ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1998
Seria: AUTOMATYKA z.122 Nr kol. 1388

Adam GAŁUSZKA

ROZWIĄZYWANIE ZADAŃ ZE ŚWIATA KLOCKÓW

Streszczenie. Świat Klocków jest wygodną reprezentacją dla problemów planowania. Istnieje wiele różnych algorytmów, które używają reprezentacji zadania typu STRIPS przy rozwiązywaniu zadań ze Świata Klocków. Niektóre z nich przy poszukiwaniu rozwiązania przeszukują przestrzeń stanów świata, inne przestrzeń stanów planu. W artykule prezentowana jest rozszerzona hierarchia podcelów oraz różne dziedziny operatorów. Niektóre z podcelów są trudne do rozwiązania dla wszystkich algorytmów. Główną częścią pracy jest propozycja innego podejścia do przeszukiwania przestrzeni stanów. Proponowany algorytm znajduje rozwiązanie zadania poprzez przeszukiwanie jednej tylko ścieżki w grafie przestrzeni stanów dla wszystkich typów podcelów. Ceną szybkości algorytmu jest suboptymalność rozwiązania. Przykłady wyjaśniają propozycję innego podejścia.

SOLVING PROBLEMS IN BLOCCK WORLD

Summary. Block world is a cconvenient representation for planning problems. There are many different algorithms of state space searching which use STRIPS representation for planning. Some of them search for solution through a space of world-states, some of them through a space of plan-states. In the paper an extended hierarchy of subgoal collections and different domains of operators are presented. Some subgoals are difficcult for all algorithms. Main of the work is a proposition of new algorithm. The proposal of another shift for state space searching finds solution of STRIPS problem by searching only one path in state-space graph for all subgoal types in block world. The price for one path searching is suboptimality of the solution. Two examples explain proposition of the shift.

1. Wprowadzenie

Zadanie w systemie STRIPS jest reprezentowane przez trzy składniki: (υ, Σ, Ω). Innymi słowy, na początku pracy system otrzymuje informacje dotyczące:

– zbioru opisów operatorów (modeli działań robota) - υ,

– opisu początkowej sytuacji problemowej - Σ,
– opisu końcowej sytuacji problemowej - Ω.

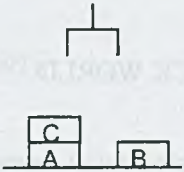
Wynikiem algorytmu jest plan działań robota, tzn. lista operatorów, których wykonanie prowadzi do rozwiązania zadania.

Opis początkowej sytuacji problemowej oznacza zbiór faktów, które opisują obiekty występujące w zadaniu oraz związki między nimi. W ten sposób przedstawiony jest opis końcowej sytuacji problemowej (celu) oraz wszystkich możliwych stanów pośrednich.

Operatory w systemie są reprezentowane przez listę, która z kolei składa się z trzech podlist: *listy warunków stosowalności (precondition list), listy skreśleń (delete list) oraz listy dopisków (add list).*

Lista warunków stosowalności opisuje warunki, jakie muszą być spełnione, aby dany operator mógł zadziałać. *Lista skreśleń* zawiera fakty, które przestają być prawdziwe w wyniku zastosowania danego operatora. Fakty te zostają skreślone z listy opisującej bieżącą sytuację w zadaniu, jeśli zastosujemy dany operator. *Lista dopisków* natomiast zawiera fakty, które stają się prawdziwe w wyniku działania operatora. Fakty te zostają dopisane do listy opisującej sytuację bieżącą w zadaniu, jeśli zastosujemy dany operator. Podsumowując można powiedzieć, że dwie ostatnie listy opisują efekty działania operatora na stan, w jakim znajduje się zadanie [5].

Przykładowa sytuacja początkowa jest przedstawiona na ryunku 1.



