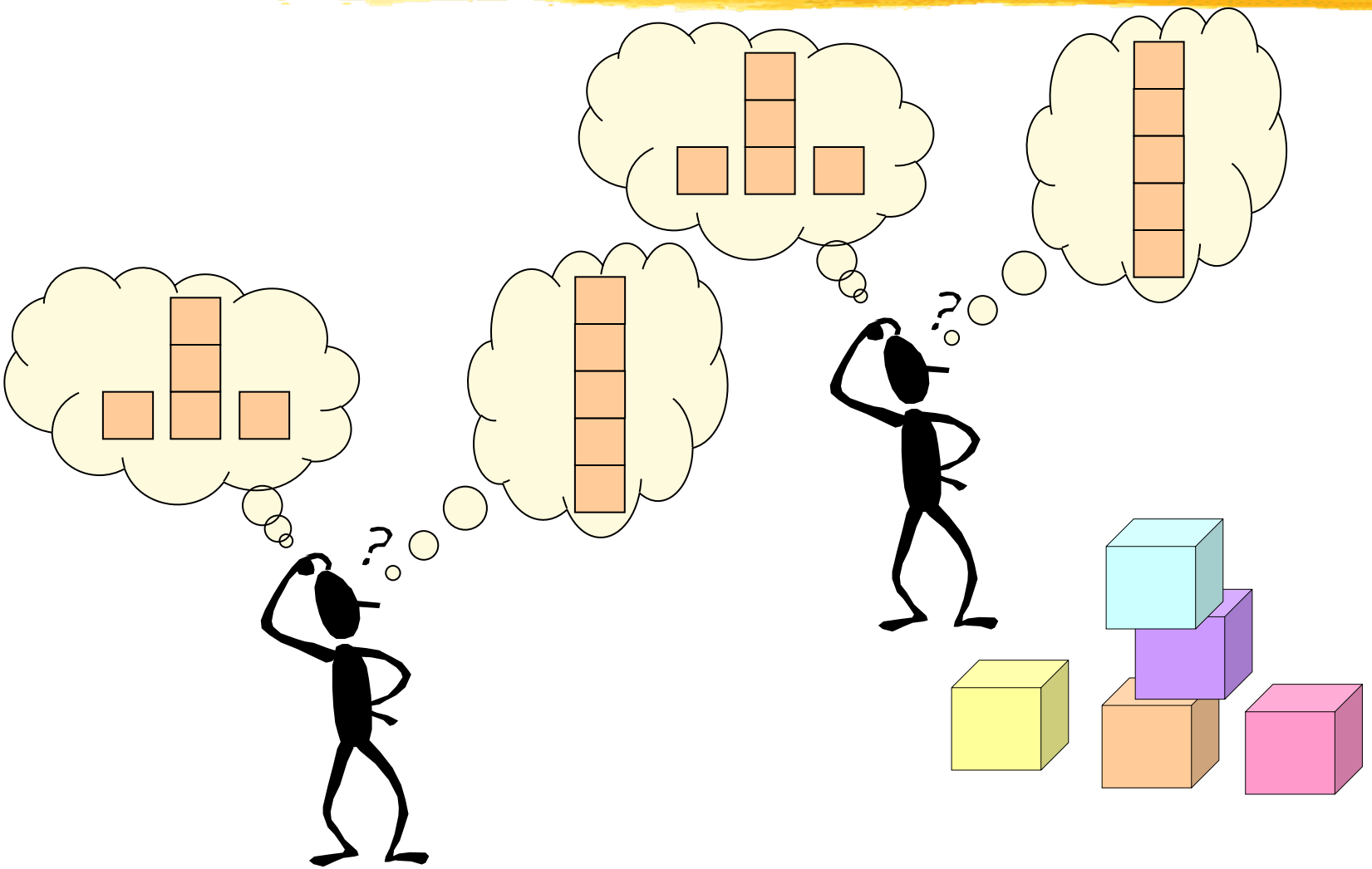
Tema 4

“Búsqueda Multiagente”

Año académico 2019/20

Profesores:
Alberto Fernández, Holger Billhardt

Resolución de problemas con múltiples agentes



Agentes especializados

Situación:

- múltiples agentes de resolución de problemas actúan en el *mismo entorno*
- las acciones de los demás agentes *influyen* en la medida de rendimiento de cada agente
- ningún agente puede *controlar* las acciones de los demás agentes
- hasta cierto punto, un agente puede *predecir* las acciones de los demás

Tipos de problemas multiagente :

- escenarios *cooperativos*: metas compartidas
- escenarios *parcialmente cooperativos*: algunas metas compartidas, otras opuestas
- escenarios *antagónicos*: metas opuestas

Escenarios antagónicos: Juegos

Juegos:

- ejemplo "clásico" de escenarios *antagónicos* (juegos de suma nula)
- el escenario está totalmente definido por las reglas del juego, y los agentes jugadores los conocen completamente

Tipos de juegos:

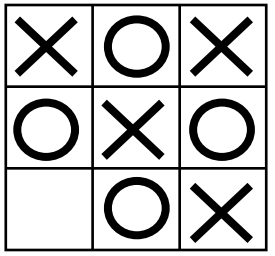
- número de jugadores :
  - bipersonales (damas) / múltiples jugadores (Monopoly)
- elementos de azar:
  - con elementos de azar (backgammon) / sin elementos de azar (damas)
- información:
  - información perfecta (damas) / información incompleta (póker)

juegos *bipersonales* con *información perfecta* (con y sin *elementos de azar*)

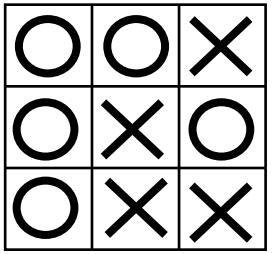
Ejemplo: Tres en Raya

Tres en Raya:

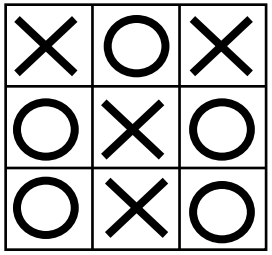
- dos jugadores (*min* y *max*)
- los jugadores van poniendo fichas en las casillas de un tablero 3x3
  - *max* usa las fichas **X** / *min* usa las fichas **O**
  - una casilla puede contener como mucho una ficha
- Reglas:
  - Inicialmente el tablero está vacío
  - *max* empieza y los jugadores se van alternando en poner sus fichas
  - *max* gana si obtiene una raya de tres fichas **X**
  - *min* gana si obtiene una raya de tres fichas **O**
  - si todas las casillas están ocupadas sin que haya una raya de 3 fichas del mismo tipo, hay empate



