BLOCK WORLD PLANNER – A.I. 2017/2018

Valentina Cecchini (mat. 255596) – Stefano Valentini (mat. 254825)

INTRODUZIONE

Lo scopo del progetto è quello di produrre un applicativo che sia in grado di interpretare asserzioni, comandi e query relative al problema del block world e di somministrarle ad un engine in grado di fornire risposte e planning adeguati.

IMPLEMENTAZIONE

L'implementazione è stata realizzata utilizzando Python e NLTK per la parte riguardante la traduzione da linguaggio naturale e la parte relativa al planner è stata realizzata in Answer Set Programming, in articolare attraverso il tool clingo.

Interprete

La parte relativa alla "*traduzione*" da linguaggio naturale a direttive Answer Set Programming è stata realizzata tramite il framework NLTK (Natural Language Toolkit).

L'applicativo è strutturato in script Python.

Il file *sentence.py* contiene la classe *Sentence*, questa contiene la descrizione e i metodi atti alla gestione delle frasi fornite in input dall'utente. Quando una frase viene inserita questa viene analizzata da NLTK per produrre una sequenza di token e tag che descrivono le entità rappresentate dalla frase stessa.

NLTK, data una frase, la parsa per associare tag ai token analizzati. Questi tag possono essere raggruppati in *chunks*.

La seguente immagine mostra le regole che sono utilizzate per catturare i chunk:

```
grammar = r"""
                LENGTH: {<IN><NN><CD>}
                BLOCK: {<DT><JJ><NN><NN.?><LENGTH>?}
                POS: {<VBZ><IN>}
                OBJECT: {<DT><NN>}
                ASSERTION: {<BLOCK><POS><BLOCK|OBJECT>}
                QUERY: {<VBZ><BLOCK><IN><BLOCK|OBJECT><.>}
                COMMAND: {<VB.?><BLOCK><IN><BLOCK|OBJECT>}
            """
```

- **LENGTH** indica un chunk formato da una preposizione (<IN>), un nome (<NN>) ed un digit (<CD>)
- **BLOCK** indica un chunk formato da un articolo determinativo (<DT>), un aggettivo (<JJ>), un nome (<NN>), un nome/nome proprio (<NN.?>) ed infine è seguito dal chunk LENGTH; questo chunk ha lo scopo di catturare segmenti di frase del tipo: "*The red block X of length 5 …"*
- **POS** indica un chunk formato da un verbo in terza persona singolare presente (<VBZ>) ed una preposizione (<IN>), cattura segmenti di frase del tipo: *"… is on …"*
- **OBJECT** indica un chunk formato da un articolo determinativo (<DT>) e un nome (<NN>), il suo scopo è quello di catturare segmenti di frase come: *"… the table."*

- **ASSERTION** indica un chunk che cattura intere asserzioni, racchiude il chunk BLOCK seguito dal chunk POS, seguito dal chunk BLOCK oppure dal chunk OBJECT, cattura frasi del tipo: *“The red block X of length 5 is on the table”.*
- **QUERY** indica un chunk che cattura le interrogazioni alla knowledge base, è composto da un verbo alla terza persona singolare (<VBZ>), seguito dal chunk BLOCK, seguito da una preposizione (<IN>), seguito da un chunk BLOCK oppure OBJECT e che termina con un segno di punteggiatura (<.>); cattura frasi del tipo: *“Is the red block X on the table?”*
- **COMMAND** indica un chunk che cattura comandi da relizzare attraverso il planner, è composto da un verbo (<VB.?>), il chunk BLOCK, una preposizione (<IN>) ed un chunk BLOCK oppure OBJECT. Cattura frasi del tipo: *“Put the red block X on the table.”*

Una volta analizzata la frase l’applicativo restituisce una serie di informazioni estrapolate utilizzando l’operazione di *chunking* eseguita attraverso le regole appena descritte. Un esempio di analisi

```
Insert a phrase: the blue block C is on the table

text: the blue block C is on the table
tagged tokens: [('the', 'DT'), ('blue', 'JJ'), ('block', 'NN'), ('C', 'NNP'), ('is', 'VBZ'), ('on', 'IN'), ('the', 'DT'), ('table', 'NN')]
chunks: (S
  (ASSERTION
    (BLOCK the/DT blue/JJ block/NN C/NNP)
    (POS is/VBZ on/IN)
    (OBJECT the/DT table/NN)))
query: False
assertion: True
command: False
```

Nell’esempio viene analizzata la frase *“the blue block C in on the table”* e viene dedotto che si tratta di una asserzione composta da un BLOCK di nome B e di colore blu e da un OBJECT (il tavolo).