Rys.1. Przykładowa sytuacja początkowa
Fig.1. An example initial state

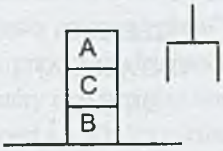
Sytuacja ta w systemie STRIPS może być opisana za pomocą następującej listy [3]:

CLEAR (B)
CLEAR (C)
ON (C,A)
HANDEMPTY
ONTABLE (A)
ONTABLE (B)

Znaczenie poszczególnych faktów jest następujące: CLEAR (X) oznacza, że na klocku X nie znajduje się żaden inny klocek; ON(X,Y) - klocek X znajduje się na klocku Y; HANDEMPTY - ręka robota jest pusta; ONTABLE (X) - klocek X leży na stole.

Przypadek, gdy robot przenosi klocek X, opisuje się za pomocą predykatu HOLDING(X).

Sytuacja końcowa w systemie jest przedstawiona jako koniunkcja predykatów ON(X,Y). Przykładowy cel może być następujący: ON(A,C) ^ ON(C,B). Ilustracja celu jest przedstawiona na rysunku 2.



Rys.2. Sytuacja końcowa
Fig.2. An example goal state

Na świat opisany w powyższy sposób możemy działać następującymi operatorami:

1) **pickup (x)** - oznacza podniesienie klocka X ze stołu;
Lista warunków stosowalności: ONTABLE(X), CLEAR(X), HANDEMPTY
Lista skreśleń: ONTABLE(X), CLEAR(X), HANDEMPTY
Lista dopisków: HOLDING(X)

2) **putdown (x)** - oznacza położenie klocka X na stół;
Lista warunków stosowalności: HOLDING(X)
Lista skreśleń: HOLDING(X)
Lista dopisków: ONTABLE(X), CLEAR(X), HANDEMPTY

3) **stack(x,y)** - oznacza położenie klocka X na klocku Y;
Lista warunków stosowalności: HOLDING(X), CLEAR(Y)
Lista skreśleń: HOLDING(X), CLEAR(Y)
Lista dopisków: HANDEMPTY, ON(X,Y), CLEAR(X)

4) **unstack(x,y)** - oznacza ściągnięcie klocka X z klocka Y;
Lista warunków stosowalności: HANDEMPTY, CLEAR(X), ON(X,Y)
Lista skreśleń: HANDEMPTY, CLEAR(X), ON(X,Y)
Lista dopisków: HOLDING(X), CLEAR(Y)

Jak widać, w tym przypadku listy warunków stosowalności i skreśleń pokrywają się. Spróbujmy teraz zadziałać jednym z operatorów w świecie opisanym na rysunku 1. Operatorów **putdown(x)** oraz **stack(x,y)** nie możemy zastosować, gdyż w obu *przypadkach lista warunków stosowalności* wymaga, aby ręka robota trzymała klocek X (opis sytuacji musiałby zawierać predykat HOLDING(X)). Jedynie dwa działania, jakie możemy wykonać, to: **unstack(x,y)** (po podstawieniu x=C oraz y=A) lub **pickup(x)** (po podstawieniu x=B). W celu ilustracji wybierzmy drugi.

Operator **pickup(B)** wymaga, aby opis sytuacji bieżącej zawierał listę: <ONTABLE(B), CLEAR(B), HANDEMPTY>. Po sprawdzeniu (rys.1) wykreślamy z opisu sytuacji fakty zawarte w *liście skreśleń*. Teraz opis sytuacji bieżącej ma postać:

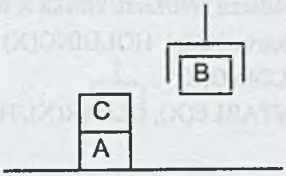
CLEAR(C)
ONTABLE(A)
ON(C,A)

Czynność ostania, którą musimy wykonać, to dopisanie do opisu sytuacji faktów zawartych w *liście dopisków*, czyli predykatu <HOLDING(B)>.

