Plánovací problém hry Sokoban

Adam Gonšenica

8.1.2023

1. Úvod

Práca sa zaoberá tvorbou programu určeného na vyriešenie plánovacieho problému hry Sokoban pomocou programovacieho jazyku Prolog. Tento dokument oboznamuje čitateľa s pravidlami a cieľom logickej hry Sokoban. Detailne opisuje, akým spôsobom som sformalizoval stav hry a pravidla a akým spôsobom som implementoval nájde riešenia v Prologu.

2. Pravidlá a cieľ hry

Hra Sokoban sa hrá na obdĺžnikovej hracej ploche.. Hráč ovláda postavičku pomocou ktorej posúva krabice. Postavička sa dá posunúť v jednom kroku iba o jedno políčko a to buď vľavo, vpravo, hore alebo dole. Nie je povolené ju posúvať diagonálne. Postavička sa presunie na zvolené políčko iba v prípade, ak je prázdne. Nesmie vstúpiť na políčko kde je stena alebo box. Je povolené iba tlačenie boxov, nedajú sa ťahať alebo preskakovať. Box sa dá posunúť na políčko ak je prázdne - nesmie tam byť stena alebo nejaký box. Cieľom hry je presunúť všetky boxy do cieľových miest vyznačených na mape.

3. Formalizácia

V tejto kapitole opisujem, akým spôsobom som sformalizoval stav hry a pravidla do predikátov.

3.1 Počiatočný stav

Počiatočný stav budú tvoriť pozície xXyY kde X bude index stĺpca a Y index riadku. V ľavom hornom rohu je políčko x0y0. Vzájomnú polohu políčok zapíšeme nasledovne:

- right(xXyY1, xXyY2) – políčko xXyY1 je na pravo od xXyY2
- top(xXyY1, xXyY2) – políčko xXyY1 je nad políčkom xXyY2

Počiatočný stav budú ďalej tvoriť informácie o pozícii hráča, boxov, cieľov a prázdnych miest na mape. Formálne sa to zapíše nasledovne:

- sokoban(xXyY) - hráč je na pozícii XY
- box(xXyY) - krabica je na pozícii XY
- empty(xXyY) - pozícia XY je prázdna

Po neskoršej optimalizácii som počiatočný stav rozdelil do podmnožín obsahujúcich mapu políčok, Sokobanovu pozíciu, pozície boxov a pozície prázdnych políčok osobitne.

3.1 Cieľový stav

Cieľový stav budú tvoriť informácie o pozícii boxov box(xXyY).

3.2 Akcie

Zadefinoval som 8 akcií, 4 na posunutie hráča a 4 na posunutie krabice.

3.2.1 Akcia pushU(Sok,From,To)

Akcia predstavuje presun hráča a zároveň posunutie boxu o políčko do hora.

Musí platiť, že na pozícií Sok je sokoban, pozícií From je box a pozícia To je prázdna. Taktiež pozícia To je nad pozíciou From a pozícia From je nad pozíciou Sok.

Po akcii sa zo stavu odoberú pozície sokoban(Sok), box(From), empty(To) a pridajú sa sokoban(From), box(To), empty(Sok). .

```
opn(pushU(Sok,From,To), %name
    [[top(To,From),top(From,Sok)],sokoban(Sok),[ box(From)],  empty(To) ],
%preconditions
    [sokoban(From), empty(Sok), [box(To)]], %add list
    [sokoban(Sok),  empty(To),[box(From)]]). %delete list
```

3.2.2 Akcia pushD(Sok,From,To)

Akcia predstavuje presun hráča a zároveň posunutie boxu o políčko do dola. Je totožná s akciou pushU(Sok,From,To), až na to , že musí platiť, že pozícia From je nad pozíciou To a pozícia Sok je nad pozíciou From.

3.2.3 Akcia pushL(Sok,From,To)

Akcia predstavuje presun hráča a zároveň posunutie boxu o políčko do ľava. Je totožná s akciou pushU(Sok,From,To), až na to , že musí platiť, že pozícia From je na pravo od pozície To a pozícia Sok je na pravo od pozície From.

```
opn(pushL(Sok,From,To), %name
    [[right(From,To), right(Sok,From)], sokoban(Sok),[box(From)],  empty(To)],
%preconditions
    [sokoban(From), empty(Sok), [box(To)]], %add list
    [sokoban(Sok),  empty(To), [box(From)]]). %delete list
```

3.2.4 Akcia pushR(Sok,From,To)

Akcia predstavuje presun hráča a zároveň posunutie boxu o políčko do prava. Je totožná s akciou pushR(Sok,From,To), až na to , že musí platiť, že pozícia To je na pravo od pozície From a pozícia From je na pravo od pozície To.

3.2.5 Akcia moveU(From,To)

Akcia predstavuje presun hráča o políčko do hora.

Musí platiť, že na pozícií From je sokoban a pozícia To je prázdna. Taktiež pozícia To je nad pozíciou From.

Po akcii sa zo stavu odoberú pozície sokoban(From), empty(To) a pridajú sa sokoban(To), empty(From).

```
opn(moveU(From,To), %name
    [[top(To,From)],sokoban(From),[], empty(To)], %preconditions
```

```
    [sokoban(To), empty(From),[]], %add list
    [sokoban(From), empty(To),[]]). %delete list
```

3.2.6 Akcia moveD(From,To)

Akcia predstavuje presun hráča o políčko do dola.

Je totožná s akciou moveU(From,To), až na to , že musí platiť, že pozícia From je nad pozíciou To.