L’analisi di frasi di tipo **asserzione** portano alla produzione di *fatti* che formano la knowledge base, per esempio:

```
Insert a phrase: the yellow block D of length 4 is on the green block F of size 6

text: the yellow block D of length 4 is on the green block F of size 6
tagged tokens: [('the', 'DT'), ('yellow', 'JJ'), ('block', 'NN'), ('D', 'NNP'), ('of', 'IN'), ('length', 'NN'), ('4', 'CD'), ('is', 'VBZ'), ('on', 'IN'),
chunks: (S
  (ASSERTION
    (BLOCK
      the/DT
      yellow/JJ
      block/NN
      D/NNP
      (LENGTH of/IN length/NN 4/CD))
    (POS is/VBZ on/IN)
    (BLOCK
      the/DT
      green/JJ
      block/NN
      F/NNP
      (LENGTH of/IN size/NN 6/CD))))
query: False
assertion: True
command: False
added to planner: block(d, yellow, 4).
added to planner: block(f, green, 6).
added to planner: on(d, f, 0).
```

Il fatto *block(X, Y, Z)* indica l’esistenza di un blocck di nome X, di colore Y e di lunghezza Z.
Il fatto *on(A, B, C)* indica che c’è un blocco di nome A sopra un blocco di nome B nello step 0 (la struttura a step verrà descritta nella sezione *planner*).

L’analisi di frasi di tipo **comando** portano all’aggiunta di clausole alla regola che definisce il goal del planner, per esempio:

```
Insert a phrase: Put the red block K on the table

text: Put the red block K on the table
tagged tokens: [('Put', 'VB'), ('the', 'DT'), ('red', 'JJ'), ('block', 'NN'), ('K', 'NNP'), ('on', 'IN'), ('the', 'DT'), ('table', 'NN')]
chunks: (S
  (COMMAND
    Put/VB
    (BLOCK the/DT red/JJ block/NN K/NNP)
    on/IN
    (OBJECT the/DT table/NN)))
query: False
assertion: False
command: True
added to goal: on(k, table, T),
```

Dopo aver aggiunto il segmento prodotto dall'analisi della frase precedente abbiamo che l'intera regola che definisce il goal del planner è: *""goal(T) :- time(T), on(k, table, T)."* la quale indica che il goal è raggiunto quando in un qualsiasi istante di tempo (o step) T abbiamo che il blocco K è sul tavolo (la regola del goal può contenere N direttive che andranno in AND).

L'analisi di frasi di tipo **query** genera *regole* che andranno ad essere valutate sulla knowledge base prodotta, le query non fanno uso del planner e quindi in questo caso non viene generata la regola di goal.
Esempio:

```
Insert a phrase: is the red block A on the table?

text: is the red block A on the table?
tagged tokens: [('is', 'VBZ'), ('the', 'DT'), ('red', 'JJ'), ('block', 'NN'), ('A', 'NNP'), ('on', 'IN'), ('the', 'DT'), ('table', 'NN'), ('?', '.')]
chunks: (S
  (QUERY
    is/VBZ
    (BLOCK the/DT red/JJ block/NN A/NNP)
    on/IN
    (OBJECT the/DT table/NN)
    ?/.))
query: True
assertion: False
command: False
```

Le regole di query generate dall'esempio sono: "*query(yes) :- on(a, table, _). query(no) :- not query(yes).",* il che indica che la query "restituisce" *yes* se nella knowledge base esiste un fatto "on(a, table, _); il "_" serve ad indicare che qualsiasi cosa è accettata nella posizione dedicata allo step.