gana *max*



gana *min*



empate

Modelo de juegos bipersonales

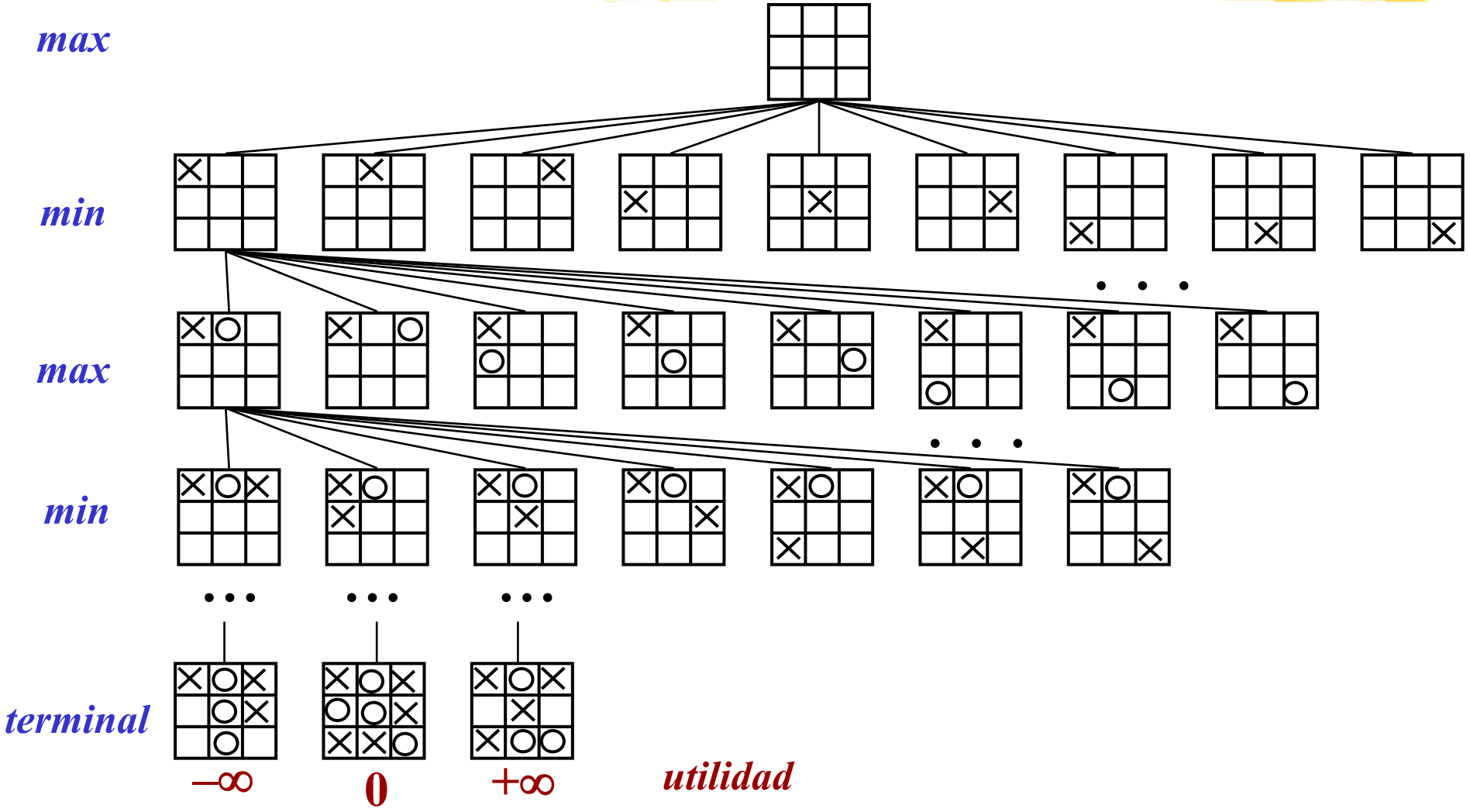
Conocimientos mínimos a priori de los agentes *max* y *min* :

- s_0 — *posición inicial* (estado inicial)
- *expandir*: $s \mapsto \{s_{i_1}, ..., s_{i_n}\}$ — cjto. finito de *posiciones sucesores*
- *terminal?*: $s \mapsto$ true | false — *prueba terminal* (si ha acabado juego)
- U: $s \mapsto k$, $k \in \mathbb{R}$ — función parcial de *utilidad* del juego

Nótese:

- la función *expandir*
  - codifica las jugadas (acciones) permitidas en una posición s
  - supone implícitamente que los jugadores se alternan en realizar las jugadas
- la función de utilidad está definida sólo en los estados terminales s
  - juegos de suma nula: *max* gana si y sólo si *min* pierde
  - gana *max*: $U(s) = +\infty$ / gana *min* : $U(s) = -\infty$ / empate: $U(s) = 0$

Ejemplo: Árbol de juego para Tres en Raya



Árboles de juego

Definición:

Sea N un conjunto de nodos, $E \subseteq N \times N$, $L = \{max,min\}$, y $G = (N,E,L)$ un árbol etiquetado. G es un *árbol de juego* si

- G no es vacío
- la raíz está etiquetada *max*
- todos los sucesores de *max* son etiquetados *min*
- todos los sucesores de *min* son etiquetados *max*

Observaciones:

- cada nivel del árbol de juego representa un *ply* (media jugada)
  - en los nodos etiquetados *max*, es el turno del agente *max*
  - en los nodos etiquetados *min*, es el turno del agente *min*
- las hojas de un árbol de juego (completamente desarrollado) representan las posiciones terminales del juego

Estrategias

Problema del agente *max*: ¿cómo determinar su mejor jugada?

- *max* podría aplicar métodos de búsqueda estándar, usando las posiciones en las que él gana como estados meta
- pero *min* no querría realizar las acciones que el plan de *max* prevé para él !

Estrategia:

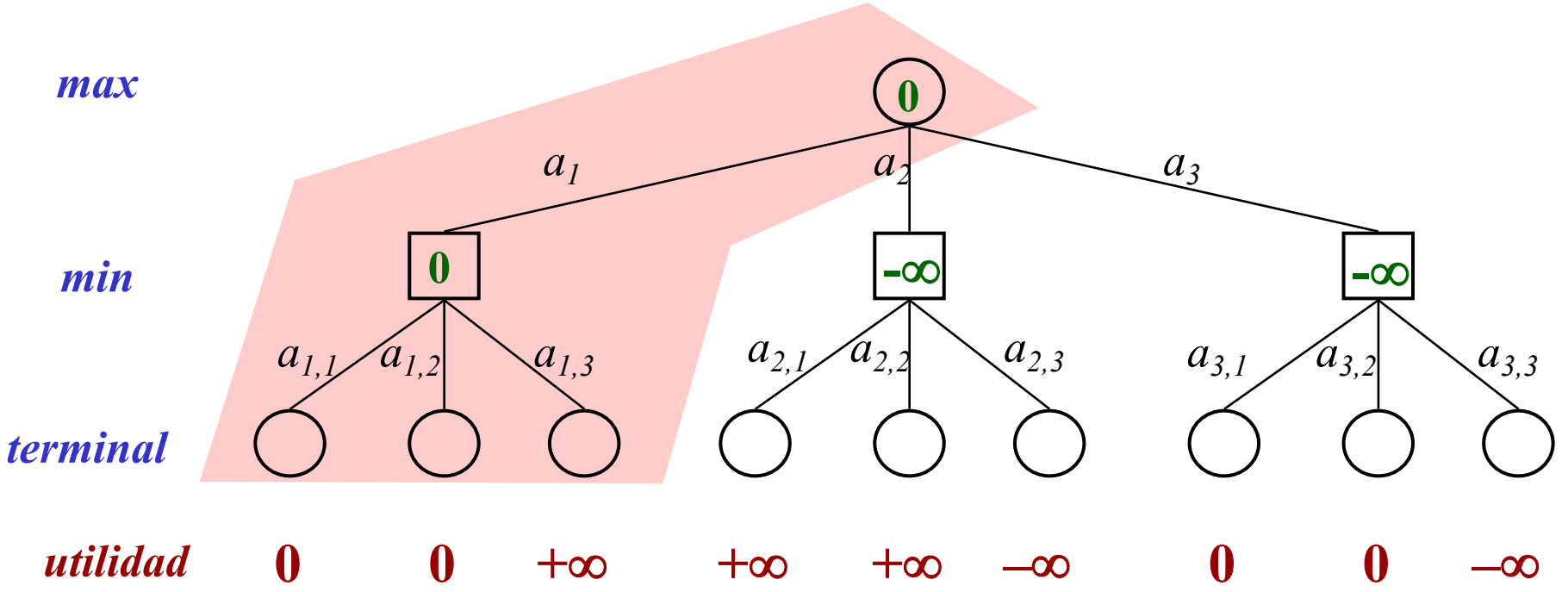
- define las jugadas de *max* para cada posible jugada de *min*
- un subárbol del árbol de juego

Estrategia óptima (ó racional) :

- la estrategia que implica el mejor resultado *garantizado* para *max*
- escenarios totalmente antagónicos con agentes racionales:
  - *max* puede asumir que *min* hará lo *mejor* para sí mismo, lo cual es lo *peor* para *max*
- la estrategia óptima para *max* es la estrategia *minimax*:
  - maximizar la utilidad mínima en cada jugada

Ejemplo: estrategia minimax

estrategia óptima: mejor jugada de *max*: a_1



Método minimax

Método Minimax:

1. Generar el árbol de juego completo
2. Aplicar la función de utilidad en cada nodo terminal
3. Propagar las utilidades hacia arriba
   - en los nodos *max*, usar la utilidad máxima de los sucesores
   - en los nodos *min*, usar la utilidad mínima de los sucesores
4. Finalmente los valores de utilidad llegan al nodo raíz (*max*)
5. La jugada óptima de *max* es la que lleva al sucesor de utilidad máxima

