Politechnika
Warszawska

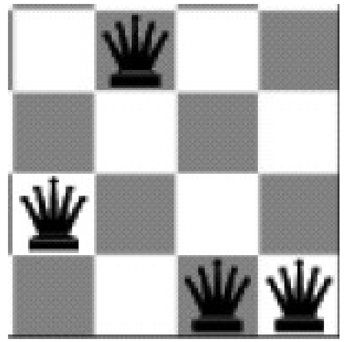
MSI

C4. Poszukiwanie celu. CSP. Gry. Planowanie

Włodzimierz Kasprzak

Zadanie 1. Poszukiwanie celu

W **problemie *4* hetmanów** (królowych) posłużymy się **funkcją oceny kosztu stanu**, która podaje liczbę atakujących się wzajemnie par hetmanów (bezpośrednio lub pośrednio). Załóżmy, że jedyne możliwe akcje to dowolne **przesuwanie pionków w kolumnach**. Przyjmując podany obok **stan początkowy** i wspomnianą funkcję oceny kosztu stanu prześledzić rozwijane **drzewo decyzyjne** przy zastosowaniu **algorytmu poszukiwania celu minimalizującego koszt stanu.**



Stan początkowy *Start*

Rozwiązanie 1

Inicjalizacja: OPEN(0) = {*Start*}

A

Węzeł „*Start*”

f(*Start*) = h(*Start*) = 1

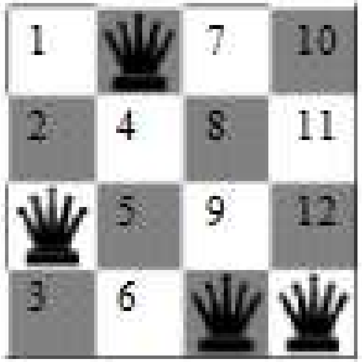
Rozwiązanie 1. (c.d.)

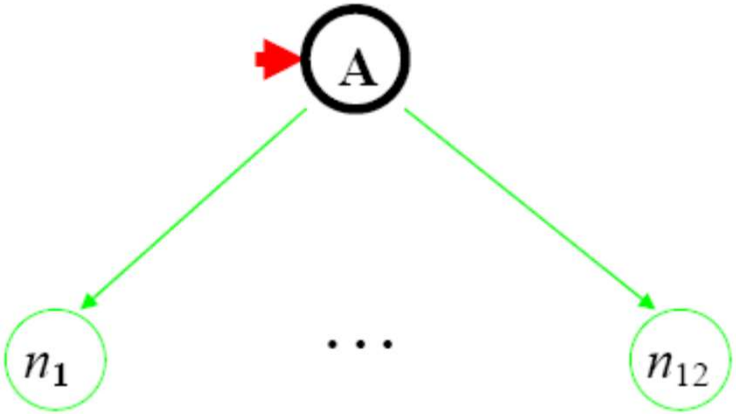
Iteracja 1:

- wybierz węzeł *Start* ze skraju OPEN
- WarunekStopu(*Start*) $\rightarrow$ False
- generuj następniki *Start*: $n_1, n_2, \ldots, n_{12}$, o heurystyce:

i	1	2	3	4	5	6	7	8	9	10	11	12
$h(n_i)$	2	2	2	2	2	3	2	1	2	1	0	3

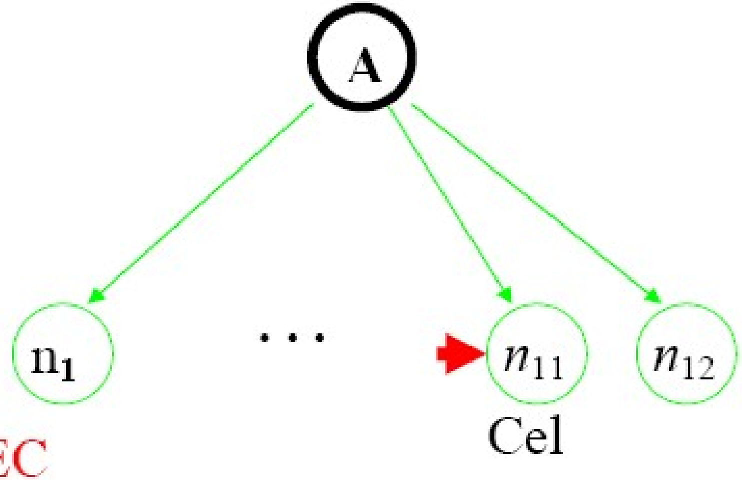
Indeksy następników stanu *Start*





Iteracja 2:

- wybierz węzeł n_{11} ze skraju OPEN
- WarunekStopu(n_{11}) $\rightarrow$ True
- Zwracany wynik: n_{11} (ścieżka nie jest istotna)



KONIEC

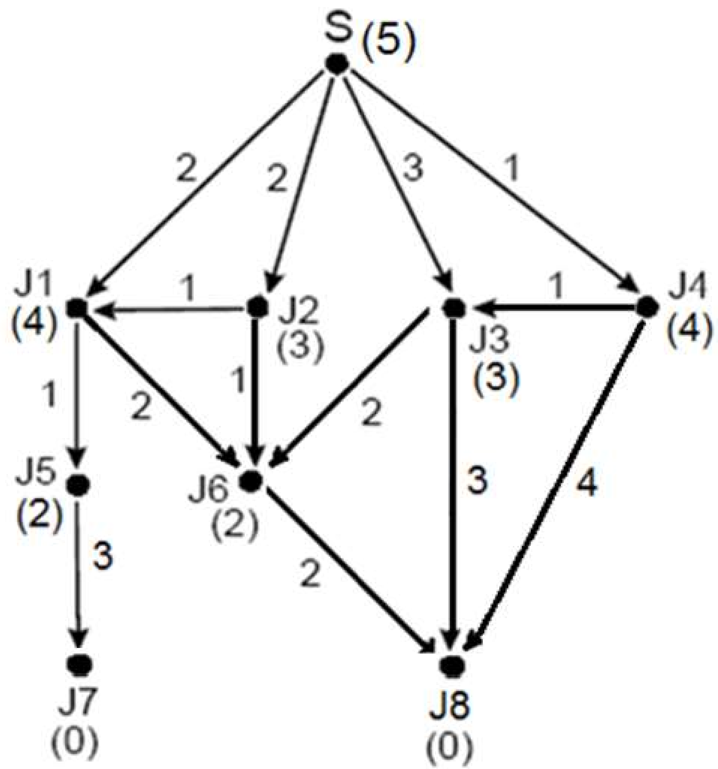
Zadanie 2. Poszukiwanie celu z funkcją heurystyczną kosztu stanu

Rozwiązać problem znalezienia celu stosując do oceny stanu funkcję heurystyczną kosztów resztkowych i strategię poszukiwanie lokalnego „przez wspinanie” (w wersji minimalizującej wartość kosztu).

W nawiasach przy węzłach podano heurystykę (oszacowanie kosztów resztkowych) dla funkcji kosztu danego węzła, a przy łukach – koszty akcji przejścia pomiędzy węzłami.

Uwaga: 1) Zgodnie z zadaniem strategia poszukiwania celu pomija rzeczywisty koszt akcji i uwzględnia jedynie heurystykę kosztów resztkowych.

2) W strategii „przez wspinanie” warunek stopu zastąpiony jest przez „test kontynuacji”.



Rozwiązanie 2 (1)

Krok	n ← wybrany węzeł	Test kontynuacji – czy poprawa ?	Następniki	Ścieżka	OPEN	Uwagi
Init				{}	{ S }	
1	S, f(S) = 5	TAK – kontynuuj	J1, J2, J3, J4	{S}	[J2, J3, J1, J4] [3, 3, 4, 4]	Może też być [J3, J2,…]
2	J2, f(J2) = 3	3< 5? TAK - kontynuuj	J1', J6	{S, J2}	[J6, J1'] [2, 4]	
3	J6, f(J6)=2	2 < 3? TAK - kontynuuj	J8	{S, J2, J6}	[J8] [0]	
4	J8, f(J8)=0	0<2 TAK – kontynuuj	Brak następników	→STOP		

Rozwiązanie 2 (2)

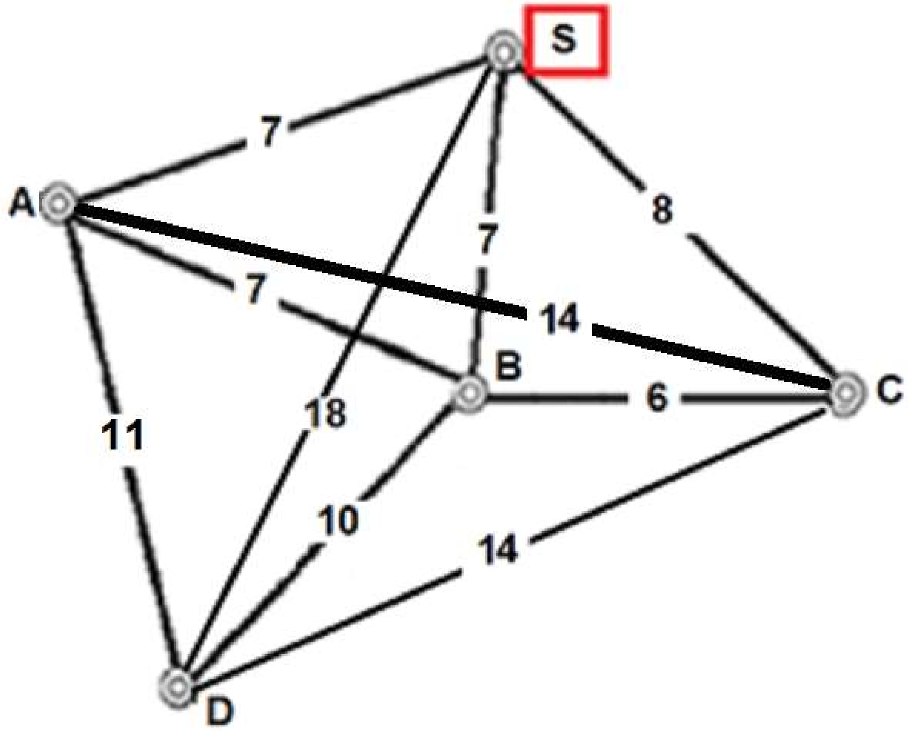
- Ścieżka: S→ J2 → J6 → J8
- Cel znaleziony (h(J8) = 0)
- Liczba wybieranych węzłów: 4

Zbiór OPEN w strategii lokalnego przeszukiwania ma lokalny charakter – zachowuje jedynie sąsiedztwo aktualnie wybranego węzła – nie występuje nawrót ani przeskok do innej ścieżki.

