"Sapienza" Università di Roma

Facoltà di Ingegneria

Corso di Laurea Specialistica in Ingegneria Informatica

Tesina per il corso di Metodi Formali per l'Ingegneria del Software

Existential Second Order Logic e le sue applicazioni

Autori:

Angeloni Michela

Franchi Junio Valerio

Cangialosi Piero

Docente:

Toni Mancini

Indice:

1. Introduzione

Quando ci si trova di fronte a problemi combinatori si ricorre spesso all'utilizzo di programmi ad hoc per generare istanze SAT, in cui i modelli dell'istanza sono in relazione 1 a 1 con le soluzioni del problema, cioè in cui un'assegnazione è un modello se e solo se è una soluzione del problema combinatorio.
Questo approccio può rivelarsi costoso in quanto richiede la scrittura di un nuovo generatore per ogni nuovo problema, inoltre comporta un enorme dispendio di energie e di tempo ogni qualvolta è necessario modificare il problema.
In realtà in logica proposizionale non si riesce a fornire una vera e propria rappresentazione di un problema ma solo una specifica istanziazione, inoltre, come sappiamo, in logica proposizionale non si possono esprimere asserzioni sulle proprietà di insiemi di oggetti, ma si può solo associare ad ogni lettera il ruolo di rappresentare la verità o la falsità di un concetto. Mostriamo di seguito degli estratti del file DIMACS che rappresenta il problema del Social Golfer (cfr. par 3.3) per 8 giocatori, 4 gruppi e 3 settimane: come si può notare la rappresentazione del problema è lunga, complessa e poco leggibile, un utente che si trova di fronte a tale rappresentazione farà non poca fatica per comprendere le caratteristiche del problema.

```
c Variables:
c - plays(player, group, week)
c - all_in_group(group, week, <players>)
c START DICTIONARY
c plays(1,1,1) = 1
c plays(1,1,2) = 33
c plays(1,1,3) = 65
c plays(1,2,1) = 2
c plays(1,2,2) = 34
c plays(2,1,1) = 5
c plays(2,1,2) = 37
c plays(2,1,3) = 69
c plays(2,2,1) = 6
c plays(2,2,2) = 38
[...]
c all_in_group(1,1,<1,2>) = 97
c all_in_group(1,1,<1,3>) = 98
c all_in_group(1,1,<1,4>) = 99
c all_in_group(1,1,<1,5>) = 100
c all_in_group(1,1,<1,6>) = 101
c all_in_group(1,1,<1,7>) = 102
c all_in_group(1,1,<1,8>) = 103
c all_in_group(1,1,<2,3>) = 104
c all_in_group(1,1,<2,4>) = 105
c all_in_group(1,1,<2,5>) = 106
[...]
p cnf 432 4551
1 2 3 4 0
5 6 7 8 0
9 10 11 12 0
13 14 15 16 0
17 18 19 20 0
21 22 23 24 0
25 26 27 28 0
29 30 31 32 0
33 34 35 36 0
37 38 39 40 0
41 42 43 44 0
45 46 47 48 0
49 50 51 52 0
[...]
-90 -91 0
-90 -92 0
-91 -92 0
-93 -94 0
-93 -95 0
-93 -96 0
-94 -95 0
-94 -96 0
-95 -96 0
-1 -5 -33 -37 0
-1 -5 -65 -69 0
-33 -37 -65 -69 0
-1 -5 -34 -38 0
-1 -5 -66 -70 0
-33 -37 -66 -70 0
-1 -5 -35 -39 0
-1 -5 -67 -71 0
-33 -37 -67 -71 0
-1 -5 -36 -40 0
-1 -5 -68 -72 0
-33 -37 -68 -72 0
[...]
1 5 9 13 -17 21 -25 29 120 0
1 5 9 13 -17 21 25 -29 121 0
1 5 9 13 17 -21 -25 29 122 0
1 5 9 13 17 -21 25 -29 123 0
1 5 9 13 17 21 -25 -29 124 0
-33 -37 41 45 49 53 57 61 125 0
-33 37 -41 45 49 53 57 61 126 0
-33 37 41 -45 49 53 57 61 127 0
```

```
-33 37 41 45 -49 53 57 61 128 0            -33 37 41 45 49 53 -57 61 130 0
-33 37 41 45 49 -53 57 61 129 0
```

Il file DIMACS completo è poi passato come input ad un solutore, nel nostro caso zChaff (cfr. http://www.princeton.edu/~chaff/zchaff.html), che avrà il compito di dire se il problema è soddisfacibile o meno; per il problema precedente si avrà il seguente output:

```
Z-Chaff Version: zChaff 2007.3.12
Solving formula.txt ......

c 4551 Clauses are true, Verify Solution successful.
Instance Satisfiable
1 -2 -3 -4 -5 -6 7 -8 -9 -10 -11 12 -13 -14 -15 16 -17 -18 19 -20 21 -22 -23 -
24 -25 26 -27 -28 -29 30 -31 -32 33 -34 -35 -36 -37 -38 -39 40 -41 -42 43 -44 -
45 -46 -47 48 -49 -50 51 -52 -53 54 -55 -56 57 -58 -59 -60 -61 62 -63 -64 65 -
66 -67 -68 -69 -70 -71 72 -73 -74 -75 76 -77 -78 79 -80 -81 -82 83 -84 -85 86 -
87 -88 -89 90 -91 -92 93 -94 -95 -96 -97 -98 -99 -100 101 -102 -103 -104 -105 -
106 -107 -108 -109 -110 -111 -112 -113 -114 -115 -116 -117 -118 -119 -120 -121
-122 -123 -124 -125 -126 -127 -128 -129 130 -131 -132 -133 -134 -135 -136 -137
-138 -139 -140 -141 -142 -143 -144 -145 -146 -147 -148 -149 -150 -151 -152 -153
-154 -155 -156 -157 -158 159 -160 -161 -162 -163 -164 -165 -166 -167 -168 -169
-170 -171 -172 -173 -174 -175 -176 -177 -178 -179 -180 -181 -182 -183 -184 -185
-186 -187 -188 -189 -190 -191 -192 -193 -194 -195 -196 -197 -198 -199 -200 -201
-202 -203 -204 -205 -206 -207 208 -209 -210 -211 -212 -213 -214 -215 -216 -217
-218 -219 -220 -221 -222 -223 -224 -225 -226 -227 -228 -229 -230 -231 -232 -233
-234 235 -236 -237 -238 -239 -240 -241 -242 -243 -244 -245 -246 -247 -248 -249
-250 -251 -252 -253 -254 -255 -256 -257 -258 -259 -260 -261 262 -263 -264 -265
-266 -267 -268 -269 -270 -271 -272 -273 274 -275 -276 -277 -278 -279 -280 -281
-282 -283 -284 -285 -286 -287 -288 -289 -290 -291 -292 -293 -294 -295 -296 -297
-298 -299 -300 -301 -302 -303 -304 -305 -306 307 -308 -309 -310 -311 -312 -313
-314 -315 -316 -317 -318 -319 -320 -321 -322 -323 -324 -325 -326 -327 -328 -329
-330 -331 -332 -333 -334 -335 -336 -337 -338 339 -340 -341 -342 -343 -344 -345
-346 -347 -348 -349 -350 -351 -352 -353 -354 -355 -356 -357 -358 -359 -360 -361
362 -363 -364 -365 -366 -367 -368 -369 -370 -371 -372 -373 -374 -375 -376 -377
-378 -379 -380 -381 -382 -383 -384 385 -386 -387 -388 -389 -390 -391 -392 -393
-394 -395 -396 -397 -398 -399 -400 -401 -402 -403 -404 -405 -406 -407 -408 -409
-410 -411 412 -413 -414 -415 -416 -417 -418 -419 -420 -421 -422 -423 -424 -425
-426 -427 -428 -429 -430 -431 -432 Random Seed Used                    0
Max Decision Level                        25
Num. of Decisions                   26
( Stack + Vsids + Shrinking Decisions )        0 + 25 + 0
Original Num Variables                    432
Original Num Clauses                      4551
Original Num Literals                     14499
Added Conflict Clauses                    0
Num of Shrinkings                   0
Deleted Conflict Clauses                  0
Deleted Clauses                           0
Added Conflict Literals                   0
Deleted (Total) Literals                  0
Number of Implication                     432
Total Run Time                            0.004001
RESULT:     SAT
```

Come si può vedere l'output fornito dal solutore è complesso e poco espressivo.

Un approccio meno costoso potrebbe essere quello di descrivere il problema in logica, quindi attraverso un linguaggio più dichiarativo, e specificare il problema e le caratteristiche delle soluzioni.

A tale scopo scegliamo di utilizzare la logica del prim'ordine con il meccanismo della model expansion: si specifica il problema con una formula FOL su un insieme di predicati che codificano l'istanza e una sua soluzione e si utilizzano poi tecniche di ragionamento automatico per trovare soluzioni; ovvero cerco, se esiste, una espansione di questa interpretazione ad un modello.
In altre parole si passa alla programmazione dichiarativa che consiste nello specificare solo il problema e le caratteristiche che dovranno avere le soluzioni, ma non si specifica l'algoritmo per trovarle. Si usa quindi un ricercatore di modelli per trovare la soluzione.

Esempio: **Graph 3-coloring**

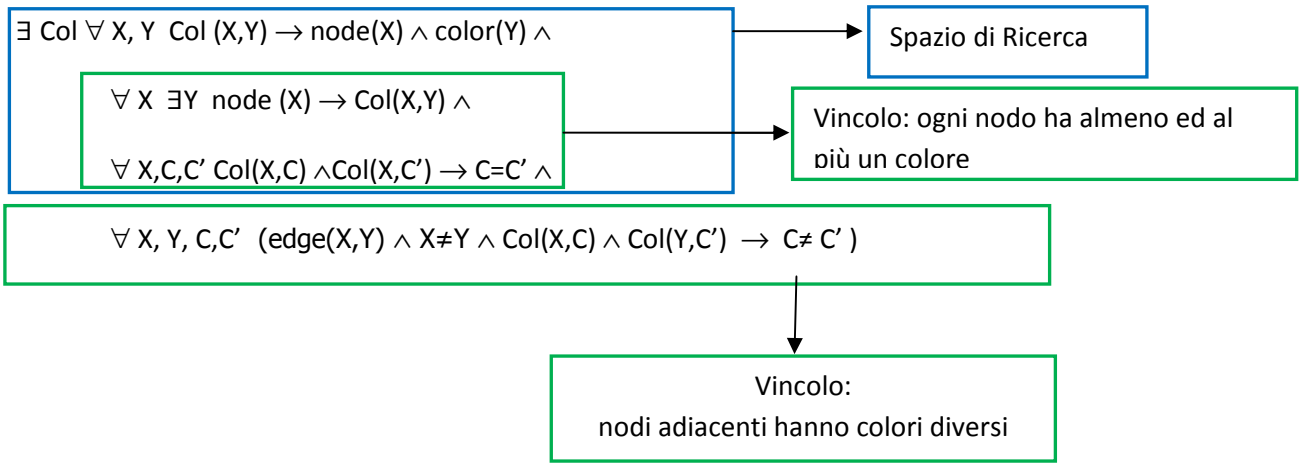
Specifica: Dato un grafo ed un insieme di 3 colori, trovare un'assegnazione di colori ai nodi tale che i nodi adiacenti hanno tutti colore diverso.
Vediamo come è fatto il file di input al ricercatore di modelli:

```
set(auto).
formula_list(usable).

% SP = { edge/2, r/1, g/1, b/1 }
% Nota: il dominio di interpretazione costituisce l'insieme dei nodi del grafo
%(non c'e' bisogno del predicato node/1)

% ogni nodo e' colorato
all x (r(x) | g(x) | b(x)).

% ogni nodo ha al più un colore
all x (r(x) -> -g(x)).
all x (r(x) -> -b(x)).
all x (g(x) -> -b(x)).

% vincoli di buona colorazione
all x y (edge(x,y) -> -(r(x) & r(y)) ).
all x y (edge(x,y) -> -(g(x) & g(y)) ).
all x y (edge(x,y) -> -(b(x) & b(y)) ).

%% codifica di M:
%% GRAFO
%%  R 0 ------ 1 B
%%    |        |
%%  G 2 ------ 3 R
% DCA
all x (x=0 | x=1 | x=2 | x=3).

% LR
edge(0,1).
edge(1,0).
edge(0,2).
edge(2,0).
edge(1,3).
edge(3,1).
edge(2,3).
```

```
edge(3,2).
r(0).
b(1).
g(2).
r(3).

% CWA
-edge(0,3).
-edge(3,0).
-edge(1,2).
-edge(2,1).
-edge(0,0).
-edge(1,1).
-edge(2,2).
-edge(3,3).
% Gli altri letterali sono implicati (ad es. -r(1), -g(1), ecc.)
end_of_list.
```

Questo file viene passato ad un solutore, nel nostro caso MACE (http://www.cs.unm.edu/~mccune/otter/) , che verifica che la codifica del modello fornita rispetti i requisiti e tutti i vincoli espressi come formule della logica del primo ordine, in altre parole abbiamo fatto una validazione di modello. Avremmo potuto omettere la codifica del modello e chiedere a MACE di trovarlo lui stesso.

Volendo fare un ulteriore passo in avanti si può utilizzare la logica del secondo ordine che permette di quantificare sui predicati.
In particolare molto utile è il frammento della logica del secondo ordine che utilizza solo il quantificatore esistenziale sui predicati: ESO.
ESO può essere usato come linguaggio di modellazione di problemi e, a differenza di quello che accade nella model expansion, non va ad espandere l'interpretazione ma si chiede se **esiste** una tale espansione.
Il problema del graph coloring viene quindi modellato come di seguito:



La differenza principale tra le due modellazioni del problema è che con la logica del primo ordine posso quantificare solo sulle variabili e quindi nella ricerca delle soluzioni cerco di trovare un'assegnazione di variabili che verifichi tutti i vincoli, mentre in ESO quantifico sul predicato Col e cerco, se esiste, una colorazione che rispetti tutti i vincoli.

2. Logica del Secondo Ordine

Tutti i linguaggi di programmazione correnti possono essere visti come frammenti della Logica del Secondo Ordine.
La Logica del Secondo Ordine è un'estensione della logica del primo ordine e differisce da quest'ultima per la possibilità di quantificare predicati e funzioni, a questo scopo la sintassi della logica del primo ordine è arricchita con nuovi simboli logici:

- *Variabili Predicato:* $X^n_1, X^n_2 \ldots$
- *Variabili Funzione:* $F^n_1, F^n_2 \ldots$

Quelle che erano considerate variabili nella logica del primo ordine ora vengono chiamate "*variabili individuali*".