Algoritmo *Minimax* (versión preliminar)

Algoritmo:

- funciones mutuamente recursivas
- *estado* es el estado actual

- α : máximo de la utilidad de los sucesores de un nodo *max*
- β : mínimo de la utilidad de los sucesores de un nodo *min*

{*MaxValor* en el Minimax básico}

Función MaxValor(estado)
  Si *terminal?*(estado) **entonces**
    devolver(*U*(estado))
  sucesores ← *expandir*(*max*, estado)
  $\alpha \leftarrow -\infty$
  Para cada s∈sucesores **hacer**
    $\alpha \leftarrow \max(\alpha, \text{MinValor}(s))$
  devolver(α)
  Fin {MaxValor}

{*MinValor* en el Minimax básico}

Función MinValor(estado)
  Si *terminal?*(estado) **entonces**
    devolver(*U*(estado))
  sucesores ← *expandir* (*min*, estado)
  $\beta \leftarrow +\infty$
  Para cada s∈sucesores **hacer**
    $\beta \leftarrow \min(\beta, \text{MaxValor}(s))$
  devolver(β)
  Fin {MinValor}

Decisiones imperfectas

Problema: crecimiento exponencial del árbol de juego

- incluso en juegos muy simples, es imposible desarrollar el árbol de juego completo hasta todos sus nodos terminales

Ejemplo: Ajedrez

- Factor de ramificación = 35
- Partidas a 50 movimientos por jugador: árbol de búsqueda $35^{100} = 10^{154}$ nodos (aunque grafo de búsqueda tenga "sólo" aproximadamente 10^{40} nodos distintos)
- Supongamos 10^6 tableros/segundo:
  - $10^{154}/10^6 = 10^{148}$ segundos
  - 1 año = $3.15 * 10^7$ segundos
  - Árbol completo en $3.17*10^{140}$ años
  - Seguimos???? …
  - … edad del universo = "sólo" $13.7*10^9$ años

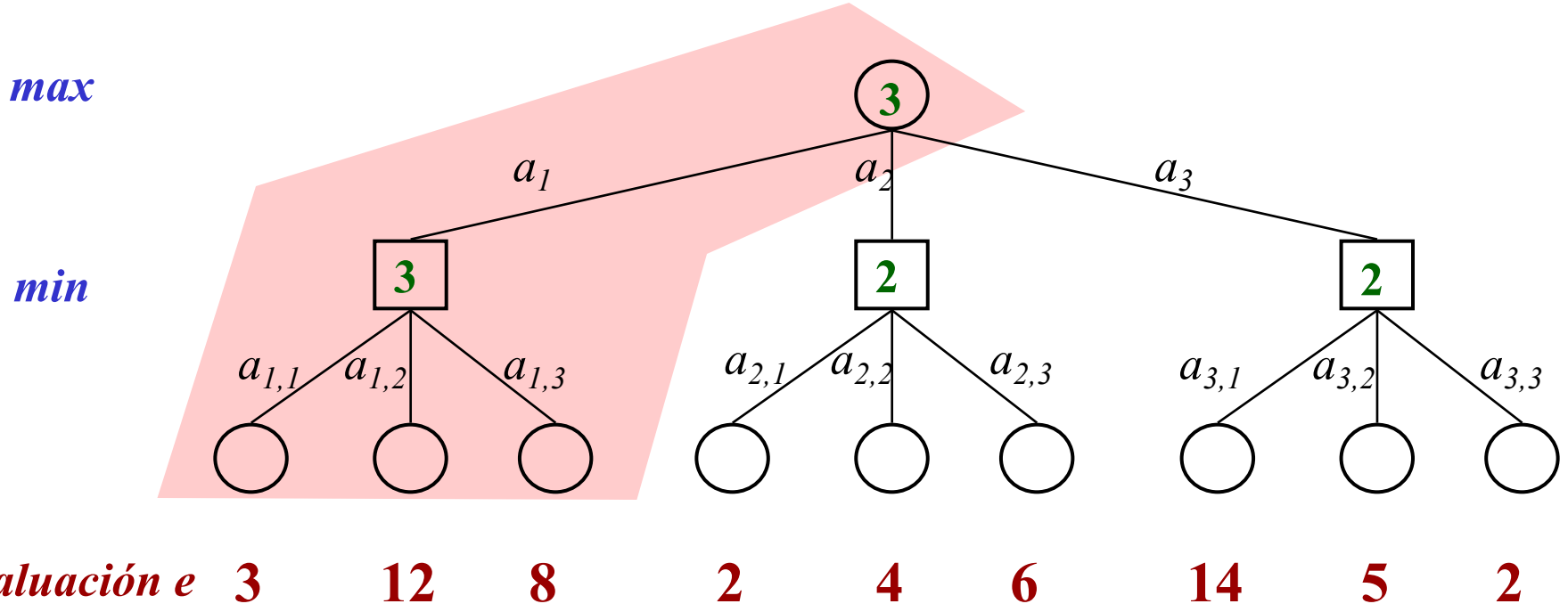
Decisiones imperfectas

Solución: *Heurísticas*

- sustituir la prueba terminal por una *prueba suspensión* que detiene la búsqueda aún sin llegar a una posición terminal:
  - límite de profundidad fijo, posiciones "en reposo", …
- aplicar una *función de evaluación e*, que estime la utilidad esperada del juego correspondiente a una posición *s* determinada
  - *e* debe coincidir con la función de utilidad *u* en los nodos terminales
  - suele ser función lineal ponderada : $e(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$
  - *Ajedrez*: $e(s)$ = "suma de los valores materiales en s"
  - *Tres en Raya*: $e(s)$ = "nº de líneas abiertas para líneas *max* en s" – "nº de líneas abiertas para líneas *min* en s"
- heurísticas fuertes:
  - las jugadas del *Minimax* ya no serán óptimas, pero dependerán de la calidad de las heurísticas

Ejemplo: *Minimax* con suspensión

estrategia óptima: mejor jugada de *max*: a_1



Algoritmo *Minimax*

Algoritmo:

- funciones mutuamente recursivas
- *estado* es el estado actual

- α : máximo de la evaluación de los sucesores de un nodo *max*
- β : mínimo de la evaluación de los sucesores de un nodo *min*

