UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Answer Set Programming

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Prof. MARCO GAVANELLI

Answer Set Programming (ASP)

- *ASP è un nuovo insieme di linguaggi logici*
- *Maggiore dichiarattività rispetto a Prolog*
  - *L'ordine in cui si scrivono le clausole è irrilevante*
  - *L'ordine in cui si scrivono i letterali nel body delle clausole è irrilevante*
  - *Non c'è il cut (!)*
- *Non va mai in loop (se non si usano simboli di funzione)*
- *Non c'è floundering. Qualunque programma dà risultati corretti (oppure non si compila)*
- *Aritmetica bidirezionale (non come il predicato "is"), anche se non sempre efficiente*
- *Efficienza paragonabile a quella dei SAT solver (ma dipende dal tipo di problema)*
- *Valutazione bottom-up, invece di top-down come Prolog*
- *Molto orientato a risolvere CSP*

ASP: contro

- *Meno potente di Prolog a livello di linguaggio*
  - *Non c'è I/O*
  - *Non è Turing-completo*
- *Si possono risolvere problemi fino a Σ^P_2. (mentre in SAT si possono risolvere solo problemi in NP).*
- *I programmi logici trattabili con ASP devono seguire delle restrizioni sull'uso delle variabili*
- *Ci può essere uso di molta memoria (casi di out-of-memory)*

UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Answer Set Programming

Semantica Dichiarativa dei programmi positivi

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Prof. MARCO GAVANELLI

QUESTO MATERIALE DIDATTICO È PER USO PERSONALE DELLO STUDENTE ED È COPERTO DA COPYRIGHT. NE È SEVERAMENTE VIETATA LA RIPRODUZIONE O IL RIUTILIZZO ANCHE PARZIALE, AI SENSI E PER GLI EFFETTI DELLA LEGGE SUL DIRITTO D'AUTORE.

Ripasso sulla Semantica dichiarativa

- *Una semantica dichiarativa è un modo per stabilire il significato di un programma logico*
- *Il significato viene dato tramite un modello, che rappresenta l'insieme dei letterali veri in quel programma*
- *Per programmi positivi (senza negazione), il modello si calcola tramite il Modello Minimo di Herbrand*

Modello di Herbrand

- Dato un programma *P*, un modello *M* è un insieme di atomi ground (costruiti coi simboli di predicato, di costante e di funzione che sono in *P*) tali che tutte le clausole del programma sono entailed
  - Ovvero: se una clausola ha il body vero (tutti i suoi atomi sono nel modello *M*), allora anche la sua testa deve essere nel modello
- Es:

```
p(1).
q(2).
q(s(X)) :- p(X).
```

$M_0 = \emptyset$
M_1 = {p(1),q(2),q(3),q(s(1))}
M_2 = { p(1), p(2), q(2),q(s(1)),q(s(2))}
M_3 = {p(1),q(2),q(s(1))}
M_4 = {p(1),p(2),q(2),q(s(1))}

Modello minimo di Herbrand

- *Nei programmi positivi, ha senso considerare veri solo gli atomi che sono nel modello minimo di Herbrand*
- *È il modello che è minimo rispetto all'inclusione insiemistica*
- *È l'intersezione di tutti i modelli di Herbrand*

Operatore T_P

- Per calcolare il modello minimo di Herbrand, si può applicare l'operatore T_P
- Si parte da un insieme $X_0 = \emptyset$
- Partendo dall'ipotesi che gli atomi in X_i siano veri, si crea X_{i+1}, in cui si inseriscono tutte le teste delle clausole il cui body è vero
- Si continua fino al punto fisso

```
p(1).
q(2).
q(s(X)) :- p(X).
```

$X_0 = \emptyset$
$X_1 = T_P(\emptyset) = \{p(1),q(2)\}$
$X_2 = T_P^2(\emptyset) = \{p(1),q(2),q(s(1))\}$
$X_3 = T_P^3(\emptyset) = \{p(1),q(2),q(s(1))\}$

Punto fisso!

Negazione e operatore T_P

- *c :- not d.*
- *d :- c.*

$X_0 = \emptyset$
$X_1 = \{c\}$
$X_2 = \{c,d\}$
$X_3 = \{d\}$
$X_4 = \emptyset$

Loop: non esiste un punto fisso

Negazione

- Trattando di programmi con la negazione, il modello minimo di Herbrand (e l'operatore T_P) non sono più sufficienti
- Varie semantiche dichiarative sono state proposte (completamento di Clark, Completamento a 3 valori, modelli perfetti, modelli stabili, ...)

UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Answer Set Programming

Modelli Stabili

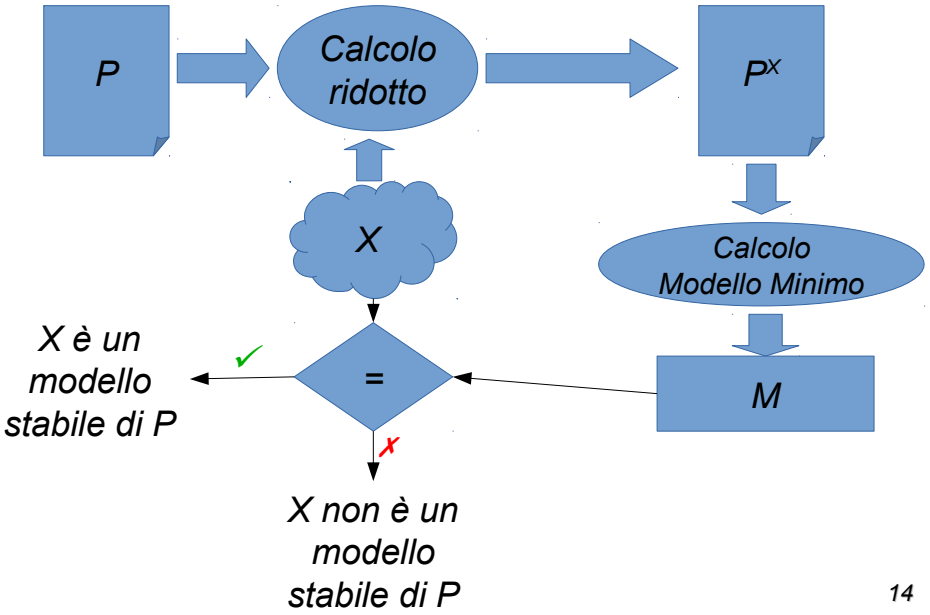
Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Prof. MARCO GAVANELLI

Modelli Stabili (Gelfond-Lifschitz)

- Per calcolare se un insieme X è un modello stabile di un programma proposizionale P:
- Si calcola il programma ridotto P^X, ottenuto come segue:
  - Se un atomo $p \in X$, allora si eliminano tutte le clausole che hanno nel body *not*(p)
  - Se $p \notin X$, allora si elimina *not*(p) da tutte le clausole che lo contengono
- Il programma ridotto P^X è un programma positivo → posso calcolare il modello minimo di Herbrand
- Se il *modello minimo di Herbrand* = X, allora X è un modello stabile

Verifica se X è un modello stabile



Es

a.

c:- ~~not b, not d.~~

~~d:- not a, not c.~~

- *X={a,c}*
- *Mod. Minimo = {a,c}*
- *X=mod minimo*
  → È un modello stabile

Es

a.

~~c:- not b, not d.~~

~~d:- not a, not c.~~

- *X={a,b,c,d}*
- *Mod. Minimo = {a}*
- *X≠mod minimo*
  → Non è un modello stabile

Trovare i modelli stabili

1)p:- p.
q:- not p.

2)p:- not p.

3)p:- not q.
q:- not p.

4)p:- not q.
q:- not p.
p:- not p.

UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Answer Set Programming

Alcune proprietà dei Modelli Stabili

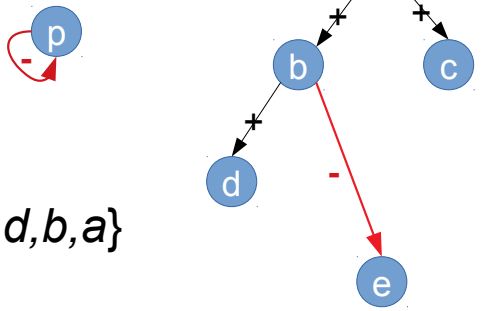
Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Prof. MARCO GAVANELLI

Valutazione top-down?

- Consideriamo il programma

a:- b,c.
c.
b:- d, not e.
d.



$M=\{c,d,b,a\}$

- E il goal *a*
- Cosa succede se aggiungiamo la clausola *p:- not p* ?

Proprietà dei modelli stabili

- *Un programma può avere zero, uno, o più modelli stabili*
- *I modelli stabili non sono relevant: possono non esistere modelli stabili perché una parte del programma è inconsistente, anche se quella parte di programma non è necessaria alla valutazione del goal*
- *Quindi per calcolare se qualcosa è vero o falso, non posso partire solo dal goal: devo comunque considerare tutto il programma*
- *Per rispondere ad una query, si parte comunque dal programma; la valutazione non è top-down come in Prolog, ma è bottom-up*

Proprietà

- ***Teorema**: Dato un programma P, ogni answer set di P è un modello minimale di P.*
- ***Teorema**: Sia P un programma e sia S un insieme di atomi. Allora S è answer set di P se e solo se*
  - *S è modello di P; e*
  - *per ogni S', se S' è modello di P^S allora $S \subseteq S'$*
- ***Teorema**: Il problema di stabilire se un programma generale ground P ammette modelli stabili è NP-completo.*

UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Answer Set Programming

Modelli Stabili – esempi

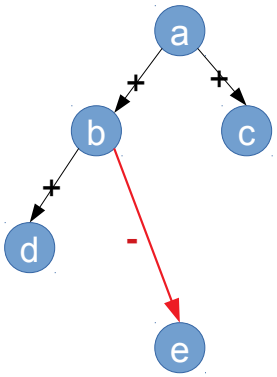
Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Prof. MARCO GAVANELLI

Valutazione top-down?

- Consideriamo il programma

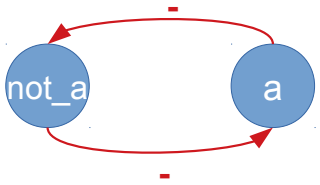
a:- b,c.
c.
b:- d, not e.
d.