Il file *main.py* contiene la funzione che si occupa di reperire le frasi inserite dall'utente e di avviare le operazioni di analisi e traduzione contenute nella classe *Sentence*.
Ogni *fatto/regola* viene inserito in un file che andrà ad includere l'engine fisso del planner, una volta terminato l'inserimento delle frasi viene lanciato il comando "*clingo nome_file.lp"* il quale legge i fatti, le regole e il planning principale e lo risolve, restituendo il planning ottimo (ovvero, il planning che comporta il minor numero di mosse) se è stata inserita una serie di comandi, oppure *yes* o *no* nel caso in cui ci sia una query.

Planner

Si è deciso di implementare il planner tramite l'utilizzo del paradigma logico-dichiarativo dell'Answer Set Programming.
La decisione è dovuta al fatto che tale paradigma è in grado di fornire diverse soluzioni, tra cui una ottima (calcolata sul minimo numero di mosse che è possibile fare), su cui eventualmente poter fare machine learning.

Per l'implementazione del programma è stato utilizzato il tool clingo (combinazione del grounder gringo e del solver clasp e fornito dalla *Potsdam Answer Set Solving Collection - Potassco*).
L'engine del planning, che è possibile esaminare nel file *"_block_word_base_engine.lp"*, consiste in una serie di regole, opportunamente elaborate, che definiscono il tipo di computazione da effettuare per arrivare all'obiettivo desiderato.

Nella parte iniziale del programma engine, vengono definite le caratteristiche del tavolo su cui giaceranno i vari blocchi e il goal da raggiungere. Per quanto riguarda il tavolo, questo viene considerato come un generico oggetto di nome table, di colore white e lungo 100 (queste due ultime caratteristiche possono essere considerate accessorie); invece, il goal da raggiungere viene definito tramite linea di comando dall'utente e poi passato al resto del programma per essere computato.

Dopodiché, è possibile trovare la descrizione delle PRE-CONDIZIONI che devono verificarsi affinché sia possibile effettuare una generica mossa. Tale operatore è rappresentato dal predicato *moveop*, il quale è composto da tre termini: X, Y, T. X e Y sono due blocchi o un blocco e un tavolo e verifica che sia effettivamente possibile muovere il blocco X sopra il blocco/tavolo Y nel tempo T, rispettando quelli che sono i vincoli dati sul colore e le lunghezze.

Segue la descrizione degli effetti che si verificano dopo che viene effettuata una mossa: il predicato *on(X, Y, T)* rappresenta la situazione in cui il blocco X si trova al di sopra del blocco Y nell'istante di tempo T; mentre il predicato *covered(X, T)*, rappresenta il fatto che nell'istante T il generico blocco X è stato coperto, cioè vi è stato messo sopra un altro blocco.

A seguire è possibile osservare i cosiddetti frame axioms, i quali sono utilizzati per descrivere gli effetti che seguono ad azioni, senza dover necessariamente passare per ulteriori azioni che causerebbero effetti banali.

Nel nostro caso, il predicato *on(X, Y, T*) può essere considerato effettivamente vero solo se nello step precedente si è effettivamente eseguita una mossa, situazione rappresentata dal predicato *moving(X, T)*.

Seguono i vincoli tipici del mondo dei blocchi: il primo è quello che impedisce di muovere un blocco X sopra un altro blocco Y che si sta spostando nel medesimo istante su un altro blocco Z; il secondo è quello che vieta di muovere uno stesso blocco X per più di una volta nello stesso istante di tempo; il terzo vieta che due blocchi vengano spostati sullo stesso oggetto nello stesso momento; il quarto vieta le mosse in cui un blocco che è già posizionato sul tavolo venga riposizionato sul tavolo; il quinto ed ultimo vincolo vieta che un blocco venga prima messo sopra un oggetto e poi venga di nuovo preso e messo sul tavolo.

Seguono le regole che descrivono la struttura incrementale degli istanti di tempo e il riconoscimento dei blocchi come generici oggetti.

Nella parte finale si trovano le regole che verificano la risposta alle query poste e lo statement di minimizzazione utilizzato per trovare il numero più breve di passi da dover effettuare per poter raggiungere l'obiettivo. In particolare, per rispondere alle query si è utilizzato un predicato query che come argomento ha la costante "*yes*" se il fatto richiesto è presente ed è vero nella KB, "*no*" altrimenti. Invece, lo statement di minimizzazione viene eseguito sul numero di mosse che è possibile fare e minimizza il numero totale di mosse da eseguire (contate tramite il predicato *number_of_moveop(M)*).

