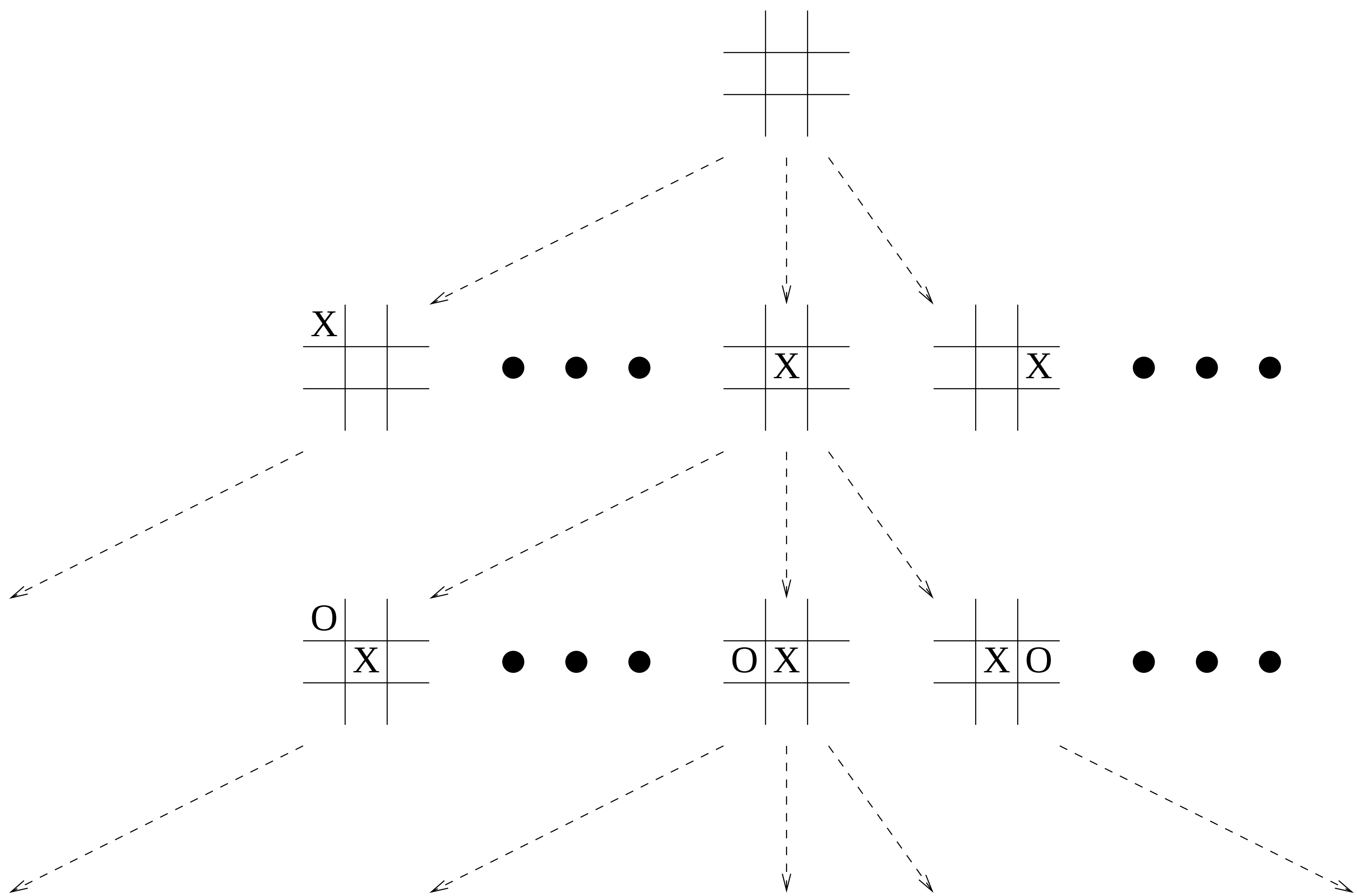
Game Trees

- Suppose that we have a two-player game in which the players take turns making moves.
  - Further, we assume that the game will always end in either (1) a victory by one player over the other or (2) a draw.
- We can model the game as a *multiway* tree.
  - Each node in the *game tree* corresponds to a possible configuration of the "game board".
  - The initial configuration of the "game board" is the root node.
  - A leaf node corresponds to an end-of-game configuration: a win, a loss, or a draw.

Game Trees: An Example



Game Trees: An Example (cont.)

- Assume the players are John and Mary.
- When John is to make a move, he will move to a game node that is best for him.
  - What is best for John is based on some numerical value that he associates with each of the possible next moves.
    - How are the value of each of the game node calculated?
      - What follows is one such possible computation.

Min-Max Algorithm

- For each *leaf* node X, John can assign a value of 1 to X if he wins, 0 if he ties, and -1 if he loses.
  - Conceptually, John defines a "pay-off" function P as follows:

```
            1 if John wins
   P(X) =   0 if John ties
           -1 if John loses

     where X is a leaf.
```

Min-Max Algorithm (cont.)

- John assigns a value to a *non-leaf* node X, as follows:

```
V(X) =
   max { V(c) | c is a child node of X }
      if John moves next

   min { V(c) | c is a child node of X }
      if Mary moves next

  where X is not a leaf.
```

  - The idea is that John would move to a node that has the maximum value for him, and Mary would do her best by moving to a node that has the minimum value (from John's perspective).

Min-Max Algorithm (cont.)

- In general, it is not possible to examine all leaf nodes of a non-trivial game.
  - At best, you can examine the game tree up to certain depth.

Modified Min-Max Algorithm

- Instead of flip-flopping between max and min as described above, we can reformulate the min-max strategy based on the simple mathematical formula:

```
max(a, b) = -min(-a, -b)
```

Modified Min-Max Algorithm (cont.)