- *p:- not p*

Tecniche: non determinismo

```
a :- not not_a.   not_a :- not a.
```



- *Quali sono i modelli stabili?*

{a} {not_a}

~~{a,not_a}~~ ~~∅~~

Tecniche: non determinismo

```
a :- not not_a.   not_a :- not a.
b :- not not_b.   not_b :- not b.
```

- *Quali sono i modelli stabili?*

{a,b} {not_a,b}
{a,not_b} {not_a,not_b}

Tecniche: non determinismo

```
a :- not not_a.   not_a :- not a.
b :- not not_b.   not_b :- not b.
c :- not not_c.   not_c :- not c.
```

- *Quali sono i modelli stabili?*

{a,b,c}
{a,not_b,c}
{a,b,not_c}
{a,not_b,not_c}
{not_a,b,c}
{not_a,not_b,c}
{not_a,b,not_c}
{not_a,not_b,not_c}

Tecniche: non determinismo

```
a :- not not_a.   not_a :- not a.
b :- not not_b.   not_b :- not b.
c :- not not_c.   not_c :- not c.
p :- not p.
```

- *Quali sono i modelli stabili?*

{a,b,c}
{a,not_b,c}
{a,b,not_c}
{a,not_b,not_c}
{not_a,b,c}
{not_a,not_b,c}
{not_a,b,not_c}
{not_a,not_b,not_c}

Tecniche: non determinismo

```
a :- not not_a.   not_a :- not a.
b :- not not_b.   not_b :- not b.
c :- not not_c.   not_c :- not c.
p :- a, not p.
```

- *Quali sono i modelli stabili?*

{a,b,c}
{a,not_b,c}
{a,b,not_c}
{a,not_b,not_c}
{not_a,b,c}
{not_a,not_b,c}
{not_a,b,not_c}
{not_a,not_b,not_c}

Tecniche: non determinismo

```
a :- not not_a.   not_a :- not a.
b :- not not_b.   not_b :- not b.
c :- not not_c.   not_c :- not c.
p :- a, not b, not p.
```

- *Quali sono i modelli stabili?*

{a,b,c}
{a,not_b,c}
{a,b,not_c}
{a,not_b,not_c}
{not_a,b,c}
{not_a,not_b,c}
{not_a,b,not_c}
{not_a,not_b,not_c}

UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Answer Set Programming

SAT solver in ASP

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Prof. MARCO GAVANELLI

Da SAT ad ASP

- Dato un problema SAT

  -a ∨ *b* *a* ∨ *-d* *-b* ∨ *c* ∨ *d* *d* ∨ *-c*

- Possiamo creare un predicato per ogni variabile SAT e renderlo "decisionale"

```
a :- not not_a.   not_a :- not a.
b :- not not_b.   not_b :- not b.
…
```

- Poi si eliminano gli assegnamenti vietati dalle clausole SAT
- *-a* ∨ *b* → l'unica combinazione vietata è *a* ∧ *not b*
  → lo inserisco in un loop negativo

```
nuovoNomePred :- a, not b, not nuovoNomePred.
```


ASP Solvers

- *Quindi se ho un software che calcola i modelli stabili di un programma logico, posso facilmente usarlo come SAT solver*
- *Ogni modello stabile corrisponde ad una soluzione del SAT*
- *Quindi, per il teorema di Cook, posso risolvere tutti i problemi nella classe NP*

UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Answer Set Programming

Scelte e Vincoli di Integrità

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Prof. MARCO GAVANELLI

QUESTO MATERIALE DIDATTICO È PER USO PERSONALE DELLO STUDENTE ED È COPERTO DA COPYRIGHT. NE È SEVERAMENTE VIETATA LA RIPRODUZIONE O IL RIUTILIZZO ANCHE PARZIALE, AI SENSI E PER GLI EFFETTI DELLA LEGGE SUL DIRITTO D'AUTORE.

Sintassi: scelte

- *In ASP i loop pari si possono usare per generare delle scelte.*

```
a :- not not_a.      not_a :- not a.
```

- *Sintassi semplificata:*

```
{a}.
```

- *Si possono avere delle scelte condizionate*

```
a :- b, not not_a.      not_a:- b, not a.
```

- *Significato: se* b *è vero, allora si può scegliere* a *(altrimenti, se* b *è falso, anche* a *è falso). Sintassi semplificata:*

```
{a} :- b.
```


Sintassi: Scelte

- In generale

  $\{h_1,h_2,...,h_n\}$:- b_1, b_2, … b_m, not d_1, ... not d_k.

- Significa che se $b_1...b_m$ sono veri
  e $d_1...d_k$ sono falsi
  allora si possono scegliere valori per $h_1...h_n$.

- La traduzione è

  nuovo :- b_1, b_2, … b_m, not d_1, ... not d_k.
  h_1:- *nuovo, not not_*h_1. *not_*h_1 :- *nuovo, not* h_1.
  ...
  h_n:- *nuovo, not not_*h_n. *not_*h_n :- *nuovo, not* h_n.

- Si usano quindi 2*n*+1 regole e *n*+1 nuovi atomi

Cicli dispari

- In ASP, i loop dispari eliminano dei modelli stabili
- `p :- not p.`
  elimina tutti i modelli stabili
- `p :- a, not p.`
  elimina tutti i modelli stabili in cui *a* è vero
- `p :-` a_1`,...,` a_n`, not p.`
  elimina i modelli stabili in cui tutti gli $a_1...a_n$ sono veri
- `p :-` a_1`,...,` a_n`, not` b_1`,...,not` b_m`, not p.`
  elimina i modelli stabili in cui tutti gli $a_1...a_n$ sono veri e tutti i $b_1,...,b_m$ sono falsi

Sintassi: Vincoli di integrità

`:-` a_1`,...,` a_n`, not` b_1`,...,not` b_m`.`

- Viene riscritto come:

```
nuovo :- a1,..., an, not b1,...,not bm,
         not nuovo.
```