3.2.7 Akcia moveL(From,To)

Akcia predstavuje presun hráča o políčko do lava. Je totožná s akciou moveU(From,To), až na to , že musí platiť, že pozícia From je na pravo od pozície To.

```
opn(moveL(From,To), %name
    [[right(From,To)], sokoban(From),[], empty(To)], %preconditions
    [sokoban(To), empty(From),[]], %add list
    [sokoban(From), empty(To),[]]). %delete list
```

3.2.8 Akcia moveR(From,To)

Akcia predstavuje presun hráča o políčko do prava. Je totožná s akciou moveL(From,To), až na to , že musí platiť, že pozícia To je na pravo od pozície From

4. Hľadanie riešenia

4.1 Nájdenie najkratšieho riešenia a prevencia zacyklenia

Na nájdenie riešenia sa používa upravený plánovač z cvičení.

Nájdenie najkratšieho riešenia je zabezpečené iteratívnym hľadaním plánu najskôr dĺžky 1, ak sa nenájde, hľadá sa dĺžky 2 a tak ďalej.

Priamym dôsledkom iteratívneho hľadania je vznik nekonečného cyklu. Tomu sa vyhneme uložením si maximálnej dĺžky cesty počas hľadania do dynamického predikátu. Ak maximálna dĺžka nedosiahla dĺžku hľadaného plánu nemá zmysel ďalej hľadať preto sa ďalšia iterácia nevykoná.

```
test(Plan, Len):-
   maxLength(ActualLen),
   ActualLen==Len,
   Len2 is Len + 1,
   test(Plan, Len2).

maxLength(0).
:- dynamic maxLength/1.

saveMaxLen(SoFar):-
   maxLength(MaxLen),
   length(SoFar,Len),
```

```
  Len>MaxLen,
  retractall(maxLength(_)),
  assertz(maxLength(Len)),
  false.
```

4.2 Optimalizácie

V tejto kapitole opisujem kroky ktoré som vyskúšal na zlepšenie optimalizácie:

1. Vykonanie Push akcií skorej môže zefektívniť rýchlosť riešenia.

2. S nádejou na efektívnejšie riešenie som vytvoril aj funkčnú breath-first teoriu, ktorá je však približne 3x pomalšia z dôvodu veľkého množstva operácii so zoznamami a pamäťových nárokov ukladania stavov a ciest vo fronte. Aj tak mi príde zaujímavá a je priložená v súbore bfs.pl

3. Tabling, alebo dynamické ukladanie predikátu subset/2 vie zrýchliť beh programu, ale pri dlhých riešeniach vie zaplniť celú vyčlenenú pamäť.

4. Stav svetu som rozdelil do 2d zoznamu ktorý osobitne obsahuje mapu políčok, Sokobanovu pozíciu, pozície boxov, pozície prázdnych políčok. Podobne som rozdelil preconditions a effects v akciách. Následne namiesto hľadania podmnožiny stavu a preconditions akcie sa hľadá nad adekvátnymi podmnožinami stavu. Podobne pre následne, odoberanie (subtract/3) a pridávanie (append/3) efektov akcie.

```
solve([_,_,B,_],Goal,Plan,_,Plan):-subset(Goal,B),!.
solve([Map,SS,BB,EE],Goal,SoFar,Len,Plan):-
    \+saveMaxLen(SoFar),
    \+length(SoFar,Len),
    opn(Op,[MapP,S,B,E],[As,Ae,Ab],[Ds,De,Db]),
    subset(MapP,Map),
    SS==S,
    subset(B,BB),
    subset([E],EE),
    \+ member(Op,SoFar),
    subtract(BB,Db,Boxes),
    subtract(EE,[De],Empty),
    append(Boxes,Ab,Boxes2),
    append(SoFar,[Op],SoFar2),
    append(Empty,[Ae],Empty2),
    solve([Map,As,Boxes2,Empty2],Goal,SoFar2,Len,Plan).
```

Optimalizácie som testoval na mape level2 resp. map2.txt .

Po rozdelení stavu sveta som 26 sekundové riešenie zrýchlil na 7s, a po tablingu predikátu subset/2 som riešenie zrýchlil na 4.5 sekundy, čo je skoro 6x rýchlejšie.

Pri tomto nastavení solve/5 hráč nevie spraviť 2x tú istú akciu kvôli `\+ member(Op,SoFar)` . To však môžeme za komentovať čo však zvyšuje výpočtovú zložitosť riešenia a prestane fungovať vyššie spomínaná prevencia zacyklenia.

5. Implementácia

Program som implementoval v mne blízkom programovacom jazyku JavaScript. Po spustení servera je na adrese localhost:9000 dostupné grafické rozhranie, kde sa dá editovať mapa, alebo vybrať z prednastavených riešení. Taktiež sa dole dá vybrať súbor s mapou. Po stlačení tlačidla solve sa pošle request s počiatočným a cieľovým stavom na server, ktorý ich doplní do teórie a skúsi vyriešiť. Po vyriešení sa odošle odpoveď s cestou ktorá sa u klienta vizualizuje, alebo vypíše error.

6. Záver a zhodnotenie riešenia

V projekte som sa zameral čisto na riešenie problému pomocou jazyku Prolog. Ukázaný princíp z cvičení som obohatil o iteratívne hľadanie a prevenciu jeho zacyklenia. Naučil som sa nové možnosti jazyku ako je tabling alebo dynamický predikát. Breath-first teoria sa síce neukázal efektívnejšia, ale jej programovanie ma naučilo zas niečo nové.

Samotné riešenie však kvôli vysokej zložitosti nie je dostatočne efektívne na väčšie mapy z bonusových máp.