Formalmente possiamo dare la seguente definizione:

<u>Definizione 2.1</u> (Alfabeto della logica del secondo ordine) L'alfabeto della logica del secondo ordine è formato da:

- *un insieme* ***VP*** *di* variabili predicato (per ogni $n \geq 0$ $X^n_1, X^n_2 \ldots$*);*
- *un insieme* ***VF*** *di* variabili funzione (per ogni $n \geq 0$ $F^n_1, F^n_2 \ldots$*;*
- *un insieme* ***V*** *di* variabili individuali;
- *un insieme* ***SF*** *di* simboli di funzione*, ognuno dei quali ha associato il suo numero di argomenti detto arità;*
- *un insieme* ***SP*** *di* simboli di predicato*, ognuno dei quali ha associato il suo numero di argomenti detto arità;*
- *i connettivi logici* $\neg, \wedge, \vee, \rightarrow, \equiv$;
- *i quantificatori* $\forall$ *ed* $\exists$*, denominati rispettivamente quantificatore* universale *e quantificatore* esistenziale*;*
- *i simboli speciali " (", ") " e " , " (virgola).*

I simboli di funzione di arità 0 sono detti simboli di costante.

I *Termini* diventano espressioni costruite a partire da simboli di costante e variabili individuali applicando simboli di funzione o variabili funzione; le *Formule Atomiche* sono sempre della forma $P(t_1,...,t_n)$ dove t_i sono termini e P è un simbolo di predicato n-ario oppure una variabile di predicato.

<u>Definizione 2.2</u> (Termini) *L'insieme dei* Termini *è definito induttivamente come segue:*

- *ogni variabile individuale in V è un termine;*
- *ogni simbolo di costante in SF è un termine;*
- *se f è un simbolo di funzione di arità n e t1…tn sono termini allora f(t1…tn) è un termine;*
- *se* F^n *è una variabile funzione allora F(t1…tn) è un termine.*

<u>Definizione 2.3</u> (Formule) *L'insieme delle* formule *è definito induttivamente come segue:*

- *se p è un simbolo di predicato di arità n oppure una variabile predicato e* $t1, \ldots, tn$ *sono termini, allore* $p(t_1, \ldots, t_n)$ *è una formula (detta formula* atomica*);*

- *se φ e ψ sono formule, lo sono anche:*
  - (φ)
  - $\neg\varphi$
  - $\varphi \vee \psi$
  - $\varphi \wedge \psi$
  - $\varphi \rightarrow \psi$
  - $\varphi \equiv \psi$
- *se φ è una formula e V è una variabile individuale allora anche:*
  - $\forall V\ \varphi$
  - $\exists V\ \varphi$

  sono formule;
- *se φ è una formula allora lo sono anche :*
  - $(\forall X_i^n)\ \varphi$
  - $(\forall F_i^n)\varphi$

Per quanto riguarda la semantica occorre estendere il concetto di soddisfazione per includere i nuovi quantificatori. Un'interpretazione I è definita come per la logica del primo ordine; si ha la seguente definizione di soddisfacibilità:

<u>Definizione 2.4</u> : *Sia I un'interpretazione, V l'insieme di tutte le variabili (individuali, predicato e funzione) ed s una funzione su V che assegna ad ogni variabile il tipo esatto di ogni oggetto nell'interpretazione.*
Una formula è soddisfatta in un'interpretazione se sono verificate tutte le condizioni che valgono al primo ordine oppure nei seguenti casi:

- $\models I\ (\forall X^n)\ \varphi[s]$ *se e solo se per ogni relazione n-aria* $R \subseteq |I|^n$, $\models I\ \varphi[s(X^n|R)]$. *Dove* $s(X^n|R)$ *è la funzione che è esattamente come s eccetto che per la variabile predicato* X^n *essa prende il valore R.*
- $\models I\ (\forall F^n)\ \varphi[s]$ *se e solo se per ogni funzione n-aria f:* $|I|^n \rightarrow |I|$, $\models I\ \varphi[s(F^n|f)]$. *Dove* $s(F^n|f)$ *è la funzione che è esattamente come s eccetto che per la variabile funzione* F^n *essa prende il valore f.*

In altre parole $(\forall X^n)\varphi$ significa che indipendentemente dall'estensione della variabile predicato n-aria che si ha nell'interpretazione, φ sarà vera in quella interpretazione.

La Logica del Secondo Ordine è molto usata per trovare gli insiemi che soddisfano dei requisiti, inoltre al secondo ordine è possibile interpretare quantificatori della logica del primo ordine come "predicati" che esprimono delle proprietà di un insieme di individui, ad esempio il quantificatore $\exists$ esprime la proprietà di essere un insieme non vuoto, mentre $\forall$ esprime la proprietà di denotare l'intero dominio D.

3. Existential Second Order Logic

In realtà i linguaggi di modellazione sono considerati estensione di un sottoinsieme della logica del secondo ordine, ESO (Existential Second Order Logic) che usa il quantificatore esistenziale per definire lo spazio di ricerca; ESO è particolarmente indicato per essere usato come linguaggio formale per modellare le specifiche di un problema a vincoli.

Abbiamo deciso di prendere in considerazione un modello di ESO che non usa funzioni, cioè che non ha un supporto nativo per gestire le funzioni, dal momento che non c'è una perdita di potere espressivo, infatti questa mancanza può essere colmata in altri modi, come vedremo in seguito.

Formalmente una specifica ESO che descrive un problema di ricerca $\boldsymbol{\pi}$ è una formula del tipo:

$$\psi \doteq \exists \boldsymbol{S}\, \phi(\boldsymbol{S},\boldsymbol{R})$$

dove

- $\boldsymbol{R}$ è un insieme di predicati ed è l'input dello schema relazionale, in altre parole $\boldsymbol{R}$ è un insieme fissato di relazioni di arità date, che denota lo schema per tutte le istanze di input di $\boldsymbol{\pi}$. Un'istanza $\boldsymbol{I}$ del problema è data come database relazionale sullo schema $\boldsymbol{R}$, cioè come estensione di tutte le relazioni in $\boldsymbol{R}$.
- $\boldsymbol{S}$ è l'insieme dei predicati esistenzialmente quantificati, tali predicati sono chiamati *guessed* (indovinati) e le loro possibili estensioni con tuple nell'Universo di Herbrand codificano i punti dello spazio di ricerca con per il problema $\boldsymbol{\pi}$ sull'istanza data $\boldsymbol{I}$.
- ϕ è una formula chiusa del primo ordine, sul vocabolario relazionale $\boldsymbol{S} \cup \boldsymbol{R} \cup \{=\}$, che codifica i vincoli che un'estensione di predicati in $\boldsymbol{S}$ deve soddisfare per essere considerata una soluzione di $\boldsymbol{\pi}$ sull'istanza $\boldsymbol{I}$.

<u>Definizione 3.1</u>(Soddisfacibilità): *Un'istanza $\boldsymbol{I}$ di $\boldsymbol{\pi}$, codificata come database relazionale con schema $\boldsymbol{R}$, è soddisfacibile se e solo se esiste una lista di relazioni $\boldsymbol{\Sigma_1,\ldots\ \Sigma_n}$ che insieme ad $\boldsymbol{I}$ soddisfa la parte di formula al primo ordine, cioè tale che*

$$(\Sigma_1,\ldots,\Sigma_n, I) \models \phi.$$

In altre parole possiamo dire che la semantica di una formula ESO del tipo illustrato è trovare, data un'estensione di predicati in $\boldsymbol{R}$ (istanza $\boldsymbol{I}$), un'estensione di predicati in $\boldsymbol{S}$, con elementi nell'universo di Herbrand, che soddisfi la formula $\boldsymbol{\phi}$.

<u>Definizione 3.2</u> (Clausola): *Una clausola è una formula del tipo*

$$\forall X_1, \ldots, X_l\ \neg A_1 \vee \ldots \vee \neg A_n \vee B_1 \vee \ldots \vee B_m$$

ovvero una formula quantificata universalmente che viene spesso scritta come

$$A_1 \wedge \ldots \wedge A_n \rightarrow B_1 \vee \ldots \vee B_m.$$

dove $X_1 \ldots X_l$ sono Variabili, $A_1 \ldots A_n$ e $B_1 \ldots B_m$ sono simboli di predicato.

Riteniamo opportuno fornire la definizione dell'Universo di Herbrand:

<u>Definizione 3.3</u> (Universo di Herbrand): *Dato un insieme di clausole C, l'**Universo di Herbrand** H(C) è definito nel seguente modo:*

- *H(C) contiene i simboli di costante che occorrono in C;*
- *se f è un simbolo di funzione n-aria che occorre in C e $h_1, \ldots, h_n$ sono elementi di H(C), allora anche $f(h_1,..,h_n)$ sta in H(C)*

Ad esempio se C = {p(a), p(X), q(Y), q(f(Y)} allora H(C) ={a, f(a), f(f(a), ….. }.
Nel nostro caso, per l'insieme ***S*** dei predicati guessed, l'Universo di Herband è formato dalle costanti che occorrono in ***I*** più le costanti che occorrono in **ϕ.**
Altri concetti importanti relativi all'universo di Herbrand sono la **Base di Herbrand** e l'**Interpretazione di Herbrand** che sono definiti nel seguente modo:

<u>Definizione 3.4</u> (Base di Herbrand B(C)): *Dato un insieme C di clausole e l'universo di Herbrand H(C) per C, la base di Herbrand B(C) è l'insieme delle istanze ground delle formule atomiche che occorrono in C.*

Ad esempio, dati C ed H(C) come nell'esempio precedente B(C)= {p(a), p(f(a)), p(f(f(a))),… ,q(a), q(f(a)), q(f(f(a))),…}.

<u>Definizione 3.5</u> (Interpretazione di Herbrand): *Interpretazione di un insieme C di clausole in cui:*

- *il dominio è l'universo di Herbrand H(C),*
- *ogni simbolo di costante è interpretato sulla corrispondente costante in H(C),*
- *ogni simbolo di funzione è interpretato come funzione che trasforma $h_1,\ldots h_n$ in $f(h_1, \ldots h_n)$,*
- *ogni simbolo di predicato in una relazione su B(C).*

Quindi definire un'interpretazione di Herbrand corrisponde a dire quale sottoinsieme della base di Herbrand è vera nell'interpretazione.
Le interpretazioni di Herbrand giocano un ruolo importante in quanto ci si può limitare ad esse nella dimostrazione di teoremi, infatti un insieme C di clausole è insoddisfacibile se e solo se non esiste un'interpretazione di Herbrand che lo soddisfa; quindi nel processo di dimostrazione ci si può limitare a considerare tali interpretazioni.

Da un punto di vista astratto tutti i linguaggi usati per trattare le specifiche dei problemi possono essere visti come un'estensione di ESO su database finiti, dove il quantificatore esistenziale al secondo ordine e la formula ϕ del primo ordine rappresentano, rispettivamente, le fasi "indovina" e "verifica" del paradigma di programmazione a vincoli.
ESO può essere considerata come la base logica formale per tutti i linguaggi per la modellazione a vincoli, essendo in grado di rappresentare tutti i problemi di ricerca nella classe di complessità NP, infatti per il **teorema di Fagin** (cfr. Neil Innermann, Descriptive Complexity,ed. Springer, par 7.2) si ha che un problema di decisione è in NP se e solo se esiste una formula ESO che lo esprime, quindi questo frammento della logica del secondo ordine può essere considerato un linguaggio di modellazione astratto per i problemi a vincoli.
Nello specifico l'enunciato del teorema è il seguente:

<u>Teorema 3.1</u>(Fagin): *NP è uguale all'insieme delle queries booleane esistenziali del secondo ordine, ovvero NP=ESO. Inoltre quest'uguaglianza resta vera quando la parte al prim'ordine, della formula al secondo ordine, è quantificata universalmente.*

Questo teorema caratterizza la complessità di NP in un modo elegante e indipendente dalla macchina. Un'interessante conseguenza di questo teorema è espressa dal seguente corollario:

<u>Corollario 3.1:</u> *Ogni query computabile in tempo polinomiale è esprimibile come un formula esistenziale di Horn al secondo ordine: $P \subseteq$ ESO-Horn.*

Ricordiamo che una formula di Horn è una formula in forma normale congiuntiva con al massimo un letterale positivo per clausola.

3.1 Vantaggi di ESO

Avere un linguaggio di modellazione basato sulla logica del secondo ordine è vantaggioso perché:

- è pienamente dichiarativo;
- ha proprietà computazionali chiare: cioè per quanto riguarda il potere espressivo si ha che ESO cattura NP, mentre per quanto riguarda la complessità dei dati è NP-completo;
- può fornire un ragionamento automatico su formule logiche;
- la letteratura esistente può essere applicata al ragionamento.

Il vantaggio principale riguarda il fatto che, se si vuole verificare se l'applicazione delle varie tecniche di formulazione riduce il lavoro di verifica delle proprietà delle formule logiche (spesso del primo ordine), si può usare la letteratura esistente sulla logica e si possono usare tool automatici per realizzare molti task di ricognizione.
Inoltre il fatto che ESO catturi NP lo rende adatto per molti problemi interessanti (CSP).

3.2. Svantaggi di ESO

Rispetto ai linguaggi correnti, il modello di ESO preso in considerazione, non fornisce un supporto nativo per trattare alcune cose che permettono di facilitare la scrittura delle specifiche, in particolare mancano:

- Un supporto nativo per tipi e relazioni tipizzate;
- Un supporto nativo per le funzioni;
- Un supporto nativo per interi limitati e aritmetiche.

Queste mancanze possono essere colmate lavorando da un punto di vista sintattico, in modo da rendere le specifiche in ESO più compatte e più vicine alla loro controparte nei linguaggi correnti, lasciando inalterato il potere espressivo.
In particolare per quanto riguarda i tipi e le relazioni tipizzate si può risolvere il problema usando un linguaggio al primo ordine multi-sorta, oppure usando dei predicati monadici aggiuntivi nell'insieme ***R***; per le funzioni il supporto può essere introdotto usando vincoli aggiuntivi al primo

ordine; ad esempio, riprendendo il problema del graph coloring, vediamo come vengono utilizzate queste strategie con l'aggiunta di vincoli al prim'ordine.

