UNIVERSITAT POLITÈCNICA DE VALÈNCIA

SIN2122: examen del bloc 1
5 de novembre de 2021

Alfons Juan

DSIC

Departament de Sistemes Informàtics i Computació

2021-11-05: Qüestió 1

Donada la base de fets inicial: BF={`(llista 6 3 5 1 4 7 2 6 3) (parells 0)`} i la següent regla per a calcular el nombre de parells en una llista de nombres naturals

```
(defrule comptar-parells
  ?f1 <- (llista $?a ?b $?c)
  ?f2 <- (parells ?p)
  (test (= 0 (mod ?b 2)))
  =>
  (assert (llista $?a $?c))
  (assert (parells (+ 1 ?p))))
```

Si el nostre objectiu és obtenir una base de fets final (després de l'execució successiva de la regla) en la qual el fet (parells...) només pot aparèixer una vegada (contenint el nombre de parells en la llista). Quina de les següents afirmacions és CERTA per tal d'obtenir el nostre objectiu?:

A) La regla és correcta

B) Seria necessari afegir `(retract ?f1)`

C) Seria necessari afegir `(retract ?f1)` i `(retract ?f2)`

D) Seria necessari afegir `(retract ?f2)`

Q1A.clp

```
(deffacts bf
  (llista 6 3 5 1 4 7 2 6 3)
  (parells 0))
(defrule comptar-parells
  ?f1 <- (llista $?a ?b $?c)
  ?f2 <- (parells ?p)
  (test (= 0 (mod ?b 2)))
  =>
  (assert (llista $?a $?c))
  (assert (parells (+ 1 ?p))))
(watch facts)
(watch activations)
(progn (reset) (run) (exit))
```

Genera `(parells n)` per a tot $n = 1, 2, \ldots$ i no acaba.

Sense `(retract ?f1)` torna a instanciar f-1 amb `(parells n+1)`.

clips -f2 Q1A.clp

```
<== f-0     (initial-fact)
==> f-0     (initial-fact)
==> f-1     (llista 6 3 5 1 4 7 2 6 3)
==> f-2     (parells 0)
==> A 0 comptar-parells: f-1,f-2
==> A 0 comptar-parells: f-1,f-2
==> A 0 comptar-parells: f-1,f-2
==> A 0 comptar-parells: f-1,f-2
==> f-3     (llista 6 3 5 1 4 7 2 3)
==> A 0 comptar-parells: f-3,f-2
==> A 0 comptar-parells: f-3,f-2
==> A 0 comptar-parells: f-3,f-2
==> f-4     (parells 1)
==> A 0 comptar-parells: f-3,f-4
==> A 0 comptar-parells: f-3,f-4
==> A 0 comptar-parells: f-3,f-4
==> A 0 comptar-parells: f-1,f-4
==> A 0 comptar-parells: f-1,f-4
==> A 0 comptar-parells: f-1,f-4
==> A 0 comptar-parells: f-1,f-4
==> f-5     (parells 2)
==> A 0 comptar-parells: f-3,f-5
==> A 0 comptar-parells: f-3,f-5
==> A 0 comptar-parells: f-3,f-5
==> A 0 comptar-parells: f-1,f-5
==> A 0 comptar-parells: f-1,f-5
==> A 0 comptar-parells: f-1,f-5
==> A 0 comptar-parells: f-1,f-5
==> f-6     (parells 3)
==> A 0 comptar-parells: f-3,f-6
==> A 0 comptar-parells: f-3,f-6
==> A 0 comptar-parells: f-3,f-6
==> A 0 comptar-parells: f-1,f-6
==> A 0 comptar-parells: f-1,f-6
==> A 0 comptar-parells: f-1,f-6
==> A 0 comptar-parells: f-1,f-6
==> f-7     (parells 4)
...
```

Q1B.clp

```
(deffacts bf
  (llista 6 3 5 1 4 7 2 6 3)
  (parells 0))
(defrule comptar-parells
  ?f1 <- (llista $?a ?b $?c)
  ?f2 <- (parells ?p)
  (test (= 0 (mod ?b 2)))
  =>
  (retract ?f1)
  (assert (llista $?a $?c))
  (assert (parells (+ 1 ?p))))
(watch facts)
(watch activations)
(progn (reset) (run) (exit))
```

Genera `(parells n)` per a $n = 1, 2, 3, 4$ i acaba.

Sense `(retract ?f2)` no és possible acabar amb un únic `(parells n)` on n siga el nombre de parells de la llista.

