Zawartość programu w Matlab

Operując w Świecie Klocków można posługiwać się operatorami, predykatami oraz listami stosowalności, dopisków i skleśleń. W podobny sposób napisany jest zbiór skryptów i funkcji w środowisku Matlab. Poniżej znajduje się lista dostępnych funkcji wraz z parametrami oraz możliwościami zastosowania.

Każda plik to funkcja którą można wywołać. Funkcje spełniają rolę operatorów, predykatów, funkcji inicjalizujących, pomocniczych i algorytmy planowania.

Dane

Główną daną na której prowadzone są operacje jest struktura Blocks przyjmująca predykaty

```
Blocks=struct('name',char(65),'ontable',0,'clear',0','on',0,'holding',0);
```

Każdy klocek ma swoją nazwę *name*, która w domyśle zapisana jest dużą literą alfabetu, zaczynając od „A” (znak ASCII 65). Predykaty *ontable* i *clear* to zmienne typu prawda/fałsz przyjmujące wartości 0 lub 1. Predykat *on*, w którym napisana jest nazwa klocka (znak ASCII) na którym się znajduje lub 0 w przypadku, gdy nie znajduje się na żadnym. Predykat *holding* nie opisuje stanu chwytaka, jednak stan klocka, wskazując, czy klocek jest trzymany czy nie.

Struktura Blocks zawiera opis wszystkich klocków ze sceny i jej rozmiar powinien być równy liczbie dostępnych klocków. Jest to zatem opis danej sytuacji (początkowej, docelowej lub pośredniej) ze świata klocków.

Na scenę można wpływać operacjami, które przechowywane są w strukturze

```
Solutions=struct('f',@CreateBlocks,'var',0,'prev',0,'state',Init,'cost',0);
```

Zmienna *f* to odnośnik do użytej funkcji, która doprowadziła do obecnej sytuacji zapisanej w strukturze *state*, czyli poprzednio opisaną strukturę. Poprzedni krok, czyli stan przed zastosowaniem operatora *f* wskazuje *prev*, czyli wskaźnik do pozycji w tablicy struktur. Zmienna *var* to parametry funkcji *f*. Na obecną chwilę funkcje przyjmują albo 1 albo 2 parametry, czyli przykładowo [‘A’] w pickup(A) lub [‘A’ ‘B’] w stack(A,B). *Cost* to koszt dotarcia do danego rozwiązania. W przypadku funkcji heurystycznej, koszt zapisywany jest jako 2 składowe [g(v) h(v)].

Inicjalizacja

Dotyczy funkcji:

- `function Blocks=CreateBlocks(N,Blocks)`
- `function Blocks=RandomSetBlocks(N)`
- `function Blocks=ReadSTRIPS(fileName)`
- `Test.m`

Warto zwrócić uwagę na to, że prawie każda funkcja przyjmuje jako ostatni parametr strukturę *Blocks*, modyfikuje ją i ją zwraca. Istnieje jednak możliwość wywołania funkcji bazując na zmiennej globalnej.

CreateBlocks przyjmuje jako parametr liczbę N i przygotowuję pustą strukturę Blocks zawierającą N elementów.

RandomSetBlocks losowo układa klocki na stole tworząc losową sytuację. Wewnątrz funkcji, w pierwszej linijce można znaleźć parametr PrawdopWiezy. Zmiana parametru między 0 a 1 odnosi się do tendencji tworzenia wysokich wież.

ReadSTRIPS służy do wczytywania sytuacji opisanej w pliku. Wywoływane są wszystkie predykaty, które wypisane są w kolejnych linijkach pliku tekstowego. W celu zapoznania się ze składnią należy sprawdzić pliki 'Final.txt' oraz 'Initial.txt'.

Test jest to skrypt testowy, który pokazuje sposób uruchomienia przykładu.

Predykaty

Dostępne predykaty to:

- `function Blocks=holding(Name,Blocks)`
- `function Blocks=on(ktory,gdzie,Blocks)`
- `function Blocks=ontable(Name,Blocks)`

Nazwy funkcji odpowiadają nazwom predykatów oraz pozycjom ze struktury Blocks.

