

Razonamiento basado en árboles en CLIPS.

Ramón Trinidad Acedo

10 de febrero de 2026

- 1 Árboles de decisión. Definición y ejemplos
- 2 Implementación en CLIPS
- 3 Diferencias con razonamiento basado en casos

Árboles de decisión. ¿Qué son?

Definición

Un árbol de decisión es un grafo compuesto por nodos y ramas. Las ramas conectan nodos padres con nodos hijos (de arriba a abajo). El único nodo que no tiene padre en el árbol se denomina nodo raíz. Los demás nodos tienen un único nodo padre (no hay ciclos). Los nodos que no tienen hijos son los nodos hoja.

Árbol de decisión como sistema experto

Consiguen soluciones mediante la reducción del conjunto de posibles soluciones gracias a una consecución de preguntas que podan el espacio de búsqueda.

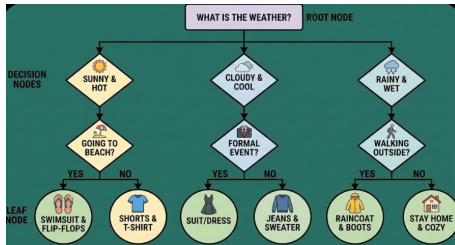


Figura 1: Ejemplo ilustrado de sistema de árbol de decisión

Nodos de decisión y nodos respuesta

- Nodos respuesta: Se corresponden con los nodos hoja, es decir son todas las posibles soluciones que puede devolver el árbol.
- Nodos de decisión: Representan las preguntas que una vez respondidas determinan la rama del árbol que se ha de seguir.

Árboles de decisión binarios

Consideremos la siguiente heurística para seleccionar apropiadamente qué vino servir en una comida, y la queremos representar en forma de árbol:

- 1: **if** el plato principal es carne roja **then**
- 2: Servir vino tinto
- 3: **else if** el plato principal es ave **y** es pavo **then**
- 4: Servir vino tinto
- 5: **else if** el plato principal es ave **y no** es pavo **then**
- 6: Servir vino blanco
- 7: **else if** el plato principal es pescado **then**
- 8: Servir vino blanco
- 9: **end if**

La manera en la que representamos este conjunto de condiciones lógicas como árbol binario sería:

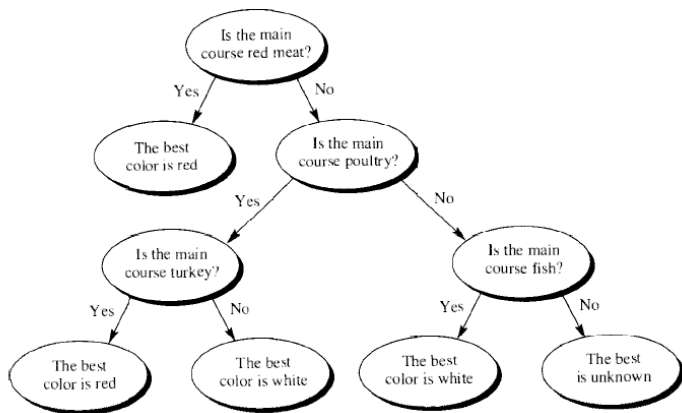


Figura 2: Ejemplo

Algoritmo para llegar a un nodo respuesta

```
1: procedure respuesta_arbol_binario
2:   Establecer la ubicación actual al nodo raíz.
3:   while localización actual es nodo de decisión do
4:     Responder la pregunta del nodo actual
5:     if Respuesta es SÍ then
6:       Mover localización a rama SÍ
7:     else
8:       Mover localización a rama NO
9:     end if
10:  end while
11:  return respuesta del nodo respuesta actual
12: end procedure
```

Árboles de decisión con múltiples ramas

En el caso anterior hemos necesitado hasta cuatro nodos de profundidad en el árbol para representar el problema del color del vino (y eso que era una heurística bastante sencilla).

Solución: Árboles de decisión con ramas múltiple

Si cambiáramos el formato de las preguntas SÍ/NO a preguntas del estilo ¿Cuál es el plato principal?, podríamos sacar cuatro ramas del nodo de decisión con los cuatro tipos de platos diferentes.