clips -f2 Q1B.clp

```
<== f-0     (initial-fact)
==> f-0     (initial-fact)
==> f-1     (llista 6 3 5 1 4 7 2 6 3)
==> f-2     (parells 0)
==> Activation 0      comptar-parells: f-1,f-2
==> Activation 0      comptar-parells: f-1,f-2
==> Activation 0      comptar-parells: f-1,f-2
==> Activation 0      comptar-parells: f-1,f-2
<== f-1     (llista 6 3 5 1 4 7 2 6 3)
<== Activation 0      comptar-parells: f-1,f-2
<== Activation 0      comptar-parells: f-1,f-2
<== Activation 0      comptar-parells: f-1,f-2
==> f-3     (llista 6 3 5 1 4 7 2 3)
==> Activation 0      comptar-parells: f-3,f-2
==> Activation 0      comptar-parells: f-3,f-2
==> Activation 0      comptar-parells: f-3,f-2
==> f-4     (parells 1)
==> Activation 0      comptar-parells: f-3,f-4
==> Activation 0      comptar-parells: f-3,f-4
==> Activation 0      comptar-parells: f-3,f-4
<== f-3     (llista 6 3 5 1 4 7 2 3)
<== Activation 0      comptar-parells: f-3,f-4
<== Activation 0      comptar-parells: f-3,f-2
<== Activation 0      comptar-parells: f-3,f-4
<== Activation 0      comptar-parells: f-3,f-2
<== Activation 0      comptar-parells: f-3,f-2
==> f-5     (llista 6 3 5 1 4 7 3)
==> Activation 0      comptar-parells: f-5,f-2
==> Activation 0      comptar-parells: f-5,f-4
==> Activation 0      comptar-parells: f-5,f-2
==> Activation 0      comptar-parells: f-5,f-4
==> f-6     (parells 2)
==> Activation 0      comptar-parells: f-5,f-6
==> Activation 0      comptar-parells: f-5,f-6
<== f-5     (llista 6 3 5 1 4 7 3)
<== Activation 0      comptar-parells: f-5,f-6
<== Activation 0      comptar-parells: f-5,f-4
<== Activation 0      comptar-parells: f-5,f-2
<== Activation 0      comptar-parells: f-5,f-4
<== Activation 0      comptar-parells: f-5,f-2
==> f-7     (llista 6 3 5 1 7 3)
==> Activation 0      comptar-parells: f-7,f-2
==> Activation 0      comptar-parells: f-7,f-4
==> Activation 0      comptar-parells: f-7,f-6
==> f-8     (parells 3)
==> Activation 0      comptar-parells: f-7,f-8
<== f-7     (llista 6 3 5 1 7 3)
<== Activation 0      comptar-parells: f-7,f-6
<== Activation 0      comptar-parells: f-7,f-4
<== Activation 0      comptar-parells: f-7,f-2
==> f-9     (llista 3 5 1 7 3)
==> f-10    (parells 4)
```

Q1C.clp

```
(deffacts bf
  (llista 6 3 5 1 4 7 2 6 3)
  (parells 0))
(defrule comptar-parells
  ?f1 <- (llista $?a ?b $?c)
  ?f2 <- (parells ?p)
  (test (= 0 (mod ?b 2)))
  =>
  (retract ?f1) (retract ?f2)
  (assert (llista $?a $?c))
  (assert (parells (+ 1 ?p))))
(watch facts)
(watch activations)
(progn (reset) (run) (exit))
```

OK!

clips -f2 Q1C.clp

```
<== f-0     (initial-fact)
==> f-0     (initial-fact)
==> f-1     (llista 6 3 5 1 4 7 2 6 3)
==> f-2     (parells 0)
==> Activation 0      comptar-parells: f-1,f-2
==> Activation 0      comptar-parells: f-1,f-2
==> Activation 0      comptar-parells: f-1,f-2
==> Activation 0      comptar-parells: f-1,f-2
<== f-1     (llista 6 3 5 1 4 7 2 6 3)
<== Activation 0      comptar-parells: f-1,f-2
<== Activation 0      comptar-parells: f-1,f-2
<== Activation 0      comptar-parells: f-1,f-2
<== f-2     (parells 0)
==> f-3     (llista 6 3 5 1 4 7 2 3)
==> f-4     (parells 1)
==> Activation 0      comptar-parells: f-3,f-4
==> Activation 0      comptar-parells: f-3,f-4
==> Activation 0      comptar-parells: f-3,f-4
<== f-3     (llista 6 3 5 1 4 7 2 3)
<== Activation 0      comptar-parells: f-3,f-4
<== Activation 0      comptar-parells: f-3,f-4
<== f-4     (parells 1)
==> f-5     (llista 6 3 5 1 4 7 3)
==> f-6     (parells 2)
==> Activation 0      comptar-parells: f-5,f-6
==> Activation 0      comptar-parells: f-5,f-6
<== f-5     (llista 6 3 5 1 4 7 3)
<== Activation 0      comptar-parells: f-5,f-6
<== f-6     (parells 2)
==> f-7     (llista 6 3 5 1 7 3)
==> f-8     (parells 3)
==> Activation 0      comptar-parells: f-7,f-8
<== f-7     (llista 6 3 5 1 7 3)
<== f-8     (parells 3)
==> f-9     (llista 3 5 1 7 3)
==> f-10    (parells 4)
```

Q1D.clp

```
(deffacts bf
  (llista 6 3 5 1 4 7 2 6 3)
  (parells 0))
(defrule comptar-parells
  ?f1 <- (llista $?a ?b $?c)
  ?f2 <- (parells ?p)
  (test (= 0 (mod ?b 2)))
  =>
  (retract ?f2)
  (assert (llista $?a $?c))
  (assert (parells (+ 1 ?p))))
(watch facts)
(watch activations)
(progn (reset) (run) (exit))
```

Genera `(parells n)` per a tot $n = 1, 2, \ldots$ i no acaba.

