Sokoban solver

Plánovací problém

Milan Cifra

20. november 2018

1 Úvod

Práca sa zaoberá tvorbou programu určenému na vyriešenie hry Sokoban pomocou logického plánovania a SAT solvera. Tento dokument oboznamuje čitateľa s pravidlami a cieľom logickej hry Sokoban. Detailne opisuje, akým spôsobom som sformalizoval hru do predikátov prvorádovej logiky a akým spôsobom som implementoval zakódovanie teórie do formátu CNF, ktorú SAT solver (Minisat) vyrieši a nájde riešenie hry.

2 Pravidlá a cieľ hry

Logická hra *Sokoban* sa štandardne hrá na hracej ploche veľkosti $N \times M$. Hráč ovláda postavičku pomocou ktorej posúva boxy. 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 alebo je označené ako cieľ. Nesmie vstúpiť na políčko kde je stena alebo nejaký 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. Mapa musí byť vytvorená tak, aby počet boxov zodpovedal počtu cieľových políčok.

3 Formalizácia problému

V tejto sekcii opisujem, akým spôsobom som sformalizoval riešenie hry ako plánovací problém v STRIPS notácii a ako som ho zakódoval do CNF tvaru.

3.1 Hracia plocha

Mapa je rozdelená na políčka, kde každé políčko je identifikované súradnicami začínajúcimi [0,0] v ľavom hornom rohu. Súradnice sú ako indexy dvojrozmerného poľa, teda políčko so súradnicami [1,0] je v druhom riadku a v prvom stĺpci mapy.

Predpokladajme, že označenie XY určuje súradnice nejakého políčka na mape. Pozíciu hráča na mape určuje predikát *player(XY)*. Pozíciu krabice s číslom *boxID* určuje predikát *at(boxID, XY)*. Parameter *boxID* je celé číslo väčšie ako nula. Prázdne políčko určuje predikát *empty(XY)*. Políčko, ktoré je na mape označené ako cieľ, bude označovať predikát *target(XY)*.

3.2 Počiatočný stav

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

- *player(XY, 0)* - hráč je na pozícii *XY*
- *-player(XY, 0)* - hráč nie je na pozícii XY
- *at(boxID, XY, 0)* - krabica boxID je na pozícii XY
- *-at(boxID, XY, 0)* - krabica boxID nie je na pozícii XY
- *empty(XY, 0)* - pozícia XY je prázdna
- *-empty(XY, 0)* - pozícia XY nie je prázdna
- *inTarget(boxID, 0)* - krabica *boxID* je v cieli
- *-inTarget(boxID, 0)* - krabica *boxID* nie je v cieli

3.3 Cieľ

Hra končí, keď sa postavičke podarí dostať *všetky* krabice do cieľa. Formálne to zapíšem ako *inTarget(boxID, k)* pre všetky krabice na mape, kde *k* je číslo koncového stavu.

3.4 Akcie

Zadefinoval som celkom tri akcie, posunutie hráča, posunutie krabice a posunutie krabice do cieľa.
Akcia *move(fromXY, toXY)* presúva hráča z pozície *fromXY* na pozíciu *toXY*. Musí platiť, aby bol hráč pred vykonaním akcie na pozícii *fromXY*, a aby bolo políčko na pozícii *toXY* prázdne. Po vykonaní akcie bude pozícia hráča *toXY* a políčko *fromXY* bude prázdne. V STRIPS notácii sa to zapíše nasledovne:

$$move(fromXY,\ toXY) = (p^+,\ p^-,\ e^+,\ e^-)$$

$$p^+ = \{player(fromXY),\ empty(toXY)\} \quad p^- = \{\}$$

$$e^+ = \{player(toXY),\ empty(fromXY)\} \quad e^- = \{\}$$

Po prepísaní do CNF tvaru (nech *i* je číslo kroku):

$$move(fromXY, toXY, i) \Rightarrow player(fromXY, i-1) \wedge empty(toXY, i-1) \wedge player(toXY, i) \wedge empty(fromXY, i)$$

Akcia *push(boxID, playerXY, fromXY, toXY)* presúva krabicu *boxID* z pozície *fromXY* na pozíciu *toXY*. Pozíciu hráča určuje parameter *playerXY*.

$$push(boxID,\ playerXY,\ fromXY,\ toXY) = (p^+, p^-,\ e^+,\ e^-)$$

$$p^+ = \{empty(toXY),\ player(playerXY),\ at(boxID,\ fromXY)\} \quad p^- = \{target(toXY)\}$$

$e^+ = \{player(fromXY),\ at(boxID,\ toXY),\ empty(playerXY)\}$ $e^- = \{inTarget(boxID)\}$

Po prepísaní do CNF tvaru (nech *i* je číslo kroku):

$push(boxID, playerXY, fromXY, toXY, i) \Rightarrow empty(toXY, i-1) \wedge player(playerXY, i-1) \wedge$
$at(boxID, fromXY, i-1) \wedge \neg target(toXY, i-1) \wedge player(fromXY, i) \wedge at(boxID, toXY, i) \wedge$
$empty(playerXY, i) \wedge \neg inTarget(boxID, i)$

Akcia *push_t(boxID, playerXY, fromXY, toXY) je podobná ako push,* s rozdielom, že krabica bude po vykonaní v cieli. V CNF sa to zapíše nasledovne (nech *i* je číslo kroku):