{*MaxValor*: Minimax con suspensión}

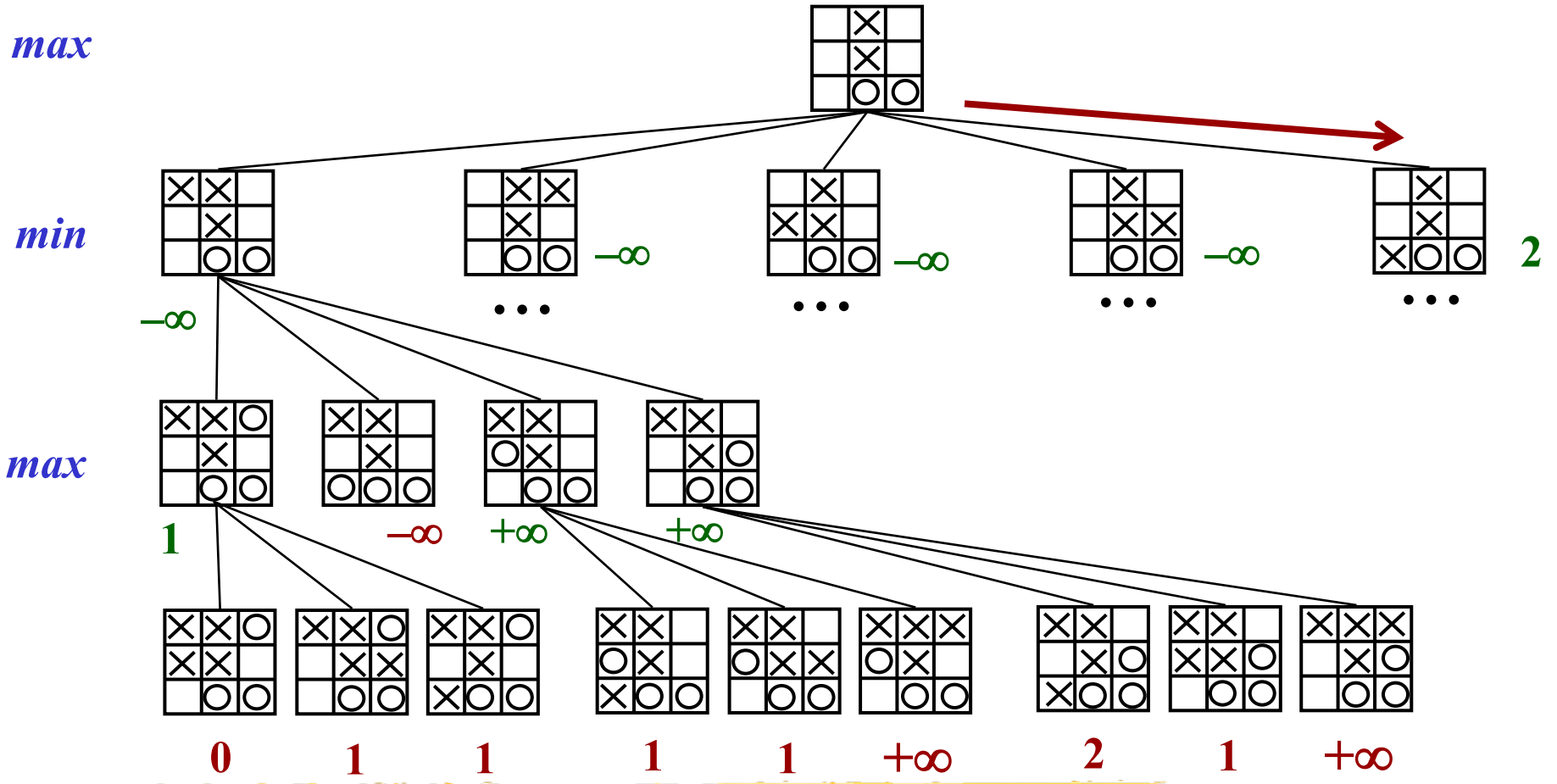
Función MaxValor(estado)
  Si *suspensión?*(estado) **entonces**
    devolver(*e*(estado))
  sucesores ← *expandir*(*max*, estado)
  $\alpha \leftarrow -\infty$
  Para cada s∈sucesores **hacer**
    $\alpha \leftarrow \max(\alpha, \text{MinValor}(s))$
  devolver(α)
  Fin {MaxValor}

{*MinValor*: Minimax con suspensión}

Función MinValor(estado)
  Si *suspensión?*(estado) **entonces**
    devolver(*e*(estado))
  sucesores ← *expandir* (*min*, estado)
  $\beta \leftarrow +\infty$
  Para cada s∈sucesores **hacer**
    $\beta \leftarrow \min(\beta, \text{MaxValor}(s))$
  devolver(β)
  Fin {MinValor}

Ejemplo: Tres en Raya

Suspensión en ply 3

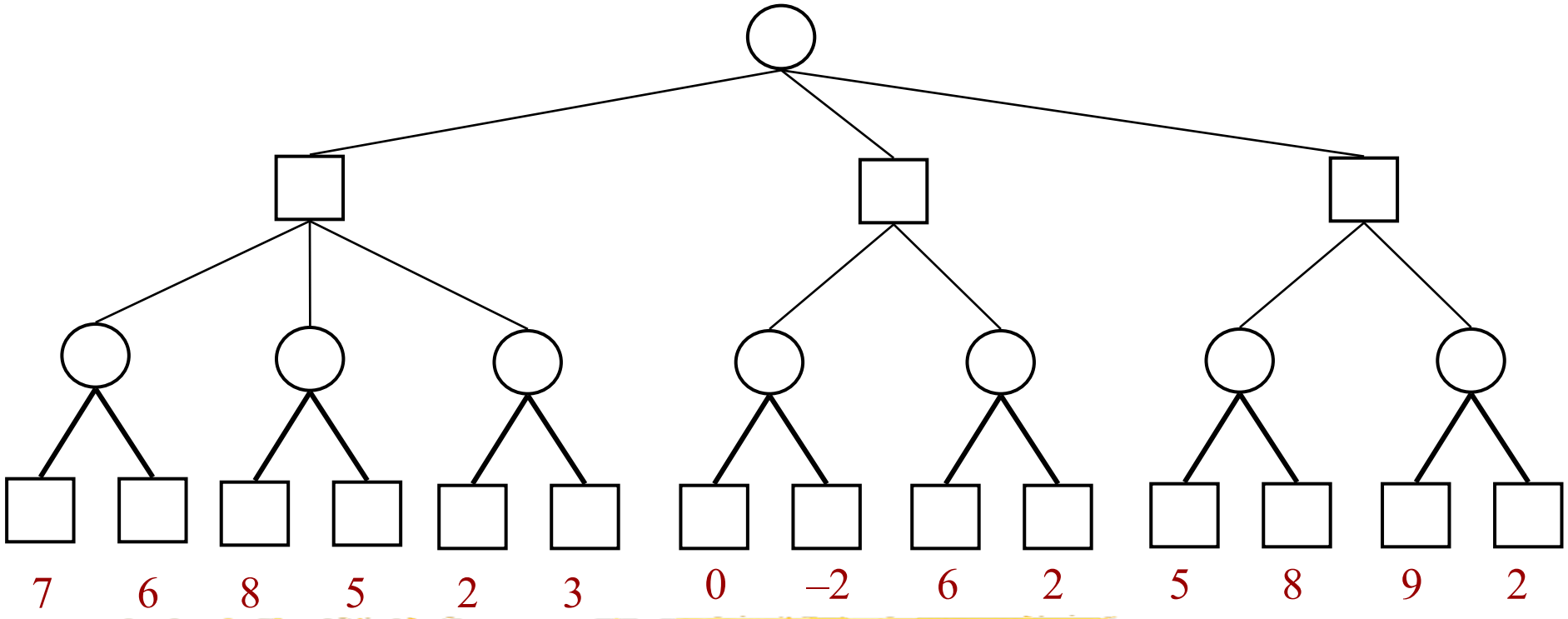


Ejercicio

Considérese el siguiente árbol de juego desarrollado hasta ply 3. Los nodos están etiquetados con los valores de la función de evaluación *e*.

a) Evalúe el árbol del juego en base al algoritmo minimax.

b) ¿Cuál es la mejor jugada para el agente *max*?



Resumen Minimax

Complejidad:

- *Minimax* genera el árbol de juego completo, en profundidad primero, hasta los nodos (o el nivel de) suspensión
- Complejidad : $O(b^d)$ (factor de ramificación b; número de *plys* explorados d)

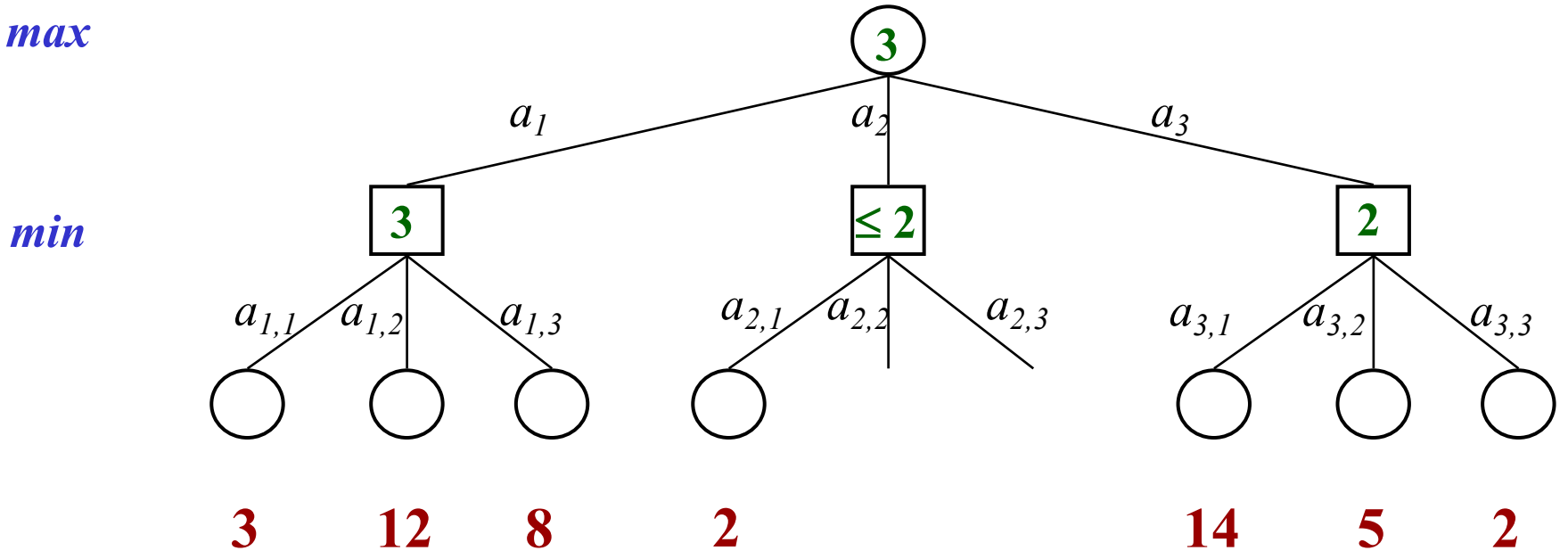
Extensiones del *Minimax*:

- mejorar la complejidad (poda α–β: descartar nodos irrelevantes)
- juegos con elementos de azar (*ExpectMinimax*: hay un nuevo jugador "*azar*")
- juegos con información parcial (búsqueda en *estados de creencias*)
- mejorar las heurísticas :
  - aprender funciones de evaluación y suspensión (p.e. por el "efecto horizonte")
- relajar la suposición de racionalidad perfecta de *min*:
  - suponga que *max* está a punto de perder si *min* juega de forma óptima
  - sin embargo, hay una jugada que hace ganar a *max*, si *min* hace un solo error
- …

Poda α-β

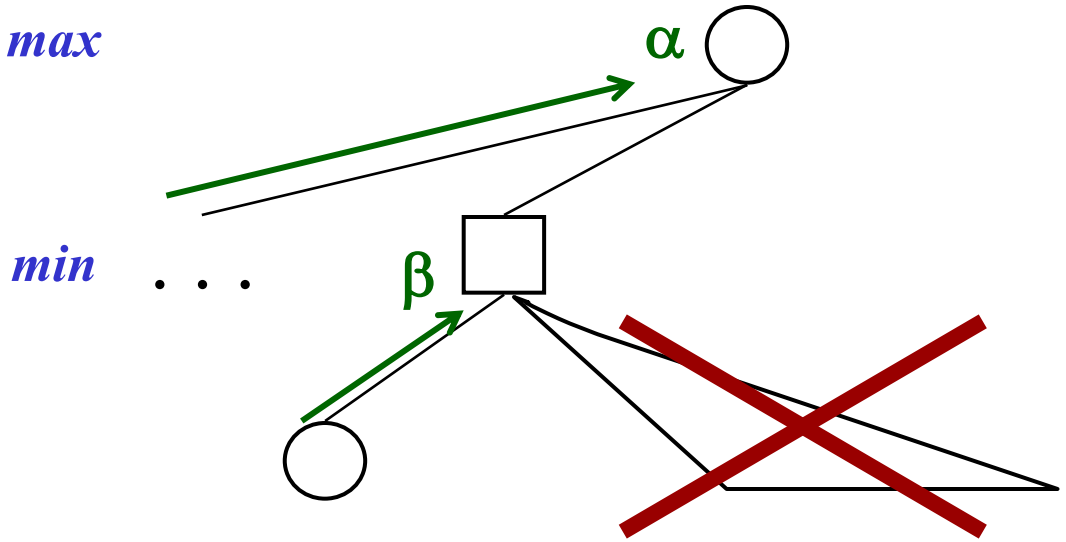
Nótese:

- a veces es posible calcular el valor *exacto* de un nodo sin tener que evaluar todos sus sucesores
- Ejemplo:



Poda α-β

Utilidad más alta encontrada en un nodo *max* hasta el momento: α



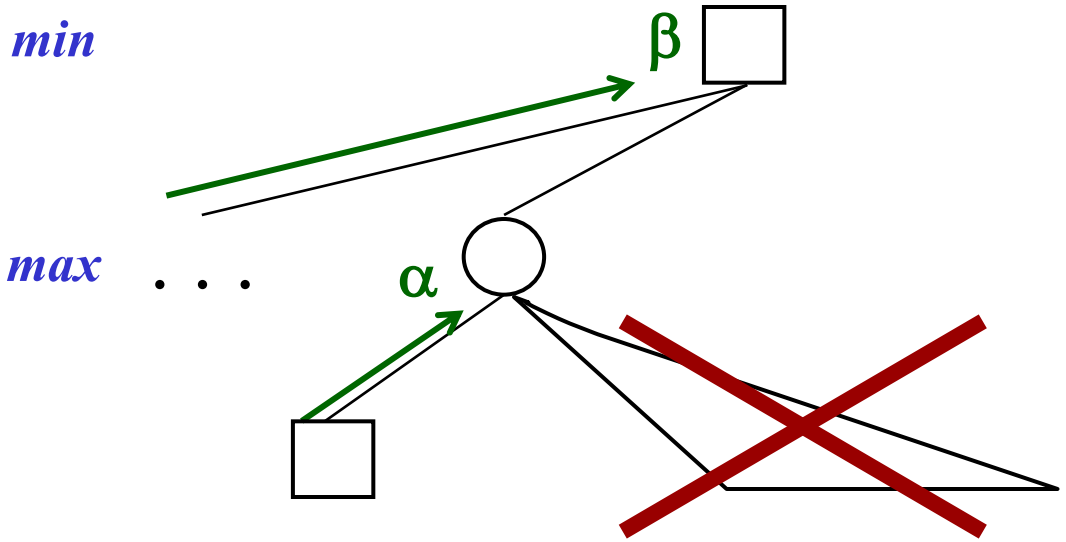
Condición de poda: β≤α

- La utilidad U_{min} del nodo *min* será como mucho β
- La utilidad U_{max} del nodo *max* será al menos α
- No es necesario explorar los sucesores restantes de *min*, ya que se cumple en todo caso:

$$U_{min} \leq \beta \leq \alpha \leq U_{max}$$

Poda α-β

Utilidad más baja encontrada en un nodo *min* hasta el momento: β



Condición de poda: α≥β

- La utilidad U_{max} del nodo *max* será al menos α
- La utilidad U_{min} del nodo *min* será como mucho β
- No es necesario explorar los sucesores restantes de *max*, ya que se cumple en todo caso:

$$U_{min} \leq \beta \leq \alpha \leq U_{max}$$

Minimax con poda α-β

Algoritmo:

- funciones mutuamente recursivas
- *estado* es el estado actual

- α es el mejor valor de evaluación para *max* en el camino hasta *estado*
- β es el mejor valor de evaluación para *min* en el camino hasta *estado*