- Si sottointende che la testa è *false*
- Se il body è vero, allora il modello non è stabile

UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Answer Set Programming

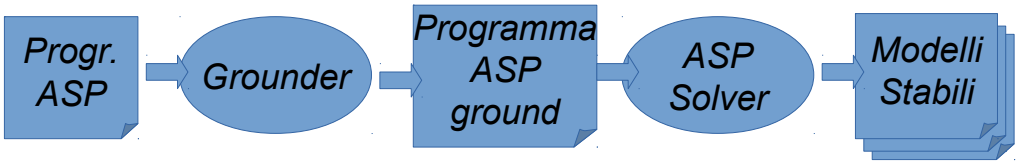
Programmi con variabili

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Prof. MARCO GAVANELLI

Programmi con variabili

- *Fino ad ora, abbiamo parlato solo di programmi proposizionali*
- *Come si possono calcolare i modelli stabili di un programma che non è ground?*
- *Per prima cosa lo si rende ground, sostituendo alle variabili le costanti che compaiono nel programma in tutti i modi possibili*



ASP solvers

- *Smodels, con grounder lparse (1995)*
- *Cmodels*
- *DLV (1997)*
- *clasp, con grounder gringo*
  - *Vince una Gold medal per la categoria UNSAT/Crafted nella SAT competition 2011*
- *WASP ...*

Es

```
arc(1,1).
arc(1,2).
edge(X,Y) :-
      arc(X,Y), arc(Y,X).
```

- *Programma ground:*

```
arc(1,1).
arc(1,2).
edge(1,1) :- arc(1,1), arc(1,1).
edge(1,2) :- arc(1,2), arc(2,1).
edge(2,1) :- arc(2,1), arc(1,2).
edge(2,2) :- arc(2,2), arc(2,2).
```

Alcune di queste clausole sono inutili!
Un grounder intelligente può generare un ***programma equivalente*** *al programma ground senza queste clausole*

Programmi con simboli di funzione

- *Se il programma contiene simboli di funzione, per produrre il programma ground si deve sostituire alle variabili i termini ground che si riescono a costruire con i simboli di funzione e i simboli di costante presenti nel programma*
- *il programma ground è infinito*
- ```
  q(f(a)).
  p(X) :- q(X).
  ```
- *Programma ground:*

```
q(f(a)).
p(a) :- q(a).
p(f(a)) :- q(f(a)).
p(f(f(a))) :- q(f(f(a))).
p(f(f(f(a)))) :- q(f(f(f(a)))).
...
```

*In questo esempio, si può generare un **programma finito equivalente** al programma ground*

Programmi con simboli di funzione e negazione

- *Aggiungendo la negazione:*
- ```
  q(f(a)).
  p(X) :- not q(X).
  ```
- *Programma ground:*

```
q(f(a)).
p(a) :- not q(a).
p(f(a)) :- not q(f(a)).
p(f(f(a))) :- not q(f(f(a))).
p(f(f(f(a)))) :- not q(f(f(f(a)))).
...
```

Con la negazione, tutte le `X` che non sono `f(a)` devono essere nel programma ground equivalente. Questo significa che anche i programmi equivalenti al programma ground hanno infinite clausole

Evitare grounding infiniti

- *Per evitare queste situazioni, gli ASP solver impongono delle restrizioni sintattiche*
- *Smodels/lparse:*
  - *il programma deve essere Strongly Range Restricted*
- *Clasp/gringo:*
  - *le regole devono essere safe*
  - *Una regola è safe se tutte le variabili che compaiono nella regola compaiono in un letterale positivo nel body.*

CLASP

- *Clasp è un ASP solver molto efficiente*
- *Utilizza un grounder chiamato Gringo*
- *Per ottenere i modelli stabili di un programma*

```
gringo programma.pl | clasp N
```

dove N è il numero di modelli stabili che si vogliono avere (0 = tutti).

- *Oppure*

```
clingo programma.pl
```

- *Per avere l'output di Gringo in formato leggibile:*

```
gringo --text programma.pl
```

UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Answer Set Programming

Esercizio: graph reachability

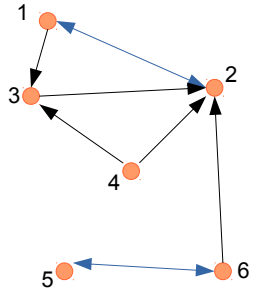
Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Prof. MARCO GAVANELLI

Raggiungibilità in un grafo diretto

- *Un grafo diretto viene rappresentato con fatti*

```
edge(1,2).
edge(2,1).
edge(1,3).
...
```



- *Si vuole sapere quali nodi sono raggiungibili a partire dal nodo* 1

Es graph coloring

- Le seguenti regole chiedono che un nodo *X* sia colorato di rosso, giallo o verde:

  color(X, red) :- nodo(X), not color(X, green), not color(X, yellow).

  color(X, green) :- nodo(X),not color(X, red), not color(X, yellow).

  color(X, yellow) :- nodo(X),not color(X, green), not color(X, red).