$\exists$ Col $\forall$ X, Y Col (X,Y) $\rightarrow$ node(X) $\wedge$ color(Y) $\wedge$

Tipizzazione di Col:
- prima colonna: nodo
- seconda colonna: colore

$\forall$ X $\exists$Y node (X) $\rightarrow$ Col(X,Y) $\wedge$

$\forall$ X,C,C' Col(X,C) $\wedge$Col(X,C') $\rightarrow$ C=C' $\wedge$

Col è una funzione totale:
- primo vincolo: totalità
- secondo vincolo: mono-valore

$\forall$ X, Y, C,C' (edge(X,Y) $\wedge$ X$\neq$Y $\wedge$ Col(X,C) $\wedge$ Col(Y,C') $\rightarrow$ C$\neq$ C')

Nello specifico ricordiamo che la logica multi-sorta ci permette di trattare l'universo non come una collezione omogenea di oggetti ma di tipizzare tali oggetti, come avviene nella programmazione tipizzata: ci permette, quindi, di creare delle partizioni di tipi diversi.
Sia le parti funzionali che le asserzioni del linguaggio della logica riflettono questo partizionamento tipizzato dell'universo anche a livello sintattico: sostituzioni e passaggi di argomenti possono essere fatti solo nel rispetto delle sorte.
Una logica multi-sorta può essere formalizzata in più modi, in rispetto con le regole dette sopra; nella maggior parte dei casi sono caratterizzati da un insieme Sort di sorte e un'appropriata generalizzazione di SIGNATURE per essere in grado di gestire le informazioni addizionali che stanno insieme alle sorte. Una SIGNATURE è una tupla <*Sort,Con,Fun,Pred*>, dove *Sort* è un insieme non vuoto di sorte, *Con* è un insieme numerabile di simboli di costante, *Fun* è un insieme numerabile di simboli di funzione, *Pred* è un insieme numerabile di simboli di predicato.
Intuitivamente quello che ci interessa è dividere gli oggetti del nostro dominio di interpretazione in sottoinsiemi: ad esempio se volessimo partizionare il nostro universo di Persone in base al sesso potremmo introdurre due predicati monadici che asseriscono il sesso di ogni individuo. Quindi avremmo: {Uomo/1, Donna/1}.
Questa tecnica ci permette di capire quanto sia utile, in un universo in cui gli oggetti sono molteplici, partizionare un insieme grande in sottoinsiemi più piccoli che asseriscono l'appartenenza di un individuo ad una specifica classe.

Per quanto riguarda gli interi e le aritmetiche si può lavorare in due modi, si può pre-interpretare oppure indovinare: nel primo caso si assume che esistano in ***R*** delle speciali relazioni, che rappresentano (pre-interpretano) gli interi limitati e le operazioni tra essi, in altre parole si aggiungono in ***R*** le relazioni per il dominio dei numeri e le operazioni necessarie (vedi esempio a seguire); nel secondo caso si aggiungono in ***S*** delle relazioni che codificano il dominio limitato per gli interi e le operazioni necessarie, cioè si indovinano delle relazioni aggiuntive che permettono di vedere le costanti del dominio di Herbrand H (o le tuple di H_k, per un k abbastanza grande) come interi, trovando un ordinamento totale su di esse.
Ad esempio, con il primo metodo, se sono necessari i numeri da 0 a 10 si aggiungono in R le seguenti relazioni:

Numbers
0
1
2
3
4
5
6
7
8
9
10

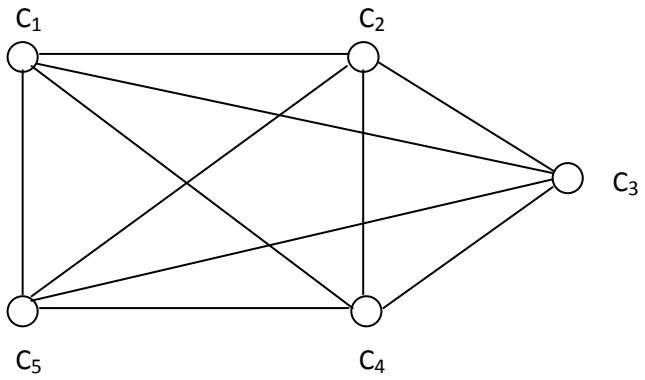
Succ (Y=X+1)	
X	Y
0	1
1	2
2	3
3	4
4	5
5	6
6	7
7	8
8	9
9	10

Sum (X+Y=Z)		
X	Y	Z
0	0	0
0	1	1
0	2	2
...	...	...
5	2	7
5	3	8
5	4	9
5	5	10

Diff (X-Y=Z)		
X	Y	Z
0	0	0
1	0	1
1	1	0
...	...	...
10	3	7
10	2	8
10	1	9
10	0	10

Con il secondo metodo, supponiamo di avere la necessità di utilizzare operazioni aritmetiche su interi limitati da 1 a size(H) dove H è l'universo di Herbrand. Per avere a disposizione l'insieme di interi di interesse utilizziamo gli elementi stessi del dominio di interpretazione, che sono size(H), indovinando, mediante una formula ESO ausiliaria, un ordinamento totale tra di essi. A questo punto è necessario risolvere un problema ausiliario in NP, cioè è necessario trovare un cammino Hamiltoniano sul grafo completo; questo percorso individua una funzione successore sulle costanti viste come interi. Dato che il problema è sicuramente in NP, per il teorema di Fagin siamo sicuri di poterlo specificare con una formula ESO.
Se servono relazioni che codifichino operazioni (ad es., somma, sottrazione) è necessario indovinare altre relazioni (guessed) con formule ESO opportune da mettere in and con il problema. Tali formule indovineranno opportune estensioni di nuovi predicati guessed (S, ad es., somma()) usando come input (R) l'estensione indovinata di succ().
Se abbiamo la necessità di utilizzare interi più grandi ma sempre limitati, invece di considerare singoli elementi del dominio di Herbrand come interi, si utilizzano tuple di dimensione k (fissato). In questo modo si può contare fino a $|H|^k$. Se infine servono interi negativi si può usare il complemento, oppure modificare tutti i vincoli di modo da avere solo interi negativi.
Il fatto che tali relazioni si possano indovinare viene dall'osservazione che tutti questi problemi (una volta che gli interi sono limitati) sono in NP, ad esempio per verificare che un'estensione della relazione guessed "somma" sia corretta, è sufficiente un tempo polinomiale dal momento che si avranno al più $|H|^3$ tuple, e posso verificare che tutte le triple <X,Y,Z> che sono presenti sono corrette (X+Y=Z) e che tutte quelle assenti sono tali che $X+Y \neq Z$.
Quindi, essendo tutti questi problemi in NP, per il teorema di Fagin esistono formule ESO che li codificano. Di conseguenza, come accennato precedentemente, aggiungere aritmetica su interi limitati NON altera il potere espressivo del linguaggio.

Per rendere la cosa più chiara consideriamo un piccolo esempio in cui abbiamo come uniche costanti del dominio $\{C_1,C_2,C_3,C_4,C_5\}$.
Per creare un ordinamento tra le costanti dobbiamo trovare un cammino Hamiltoniano sul seguente grafo, in cui ad ogni nodo corrisponde una costante.



PATH → SUCC()

C_5 C_1
C_1 C_3
C_3 C_4
C_4 C_2
C_2 C_5

Trovando un ordinamento tra i nodi si crea la funzione successore.
Il problema originariamente espresso come $\exists$ S Φ(S,R) diventa a questo punto:

$$\exists \text{ S,PATH } \Phi(S,R) \wedge \Psi(PATH,R)$$

N.B. Occorre fare attenzione al fatto che non basta generare la colonna destra di SUCC permutando arbitrariamente gli elementi della sinistra. Questo è noto come il problema dei "sottocicli": infatti considerando i 5 elementi c1...c5 si potrebbe avere:
SUCC
c1 c2
c2 c3
c3 c1
c4 c5
c5 c4
Entrambe le colonne sono una permutazione dell'insieme degli elementi del dominio, ma la relazione NON individua un ordinamento totale tra di esse!

3.3 Esempi di problemi modellati in ESO

Ramsey Problem
Il problema originario consiste nel colorare gli archi di un grafo completo con n nodi, usando il minor numero di colori, in modo tale che non ci siano triangoli monocromatici nel grafo, cioè in modo che in ogni triangolo al massimo due archi hanno lo stesso colore; dal momento che non è possibile esprimere problemi di ottimizzazione in ESO consideriamo il problema decisionale associato che dato un certo numero di colori decide se esiste una colorazione che soddisfa i requisiti.
L'input del problema è quindi dato dal numero di nodi del grafo e dal numero di colori, mentre lo spazio di ricerca è costituito da tutte le possibili assegnazioni di colori agli archi del grafo.

L'unico vincolo presente nel problema è quello che afferma che per ogni tripla di nodi differenti, non tutti gli archi che collegano coppie di essi hanno lo stesso colore.

$\exists$ Col $\forall$ X, Y , Z Col (X,Y, Z) $\rightarrow$ node(X) $\wedge$ node(Y) $\wedge$ color(Z) $\wedge$

$\forall$ X, Y $\exists$Z Col(X,Y,Z) $\wedge$

$\forall$ X,Y,C,C' Col(X,Y,C) $\wedge$Col(X,Y,C') $\rightarrow$ C=C' $\wedge$

Spazio di Ricerca

$\forall$ X, Y,Z ,C,C',C" (X$\neq$Y $\wedge$ X$\neq$Z $\wedge$ Y $\neq$ Z $\wedge$ Col(X,Y,C) $\wedge$ Col(X,Z,C') $\wedge$ Col(Y, Z,C"))

$\rightarrow$ (C$\neq$ C' $\vee$ C$\neq$ C" $\vee$ C'$\neq$ C")

Vincoli:
Se esiste un triangolo di nodi allora al massimo due archi hanno lo stesso colore

I vincoli espressi servono a definire le proprietà dello spazio di ricerca e ad esprimere i vicoli che devono essere soddisfatti, nello specifico:

1. $\forall$ X, Y , Z Col (X,Y, Z) $\rightarrow$ node(X) $\wedge$ node(Y) $\wedge$ color(Z) esprime la tipizzazione, cioè afferma che per ogni colorazione Col i primi due argomenti sono nodi e il terzo è un colore;
2. $\forall$ X, Y $\exists$Z Col(X,Y,Z) per ogni coppia di nodi esiste almeno un colore associato;
3. $\forall$ X,Y,C,C' Col(X,Y,C) $\wedge$Col(X,Y,C') $\rightarrow$ C=C' per ogni coppia di nodi esiste al più un colore associato;

I vincoli 2 e 3 esprimono quindi che Col è una funzione totale e mono valore.

4. $\forall$ X, Y, Z ,C,C',C" (X$\neq$Y $\wedge$ X$\neq$Z $\wedge$ Y $\neq$ Z $\wedge$ Col(X,Y,C) $\wedge$ Col(X,Z,C') $\wedge$ Col(Y, Z,C")) $\rightarrow$ (C$\neq$ C' $\vee$ C$\neq$ C" $\vee$ C'$\neq$ C") esprime il vincolo proprio del problema affermando che per ogni tripla di nodi diversi e per ogni tripla di colori associati alle possibili coppie di nodi, non si ha che tutti i colori sono uguali, ma almeno due di essi sono diversi.

Word design

Il problema consiste nel trovare quanto è grande l'insieme S di stringhe di lunghezza 8 sull'alfabeto W = {A, C, G , T} con le seguenti proprietà:

1. Ogni parola in S ha 4 simboli in {C, G};
2. Ogni coppia di parole distinte in S differisce in almeno 4 posizioni;
3. Ogni coppia di parole x ed y in S (con x ed y che possono essere uguali) è tale che R(x) e C(y) differiscono in almeno 4 posizioni.

Dove R(x1,…,x8)= x8,…,x1 è il reverse di x1,…,x8 e C(y1,…,y8) è il complemento di y, cioè la parola in cui ogni A è sostituita con una T e viceversa e ogni C è sostituita da una G e vice versa.
Si considera una versione decisionale del problema, cioè dato un intero n il problema consiste nel trovare un insieme di n parole con le proprietà specificate.
Lo spazio di ricerca è costituito da tutti i possibili insiemi di n parole di lunghezza 8 con elementi in {A, C, G, T}.

I vincoli sono quelli specificati dalle 3 proprietà da soddisfare.

alphabet replacement(alphabet) = # [A=T, C=G, G=C, T=A]#; → Funzione ausiliaria

∃ words ∀ X, Y, Z words(X,Y,Z) → words_idx(X) ∧ letters_idx(Y) ∧letter(z) ∧ → Spazio di Ricerca

∀ w words_idx(w) → ∃l1, l2, l3, l4, z1, z2 (l1 ≠l2 ∧ l1 ≠l3 ∧ l1 ≠l4 ∧ l2 ≠l3 ∧ l2 ≠l4 ∧ l3 ≠l4) ∧
letters_idx(l1) ∧ letters_idx(l2) ∧ letters_idx(l3) ∧ letters_idx(l4)∧ letter(z1) ∧ letter(z2) ∧

(words(w, l1, z1) ∨ words(w, l1, z2)) ∧ (words(w, l2,z1) ∨ words(w, l2,z2) ∧

(words(w, l3,z1) ∨ words(w, l3,z2)) ∧ (words(w, l4,z1) ∨ words(w, l4,z2)) ∧

(z1=C) ∧(z2=G)∧

∀ w, l1, l2,l3,l4,l5, z1, z2 words_idx(w) ∧ letters_idx(l1) ∧ letters_idx(l2) ∧ letters_idx(l3) ∧
letters_idx(l4) ∧ letters_idx(l5) ∧ letter(z1) ∧ letter(z2) ∧

(words(w, l1,z1) ∨ words(w, l1, z2)) ∧ (words(w, l2,z1) ∨ words(w, l2,z2)) ∧

(words(w, l3,z1) ∨ words(w, l3,z2)) ∧ (words(w, l4,z1) ∨ words(w, l4,z2)) ∧

(words(w, l5,z1) ∨ words(w, l5,z2)) ∧ (z1=C) ∧(z2=G) →

(l1=l2 ∨ l1=l3 ∨ l1=l4 ∨ l1=l5 ∨ l2=l3 ∨ l2=l4 ∨ l2=l5 ∨ l3=l4 ∨ l3=l5 ∨ l4=l5) ∧

Vincolo 1: ogni parola ha esattamente 4 simboli in {C, G}

∀ w1, w2 words_idx(w1) ∧ words_idx(w2) ∧ w1≠w2 →

∃ l1, l2, l3, l4, z1, z2, z3, z4 (l1 ≠l2 ∧ l1 ≠l3 ∧ l1 ≠l4 ∧ l2 ≠l3 ∧ l2 ≠l4 ∧ l3 ≠l4) ∧

(¬words(w1, l1,z1) ∨ ¬words(w2, l1,z1)) ∧ (¬words(w1, l2, z2) ∨ ¬ words(w2, l2,z2)) ∧

(¬words(w1, l3,z3) ∨ ¬ words(w2, l3,z3)) ∧ (¬words(w1, l4,z4) ∨ ¬ words(w2, l4,z4)) ∧

Vincolo 2: ogni coppia di parole distinte differisce di almeno 4 posizioni

∀ w1, w2, words_idx(w1) ∧ words_idx(w2) →

∃ l1, l2, l3, l4 , z1, z2, z3, z4,z5,z6,z7,z8 (l1 ≠l2 ∧ l1 ≠l3 ∧ l1 ≠l4 ∧ l2 ≠l3 ∧ l2 ≠l4 ∧ l3 ≠l4) ∧

(¬words(w1, 9 –l1,z1) ∨ ¬ (words(w2, l1, z2)∧ z1=replacemet(z2))) ∧

(¬words(w1, 9 –l2,z3) ∨ ¬ (words(w2, l2, z4)∧ z3=replacemet(z4))) ∧

(¬words(w1, 9 –l3,z5) ∨ ¬ (words(w2, l3, z6)∧ z5=replacemet(z6))) ∧

(¬words(w1, 9 –l4,z7) ∨ ¬ (words(w2, l4, z8)∧ z7=replacemet(z8))) ∧

Vincolo 3: per ogni coppia di parole, il reverse della prima e il complemento della seconda differiscono di almeno 4 posizioni

Per scrivere la specifica ESO di questo problema abbiamo avuto bisogno di definire una funzione ausiliaria, replacement e abbiamo scelto di "pre-interpretarla".
Analizziamo in dettaglio i vari vincoli:

1. $\forall$ X, Y, Z words(X,Y,Z) $\rightarrow$ words_idx(X) $\wedge$ letters_idx(Y) $\wedge$letter(Z) : come nell'esempio precedente questo vincolo esprime la tipizzazione del predicato word affermando che il primo argomento è un indice di parola, il secondo è un indice di lettera e il terzo è una lettera.