{*MaxValor*: Minimax con poda α–β}

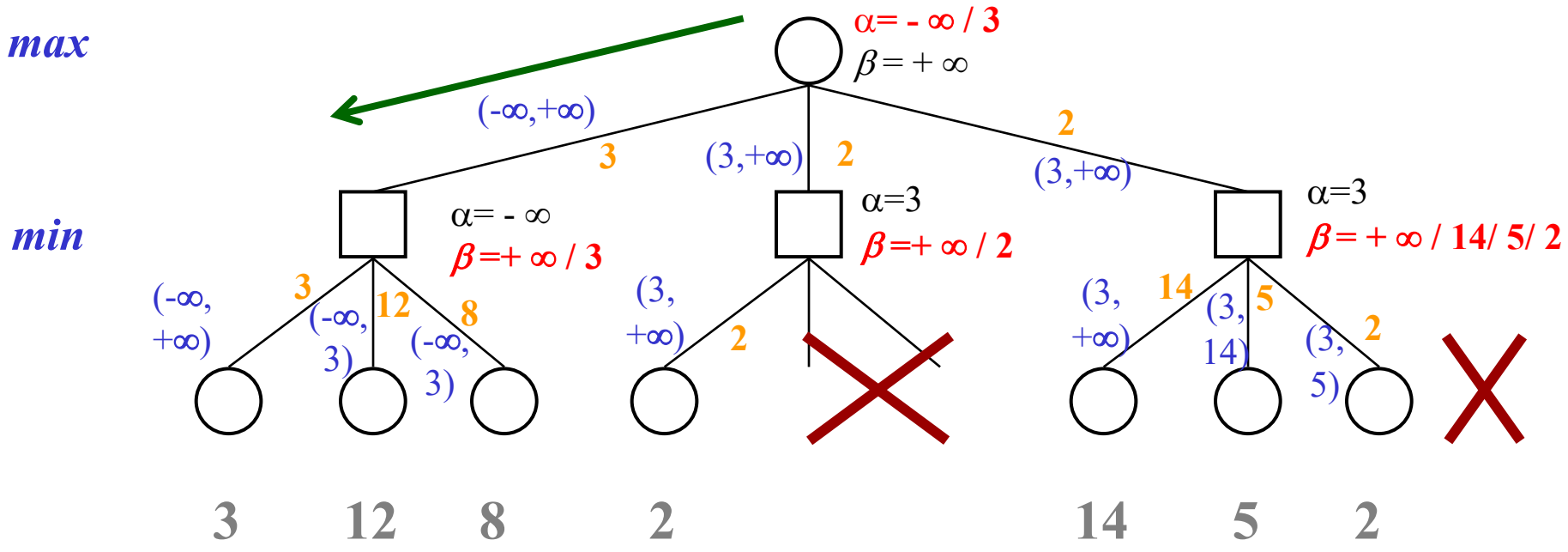
```
Función MaxValor(estado,α,β)
    Si suspensión?(estado) entonces
            devolver(e(estado))
    sucesores ← expandir(max, estado)
    Para cada s∈sucesores hacer
            α ← max(α, MinValor(s,α,β ))
            Si α ≥ β entonces devolver(α)
    devolver(α)
    Fin {MaxValor}
```

{*MinValor*: Minimax con poda α–β}

```
Función MinValor(estado,α,β)
    Si suspensión?(estado) entonces
            devolver(e(estado))
    sucesores ← expandir (min, estado)
    Para cada s∈sucesores hacer
            β ← min(β,MaxValor(s,α,β ))
            Si β ≤ α entonces devolver(β)
    devolver(β)
    Fin {MinValor}
```

Ejemplo: *Minimax* con poda α-β

- Simulación del ejemplo con el algoritmo :
  - **Negro** / **Rojo** : valores de α y β dentro de las funciones *MaxValor* y *MinValor*
  - **Azul**: valores de los parámetros en las llamadas a *MaxValor* y *MinValor*
  - **Amarillo**: valores devueltos por *MaxValor* o *MinValor*



Resumen *Minimax* con poda α-β

Análisis:

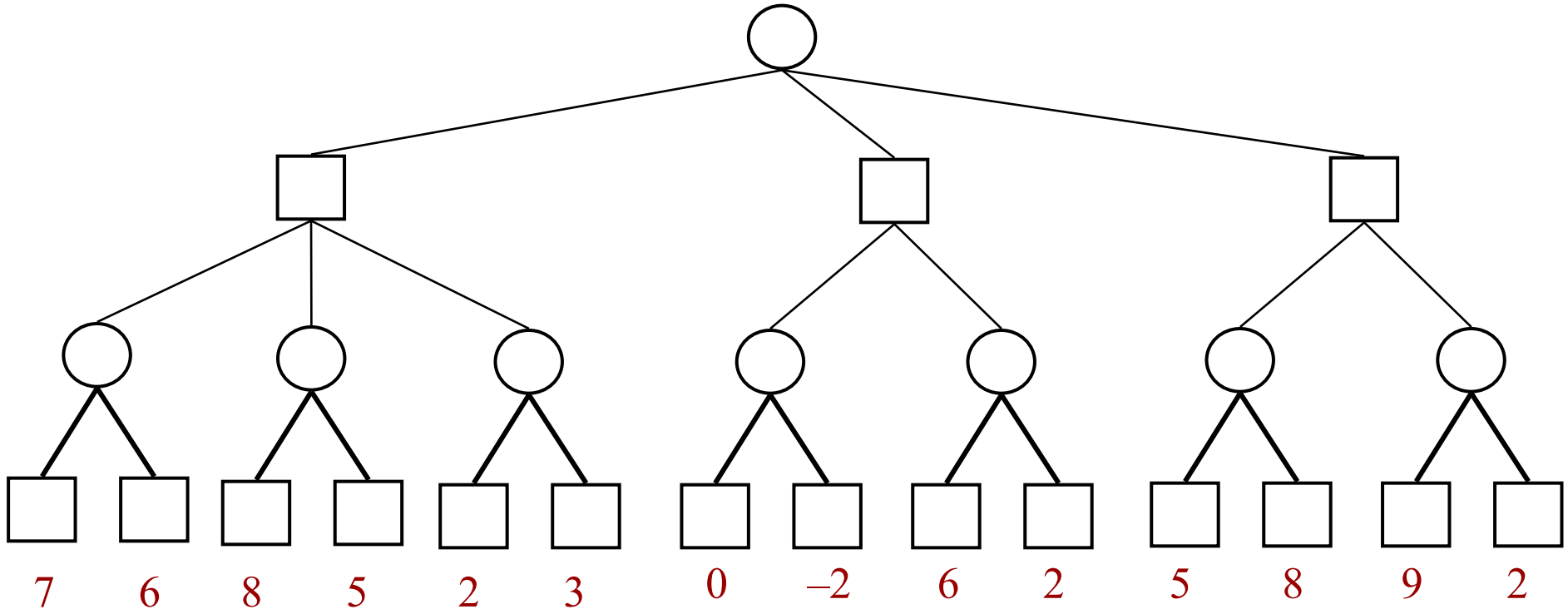
- El algoritmo *Minimax* con poda α-β siempre produce el mismo resultado que sin la poda
- La eficiencia de *Minimax* con poda α-β depende del orden en el que se exploran los nodos
- Complejidad (factor de ramificación *b*; número de *plys* explorados *d*)
  - Minimax sin poda α-β : $O(b^d)$ (p.e. 10^4= 10.000 nodos)
  - Minimax poda α-β (mejor caso): $O(b^{d/2})$ (p.e. 10^2= 100 nodos)
  - Minimax poda α-β (caso medio): $O(b^{3d/4})$ (p.e. 10^3= 1.000 nodos)

Ejercicio

Considérese el siguiente árbol de juego desarrollado hasta ply 3. Los nodos están etiquetados con los valores de la función de evaluación *e*.

a) Evalúe el árbol del juego en base al algoritmo minimax con poda alfa-beta

b) ¿Cuál es la mejor jugada para el agente *max*?

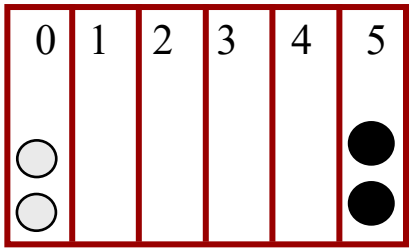


Juegos bipersonales con elemento de azar

- Algoritmo: *ExpectMinimax*
- Idea:
  - Utilizar el algoritmo *Minimax*
  - Añadir un nuevo jugador: "azar" que se incluye en el árbol siempre que haya un evento independiente de los jugadores y cuyo resultado es aleatorio
  - Los sucesores de un nodo "azar" son las posibles situaciones que podrían ser el resultado de este elemento de azar
    - p.e.: todos los posibles resultados de tirar un dado
  - Cada uno de los sucesores de un nodo "azar" tiene asociado la probabilidad de que este resultado ocurra
    - p.e.: en el caso del dado: $p(1)=1/6$, …, $p(6)=1/6$

Ejemplo: Backgammon simplificado

- Estado inicial:



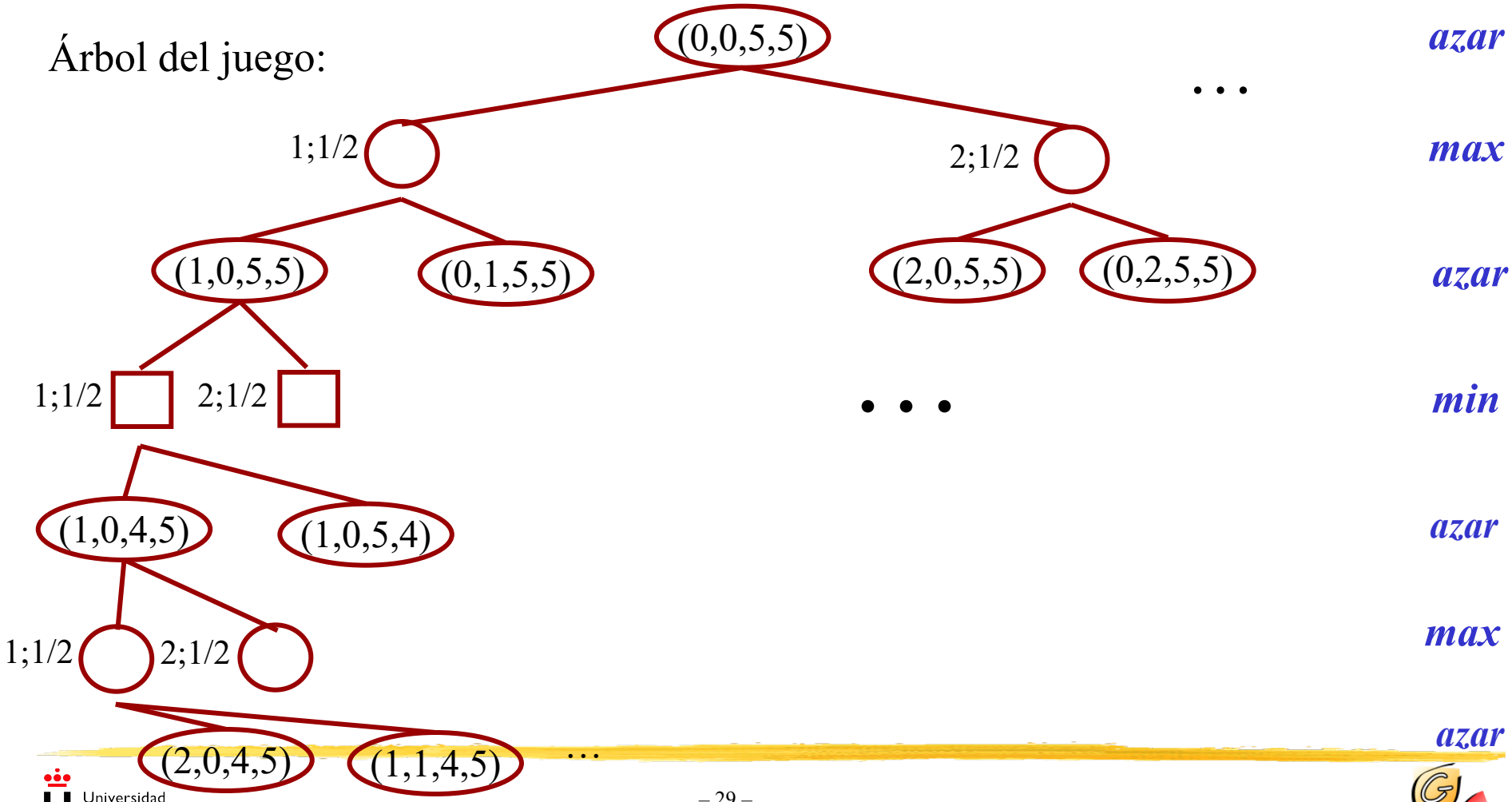
- Objetivo:
  - mover las fichas al lado opuesto (*max*=○ , al campo 5 y *min*= ● al campo 0)
- Reglas:
  - *max* empieza y los jugadores se van alternando sus jugadas
  - Cada jugada consiste primero en tirar una moneda; la cara tiene el valor 1 y la cruz el valor 2. Después se mueve una de las fichas 1 o 2 campos en la dirección deseada (dependiendo del resultado de la tirada de la moneda)
  - No es posible mover una ficha a un campo que tiene una ficha del oponente
  - Si un jugador no puede mover sus fichas pierde su turno (si puede, tiene que mover una ficha)
  - Gana el jugador que primero ha movido ambas fichas al campo deseado
- El elemento de azar ocurre antes de elegir la jugada

Ejemplo: Backgammon simplificado

Representación de estados: (x_1,x_2,y_1,y_2)

x_1 y x_2 posiciones de las fichas blancas e y_1 y y_2 posiciones de las fichas negras

Árbol del juego:



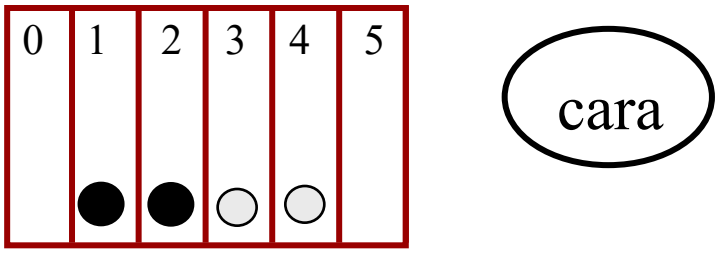
ExpectMinimax

- Objetivo: elegir la mejor jugada para *max*
- ¿Cómo propagar los valores de utilidad/evaluación de los nodos hoja a los nodos superiores?
- Solución:

$$ExpectMinimax(n) = \begin{cases} \mathrm{e}(n) & \text{si } n \text{ es nodo suspensión} \\ \max_{s \in \text{expandir}(n)} (ExpectMinimax(s)) & \text{si } n \text{ es nodo } max \\ \min_{s \in \text{expandir}(n)} (ExpectMinimax(s)) & \text{si } n \text{ es nodo } min \\ \sum_{s \in expandir(\mathrm{n})} (\mathrm{p}(s) \cdot ExpectMinimax(s)) & \text{si } n \text{ es nodo azar} \end{cases}$$

