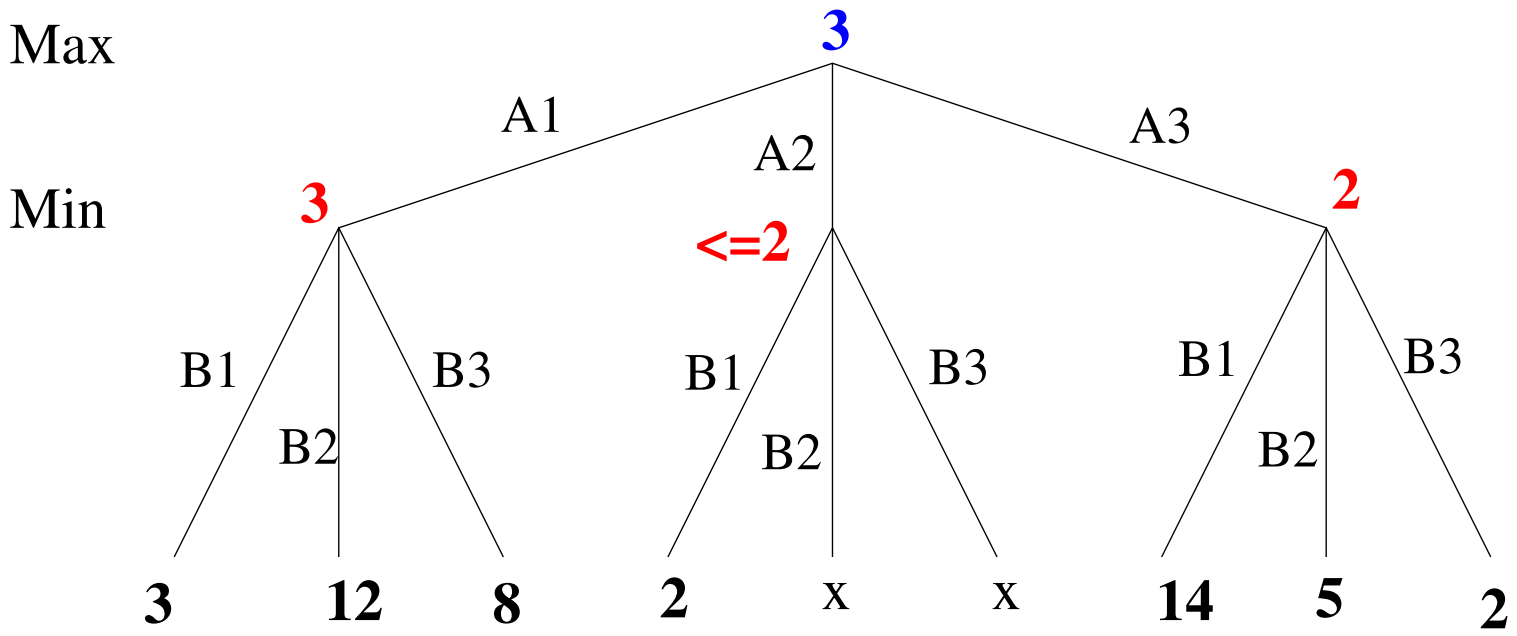
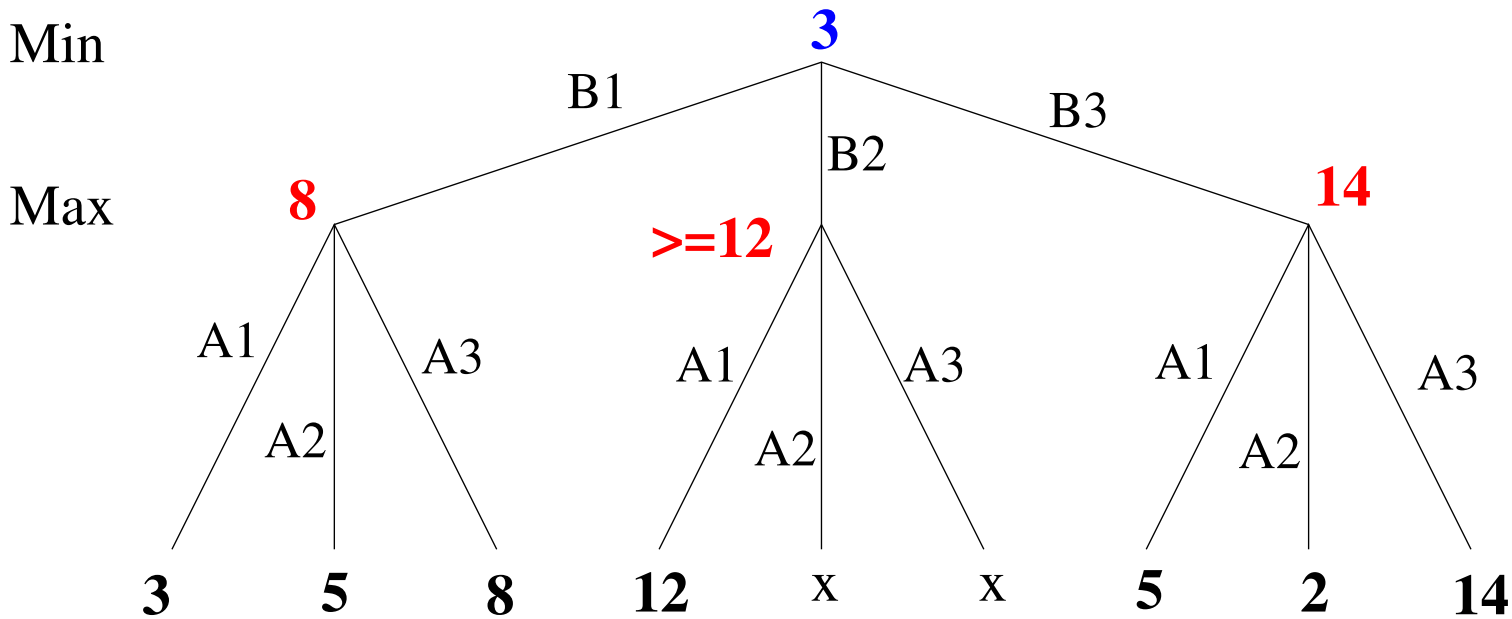
Alpha-Beta pruning: explanation

- minimax search can be wasteful
- a node n_1 at depth d has a known minimax value of m_1
