INTELLIGENZA ARTIFICIALE E LABORATORIO: ANSWER SET PROGRAMMING E CLINGO

28 settembre 2017

Andrea Forgione: 809240
Arthur Capozzi: 802586
Leonardo Zanchi: 800935

Indice

Introduzione

In questa relazione saranno presentati gli esercizi riguardandi la parte di *Answer Set Programming* e Clingo. I domini scelti sono due: le cinque case e il trasporto aereo di merci. Il lavoro sarà diviso quindi in due capitoli, uno per ciascun problema considerato. La versione di Clingo utilizzata è *Clingo 5.2.1 WIN64*.

Capitolo 1

Cinque case

Il dominio delle cinque case ha come obiettivo quello di trovare chi tra le persone descritte ha come animale preferito la zebra. La soluzione si trova all'interno del file *house_problem.lp*. La struttura del programma segue il design *Generate-Define-Test*, e quindi l'implementazione si articola in tre sezioni differenti, ognuna con le sue caratteristiche:

- *Generate*;
- *Define*;
- *Test*;

1.1 GENERATE

Questa sezione definisce una collezione di *answer sets* che possono essere delle potenziali soluzioni al problema. Aggregati e *choice rules* sono tipicamente contenute in questa sezione. Con le regole riportate nel listato, si definisce che le caratteristiche come colore, nazionalità, animale, bevanda e professione siano uniche, cioè, per esempio, può esistere al più una casa con un stesso colore. Queste condizioni sono espresse attraverso delle *cardinality constraint*, che, sempre in riferimento al colore, stabiliscono di assegnare mediante la funzione *colore* ad ogni fatto *case* uno (ed un solo) fatto *colori*. Con la seconda regola, invece, si afferma il contario, cioè che ad ogni fatto *colori* corrisponda un solo fatto *case*. Come già detto, il medesimo ragionamento vale anche per le altre caratteristiche.

```
% Colore
1 { colore(Casa, Colore) : colori(Colore) } 1 :- case(Casa).
1 { colore(Casa, Colore) : case(Casa) } 1 :- colori(Colore).
```

1.2 DEFINE

Questa parte esprime concetti aggiuntivi e connette la parte *generate* a quella *test*. Sono riportati tutti i fatti che l'enunciato descrive:

- le nazionalità dei personaggi (definita *nazionalitas* per conflitto e mancanza di fantasia sul nome);
- le case, definite con un intervallo, assegnando a ciascuna un numero;
- gli animali, le bevande ed il colore;

È stata inserita anche una regola per definire l'adiacenza di una casa all'altra. Per esempio *casa(1)* e *casa(2)* sono adiacenti perché $|1-2| == 1$, mentre *casa(5)* e casa(3) perché $|5-3|! = 1$. Per convenzione si è stabilito che la casa all'estrema sinistra sia identificata dal numero 1, mentre quella all'estrema destra da 5. Inoltre, la casa X si trova a sinistra della casa Y se $X < Y$, viceversa se la casa X si trova a destra della casa Y, allora $X > Y$.
È presente anche un'altra regola, *chi_ha_la_zebra(Nazionalita)*, che ha la funzione di restituire la risposta cercata. La regola ha due goal:

- *nazionalita(Casa, Nazionalita)*, che consente di connettere la nazionalità alla casa;
- *animale(Casa, zebra)*, che permette di fare il passo finale, cioè legare la casa all'animale di nostro interesse;

```
% Nazionalita'
nazionalitas(inglese;spagnolo;giapponese;italiano;norvegese).

% Casa
case(1..5).

% Funzione di adiacenza
next_to(X, Y) :- case(X), case(Y), |X - Y| == 1.

chi_ha_la_zebra(Nazionalita) :- nazionalita(Casa, Nazionalita), animale(
   Casa, zebra).
```

1.3 TEST

Questo modulo consiste in regole che eliminano gli *answer sets* della parte *generate* che non possono essere delle soluzioni. L'implementazione contiene tutti i vincoli che sono espressi dall'enunciato. Per esempio, il fatto 1 è espresso dal vincolo riportato nel listato, dove si escludono dall'*answer set* le soluzioni che hanno il personaggio inglese in una casa che non è rossa. Un discorso analogo può essere fatto per la regola 14, dove invece si escludono dall'*answer set* le soluzioni che prevedono che il cavallo non sia adiacente alla casa del diplomatico.

```
% 1. L'inglese vive nella casa rossa.
:- nazionalita(X, inglese), colore(X, Y), Y != rossa.
% 14. Il cavallo e' nella casa adiacente a quella del diplomatico
:- animale(X, cavallo), professione(Y, diplomatico), not next_to(X, Y).
```