- Si scriva un programma ASP che calcola il seguente graph coloring:

```
near(1,2).
near(1,3).
near(2,3).
near(2,4).
near(2,5).
near(3,4).
```

```
nodo(1).
nodo(2).
nodo(3).
nodo(4).
nodo(5).
```


UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Answer Set Programming

Operatori relazionali

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Prof. MARCO GAVANELLI

QUESTO MATERIALE DIDATTICO È PER USO PERSONALE DELLO STUDENTE ED È COPERTO DA COPYRIGHT. NE È SEVERAMENTE VIETATA LA RIPRODUZIONE O IL RIUTILIZZO ANCHE PARZIALE, AI SENSI E PER GLI EFFETTI DELLA LEGGE SUL DIRITTO D'AUTORE.

Operatori relazionali

- *Gli operatori relazionali vengono interpretati direttamente dal grounder*

```
p(1). p(3). p(5).
q(X) :- p(X), X>1.
```

- *Il programma ground*

```
p(1). p(3). p(5).
q(1) :- p(1), 1>1.
q(3) :- p(3), 3>1.
q(5) :- p(5), 5>1.
```

Un grounder intelligente

- *evita di aggiungere le clausole in cui l'operatore relazionale è falso*
- *Evita di aggiungere gli operatori già veri*

Quindi nel programma finale gli operatori relazionali non compaiono

Operatori disponibili

= *(equal),*

!= *(not equal),*

< *(less than),*

<= *(less than or equal),*

> *(greater than),*

>= *(greater than or equal)*

Safety e predicati relazionali

- *Anche i programmi con predicati relazionali devono essere safe: tutte le variabili devono comparire in un atomo positivo (e gli atomi con gli operatori relazionali vengono tolti)*
- *Dire quali sono safe:*
- `p(X) :- X>0.`
- `p(X) :- X>0, X<4.`
- `p(X) :- X>0, X<4, q(X).`

Es graph coloring

- Si supponga ora che i colori disponibili vengano dati con un predicato `palette`/1. Es:
  `palette(red).` `palette(green).` `palette(yellow).`
- Si risolva il graph coloring precedente usando il predicato `palette`.
- Suggerimento:
  `{color(X,C)} :- nodo(X), palette(C).`
- però così quanti colori posso dare a un nodo? Come risolvere?
- Istanza:

```
near(1,2).
near(1,3).
near(2,3).
near(2,4).
near(2,5).
near(3,4).
```

```
nodo(1).
nodo(2).
nodo(3).
nodo(4).
nodo(5).
```

UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Answer Set Programming

Aritmetica

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Prof. MARCO GAVANELLI

Aritmetica

- *I programmi ASP possono contenere simboli aritmetici*
- *Gli operatori aritmetici vengono interpretati (diversamente da Prolog che li considera simboli di funzione)*
- *L'interpretazione è data dal grounder*

```
q(X+1) :- p(X).          p(0).
                         p(1).
```

- *Viene riscritto dal grounder in*

```
q(1) :- p(0).            p(0).
q(2) :- p(1).            p(1).
```

- *Anche il seguente programma ha lo stesso programma ground equivalente*

```
q(X) :- p(X-1).          p(0).
                         p(1).
```


Operatori disponibili

+	(addition),
-	(subtraction, unary minus)
*	(multiplication),
/ or #div	(integer division),
\ or #mod	(modulo function),
** or #pow	(exponentation),
| · | or #abs	(absolute value),
&	(bitwise AND),
?	(bitwise OR),
^	(bitwise exclusive OR)
~	(bitwise complement).

=	(equal),
!=	(not equal),
<	(less than),
<=	(less than or equal),
>	(greater than),
>=	(greater than or equal)

Aritmetica e safety

- *"Una regola è safe se tutte le variabili che compaiono nella regola compaiono in un letterale positivo nel body."*
- *In generale, gli operatori aritmetici non vengono considerati come letterali positivi ai fini della safety*
- *In alcuni casi semplici, in cui è possibile immediatamente calcolare il valore di una funzione, vengono accettati come atomi positivi*
  - *Funzioni di una sola variabile, in cui la variabile compare solo una volta, costruite con +, -, * e in cui i coefficienti della moltiplicazione non possono essere* 0
- *Es:*
  - `q(X) :- p(2*(X+1)).` *È safe*
  - `q(X) :- p(X+X).` *non è safe*

N-queens

- Si scriva un programma ASP che risolve il problema delle *N* regine.
- Si supponga che venga dato un predicato *scacchiera(K)* che è vero per tutti i $1 \leq K \leq N$
- Suggerimento:
  - Usare un predicato *queen*(*R*,*C*) che è vero se è stata posizionata una regina nella riga *R* colonna *C*

UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Answer Set Programming

Programmi disgiuntivi

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Prof. MARCO GAVANELLI

Programmi disgiuntivi

- In ASP, si possono avere clausole con teste multiple, che vengono considerate in OR

$$a_1 ; a_2 ; \ldots ; a_m :- b_1, \ldots, b_n, not\ c_1, \ldots not\ c_k$$

- Significato: *se il body è vero, allora almeno una delle teste deve essere vera.*
- In questo caso, la semantica dei modelli stabili diventa:
  - Data un'interpretazione *X*
  - Si calcola il programma ridotto P^X
  - Se *X* è un modello minimale di P^X allora è un modello stabile

Esempi di Programmi disgiuntivi

c.
a ; *b* :- *c*.

2 modelli stabili

- {*a*,*c*}
- {*b*,*c*}

c.
a ; *b* :- *c*.
a.