Zbiór CLOSED nie jest potrzebny, gdyż ścieżka rozwiązania nie jest istotna a chodzi jedynie o znalezienie celu. Nie ma też sprawdzania powtarzania się stanów.

Zadanie 3. Poszukiwanie lokalne w problemie komiwojażera ze stanem zupełnym

Dany jest **problem komiwojażera** dla 5 miast: S, A, B, C, D; schematycznie przedstawiony na rysunku poniżej. Miasto S jest jednocześnie miastem **startowym** i **końcowym**. Rozwiązaniem jest ścieżka wiodąca od S do S przez wszystkie pozostałe miasta wizytowane jednokrotnie (S→.....→S). Koszty przejazdów miedzy miastami podane są jako etykiety łuków.



3. Poszukiwanie lokalne (c.d.)

Rozwiązać podany problem stosując **poszukiwanie lokalne „przez wspinanie” (algorytm „wspinaczkowy”)** w przestrzeni stanów **zupełnych** (ścieżek całkowitych), przy następujących dalszych założeniach problemu:

- stan początkowy odpowiada ścieżce **całkowitej (S→A→B→C→D→S) ;**
- dopuszczalne akcje polegają na wymianie kolejności dwóch miast (z wyłączeniem obu brzegowych S) **sąsiadujących** ze sobą w aktualnym stanie.

Ponieważ z każdym stanem związany jest koszt odpowiadającej jej ścieżki, przeszukiwanie lokalne realizuje zamiast „wspinania” się dualnie **minimalizację** kosztu stanu. Należy:

1. przedstawić kolejne kroki decyzyjne i powstające **drzewo przeszukiwania przestrzeni stanów** zarządzane przez algorytm „wspinaczkowy”.
2. podać końcowy wynik przeszukiwania.

Rozwiązanie 3

Drzewo decyzyjne:

Poziom 0: Korzeń n0 = stan(S→A→B→C→D→S) , koszt f(n0) = 52.

1:Następniki n0: n1=(S→B→A→C→D→S) , koszt f(n1) = 60

n2= stan(S→A→C→B→D→S) , koszt f(n2) = 55

n3= stan(S→A→B→D→C→S) , koszt f(n3) = 46

Wybierany jest węzeł n3.

2:Następniki n3: n4=(S→B→A→D→C→S) , koszt f(n4) = 47

n5= stan(S→A→D→B→C→S) , koszt f(n5) = 42

n6= stan(S→A→B→C→D→S) , koszt f(n6) = 52

3:Następniki n5: n7=(S→D→A→B→C→S) , koszt f(n7) = 50

n8= stan(S→A→B→D→C→S) , koszt f(n8) = 46

n9= stan(S→A→D→C→B→S) , koszt f(n9) = 45

Brak poprawy wyniku → STOP

Znaleziony cel: stan_końcowy(S→A→D→B→C→S) ,
koszt f(stan_końcowy) = 42

Zad. 4. CSP i poszukiwanie przyrostowe

Dane jest zadanie kryptograficzne: podstaw cyfry dziesiętne pod zmienne, tak, aby operacja dodawania była poprawna

$$\begin{array}{r} \mathbf{TWO} \\ +\,\mathbf{TWO} \\ \hline \mathbf{FOUR} \end{array}$$

Każda zmienna reprezentuje inną cyfrę.

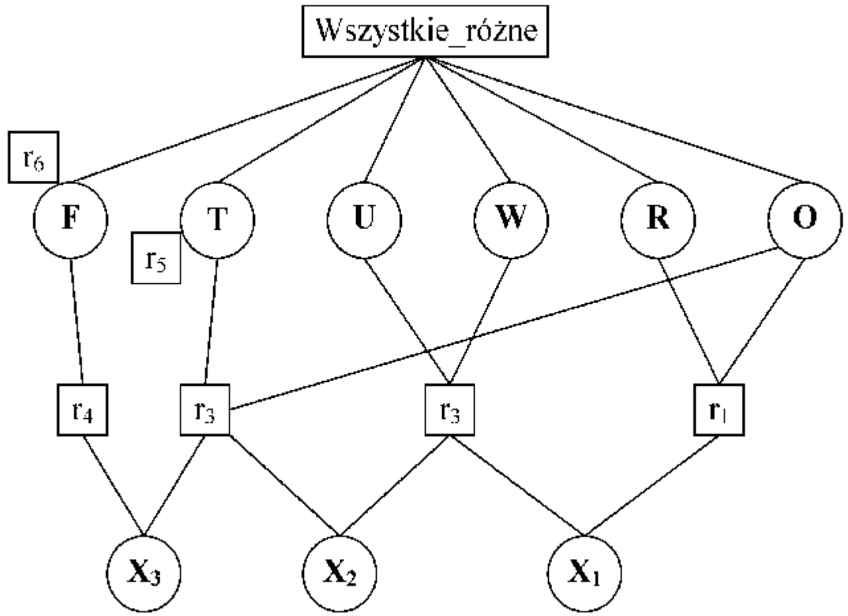
Należy zdefiniować graf ograniczeń dla tego problemu.

Przeprowadzić symulację (2 pierwsze iteracje) działania podstawowego algorytmu dla CSP – przeszukiwania przyrostowego z nawrotami.

Wskazówka:

Odwołać się do rysunku →

ale zastąpić relację „Wszystkie_różne" przez zbiór relacji 2-argumentowych.



Zad. 5. Usprawnienia algorytmu CSP

Problem: podstawić cyfry pod zmienne, tak aby powstał poprawny zapis operacji arytmetycznej, przy założeniu, że każdej zmiennej przypisana jest inna cyfra dziesiętna:

$$\begin{array}{r} \mathbf{ABC} \\ +\ \mathbf{ABC} \\ \hline \mathbf{DCEF} \end{array}$$

Należy zilustrować strategię przeszukiwania właściwą dla problemu z ograniczeniami, w szczególności:

A) podać graf ograniczeń dla problemu,

B) Przeprowadzić symulację (tylko 2 pierwsze iteracje) działania podstawowego algorytmu przeszukiwania z wykorzystaniem usprawnień: (a) najbardziej ograniczona zmienna, (b) najbardziej ograniczająca zmienna; i (c) najmniej ograniczająca wartość.

C) Podać rozwijane drzewo przeszukiwania (decyzyjne) w tej symulacji.

6. Poszukiwanie ze stanem zupełnym w problemie CSP

Dany jest problem kryptograficzny:

$$\begin{array}{r} \mathbf{ABC} \\ +\,\mathbf{ADC} \\ \hline \mathbf{CEF} \end{array}$$

Możliwe rozwiązanie to:

$$\begin{array}{r} 173 \\ +153 \\ \hline 326 \end{array}$$

Wyznaczyć graf ograniczeń dla problemu.

Zaproponować heurystyczną ocenę dla stanów zupełnych (rozwiązań).

Zastosować przeszukiwanie lokalne „[przez wspinanie” korzystające z tej funkcji oceny.

Stan początkowy dany jest jako: $\{A=0, B=1, C=2, D=3, E=4, F=5\}$.

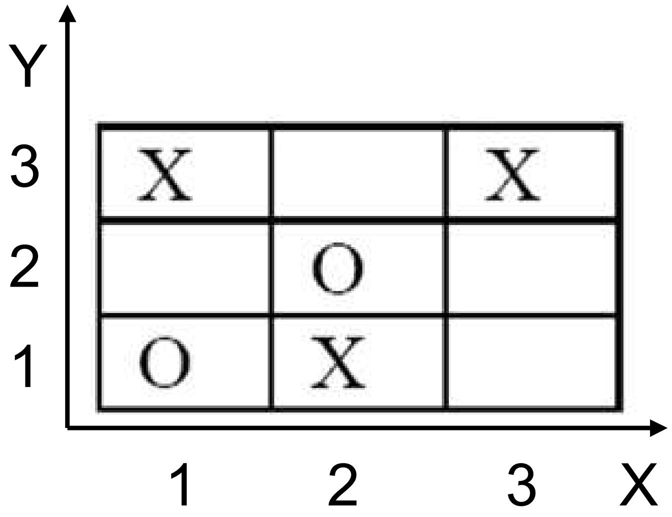
Zad. 7. „Przeszukiwanie Mini-max” i „cięcia **α-β**”

Zilustrować zasady działania strategii:

A) „**przeszukiwanie Mini-max”** i

B) **„cięcia α-β”** ,

na przykładzie „gry w kółko i krzyżyk”, w poniższej sytuacji:

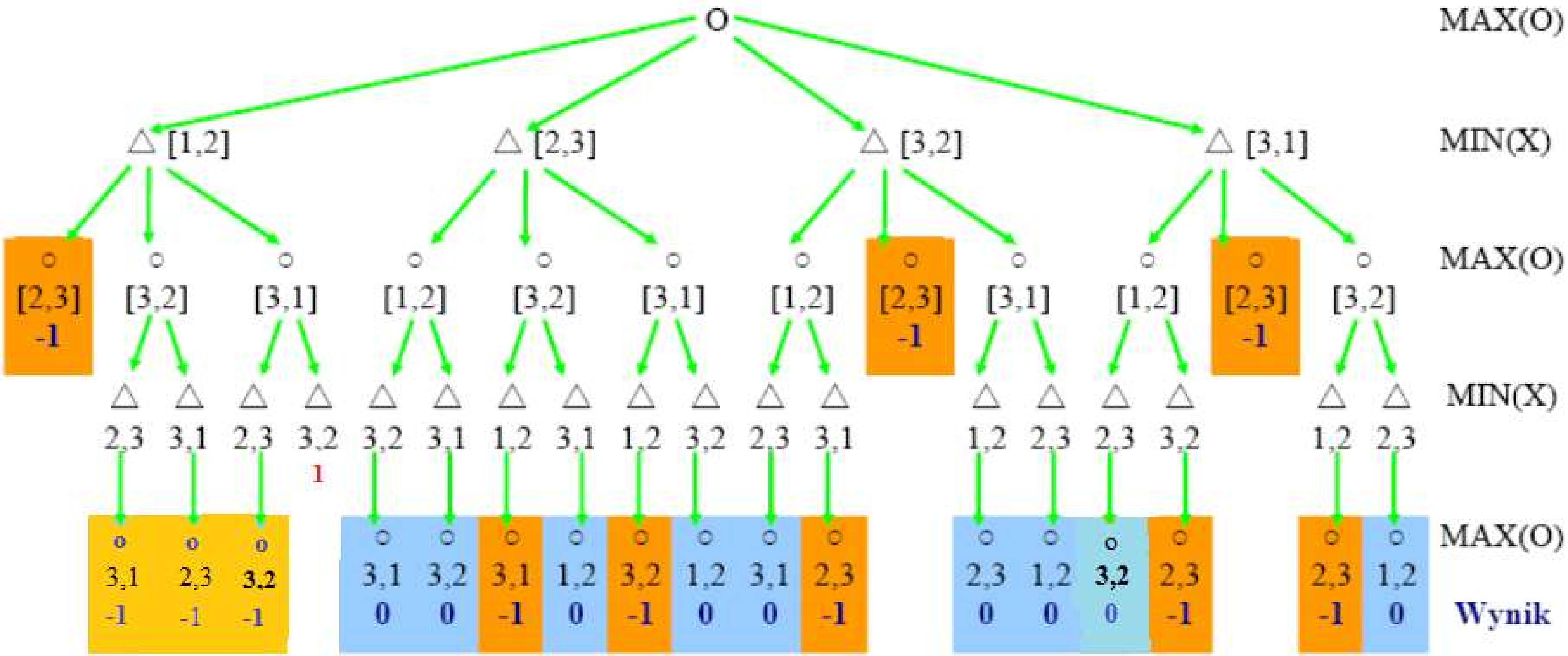


Podać drzewa decyzyjne rozwijane w obu strategiach.

