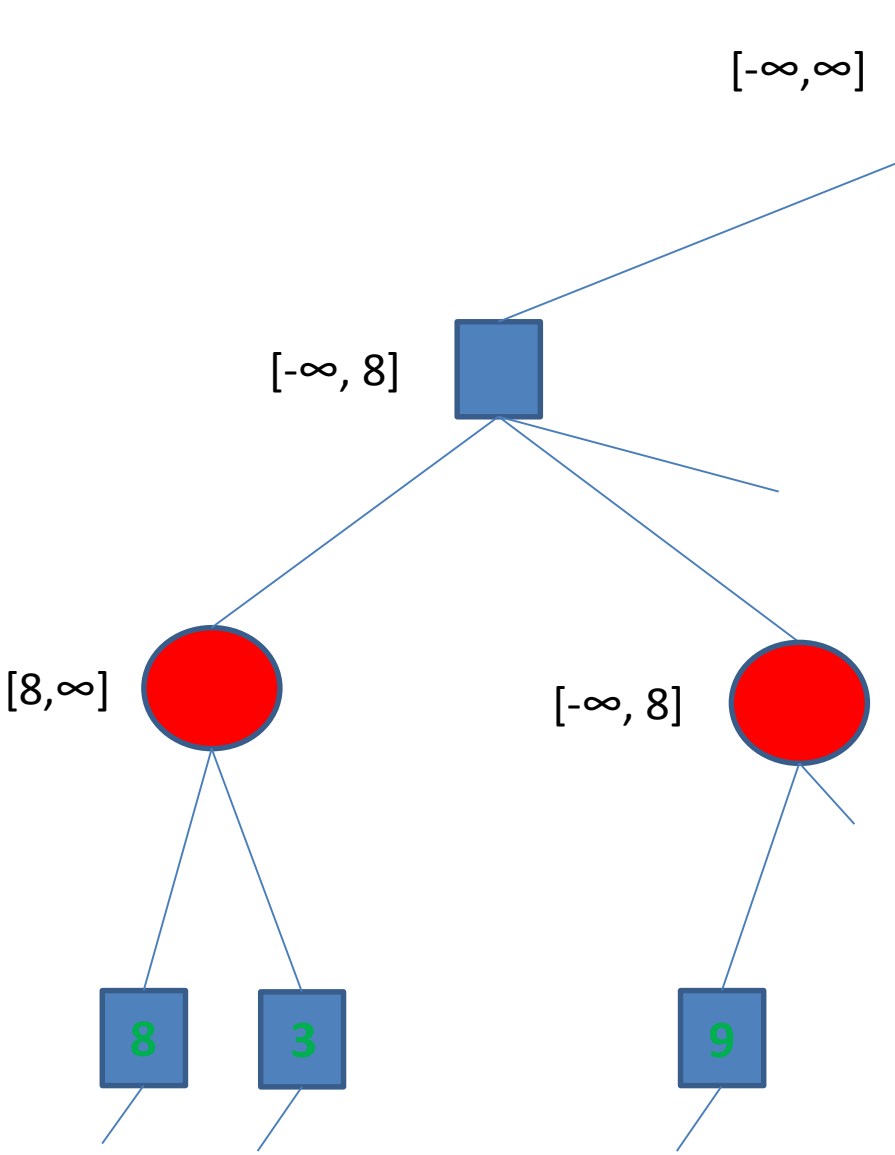


We have gone through all children of this node, so we return alpha, which is 8, to its parent.

This is a MIN node and 8 is less than beta = inf.
So we assign beta = 8.