Esempi di esecuzione

Esempio di **planning** (video dimostrativo - https://youtu.be/Wh-pbzR0Izs):

Istanza

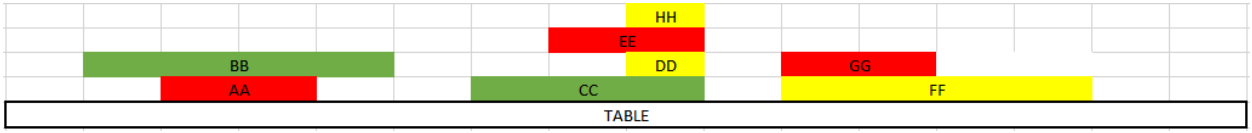
the red block AA of length 2 is on the table
the green block BB of length 4 is on the red block AA
the green block CC of length 3 is on the table
the yellow block DD of length 1 is on the green block CC
the red block EE of length 2 is on the yellow block DD
the yellow block HH of length 1 is on the red block EE
the yellow block FF of length 4 is on the table
the red block GG of length 2 is on the yellow block FF

put the red block AA on the table
put the green block CC on the table
put the yellow block FF on the table
put the red block EE on the green block CC
put the yellow block HH on the red block EE
put the green block BB on the yellow block FF
put the red block GG on the green block BB
put the yellow block DD on the red block GG

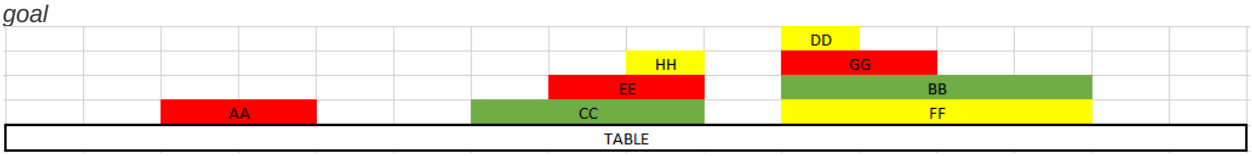
done

Rappresentazione grafica:

istanza



goal



Nel video dimostrativo viene evidenziata la produzione di diversi answer set, tra cui il ***"peggiore"*** secondo i termini di ottimizzazione (minor numero di mosse):

moveop(gg,table,0)
moveop(hh,bb,0)
moveop(ee,table,1)
moveop(hh,gg,1)
moveop(dd,table,2)
moveop(bb,table,3)
moveop(ee,cc,8)
moveop(hh,ee,9)
moveop(bb,ff,13)
moveop(dd,gg,15)
moveop(gg,bb,14)
goal

Optimization: 11

Answer set ***"migliore"***:

goal
moveop(gg,table,0)
moveop(bb,ff,1)
moveop(hh,table,1)
moveop(ee,table,2)
moveop(gg,bb,2)
moveop(dd,gg,3)
moveop(ee,cc,4)
moveop(hh,ee,5)

Optimization: 8

Lasciando girare l'applicativo si possono ottenere answer set ancora migliori oppure la conferma che l'ultimo trovato è effettivamente l'ottimo.

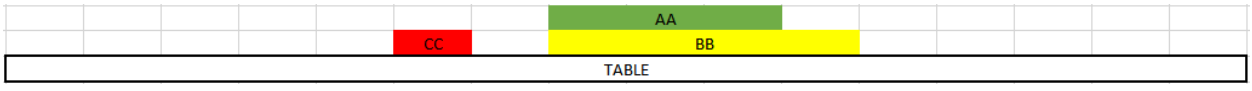
Esempio di **query** (video dimostrativo - https://youtu.be/8Km2TZIdaKU):

Istanza:

the red block CC is on the table
the yellow block BB of size 4 is on the table
the green block AA of size 3 is on the yellow block BB

is the green block AA on the yellow block BB?

Rappresentazione grafica:



Output:

query(yes)