$push\ t(boxID, playerXY, fromXY, toXY, i) \Rightarrow empty(toXY, i-1) \wedge player(playerXY, i-1) \wedge$
$at(boxID, fromXY, i-1) \wedge target(toXY, i-1) \wedge player(fromXY, i) \wedge at(boxID, toXY, i) \wedge$
$empty(playerXY, i) \wedge inTarget(boxID, i)$

3.5 Background knowledge

V každom kroku sa musí vykonať minimálne jedna akcia, a zároveň nemôžu sa vykonať dve naraz. Zapísal som to nasledovne:

move(0_0,0_1, i) ∨ move(0_0,0_2, i) ∨ ... push(0_0,0_1,0_2,0_3,i) ∨ ... ∨ push_t(0_0,0_1,0_2,0_3,i)

3.6 Frame problem

Ak sa vykoná akcia *move*, nezmení sa poloha všetkých krabíc. Nech sa akcia vykoná v kroku *i,* potom

- $move(fromXY,\ i) \wedge at(boxID,\ XY,\ i-1) \Rightarrow at(boxID,\ XY,\ i)$
- $move(fromXY,\ i) \wedge \neg at(boxID,\ XY,\ i-1) \Rightarrow \neg at(boxID,\ XY,\ i)$

Ak sa vykoná akcia *move*, krabice ktoré boli, resp. neboli v cieli, tak zostanú, resp. nebudú v cieli. Nech sa akcia vykoná v kroku *i*, potom

- $move(fromXY,\ i) \wedge inTarget(boxID,\ i-1) \Rightarrow inTarget(boxID,\ i)$
- $move(fromXY,\ i) \wedge \neg inTarget(boxID,\ i-1) \Rightarrow \neg inTarget(boxID,\ i)$

Ak sa vykoná akcia *push* (alebo *push_t*), ostatné boxy nezmenia svoju polohu. Nech sa akcia vykoná v kroku *i* a $boxID \neq boxID2$, potom (analogiky pre *push_t*)

- $push(boxID,\ playerXY,\ fromXY,\ toXY,\ i) \wedge at(boxID2,\ XY,\ i-1) \Rightarrow at(boxID2,\ XY,\ i)$
- $push(boxID,\ playerXY,\ fromXY,\ toXY,\ i) \wedge \neg at(boxID2,\ XY,\ i-1) \Rightarrow \neg at(boxID2,\ XY,\ i)$

Ak sa vykoná akcia *push* (alebo *push_t*), ostatné boxy ktoré boli, resp. neboli v cieli, tak zostanú, resp. nebudú v cieli. Nech sa akcia vykoná v kroku *i* a $boxID \neq boxID2$, potom (analogiky pre *push_t*)

- $push(boxID, playerXY, fromXY, toXY,\ i) \wedge inTarget(boxID2, i-1) \Rightarrow inTarget(boxID2, i)$
- $push(boxID, playerXY, fromXY, toXY,\ i) \wedge \neg inTarget(boxID2, i-1) \Rightarrow \neg inTarget(boxID2, i)$

4 Implementácia

Program som implementoval v programovacom jazyku Python verzie 3.6. Jadro celého programu sa nachádza v súbore `src/SokobanSolver.py`. Konštruktor triedy `SokobanSolver` prijíma názov súboru v ktorom je zakreslená mapa. Na zapisovanie teórie do súboru využíva knižnicu `src/lib/theoryWriter.py`, ktorá zapisuje v CNF syntaxi klauzuly do súboru `src/cnf.txt`.

Po zapísaní teórie do CNF program pomocou knižnice `src/lib/text2dimacs.py` teóriu preloží do formátu DIMACS a uloží ju do súboru `src/dimacs.txt`. Vytvorí aj pomocný súbor `src/variables.txt`, v ktorom sú uložené hodnoty premenných. Následne program spustí Minisat a výstupom je súbor `src/out.txt`.

Obslužný skript v súbore `src/sokoban.py` prijíma od používateľa meno súboru v ktorom je zakreslená mapa. Toto meno pošle konštruktoru triedy `SokobanSolver` a spustí riešenie.

Pre spustenie programu treba zadať do konzoly nasledovný príkaz (musíme byť v priečinku `src`):

```
python sokoban.py maps/map0.txt
```

Po spustení program vypíše pre každú iteráciu čo práve robí. Na konci (ak nájde riešenie) vypíše zoznam akcií (uvedených v stati 3.4) ktoré treba vykonať, aby sa hra skončila. Voliteľne je možné definovať aj druhý argument (za názvom súboru mapy) - limit iterácií. Štandardne má hodnotu 20.

4.1 Mapa

Mapu hry program načítava zo súboru, ktorá musí byť zapísaná v správnej syntaxi. Znak `#` znamená stenu, znak `S` hráča, znak `T` cieľové miesto, znak `B` krabicu a znak `X` symbolizuje hráča postaveného na cieľovom mieste. Prázdne miesto sa označuje medzerou. Žiadne iné znaky nie sú v súbore povolené.

5 Záver

Výsledkom tejto práce je funkčný program, ktorý rieši hru Sokoban pomocou SAT solveru Minisat, ktorému iteratívne generuje vstupy vo formáte CNF. Na zakódovanie problému využíva STRIPS notáciu, ktorú som detailne popísal v kapitole 3. Program bol implementovaný v jazyku Python.

Program nie je dokonalý, pri väčších mapách a väčšom počte krabíc mu vyriešenie trvá dlhší čas. Dôvodom tohto nedostatku je, že pri každej novej iterácii zapisuje všetky predchádzajúce iterácie od začiatku.