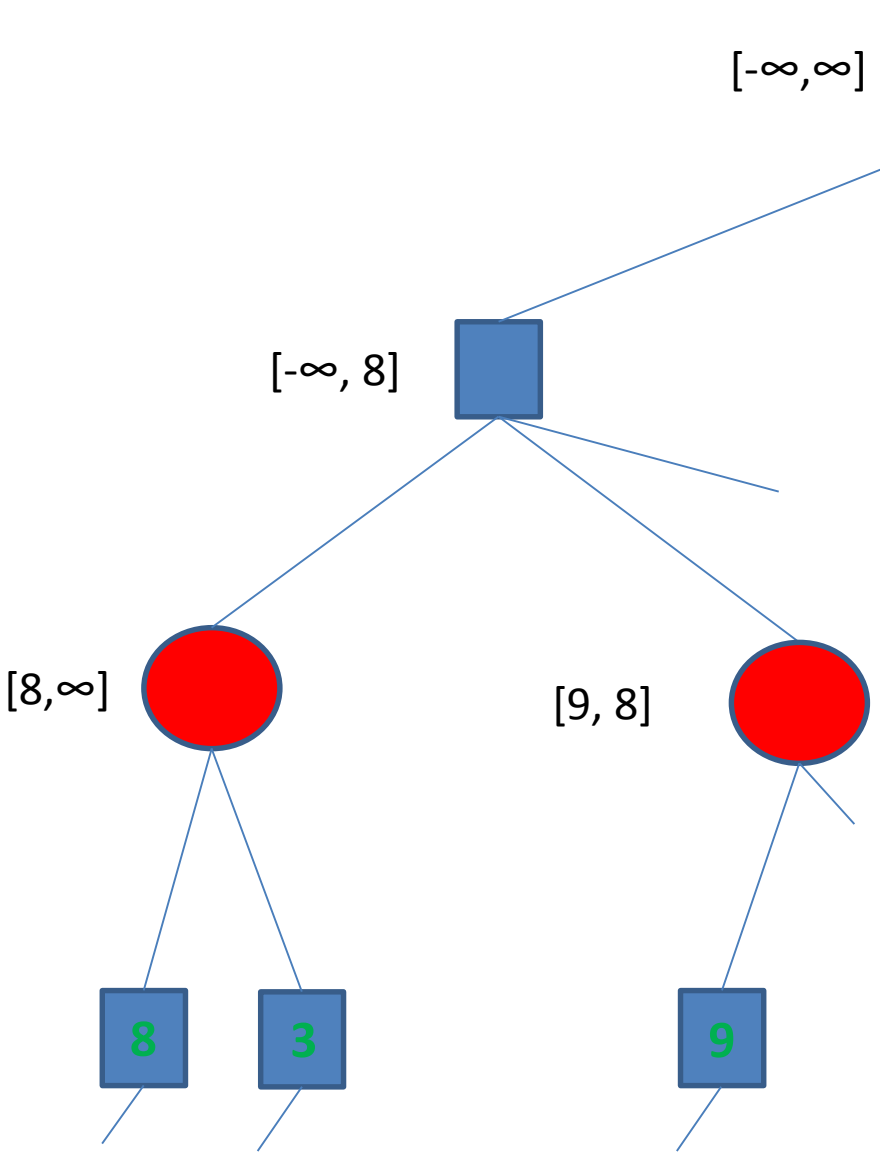
Interpretation: MIN knows that she can obtain at worst an 8 score from this position. So she will never choose any move from this position that leads to more than 8.


[-∞,∞]
[-∞, 8]
We call alphabeta(child,1,-inf,8) on the second child of this MIN node...
[8,∞]
[-∞, 8]
8
3



