ESO:
Existential Second Order Logic

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

Autori:
Angeloni Michela
Franchi Junio Valerio
Cangialosi Piero

Docente:
Toni Mancini

Introduzione (1/5)

- Spesso per i problemi combinatori si ricorre 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.
- Questo approccio è costoso in quanto richiede la scrittura di un nuovo generatore per ogni nuovo 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 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

Introduzione (2/5)

- Vediamo alcune porzioni del DIMACS del problema del Social Golfer per 8 giocatori, 4 gruppi e 3 settimane e l'output fornito da un solutore.
- La rappresentazione è lunga, complessa e poco leggibile, l'output del solutore è complesso e poco espressivo.

```
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
[…]
```

```
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
[…]
```

```
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 -… …
…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
```

Introduzione (3/5)

- Un approccio meno costoso potrebbe essere quello di descrivere il problema in logica, attraverso un linguaggio più dichiarativo, e specificare il problema e le caratteristiche delle soluzioni.
- Usiamo 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.
  - Si specifica il problema e le caratteristiche delle soluzioni ma non l'algoritmo per trovarle (programmazione dichiarativa).

Introduzione (4/5)

- Esempio: graph coloring in FOL con uso di Mace
- In base alle proprietà espresse il ricercatore di modelli troverà un'assegnazione di variabili che verifichi tutti i vincoli

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

% 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)) ).

