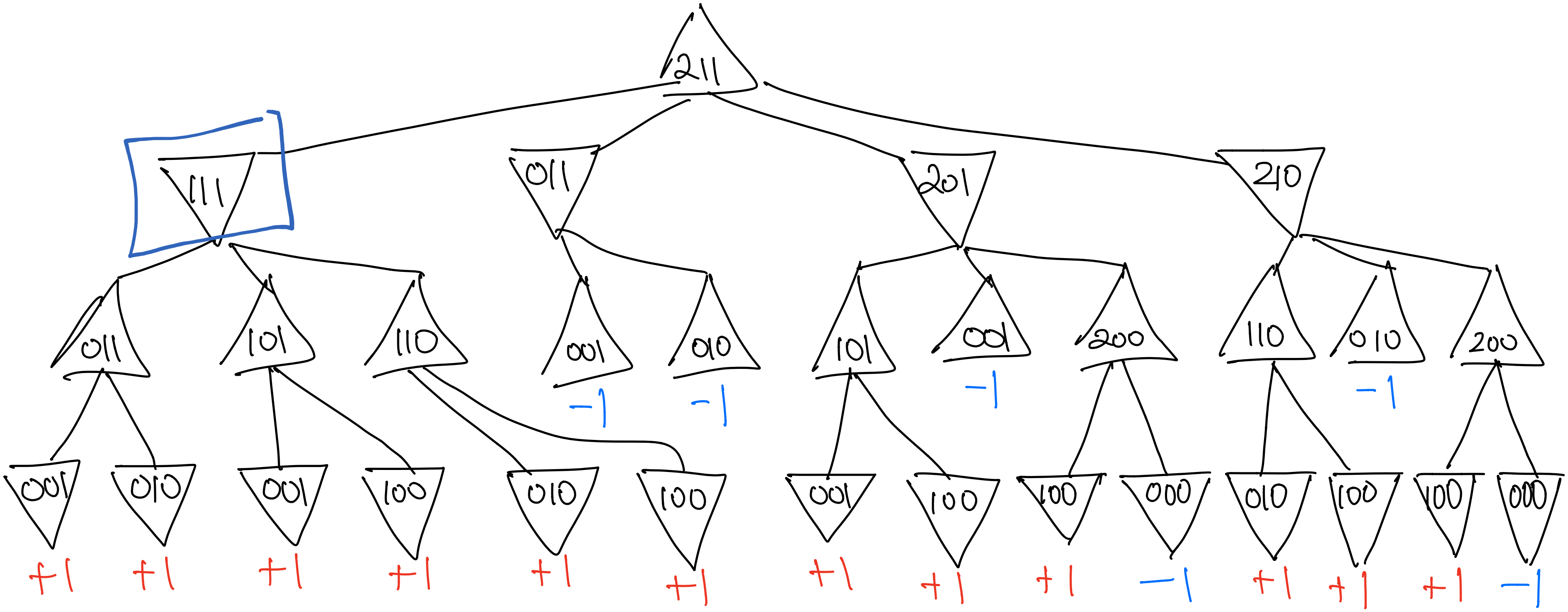
Real-world use of alpha-beta

- (Regular) minimax is normally run as a preprocessing step to find the optimal move from every possible situation.
- Minimax with alpha-beta can be run as a preprocessing step, but might have to re-run during play if a non-optimal move is chosen.
- Save states somewhere so if we re-encounter them, we don't have to recalculate everything.

Real-world use of alpha-beta

- States get repeated in the game tree because of *transpositions*.
- When you discover a best move in minimax or alpha-beta, save it in a lookup table (probably a hash table).
  - Called a *transposition table*.


211
111
011
201
210
011
101
110
001
010
101
001
200
110
010
200
−1
−1
−1
−1
001
010
001
100
010
100
001
100
100
000
010
100
100
000
+1
+1
+1
+1
+1
+1
+1
+1
+1
−1
+1
+1
+1
−1

Real-world use of alpha-beta

- In the real-world, alpha-beta does not "pre-generate" the game tree.
  - The whole point of alpha-beta is to not have to generate all the nodes.
- The DFS part of minimax/alpha-beta is what generates the tree.

Summary so far

- Minimax: Find the best move for each player, assuming the other player plays perfectly.
    - Based on DFS; searches the whole game tree.
    - Usually used as a preprocessing step (too slow for real time).
- Alpha-beta: Always gives same result as minimax, but prunes sub-optimal branches.
    - Can be used to preprocess game tree, but sub-optimal moves will necessitate rerunning.
    - Can be used in real time, but often still too slow.

Improving on alpha-beta

- Alpha-beta still must search down to terminal nodes sometimes.
  - (and minimax has to search to terminal nodes all the time!)
- Improvement idea: can we get away with only looking a few moves ahead?