Sense `(retract ?f1)` torna a instanciar f-1 amb `(parells n+1)`.

clips -f2 Q1D.clp

```
<== f-0     (initial-fact)
==> f-0     (initial-fact)
==> f-1     (llista 6 3 5 1 4 7 2 6 3)
==> f-2     (parells 0)
==> A 0 comptar-parells: f-1,f-2
==> A 0 comptar-parells: f-1,f-2
==> A 0 comptar-parells: f-1,f-2
==> A 0 comptar-parells: f-1,f-2
<== f-2     (parells 0)
<== A 0 comptar-parells: f-1,f-2
<== A 0 comptar-parells: f-1,f-2
<== A 0 comptar-parells: f-1,f-2
==> f-3     (llista 6 3 5 1 4 7 2 3)
==> f-4     (parells 1)
==> A 0 comptar-parells: f-3,f-4
==> A 0 comptar-parells: f-3,f-4
==> A 0 comptar-parells: f-3,f-4
==> A 0 comptar-parells: f-1,f-4
==> A 0 comptar-parells: f-1,f-4
==> A 0 comptar-parells: f-1,f-4
==> A 0 comptar-parells: f-1,f-4
<== f-4     (parells 1)
<== A 0 comptar-parells: f-1,f-4
<== A 0 comptar-parells: f-1,f-4
<== A 0 comptar-parells: f-1,f-4
<== A 0 comptar-parells: f-3,f-4
<== A 0 comptar-parells: f-3,f-4
<== A 0 comptar-parells: f-3,f-4
==> f-5     (parells 2)
==> A 0 comptar-parells: f-3,f-5
==> A 0 comptar-parells: f-3,f-5
==> A 0 comptar-parells: f-3,f-5
==> A 0 comptar-parells: f-1,f-5
==> A 0 comptar-parells: f-1,f-5
==> A 0 comptar-parells: f-1,f-5
==> A 0 comptar-parells: f-1,f-5
<== f-5     (parells 2)
...
```

2021-11-05: Qüestió 2

Es vol una regla en CLIPS que faça matching amb el següent fet: `(llista a b a a b b c a b c)`. Quin dels següents patrons caldria incloure en la part esquerra de la dita regla?

A) ***`(llista $? $?x $? $?x ?)`***

B) `(llista $?y ?x ?x $?y $?)`

C) `(?a $?c)`

D) `(llista $? $?y ?x ?x ?x $? $?y $? ?x)`

Q2A.clp

```
(deffacts bf (llista a b a a b b c a b c))
(defrule R (llista $?a $?x $?b $?x ?c) =>
  (printout t "x=" $?x " a=" $?a " b=" $?b " c=" ?c crlf))
(progn (watch facts) (watch activations) (reset) (run) (exit))
```

clips -f2 Q2A.clp

```
<== f-0     (initial-fact)
==> f-0     (initial-fact)
==> f-1     (llista a b a a b b c a b c)
==> Activation 0      R: f-1
==> Activation 0      R: f-1
==> Activation 0      R: f-1
==> Activation 0      R: f-1
==> Activation 0      R: f-1
==> Activation 0      R: f-1
==> Activation 0      R: f-1
==> Activation 0      R: f-1
==> Activation 0      R: f-1
==> Activation 0      R: f-1
==> Activation 0      R: f-1
==> Activation 0      R: f-1
==> Activation 0      R: f-1
==> Activation 0      R: f-1
x=() a=() b=(a b a a b b c a b) c=c
x=(a b) a=() b=(a a b b c) c=c
x=() a=(a) b=(b a a b b c a b) c=c
x=(b) a=(a) b=(a a b b c a) c=c
x=() a=(a b) b=(a a b b c a b) c=c
x=() a=(a b a) b=(a b b c a b) c=c
x=(a b) a=(a b a) b=(b c) c=c
x=() a=(a b a a) b=(b b c a b) c=c
x=(b) a=(a b a a) b=(b c a) c=c
x=() a=(a b a a b) b=(b c a b) c=c
x=(b) a=(a b a a b) b=(c a) c=c
x=() a=(a b a a b b) b=(c a b) c=c
x=() a=(a b a a b b c) b=(a b) c=c
x=() a=(a b a a b b c a) b=(b) c=c
x=() a=(a b a a b b c a b) b=() c=c
```

Q2B.clp

```
(deffacts bf (llista a b a a b b c a b c))
(defrule R (llista $?y ?x ?x $?y $?a) =>
  (printout t "x= " ?x " y=" $?y " a=" $?a crlf))
(progn (watch facts) (watch activations) (reset) (run) (exit))
```

clips -f2 Q2B.clp

```
<== f-0     (initial-fact)
==> f-0     (initial-fact)
==> f-1     (llista a b a a b b c a b c)
```

Q2C.clp

```
(deffacts bf (llista a b a a b b c a b c))
(defrule R (?a $?c) =>  (printout t "a=" ?a " c=" $?c crlf))
(progn (reset) (run) (exit))
```

clips -f2 Q2C.clp

```
[PRNTUTIL2] Syntax Error:  Check appropriate syntax for the first field
↪  of a pattern.

ERROR:
(defrule MAIN::R
   (?a
```