1 modello stabile:

- {*a*,*c*}
- {*a*,*b*,*c*} *non è stabile, perché non è minimo, in quanto esiste* $\{a,c\} \subset \{a,b,c\}$

Disgiunzione

- La disgiunzione è molto potente
- da un punto di vista teorico permette di affrontare problemi fino a $\sum^P_2$ (che contiene *NP*)
- Da un punto di vista di sintassi, permette a volte di scrivere programmi in maniera più semplice
- Es. map coloring:

color(*X*,*red*) ; *color*(*X*,*blue*) ; *color*(*X*,*green*) :- *nodo*(*X*).

UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Answer Set Programming

Aggregati

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Prof. MARCO GAVANELLI

Aggregati

- *Supponiamo di voler esprimere che* `a` *è vero se almeno 2 atomi fra* {`b,c,d,e`} *sono veri.*
- ```
  a :- b,c.
  a :- b,d.
  a :- b,e.
  a :- c,d.
  a :- c,e.
  a :- d,e.
  ```
- *Programma molto lungo. Se dovessi esprimere che è vero se almeno k atomi sono veri su un totale di n, diventerebbero* $\binom{n}{k}$

Aggregati

- Il grounder gringo accetta programmi con regole

  a :- *low* { a_1; a_2; …; a_n; *not* b_1;...;*not* b_m} *up.*
- Dove *low* e *up* sono due numeri interi (*low* ≤ *up*)
- Il significato è che *a* è vero se sono veri
  - da un minimo di *low*
  - ad un massimo di *up*