2. $\forall$ w words_idx(w) $\rightarrow$ $\exists$l1, l2, l3, l4, z1, z2 (l1 $\neq$l2 $\wedge$ l1 $\neq$l3 $\wedge$ l1 $\neq$l4 $\wedge$ l2 $\neq$l3 $\wedge$ l2 $\neq$l4 $\wedge$ l3 $\neq$l4) $\wedge$ letters_idx(l1) $\wedge$ letters_idx(l2) $\wedge$ letters_idx(l3) $\wedge$ letters_idx(l4)$\wedge$ letter(z1) $\wedge$ letter(z2) $\wedge$ (words(w, l1, z1) $\vee$ words(w, l1, z2)) $\wedge$ (words(w, l2,z1) $\vee$ words(w, l2,z2) $\wedge$ (words(w, l3,z1) $\vee$ words(w, l3,z2)) $\wedge$ (words(w, l4,z1) $\vee$ words(w, l4,z2)) $\wedge$ (z1=C) $\wedge$(z2=G): rappresenta la prima parte del vincolo 1, cioè il fatto che ogni parola deve contenere almeno 4 simboli uguali a C oppure uguali a G.

3. $\forall$ w, l1, l2, l3, l4, l5, z1, z2 words_idx(w) $\wedge$ letters_idx(l1) $\wedge$ letters_idx(l2) $\wedge$ letters_idx(l3) $\wedge$ letters_idx(l4) $\wedge$ letters_idx(l5) $\wedge$ letter(z1) $\wedge$ letter(z2) $\wedge$ (words(w, l1,z1) $\vee$ words(w, l1, z2)) $\wedge$ (words(w, l2,z1) $\vee$ words(w, l2,z2)) $\wedge$ (words(w, l3,z1) $\vee$ words(w, l3,z2)) $\wedge$ (words(w, l4,z1) $\vee$ words(w, l4,z2)) $\wedge$ (words(w, l5,z1) $\vee$ words(w, l5,z2)) $\wedge$ (z1=C) $\wedge$(z2=G) $\rightarrow$ (l1=l2 $\vee$ l1=l3 $\vee$ l1=l4 $\vee$ l1=l5 $\vee$ l2=l3 $\vee$ l2=l4 $\vee$ l2=l5 $\vee$ l3=l4 $\vee$ l3=l5 $\vee$ l4=l5): è la parte duale del vincolo precedente e quindi la seconda parte del vincolo 1, esprime il fatto che ogni parola deve contenere al più 4 simboli uguali a C oppure uguali a G.

4. $\forall$ w1, w2 words_idx(w1) $\wedge$ words_idx(w2) $\wedge$ w1$\neq$w2 $\rightarrow$ $\exists$ l1, l2, l3, l4, z1, z2, z3, z4 (l1 $\neq$l2 $\wedge$ l1 $\neq$l3 $\wedge$ l1 $\neq$l4 $\wedge$ l2 $\neq$l3 $\wedge$ l2 $\neq$l4 $\wedge$ l3 $\neq$l4) $\wedge$ ($\neg$words(w1, l1,z1) $\vee$ $\neg$words(w2, l1,z1)) $\wedge$ ($\neg$words(w1, l2, z2) $\vee$ $\neg$ words(w2, l2,z2)) $\wedge$ ($\neg$words(w1, l3,z3) $\vee$ $\neg$ words(w2, l3,z3)) $\wedge$ ($\neg$words(w1, l4,z4) $\vee$ $\neg$ words(w2, l4,z4)): rappresenta il vincolo 2 ed esprime il fatto che per ogni coppia di parole diverse w1 e w2 devono esistere almeno quattro indici di lettera differenti (cioè quattro posizioni) tali che se w1 contiene una lettera in una posizione allora in w2 non si avrà la stessa lettera nella stessa posizione.

5. $\forall$ w1, w2, words_idx(w1) $\wedge$ words_idx(w2) $\rightarrow$ $\exists$ l1, l2, l3, l4 , z1, z2, z3, z4,z5,z6,z7,z8 (l1 $\neq$l2 $\wedge$ l1 $\neq$l3 $\wedge$ l1 $\neq$l4 $\wedge$ l2 $\neq$l3 $\wedge$ l2 $\neq$l4 $\wedge$ l3 $\neq$l4) $\wedge$ ($\neg$words(w1, 9 –l1,z1) $\vee$ $\neg$ (words(w2, l1, z2)$\wedge$ z1=replacemet(z2))) $\wedge$($\neg$words(w1, 9 –l2,z3) $\vee$ $\neg$ (words(w2, l2, z4)$\wedge$ z3=replacemet(z4))) $\wedge$($\neg$words(w1, 9 –l3,z5) $\vee$ $\neg$ (words(w2, l3, z6)$\wedge$ z5=replacemet(z6))) $\wedge$($\neg$words(w1, 9 –l4,z7) $\vee$ $\neg$ (words(w2, l4, z8)$\wedge$ z7=replacemet(z8))) : è l'espressione del terzo vincolo da soddisfare ed ha una struttura simile al precedente, con la differenza che si deve verificare la diversità tra il reverse della prima parola e il complemento della seconda. Per esprimere il reverse si contano gli indici dalla fine anziché dall'inizio, mentre per esprimere il complemento si fa uso della funzione ausiliaria replacement.

Social Golfer:
In un golf club ci sono 32 giocatori che giocano una volta a settimana in 8 gruppi da 4. Il problema consiste nel trovare uno schedule per quante più settimane possibili, tale che non ci siano due giocatori che giocano nello stesso gruppo per più di una volta.
Consideriamo la versione decisionale del problema che assegna il numero di settimane, e passa in input il numero di giocatori e la dimensione del gruppo.
I vincoli da soddisfare sono quindi:

1. Ogni gruppo ha esattamente 4 giocatori;
2. Ogni coppia di giocatori si incontra al massimo una volta

$\exists$ Assign $\forall$ X, Y, Z Assign(X,Y,Z) $\rightarrow$ player(X) $\wedge$ group(Y) $\wedge$ week(Z) $\wedge$ $\longrightarrow$ Spazio di Ricerca

$\forall$ g, w group(g) $\wedge$ week(w) $\rightarrow$

$\exists$ p1, p2, p3, p4 (p1$\neq$p2 $\wedge$ p1$\neq$p3 $\wedge$ p1$\neq$p4 $\wedge$ p2$\neq$p3 $\wedge$ p2$\neq$p4 $\wedge$ p3$\neq$p4) $\wedge$ (player(p1) $\wedge$ player(p2) $\wedge$ player(p3) $\wedge$ player(p4) $\wedge$ Assign(p1, g, w) $\wedge$ Assign(p2, g, w) $\wedge$ Assign(p3, g, w) $\wedge$ Assign(p4, g, w)) $\wedge$

Vincolo 1: ogni gruppo ha esattamente 4 giocatori

$\forall$ g, w, p1, p2, p3, p4, p5 player(p1) $\wedge$ player (p2) $\wedge$ player (p3) $\wedge$ player (p4) $\wedge$ player(p5)$\wedge$ group(g) $\wedge$ week(w) $\wedge$ Assign(p1, g, w) $\wedge$ Assign(p2, g, w) $\wedge$ Assign(p3, g, w) $\wedge$ Assign(p4, g, w)) $\wedge$ Assign(p5, g, w) $\rightarrow$ (p1=p2 $\vee$ p1=p3 $\vee$ p1=p4 $\vee$ p1=p5 $\vee$ p2=p3 $\vee$ p2=p4$\vee$ p2=p5$\vee$ p3=p4$\vee$ p3=p5 $\vee$ p4=p5)

$\forall$ p1, p2, w1, w2 player(p1) $\wedge$ player(p2) $\wedge$ week(w1) $\wedge$ week(w2) $\wedge$ p1$\neq$p2 $\wedge$ w1$\neq$w2 $\rightarrow$ $\exists$ g group(g) $\wedge$(($\neg$(Assign(p1, g, w1) $\wedge$ Assign(p2, g, w1)) $\vee$ $\neg$(Assign(p1, g, w2)$\wedge$ Assign(p2, g, w2)))

Vincolo 2: ogni coppia di giocatori si incontra al più una volta

Esaminiamo i vincoli uno ad uno:

1. $\forall$ X, Y, Z Assign(X,Y,Z) $\rightarrow$ player(X) $\wedge$ group(Y) $\wedge$ week(Z): in ogni assegnazione Assign il primo argomento è un giocatore, il secondo è un gruppo ed il terzo è una settimana, questo vincolo serve a specificare i tipi degli oggetti coinvolti.

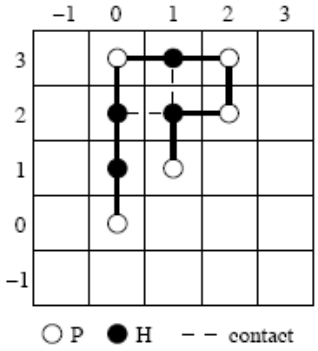
2. $\forall$g, w group(g) $\wedge$ week(w) $\rightarrow$ $\exists$ p1, p2, p3, p4 (p1$\neq$p2 $\wedge$ p1$\neq$p3 $\wedge$ p1$\neq$p4 $\wedge$ p2$\neq$p3 $\wedge$ p2$\neq$p4 $\wedge$ p3$\neq$p4) $\wedge$ (player(p1) $\wedge$ player(p2) $\wedge$ player(p3) $\wedge$ player(p4) $\wedge$ Assign(p1, g, w) $\wedge$ Assign(p2,g, w) $\wedge$ Assign(p3, g, w) $\wedge$ Assign(p4, g, w)) esprime la prima parte del vincolo 1 cioè che ogni gruppo ha almeno 4 giocatori.

3. $\forall$ g, w, p1, p2, p3, p4, p5 player(p1) $\wedge$ player (p2) $\wedge$ player (p3) $\wedge$ player (p4) $\wedge$ player(p5) $\wedge$ group(g) $\wedge$ week(w) $\wedge$ Assign(p1, g, w) $\wedge$ Assign(p2, g, w) $\wedge$ Assign(p3, g, w) $\wedge$ Assign(p4,g,w)) $\wedge$ Assign(p5, g, w) $\rightarrow$ (p1=p2 $\vee$ p1=p3 $\vee$ p1=p4 $\vee$ p1=p5 $\vee$ p2=p3 $\vee$ p2=p4$\vee$ p2=p5$\vee$ p3=p4$\vee$ p3=p5 $\vee$ p4=p5) esprime la seconda parte del vincolo 2: ogni gruppo ha al più 4 giocatori, cioè se esistono 5 assegnazioni per lo stesso gruppo nella stessa settimana almeno due di esse sono relative allo stesso giocatore.

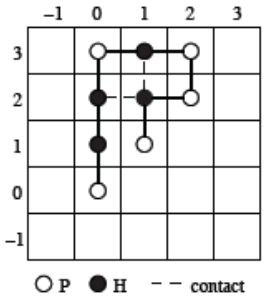
4. $\forall$ p1, p2, w1, w2 player(p1) $\wedge$ player(p2) $\wedge$ week(w1) $\wedge$ week(w2) $\wedge$ p1$\neq$p2 $\wedge$ w1$\neq$w2 $\rightarrow \exists$ g group(g) $\wedge$((¬(Assign(p1,g,w1) $\wedge$ Assign(p2,g,w1)) $\vee$ ¬(Assign(p1, g, w2)$\wedge$ Assign(p2,g,w2))) esprime il secondi vincolo da soddisfare, cioè che per ogni coppia di giocatori diversi e per ogni coppia di settimane diverse deve esistere un gruppo tale che se i giocatori p1 e p2 sono entrambi assegnati al gruppo g nella settimana w1 allora non lo saranno nella settimana w2. (Il vincolo è espresso come $\neg\varphi\vee\neg\psi$ anziché come $\varphi\rightarrow\neg\psi$).

Protein Folding:
Data una struttura primaria di proteine, cioè una sequenza di aminoacidi (20 tipi), trovare una struttura terziaria ovvero una conformazione completa 3D di una proteina.
Consideriamo il problema semplificato: 2D HP-Protein Folding
Da 20 tipi di aminoacidi passiamo a 2 classi (H e P), e dal sistema 3D passiamo a 2D e discreto.

2D HP-Protein folding:
Data una sequenza di aminoacidi della proteina, cioè una stringa su {H, P} di lunghezza n, il problema consiste nel trovare una figura connessa per essa su una griglia 2D, con coordinate nell'intervallo [-(n-1), (n-1)] a partire da (0,0),in modo da non creare incroci e tale che il numero di contatti, cioè il numero di coppie non sequenziali di H, per le quali la distanza Euclidea della posizione è 1, è massimizzato. L'energia si calcola come l'opposto del numero di contatti.
Esempio:



La figura mostra una possibile conformazione per la proteina "PHHPHPPHP" con due contatti e con energia pari a -2.
Lo spazio di ricerca è formato dalla sequenza di mosse $m_1, \ldots, m_{n-1} \in \{N, S, E, W\}$ a partire da (0,0)



Moves	
index	dir
1	N
2	N
3	N
4	E
...	...