Po wykonaniu operacji **pickup(B)** nowa sytuacja bieżąca jest opisana przez następującą listę:

CLEAR(C)
ONTABLE(A)
ON(C,A)
HOLDING(B)

Nowa sytuacja jest przedstawiona na rysunku 3.



Rys.3. Sytuacja bieżąca po zastosowaniu operatora pickup(B)
Fig.3. New state after performing pickup(B) operator

2. Algorytmy przeszukiwania stanów (planowania kolejności operatorów); zastosowanie reguły „causal link"

W punkcie tym opisywane są trzy algorytmy: POCL, TOCL, TOPI [1]. Każdy z nich w inny sposób przeszukuje przestrzeń stanów.

Zgodnie z tym, co zostało powiedziane, planowanie można rozpatrywać jako przeszukiwanie *grafu skierowanego* „stanu świata" (world states), w którym *wierzchołki* oznaczają opis sytuacji bieżącej w zadaniu (specyfikują, co jest prawdziwe w „stanie świata"), natomiast *łuki* reprezentują operatory. Problem planowania STRIP jest trójką (υ,Σ,Ω), w której υ oznacza operatory, Σ oznacza sytuację początkową, Ω - oznacza sytuację końcową.

Inne podejście do problemu planowania reprezentuje system NOAH, który przeszukuje przestrzeń „stanów planu" (plan-states), przy czym przestrzeń ta może być niekompletna.

Zgodnie z [1] „stan planu" jest trójką (S,O,B), w której S oznacza zbiór kroków w planie (akcje) (set of plan steps), O oznacza zbiór przymusów kolejności (set of ordering constraints), które ustalają kolejność w zbiorze S, natomiast B oznacza zbiór obowiązujących przymusów (set of binding constraints) nad zmiennymi wymienianymi przez kroki w S.

Przy takim podejściu łuki pomiędzy „stanami planu" reprezentują rozszerzenia niekompletnego planu (np. dodanie kroku, przymus kolejności). Początkowy „stan planu" zawiera dwa kroki s_0 (który dodaje Σ) i s_∞ (który ma Ω za listę warunków stosowalności). Zbiór O ma jeden przymus, który wymusza s_0 przed s_∞. Zbiór υ definiuje, co może być dodane do zbioru S w „stanie planu". Celem przeszukiwania jest znalezienie docelowego stanu planu. Zgodnie z [1] docelowym stanem planu (S,O,B) jest stan planu, w którym listy warunków stosowalności każdego kroku s_i są prawdziwe (necessarily true) w sytuacji wejściowej s_i.

Poniżej krótko omówiono trzy algorytmy, które przeszukują stany planu [1].

2.1. POCL *(Partial - Order, Causal - Link algorithm)*

Algorytm POCL przy przeszukiwaniu stanów planu (planowaniu kolejności operatorów) korzysta z tzw. reguły „causal link". Reguła ta jest używana do zapamiętywania zamiaru wprowadzenia kroku do planu oraz do ochrony tego zamiaru (goal protection). Na przykład, istnieje pięć różnych przymusów, które mogą być dodane do ochrony

$$s_i \xrightarrow{on(x,y)} s_j$$

przed krokiem (operatorem) s_k, kasującym on(a,b) (lista skreśleń tego operatora zawiera predykat on(a,b)):

1. s_k przed s_i,
2. s_k po s_j,
3. s_k pomiędzy s_i i s_j oraz $x \neq a$, $y \neq b$,
4. s_k pomiędzy s_i i s_j oraz $x \neq a$, $y = b$,
5. s_k pomiędzy s_i i s_j oraz $x = a$, $y \neq b$.

Dla algorytmu POCL informacją wejściową są: początkowy stan planu (S,O,B), lista warunków stosowalności s_∞ w G oraz pusta lista L. Każda lista warunków stosowalności p kroku stanu-planu S_j jest warunkiem otwartym (open condition), jeśli nie odpowiada causallinkowi $S_i \xrightarrow{p} S_j \in L$. Algorytm poszukuje docelowy stan planu przez eliminację otwartych warunków w G po upewnieniu się, że warunki causal-link w liście L są zachowane.