Q2D.clp

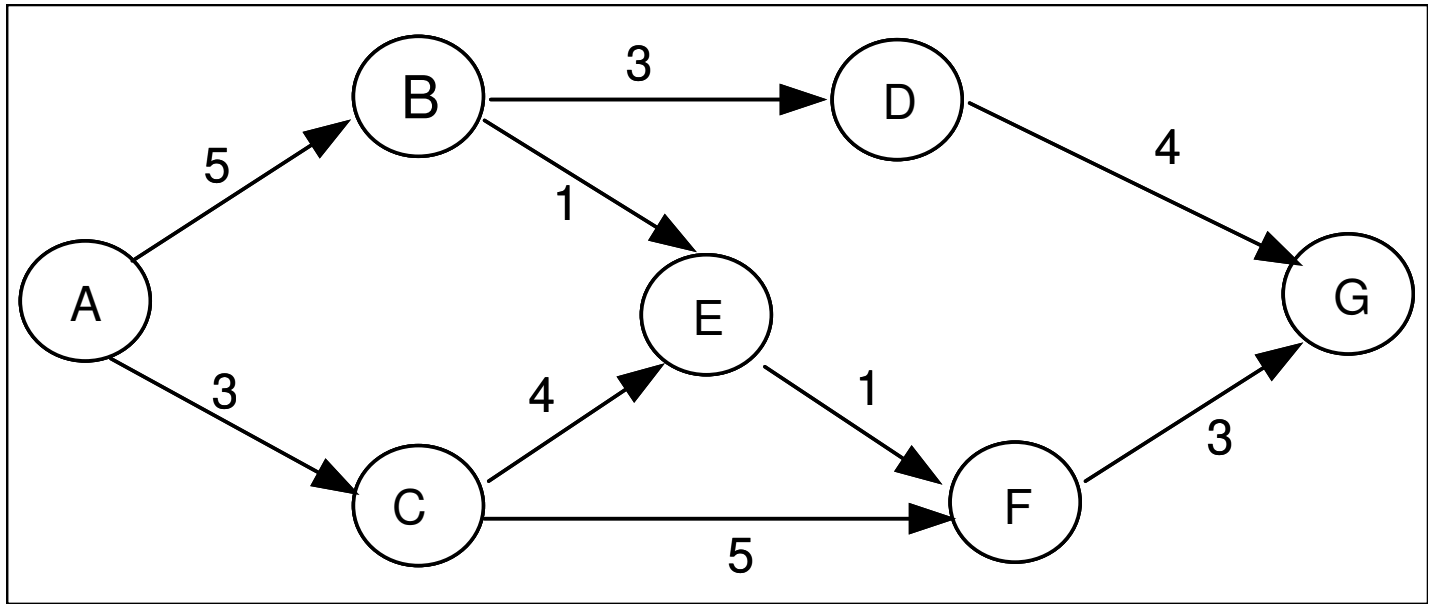
```
(deffacts bf (llista a b a a b b c a b c))
(defrule R (llista $?a $?y ?x ?x ?x $?b $?y $?c ?x) =>
  (printout t "x= " ?x " y=" $?y " a=" $?a " b=" $?b " c=" $?c crlf))
(progn (watch facts) (watch activations) (reset) (run) (exit))
```

clips -f2 Q2D.clp

```
<== f-0     (initial-fact)
==> f-0     (initial-fact)
==> f-1     (llista a b a a b b c a b c)
```

2021-11-05: Qüestió 3

Donat el graf de la figura, on el node G és el node Meta, indica la resposta CORRECTA (davant dos nodes amb el mateix valor de f(n) s'expandeix abans el node alfabèticament anterior):



A) Una cerca per aprofundiment iteratiu troba la solució en 5 iteracions ***No, en 3, ABDG***

B) Una cerca en amplària troba la solució de cost òptim ***No, ABDG, cost 12; òptim ABEFG, cost 10***

C) Una cerca de cost uniforme troba la solució més curta ***No, ABEFG***

D) Una cerca en profunditat amb màxim nivell de profunditat m=4 troba la solució més curta ***Sí, ABDG***

2021-11-05: Qüestió 4

Siga una cerca de tipus A (f(n)=g(n)+h(n)) on la funció h(n) és admissible i consistent. L'algorisme retorna una solució des del node inicial A al node objectiu G que travessa un node n1. Indica quina de les següents afirmacions és CORRECTA:

A) f(n1) $<=$ f(G) ***Sí, n1 en camí òptim i h admissible***

f(n1) = g(n1) + h(n1) $<=$ g(n1) + h*(n1) = f(G)

B) f(G) $<$ h*(A) ***f(G) = h*(A)***

C) h*(A) $<$ h(n1) ***h*(A) = g(n1) + h*(n1) $\geq$ g(n1) + h(n1) $>$ h(n1)***

D) Cap de les opcions anteriors és correcta

2021-11-05: Qüestió 5

Siguen f1(n)=g(n)+h1(n), f2(n)=g(n)+h2(n) i f3(n)=g(n)+h3(n) tres funcions tal que se sap que h2(n) és admissible, h3(n) no ho és i $\forall$n h1(n) $<=$ h2(n). Assumint que G1 és el node solució que retorna la cerca amb f1, G2 el que retorna la cerca amb f2 i G3 el que retorna la cerca amb f3, indica la resposta CORRECTA:

A) Es compleix f1(G1) $<$ f2(G2) ***h2 admissible i domina h1 $\rightarrow$ h1 admissible; G1 i G2 tornen un òptim, f1(G1)=f2(G2)***

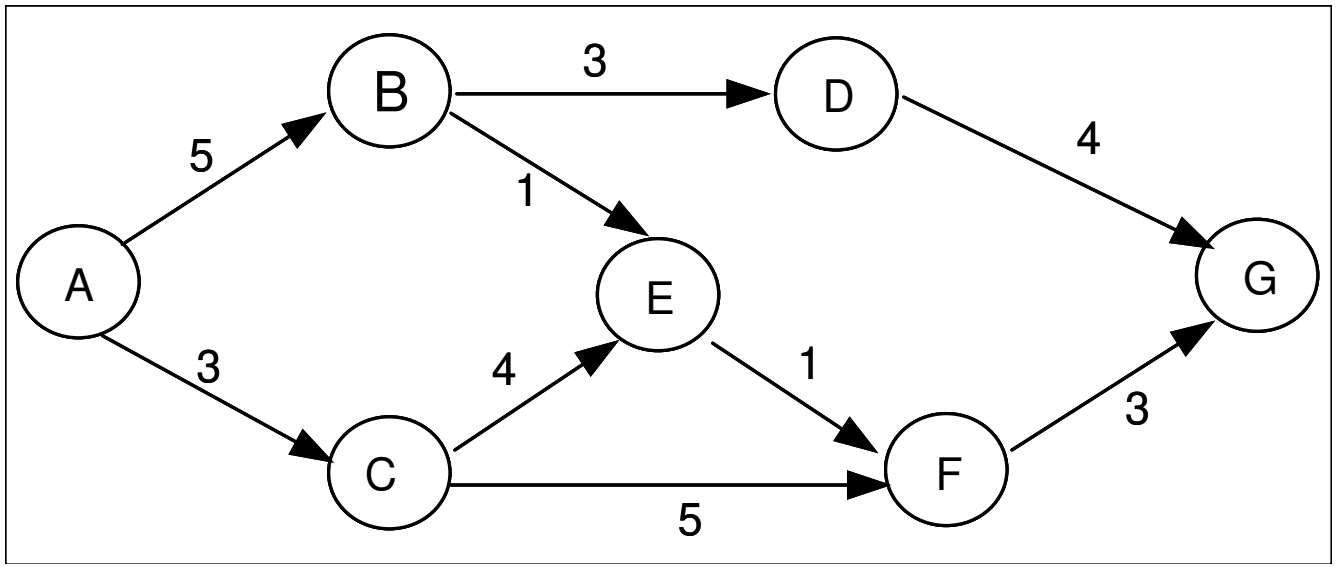
B) Es compleix f2(G2) $<=$ f3(G3) Sí, G2 és òptim però potser G3 no ja que h3 no és admissible

C) Es compleix g(G1) $<$ f2(G2) ***g(G1)=f(G1)=f(G2)***

D) Cap de les respostes anteriors és correcta

2021-11-05: Qüestió 6

Apliquem una heurística h(n) al graf de la pregunta 3 (G Meta)



Siguen h(A)=8 i h(n')=h(n)-2 per a tot node n i tot node successor d'n, n'. Si n' té i nodes pare, llavors h(n')=min(h(n1),...,h(ni))-2. Indica la resposta CORRECTA:

A) L'heurística és admissible ***No: A 8 B 6 $>$ 5=h*(B) C 6 D 4 E 4 F 2 G 0***

B) L'heurística és consistent ***No: 6=h(B) $\not\leq$ w(B,E)+h(E)=1+4=5***

C) L'aplicació d'una estratègia de tipus A (f(n)=g(n)+h(n)) no retorna la solució òptima perquè l'heurística no és admissible ***No, G única: A0+8 $\rightarrow$ C3+6 B5+6 $\rightarrow$ E6+4 F8+2 B5+6 $\rightarrow$ F7+2 B5+6 $\rightarrow$ G10+0 B5+6 $\rightarrow$ G***