And then we call alphabeta(child,0,-inf,8) on it's child

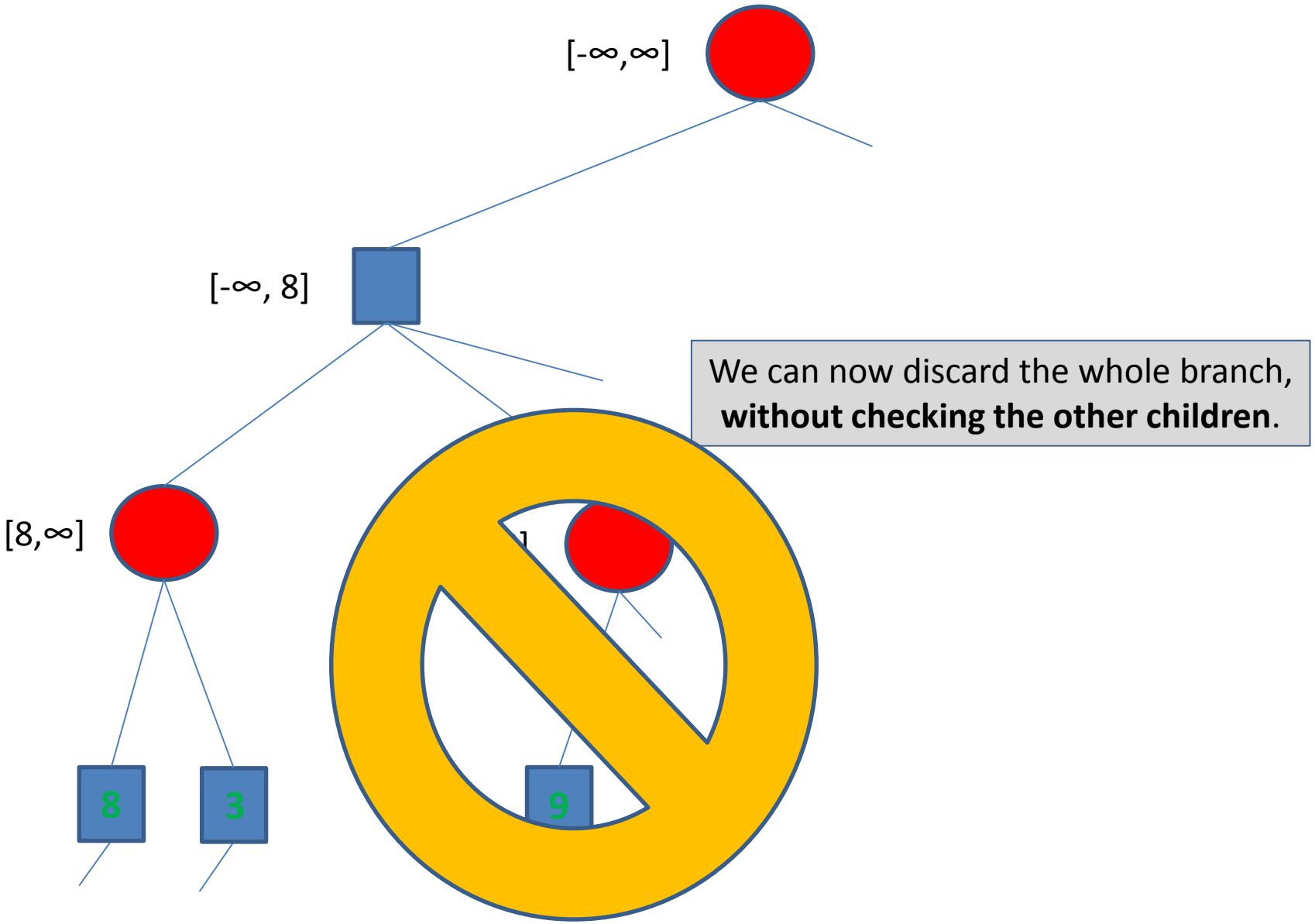
The recursion comes to an end since d=0. The heuristic value of this node is 9, and so this value is returned.



9 is bigger than alpha=–inf, so alpha is assigned 9.
Now alpha=9 is bigger than beta=8!

Interpretation: One of the players will ensure that this position never occurs.

In this case, it is MIN, since MIN can achieve 8 by choosing a different move, and this move results in MAX being able to force at least 9.


[-∞,∞]
[-∞, 8]
[8,∞]
8
3
9
We can now discard the whole branch,
without checking the other children.

Alpha Beta Pruning

- Increases the searchable depth by orders of magnitude.
- Selecting initial moves that return a small [alpha,beta] interval of possible moves can help enormously since these will rule out lots of following braches.
  - But we need some additional heuristic for selecting promising initial moves!

Issues

- These approaches does not permit the algorithm to ’wager’ that an opponent will make a mistake.
- It can suffer from horizen effects.
  - a bad thing will surely happen, but
  - you can postpone it beyond the search depth

    (Eg Sacrificing other pieces to put off having a doomed queen taken.)