2.2. TOCL *(Total - Order, Causal - Link algorithm)*

Drugi algorytm, TOCL, jest podobny do POCL, z tym że generuje on plany zawierające całkowicie zdeterminowaną kolejność kroków. Algorytm ten może wstawiać kroki gdziekol-

wiek pomiędzy s_0 i s_∞, , ale nie może zmieniać raz ustalonej ich kolejności. Reguła causal-link jest używana jedynie do ustalenia możliwego położenia nowego kroku.

2.3. TOPI *(Total - Order, Prior - Insetion algorithm)*

Ostatni z algorytmów, TOPI, jest najprostszy. Dodaje on kolejne kroki do początku planu, w związku z czym nie używa reguły causal-link i jest odpowiednikiem algorytmu wniskowania wstecz (backward chaining) [3]. Rozważa on wszystkie możliwe kroki, które mogłyby dodać podcel do planu bez niszczenia podceli jeszcze nie rozwiązanych.

3. Rozszerzona hierarchia podcelów (an extended hierarchy of subgoal collections)

3.1. Podcele niezależne *(independent subgoals)*

Podcele nazywamy podcelami niezależnymi, jeśli każdy operator zmienia dystans tylko do pojedynczego podcelu. Innymi słowy globalnie optymalne rozwiązanie można uzyskać przez optymalne rozwiązywanie poszczególnych podceli w jakimkolwiek porządku [1, str.71].

3.2. Podcele liniowe *(serializable subgoals)*

Podcele nazywamy podcelami liniowymi, jeśli istnieje taka kolejność wśród podceli, że mogą one być zawsze rozwiązane sekwencyjnie bez niszczenia podceli już osiągniętych w tej kolejności [1, str.71].

3.2.1. Podcele trywialnie liniowe (trivially serializable)

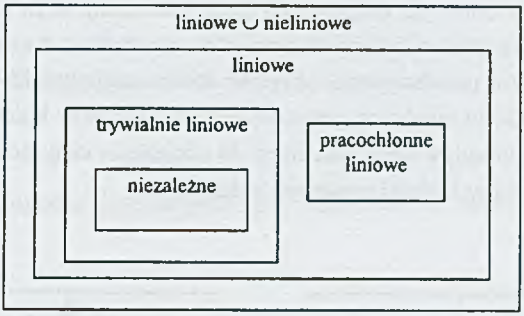
Podcele nazywamy podcelami trywialnie liniowymi, jeśli mogą być rozwiązane w jakimkolwiek porządku bez niszczenia podceli już osiągniętych [1].

3.2.2. Podcele trudnoliniowe (laboriously serializable)

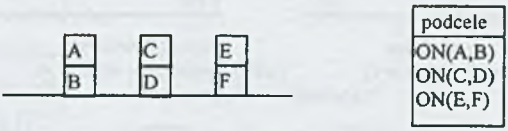
Podcele nazywamy podcelami trudnoliniowymi, jeśli istnieje przynajmniej jedna kolejność realizacji podcelów, która nie może być rozwiązana sekwencyjnie bez niszczenia któregoś z podceli już osiągniętych [1].

3.3. Podcele nieliniowe *(nonserializable subgoals)*

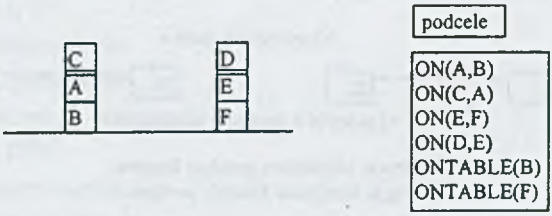
Podcele nazywamy podcelami nieliniowymi, jeśli nie można ustalić takiej kolejności realizacji podcelów, której realizacja nie niszczyłaby któregoś z podceli już osiągniętych [2].