Il vincolo da soddisfare è il seguente: non ci devono essere incroci

$\exists$ Move $\in$ index $\rightarrow$ {N,S, E,W} (total),
$\exists$X,Y $\in$ index $\rightarrow$[-n+1, n-1] (total)

Predicati Guessed S: mosse delle stringhe e posizioni assolute

X(0, 0) $\wedge$ Y(0,0) $\wedge$

Posizione iniziale (0,0)

$\forall$ I, I' index (I) $\wedge$ index (I') $\wedge$ I' = I – 1 $\rightarrow$
$\forall$ D, X,Y, X', Y' Move(I',D) $\wedge$ X(I,X) $\wedge$X(I',X') $\wedge$ Y (I, Y) $\wedge$Y (I', Y')$\rightarrow$
D = N $\rightarrow$ X = X'$\wedge$ Y = Y' + 1 $\wedge$
D = S $\rightarrow$ X = X' $\wedge$ Y = Y' – 1 $\wedge$
D = E $\rightarrow$ X = X' + 1 $\wedge$ Y = Y' $\wedge$
D = W $\rightarrow$ X = X' – 1 $\wedge$ Y = Y' $\wedge$

Vincoli di movimento: definisce X ed Y in termini di Move

$\forall$I, I', X, X', Y, Y' I $\neq$ I' $\wedge$ X(X, I) $\wedge$ X(X', I') $\wedge$ Y (Y, I) $\wedge$Y (Y', I')
$\rightarrow$ X $\neq$ X' $\vee$ Y $\neq$ Y'$\wedge$

Vincolo di non incrocio (in termini di X ed Y)

$$\left| \left\{ \begin{array}{l} \langle I, I' \rangle \mid \text{index}(I) \wedge \text{index}(I') \wedge (I + 1 < I') \wedge \text{elem}(I,H) \wedge \text{elem}(I',H) \wedge \\ \forall X, X', Y, Y' \; X(X, I) \wedge X(X', I') \wedge Y(Y, I) \wedge Y(Y', I') \wedge \\ |X - X'| + |Y - Y'| = 1 \end{array} \right\} \right| \geq k$$

Versione decisionale: soglia minima sul numero di contatti

4. Ambiti di applicazione: ESO vs CSP- SAT

Una formula ESO può essere usata per modellare la struttura di un problema di decisone (la specifica), indipendentemente dai dati di input (l'istanza). I sistemi correnti per la programmazione a vincoli che supportano la modellazione dichiarativa operano una netta separazione tra la specifica del problema e la sua istanza, offrendo dei linguaggi di specifica che sono essenzialmente ESO, infatti tutti i linguaggi e i sistemi usati attualmente per la modellazione dichiarativa a vincoli, possono essere visti come un'estensione di ESO, se pur con una sintassi più ricca e costrutti più complessi.

Quando vengono assegnati dei dati di input (fissando un'estensione per i simboli di predicato nell'insieme R) si ottiene un classico problema CSP (Constraint Satisfaction Problem), cioè si ottiene una tripla < X, C, D> dove:

- X è un insieme di variabili linearmente ordinato;
- D è un insieme linearmente ordinato di valori di dominio, uno per ogni variabile;
- C è un insieme di vincoli, ognuno dei quali è definito su un sottoinsieme linearmente ordinato delle variabili e codifica un sottoinsieme del prodotto cartesiano dei rispettivi domini.

Il CSP si occupa di problemi di assegnazione.
Nei problemi di assegnazione si hanno le seguenti caratteristiche:

- non c'è l'interesse ad ottenere un percorso risolvente;
- non c'è (generalmente) un costo associato ad ogni passo;
- non si possiede uno stato obiettivo (possedere uno stato obiettivo coincide con l'aver risolto il problema).

I problemi CSP possono essere divisi in due gruppi: CSP finiti, ovvero quella classe di problemi con un dominio finito di valori, e CSP infiniti, quei problemi con un dominio infinito di valori; questa seconda categoria caratterizza i problemi affrontati nella programmazione lineare.
SAT è un problema CSP finito, e ogni CSP può essere riducibile a SAT. Risulta quindi chiaro che i CSP finiti ricadono nella classe di complessità NP. Un esempio classico è il Knapsack Problem, noto problema NP-completo.

Il Constraint Programming è un insieme di metodologie che mirano alla risoluzione dei CSP e richiede 3 scelte progettuali:

- modello (definito con framework CSP)
- algoritmo
- euristica

Ognuna di queste scelte influenza l'efficienza della risoluzione.

Vediamo in un piccolo esempio come il problema della soddisfacibilità (SAT) può essere modellato come un CSP:

$$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2)$$

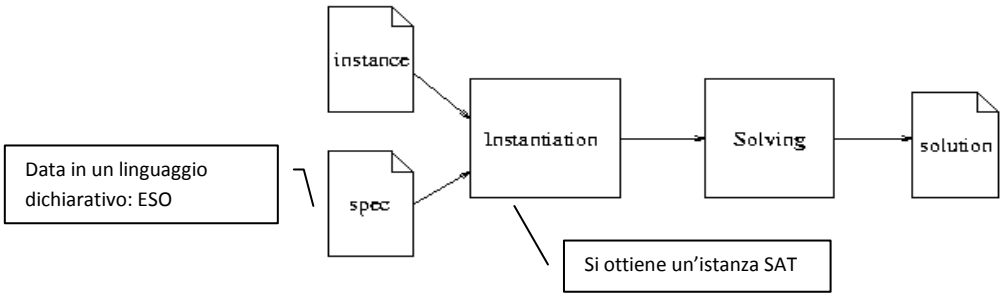
- Variabili: x_1, x_2, x_3
- Domini D_i = {true,false}
- Vincoli: il valore di ogni clausola deve essere TRUE

Quindi:

- $(x_1 \vee x_2 \vee x_3)$ = **TRUE**
- $(\neg x_1 \vee x_2 \vee x_3)$ = **TRUE**
- $(x_1 \vee \neg x_2)$ = **TRUE**

Una soluzione di un CSP è un'assegnazione di valori del dominio alle variabili, che soddisfa tutti i vincoli.
Il limiti principali dei CSP risiedono nel fatto che uno stesso problema può essere codificato da differenti CSP, con diverse scelte e diversa efficienza, inoltre anche il fatto che un CSP include i dati di input può costituire un limite in determinati casi.

La programmazione a vincoli sta diventando molto importante per risolvere varie classi di problemi, dal momento che offre il vantaggio di una prototipazione veloce e un'alta dichiaratività esibita dai modelli dei problemi.
I sistemi correnti per la soluzione dei problemi a vincoli permettono al programmatore di modellare il problema in modo molto dichiarativo, supportando una netta separazione della specifica dalle sue istanze.



Tale separazione permette al programmatore di concentrarsi sugli aspetti strutturali e combinatori del problema prima di avere a che fare con i dati di input, e quindi permette una modellazione del problema a livello più alto.
Nello schema mostrato ESO viene usato per esprimere le specifiche del problema e dopo l'istanziazione si ottiene un'istanza SAT.

Nonostante tutto, il modello di problema ottenuto in questo modo spesso non è efficiente e richiede un ulteriore ragionamento per la riformulazione e per accelerare il processo di soluzione.
A questo scopo sono stati proposti diversi approcci in letteratura, come la scoperta di simmetrie, l'aggiunta di vincoli implicati, la cancellazione o astrazione di alcuni dei vincoli oppure l'uso di modelli ridondanti, il tutto al fine di incrementare la propagazione dei vincoli; la maggior parte di questi approcci sono pensati per uno specifico problema a vincoli, oppure agiscono a livello di istanza ma si è lavorato molto poco al livello della specifica del problema. In realtà molte delle proprietà dei problemi a vincoli destinati all'ottimizzazione, dipendono fortemente dalla struttura del problema. Riconoscere a livello delle specifiche le proprietà e le dipendenze funzionali che possono esserci tra le variabili nelle specifiche dei problemi dichiarativi, porterebbe benefici sia da un punto di vista metodologico, che da un punto di vista di efficienza. Data una specifica, una variabile è detta essere funzionalmente dipendente dalle altre se, per ogni soluzione di ogni istanza, il suo valore è determinato da quelli assegnati alle altre.

Un ulteriore problema che si può incontrare quando si usa un formalismo dichiarativo come CSP, o come i linguaggi di modellazione dichiarativi a vincoli, è che il programmatore perde il potere di distinguere le variabili i cui valori devono essere trovati attraverso una vera ricerca, dalle quelle i cui valori possono essere calcolati a partire dalle altre, dal momento che tutte le variabili diventano della stessa natura.
Di conseguenza lo spazio di ricerca esplorato dal sistema può diventare inefficientemente molto più largo, e il programmatore potrebbe dover richiedere informazioni aggiuntive per distinguerle, riducendo così di molto la dichiaratività della specifica. L'abilità del sistema di riconoscere automaticamente se una variabile è funzionalmente dipendente dalle altre diventa di grande importanza dal punto di vista dell'efficienza, dal momento che può portare a riduzioni significative dello spazio di ricerca.
Cercare le dipendenze funzionali a livello delle specifiche piuttosto che dopo l'istanziazione, può essere molto più naturale dal momento che questo problema dipende strettamente dalla struttura del problema.

5. Dipendenze funzionali

Quando si deve creare l'istanza SAT di un problema a partire dalle specifiche può essere utile riconoscere quali sono le variabili che dipendono funzionalmente dalle altre, e portare avanti questa informazione. Intuitivamente il riconoscimento delle dipendenze funzionali permette di risparmiare spazio e tempo durante la fase di branching: è facile rendersi conto che se una variabile b dipende funzionalmente da una variabile a, nel momento in cui si determina il valore di a, automaticamente sarà assegnato anche un valore a b. E' più conveniente, quindi, ritardare il più possibile la scelta di b come variabile di branching, aspettando di vedere se si riesce a settare un valore per essa in modo automatico. Per comprendere meglio l'utilità di quanto appena detto si pensi al caso in cui il problema in esame presenta una grande quantità di variabili; riconoscere quante più possibili dipendenze funzionali permette di diminuire di gran lunga la profondità dell'albero di ricerca, e quindi di velocizzare molto la procedura di soddisfacibilità.

Prendiamo in esempio il problema dello shop scheduling, intuitivamente le uniche variabili indipendenti sono quelle di inizio attività, poiché una volta fornita la matrice che contiene la durata di ogni processo, è chiaro che le variabili di fine attività, deadline e occupazione della macchina dipendono funzionalmente dalle prime dette.
Ancora, si può prendere in esame il problema "protein folding" mostrato in precedenza, si può notare che quando vengono scelte le mosse (variabili Move) sono determinate anche le posizioni assolute (variabili X ed Y), quindi 2n variabili (X, Y) sono definite da n-1 variabili (Move). L'algoritmo di backtracking dovrebbe evitare di fare branching sulle variabili X ed Y.

Il problema di trovare le dipendenze funzionali sulle specifiche ESO si riduce al verificare le proprietà semantiche della formula al primo ordine, è possibile usare risultati e tecniche conosciuti per meccanizzare questo lavoro.
Una definizione formale che caratterizza le dipendenze funzionali tra predicati indovinati in una specifica ESO è la seguente:

<u>Definizione 5.1</u> (dipendenza funzionale) *Data una specifica di problemi su uno schema di input* ***R***

$$\psi \doteq \exists SP \ \phi(S,P,R)$$

nella quale l'insieme dei predicati indovinati è suddiviso in ***S*** *e* ***P****, i predicati in* ***P*** *dipendono funzionalmente da quelli in* ***S*** *se, per ogni istanza* ***I*** *di* ***R*** *e per ogni coppia di interpretazioni* $< \Sigma, \Pi >$, $< \Sigma', \Pi' >$ *di* ***(S,P)****, si ha che se*

- $< \Sigma, \Pi > \neq < \Sigma', \Pi' >$ *e*
- $\Sigma, \Pi, I \models \phi$ *e*
- $\Sigma', \Pi', I \models \phi$

Allora $\Sigma \neq \Sigma'$.

Questa definizione afferma che ***P*** dipende funzionalmente da ***S*** se si è nel caso in cui, indipendentemente dalle istanze, ogni coppia di soluzioni distinte di **ψ** deve differire sui predicati in **S**, cioè se non esistono due soluzioni differenti di **ψ** che coincidono sulle estensioni per i predicati in **S.**

Il problema di cercare se un sottoinsieme dei predicati indovinati in una specifica è funzionalmente dipendente dai rimanenti, si riduce a verificare delle proprietà semantiche della formula del primo ordine, in altre parole vale il seguente teorema:

<u>Teorema 5.1</u>: *Sia* $\psi \doteq \exists SP \ \phi(S,P,R)$ *la specifica di un problema con schema di input* ***R****. I predicati indovinati nell'insieme P dipendono funzionalmente da quelli in S se e solo se la formula del primo ordine*

$$[\phi(S,P,R) \wedge \phi(S',P',R') \wedge SP \neq S'P'] \rightarrow S \neq S'$$

è una tautologia.

Sfortunatamente il problema di verificare se un insieme di predicati è funzionalmente dipendente da un altro è indecidibile, come afferma il seguente teorema:

<u>Teorema 5.2</u>: *Data una specifica ESO su uno schema di input* ***R****, e una partizione (****S****,* ***P****) dei sui predicati indovinati, il problema di verificare se P dipende funzionalmente da S non è decidibile.*

Da questo teorema si può dedurre che non è sempre possibile meccanizzare il compito di stabilire se delle date dipendenze sussistono o meno, e questo può essere un problema principale nei processi automatici di riformulazione volti ad ottimizzare le specifiche dichiarative fornite da un utente, per renderle più efficientemente risolvibili.
Usando ESO come linguaggio di modellazione, si caratterizzano formalmente le dipendenze in termini di logica del primo ordine, portando alla possibilità di usare strumenti automatici per meccanizzare il compito del loro riconoscimento. Nonostante l'indecidibilità del problema generale, questo approccio è possibile in molto casi pratici, e può portare a un miglioramento delle performance del solutore.

Un approccio usato per il trattamento delle dipendenze funzionali in ESO è un metodo non invasivo e facilmente automatizzabile chiamato DDP (Delay branches on Dependent Predicates) tale metodo prevede di ritardare il branch sui predicati dipendenti, in altre parole si forza un ordine di preferenza per il branch sui predicati indovinati e si lasciano tutte le altre decisioni alle strategie di default.

Data una formula del tipo illustrato precedentemente $\exists \ S \ P \ \phi(S,P,R)$ dove P dipende da S, la procedura di ricerca DDP prevede:

- Effettua prima il branch sui predicati in S
- Poi effettua il branch sui predicati in P.

Tale metodo si applica in modo trasparente ad ogni euristica, può essere derivato autonomamente dal sistema e può essere migliorato con ottimizzazioni specifiche di ogni problema.
Applicando DDP i solutori che effettuano il backtracking hanno delle performance migliori che non vengono generalmente eguagliate con delle procedure di ricerca ad hoc, inoltre DDP può recuperare un cattivo modello e tenta di neutralizzare l'overhead dovuto alla presenza di dipendenze a volte necessarie.

6. ESO vs OPL

Uno dei linguaggi dichiarativi di modellazione a vincoli più usati è OPL, tale linguaggio è molto simile ad ESO.
OPL è fornito da Ilog OPLSTUDIO (cfr. http://www.ilog.com/products/oplstudio).
OPL (Optimization Programming Language) è un linguaggio di modellazione per l'ottimizzazione combinatoria, che semplifica la formulazione e la soluzione dei problemi di ottimizzazione. OPL fornisce un supporto sia per gli aspetti di ricerca che per gli aspetti di modellazione di un problema a vincoli, in altre parole permette all'utente di specificare non solo il problema ma anche la procedura di ricerca pensata per il problema. OPL permette di affrontare i problemi con un approccio ibrido dal momento che è basato sull'idea di esprimere le caratteristiche del problema sia dal punto di vista della programmazione matematica che da quello della programmazione a vincoli, in altre parole permette di esprimere specifiche flessibili per la ricerca come avviene per la programmazione a vincoli, ma anche specifiche di alto livello dei programmi e algoritmi specializzati di ottimizzazione tipici della programmazione matematica; in questo modo si migliora l'espressività e la leggibilità del problema. OPL colma il divario tra modellazione e soluzione del problema.