Rozwiązanie 7. „Mini-max”

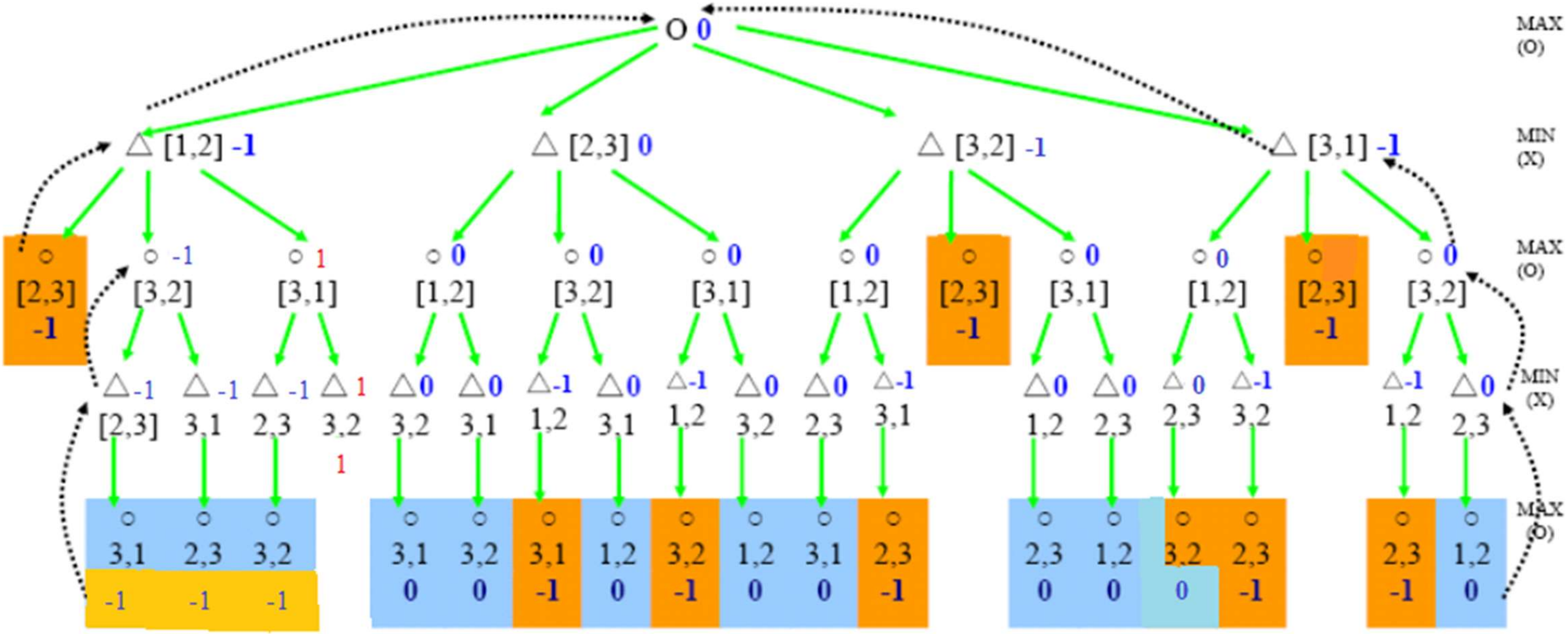
A) Drzewo decyzyjne „Mini-max” dla zadanego problemu.

W pierwszej części systematycznie rozwijamy drzewo posiadające na przemian węzły typu MAX i MIN od korzenia do liści (reprezentują stany końcowe gry).



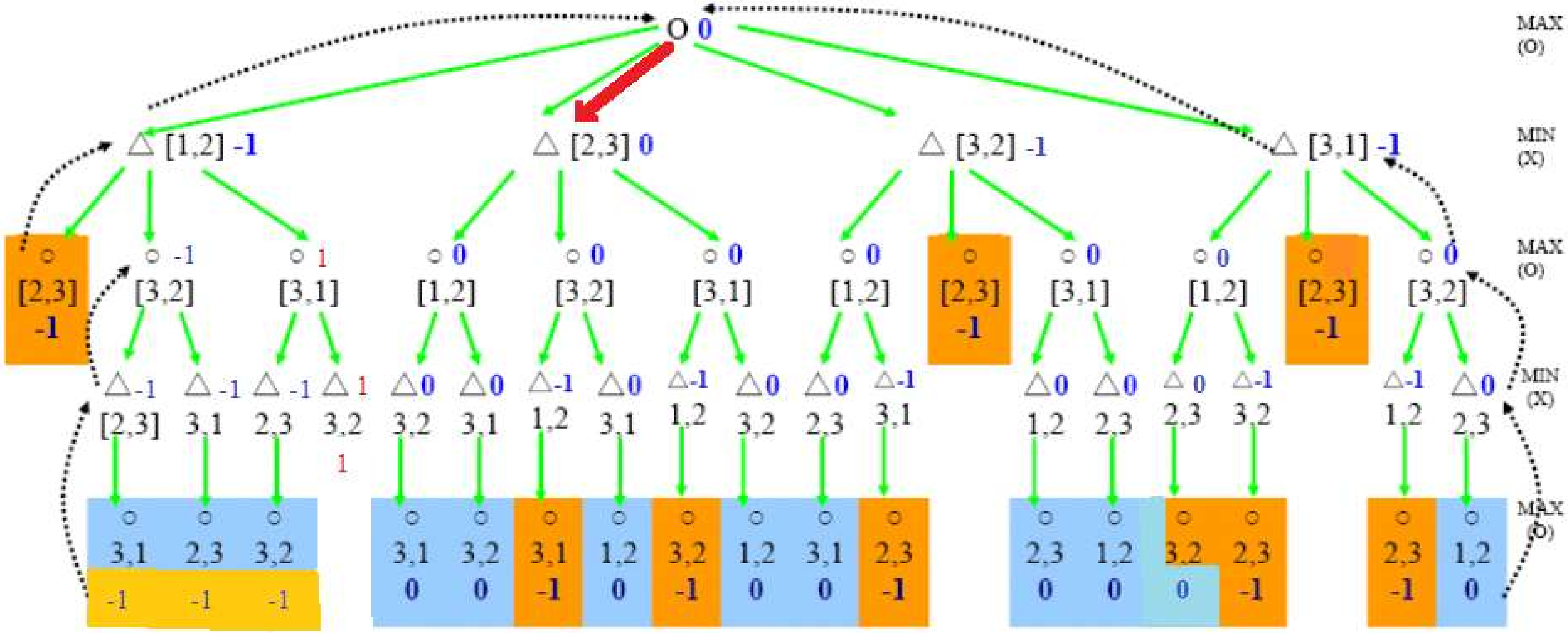
Rozwiązanie 7. (c.d.) „Mini-max”

A) c.d.) W drugim kroku systematycznie propagujemy możliwe wyniki gry od liści poprzez węzły nie-końcowe do korzenia, stosując funkcje MIN lub MAX. Najlepszy możliwy do osiągnięcia z zadanej pozycji wynik dla „naszego” gracza to „remis” (wynik 0).



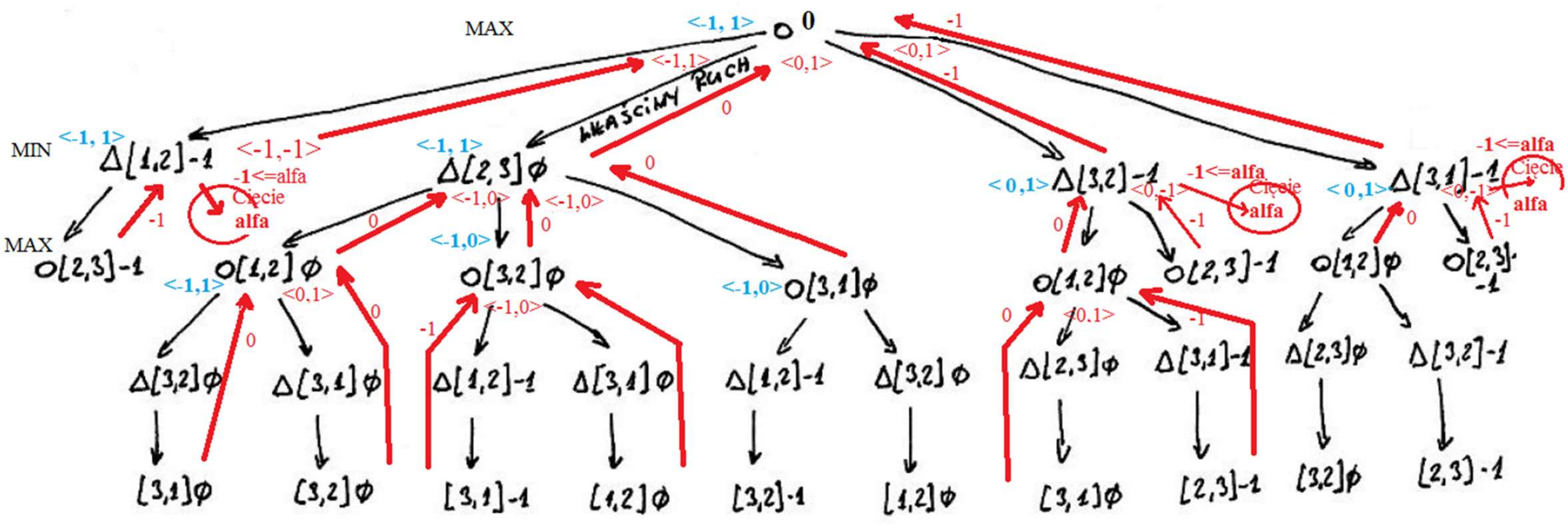
Rozwiązanie 7. (c.d.) „Mini-max”

A) c.d.) „Nasz” gracz wybiera optymalny dla siebie ruch. W tym przypadku ma do dyspozycji jeden „dobry” ruch – postawić kółko w pozycji [2,3].