Rys.4. Rozszerzona hierarchia podcelów
Fig.4. An extended subgoal hierarchy



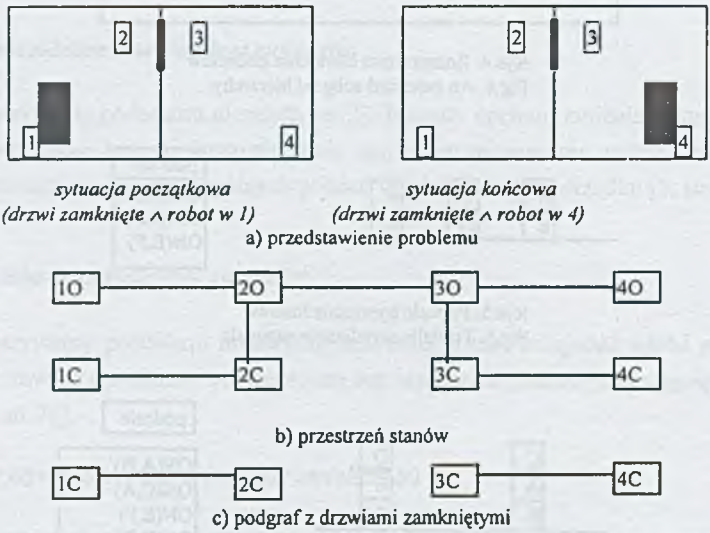
Rys.5. Podcele trywialnie liniowe
Fig.5. Trivially serializable subgoals



Rys.6. Podcele trudnoliniowe
Fig.6. Laboriously serializable subgoals

Rys.7. Podcele nieliniowe (anomalia Sussman'a)
Fig.7. Nonserializable subgoals (Sussman's anomaly)

Rysunek 8 przedstawia tzw. nieliniowy problem Roacha [2]. Zadaniem dla robota, który stoi w punkcie 1, jest przejscie do punktu 4, przy czym drzwi powinny być zamknięte po wykonaniu zadania przez robota. Cel zadania składa się zatem z listy dwóch podcelów: *< drzwi zamknięte ∧ robot w punkcie 4>*. Problem ten jest problemem nieliniowym, gdyż nie może być rozwiązany przez szeregowanie podcelów; przyjęcie jakiejkolwiek kolejności wykonywania podcelów prowadzi w efekcie do niszczenia podcelu już osiągniętego. Zastosowanie operacji ochrony celów już osiągniętych *(goal protection)* także nie prowadzi do rozwiązania problemu. W rozważanym przykładzie ochrona podcelu *<drzwi zamknięte>* prowadzi do przestrzeni stanów przedstawionej na rys.8c. Rozwiązanie problemu mogłoby polegać na analizie, której skutkiem byłoby wygenerowanie stosu podcelów liniowych. Kolejne realizacje celów na stosie podcelów doprowadziłoby do osiągnięcia celu globalnego, np.: *<robot w 2, drzwi otwarte, robot w 3, drzwi zamknięte, robot w 4>*.



Rys.8. Nieliniowy problem Roacha
Fig.8. Nonlinear Roach's problem

4. Dziedziny operatorów

Według [1] operator należy do dziedziny D^xS^y, jeśli zawiera x predykatów, które przestają być prawdziwe po jego zastosowaniu, oraz wymaganych jest y kroków do osiągnięcia pojedynczego podcelu.

Na przykład operator ze „Świata Klocków" **unstack(x,y)**, zdefiniowany przez listy:

Lista warunków stosowalności: HANDEMPTY, CLEAR(X), ON(X,Y)

Lista skreśleń: HANDEMPTY, CLEAR(X), ON(X,Y)

Lista dopisków: HOLDING(X), CLEAR(Y)

należy do dziedziny D^3S^2, gdyż jego lista skreśleń zawiera 3 predykaty, zaś osiągnięcie pojedynczego podcelu {ON(X<Y)} wymaga zastosowania dwóch operatorów, np.: unstack(X,Z) i stack(X,Y).