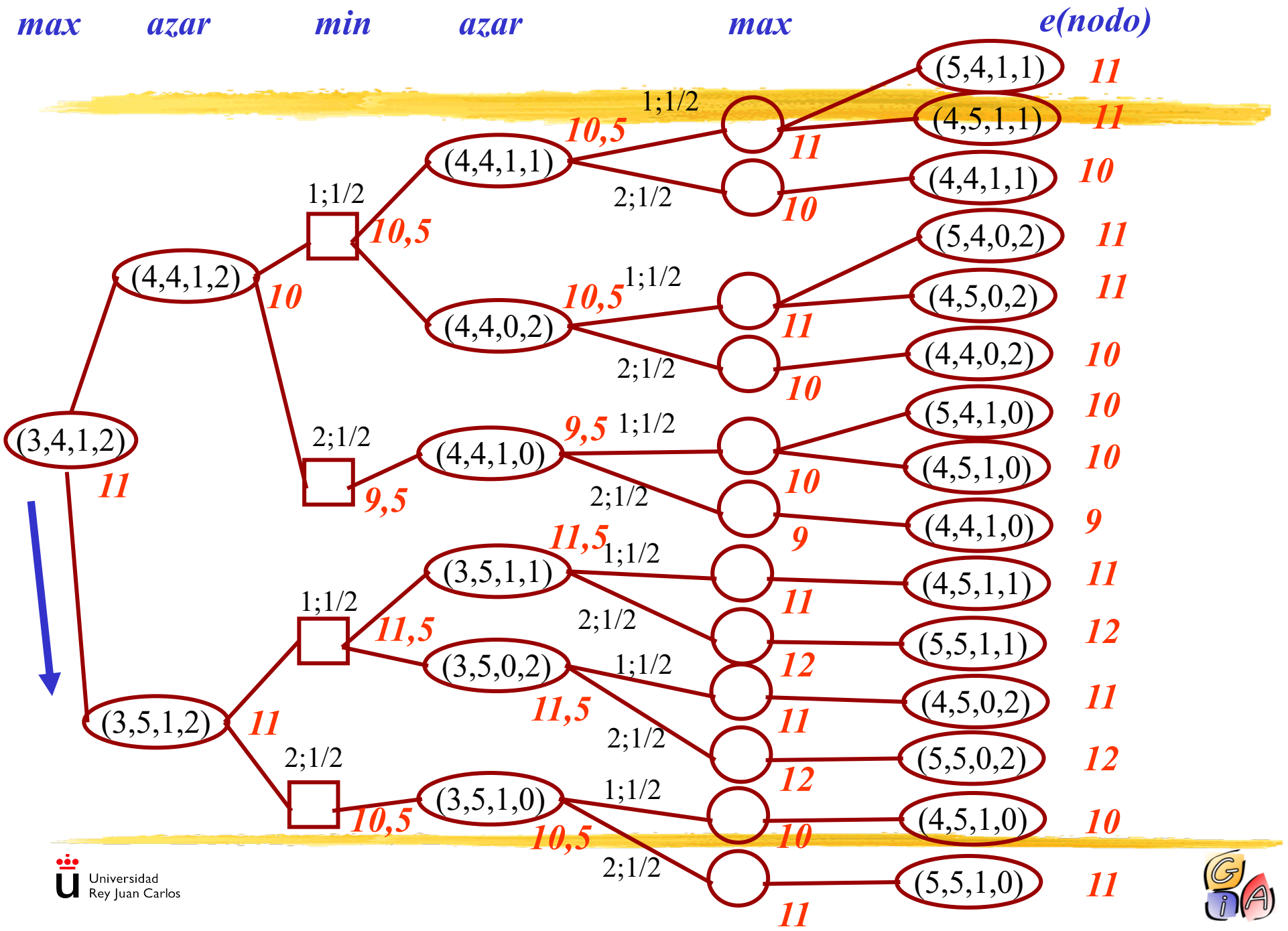
Ejemplo: Backgammon simplificado

- Situación actual: (toca a *max*)



(3,4,1,2); *max* tiene que mover una ficha (blanca) una posición

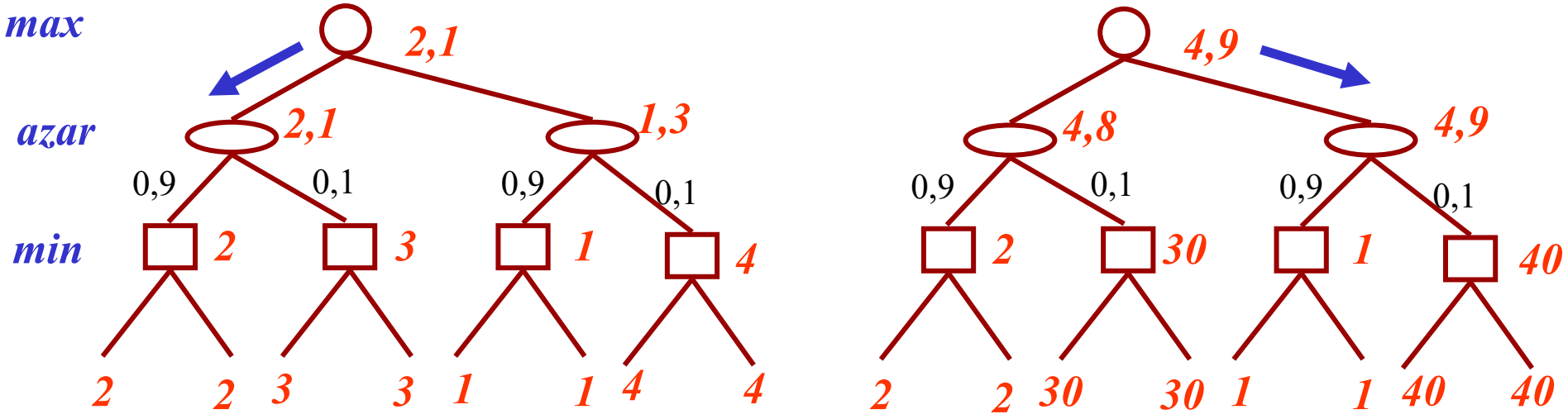
- Suponemos el algoritmo ExpectMinimax con un nivel de suspensión de 5
- Como función de evaluación se usa la siguiente: $e((a,b,c,d)) = a+b+c+d$
  - ✓ valores altos de *a* y *b* son buenos para *max* porque indican que sus fichas están cerca de la meta (5)
  - ✓ valores altos de *c* y *d* son buenos para *max* porque indican que las fichas de min están lejos de su meta (0)
  - ✓ para el estado actual: $e((3,4,1,2))=10$


max
azar
min
azar
max
e(nodo)
(3,4,1,2)
11
(4,4,1,2)
10
(3,5,1,2)
11
1;1/2
10,5
(4,4,1,1)
2;1/2
(4,4,0,2)
2;1/2
9,5
(4,4,1,0)
1;1/2
11,5
(3,5,1,1)
2;1/2
(3,5,0,2)
2;1/2
10,5
(3,5,1,0)
(5,4,1,1)
11
(4,5,1,1)
11
(4,4,1,1)
10
(5,4,0,2)
11
(4,5,0,2)
11
(4,4,0,2)
10
(5,4,1,0)
10
(4,5,1,0)
10
(4,4,1,0)
9
(4,5,1,1)
11
(5,5,1,1)
12
(4,5,0,2)
11
(5,5,0,2)
12
(4,5,1,0)
10
(5,5,1,0)
11

Funciones de evaluación/utilidad

Criterios de los funciones de evaluación/utilidad:

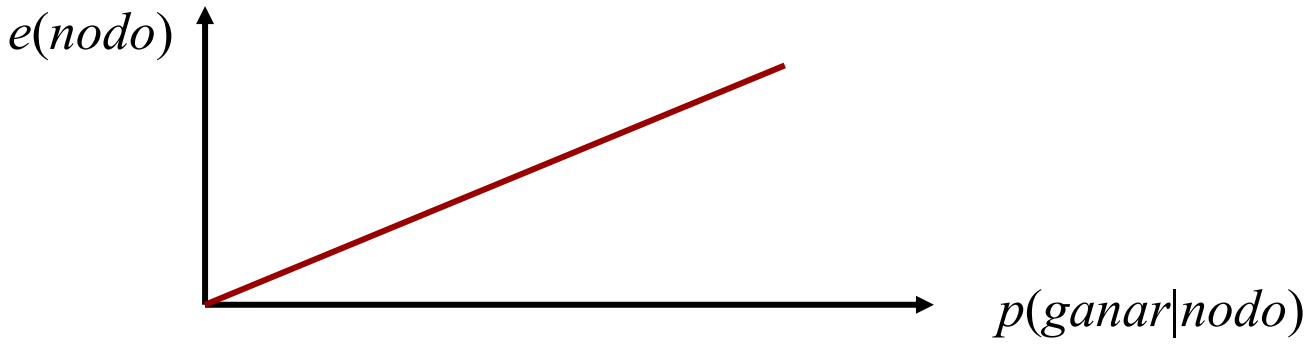
- la escala de los valores **sí** importa (no como en el algoritmo *Minimax*):



- las funciones no deben devolver $+\infty$ o $-\infty$:
  - ✓ los nodos azar tendrían siempre valores $+\infty$ o $-\infty$
  - ✓ escalar los valores a un intervalo finito (p.e. entre 0 y 1)

Funciones de evaluación

- La función de evaluación *e* debe estimar la probabilidad de ganar
  - ✓ Caso ideal: *e* es una *transformación lineal positiva* de dicha probabilidad



- Habitualmente, no es factible establecer una función *e* que cumple este criterio para todos los estados del juego (nodos)
- Normalmente, una función *e* es más "exacta", cuanto más cerca está el estado actual del juego de un estado terminal
- Es preferible aplicar la búsqueda *(Expect)Minimax* en el máximo número de *plys* posibles, y sólo al final aplicar la función de evaluación *e*

Complejidad *ExpectMinimax*

- Complejidad: proporcional al número de nodos en el árbol
  - ✓nivel de suspensión *d* (*plys*), factor de ramificación *b* (jugadas posibles), elementos de azar con *n* posibilidades
  - ✓*ExpectMinimax* (incluyendo nodos azar): $O(b^d \cdot n^d)$
  - ✓Ejemplo Backgammon: n=21 (2 dados) y $b \approx 20$

Número de *plys d* anticipados	Número de nodos en el árbol
1	20*21=420
2	176.400
3	74.088.000

- Aplicación de la poda α-β al algoritmo *ExpectMinimax*:
  - ✓véase Russell/Norvig 3ª ed., sección 5.5.1