Rozwiązanie 7. (c.d.) Cięcia alfa-beta

B) „Cięcia **α-β**”



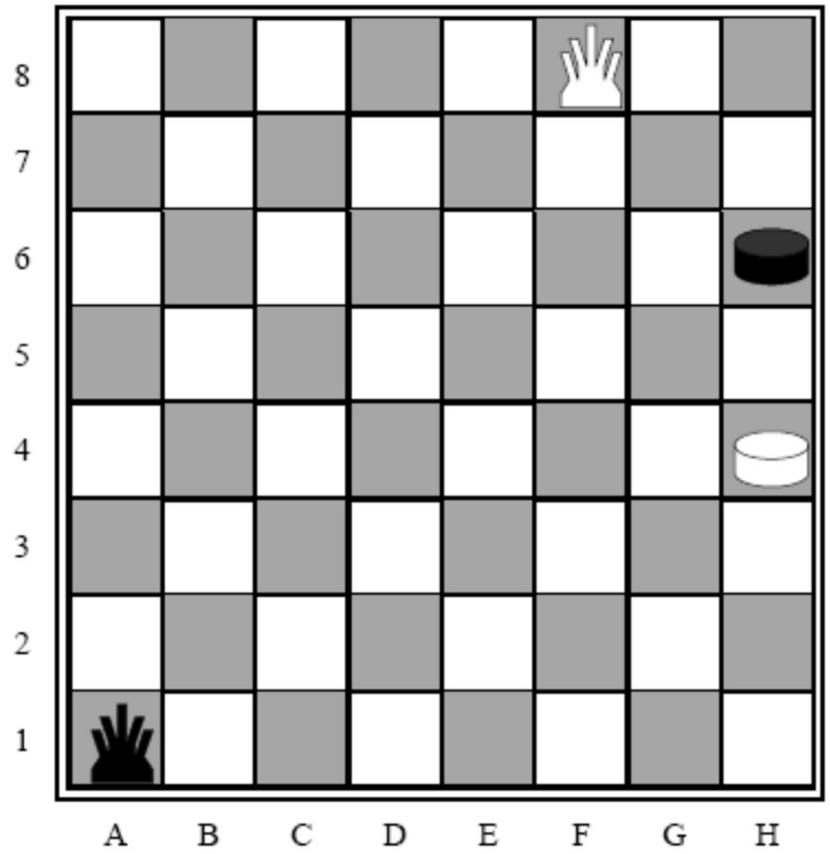
Zadanie 8. „Obcięty Mini-max”

Wyjaśnić cel stosowania i zasadę działania strategii **przeszukiwania „obciętego Mini-max-u”.**

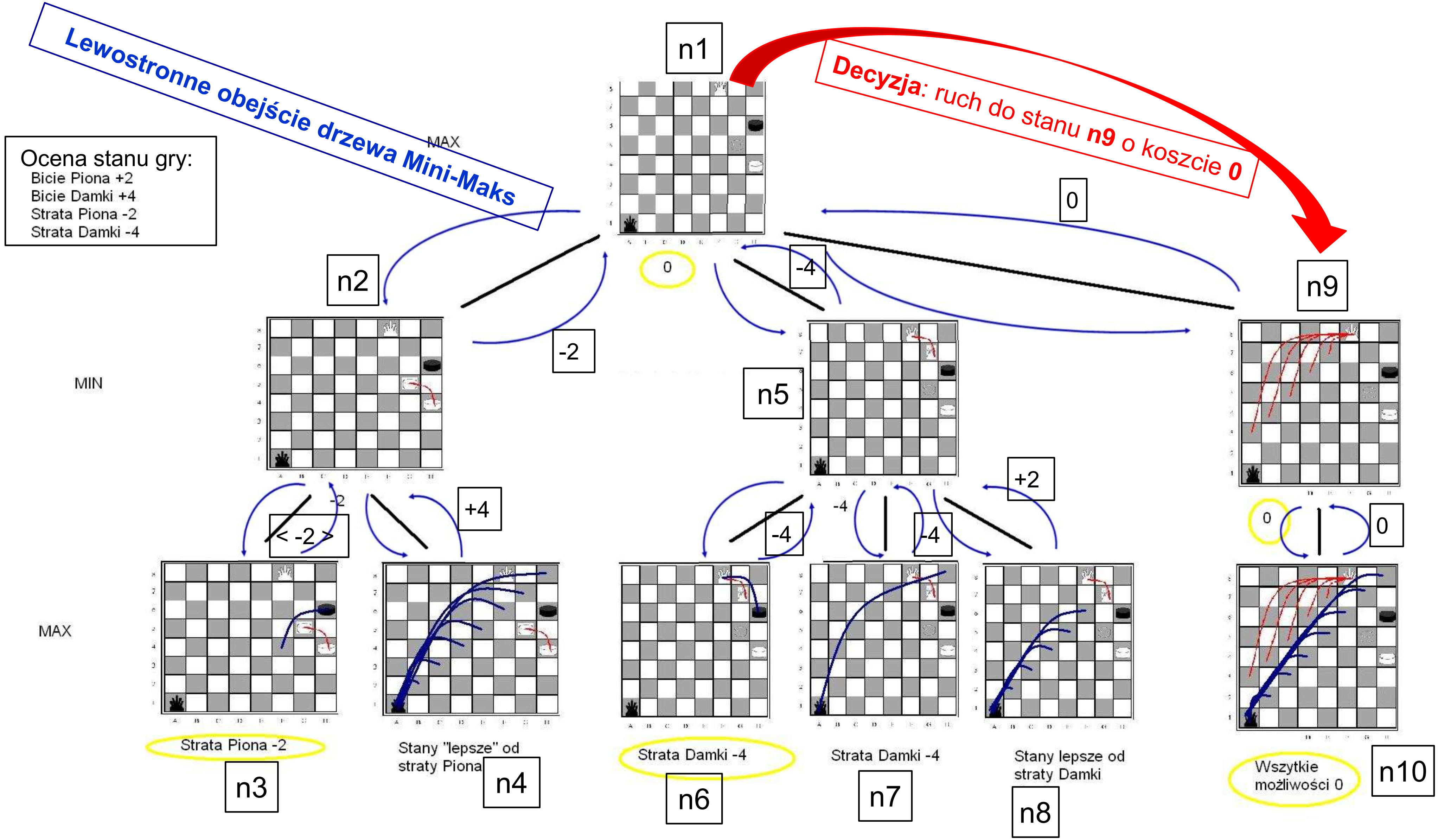
Zilustrować odpowiedź na przykładzie „gry w warcaby”, w poniższej sytuacji:

- każdej ze stron została 1 damka i 1 pionek,
- białe podążają „w górę”,
- ruch należy do białych,
- **poziom obcinania** wynosi 2.

Zaproponować ocenę stanu gry.



Rozwiązanie 8

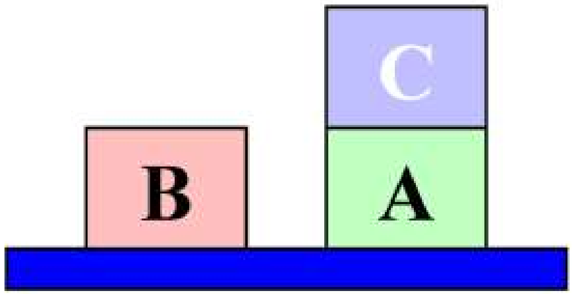


Zadanie 9. Planowanie - problem bloczków

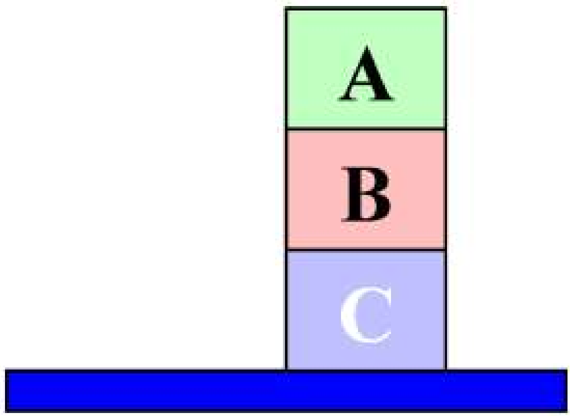
Należy uporządkować trzy bloczki (o nazwach A, B i C) stojące na stole w taki sposób, aby A znalazło się na górze B, a ten z kolei – na górze C. Jednak jednocześnie można przemieszczać jedynie jeden blok. Stan początkowy polega na tym, że B znajduje się na stole, zaś C na górze bloku A, a A – na podstawie.

Zaprojektować plan uporządkowania tych 3 bloków:

- zdefiniować właściwe operatory w STRIPS ,
- prześledzić krok-po-kroku generację planu PCzU.