Ogólnie można powiedzieć, że operatory systemu STRIPS zaprezentowane w punkcie 1 należą do dziedziny D^mS^2, gdzie m oznacza wielkość listy skreśleń.

5. Szybkości działania poszczególnych algorytmów dla różnych dziedzin operatorów

Szybkości działania algorytmów są zgrupowane poniżej i ilustrują zależność wzrostu czasu obliczeń od liczby podcelów.

Dziedzina D^0S^1:
TOPI: kwadratowa,
TOCL: liniowa,
POCL: liniowa.

Dziedzina D^mS^1:
TOPI: eksponencjalna,
TOCL: liniowa,
POCL: liniowa.

Dziedzina D^1S^1:
TOPI: eksponencjalna,
TOCL: eksponencjalna,
POCL: liniowa.

Dziedzina D^mS^2:
TOPI: eksponencjalna,
TOCL: eksponencjalna,
POCL: eksponencjalna.

6. Propozycja innego podejścia do przeszukiwania przestrzeni stanów

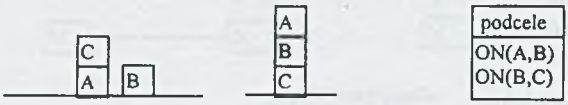
Jak wynika z powyższych rozważań, operatory należące do dziedziny D^mS^2 (np. Świat Klocków przedstawiony w punkcie 1), sprawiają problemy każdemu algorytmowi. Przy dużej liczbie podcelów czas szukania rozwiązania problemu jest bardzo długi.

Czas poszukiwania rowiazania może zostać zredukowany poprzez dekompozycję celu głównego. Warunki przeprowadzenia takiej dekompozycji zostały sformułowane w [6]. W tym przypadku czas poszukiwania rozwiązania zostaje skrócony, jednak nadal istnieje eksponencjalna zależność pomiędzy złożonością problemu a czasem jego rozwiązania. W związku z tym, opierając się na Świecie Klocków zdefiniowanym w punkcie 1, zaproponowano inne podejscie do przeszukiwania przestrzeni stanów, które gwarantuje znalezienie rozwiązania problemu w sposób trywialny, nawet dla podcelów nieliniowych.

Przez znalezienie rowiązania problemu rozumie się podanie takiej sekwencji operatorów, która przeprowadza stan zadania z sytuacji początkowej do końcowej, jednak niekoniecznie w sposób optymalny. Rezegnację z optymalności usprawiedliwia odwołanie się do jednej z definicji sztucznej inteligencji: jest to dyscyplina naukowa, której przedmiotem jest rozwiązywanie problemów sposobami wzorowanymi na naturalnych działaniach, zachowaniach i procesach poznawczych człowieka [5]. podobnie jak sposób rozumowania, tak i proponowane podejście nie zawsze prowadzi do rozwiązań optymalnych.

Dla ilustracji rozpatrzmy przykład 1.

Przykład 1

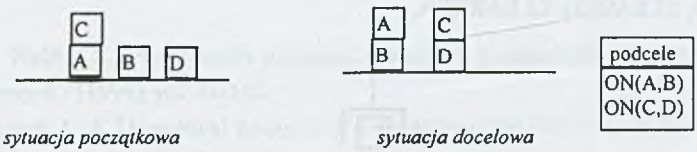


Przykład 1. Podcele nieliniowe (anomalia Sussmana)
Example 1. Nonlinear subgoals (Sussman's anomaly)

Stos podcelów jest nieliniowy, czyli nie można rozwiązać problemu przez ustalenie kolejności realizacji podcelów. Prosta analiza celu prowadzi do wniosku o prawdziwości predykatu ONTABLE (C), który przekształca stos podcelów w liniowy. Operację tę można nazwać linearyzacją celu. Kolejna realizacja podcelów ONTABLE (C), ON(B,C), ON(A,B) prowadzi do znalezienia rozwiązania optymalnego od danej sytuacji początkowej. Podobnie jest dla stosu podcelów o dowolnej długości, czyli dowolnej wysokości docelowej wieży klocków [7]. *Należy przy tym zaznaczyć, że poszukiwanie rozwiązania ogranicza się do przejścia tylko jednej ścieżki w grafie przestrzeni stanów zadania.*

Sytuacja komplikuje się przy bardziej złożonej sytuacji docelowej. Rozważmy przykład 2.