Heuristic minimax algorithm

REGULAR MINIMAX

```
minimax(s) =
  utility(s, MAX)                                           if is-terminal(s)
  $\max_{a \text{ in actions(s)}}$ minimax(result(s, $a$))    if to-move(s)=MAX
  $\min_{a \text{ in actions(s)}}$ minimax(result(s, $a$))    if to-move(s)=MIN
```

HEURISTIC MINIMAX

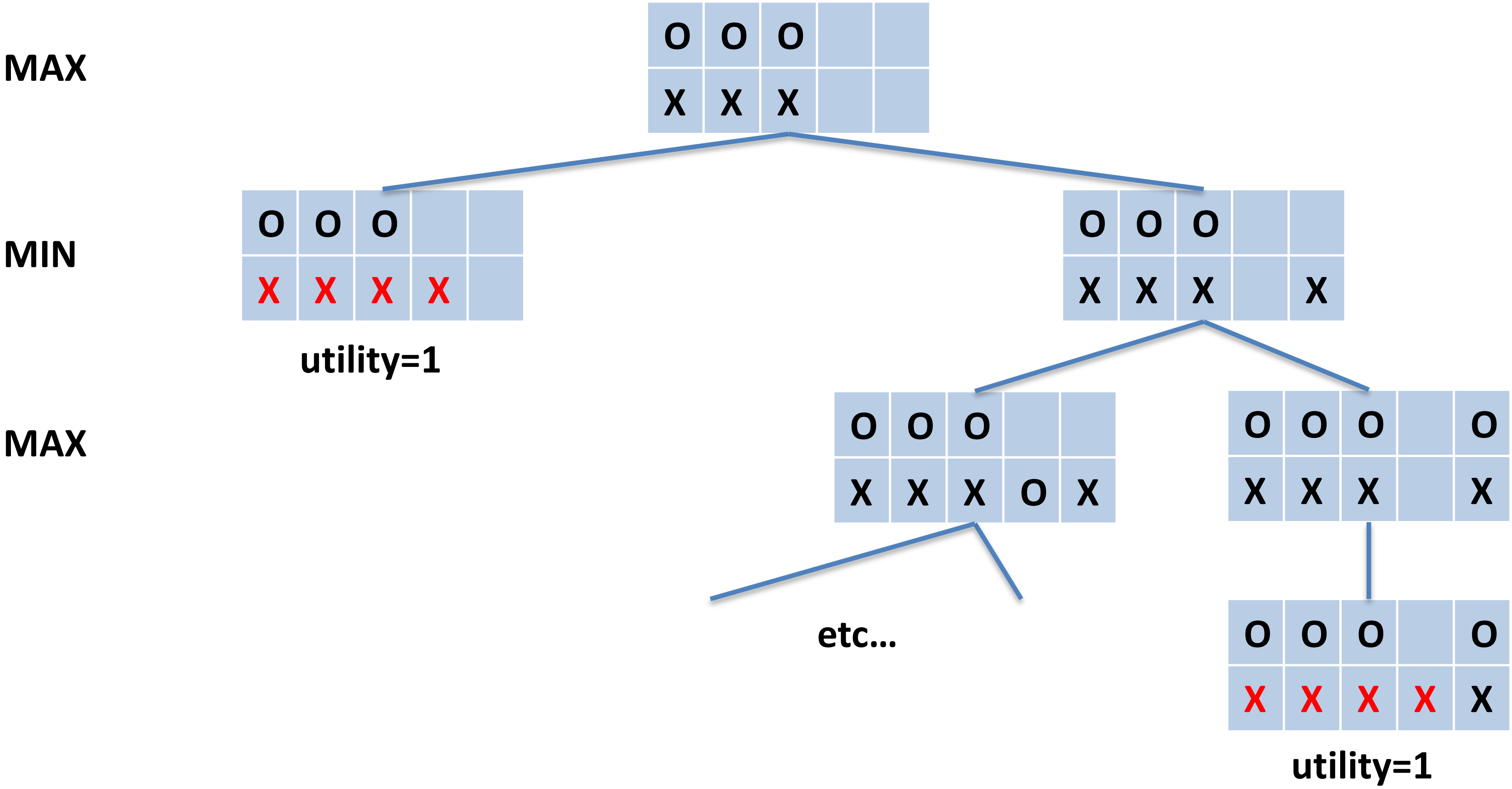
```
h-minimax(s, d) =
  eval(s, MAX)                                                    if is-cutoff(s, d)
  $\max_{a \text{ in actions(s)}}$ h-minimax(result(s, $a$), d+1)   if to-move(s)=MAX
  $\min_{a \text{ in actions(s)}}$ h-minimax(result(s, $a$), d+1)   if to-move(s)=MIN
```

result(s, a) means the new state generated
by taking action *a* in state *s*.
is-cutoff(s, d) is a boolean test that determines whether
we should stop the search and evaluate our position.

How to create a good evaluation function?

- Trying to judge the probability of winning from a given state.
- Typically use features: simple characteristics of the game that correlate well with the probability of winning.

One last point


MAX
MIN
MAX
O O O
X X X
O O O
X X X X
utility=1
O O O
X X X X
O O O
X X X O X
O O O O
X X X X
etc…
O O O O
X X X X X
utility=1