Stan początkowy



Stan docelowy

Rozwiązanie 9

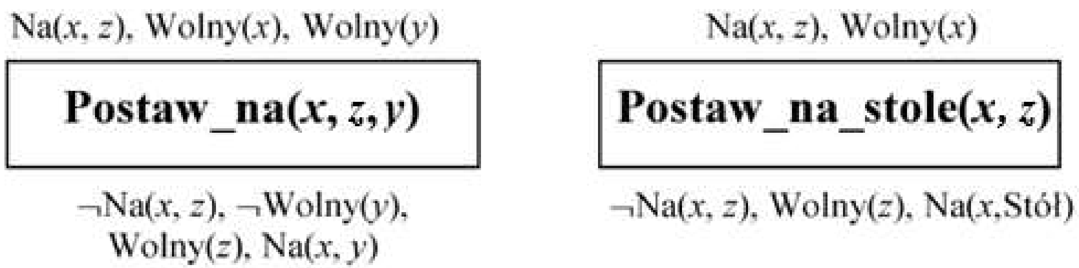
Predykaty dla tego problemu:

1. Wolny(x) – „wolny x”
2. Na(x, y) – klocek „x jest na klocku y” lub „x jest na stole”

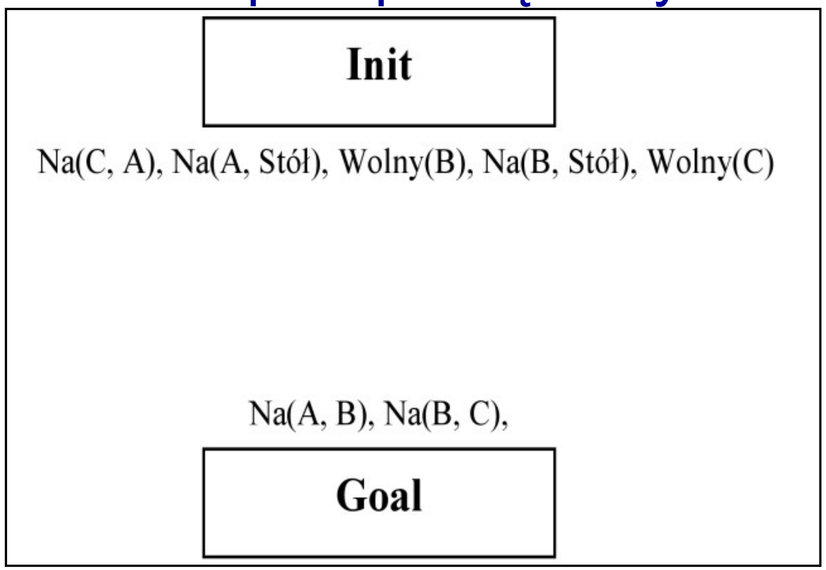
Operatory STRIPS dla tego problemu:

1. Init – sytuacja początkowa; 2. Goal – sytuacja docelowa
3. Postaw_na(x, z, y) – „pobierz klocek z „z” i postaw go na y”
4. Postaw_na_stole(x, z) – „zdejmij klocek z „z” i postaw go na stole”.

Definicje (w postaci graficznej):

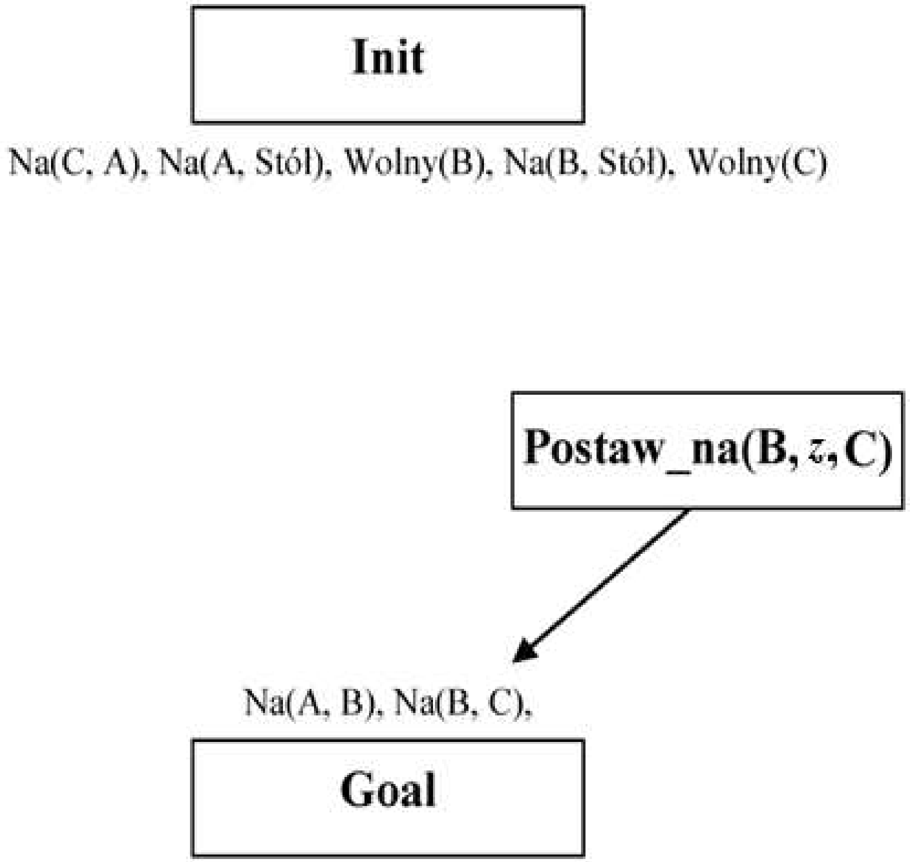


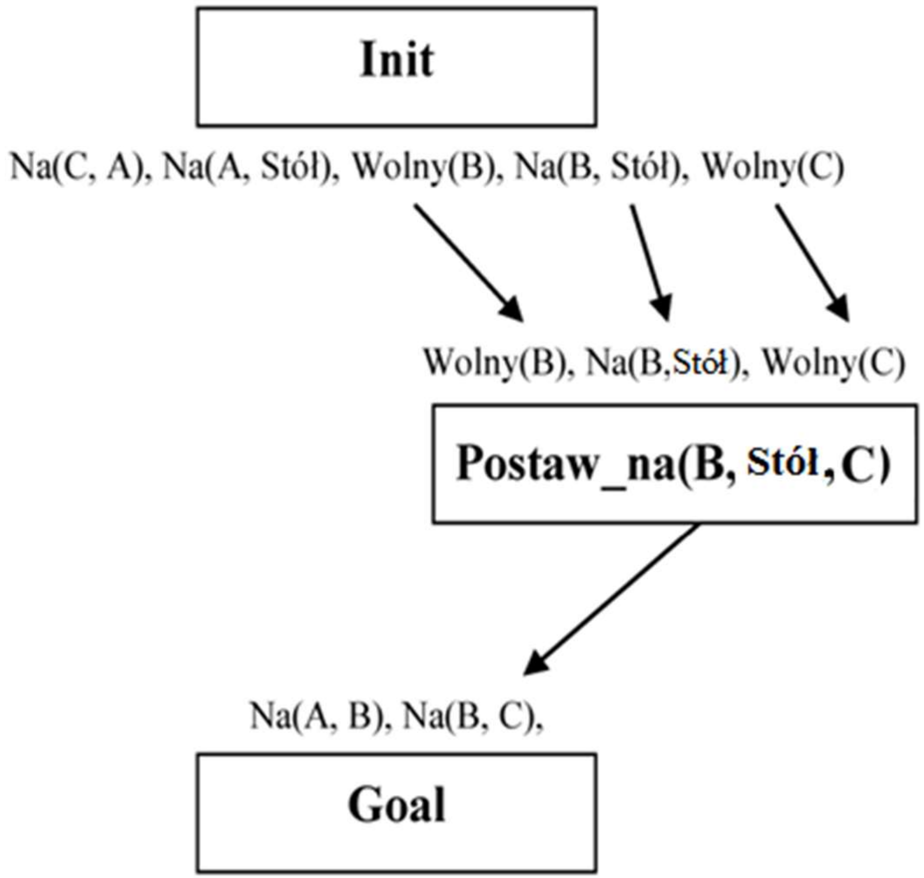
- definicje Init i Goal – jednocześnie plan początkowy



Rozwiązanie 9 (c.d.)

1. Dodaj krok: „Postaw_na(B, z, C) i jego związek przyczynowy z Goal.
2. Krok Init spełnia wszystkie warunki nowego kroku. Dodaj związek przyczynowy między nimi → krok „Postaw_na(B, Stół, C)



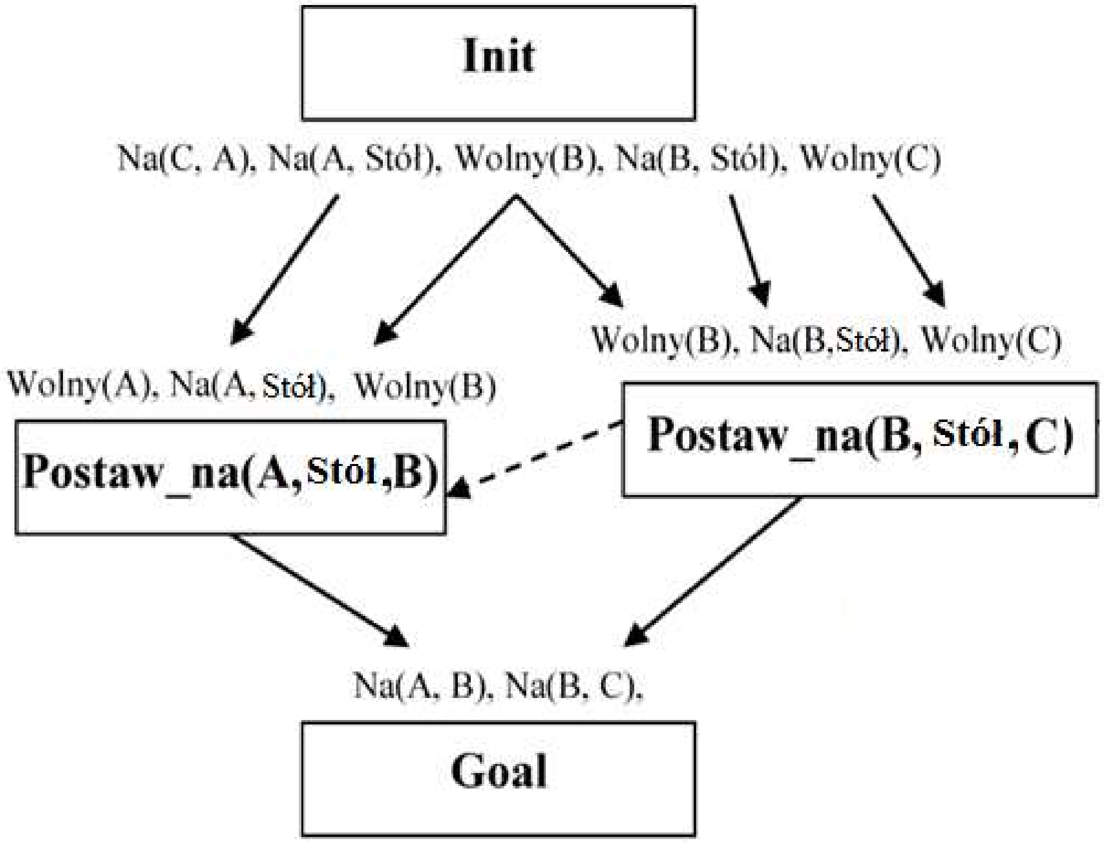


Rozwiązanie 9 (c.d.)