Predykatów można używać operując na zmiennej globalnej lub na zmiennej przedstawionej w parametrze wyjściowym i wejściowym. W ten sposób wywołanie:

- on('B','A')

Zmieni w opisie zmiennej globalnej Blocks predykaty (wskaźnik 1 i 2 odpowiada klockom A i B):

Blocks(1).clear=0
Blocks(2).on='A'
Blocks(2).clear=1
Blocks(2).holding=0

Wywołanie tej samej funkcji w sposów:

- Init=on('B','A',Init)

Spowoduje wykonanie tych samych czynności, jednak dla struktury zapisanej w zmiennej Init.

Ten sam predykat można również wywołać w **pliku tekstowym** pisząc:

- on(B,A)

W tym przypadku, wywołana zostanie funkcja z pełnym zestawem parametrów, a modyfikowana zostanie struktura, która podana jest jako parametr funkcji ReadSTRIPS.

Predykaty nie posiadają sprawdzenia stosowalności. Źle opisana sytuacja zostanie źle przedstawiona, co może powodować brak rozwiązania problemu planowania.

Operatory

Dotyczy funkcji:

- `function Blocks=pickup(Name,Blocks)`
- `function Blocks=putdown(Name,Blocks)`
- `function Blocks=stack(Name,Gdzie,Blocks)`
- `function Blocks=unstack(Name,Blocks)`

Nazwy funkcji odpowiadają nazwom operatorów. Każda funkcja skonstruowana jest zgodnie z założeniami STRIPS, czyli posiada listę stosowalności, dopisków i skreśleń. Podobnie jak w przypadku predykatów, zastosowanie bez podania struktury na wejściu i wyjściu modyfikować będzie zmienną globalną.

W przypadku nie spełnienia listy stosowalności funkcja zwraca błąd „Operacja nie jest możliwa”. Oznacza to, że przy pisaniu algorytmu planowania można użyć funkcji *try / catch*.

Funkcje pomocnicze

- `function DrawBlocks(B)`

Funkcja jako parametr wejściowy przyjmuje opis stanu w postaci struktury Blocks i wizualizuje ją na wykresie. Warto zauważyć, że w opisie świata klocków, jeżeli klocek znajduje się na stole, nie posiada on konkretnej pozycji, więc dla ułatwienia klocki wizualizowane są w odpowiadających sobie kolumnach. Klocek ‘A’ będzie zawsze na stole w kolumnie 1, klocek ‘C’ w kolumnie 3, itd.

- `function Path=FinalPath(Solutions)`

Wynikiem algorytmu planowania jest struktura Solutions, która zawiera wszystkie przeanalizowane wierzchołki drzewa. Funkcja FinalPath zapisuje do struktury Path operacje prowadzące od sytuacji początkowej do docelowej.

Dodatkowo funkcja wyposażona jest w wizualizacje kolejnych kroków, czyli uruchamiane jest DrawBlocks dla kolejnych stanów pośrednich. Każdy krok należy zatwierdzić dowolnym przyciskiem. W celu pominięcia wizualizacji można zakomentować odpowiednie linijki kodu.

Algorytmy planowania

- `function Solutions=BruteForce(Init,Final)`
- `function Solutions=Heuristic(Init,Final)`

Dostępne są dwa algorytmy planowania. Przegląd zupełny (BruteForce), który bazuje na przeglądzie w szerz od przodu oraz algorytm heurystyczny (Heuristic), który używa funkcji heurystycznej do oceny wierzchołków. W obu przypadkach funkcje przyjmują dwa opisy stanu, początkowy i docelowy, a zwracają strukturę która zawiera wszystkie przeanalizowane operacje jako struktura Solutions.