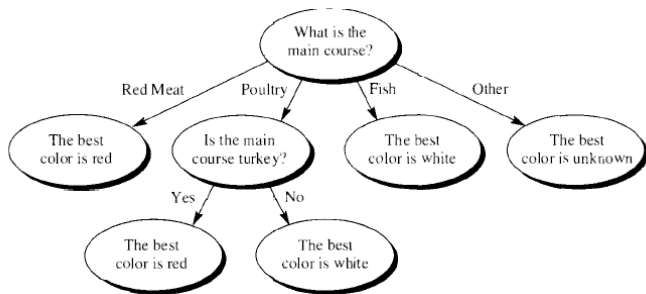


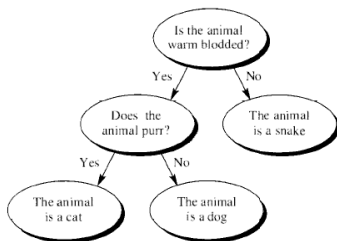
Figura 3: Ejemplo

La manera de navegar en este caso por el árbol para obtener las respuestas es:

- 1: **procedure** respuesta_arbol
- 2: Establecer la ubicación actual al nodo raíz.
- 3: **while** la ubicación actual es un nodo de decisión **do**
- 4: Hacer la pregunta en el nodo actual y
- 5: escoger la rama correspondiente.
- 6: Establecer el nodo actual al nodo hijo de la
- 7: rama asociada con la elección seleccionada.
- 8: **end while**
- 9: **return** la respuesta en el nodo actual.
- 10: **end procedure**

Estos árboles también pueden aprender

Podemos añadir conocimiento a un árbol de decisión. Consideremos el siguiente ejemplo de identificación de animales.



Aplicando el algoritmo `respuesta_arbol` obtenemos una solución. Sobre esta solución nos hacemos una pregunta: ¿Es esta la respuesta que buscábamos obtener? Si lo es, no hacemos nada. Si no lo es, modificamos el árbol con más conocimiento para obtener la respuesta esperada.

Añadir conocimiento a un árbol

El proceso para añadir conocimiento al árbol de los animales sería:

```
Is the animal warm-blooded? (yes or no) yes␣  
Does the animal purr? (yes or no) no␣  
I guess it is a dog  
Am I correct? (yes or no) no␣  
What is the animal? bird␣  
What question when answered yes will distinguish  
    a bird from a dog? Does the animal fly?␣  
Now I can guess bird  
Try again? (yes or no) no␣
```

y así sucesivamente podemos añadir cada vez más y más información.

Problema: árboles ineficientes

La desventaja es que de esta manera el árbol crece de manera irregular y carece de una forma jerárquica eficiente, ya que existen ramas muy profundas y otras con muy pocos nodos.

- 1: Preguntar si la respuesta en el nodo actual es correcta.
- 2: **if** la respuesta es correcta **then**
- 3: **return** la respuesta correcta.
- 4: **else**
- 5: Determinar la respuesta correcta real.
- 6: Determinar una pregunta que, al responder **sí**,
- 7: distinga la respuesta actual de la correcta.
- 8: Reemplazar el nodo respuesta por un nodo de decisión:
- 9: - Rama **No**: El nodo respuesta actual.
- 10: - Rama **Sí**: Un nodo con la respuesta correcta.
- 11: Establecer la pregunta del nuevo nodo como la
- 12: pregunta que distingue ambas respuestas.
- 13: **end if**

¿Cómo lo representamos?

La estructura del árbol debe ser móvil (ya hemos visto que pueden aprender). Luego, es más adecuado representar el árbol mediante hechos en vez de reglas. Usaremos la siguiente plantilla:

```
(deftemplate node
  (slot name)
  (slot type)
  (slot question)
  (slot yes_node)
  (slot no_node)
  (slot answer))
```

Árbol de animales en CLIPS

Vamos a construir el árbol inicial (antes de aprender) para identificar animales. La manera de definirlo en CLIPS es:

```
(node (name root) (type decision)
      (question "El animal es de sangre caliente?")
      (yes_node node1) (no_node node2))
(node (name node1) (type decision)
      (question "El animal ronrronea?")
      (yes_node node3) (no_node node4))
(node (name node2) (type answ) (answer serpiente))
(node (name node3) (type answ) (answer gato))
(node (name node4) (type answ) (answer perro))
```

Guardamos estos hechos iniciales en 'animal.dat'.

Algoritmo respuesta_arbol en CLIPS

Definimos la regla de inicialización para definir la ubicación actual en el nodo raíz y cargar los datos.

```
(defrule initialize
  (not (node (name root)))
=>
  (load-facts "animal.dat")
  (assert (current-node root)))
```

El siguiente paso es definir la función de pregunta, que pedirá una respuesta SÍ/NO. Esta función no solo nos servirá para las preguntas a los nodos, si no como preguntas al ingeniero del conocimiento. Y después la regla para preguntar a los nodos que se activa cuando el nodo actual es de decisión y no hay respuesta todavía.