D) Cap de les respostes anteriors és correcta

2021-11-05: Qüestió 7

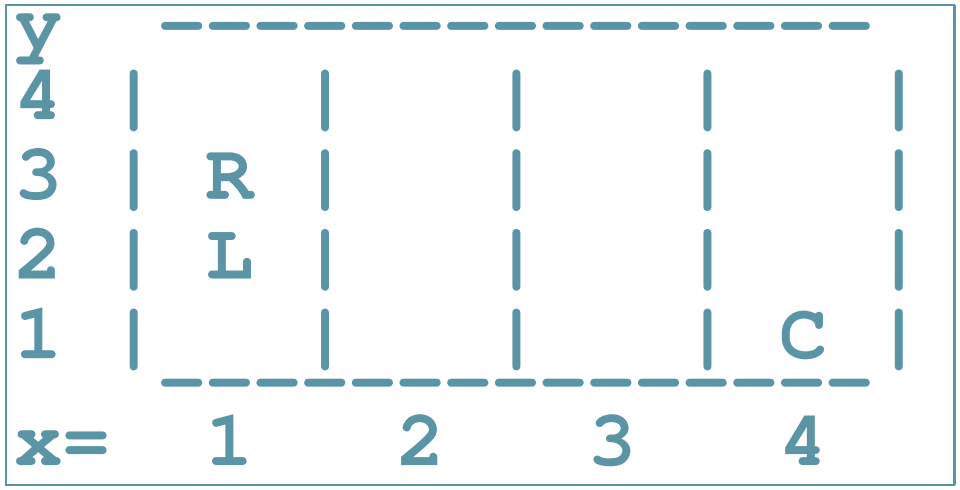
Siga un graella de 4x4 on la casella inferior esquerra és (x,y)= (1,1) i la casella superior dreta és la (x,y)=(4,4). Hi ha un robot a la casella (1,3), una llanda a la casella (1,2) i un contenidor triturador a la casella (4,1). El robot pot moure's en quatre direccions (dalt, baix, dreta i esquerra) i pot espentar la llanda en qualsevol de les quatre direccions, respectant els límits de la graella. Per a espentar una llanda, el robot ha de situar-se a una casella adjacent a la llanda. Com a resultat d'una acció d'espentar, la posició del robot i la posició de la llanda es desplacen una casella en la direcció de l'espenta, respectant els límits de la graella. L'objectiu del problema és espentar la llanda al contenidor triturador. Indica la resposta INCORRECTA aplicada a aquest estat del problema:

A) Hi ha dues accions de moviment del robot aplicables en l'estat. ***Sí, mou-dalt i mou-dreta***

B) Hi ha una acció d'espentar llanda aplicable. ***Sí, espenta-baix***

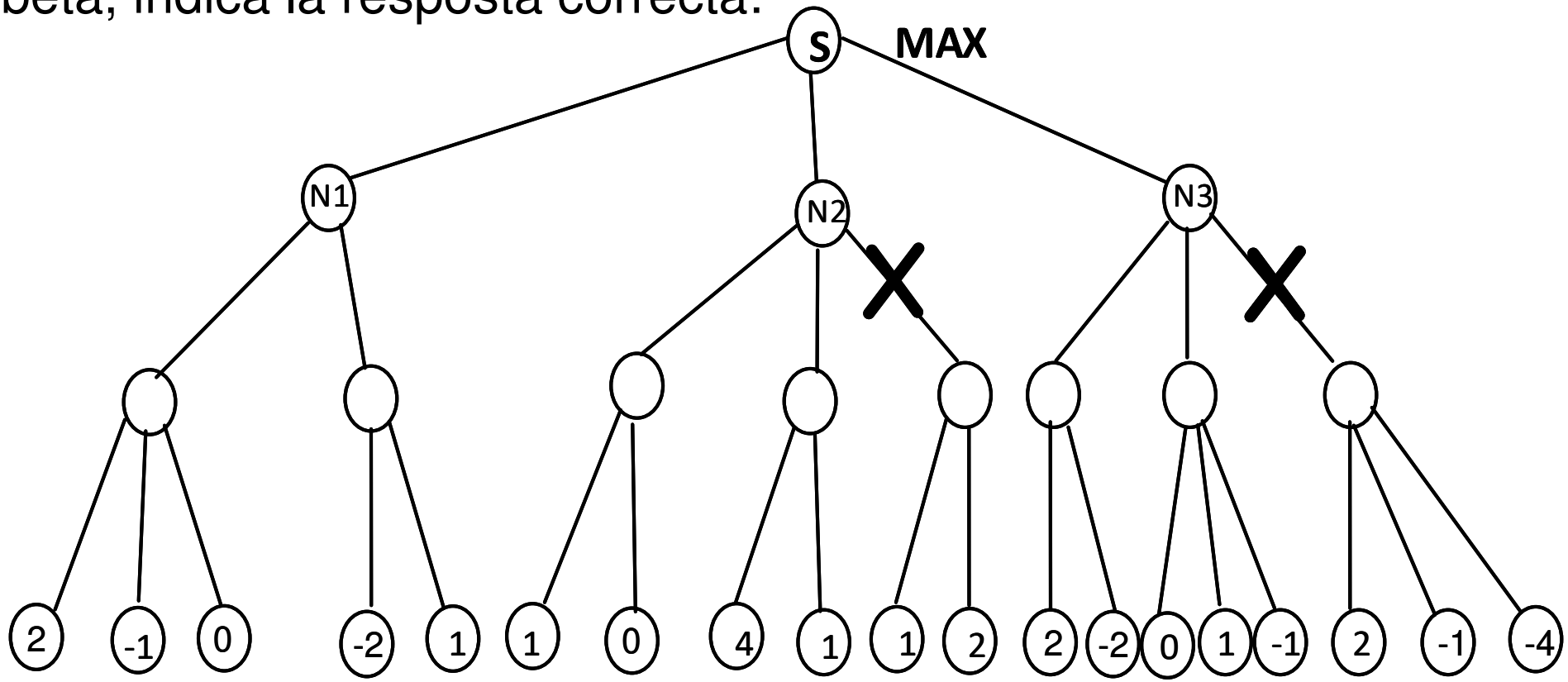
C) La solució òptima es troba en el nivell 5. No té solució