Przykład 2



Przykład 2. Złożone podcele nieliniowe
Example 2. Complex nonserialisable subgoals

Podobnie jak w przykładzie 1 proste wnioskowanie prowadzi do prawdziwości predykatów ONTABLE(B) oraz ONTABLE(D). Otrzymujemy liniowy stos podcelów: ONTABLE(B), ON(A,B), ONTABLE(D), ON(C,D). Istnieje jednak kilka kombinacji kolejności realizacji podcelów, których realizacja prowadzi do osiągnięcia celu. W przykładzie 2 jest ich pięć:

ONTABLE(B)	ONTABLE(B)	ONTABLE(D)	ONTABLE(D)	ONTABLE(D)
ON(A,B)	ONTABLE(D)	ONTABLE(B)	ONTABLE(B)	ON(C,D)
ONTABLE(D)	ON(A,B)	ON(A,B)	ON(C,D)	ONTABLE(B)
ON(C,D)	ON(C,D)	ON(C,D)	ON(A,B)	ON(A,B)

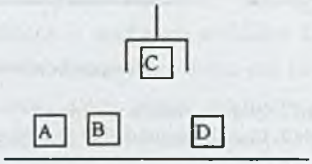
Kolejność, która rozwiązuje zadanie w sposób optymalny (tzn. w najmniejszej liczbie kroków), jest uzależniona od sytuacji początkowej w zadaniu i dla rozważanego przykładu jest to kolejność 4 lub 5. Liczba kombinacji wzrasta oczywiście wraz ze złożonością sytuacji docelowej.

Propozycja innego podejścia do rozwiązania takiego problemu jest następująca. Po operacji linearyzacji celu i ustaleniu, które podcele nie mogą być realizowane przed innymi, wybieramy **jedną** kombinację kolejności realizacji podcelów. Następnie podczas realizacji podcelów według wybranej kolejności, po każdorazowym wygenerowaniu operatora **unstack(x,y)**, należy porównać ze sobą trzy listy: cel globalny, stos podcelów pozostałych do zrealizowania oraz opis sytuacji bieżącej. Porównanie ma na celu sprawdzenie, czy podczas kolejnego kroku nie można by zrealizować któregoś z pozostałych podcelów znajdujących się na stosie celów, pod warunkiem, że podcel ten nie będzie zniszczony w przyszłości. Sens takiego postępowania może być zilustrowany na rozważanym już przykładzie 2.

Niech wybraną kombinacją kolejności realizowania podcelów będzie kolejność pierwsza: <ONTABLE(B), ON(A,B), ONTABLE(D), ON(C,D)>. Podcel pierwszy, ONTABLE(B), jest już zrealizowany w opisie sytuacji początkowej. Aby osiągnąć następny podcel, ON(A,B), należy z klocka A zdjąć klocek C i klocek A położyć na klocku B. Po wygenerowaniu operatora unstack(C,A) otrzymujemy sytuację przedstawioną na rysunku 9.

Krokiem następnym jest porównanie trzech list:
Cel globalny: <ONTABLE(B), ON(A,B), ONTABLE(D), ON(C,D)>.
Stos podcelów pozostałych do zrealizowania: <ON(A,B), ONTABLE(D), ON(C,D)>.

Opis sytuacji bieżącej: <ONTABLE(A), ONTABLE(B), ONTABLE(D), HOLDING(C), CLEAR(A), CLEAR(B), CLEAR(D)>.



