Automated Reasoning

Mattia Masci, matr.282311

Assunzioni

- Il problema affrontato in questo progetto prevede di lavorare su una griglia, quando si incontra quindi la tupla (X, Y) (con X e Y sostituiti da due numeri interi), con X si vuole indicare la riga e con Y la colonna in cui si trova l'agente. Inoltre, le righe sono numerate in ordine crescente dal basso verso l'alto, le colonne, invece, da sinistra verso destra.
- Si assume che l'agente esegua la prima mossa mentre questo si trovi nella cella di partenza specificata e che non appena entri nella cella d'arrivo il task sia completato.
- Quando nel file 'prog_inst.txt' si afferma wall(X, Y), automaticamente la cella (X, Y) della griglia diventa inutilizzabile.
- Quando nel file 'prog_inst.txt' si afferma sliding_door(X, Y), ovvero la presenza di una porta scorrevole nella cella (X, Y), ciò non significa che la cella menzionata non sia utilizzabile per intero, piuttosto che la porta impedisca il passaggio dell'agente dalla cella (X, Y) alla cella $(X - 1, Y)$ e viceversa, chiudendo quindi orizzontalmente la parte inferiore della cella.
- Si assume che i dati in input riportati nel file 'prog_inst.txt' permettano di creare un'istanza 'legale' del problema e che siano coerenti con quanto specificato in queste pagine.

Codice

Ho diviso i dati della codifica ASP in due file: uno per la codifica generale del problema ('prog_enc.txt'), l'altro per descrivere l'istanza in ingresso da risolvere ('prog_inst.txt').
Nel file 'prog_enc.txt' ho associato ai vari tipi di regole e vincoli un numero che userò qui di seguito per farvi riferimento.
La costante *h* specificata nel file 'prog_inst.txt' viene utilizzata per rappresentare il massimo numero di mosse eseguibili dall'agente, qualsiasi esecuzione del programma non andrà oltre il numero prefissato riportando un 'UNSATISFIABLE' se *h* è minore del numero di mosse minimo che occorre per risolvere quella istanza.
Ho utilizzato i predicati holds/2 e occurs/2 per specificare rispettivamente qualcosa che vale oppure che accade in un determinato istante.
Una sintassi del tipo holds(on$(X, Y), T$) sta ad indicare che nell'attimo T l'agente si trova nella cella (X, Y). Invece se si incontra occurs(moves$(X, Y), T$) ciò significa che l'agente si è mosso dalla cella corrente fino alla cella (X, Y) nel momento T, e quindi nell'istante $T + 1$ varrà holds(on$(X, Y), T + 1$).

1) La regola crea una griglia quadrata dove ogni lato contiene S celle, S è specificato nel file 'prog_inst.txt';
2) La regola rende vero per ogni sliding_door(X, Y), l'atomo sliding_odd$(X, Y + 1)$ associato che viene considerato negli istanti di tempo dispari in cui la porta rende inaccessibile la cella appena a destra di quella indicata da (X, Y);
3) La regola afferma che per ogni valore di $T = 0, \dots, h - 1$ e ogni atomo holds(on$(X, Y), T$) vero al tempo T, l'agente può scegliere un'unica mossa da fare tra cui muoversi da uno a tre passi in orizzontale o verticale, oppure aspettare. È importante notare che questo non va contro il vincolo della specifica secondo il quale l'agente non può scegliere di stare

fermo perché per come è stato scritto il codice la suddetta scelta non ricadrà mai su una *wait,* a meno che l'agente non abbia raggiunto la cella target, e quindi il task sarà da considerarsi terminato;