Można powiedzieć, że oba algorytmy działają w ten sam sposób. Dla wybranej sytuacji pośredniej lub początkowej sprawdzane są wszystkie możliwe kombinacje operatorów, a te które są możliwe do wykonania zapisywane są jako sytuacje pośrednie w strukturze Solutions. Różnicą jest sposób wyboru następnego rozwiązania pośredniego z którego poszukiwane są następne rozwiązania pośrednie. W obu przypadkach wybierane jest pierwsze rozwiązanie ze struktury, które powiada najmniejszą wartość *cost.*

W przypadku przeglądu zupełnego, kosztem jest liczba kroków, która zostały wykonane do tego etapu, czyli

Cost=g(v).

W przypadku funkcji heurystycznej wybierane są te wierzchołki, które mają najmniejszą wartość funkcji heurystycznej.

Dla funkcji heurystycznej, składowa h(v) liczona jest w linijce 78 i 79 jako:

$$h(v) = \sum_{i=1}^{N} |F(i).ontable - S(i).ontable| + \sum_{i=1}^{N} |F(i).clear - S(i).clear|$$

gdzie N to liczba wszystkich klocków, F to opis sytuacji docelowej, a S to opis sytuacji pośredniej (obecnej).

Powyższa struktura funkcji heurystycznej jest tylko propozycją. Istnieje możliwość zaproponowania innej / lepszej funkcji heurystycznej.

W przypadku **zbyt długiej pracy algorytmu,** spowodowanym zbyt skomplikowanym problemem planowania, w celu zatrzymania programu można użyć skrótu klawiszowego Ctrl+C. Główna pętla algorytmu wykonuje się 20 000 razy. Parametr ten można zmienić w 8 lub 9 linijce jako `for k=1:20000`

Warto zwrócić uwagę na jeden zestaw zmiennych lokalnych w funkcjach planowania. Są to zmienne:

```
Funs={@pickup @putdown @unstack};
Funs2p={@stack};
```

Znaleźć je można w 5-6 linijce. Zawierają one nazwy funkcji, operatorów, które są sprawdzane w każdym kroku jako możliwe do wykonania akcje. W przypadku napisania własnego operatora należy dodać nazwę funkcji do zmiennej Funs, jeżeli przyjmuje ona 1 parametr lub Func2p, jeżeli przyjmuje 2 parametry.

Program ćwiczenia

1. **(Zadanie obowiązkowe, nieoceniane)** Należy zapoznać się ze sposobem działania programu. W tym celu należy wpisać w Command Window kolejno:

   global Blocks
   CreateBlocks(5)
   RandomSetBlocks(5)

   Powinien uruchomić się wykres. Należy sprawdzić jak wygląda scena początkowa. Następnie należy zmodyfikować opis stosując predykaty i operatory. Przykład:

   ontable('A')
   DrawBlocks()
   pickup('A')

   Należy sprawdzić w Workspace zawartość zmiennej Blocks. Można również sprawdzić składnie poszczególnych funkcji.

   Należy uruchomić skrypt Test.m oraz sprawdzić jego sposób działania.

2. **(Zadanie obowiązkowe, ocena 3.0)** Należy zdefiniować zadanie zawierające 4 klocki. Zadeklarować sytuacje początkową i docelową w postaci skryptu lub pliku tekstowego a następnie rozwiązać problem z użyciem obu dostępnych algorytmów. Należy zwrócić uwagę na liczbę wykonanych kroków oraz liczbę przeanalizowanych przypadków.
3. **(ocena 3.5)** Należy sprawdzić liczbę przeanalizowanych przypadków w zależności od liczby klocków oraz liczby kroków, które potrzebne są do rozwiązania.
4. **(ocena 4,0)** Należy zaproponować zmianę funkcji heurystycznej i powtórzyć eksperymenty z punktu 3. Porównać wyniki.
5. **(ocena 4,5)** Należy zaproponować przykład i funkcje heurystyczną, w którym otrzymane rozwiązanie nie jest optymalne (liczba kroków alg. Heurystycznego jest większa niż w przypadku przeglądu zupełnego).
6. **(ocena 5.0)** Należy rozszerzyć działanie kodu. Dodać operator lub predykat w zgodzie ze standardem STRIPS i zintegrować go z istniejącym kodem. Sprawdzić wpływ na wyniki otrzymane w punkcie 3 i 4.