- another node n_2 at the same depth has a decendant with utility $m_2 < m_1$
- this means that Min will always prefer n_1 to n_2
- it also means that Max will never prefer n_1 to n_2
- in such a case, there is no point expanding the rest of n_2
- similar argument can be made for Min

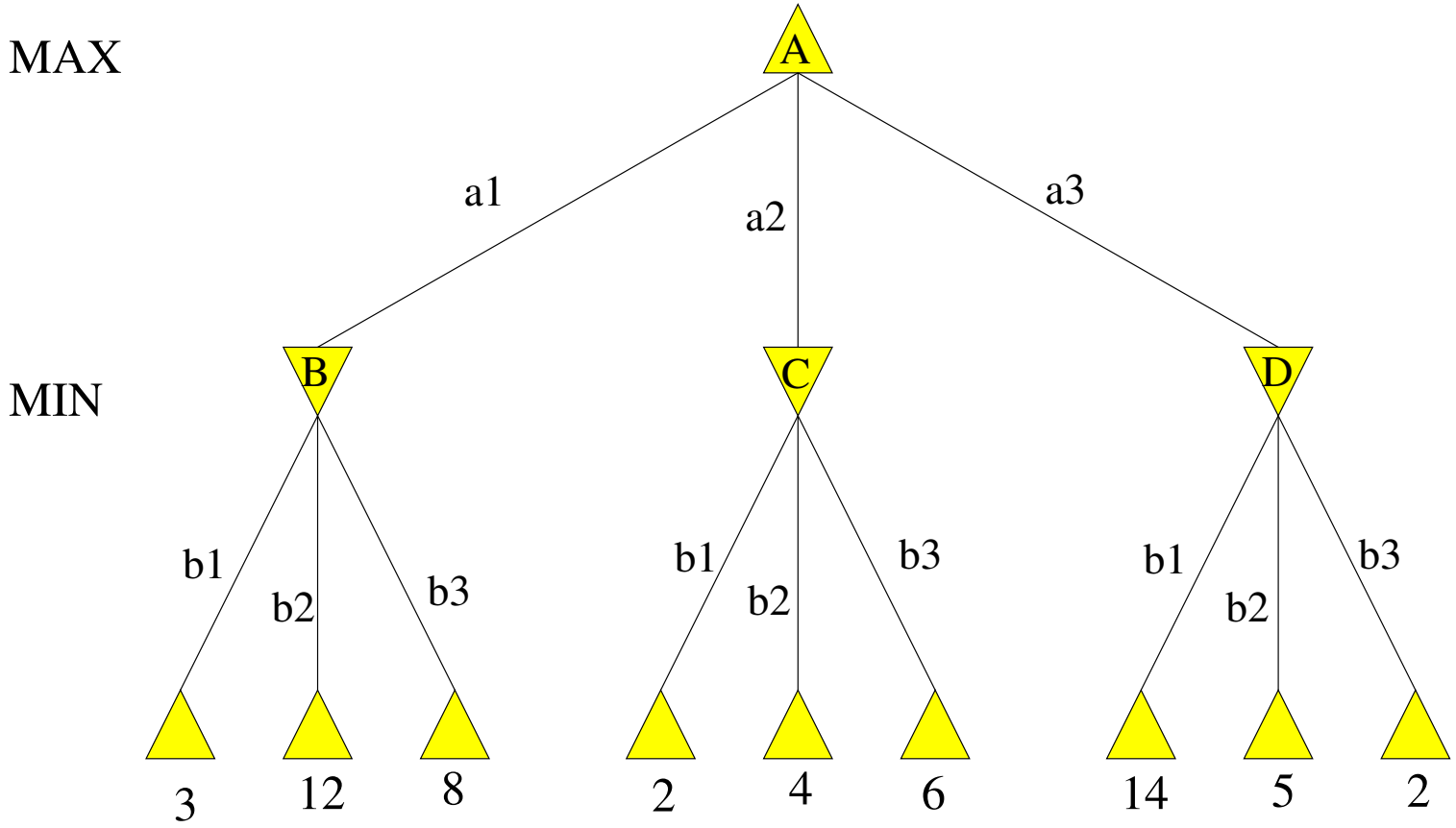
Alpha-Beta: alpha pruning



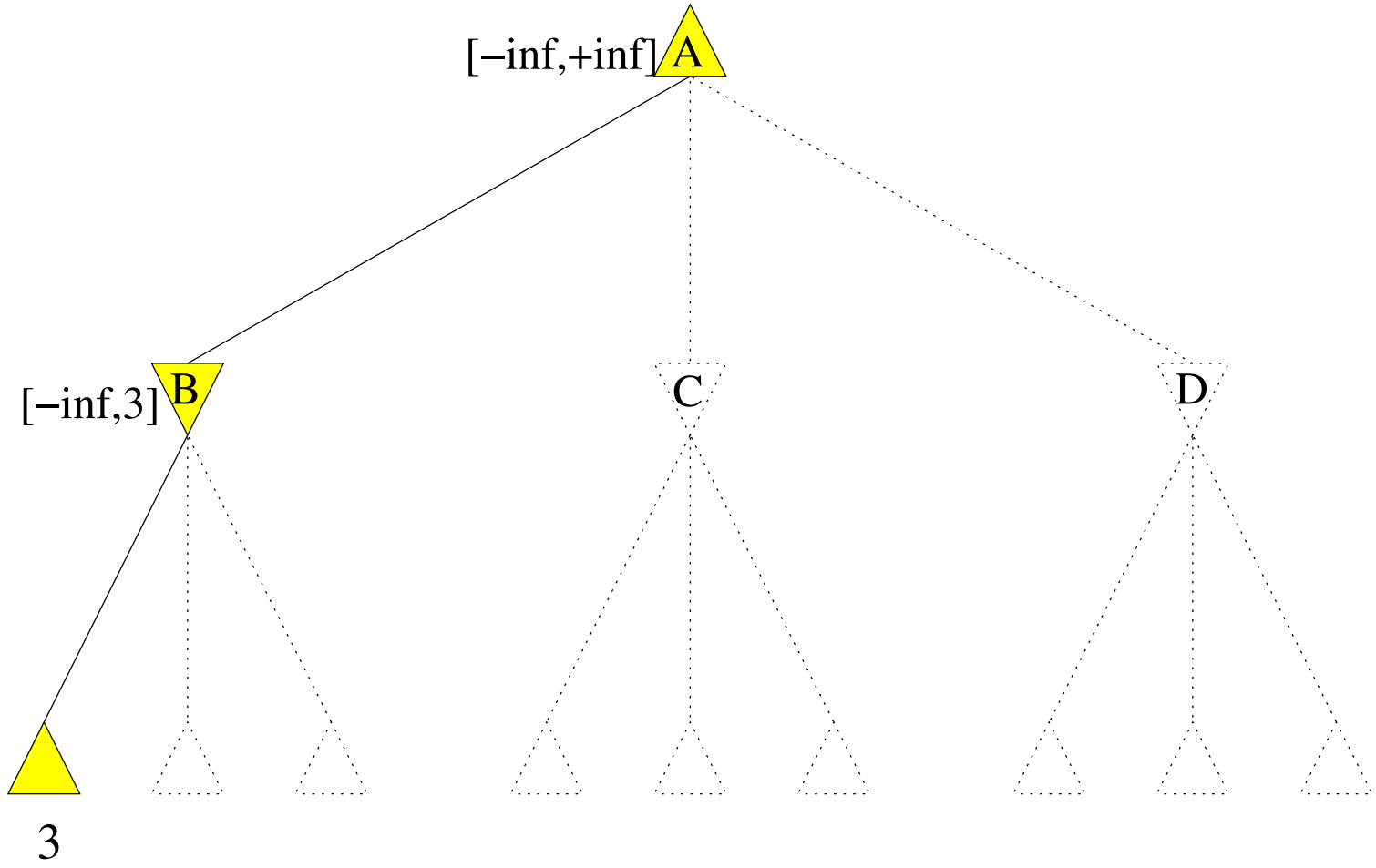
Alpha-Beta: beta pruning



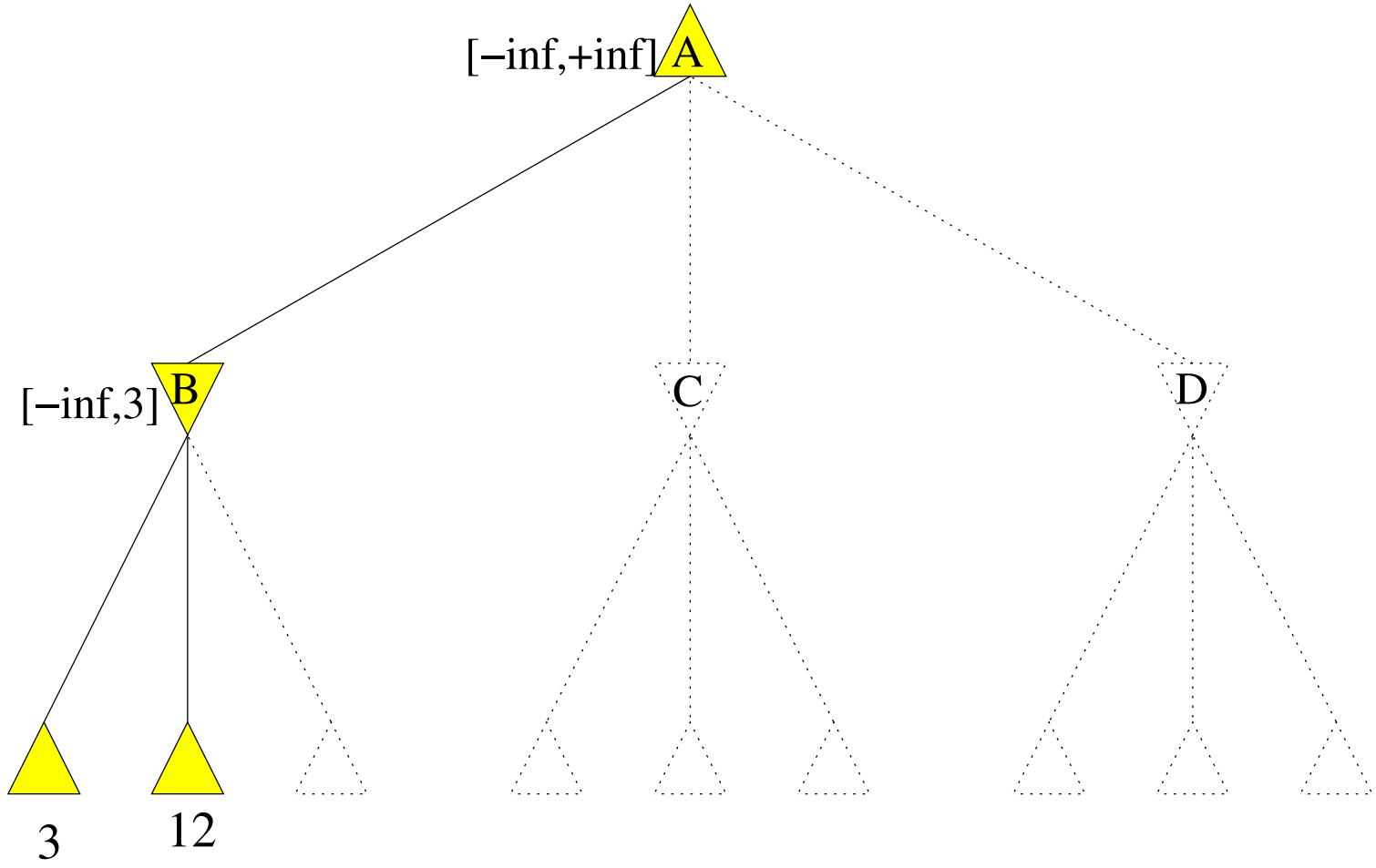
Alpha-Beta: example (R&N)



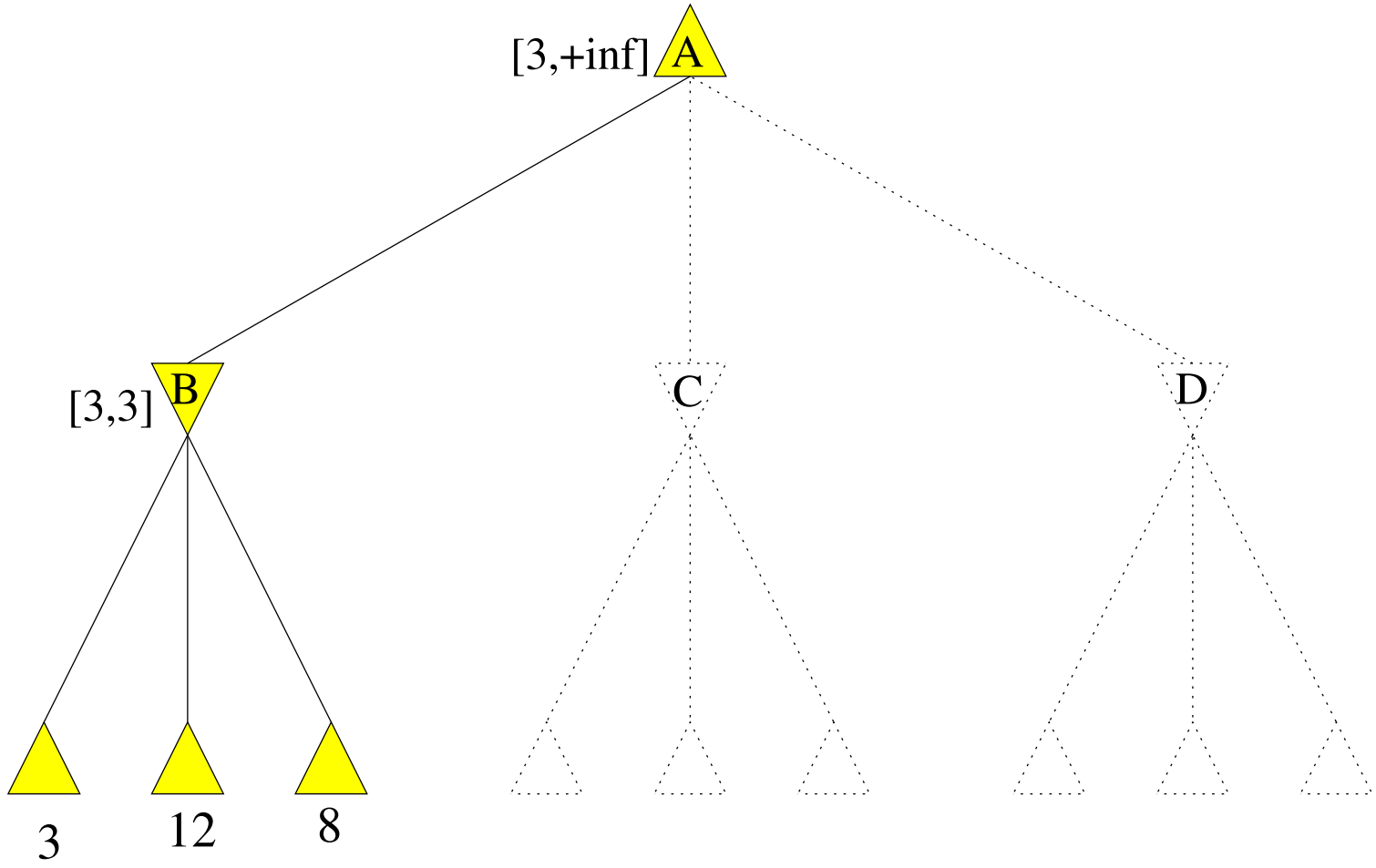