1.4 OUTPUT

Questa parte finale non è propriamente definita dal *design structure Generate-Define-Test.* Ha lo scopo di restituire la soluzione trovata. La regola *casa/6* realizza una sorta di tabella, che, attraverso l'istruzione *#show casa/6*, permette di essere stampata ed avere una visione di insieme su quello che si è ottenuto dall'esecuzione. L'istruzione *#show chi_ha_la_zebra/1* resituisce la nazionalità del possessore di questo animale utilizzando la regola definita nella sezione *define.* La soluzione restituita da Clingo dopo l'esecuzione di *clingo 0 house_problem.lp* è riportata nel listato.

```
chi_ha_la_zebra(giapponese)
casa(1,gialla,norvegese,volpe,altro,diplomatico)
casa(2,blu,italiano,cavallo,te,dottore)
casa(3,rossa,inglese,lumache,latte,scultore)
casa(4,bianca,spagnolo,cane,succo_di_frutta,violinista)
casa(5,verde,giapponese,zebra,caffe,pittore)
SATISFIABLE
```

Capitolo 2

Trasporto aereo di merci

Questo capitolo prende in esame il problema del trasporto aereo di merci, tratto dal *Russell e Norvig*, capitolo 10.1. La soluzione si trova all'interno del file *plane_problem.lp*.
A differenza della precedente soluzione, questa segue una differente struttura, divindendosi in tre parti: la prima è introdotta dalla direttiva *#program base*, la seconda da *#program step(t)* e la terza da *#program check(t)*. *Clingo*, infatti, consente la possibilità di dividere le regole di input in subprograms attraverso la direttiva *#program*. L'utilizzo di '*t*' è di fondamentale importanza, infatti essa mantiene lo stato del programma, consentendo una esecuzione incrementale del problema. Per poterne fare uso è necessario dichiarare la volontà di utilizzarlo e questo è possibile inserendo la libreria *incmode* attraverso l'istruzione *#include <incmode>.*.

2.1 SEZIONE BASE

Questa sezione è una parte dedicata del *subprogram* con una lista dei parametri vuota. Contiene essenzialmente una lista di fatti che definiscono il dominio del problema. Sono riportati infatti l'elenco degli oggetti presenti (cargo, plane e airport), lo stato iniziale e lo stato goal. È presente, inoltre, il caso base della regola di inerzia che permette di risolvere il problema. Si è deciso di utilizzare il predicato *holds(F, 0)* che racchiude il fluente *F*, che può essere sia '*in*' che '*at*', così da avere una maggiore generalità, evitando di specificare le singole regole di inerzia per ciascuno.

2.2 SEZIONE STEP

Questa sezione è la più corposa e contiene al suo interno le regole che consentono di risolvere il problema descritto dal dominio. Per comodità si può dividere questa sezione in quattro parti.

Choice rules La prima parte consiste in una singola *choice rule* che consente di speficare che è possibile una sola azione ogni step.

```
1 { load(C, P, A, t): cargo(C), plane(P), airport(A); unload(C, P, A, t):
    cargo(C), plane(P), airport(A); fly(P, FROM, TO, t): plane(P),
   airport(FROM), airport(TO), FROM != TO} 1.
```

Inerzia La seconda parte contiene le regole di inerzia del predicato *holds* in cui si specifica che il fluente non cambia il suo stato se non è esplicamente rimosso. La prima regola, per esempio, specifica che per avere il fluente *F* nello stato *t* del predicato *holds*, si aveva il fluente nello stato *t - 1* e che c'è assenza di informazione sulla negazione del predicato nello stato *t*.

```
holds(F, t) :- holds(F, t - 1), not -holds(F, t).
-holds(F, t) :- -holds(F, t - 1), not holds(F, t).
```

Effetti La terza parte contiene gli effetti delle azioni di *load*, *unload* e *fly*. Nel listato sono riportati gli effetti che l'azione *fly* comporta: volando dall'aereoporto *FROM* all'aereoporto *TO* nello stato *t*, l'aereoplano *P* si troverà nell'aereoporto *TO* e non nell'aereoporto *FROM* nello stato *t*.

```
holds(at(P, TO), t) :- fly(P, FROM, TO, t).
-holds(at(P, FROM), t) :- fly(P, FROM, TO, t).
```

Vincoli L'ultima parte contiene le precondioni sotto le quali le azioni sono possibili. Per esempio, nel listato riportato si esclude l'operazione di *load* nello stato *t* se avevo compiuto la stessa operazione sullo stesso cargo nello stato precedente *t - 1*, qualsiasi sia l'aereoplano.

```
:- load(C, P, A, t), holds(in(C, _), t - 1).
```

2.3 CHECK

Questa terza sezione verifica in ogni stato se si è raggiunto lo stato goal.

```
:- query(t), goal(F), not holds(F, t).
```

2.4 OUTPUT

La soluzione proposta da *Clingo* a questo problema è riportata nel listato. Quella inserita è la migliore che riesce a trovare sotto i vincoli espressi precedentemente. In totale *Clingo* invidivuda 22 differenti modelli di *answer set*.

```
load(c1,p1,sfo,1)
fly(p1,sfo,jfk,2)
load(c2,p2,jfk,3)
fly(p2,jfk,sfo,4)
unload(c2,p2,sfo,5)
unload(c1,p1,jfk,6)
SATISFIABLE
```