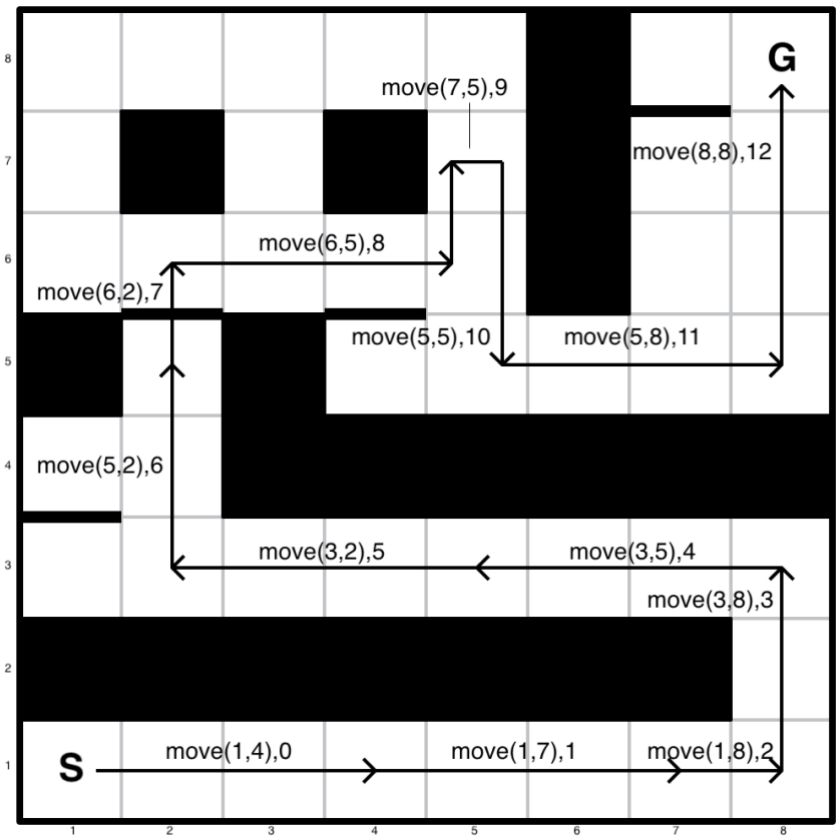
4) L'insieme di regole indica l'effetto delle mosse compiute: la prima è già stata trattata precedentemente, la seconda regola afferma che quando si esegue una *wait* nel momento T, per tutti gli attimi successivi fino allo scadere del tempo si continuerà ad eseguire questa azione. Naturalmente, la struttura del codice fa sì che l'agente deciderà di eseguire questa azione solo nel momento in cui avrà raggiunto il suo obiettivo;
5) Questo insieme di vincoli assicura che l'agente non eseguirà mai movimenti resi illegali dalla presenza di porte scorrevoli. Quindi, per esempio, i primi due impediscono all'agente di scegliere mosse che prevedano che questo salga di qualche riga rispetto alla sua posizione precedente e che in una tra le celle toccate ci sia una porta scorrevole che sbarri la strada. Per esempio nella prima:
$$:- occurs(move(X,Y),T), holds(on(R,Y),T), X > R, Q = X - R, T \setminus 2 = 0, sliding(R + (1..Q), Y).,$$
$X > R$ controlla che la riga in cui ci si vuole muovere sia maggiore rispetto a quella in cui si è, si ripone poi in Q la differenza tra le due posizioni (che in base alle specifiche può essere al massimo uguale a 3), dopodiché $T \setminus 2 == 0$ controlla che l'istante di tempo in cui si esegue l'azione sia pari, in quel caso, se esiste una porta scorrevole in almeno una delle celle $(R + (1..Q), Y) = (R + 1, Y), (R + 2, Y), (R + 3, Y)$, supponendo che Q sia uguale a 3, allora la mossa non può essere compiuta.
I vincoli seguenti si occupano di casi simmetrici, per esempio per istanti di tempo pari, oppure quando l'agente non sta salendo ma scendendo;
6) Questo insieme di vincoli assicura che l'agente non eseguirà mai movimenti resi illegali dalla presenza di muri. La sintassi è essenzialmente identica a quelli precedenti eccetto per il fatto di non considerare il tempo in cui vengono eseguite le mosse, questo perché la struttura murale rimane invariata nel tempo, a differenza delle porte scorrevoli;
7) Il vincolo assicura che l'agente non scelga mai una mossa che lo porti all'infuori della griglia costruita;
8) Questo insieme di regole descrive la relazione che intercorre tra il predicato init/1, utilizzato per prendere il dato in input che specifica la posizione iniziale dell'agente e il predicato holds/2 che lo posiziona nella griglia creata in modo da poter gestire la sua posizione all'interno del programma. Il secondo vincolo invece afferma che se esiste un goal, questo verrà raggiunto in un dato momento prima dello scadere del tempo;
9) Questa regola si occupa di minimizzare la lunghezza del cammino seguito dall'agente massimizzando il numero di volte in cui questo decide di stare fermo. Come già detto, visto il vincolo in 8) e la regola in 4), prima di eseguire moves($wait, T$) l'agente si preoccuperà di aver raggiunto il proprio obiettivo.