3. Dodaj krok Postaw_na(A, Stół, B) i związek przyczynowy z Goal .
4. Jednak Postaw_na(A, Stół, B) neguje warunek Wolny(B) i przez to zagraża krokowi Postaw_na(B, Stół, C).

Rozwiązaniem jest *degradacja* nowego kroku.

5. Dodaj związek przyczynowy pomiędzy Init a nowym krokiem.

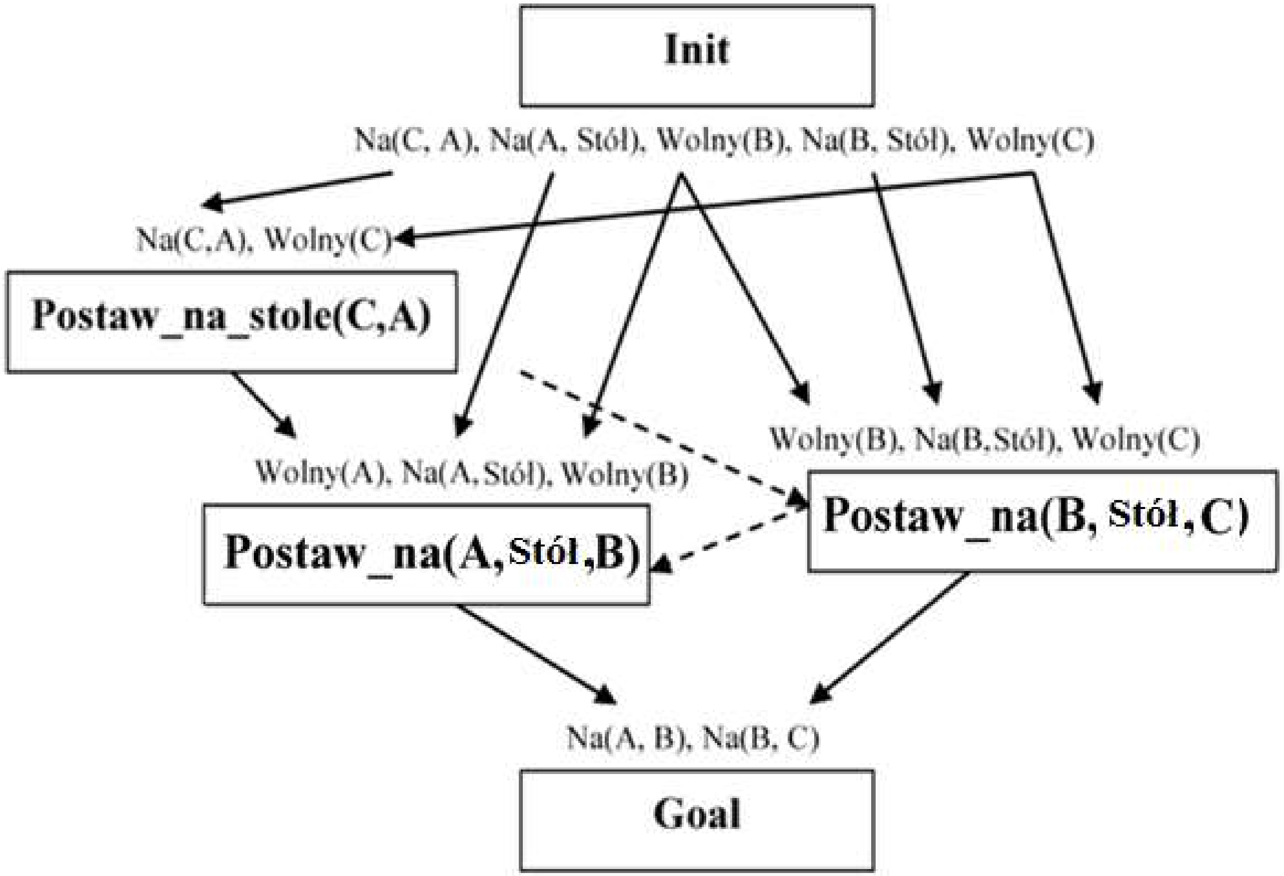


Rozwiązanie 9 (c.d.)

6. Dodaj krok Postaw_na_stole(C, A) dla spełnienia warunku kroku Postaw_na(A, Stół, B).

7. Jednak Postaw_na(B, Stół, C) neguje warunek Wolny(C) nowego operatora i zagraża mu. Rozwiązaniem jest promocja Postaw_na_stole(C,A) przed Postaw_na(B, Stół, C).

8. Dodaj związek przyczynowy pomiędzy Init a nowym krokiem.



Zadanie 10. Plan zmiany opony

Dany jest problem zmiany zepsutej opony w samochodzie:

- stan początkowy to: zepsuta opona na osi a dobra opona zapasowa w bagażniku,
- a stan docelowy to: dobra opona zapasowa założona na oś samochodu

Załóżmy też, że samochód pozostawiono w wyjątkowo niebezpiecznej okolicy, skutkiem czego w nocy ukradzione zostaną opony.

Zdefiniować powyższy problem w języku ADL .

Rozwiązanie

Obok operatorów Init i Goal zdefiniujemy 4 akcje:

1. wyjąć zapasową oponę z bagażnika (Zabierz(x:Opona, Bagażnik)),
2. zdemontować zepsutą oponę z osi (Zabierz(x:Opona, Oś)),
3. założyć zapasową oponę na oś (Załóż(x:Opona, Oś)),
4. pozostawić samochód na noc bez nadzoru (ZostawNaNoc).

W porównaniu ze STRIPS możliwy jest teraz zanegowany warunek operatora, np. $\neg$W (Kapeć, Oś).

Rozwiązanie 10

Przykładowy opis podanego problemu w języku ADL:

Init(EFFECT: W(Kapeć, Oś) $\wedge$ W(Zapas, Bagażnik)) ;

Goal(PRECOND: W(Zapas, Oś)) ;

Action(Zabierz(Zapas, Bagażnik),
    PRECOND: W(Zapas, Bagażnik)
    EFFECT: $\neg$W(Zapas, Bagażnik) $\wedge$ W(Zapas, Na_ziemi))

Action(Zabierz(Kapeć, Oś),
    PRECOND: W(Kapeć, Oś)
    EFFECT: $\neg$W(Kapeć, Oś) $\wedge$ W(Kapeć, Na_ziemi))

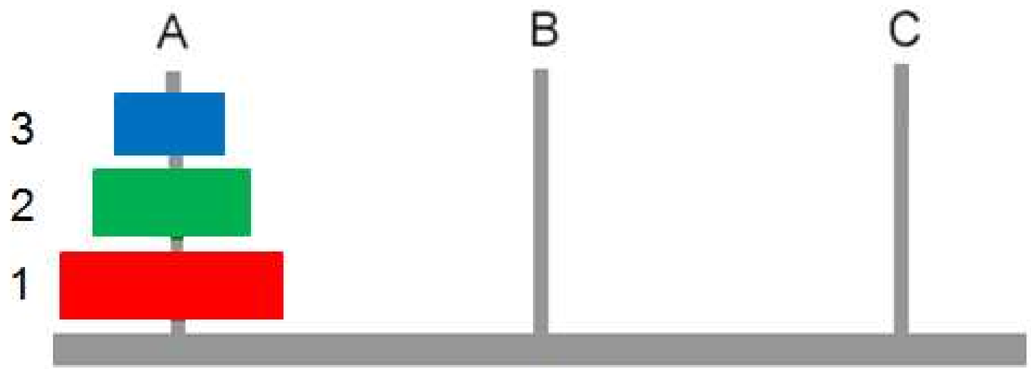
Action(Załóż(Zapas, Oś),
    PRECOND: W(Zapas,Na_ziemi) $\wedge$ $\neg$ W(Kapeć,Oś)
    EFFECT: $\neg$W(Zapas, Na_ziemi) $\wedge$ W(Kapeć,Oś))

Action(ZostawNaNoc,
    PRECOND:
    EFFECT: $\neg$W(Zapas, Na_ziemi) $\wedge$ W(Zapas,Oś) $\wedge$ $\neg$ W(Zapas, Bagażnik))

Zadanie 11. Wieże Hanoi

Problem trzech wież Hanoi:

- dane są trzy patyki (o nazwach A, B i C) na które wkładać można N krążków o różnej średnicy (oznaczmy je od największego do najmniejszego jako 1, 2 … N);
- należy umieścić N krążków na patyku docelowym C w taki sposób, aby największy krążek 1 znalazł się na dole, na nim krążek 2, itd., a na samej górze – najmniejszy krążek N. Jednocześnie można przemieszczać tylko jeden krążek z patyka na patyk.



Stan początkowy

Rozwiązaniem tego problemu może być poniższa funkcja rekurencyjna:

Zadanie 11. Wieże Hanoi (c.d.)

function Hanoi([A, B, C], n, bieżący, pomocniczy, docelowy) : [A, B, C];

gdzie: [A, B, C] – stan 3 patyków;

n – liczba górnych krążków wymagających przesunięcia z patyka "bieżący" do "docelowy".

(bieżący, pomocniczy, docelowy) – aktualne znaczenie patyków A, B, C; np. (A, C, B) oznacza, iż patyk A zawiera krążki do przeniesienia, C jest patykiem pomocniczym, a B – patykiem docelowym dla krążków.

Funkcja zwraca nowy stan patyków: A, B, C.

Wyznaczyć **plan częściowo uporządkowany** (PCzU) dla rozwiązania tego problemu:

- Zdefiniować właściwe **operatory w STRIPS**;
- Prześledzić **proces przeszukiwania** przestrzeni planów dla ***N*=3** (podać plan początkowy, trzy plany częściowe - pośrednie i plan końcowy);
- Jak **zmienią się:** przebieg planowania i plan końcowy dla ***N* > 3** ?

Wskazówka: plan odpowiada wykonaniu funkcji Hanoi().