  letterali nell'insieme

  {a_1; a_2; …; a_n; *not* b_1;...;*not* b_m}
  ```

Aggregati nella testa

- Si possono anche avere aggregati nella testa.

  low {a_1;...;a_n;*not* b_1;...;*not* b_m} *up* :- c_1..c_k, *not* d_1..*not* d_f

- Dice che se il body è vero, allora devono essere veri da un minimo di *low* ad un massimo di *up* letterali dell'insieme {a_1;...;a_n;*not* b_1;...;*not* b_m}.

- Viene trasformato in

  body :- c_1..c_k, *not* d_1..*not* d_f.
  {a_1;...;a_n} :- *body*.
  testa :- *low* {a_1;...;a_n;*not* b_1;...;*not* b_m} *up*.
  :- *body*, *not testa*.

Esempio

- *Nel graph coloring, voglio che tutti i nodi siano colorati con esattamente un colore*

1 {*color(N,red); color(N,blue); color(N,green)* } 1 :-
nodo(N).

#count

- In realtà la sintassi

  low { a_1;…; a_n } *up.*

- È solo zucchero sintattico di un caso più generale

  low $\prec_1$ **#count**{ a_1; …; a_n} $\prec_2$ *up.*

- Dove i simboli $\prec_1$ e $\prec_2$ possono essere sostituiti da operatori relazionali (a default, <=)

- Si può anche omettere uno fra *low* e *up*

#min, #max, #sum

- Oltre a **#count**, esistono anche aggregati di minimo (**#min**),
  massimo (**#max**) e
  somma (**#sum**) dei valori dell'insieme

  low $\prec_1$ **#sum**{ v_1:a_1; v_2:a_2; …; v_n:a_n } $\prec_2$ *up.*

- Dove v_1 … v_n sono dei valori (o delle espressioni)

Esercizi

- *Dato il predicato* `p(1). p(3). p(5).`
- *Scrivere un predicato* `grande` *che è vero se il massimo dei valori che rendono vero* `p` *è maggiore di 2*
- *Scrivere un predicato* `somma(S)` *in cui* `S` *è la somma dei valori che rendono vero il predicato* `p`

UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Answer Set Programming

Aggregati parte 2

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Prof. MARCO GAVANELLI

Note: safety

p(S):- S < #sum { …}.

- Non è safe, in quanto *S* non compare in un predicato positivo e gli aggregati non lo sono
- Viene accettato nel caso dell'uguale:

p(S):- S = #sum { …}.

- Si noti che il programma ground tende a diventare molto grande per programmi di questo tipo: il grounder deve generare una clausola per ogni possibile valore della somma

Note

- Gli aggregati sono degli atomi (possono essere veri o falsi), *non sono delle funzioni* (che restituiscono un valore)
- *Non si possono scrivere espressioni* del tipo

*#sum{X:p(X)} + #count{Y:q(Y)}*2 < 3*

- Ogni aggregato deve essere un atomo a sè, deve sempre avere il *low*< o il <*up*

UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Answer Set Programming

Aggregati e semantica di insieme

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Prof. MARCO GAVANELLI

Semantica di insieme

- Si noti che gli aggregati hanno come argomenti degli insiemi

$$l < \#sum\ \{\ X{:}p(X)\ \} < u$$

$$l < \#count\ \{X{:}p(X)\} < u$$

- Quindi ogni elemento viene contato una sola volta
- Es **`#sum{X:p(X)}`**
  rappresenta la *"somma dell'insieme degli X tali che p(X) è vero"*

$$\sum_{x \in \{y | p(y)\}} x$$

- **`p(1).  p(3).  p(1).`**
  `a(N):- N = #sum{ X:p(X) }.`
- Ha come answer set {*p*(1), *p*(3), *a*(4)}

Semantica di insieme

- **`p(1,1).`**
  `p(3,2).`
  `p(1,3).`

  `a(N):- N = #sum{ X:p(X,ID) }.`
- *Answer set:* {*p*(1,1), *p*(3,2), *p*(1,3), a(4)}
- *Infatti* **`#sum`**{ *X*:*p*(*X*,*ID*) } = **`#sum`** {1;3;1} = **`#sum`** {1;3} = 4

$$\sum_{x \in \{1,3,1\}} x = \sum_{x \in \{1,3\}} x = 4$$

Semantica di insieme

- In realtà la sintassi per gli aggregati permette di avere delle tuple

$$l < \#sum\{\ t_1{:}a_1\ ;\ t_2{:}a_2\ ;\ \ldots\} < u$$

- Dove ciascuno dei t_i è una sequenza di valori $v_{i,1}$ $v_{i,2}$, ..., $v_{i,n}$, ovvero

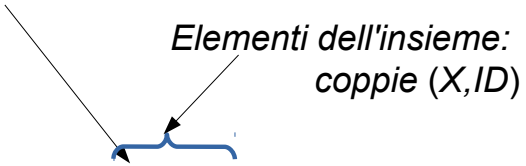
$$l < \#sum\{\ v_{1,1}, v_{1,2}, \ldots, v_{1,n}{:}a_1\ ;\ v_{2,1}, v_{2,2}, \ldots, v_{2,n}{:}a_2\ ;\ \ldots\} < u$$

- Il significato diventa *"considera l'insieme delle tuple* $(v_{i,1}, v_{i,2}, \ldots, v_{i,n})$ *tali che* a_i *è vero; calcola la somma dei* $v_{i,1}$*"*

$$\sum_{\{(v_{i,1}, v_{i,2}, \ldots, v_{i,n}) | a_i\}} v_{i,1}$$

Semantica di insieme

- `p(1,1).`
  `p(3,2).`
  `p(1,3).`
- `a(N):- N = #sum{ X,ID:p(X,ID) }.`



- *Answer set:* {*p*(1,1) *p*(3,2) *p*(1,3) *a*(5)}
- *Infatti* `#sum{` *X,ID*:*p*(*X,ID*) `}` =
  `#sum{`(1,1);(3,2);(1,3)`}` = 1+3+1 = 5

$$\sum_{(X,ID)\in\{(1,1),(3,2),(1,3)\}} X$$

UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Answer Set Programming

Ottimizzazione e Weak Constraints

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Prof. MARCO GAVANELLI

Ottimizzazione

- Fra tutti gli answer set di un programma, spesso è importante trovare quello ottimo rispetto ad un obiettivo.
- Si può aggiungere al programma un fatto maximize o minimize

  `#maximize` { t_1:a_1 ; t_2: a_2 ; … t_n:a_n }.

  `#minimize` { t_1:a_1 ; t_2: a_2 ; … t_n:a_n }.
- Di nuovo, t_i sono tuple, viene massimizzata o minimizzata la somma della prima coordinata delle tuple t_i tali che a_i è vero.

Weak constraints

- In alternativa, esistono anche i cosiddetti *weak constraints*
- Hanno sintassi simile agli Integrity Constraints

  $:\sim L_1, L_2, \ldots, L_n.\ [t_1,\ldots,t_k]$
- Gli Integrity Constraints devono tutti essere soddisfatti in qualunque modello stabile
- Intuitivamente, i Weak Constraints devono essere soddisfatti "il più possibile". Più precisamente:
  - Per ogni weak constraint soddisfatto, viene aggiunto all'insieme dei soddisfatti la tupla $(t_1,\ldots,t_k)$
  - Verrà fornita la soluzione che minimizza la somma della prima coordinata (t_1) delle tuple nell'insieme

Implementazione

- In realtà anche minimize e maximize sono implementate tramite weak constraints:
- **`#minimize`** { t_1:a_1 ; t_2: a_2 ; … t_n:a_n }.
- viene tradotto in
  :~ a_1. [t_1]
  :~ a_2. [t_2]
  …
  :~ a_n. [t_n]
- nella **`maximize`** si cambia il segno del primo elemento di ciascuna tupla

Esercizio

- *Si hanno dei file da salvare su CD-ROM; purtroppo i file non ci stanno tutti e bisogna selezionarne alcuni. Si vogliono selezionare i file da salvare sul CD-ROM, in modo da occupare il massimo spazio possibile. Il CD-ROM ha una capacità di 600MB*
- *Ogni file è descritto da un fatto*

```
file(ID,Dimensione)
```

- *Dove*
  - *ID è un identificatore univoco*
  - *Dimensione è l'occupazione in MB*
- *Si scriva un programma ASP che calcola quali file salvare sul CD-ROM*

UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Answer Set Programming

Conditional literals

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Prof. MARCO GAVANELLI

Conditional literals

- Un *letterale condizionale* ha la forma

$$l : l_1, l_2, \ldots, l_n$$

- Può essere immaginato come l'insieme di letterali $\{l | l_1, l_2, \ldots, l_n\}$ o come una implicazione $l \leftarrow l_1, l_2, \ldots l_n$.
- Il vantaggio è che i letterali condizionali possono essere inseriti all'interno di altre implicazioni.
- I letterali nell'insieme $\{l | l_1, l_2, \ldots, l_n\}$ sono considerati
  - in AND se sono nel body di una clausola
  - In OR se sono nella head

Esempi

- Se *q* è definito con `q(1). q(2). q(3).`

  `p(X) : q(X)`

  Viene espanso nell'insieme {*p*(*X*) | *q*(*X*) *è vero*}, cioè {p(1), p(2), p(3)}.

- Questi sono considerati
  - in AND se sono nel body di una clausola:
    `r :- p(X): q(X)` diventa `r:- p(1), p(2), p(3).`
  - In OR se sono nella head
    `p(X):q(X) :- s.` diventa `p(1) ; p(2) ; p(3):- s.`
- Es, dato `palette(red). palette(green). palette(blue).`

  `color(X,C): palette(C) :- node(X).`
- Espanso in

  `color(X,red); color(X,green) ; color(X,blue) :- node(X).`

Intervalli

`a(`*inf*`..`*sup*`)`

- Se l'atomo compare nella testa di una clausola, viene espanso come congiunzione