Alpha-Beta: trace 1/6



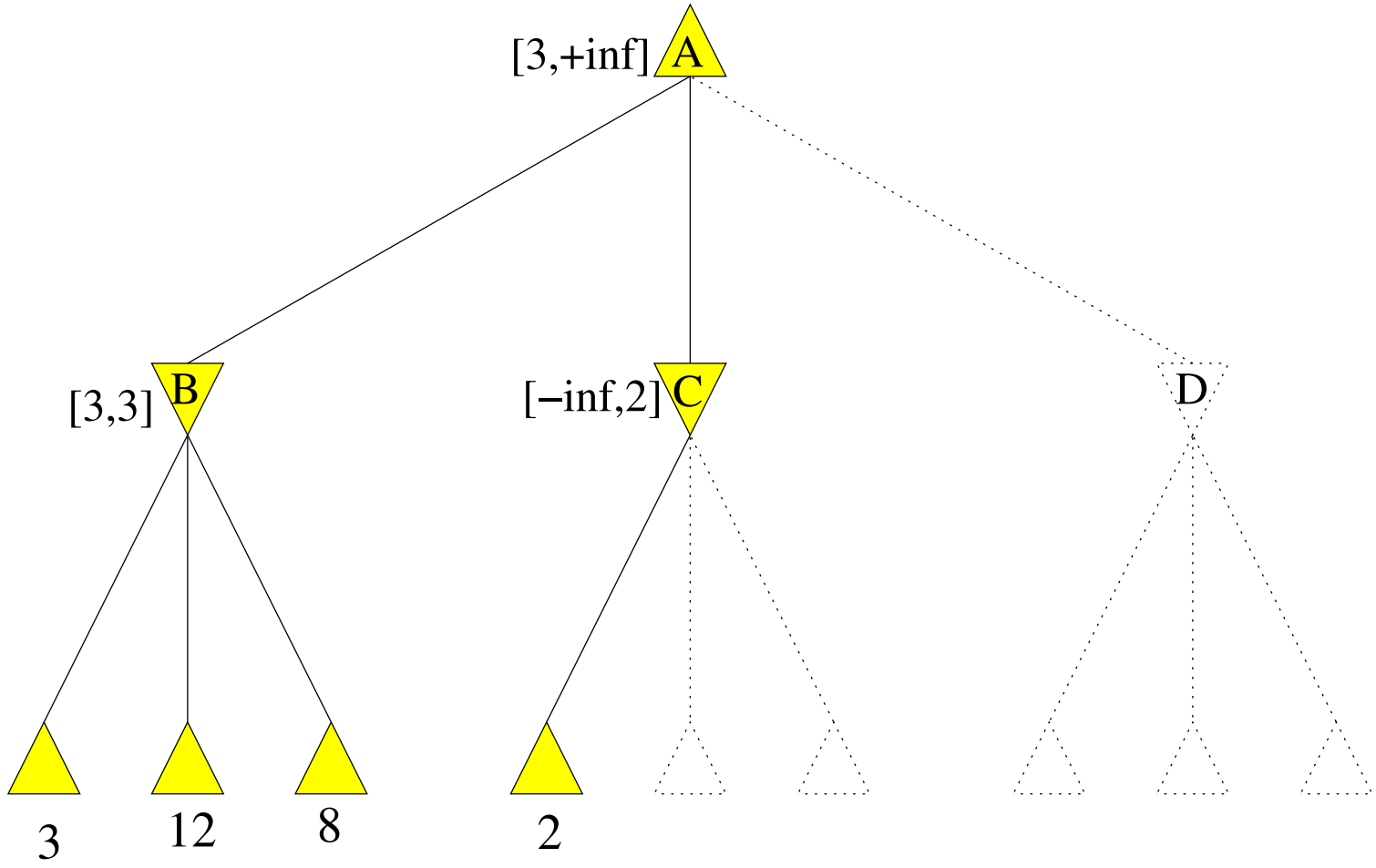
Ipha-Beta: trace 2/6



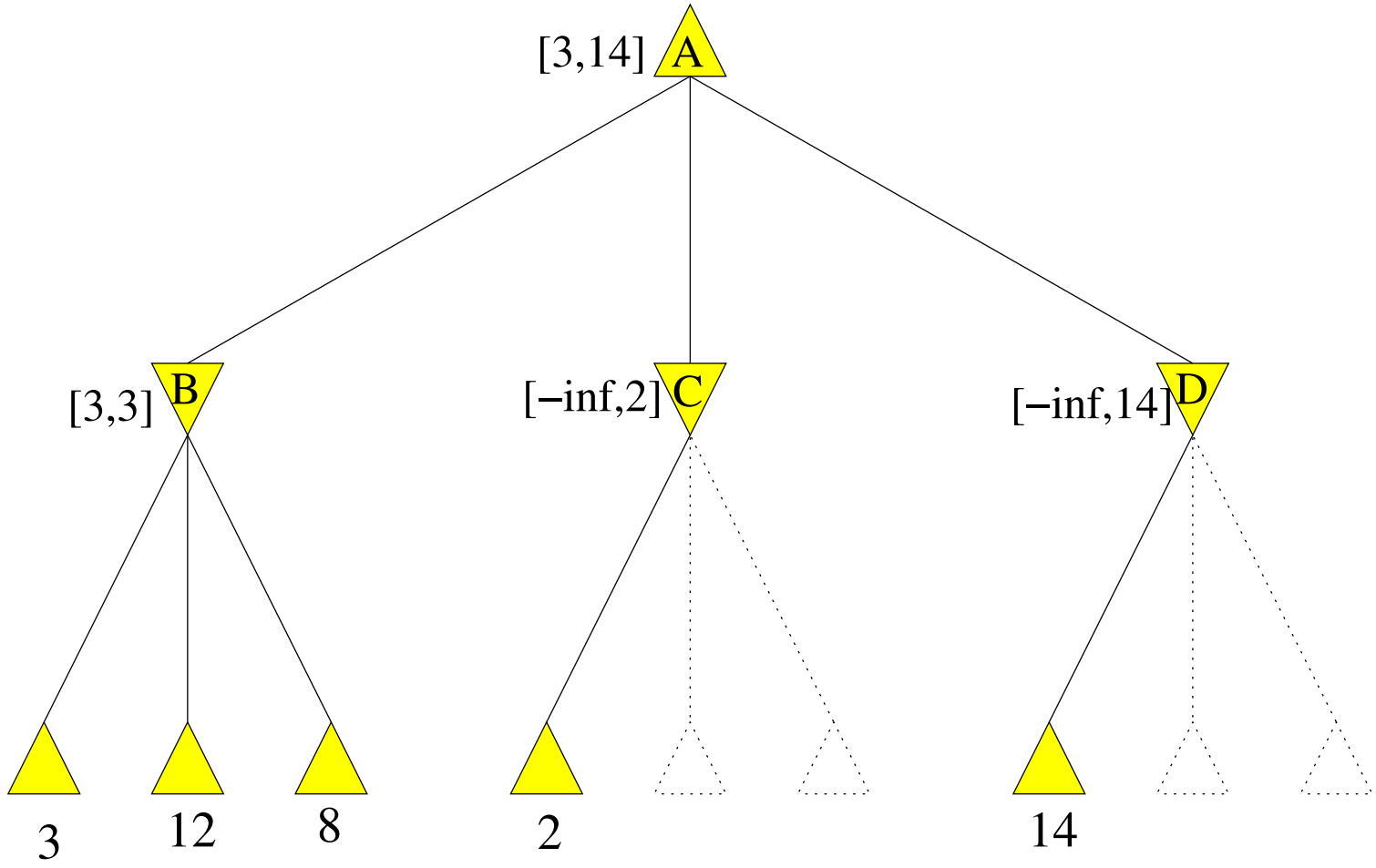
Alpha-Beta: trace 3/6



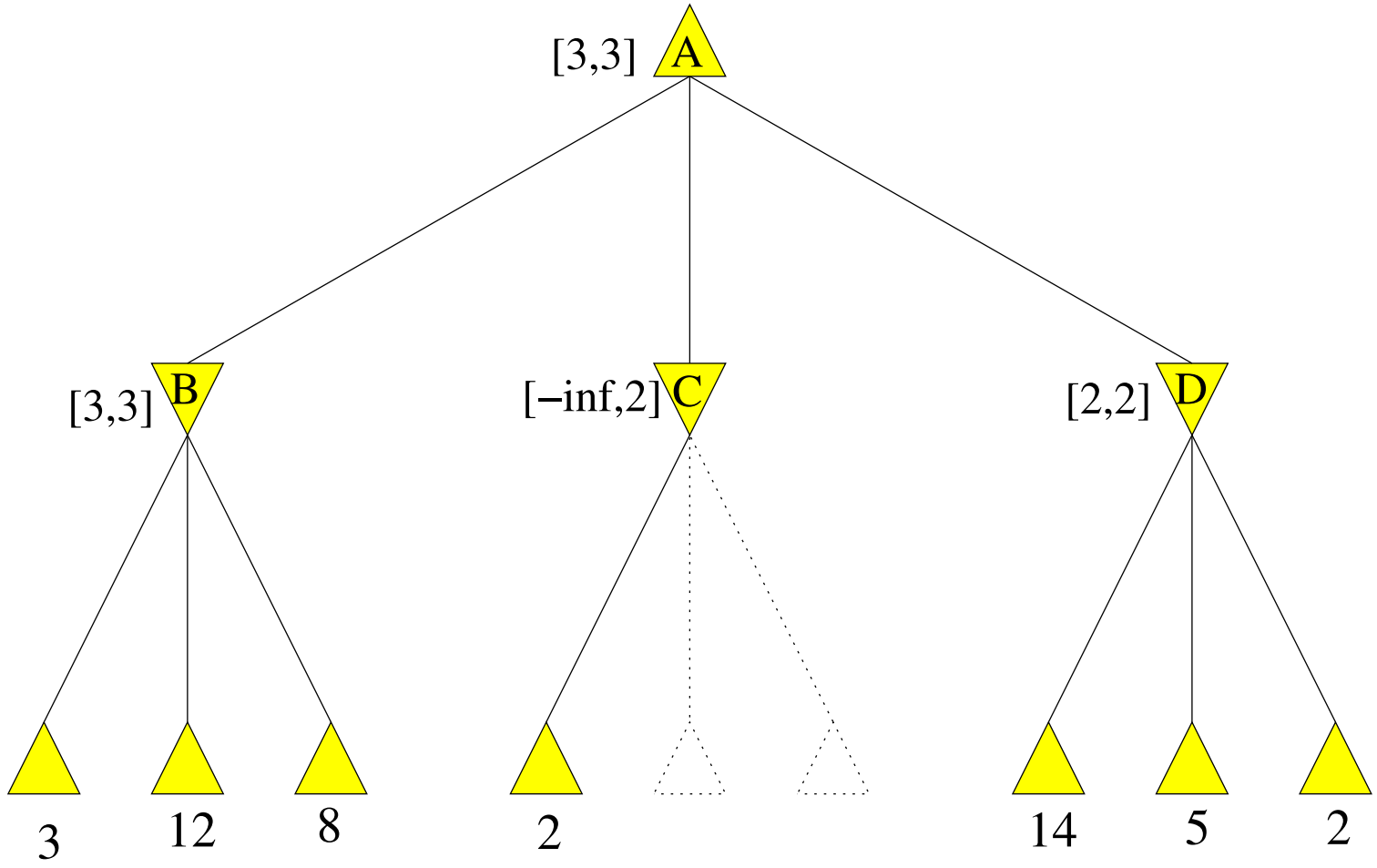
Alpha-Beta: trace 4/6



Alpha-Beta: trace 5/6



Alpha-Beta: trace 6/6



Alpha-Beta pruning: algorithm (1/2)

```
FUNCTION MaxValue(state,alpha,beta){
  if (CutOffTest(state) then
    return Eval(state)
  else
    foreach (s) in Successors(state) do{
      alpha = Max(alpha, MinValue(s,alpha,beta)
      if (alpha>=beta) then return beta
    }
    return alpha
  }
}
```

Alpha-Beta pruning: algorithm (2/2)

```
FUNCTION MinValue(state,alpha,beta){
  if (CutOffTest(state) then
    return Eval(state)
  else
    foreach (s) in Successors(state) do{
      alpha = Max(alpha, MinValue(s,alpha,beta)
      if (beta<=alpha) then return alpha
    }
    return beta
  }
}
```

Alpha-Beta: analysis

- if state successors can be ordered so that the best are tested first: $O(b^{d/2})$
- look-ahead in chess can be made 8-ply
- problem: the ordering cannot be made optimally (why?)
- if successors are ordered randomly: $O(b^{3d/4})$
- NB: all analysis is done on the idealised tree model