D) Una cerca en profunditat (limitada a màxim nivell m=5) expandiria el mateix nombre de nodes que els nodes expandits en els 5 primers nivells d'una cerca en amplària. ***Sí, perquè no té solució***



2021-11-05: Qüestió 8

Donat l'arbre de joc de la figura on s'ha aplicat un procediment alfa-beta, indica la resposta correcta:



A) Es produeix un tall en el node N2 que podaria també la branca intermèdia de N2

B) No es produeix tall en N2 i per tant no es poda la branca de la dreta de N2

C) Es produeix un tall en el node N3 que podaria també la branca intermèdia de N3

D) No es produeix tall en N3 i per tant no es poda la branca de la dreta de N3

S -i -i i, N1 i -i i, N1A -i -i i, N1A 2 2 i, N1 2 -i 2, N1B -i -i 2, N1B -2 -2 2, N1B 1 1 2, N1 1 -i 1, S 1 1 i, N2 i 1 i, N2A -i 1 i, N2A 1 1 i, N2 1 1 1 TALL, no explorem N2B ni N2C

2021-11-05: Qüestió 9

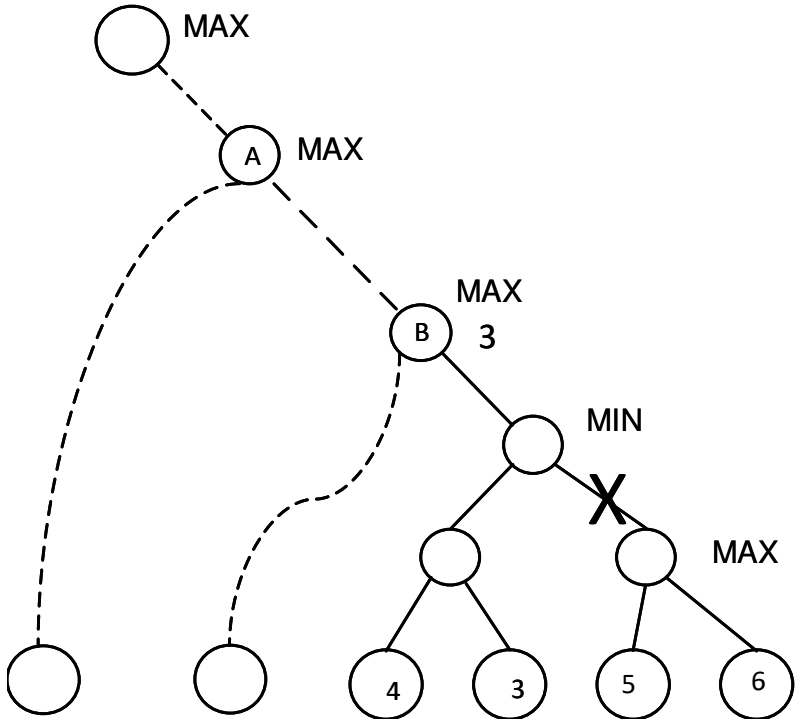
Donat el desenvolupament parcial d'una cerca alfa-beta indicat en la figura, quin valor provisional hauria de tenir el node A perquè es produïsca el tall de la figura?

A) Tots els valors compresos en l'interval $[-\infty, 3]$ provocarien el tall

B) Tots els valors compresos en l'interval $[3, +\infty]$ provocarien el tall

C) Tots els valors compresos en l'interval $[4, +\infty]$ provocarien el tall

D) El tall de la figura no es pot produir



A v [$\alpha \geq v$?], B 3 [$\alpha \geq \max(v, 3)$?], B2 i [$\alpha \geq \max(v, 3)$?], B2A -i [$\alpha \geq \max(v, 3)$?], B2A 4 [$\alpha \geq \max(v, 4)$?], B2 4 [$\alpha \geq \max(v, 3)$ $\beta \leq 4$], TALL si $\max(v, 3) \geq 4 \rightarrow v \geq 4$

2021-11-05: Problema

- Es disposa d'un conjunt de caixes escampades pel sòl cadascuna de les quals porta associada una etiqueta amb un nombre sencer. Pot haver-hi més d'una caixa amb el mateix número. Per exemple: 17, 5, 6, 22, 5, 4, 7, 12, 6, 1, 21, ...
- Es desitja apilar les caixes en torres en ordre decreixent del número d'etiqueta de manera que una caixa amb número M només es pot apilar damunt d'una caixa amb número N si M $<=$ N. Les condicions sobre les torres són:
  1. Pot haver-hi un nombre indeterminat de torres
  2. Hi ha dos tipus de torres: les que apilen caixes de nombre parell i les que apilen caixes de nombre imparell
  3. Pot haver-hi un nombre indeterminat de caixes en cada torre
- El patró per a representar un estat d'aquest problema és:

  `(apila [torre ct`^{m}`]`^{m} `sol cs`^{m}`)` on `ct` i `cs` són sencers
- El predicat `(evenp <arg>)` retorna TRUE si `<arg>` és parell.