Una specifica OPL è fatta essenzialmente di 5 parti, di cui due opzionali:

- Dichiarazione dello schema di istanza, formata da nome e tipo dei parametri e simboli di relazione.
- Dichiarazione dei predicati indovinati che codifica lo spazio di ricerca ed è indicata tramite la parola chiave "var"; OPL permette che i predicati indovinati siano tipizzati e supporta le funzioni per mezzo di array.
- Definizione (opzionale) della funzione obiettivo indicata dalle parole chiave "maximize" o "minimize".
- Specifica dei vincoli, in un linguaggio molto simile alla logica del primo ordine.
- Definizione (opzionale) della procedura di ricerca, che indica quale strategica seguire per la ricerca, nel caso tale definizione non venga data si segue la strategia di default.

OPL presenta molte similitudini con ESO, in particolare la parola chiave "var" gioca esattamente lo stesso ruolo di un quantificatore esistenziale al secondo ordine in ESO, inoltre i vincoli corrispondono alla parte del primo ordine in una specifica ESO.
In ESO però non è possibile esprimere le funzioni obiettivo, quindi se presenti, è necessario considerare le versioni decisionali dei problemi o i problemi di ricerca associati.
Riportiamo alcuni esempi di problemi modellati in OPL e in ESO.

Ramsey Problem:

OPL

```
//definizione dello schema di istanza
int+ nodes = ...;
int+ maxcolors = ...;
range allColors 1..maxcolors;
range allNodes 1..nodes;
struct edge { allNodes n;  allNodes m; };
{edge} edges = {<n,m> | n,m in allNodes : n < m};
```

```
// Search space: possibili assegnazioni di colori ai nodi
 var allColors coloring[edges];
```

Non esprimibile in ESO

```
// Objective function: Minimizza il numero di colori usati
minimize (sum (c in allColors) ((sum (e in edges) (coloring[e] = c)) > 0))
```

```
subject to {
  // vincoli: per ogni tripla di nodi differenti, non tutti gli archi hanno lo
  //stesso colore
  forall(x,y,z in allNodes: (x < y < z)) {
    not (coloring[<x,y>] = coloring[<x,z>] = coloring[<y,z>]);
  };
};
```

ESO

In ESO si esprime la versione decisionale del problema, cioè dati un numero di nodi e un numero di colori verifica se esiste una colorazione che rispetta le caratteristiche desiderate.

Schema di Istanza

$\exists$ Col $\forall$ X, Y , Z Col (X,Y, Z) $\rightarrow$ node(X) $\wedge$ node(Y) $\wedge$ color(Z) $\wedge$

$\forall$ X, Y $\exists$Z Col(X,Y,Z) $\wedge$

$\forall$ X,Y,C,C' Col(X,Y,C) $\wedge$Col(X,Y,C') $\rightarrow$ C=C' $\wedge$

Spazio di Ricerca

$\forall$ X, Y,Z ,C,C',C" (X$\neq$Y $\wedge$ X$\neq$Z $\wedge$ Y $\neq$ Z $\wedge$ Col(X,Y,C) $\wedge$ Col(X,Z,C') $\wedge$ Col(Y, Z,C"))

$\rightarrow$ (C$\neq$ C' $\vee$ C$\neq$ C" $\vee$ C'$\neq$ C")

Vincoli:
Se esiste un triangolo di nodi allora al massimo due archi hanno lo stesso colore

Word design:

OPL

```
//definizione dello schema di istanza
int+ nbwords = ...;
enum alphabet = {A,C,G,T};
int+ length = 8;
alphabet replacement[alphabet] = #[A:T,C:G,G:C,T:A]#;
range word_idx 1..nbwords;
range letter_idx 1..length;
```

```
// Search space: The possible sets of nbwords words of length 8 with components
// in {A,C,G,T}
var alphabet words[word_idx,letter_idx];
solve {
   // C1: Each word in the guessed set has 4 symbols from {C,G}
   forall(w in word_idx) {
      sum(p in letter_idx)(words[w,p]=C \/ words[w,p]=G) = 4;
   };
   // C2: Each pair of distinct words in the guessed set differ in at least 4
//positions
   forall(v,w in word_idx: v<>w) {
      sum(p in letter_idx)(words[v,p]<>words[w,p]) >= 4;
   };
   // C3: Each pair of words x and y in the guessed set (where x and y may be
//identical) are such that R(x) and C(y) differ in at least 4 positions (see
//problem specification for definitions of R(x) and C(x))
   forall(v,w in word_idx) {
      sum(p in letter_idx)(words[v,length+1-p] <> replacement[words[w,p]]) >= 4;
   };
};
```

ESO

alphabet replacement(alphabet) = # [A=T, C=G, G=C, T=A]#;

∃ words ∀ X, Y, Z words(X,Y,Z) → words_idx(X) ∧ letters_idx(Y) ∧letter(z) ∧

∀ w words_idx(w) → ∃l1, l2, l3, l4, z1, z2 letters_idx(l1) ∧ letters_idx(l2) ∧ letters_idx(l3) ∧ letters_idx(l4)∧ letter(z1) ∧ letter(z2) ∧

(words(w, l1, z1) ∨ words(w, l1, z2)) ∧ (words(w, l2,z1) ∨ words(w, l2,z2) ∧ (words(w, l3,z1) ∨ words(w, l3,z2)) ∧ (words(w, l4,z1) ∨ words(w, l4,z2)) ∧ (z1=C) ∧(z2=G)∧

∀ w, l1, l2,l3,l4,l5, z1, z2 words_idx(w) ∧ letters_idx(l1) ∧ letters_idx(l2) ∧ letters_idx(l3) ∧ letters_idx(l4) ∧ letters_idx(l5) ∧ letter(z1) ∧ letter(z2) ∧

(words(w, l1,z1) ∨ words(w, l1, z2)) ∧ (words(w, l2,z1) ∨ words(w, l2,z2)) ∧ (words(w, l3,z1) ∨ words(w, l3,z2)) ∧ (words(w, l4,z1) ∨ words(w, l4,z2)) ∧ (words(w, l5,z1) ∨ words(w, l5,z2)) ∧ (z1=C) ∧(z2=G) →

(l1=l2 ∨ l1=l3 ∨ l1=l4 ∨ l1=l5 ∨ l2=l3 ∨ l2=l4 ∨ l2=l5 ∨ l3=l4 ∨ l3=l5 ∨ l4=l5) ∧

∀ w1, w2 words_idx(w1) ∧ words_idx(w2) ∧ w1≠w2 →

∃ l1, l2, l3, l4, z1, z2, z3, z4 (l1 ≠l2 ∧ l1 ≠l3 ∧ l1 ≠l4 ∧ l2 ≠l3 ∧ l2 ≠l4 ∧ l3 ≠l4) ∧

(¬words(w1, l1,z1) ∨ ¬words(w2, l1,z1)) ∧ (¬words(w1, l2, z2) ∨ ¬ words(w2, l2,z2)) ∧ (¬words(w1, l3,z3) ∨ ¬ words(w2, l3,z3)) ∧ (¬words(w1, l4,z4) ∨ ¬ words(w2, l4,z4)) ∧

$$
\begin{aligned}
&\forall\, w1, w2,\ \ words\_idx(w1) \wedge words\_idx(w2) \ \rightarrow \\
&\exists\, l1, l2, l3, l4\ , z1, z2, z3, z4, z5, z6, z7, z8\ (l1 \neq l2 \wedge l1 \neq l3 \wedge l1 \neq l4 \wedge l2 \neq l3 \wedge l2 \neq l4\ \wedge l3 \neq l4)\ \wedge \\
&\qquad (\neg words(w1, 9 - l1, z1\ ) \vee \neg\ (words(w2, l1, z2\ ) \wedge\ z1 = replacemet(z2)\ )) \wedge \\
&\qquad (\neg words(w1, 9 - l2, z3\ ) \vee \neg\ (words(w2, l2, z4\ ) \wedge\ z3 = replacemet(z4)\ )) \wedge \\
&\qquad (\neg words(w1, 9 - l3, z5\ ) \vee \neg\ (words(w2, l3, z6\ ) \wedge\ z5 = replacemet(z6)\ )) \wedge \\
&\qquad (\neg words(w1, 9 - l4, z7\ ) \vee \neg\ (words(w2, l4, z8\ ) \wedge\ z7 = replacemet(z8)\ )) \wedge
\end{aligned}
$$

7. Tecniche di istanziazione in SAT

Uno degli aspetti importanti del lavoro con i problemi a vincoli è la traduzione di una specifica ESO in un'istanza SAT, per questo scopo esistono dei traduttori che a partire dalle specifiche restituiscono un'istanza CNF e poi un file DIMACS da passare come input a qualche solutore.
Il traduttore che noi abbiamo preso in considerazione è Spec2Sat (cfr. http://www.dis.uniroma1.it/cadoli/research/projects/NP-SPEC/code/spec2SAT).
Il cuore di questo sistema è il file *translate* che istanzia un problema di soddisfacibilità proposizionale. Un'istanza del problema originale è tradotta in una formula T di logica proposizionale in forma normale congiuntiva che è soddisfacibile se e solo se il problema originale ha una soluzione.
Una specifica S del problema è un insieme di metaregole che definiscono lo spazio di ricerca, più un insieme di regole che definiscono la condizione di ammissibilità. Regole e metaregole sono trasformate in un insieme di clausole T. La traduzione delle regole è basata sulla loro istanziazione ground sull'universo di Herbrand.
Le metaregole servono a definire sinteticamente un insieme di regole che descrive la tipologia dello spazio di ricerca: funzione, permutazione etc.; intuitivamente possono essere considerate delle scorciatoie per definire sinteticamente un insieme di vincoli. Le metaregole non aumentano il potere espressivo del linguaggio, visto che si possono definire a partire da semplice predicati guessed più dei vincoli. Ad esempio una funzione può essere formalizzata come una relazione più il vincolo di monodromicità (al più un valore del codominio per un valore del dominio).
Nello specifico si dividono in:

- Permutation
- Partition
- Subset
- intFunction

Le regole sono formate da un "head" e un "body": il "body" gioca il ruolo di implicante e l'"head" di implicato.
Spec2Sat prende in input, come specifica, un file scritto nel linguaggio NP-SPEC; NP-SPEC è una estensione non deterministica del DATALOG, dal quale eredita la sintassi, con la possibilità di un uso limitato di alcuni predicati al secondo ordine di forma predefinita. Lo stile dichiarativo della programmazione in NP-SPEC è molto simile a quello del DATALOG e, di conseguenza, è facile estendere il programma per inserire ulteriori vincoli. Nonostante il DATALOG sia un linguaggio "a regole", in realtà anche NP-SPEC (come virtualmente tutti i linguaggi di modellazione) si può vedere come una variante sintattica di ESO. Il potere espressivo di NPSPEC permette di catturare esattamente NP, come ESO.
Ogni specifica è formata da 2 sezioni: una sezione *DATABASE* che rappresenta l'istanza specifica del problema e dove vengono definite le costanti; e una sezione *SPECIFICATION* che contiene la dichiarazione dello spazio di ricerca, che corrisponde alla definizione dei guessed predicates. In questa sezione vengono definiti i vincoli che un punto dello spazio di ricerca deve soddisfare per essere una soluzione dell'istanza del problema. Tali vincoli sono dichiarati tramite un programma DATALOG che può includere i sei operatori relazionali di base e letterali negativi.

Per quanto concerne la sintassi NPSPEC offre operatori di aggregazione SQL come somma, count, min e max.
Per spiegare meglio come è formata la specifica di un problema riprendiamo l'esempio del graph 3-coloring visto precedentemente. Tale problema è NP-completo. Prendiamo come esempio il caso in cui l'istanza sia un grafo con 6 nodi e 7 archi, che sia 3-colorabile. Si ottiene la seguente specifica:

DATABASE
node = {1,2,3,4,5,6 };
edge = {(1,2), (1,3), (2,3), (6,2), (6,5), (5,4), (3,5)};

SPECIFICATION
Partition(node, coloring, 3);
fail ← *edge(X,Y), coloring(X,C), coloring(Y,C).*

Le clausole nella sezione *DATABASE* specificano un insieme di fatti, cioè delle formule ground atomiche al primo ordine senza funzioni, che definiscono l'istanza.
Le regole nella sezione *SPECIFICATION* sono clausole universalmente quantificate che specificano la soluzione del problema.
La clausola *Partition(node, coloring, 3)* ha il seguente significato:

- il predicato *coloring*, che non compare nella sezione *DATABASE* è implicitamente dichiarato di arità 2;
- l'estensione di *coloring* può essere ogni insieme del tipo:
  $\{<n,c> \mid n$ è nell'estensione di *node*, $c \in \{1,2,3\}\}$.

L'estensione di *coloring* che rende *fail* true non corrisponde ad una soluzione.

Per maggiore chiarezza forniamo anche la specifica del Ramsey Problem:

DATABASE
node = {1,2,3,4,5 };
edge = {(1,2), (1,3), (1,4), (1,5),(2,3), (2,4), (2,5), (3,4), (3,5),(4,5)};

SPECIFICATION
Partition({1..N_EDGES}, coloring, N_COLORS).
fail <-- edge(X,A,B), edge(Y,B,C), edge(Z,A,C), coloring(X,Col), coloring(Y,Col), coloring(Z,Col).

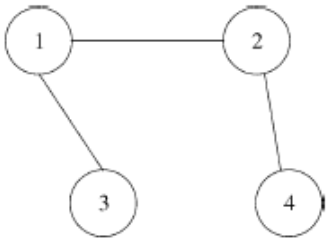
È evidente la corrispondenza tra le due sezioni di una specifica in NPSPEC e gli insiemi R ed S di una formula ESO, infatti:

- la sezione *DATABASE*, usata per specificare l'istanza del problema, è in corrispondenza con l'insieme di predicati R, che definisce lo schema per tutte le istanze di input del problema. Inoltre l'estensione dei predicati dichiarati nella sezione *DATABASE* è sempre come specificato dall'insieme dei fatti.
- La sezione *SPECIFICATION* è in corrispondenza con l'insieme S dei predicati guessed le cui estensioni codificano i punti dello spazio di ricerca per il problema; infatti in questa sezione vengono dichiarati dei predicati usati per rappresentare lo spazio di ricerca del

problema e l'estensione di tali predicati può essere ogni sottoinsieme del prodotto Cartesiano dei corrispondenti domini. In questa sezione vengono dichiarati anche i vincoli da soddisfare.