Rozwiązanie 11. Wieże Hanoi

Poznajemy algorytm funkcji Hanoi:

function Hanoi([*A,B,C*]: ABClist, *n*:**int**, *b*:**int**, *p*:**int**, *d*:**int**): ABClist
begin
if *n*=2 **then**
begin
... // przenieś górny krążek z *b* do *p*;
... // przenieś górny krążek z *b* do *d*;
... // przenieś górny krążek z *p* do *d*;
end;
else if *n*>2 **then**
begin *// wykonanie dla N krążków wymaga 2 wywołań dla N*-1 krążków*.*
[*A, B, C*] := **Hanoi**([*A, B, C*], *n*-1*, b, d, p*); // (1)
... // przenieś górny krążek z *b* do *d*;
[*A, B, C*] := **Hanoi**([*A, B, C*]*, n*-1*, p, b, d*); // (2)
end;
end;

Rozwiązanie 11.

Definiujemy operatory odpowiadające etapom wykonania tego algorytmu:

function Hanoi([*A,B,C*]: ABClist, *n*:**int**, *cur*:**int**, *temp*:**int**, *des*:**int**): ABClist

OP: Init dla zadanych n, [A,B,C]

OP: Finish dla zadanych n, [A,B,C]

begin

Predykat

if *n*=2 **then**

Begin OP: *Ustaw_Parametry_TrzechOperacji*

MoveDisc(cur, temp); // przenieś górny krążek z *cur* na *temp*;
MoveDisc(cur, des); // przenieś górny krążek z *cur* na *des*;
MoveDisc(temp, des); przenieś górny krążek z *temp* na *des*;

OP: TrzyOperacje

end;

Predykat

else if *n*>2 **then**

begin OP: *Ustaw_Parametry_Wywołań_I_JednejOperacji*

// a call for N needs two calls for N-1 discs.

[*A, B, C*] := **Hanoi**([*A, B, C*], *n*-1, *cur, des, temp*); // (1) *OP:Call_1*

MoveDisc(cur, des) // przenieś górny krążek z *cur* na *des*; *OP:JednaOperacja*

[*A, B, C*] := **Hanoi**([*A, B, C*], *n*-1, *temp, cur, des*); // (2) *OP:Call_2*

end;

return [A,B,C];

OP: return – zwróć wynik i odtwórz dane funkcji

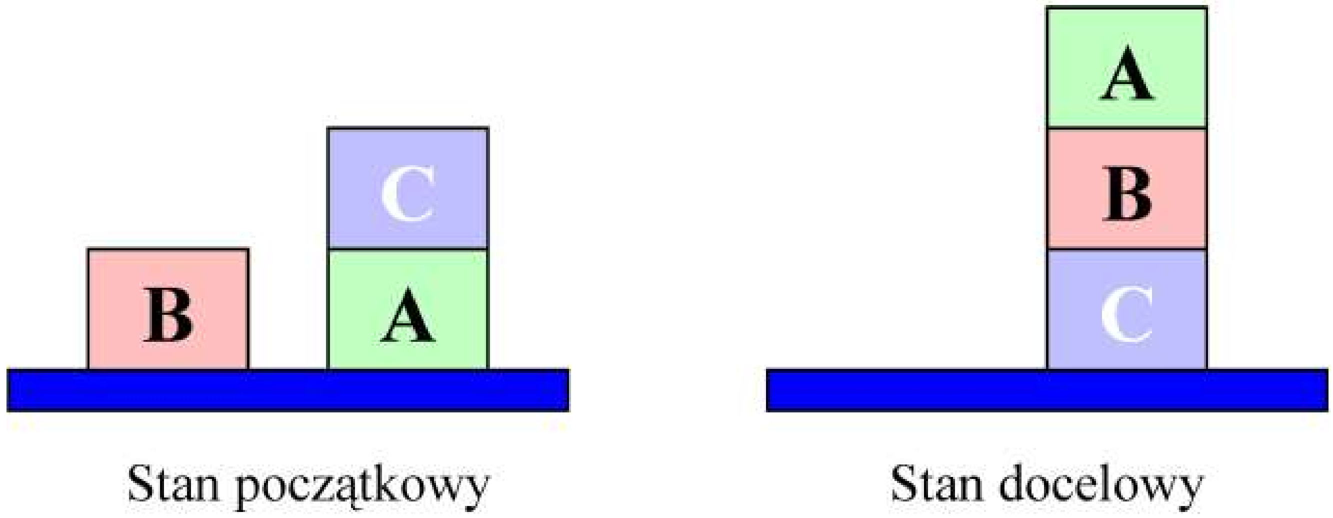
end;

Rozwiązanie 11

Plan dla N=3:

Zad. 12. Problem bloczków – „Graphplan”

Znaleźć plan rozwiązania „problemu bloczków” (opisanego w zad. 9) stosując algorytm „Graphplan”.



A. Zdefiniować potrzebne operatory w języku ADL,
B. Podać wyniki częściowe - rozwijany graf planujący, uzyskany po każdej iteracji algorytmu planowania,
C. Podać końcowy wynik.

Rozwiązanie 12 (1)

Rozwiązanie

A) Definicja operatorów w języku ADL

- Init (EFFECT: Na(C, A) ∧ Wolny(C) ∧ Wolny(B) ∧ ¬Wolny(A))
- Goal (PRECOND: Na(C, Stół) ∧ Na(B, C) ∧ Na(A, B) ∧ Wolny(A) ∧¬Na(B, Stół) ∧ ¬Na(A, Stół) ∧ ¬Wolny(C) ∧ ¬Wolny(B))
- Action(Postaw_na(k1: Klocek, k2: Klocek),

  PRECOND: Wolny(k2) ∧ Wolny(k1) ∧ ¬(k1=k2)

  EFFECT: Na(k1, k2) ∧ Wolny(k1) ∧ ¬Wolny(k2)

)
- Action(Postaw_na_stole(k1: Klocek, k2: Klocek),

  PRECOND: Wolny(k1) ∧ Na(k1, k2)

  EFFECT: Na(k1, Stół) ∧ Wolny(k1) ∧ Wolny(k2) ∧ ¬Na(k1, k2)

)

Rozwiązanie 12 (2)

Rozwiązanie (c.d.)

Uwaga:

Operatory w grafie planującym muszą mieć postać predykatową (nie mogą zawierać zmiennych). Stąd w samym grafie planującym wystąpić może:

- EFFECT operatora „Init" jako stan S0
- PRECOND operatora „Goal" jako stan warstwy końcowej S_N
- 6 instancji (konkretyzacji) operatora „Postaw_na()" o argumentach:
  - (A, B), (A, C), (B, C), (B, A), (C, A), (C, B)
- 6 instancji (konkretyzacji) operatora „Postaw_na_stole()" o argumentach:
  - (A, B), (A, C), (B, C), (B, A), (C, A), (C, B) .

W definicjach ogólnych operatora w ADL zmienne są jawnie określonego typu.

Predykat (nie-)równościowy, $\neg(k1=k2)$, służy jedynie do ograniczenia zbioru kroków (operatorów konkretnych).

Rozwiązanie 12 (3)

B) Algorytm „Graphplan”

Inicjalizacja: początkowy graf planujący składa się ze stanu początkowego (InitialState $\mathbf{S}_0$) i docelowego (GoalState $\mathbf{S}_N$).

Uwaga: przy założeniu zamkniętego świata należałoby dodać do stanu początkowego negacje własności stanu docelowego nie występujące w stanie początkowym.

$\mathbf{S}_0$		$\mathbf{S}_N$	Czy występuje w $\mathbf{S}_0$?
Na(C, A)		Na(C, stół)	Nie
Wolny(C)		Na(B, C)	Nie
Wolny(B)		Na(A, B)	Nie
~ Wolny(A)	...	Wolny(A)	Nie
		~ Wolny(B)	Nie
		~ Wolny(C)	Nie
		~ Na(A, stół)	Nie
		~ Na(B, stół)	Nie

Warunek_stopu(0):

Czy $S_0 \subseteq S_N$?

Odp. NIE

→

kontynuujemy

Rozwiązanie 12 (4)

Rozwijanie grafu planującego prowadzimy od celu do początku (do stanu początkowego).

Iteracja 1. Dodajemy wszystkie możliwe akcje w zbiorze $A_{N\text{-}1}$, których następniki znajdują się w zbiorze S_N.

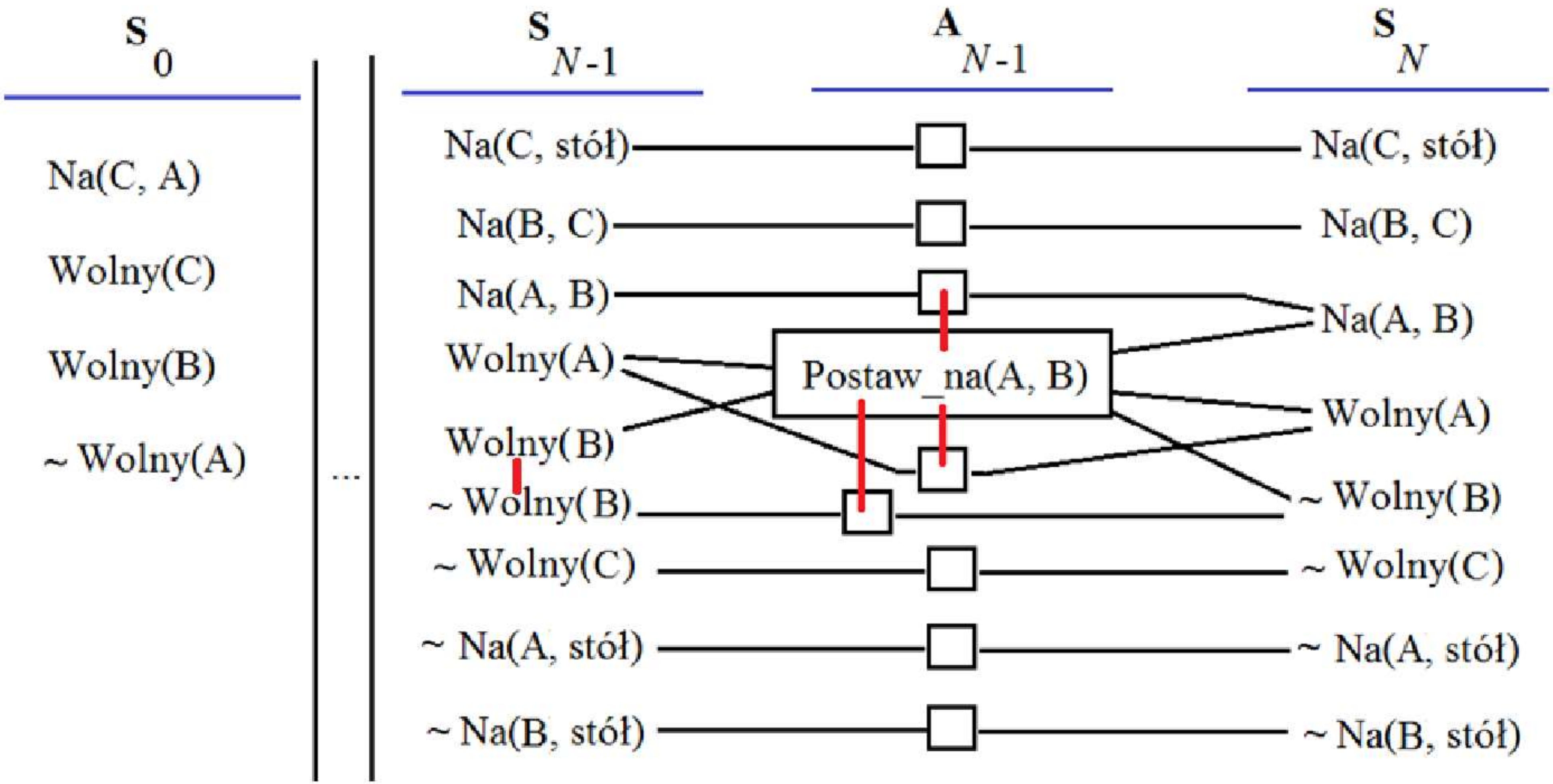
Mamy 12 możliwych kroków (*predykatowych* instancji operatorów):