Función y regla de pregunta

```
(deffunction ask-yes-or-no (?questionn)
  (printout t ?questionn " (yes or no) ")
  (bind ?answerr (read))
  (while (and (neq ?answerr yes) (neq ?answerr no))
    (printout t ?questions " (yes or no) ")
    (bind ?answerr (read)))
  (return ?answerr))

(defrule ask-decision-node
  ?node <- (current-node ?namee)
  (node (name ?namee)
        (type decision)
        (question ?questionn))
  (not (answer ?))
  =>
  (assert (answ (ask-yes-or-no ?questionn))))
```

Reglas ramas SÍ/NO

Una vez respondida la pregunta una de estas dos reglas se activará.

```
(defrule proceed-to-yes-branch
  ?node <- (current-node ?namee)
  (node (name ?namee)
        (type decision)
        (yes-node ?yes-branch))
  ?answerr <- (answ yes)
  =>
  (retract ?node ?answerr)
  (assert (current-node ?yes-branch)))
```

```
(defrule proceed-to-no-branch ...
```

La segunda regla se obvia por ser análoga a la primera Si la respuesta es sí, cambiamos la ubicación actual al nodo SÍ y viceversa para NO. Nótese que hemos eliminado el fact answ por lo que la regla ask-decision-node se activará de nuevo con el nuevo current-node.

Preguntamos al ingeniero del conocimiento si la respuesta es correcta.

```
(defrule ask-if-answernode-is-correct
  ?node <- (current-node ?namee)
  (node (name ?namee) (type answ)
        (answer ?value))
  (not (answ ?))
=>
  (printout t "I guess it is a " ?value crlf)
  (assert
    (answer (ask-yes-or-no "Am I correct?"))))
```

Fase de aprendizaje

La respuesta a la pregunta anterior activa una de estas dos reglas.

```
(defrule answernode-guess-is-correct
  ?node <- (current-node ?name)
  (node (name ?name) (type answer))
  ?answer <- (answer yes)
=>
  (assert (ask-try-again))
  (retract ?node ?answer))
```

```
(defrule answernode-guess-is-incorrect
  ?node <- (current-node ?namee)
  (node (name ?namee) (type answer))
  ?answer <- (answer no)
=>
  (assert (replace-answernode ?namee))
  (retract ?node ?answer))
```

Fase de aprendizaje

Si la respuesta es correcta, se activara una regla para que se pregunte al usuario si quiere probar de nuevo el árbol con otro animal o prefiere parar y guardar el fichero .animal.dat con la estructura final del árbol. Obviamos esas reglas para pasar directamente al caso de que la respuesta sea incorrecta:

```
(defrule replace-answer-node
  ?phase <- (replace-answer-node ?namee)
  ?data <- (node (name ?namee
                  (type ans)
                  (answer ?value)))
  =>
  (retract ?phase)
  (printout t "  Qu    animal es?")
  (bind ?new-animal (read)))
```

Fase de aprendizaje

```
(printout t "  Qu    pregunta distingue a" ?new-  
  animal "de " ?value)  
(bind ?question (readline))  
(bind ?newnode1 (gensym*))  
(bind ?newnode2 (gensym*))  
(modify ?data (type decision)  
          (question ?questionn)  
          (yes-node ?newnode1)  
          (no-node ?newnode2))  
(assert (node (name ?newnode1)  
              (type ans)  
              (answer ?new-animal)))  
(assert (node (name ?newnode1)  
              (type ans)  
              (answer ?value)))  
(assert (ask-try-again)))
```

Razonamiento basado en casos

CBR

Resuelve problemas adaptando soluciones de casos similares. La información se almacena en casos. El CBR opera en un ciclo: Recuperar, Reutilizar, Revisar y Retener.

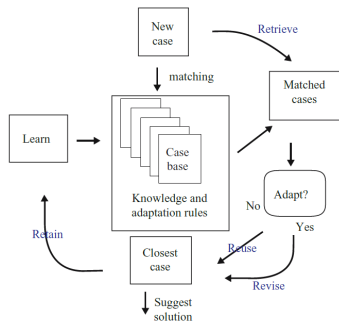


Figura 4: Esquema de un sistema CBR.

- ➊ Recuperar: Ante un problema nuevo, el sistema primero recupera de su memoria los casos que más se le parezcan.
- ➋ Reutilizar: Tras la recuperación, reutiliza este caso para aportar una solución al problema actual.
- ➌ Revisar: Se revisa la solución por si hay que ajustarla al nuevo contexto. Aquí normalmente interviene el experto humano.
- ➍ Retener: Es capaz de almacenar esta nueva experiencia, tanto el problema como la solución ya revisada. Este paso es crucial y permite construir sistemas enormemente eficientes ante problemas complejos.

Diferencias árboles de decisión y CBR

Resumiremos las diferencias de razonamiento de estos sistemas expertos en la siguiente tabla.

Factor	Basado en Reglas (Árboles)	Basado en Casos (CBR)
Conocimiento	Reglas explícitas y pequeñas	Casos grandes e implícitos
Dominio	Bien entendido	Complejo o poco entendido
Coincidencia	Exacta	Parcial

CBR vs Árboles

Los árboles son sistemas expertos basados en pequeñas reglas, utilizan conocimientos explícitos que requieren coincidencias exactas, mientras que el CBR emplea experiencias pasadas permitiendo coincidencias parciales y soluciones en dominios complejos.