Tornando a Spec2Sat possiamo dire che il modulo *parser* riceve in ingresso il file di testo contenente la specifica S in NPSPEC e i dati di istanza I, li trasforma e costruisce una sua rappresentazione interna. Come accennato precedentemente poi il modulo Spec2Sat compila $S \cup I$ in una formula CNF T in formato DIMACS e costruisce un oggetto che rappresenta un dizionario che crea una corrispondenza 1 ad 1 tra gli atomi ground della base di Herbrand di $S \cup I$ e le variabili proposizionali del vocabolario. Il file DIMACS è poi dato in input ad un solutore SAT.

La maggior parte del lavoro per creare l'istanza SAT è svolto all'interno del file t*ranslate* in cui i primi passi consistono nel computare le Metarules, i DBPredicates e l'universo di Herbrand a partire dalla specifica, questo permette di costruire il vocabolario di cui sopra.
Tale vocabolario è ottenuto aggiungendo una variabile per ogni predicato menzionato nella sezione DB (variabili di tipo α), una variabile per ogni possibile istanziazione dei predicati indovinati (variabili di tipo β), e una variabile per ogni altro tipo di predicato (variabili di tipo γ).
Tutte le variabili del primo tipo diventano una clausola di T(formula proposizionale) ed affermano che tutto ciò che è espresso in DB è vero. Le clausole che usano le variabili del secondo tipo codificano il significato delle corrispondenti metarules.
Un esempio del vocabolario costruito per il problema del graph 3-coloring che ha come istanza il grafo in figura è il seguente:



NPSPEC atom	variabile	tipo
edge(1,2)	1	α
edge(1,3)	2	α
edge(2,4)	3	α
coloring(1,0)	4	β
coloring(1,1)	5	β
coloring(1,2)	6	β
coloring(2,0)	7	β
coloring(2,1)	8	β
coloring(2,2)	9	β
coloring(3,0)	10	β
coloring(3,1)	11	β
coloring(3,2)	12	β
coloring(4,0)	13	β
coloring(4,1)	14	β
coloring(4,2)	15	β
fail	16	γ

Una volta costruito il vocabolario e computato l'Universo di Herbrand viene richiamato il metodo buildGraph() che costruisce il grafo G della specifica, dove i nodi corrispondono ai predicati della specifica e c'è un arco che va da q a p se e solo se nella specifica è presente una regola in cui p

compare nella head e q nel body. Una volta costruito il grafo i predicati vengono divisi in due insiemi:

- primitivi: sorgenti in G, sono i predicati di DB o i predicati indovinati;
- definiti: compaiono nell'head delle rules.

Una volta costruito il grafo viene invocato il metodo *ComputeInlinePositive()* che calcola l'insieme dei predicati *inline* o generati da letterali positivi, i predicati *inline* sono quelli per i quali si conoscono esattamente tutte le possibili istanziazioni, come ad esempio i predicati di DB o quelli ottenuti dai predicati di DB tramite operatori relazionali, un predicato *inline* può essere sostituito dall'insieme di regole ottenute da tutte le sue possibili istanziazioni. Questo metodo considera tutte le regole della specifica e per ogni regola considera i letterali del body e i predicati associati ad ogni letterale e controlla se possono o meno stare nell'insieme.
A questo punto viene invocato il metodo *SimplifyRules()* che istanzia le regole con i predicati inline, rimuovendo le regole originarie e aggiungendone delle nuove, in altre parole questo metodo considera tutte le regole, cercando quelle che possono essere completamente istanziate, se le trova aggiunge le regole derivanti da tutte le possibili istanziazioni e rimuove la regola originaria; in questo modo si generano delle regole più facili da processare.
Una volta semplificate le regole vengono invocati i metodi *BuildAliveAndTForDBPredicates*() e *BuildAliveForMetarules()*, il primo metodo calcola l'insieme *alive* per i predicati di DB e genera le clausole di T in relazione con DB, il secondo calcola l'insieme *alive* per le metaregole. Gli insiemi *alive* contengono i letterali che possono assumere valori sia veri che falsi, cioè quei letterali per i quali vale la pena calcolare i valori da assumere, ad esempio non conterrà i letterali del tipo Col(n1,n2) per n1 e n2 entrambi nodi. Il calcolo di questi insiemi consente di ridurre il numero di assegnazioni di valori da considerare.
A questo punto si è quasi pronti per processare le regole e a tale scopo viene definito un ordinamento topologico dei nodi del grafo, tramite il metodo *BuildTopologicalSort().* Ogni ordinamento assicura che tutti i predicati nel body di una regola vengano prima dei predicati nell'head.
I predicati vengono processati uno alla volta, tramite il metodo *processRule().* Ogni predicato contribuisce al vocabolario V(T) con un insieme di variabili proposizionali e indirettamente a T con un insieme di clausole. L'ordine in cui i predicati vengono processati è dato in accordo con l'ordinamento topologico di G, cioè nessun nodo è processato prima che tutti i suoi predecessori siano stati processati.
Il processamento permette di creare l'insieme T delle clausole che costituiscono l'istanza SAT del problema.
Una volta creato l'insieme di clausole la traduzione in SAT è completa e nel *main* si richiama il metodo *Cnf2Dimacs()* che provvede a trasformarle in formato DIMACS.

Bibliografia:

- Raymond Reiter: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems; By MIT Press, 2001

- Neil Innermann: Descriptive Complexity; By Springer, 1998

- Toni Mancini, Marco Cadoli: Exploiting functional dependencies in declarative problem specifications.
  Artificial Intelligence, 171(16--17), pages 985--1010, 2007.

- Marco Cadoli, Toni Mancini: Using a theorem prover for reasoning on constraint problems. Applied Artificial Intelligence, 21(4--5), pages 383--404, 2007.

- Toni Mancini: Declarative modeling and Constraint programming.

- Toni Mancini: Declarative constraint modeling and specification-level reasoning.

- Marco Cadoli, Toni Mancini, Fabio Patrizi: SAT as an effective solving technology for constraint problems.

- Marco Cadoli, Giovanbattista Ianni, Luigi Palopoli, Andrea Schaerf, Domenico Vasile: NPSPEC: An Executable Specification Language for Solving All Problems in NP.

Appendice A: Modifiche a MINISAT

Si vuole realizzare un SAT-solver che riduca il proprio spazio di ricerca evitando di fare branching sulle variabili funzionalmente dipendenti. In particolare, si vuole realizzare il seguente algoritmo:

- Considera l'insieme V di tutte le variabili su cui non hai ancora fatto branch.
- Considera il sottoinsieme NO delle variabili in V che:
  a) Sono nella lista delle var. dipendenti, E
  b) sono dipendenti da almeno una variabile in V (quindi non le var. dipendenti solo da variabili già assegnate)
- Se V - NO non e' vuoto, scegli la variabile su cui fare branch tra quelle in V - NO secondo LE SOLITE EURISTICHE
- Altrimenti, scegli la prossima variabile in V (= NO), secondo le solite euristiche.

Si assume di passare al programma un file con una speciale sintassi DIMACS di questo tipo:

c ...header
c ...header
c ...
c END PROPERTIES
c ...
c ...
p cnf ....
....
c START PROPERTIES
c <dimacsvar> DEPENDENT ON <dimacsvar1> <dimacsvar2> <dimacsvarN>
c <dimacsvar> DEPENDENT ON <dimacsvar1> <dimacsvar2> <dimacsvarN>
...
c <dimacsvar> DEPENDENT ON <dimacsvar1> <dimacsvar2> <dimacsvarN>

dove <dimacsvar> e' una variabile dipendente, mentre <dimacsvar1> <dimacsvar2> <dimacsvarN> sono le variabili dimacs DA CUI dipende.
La parte "ON <dimacsvar1> <dimacsvar2> <dimacsvarN>" e' opzionale: se assente, significa che <dimacsvar> dipende funzionalmente da TUTTE LE ALTRE (default).

Una variabile funzionalmente dipendente è una variabile il cui valore è univocamente determinato quando lo è quello delle variabili da cui dipende.
Es:se X dipende funzionalmente da Y e da Z, allora se Y e Z assumono un valore (vero o falso che sia) anche X assumerà un valore di verità.
L'obiettivo della modifica è quello di ritardare il più possibile il branching sulle variabili funzionalmente dipendenti. Più precisamente, il branching è impedito sulle variabili dipendenti finché quelle indipendenti non sono esaurite.

PRIMA MODIFICA:
La prima coinvolge il parser, che deve avere la capacità di riconoscere la nuova sezione del file DIMACS.

Al fine di rendere la nuova codifica DIMACS trasparente ai risolutori standard, la sezione "PROPERTIES" è realizzata come commento.
In particolare, appena il parser riconosce la stringa "c START PROPERTIES" si realizza il seguente algoritmo:

```
- WHILE ("stringa in ingresso" != "c END PROPERTIES")
 - effettua il parsing e memorizza il primo intero della riga (la
 variabile funzionalmente dipendente)
 - setta a false l'indice corrispondente all'intero "parsato" del
 vettore decision_var[]
 - salta il resto della riga
```

Ovviamente il parser non è realizzato in maniera robusta, poiché si assume che venendo l'input (almeno nelle intenzioni originali) da un modulo automatico, quest'ultimo rispetti la sintassi precisa.

NB: decision_var[] è un vettore già presente originariamente in MINISAT.

SECONDA MODIFICA:
La seconda modifica riguarda la parte relativa alla scelta delle variabili. Come gran parte dei SATSolver moderni, MINISAT tiene traccia, in una struttura dati heap, di un valore di **activity** per TUTTE le variabili in gioco. La variabile di decisione con il valore migliore viene scelta. La struttura dati HEAP è globale e viene aggiornata ad ogni passo.
L'algoritmo (originale) che implementa la scelta della variabile è il seguente (next è la variabile scelta, indef rappresenta una variabile non settata):

```
- next:=indef;
- WHILE (next è indefinita || next è stata già assegnata ||
  next NON è una variabile di decisione)
       SE (l'heap è vuoto) ALLORA next rimane indefinita
       ALTRIMENTI prendi la variabile migliore dall'heap
```

Sostanzialmente quindi, l'algoritmo cicla finché non trova la miglior variabile di decisione non ancora assegnata.
Se non la trova, next rimane indefinita con tutte le conseguenze del caso.
Si capisce dunque dall'algoritmo che nell'heap è memorizzato una valore di activity per tutte le variabili, anche quelle già assegnate o non di decisione.
Bisogna dunque evitare di buttar via una variabile non di decisione (dipendente) non assegnata, visto che a priori non sappiamo se ci sono ancora variabili di decisione candidate alla scelta.
La soluzione che abbiamo pensato è la seguente: memorizziamo la prima variabile di decisione non assegnata estratta dall'heap in next2: se non vi sono più variabili di decisione non assegnate next2 sarà la scelta, altrimenti ignora next2. L'algoritmo preciso è il seguente:

```
- next:=indef;
- next2:=indef;
- flag:=flase;

- WHILE (next è indefinita || next è stata già assegnata ||
         next è una variabile di decisione)
     SE (l'heap è vuoto) ALLORA next:=indefinita
     ALTRIMENTI prendi la variabile migliore dall'heap e
         mettila in next
         SE (flag == false && next non è assegnata && next non è
         di decisione)
         ALLORA {next2 = next; flag = true;}
- SE (next è indefinita) next = next2
```

Appendice B: Resoconto dei test

SOCIAL GOLFER
Number of players: 12
Number of groups: 4
Number of weeks: 8

NOSTRO SOLVER

```
============================[ Search Statistics ]==============================
| Conflicts |          ORIGINAL         |          LEARNT          | Progress |
|           |    Vars  Clauses Literals |    Limit  Clauses Lit/Cl |          |
===============================================================================
|         0 |     353    78396   262240 |    26132        0    nan |  0.000 % |
|       101 |     353    78396   262240 |    28745      101    131 | 35.991 % |
|       253 |     353    78396   262240 |    31619      253    136 | 35.991 % |
|       479 |     353    78396   262240 |    34781      479    129 | 35.991 % |
|       819 |     353    78396   262240 |    38259      819    130 | 35.991 % |
|      1327 |     353    78396   262240 |    42085     1327    125 | 35.991 % |
|      2086 |     353    78396   262240 |    46294     2086    124 | 35.991 % |
|      3225 |     353    78396   262240 |    50923     3225    117 | 35.991 % |
|      4933 |     353    78396   262240 |    56016     4933    113 | 35.991 % |
|      7496 |     353    78396   262240 |    61617     7496    101 | 35.991 % |
|     11340 |     353    78396   262240 |    67779    11340     97 | 35.991 % |
|     17106 |     353    78396   262240 |    74557    17106     91 | 35.991 % |
|     25755 |     353    78396   262240 |    82013    25755     86 | 35.991 % |
|     38730 |     353    78396   262240 |    90214    38730     80 | 35.991 % |
|     58191 |     353    78396   262240 |    99236    58191     78 | 35.991 % |
|     87383 |     353    78396   262240 |   109159    87383     75 | 35.991 % |
|    131175 |     353    78396   262240 |   120075    34944     67 | 35.991 % |
|    196859 |     353    78396   262240 |   132083   100628     67 | 35.991 % |
|    295385 |     353    78396   262240 |   145291    82728     62 | 35.991 % |
===============================================================================
restarts              : 19
conflicts             : 428909         (574 /sec)
decisions             : 484645         (1.79 % random) (648 /sec)
propagations          : 123765122      (165522 /sec)
conflict literals     : 27970675       (48.86 % deleted)
Memory used           : 75.37 MB
CPU time              : 747.727 s
UNSATISFIABLE
```

SOLVER ORIGINALE

```
============================[ Problem Statistics ]=============================
|                                                                             |
|  Number of variables:  7424                                                 |
|  Number of clauses:    121800                                               |
|  Parsing time:         0.05         s                                       |
============================[ Search Statistics ]==============================
| Conflicts |          ORIGINAL         |          LEARNT          | Progress |
|           |    Vars  Clauses Literals |    Limit  Clauses Lit/Cl |          |
===============================================================================
|         0 |    4752    78396   262240 |    26132        0    nan |  0.000 % |
|       101 |    4752    78396   262240 |    28745      101     39 | 35.991 % |
|       251 |    4752    78396   262240 |    31619      251     79 | 35.991 % |
|       479 |    4752    78396   262240 |    34781      479    111 | 35.991 % |
|       816 |    4752    78396   262240 |    38259      816    116 | 35.991 % |
```

```
|      1322 |    4752    78396   262240 |    42085     1322    135 | 35.991 % |
|      2081 |    4752    78396   262240 |    46294     2081    155 | 35.991 % |
|      3222 |    4752    78396   262240 |    50923     3222    166 | 35.991 % |
|      4931 |    4752    78396   262240 |    56016     4931    169 | 35.991 % |
|      7494 |    4752    78396   262240 |    61617     7494    173 | 35.991 % |
|     11338 |    4752    78396   262240 |    67779    11338    173 | 35.991 % |
|     17104 |    4752    78396   262240 |    74557    17104    170 | 35.991 % |
|     25756 |    4752    78396   262240 |    82013    25756    170 | 35.991 % |
|     38730 |    4752    78396   262240 |    90214    38730    168 | 35.991 % |
|     58193 |    4752    78396   262240 |    99236    58193    165 | 35.991 % |
|     87387 |    4752    78396   262240 |   109159    87387    162 | 35.991 % |
|    131179 |    4752    78396   262240 |   120075    37851    141 | 35.991 % |
|    196867 |    4752    78396   262240 |   132083   103539    144 | 35.991 % |
|    295394 |    4752    78396   262240 |   145291    88980    126 | 35.991 % |
|    443183 |    4752    78396   262240 |   159820   109704    129 | 35.991 % |
===============================================================================
restarts              : 20
conflicts             : 631837         (309 /sec)
decisions             : 852505         (1.66 % random) (417 /sec)
propagations          : 151628400      (74200 /sec)
conflict literals     : 84410288       (27.04 % deleted)
Memory used           : 184.70 MB
CPU time              : 2043.5 s
UNSATISFIABLE
```