- Let

```
E(n) be the pay-off function that John uses to evaluate
  a game node n.
```

- Let

```
e(n) =  E(n) if n is node from which John is
             to make the next move

       -E(n) if Mary is to make the next move.
```

Modified Min-Max Algorithm (cont.)

- Let

```
ModMinMax(x) = e(x), if x is a leaf of the game subtree

               max(-ModMinMax(c)), if x is not a leaf
                 of the game subtree and c ranges over all
                 of the (immediate) children of x.
```

Modified Min-Max Algorithm (cont.)

- Given x, a game tree node (i.e., a game board configuration), and d, the number of lookahead moves from x, compute the value of x based on the min-max formula:

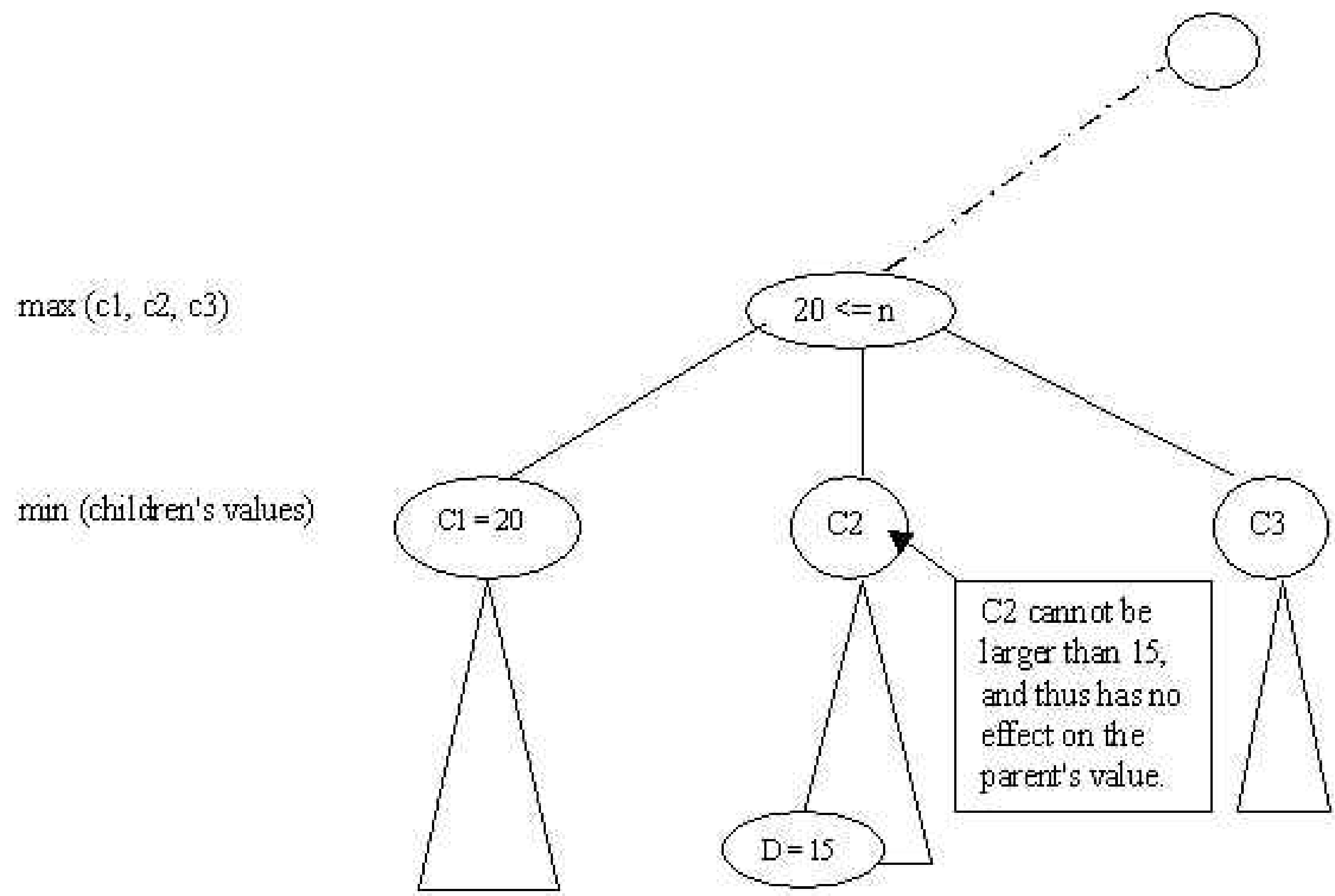
```
int ModMinMax(GameNode x, int d)

  if x is a leaf node or d == 0
    then return e(x)
    else // c[0] is the first child node of x.
      ans = -ModMinMax(c[0], d - 1)
      for (int i = 1; i < number of children of x; i++)
        temp = -ModMinMax(c[i], d - 1);
        if ans < temp
          then ans = temp;
  return ans;
```

Alpha-Beta Pruning

- In computing the min-max value of a game tree node, we can skip ("prune") the evaluation of some of the node's children.
- Let alpha be a lower bound for the value of a max node A, and let B be a child node of A.
  - If the value v of a child of B is less or equal to alpha, then we can use v as a value for B and skip the rest of the children of B.
    - This is called "alpha pruning".

Alpha Pruning

- In the figure below, alpha = 20, and we can prune the rest of the children of C2, once the value of D is (recursively) computed.



Beta Pruning

- Let beta be an upper bound for the value of a min node B, and let C be a child node of B.
  - If the value v of a child of C is greater or equal to beta, then we can use v as a value for C and skip the rest of the children of C.
    - This is called "beta pruning".