- **Postaw_na(A, B) : TAK**, następniki są w S_N. - dodaj.
- Postaw_na(A, C) : NIE, gdyż brak „Na(A, C)”
- Postaw_na(B, C) : NIE, gdyż brak „Wolny(B)”, podobnie Postaw_na(B, A), Postaw_na(C, A) i Postaw_na(C, B)
- Postaw_na_stole(C, A) : NIE, gdyż $\neg$Wolny(C), podobnie dla (C, B)
- Postaw_na_stole(B, A): NIE, gdyż $\neg$ Na(B, stół), podobnie dla (B, C)
- Postaw_na_stole(A, B): NIE, gdyż $\neg$ Na(A, stół), podobnie dla (A, C)

Tym samym dla stanu docelowego otrzymujemy jeden zgodny zbiór operatorów (złożony z operatora „Postaw_na(A, B)” i 5 zgodnych z nim operacji „przezroczystych”)

Rozwiązanie 12 (5)

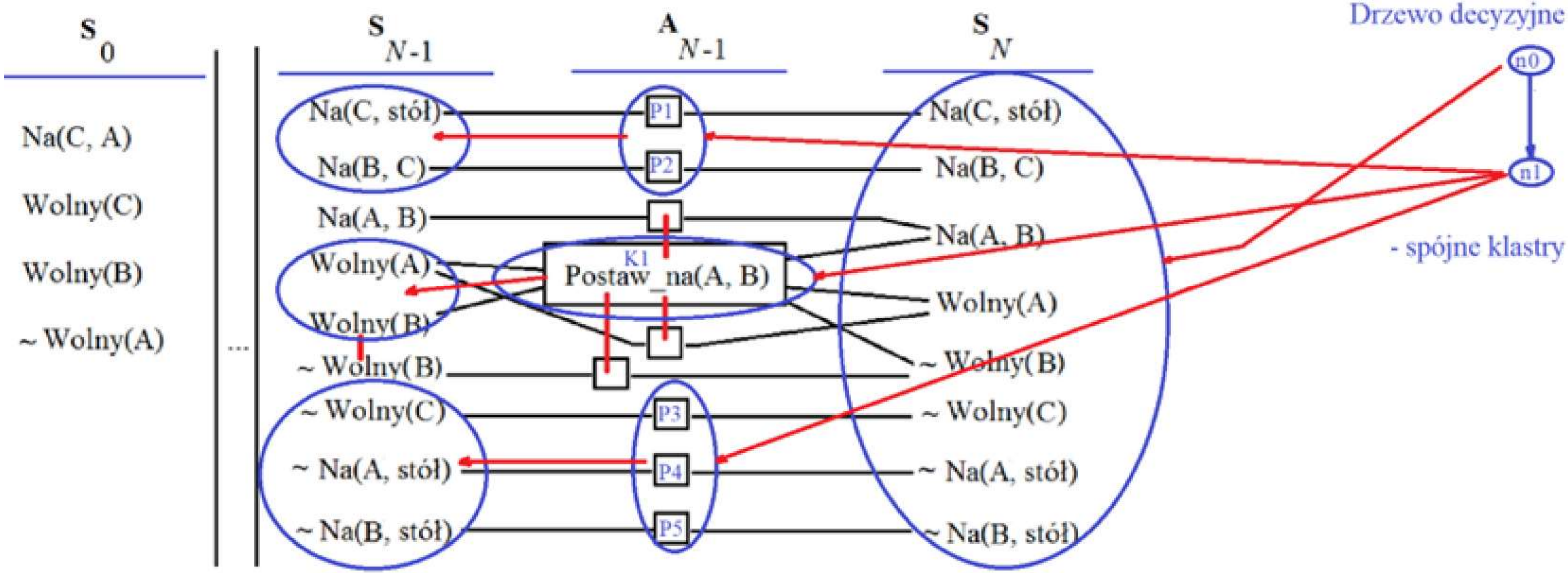
Wynik pierwszej iteracji



Rozwiązanie 12 (6)

Wynik pierwszej iteracji

Jeden spójny klaster (z rzeczywistą akcją) na poziomie $A_{N\text{-}1}$



Warunek_stopu(1): czy istnieje spójny klaster C w $S_{N\text{-}1}$ taki, że $S_0 \subseteq C$?

Mamy 1 klaster: $C(n1) = \{\text{Warunki}(K1, P1, P2, P3, P4, P5)\}$

Odp. NIE

→ Jeśli nie osiągnięto maksymalnej liczby poziomów to kontynuujemy.

Rozwiązanie 12 (6)

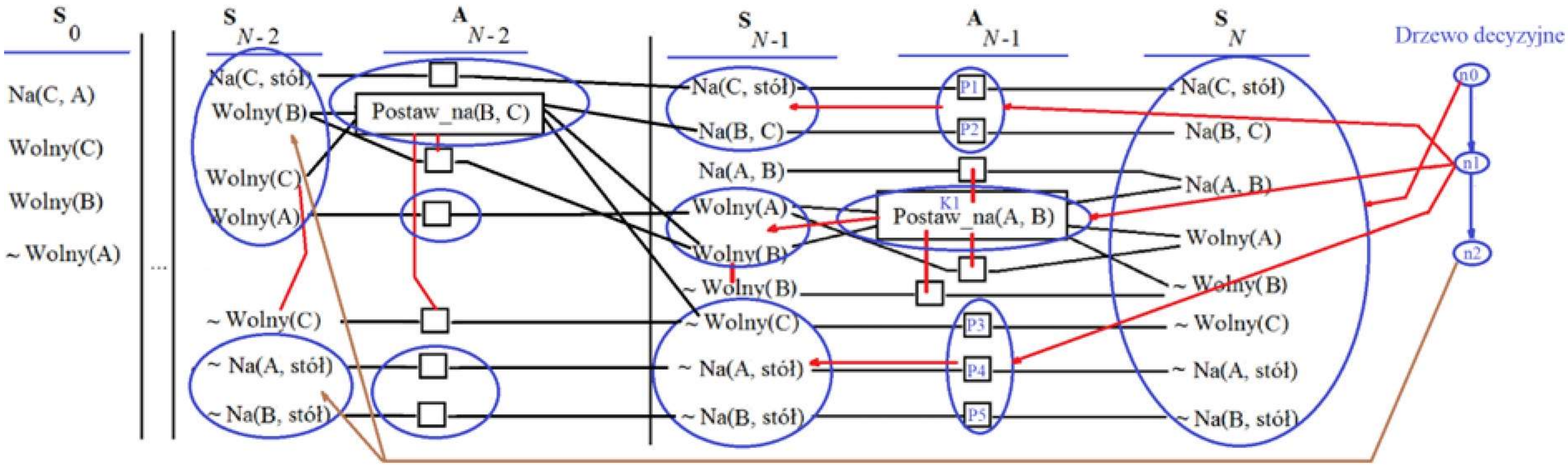
Iteracja 2: Rozwijamy operacje, których następniki istnieją w stanie S_{N-1} i są powiązane z klastrem K1 (w ogólności – dowolnym klastrem) poziomu 1 drzewa decyzji, itd. Ponownie sprawdzamy kroki:

- Postaw_na(A, B): NIE, gdyż teraz jest „Wolny(B)".
- Postaw_na(A, C): NIE, gdyż brak „Na(A, C)"
- **Postaw_na(B, C): TAK**, gdyż są w S_{N-1} jego efekty powiązane z klastrem K1: „Na(B, C)", „Wolny(B)" i „¬Wolny(C)".
- Postaw_na(B,A): NIE, gdyż nie ma „¬Wolny(A)".
- Postaw_na(C,A): NIE, gdyż nie ma „¬Wolny(A)".
- Postaw_na(C,B): NIE, gdyż nie ma „¬Wolny(B)".
- Postaw_na_stole(C, A) : NIE, gdyż jest ¬Wolny(C) – podobnie (C, B)
- Postaw_na_stole(B, A): NIE, gdyż ¬Na(B, stół), podobnie (B, C)
- Postaw_na_stole(A, B): NIE, gdyż ¬Na(A, stół), podobnie (A, C)

Ponownie jest możliwe dodanie do planu jedynie jednej akcji.

Rozwiązanie 12 (6)

Wynik po iteracji 2:



Rozwiązanie 12 (6)

Iteracja 3:

Podobnie jak w poprzednich 1 i 2 sprawdzamy wszystkie możliwe kroki, których następniki istnieją w poprzednim stanie (teraz S_{N-2}) i należą do jakiegokolwiek klastra (w ogólności - klastrów) poprzedniego poziomu (teraz 2) drzewa decyzji.

Jest możliwy jeden właściwy krok: Postaw_na_stole(C, A).

W wyniku tej iteracji warunek stopu zostanie spełniony – jeden klaster predykatów stanu S_{N-3} zawiera niesprzeczne ze sobą predykaty i klaster ten zawiera wszystkie predykaty stanu początkowego.

c) Plan

Plan końcowy to sekwencja „klastrów" rzeczywistych akcji (kroków). W tym przypadku w każdym klastrze jest tylko pojedynczy krok:

Krok 1: Postaw_na_stole(C, A),

Krok 2: Postaw_na(B, C),

Krok 3: Postaw_na(A, B).