SOCIAL GOLFER
Number of players: 12
Number of groups: 4
Number of weeks: 4

NOSTRO SOLVER

```
============================[ Search Statistics ]==============================
| Conflicts |          ORIGINAL         |          LEARNT          | Progress |
|           |    Vars  Clauses Literals |    Limit  Clauses Lit/Cl |          |
===============================================================================
|         0 |     409   285182   889576 |    95060        0    nan |  0.000 % |
===============================================================================
restarts              : 1
conflicts             : 21             (68 /sec)
decisions             : 590            (1.36 % random) (1915 /sec)
propagations          : 49556          (160886 /sec)
conflict literals     : 14399          (4.22 % deleted)
Memory used           : 29.47 MB
CPU time              : 0.308019 s
SATISFIABLE
```

SOLVER ORIGINALE

```
============================[ Search Statistics ]==============================
| Conflicts |          ORIGINAL         |          LEARNT          | Progress |
|           |    Vars  Clauses Literals |    Limit  Clauses Lit/Cl |          |
===============================================================================
|         0 |   14552   285182   889576 |    95060        0    nan |  0.000 % |
|       100 |   14552   285182   889576 |   104566      100    112 | 27.298 % |
===============================================================================
restarts              : 2
conflicts             : 135            (312 /sec)
decisions             : 52589          (1.37 % random) (121726 /sec)
```

```
propagations          : 337283         (780699 /sec)
conflict literals     : 15968          (2.43 % deleted)
Memory used           : 28.61 MB
CPU time              : 0.432027 s
SATISFIABLE
```

ROSTERING

c Number of employees: 24
c Number of slots in each day: 4
c Number of night slots: 2
c Min number of employees that must be at work in each slot: 3
c Min number of break slots: 2
c Min number of working slots: 2

NOSTRO SOLVER

```
============================[ Search Statistics ]==============================
| Conflicts |          ORIGINAL         |          LEARNT          | Progress |
|           |    Vars  Clauses Literals |    Limit  Clauses Lit/Cl |          |
===============================================================================
|         0 |     673   257508   703744 |    85836        0    nan |  0.000 % |
|       101 |     673   257508   703744 |    94419      101   2975 |  0.000 % |
|       251 |     673   257508   703744 |   103861      251   3315 |  0.000 % |
|       476 |     673   257508   703744 |   114247      476   4325 |  0.000 % |
===============================================================================
restarts              : 4
conflicts             : 510            (111 /sec)
decisions             : 2301           (1.69 % random) (503 /sec)
propagations          : 811329         (177290 /sec)
conflict literals     : 2227232        (3.45 % deleted)
Memory used           : 34.31 MB
CPU time              : 4.57629 s
SATISFIABLE
```

SOLVER ORIGINALE

```
============================[ Search Statistics ]==============================
| Conflicts |          ORIGINAL         |          LEARNT          | Progress |
|           |    Vars  Clauses Literals |    Limit  Clauses Lit/Cl |          |
===============================================================================
|         0 |   66440   257508   703744 |    85836        0    nan |  0.000 % |
|       102 |   66440   257508   703744 |    94419      102   4312 |  0.000 % |
|       252 |   66440   257508   703744 |   103861      252   5071 |  0.000 % |
===============================================================================
restarts              : 3
conflicts             : 260            (285 /sec)
decisions             : 38919          (1.37 % random) (42672 /sec)
propagations          : 340737         (373592 /sec)
conflict literals     : 1306118        (0.10 % deleted)
Memory used           : 30.09 MB
CPU time              : 0.912057 s
SATISFIABLE
```

SOCIAL GOLFER
number of players = 10
number of groups = 5
number of weeks = 10

SOLVER ORIGINALE

```
============================[ Search Statistics ]==============================
| Conflicts |          ORIGINAL          |          LEARNT          | Progress |
|           |    Vars  Clauses Literals  |    Limit  Clauses Lit/Cl |          |
===============================================================================
|         0 |    1980    57245   212490  |    19081        0    nan |  0.000 % |
|       100 |    1980    57245   212490  |    20989      100     10 | 28.000 % |
|       251 |    1980    57245   212490  |    23088      251     20 | 28.000 % |
|       476 |    1980    57245   212490  |    25397      476     26 | 28.000 % |
|       814 |    1980    57245   212490  |    27937      814     29 | 28.000 % |
|      1322 |    1980    57245   212490  |    30731     1322     31 | 28.000 % |
|      2082 |    1980    57245   212490  |    33804     2082     32 | 28.000 % |
|      3221 |    1980    57245   212490  |    37184     3221     32 | 28.000 % |
|      4931 |    1980    57245   212490  |    40903     4931     32 | 28.000 % |
|      7494 |    1980    57245   212490  |    44993     7494     32 | 28.000 % |
|     11339 |    1980    57245   212490  |    49492    11339     31 | 28.000 % |
|     17105 |    1980    57245   212490  |    54442    17105     30 | 28.000 % |
|     25754 |    1980    57245   212490  |    59886    25754     30 | 28.000 % |
|     38728 |    1980    57245   212490  |    65875    38728     30 | 28.000 % |
|     58190 |    1980    57245   212490  |    72462    58190     29 | 28.000 % |
===============================================================================
restarts              : 15
conflicts             : 69048          (1723 /sec)
decisions             : 80972          (1.78 % random) (2020 /sec)
propagations          : 7261993        (181176 /sec)
conflict literals     : 1955003        (16.33 % deleted)
Memory used           : 23.32 MB
CPU time              : 40.0825 s
UNSATISFIABLE
```

NOSTRO SOLVER

```
============================[ Search Statistics ]==============================
| Conflicts |          ORIGINAL          |          LEARNT          | Progress |
|           |    Vars  Clauses Literals  |    Limit  Clauses Lit/Cl |          |
===============================================================================
|         0 |     451    57245   212490  |    19081        0    nan |  0.000 % |
|       100 |     451    57245   212490  |    20989      100     31 | 28.000 % |
|       250 |     451    57245   212490  |    23088      250     30 | 28.000 % |
|       476 |     451    57245   212490  |    25397      476     33 | 28.000 % |
|       814 |     451    57245   212490  |    27937      814     33 | 28.000 % |
|      1320 |     451    57245   212490  |    30731     1320     34 | 28.000 % |
|      2081 |     451    57245   212490  |    33804     2081     34 | 28.000 % |
|      3221 |     451    57245   212490  |    37184     3221     35 | 28.000 % |
|      4930 |     451    57245   212490  |    40903     4930     37 | 28.000 % |
|      7492 |     451    57245   212490  |    44993     7492     37 | 28.000 % |
|     11336 |     451    57245   212490  |    49492    11336     38 | 28.000 % |
|     17102 |     451    57245   212490  |    54442    17102     38 | 28.000 % |
|     25753 |     451    57245   212490  |    59886    25753     39 | 28.000 % |
|     38727 |     451    57245   212490  |    65875    38727     39 | 28.000 % |
|     58189 |     451    57245   212490  |    72462    58189     40 | 28.000 % |
|     87385 |     451    57245   212490  |    79708    30002     38 | 28.000 % |
|    131174 |     451    57245   212490  |    87679    73791     39 | 28.000 % |
|    196862 |     451    57245   212490  |    96447    67981     37 | 28.000 % |
```

```
|    295389 |     451    57245   212490 |   106092    87035     36 | 28.000 % |
|    443179 |     408    45336   166816 |   116701    11396     26 | 33.091 % |
===============================================================================
restarts              : 20
conflicts             : 445112         (1514 /sec)
decisions             : 486418         (1.78 % random) (1655 /sec)
propagations          : 42362054       (144091 /sec)
conflict literals     : 15696736       (22.99 % deleted)
Memory used           : 45.68 MB
CPU time              : 293.994 s
UNSATISFIABLE
```

SOCIAL GOLFER
Number of players: 12
Number of groups: 4
Number of weeks: 4

NOSTRO SOLVER

```
|  Parsing time:         0.10         s                                       |
============================[ Search Statistics ]==============================
| Conflicts |          ORIGINAL         |          LEARNT          | Progress |
|           |    Vars  Clauses Literals |    Limit  Clauses Lit/Cl |          |
===============================================================================
|         0 |     968   202972   791032 |    67657        0    nan |  0.000 % |
===============================================================================
restarts              : 1
conflicts             : 0              (0 /sec)
decisions             : 449            (1.78 % random) (2159 /sec)
propagations          : 2304           (11076 /sec)
conflict literals     : 0              ( nan % deleted)
Memory used           : 19.71 MB
CPU time              : 0.208013 s

SATISFIABLE
```

SOLVER ORIGINALE

```
============================[ Problem Statistics ]=============================
|                                                                             |
|  Number of variables:  2304                                                 |
|  Number of clauses:    287624                                               |
|  Parsing time:         0.11         s                                       |
============================[ Search Statistics ]==============================
| Conflicts |          ORIGINAL         |          LEARNT          | Progress |
|           |    Vars  Clauses Literals |    Limit  Clauses Lit/Cl |          |
===============================================================================
|         0 |    1936   202972   791032 |    67657        0    nan |  0.000 % |
===============================================================================
restarts              : 1
conflicts             : 0              (0 /sec)
decisions             : 445            (1.12 % random) (1918 /sec)
propagations          : 2304           (9930 /sec)
conflict literals     : 0              ( nan % deleted)
Memory used           : 18.71 MB
CPU time              : 0.232014 s

SATISFIABLE
```

SOCIAL GOLFER
Number of players: 14
Number of groups: 7
Number of weeks: 10

NOSTRO SOLVER

```
|  Parsing time:         0.12         s                                      |
============================[ Search Statistics ]==============================
| Conflicts |          ORIGINAL         |          LEARNT          | Progress |
|           |    Vars  Clauses Literals |    Limit  Clauses Lit/Cl |          |
===============================================================================
|         0 |     911   242845   900250 |    80948        0    nan |  0.000 % |
===============================================================================
restarts              : 1
conflicts             : 69             (319 /sec)
decisions             : 4890           (1.72 % random) (22638 /sec)
propagations          : 64440          (298315 /sec)
conflict literals     : 4463           (6.57 % deleted)
Memory used           : 21.59 MB
CPU time              : 0.216013 s

SATISFIABLE
```

SOLVER ORIGINALE

```
============================[ Problem Statistics ]=============================
|                                                                             |
|  Number of variables:  7350                                                 |
|  Number of clauses:    299365                                               |
|  Parsing time:         0.12         s                                      |
============================[ Search Statistics ]==============================
| Conflicts |          ORIGINAL         |          LEARNT          | Progress |
|           |    Vars  Clauses Literals |    Limit  Clauses Lit/Cl |          |
===============================================================================
|         0 |    5720   242845   900250 |    80948        0    nan |  0.000 % |
|       101 |    5720   242845   900250 |    89043      101     20 | 22.177 % |
|       251 |    5720   242845   900250 |    97947      251     22 | 22.177 % |
|       476 |    5720   242845   900250 |   107742      476     22 | 22.177 % |
|       813 |    5720   242845   900250 |   118516      813     22 | 22.177 % |
|      1319 |    5720   242845   900250 |   130368     1319     23 | 22.177 % |
===============================================================================
restarts              : 6
conflicts             : 1614           (1552 /sec)
decisions             : 177408         (1.36 % random) (170574 /sec)
propagations          : 1463743        (1407357 /sec)
conflict literals     : 37219          (3.03 % deleted)
Memory used           : 21.88 MB
CPU time              : 1.04007 s

SATISFIABLE
```

SOCIAL GOLFER
Number of players: 16
Number of groups: 8
Number of weeks: 12

NOSTRO SOLVER

```
|  Parsing time:         0.27         s                                      |
============================[ Search Statistics ]==============================
| Conflicts |          ORIGINAL         |          LEARNT          | Progress |
|           |    Vars  Clauses Literals |    Limit  Clauses Lit/Cl |          |
===============================================================================
|         0 |    1441   593826  2224080 |   197942        0    nan |  0.000 % |
===============================================================================
restarts              : 1
conflicts             : 0              (0 /sec)
decisions             : 527            (1.71 % random) (1176 /sec)
propagations          : 13056          (29141 /sec)
conflict literals     : 0              ( nan % deleted)
Memory used           : 48.70 MB
CPU time              : 0.448028 s

SATISFIABLE
```

SOLVER ORIGINALE

```
============================[ Problem Statistics ]=============================
|                                                                             |
|  Number of variables:  13056                                                |
|  Number of clauses:    708396                                               |
|  Parsing time:         0.22         s                                      |
============================[ Search Statistics ]==============================
| Conflicts |          ORIGINAL         |          LEARNT          | Progress |
|           |    Vars  Clauses Literals |    Limit  Clauses Lit/Cl |          |
===============================================================================
|         0 |   10440   593826  2224080 |   197942        0    nan |  0.000 % |
|       100 |   10440   593826  2224080 |   217736      100     18 | 20.037 % |
|       250 |   10440   593826  2224080 |   239509      250     32 | 20.037 % |
|       475 |   10440   593826  2224080 |   263460      475     25 | 20.037 % |
|       812 |   10440   593826  2224080 |   289806      812     25 | 20.037 % |
|      1318 |   10440   593826  2224080 |   318787     1318     23 | 20.037 % |
|      2078 |   10440   593826  2224080 |   350666     2078     25 | 20.037 % |
|      3217 |   10440   593826  2224080 |   385732     3217     26 | 20.037 % |
|      4925 |   10440   593826  2224080 |   424306     4925     28 | 20.037 % |
|      7487 |   10440   593826  2224080 |   466736     7487     29 | 20.037 % |
===============================================================================
restarts              : 10
conflicts             : 9190           (998 /sec)
decisions             : 1546498        (1.53 % random) (167868 /sec)
propagations          : 12287181       (1333740 /sec)
conflict literals     : 275456         (4.93 % deleted)
Memory used           : 52.42 MB
CPU time              : 9.21257 s

SATISFIABLE
```