Usant CLIPS i cerca en grafs (GRAPH-SEARCH), es demana:

1. Escriu la base de fets inicial sabent que inicialment hi ha dues torres, una conté les caixes 13 i 17, i l'altra conté la 22. A més, al sòl tenim les caixes: 17, 5, 6, 22, 5, 4, 7, 12, 6, 1, 21.

Prob1.clp

```
(deffacts bf
  (apila torre 13 17 torre 22 sol 17 5 6 22 5 4 7 12 6 1 21))
(progn (watch facts) (reset) (run) (exit))
```

clips -f2 Prob1.clp

```
==> f-1 (apila torre 13 17 torre 22 sol 17 5 6 22 5 4 7 12 6 1 21)
```

2. Assumint que ja existeixen diverses torres creades, escriu una única regla que permeta agafar una caixa qualsevol del sòl i apilar-la en alguna de les torres existents.

Prob2.clp

```
(deffacts bf
  (apila torre 13 17 torre 22 sol 17 5 6 22 5 4 7 12 6 1 21))
(defrule apila
  (apila $?x torre ?cim $?resta sol $?y ?caixa $?z)
  (test (or (and (evenp ?cim)(evenp ?caixa))
            (and (not (evenp ?cim))(not (evenp ?caixa)))))
  (test (<= ?caixa ?cim))
  => (assert (apila $?x torre ?caixa ?cim $?resta sol $?y $?z)))
(progn (watch facts) (watch activations) (reset) (run) (exit))
```

clips -f2 Prob2.clp | head -n 15

```
==> f-1     (apila torre 13 17 torre 22 sol 17 5 6 22 5 4 7 12 6 1 21)
==> Activation 0      apila: f-1
==> Activation 0      apila: f-1
==> Activation 0      apila: f-1
==> Activation 0      apila: f-1
==> Activation 0      apila: f-1
==> Activation 0      apila: f-1
==> Activation 0      apila: f-1
==> Activation 0      apila: f-1
==> Activation 0      apila: f-1
==> f-2     (apila torre 5 13 17 torre 22 sol 17 6 22 5 4 7 12 6 1 21)
==> Activation 0      apila: f-2
==> Activation 0      apila: f-2
==> Activation 0      apila: f-2
==> Activation 0      apila: f-2
```

3. Assumint que ja hi ha diverses torres creades, volem dissenyar una operació de desapilar una caixa amb etiqueta X d'una torre T (desapilar=llevar la caixa que està al cim de la torre) amb la condició que hi haja una o més torres diferents de T on la caixa que està al cim també tinga la mateixa etiqueta X. Escriu una única regla que permeta realitzar aquesta operació.

Prob3.clp

```
(deffacts bf
  (apila torre 13 17 torre 13 19 sol 17 5 6 22 5 4 7 12 6 1 21))
(defrule desapila
  (apila $?x1 torre ?cim $?x2 sol $?s)
  (test (or (member (create$ torre ?cim) $?x1)
            (member (create$ torre ?cim) $?x2)))
  => (assert (apila $?x1 torre $?x2 sol $?s)))
(progn (watch facts) (watch activations) (reset) (run) (exit))
```

clips -f2 Prob3.clp

```
==> f-1 (apila torre 13 17 torre 13 19 sol 17 5 6 22 5 4 7 12 6 1 21)
==> Activation 0  desapila: f-1
==> Activation 0  desapila: f-1
==> f-2 (apila torre 17 torre 13 19 sol 17 5 6 22 5 4 7 12 6 1 21)
==> f-3 (apila torre 13 17 torre 19 sol 17 5 6 22 5 4 7 12 6 1 21)
```

4. Assumint que ja hi ha diverses torres creades, escriu una única regla que retorne un missatge SI HI HA ALMENYS DUES caixes al sòl amb el mateix número d'etiqueta X tal que cap de les torres conté caixes amb la mateixa etiqueta X. La regla ha de mostrar el següent missatge: "Hi ha almenys dues caixes amb número X al sòl i cap torre conté caixes amb número X".

Prob4.clp

```
(deffacts bf
  (apila torre 13 17 torre 22 sol 17 5 6 22 5 4 7 12 6 1 21))
(defrule comprova
  (apila $?torres sol $?x1 ?c $?x2 ?c $?x3)
  (test (not (member ?c $?torres)))
  => (printout t "Hi ha almenys dues caixes amb numero " ?c " al sol i
  ↪  cap torre conté caixes amb aquest número " crlf))
(progn (watch facts) (watch activations) (reset) (run) (exit))
```

clips -f2 Prob4.clp

```
==> f-1 (apila torre 13 17 torre 22 sol 17 5 6 22 5 4 7 12 6 1 21)
==> Activation 0  comprova: f-1
==> Activation 0  comprova: f-1
Hi ha almenys dues caixes amb numero 5 al sol i cap torre conté caixes
↪  amb aquest número
Hi ha almenys dues caixes amb numero 6 al sol i cap torre conté caixes
↪  amb aquest número
```