Rys.9. Sytuacja w zadaniu po zastosowaniu operatora unstack(C,A)
Fig.9. New state after performing unstack(C,A) operator

W kroku następnym można zrealizować podcel ON(C,D) za pomocą operatora **stack(C,D)**, gdyż klocek D jest już „na swoim miejscu" (zarówno cel globalny, jak i opis sytuacji bieżącej zawierają predykat ONTABLE(D)) i nie ma na nim żadnego innego klocka: predykat CLEAR(D). Listę warunków stosowalności dla operatora stack (C,D) dopełnia lista dopisków operatora unstack(C,A).

Ostateczna lista operatorów po zastosowaniu innego podejścia do analizy problemu ma postać:

unstack(C,A)
stack(C,D)
pickup(A)
stack(A,B)

7. Podsumowanie

Świat Klocków jest wygodną reprezentacją dla problemów planowania. Istnieje wiele różnych algorytmów, które używają reprezentacji zadania typu STRIPS przy rozwiązywaniu zadań ze Świata Klocków. Niektóre z nich przy poszukiwaniu rozwiązania przeszukują przestrzeń stanów świata, inne przestrzeń stanów planu. W artykule prezentowana jest rozszerzona hierarchia podcelów oraz różne dziedziny operatorów.

Niektóre z podcelów są trudne do rozwiązania dla wszystkich algorytmów. Propozycja innego podejścia przeszukiwania przestrzeni stanów znajduje rozwiązanie poprzez przeszukiwanie jednej tylko scieżki w grafie przestrzeni stanów dla wszystkich typów podcelów. Ceną szybkości algorytmu jest suboptymalność rozwiązania. Przykłady wyjaśniają propozycję innego podejścia.

Artykuł finansowany z funduszu badań własnych BW 421/Raul/97/1.

LITERATURA

1. Barrett, A., Weld D.S.: Partial-order planning: evaluating possible efficiency gains. Artifical Intelligence 67 (1994), pp. 71-112.
2. Joslin D., Roach J.: A Theoretical Analysis of Conjuctive-Goal Problems. Artificial Intelligence 41 (1989/1990), pp.97-106.
3. Nilson N.J.: Principles of Artificial Intelligence. Tioga Publishing Company, Palo Alto, California. [w tłum.] R. M. Abdusamatow i J.I. Kriukow: Principy iskusstwiennogo intieliekta. Radio i Swiaz, Moskwa 1985.
4. Podstawy Robotyki. Praca zbiorowa pod redakcją A. Moreckiego i J. Knapczyka. Rozdział III: Planowanie zadań robotów. WNT, Warszawa 1994.
5. Kowalowski H.: W sprawie metod komputerowego wspomagania podejmowania decyzji sztucznej inteligencji w robotyce [w] ZN Politechniki Śląskiej, s. Automatyka, 1990, nr 101.
6. Sicrocki I.: A Serial decompresition of Planning Problems. Proc. Fourth International Symposium on Methods and Models in Automation and Robotics. Międzyzdroje, Poland, pp. 1179-1184, 1997.
7. Gałuszka A.: System umożliwiający rozwiązywanie prostych zadań ze „Świata Klocków". Praca magisterska. Gliwice, 1996.

Recenzent: Prof.dr hab.inż. Zbigniew Banaszak

Wpłynęło do Redakcji 19.05.1997 r.

Abstract

Block world is a convenient representation for planning problems. There are many different algorithms of state space searching which use STRIPS representation for planning. Some of them searche for solution through a space of world-states, some of them through a space of plan-states. In the paper an extended hierarchy of subgoal collections and different domains of operators are presented. Some subgoals are difficult for all algorithms. The proposal of another shift for state space searching finds solution of STRIPS problem by searching only one path in state-space graph for all subgoal types in block world. The price for one path searching is suboptimality of the solution. Two examples explain proposition of the shift.