Di seguito riporto l'ottimo trovato dal solver per un'istanza abbastanza complicata, che comunque clingo impiega circa 4 secondi per risolvere. Come si può notare, l'insieme stabile ottimo indicato corrisponde all'ottava risposta, questo perché nell'esempio ho impostato la costante h uguale a 20 e il numero di mosse minimo per risolvere il problema per la data istanza è 13, quindi il solver riporta anche insiemi stabili per numero di mosse uguale a $14, 15, 16$…

```
Answer: 8
holds(on(1,1),0) holds(on(1,4),1) occurs(move(1,4),0) holds(on(1,7),2) occurs(mo
ve(1,7),1) holds(on(1,8),3) occurs(move(1,8),2) holds(on(3,8),4) occurs(move(3,8
),3) holds(on(3,5),5) occurs(move(3,5),4) holds(on(3,2),6) occurs(move(3,2),5) h
olds(on(5,2),7) occurs(move(5,2),6) holds(on(6,2),8) occurs(move(6,2),7) holds(o
n(6,5),9) occurs(move(6,5),8) holds(on(7,5),10) occurs(move(7,5),9) holds(on(5,5
),11) occurs(move(5,5),10) holds(on(5,8),12) occurs(move(5,8),11) holds(on(8,8),
13) occurs(move(8,8),12) occurs(wait,20) occurs(wait,19) occurs(wait,18) occurs(
wait,17) occurs(wait,16) occurs(wait,15) occurs(wait,14) occurs(wait,13)
Optimization: -8
OPTIMUM FOUND

Models       : 8
  Optimum    : yes
Optimization : -8
Calls        : 1
Time         : 4.567s (Solving: 0.11s 1st Model: 0.09s Unsat: 0.00s)
CPU Time     : 4.557s
```

I dati in input relativi a questo esempio sono riportati nel file 'prog_inst.txt' e creano la griglia sottostante. Le frecce indicano le mosse che l'agente sceglie nei vari istanti di tempo fino ad arrivare nella cella target.



Le linee orizzontali alla base delle celle $(4,1)$, $(6,2)$, $(6,4)$ e $(8,7)$ rappresentano porte scorrevoli. La loro posizione nella foto si riferisce all'attimo di partenza, e di conseguenza ai momenti pari. Nei momenti dispari possiamo immaginare ognuna di queste linee traslata di una cella a destra.

La codifica presente nei file è il risultato di diversi esperimenti provati nel tentativo di ottenere la miglior complessità computazionale possibile per il programma. Nel decidere quale fosse il modo

migliore di esprimere determinati concetti, oltre a calcolare quale poteva essere il peso di ogni riga in termini di quantità di regole derivate dopo il grounding del programma, mi sono basato sul tempo impiegato dal solver per risolvere istanze uguali con diverse codifiche. Un esempio è l'utilizzo della regola:

$$:- not\ holds(C, \_), goal(C).,$$

al posto di:

$$:- not\ final(C), goal(C).$$
$$final(C) :- holds(C, \_), goal(C).,$$

che potrebbe sembrare meno intuitiva ma diminuisce parecchio il tempo impiegato dal solver per risolvere una qualsiasi istanza. Per quella mostrata in figura, per esempio, ci vogliono 3 secondi in meno circa.