```
p(1..3):-r.
```

```
p(1):-r.
p(2):-r.
p(3):-r.
```

- Se l'atomo compare nel body, espanso come disgiunzione

```
r:- p(1..3).
```

```
r:- p(1).
r:- p(2).
r:- p(3).
```


UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Answer Set Programming

Intervalli

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Prof. MARCO GAVANELLI

UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Answer Set Programming

File multipli. Costanti

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Prof. MARCO GAVANELLI

QUESTO MATERIALE DIDATTICO È PER USO PERSONALE DELLO STUDENTE ED È COPERTO DA COPYRIGHT. NE È SEVERAMENTE VIETATA LA RIPRODUZIONE O IL RIUTILIZZO ANCHE PARZIALE, AI SENSI E PER GLI EFFETTI DELLA LEGGE SUL DIRITTO D'AUTORE.

File multipli

- *Gringo e clingo possono accettare più file; in questo caso viene creato il programma ground che contiene tutte le clausole dei vari file*
- *Questo è comodo per avere separata l'istanza dal problema*
- *Es*
  - *graph.pl contiene la definizione di un grafo*
  - *coloring.pl contiene il programma per il graph coloring*

```
clingo graph.pl coloring.pl
```


Costanti

- A volte è comodo definire costanti che si possano impostare da linea di comando
- `#const nomeCostante=val.`
- Imposta nel programma *nomeCostante* come una costante con valore a default *val.*
- Da linea di comando

```
clingo prog.pl -c nomeCostante=3
```

- Es

```
clingo queens.pl -c n=4
```

```
#const n=8.
riga(1..n).
colonna(1..n).

{regina(R,C)}:-
    riga(R),
    colonna(C).
...
```

queens.pl

UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Answer Set Programming

Esercizi: TSP, maximum clique

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Prof. MARCO GAVANELLI

Travelling Salesperson Problem

- *Nel TSP si richiede che un viaggiatore visiti tutti i nodi di un grafo esattamente una volta e torni al nodo iniziale, trovando il percorso di lunghezza minima*
- *Una proprietà del TSP è che in ogni nodo viene percorso esattamente un arco entrante ed uno uscente e che tutti i nodi sono raggiungibili passando solo per archi del percorso*

Travelling salesperson problem

- *Input: predicato node per ciascun nodo:*
  `node(1..6).`
- *Predicato cost per ciascun arco con relativo costo*

```
cost(1,2,2).  cost(1,3,3).  cost(1,4,1).
cost(2,4,2).  cost(2,5,2).  cost(2,6,4).
cost(3,1,3).  cost(3,4,2).  cost(3,5,2).
cost(4,1,1).  cost(4,2,2).  cost(5,6,1).
cost(5,3,2).  cost(5,4,2).  cost(6,5,1).
cost(6,2,4).  cost(6,3,3).
```

- *Può essere comodo definire un predicato che riporta gli archi:*
  `edge(X,Y):- cost(X,Y,C).`

Travelling salesperson problem

- *Decisioni: per ogni arco, decidiamo se va preso o no:* `cycle(X,Y).`
- *Da ogni nodo esce esattamente un arco selezionato:*

```
{ cycle(X,Y) : edge(X,Y) } = 1 :- node(X).
```

- *In ogni nodo entra esattamente un arco selezionato:*

```
{ cycle(X,Y) : edge(X,Y) } = 1 :- node(Y).
```

- *Tutti i nodi devono essere raggiungibili a partire da un nodo qualunque (es. Nodo 1)*

```
reached(Y) :- cycle(1,Y).
reached(Y) :- cycle(X,Y), reached(X).

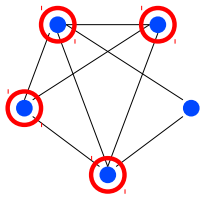
:- node(Y), not reached(Y).
```

- *Ottimizzazione:*

```
#minimize {C,X,Y : cycle(X,Y), cost(X,Y,C) }.
```


Es: maximum clique

- *Dato un grafo non diretto definito con dei fatti*

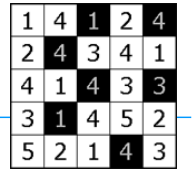


node(X)

arc(A,B)

- *Si vuole trovare l'insieme più grande di nodi tali che ogni coppia di nodi dell'insieme sia collegata da un arco*

Hitori



1	4	1	2	4
2	4	3	4	1
4	1	4	3	3
3	1	4	5	2
5	2	1	4	3

Nel puzzle Hitori viene data uno schema costituito da una matrice di numeri

Scopo del gioco è annerire alcune celle in modo che

- *Due celle nere non confinino in verticale o orizzontale*
- *In ogni riga o colonna, non compaia due volte lo stesso numero (nelle celle bianche)*
- *Le celle bianche siano tutte connesse*

Lo schema viene dato con dei fatti

`schema(Riga,Colonna,Valore).`