end_of_list.
```

Introduzione (5/5)

- Si può fare un ulteriore passo avanti utilizzando la logica del secondo ordine (in particolare ESO) che permette di quantificare sui predicati
- Non si espande l'interpretazione ma ci si chiede **se esiste** una tale interpretazione
- Il problema del Graph coloring diventa:

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

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

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

Spazio di Ricerca

Vincolo: ogni nodo ha almeno ed al più un colore

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

Vincolo:
nodi adiacenti hanno colori diversi

Logica del Secondo Ordine

- La Logica del Secondo Ordine differisce da quella del Primo Ordine 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*".
- 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 $P(t_1,\ldots,t_n)$ dove t_i sono termini e P è un simbolo di predicato n-ario oppure una variabile di predicato.

Logica del Secondo Ordine: semantica

- Per quanto riguarda la semantica occorre estendere il concetto di soddisfazione per includere i nuovi quantificatori. Si ha :
- ***Definizione**: 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 se:*
  - $\models I\ (\forall X^n)\ \phi[s]$ *se e solo se per ogni relazione n-aria* $R \subseteq |I|^n$, $\models I\ [s(X^n|R)]$. *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.*
  - $\models I\ (\forall F^n)\ \phi[s]$ *se e solo se per ogni funzione n-aria* $f:|I|^n \rightarrow |I|$, $I\ \phi[s(F^n|f)]$. *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.*

Existential Second-Order logic

- La Logica del Secondo Ordine può essere vista come un linguaggio di modellazione e tutti i linguaggi di programmazione correnti possono essere visti come frammenti della Logica del Secondo Ordine.
- In particolare ci soffermeremo su ESO (Existential Second-Order Logic) ovvero il linguaggio della logica del secondo ordine che utilizza 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.
- Usiamo un modello che non ha un supporto per gestire le funzioni in quanto questo non altera il potere espressivo.

Sintassi di ESO

Una formula in ESO è del tipo:

$$\exists\, \mathbf{S}\ \varphi\ (\mathbf{S}, \mathbf{R})$$

- **R** è un insieme di predicati e denota l'istanza di uno schema relazionale;
- **S** è l'insieme dei predicati esistenzialmente quantificati, chiamati *guessed* (indovinati) e le loro possibili estensioni con tuple nell'universo di Herbrand codificano lo spazio di ricerca;
- φ è una formula chiusa del primo ordine e codifica i vincoli da soddisfare.

Un'istanza $\boldsymbol{I}$ del problema è data come database relazionale sullo schema $\boldsymbol{R}$, cioè come estensione di tutte le relazioni in $\boldsymbol{R}$.

Semantica di ESO

- **Definizione:** *Un'istanza* I *di* π*, codificata come database relazionale con schema* R*, è* <u>*soddisfacibile*</u> *se e solo se esiste una lista di relazioni* $\Sigma_1, \ldots \Sigma_n$ *che insieme ad* I *soddisfa la parte di formula al primo ordine, cioè tale che* $(\Sigma_1, \ldots \Sigma_n, I) \models \phi$.
- In altre parole si vuole trovare, data un'estensione di predicati in **R** (istanza **I**), se esiste un'estensione di predicati in **S**, con elementi nell'Universo di Herbrand, tale che la formula φ è soddisfatta.

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, …, h_n sono elementi di H(C), allora anche $f(h_1,..,h_n)$ sta in H(C)*
- 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 Herbrand è formato dalle costanti che occorrono in **I** più le costanti che occorrono in **φ**.

Base e Interpretazione di Herbrand

- **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.*

- 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))),…}.

- **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.

Vantaggi di ESO

- Avere un linguaggio di modellazione basato ESO è vantaggioso perché:
  - è pienamente dichiarativo;
  - ha proprietà computazionali chiare:
    - ✓potere espressivo : cattura NP
    - ✓complessità dei dati : NP-completo
  - la letteratura e i tool esistenti possono essere applicati al ragionamento sulle formule logiche.
- ESO è un formalismo indicato per la modellazione a vincoli perché ben si adatta a questa classe di problemi, essendo in grado di rappresentare tutti i problemi di ricerca della classe NP (teorema di Fagin).

Teorema di Fagin

- Il teorema di Fagin afferma che un problema di decisione è in NP se e solo se esiste una formula ESO che lo esprime.
- **Teorema:**

  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.
- **Corollario:**

  Ogni query computabile in tempo polinomiale è esprimibile come una formula esistenziale di Horn al secondo ordine: P $\subseteq$ ESO-Horn.

Cosa manca in ESO

- Non c'è un costrutto base per esprimere la tipizzazione
- Non è presente un supporto nativo per le funzioni
- Non si possono esprimere interi limitati ed aritmetica

Esistono dei metodi per compensare queste mancanze che lasciano inalterato il potere espressivo

Trattare tipizzazione e funzioni

- Si può usare un linguaggio al primo ordine multi-sorta
- Si possono usare predicati monadici aggiuntivi nell'insieme R
- Per le funzioni è sufficiente usare vincoli aggiuntivi al primo ordine

Trattare tipizzazione e funzioni: logica multi-sorta

- La logica multi-sorta ci permette di trattare l'universo non come una collezione omogenea di oggetti ma di tipizzare tali oggetti e di creare delle partizioni di tipi diversi.

- 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}.

Trattare tipizzazione e funzioni: esempio

- Graph 3-coloring: vengono aggiunti vincoli al primo 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')

Numeri finiti e Aritmetica

- Possono essere pre-interpretati: si aggiungono in R le relazioni che rappresentano (pre-interpretano) gli interi limitati e le operazioni tra essi, cioè si aggiungono le relazioni per il dominio dei numeri e le operazioni necessarie.
- Possono essere "guessed" aggiungendo in S relazioni che codificano i domini finiti per gli interi e le operazioni necessarie, cioè si indovinano delle relazioni aggiuntive che permettono di vedere le costanti del dominio di Herbrand H come interi, trovando un ordinamento totale su di esse.

Numeri finiti e Aritmetica: Pre-interpretare

- 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

Numeri finiti e Aritmetica: “Indovinare” (1/3)

- Supponiamo di avere la necessità di utilizzare operazioni aritmetiche su interi limitati da 1 a size(H) dove H è l’Universo di Herbrand.
- Utilizziamo gli elementi stessi del dominio di interpretazione, che sono size(H), indovinando, mediante una formula ESO ausiliaria, un ordinamento totale tra di essi.
- E’ necessario trovare un cammino Hamiltoniano sul grafo completo; questo percorso individua una funzione successore sulle costanti viste come interi.

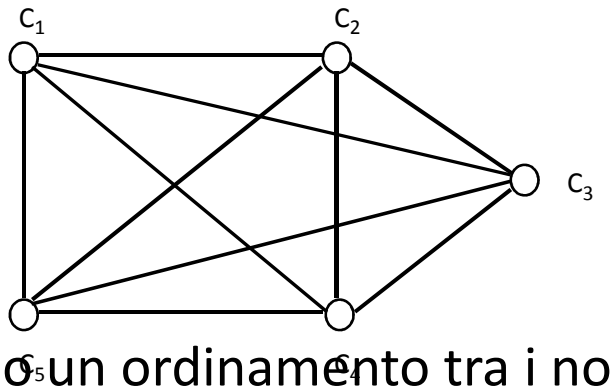
Numeri finiti e Aritmetica: “Indovinare” (2/3)

- 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.

- Aggiungere aritmetica su interi limitati **NON** altera il potere espressivo del linguaggio.

Numeri finiti e Aritmetica: “Indovinare” (3/3)

- Consideriamo il caso 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 grafo.



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:

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

ESEMPI: Ramsey Problem

- Il problema 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, 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.

ESEMPI: Ramsey Problem

Spazio di Ricerca

$\exists$ 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' $\wedge$

$\forall$ 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

ESEMPI: 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

ESEMPI: Social Golfe

Spazio di Ricerca

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

$\forall$ 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 1a: ogni gruppo ha almeno 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)

Vincolo 1b: ogni gruppo ha al più 4 giocatori

$\forall$ 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, 2)))

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

ESEMPI: 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:
  - Ogni parola in S ha 4 simboli in {C, G};
  - Ogni coppia di parole distinte in S differisce in almeno 4 posizioni;
  - 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 l'inverso di x1,…,x8 e C(y1,…,y8) è la parola in cui ogni A è sostituita con una T e viceversa e ogni C è sostituita da una G e viceversa.
- Si considera una versione decisionale del problema.

ESEMPI: Word Design

Spazio di Ricerca

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

$\exists$ words $\forall$ X, Y, Z words(X,Y,Z) $\rightarrow$ words_idx(X) $\wedge$ letters_idx(Y) $\wedge$letter(z) $\wedge$

$\forall$ 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)$\wedge$

Vincolo1: ogni parola ha esattamente 4 simboli in {C,G}

$\forall$ 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)$\wedge$

ESEMPI: Word Design

$\forall$ 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)) $\wedge$

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

$\forall$ 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))) $\wedge$

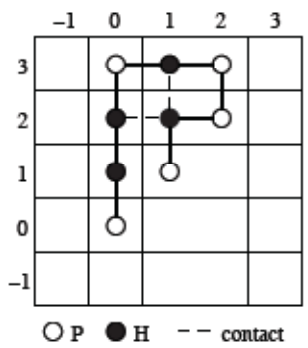
Vincolo 3:
per ogni coppia di parole, il reverse della prima e il complemento della seconda differiscono di almeno 4 posizioni

ESEMPI: Protein Folding (1/3)

- Data una struttura primaria di proteine, cioè una sequenza di aminoacidi (20 tipi), trovare una una conformazione completa 3D di una proteina.
- Consideriamo il problema semplificato: **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.

ESEMPI: Protein Folding (2/3)

- 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

ESEMPI:Protein Folding (3/4)

$\exists$ 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)

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

Versione decisionale: soglia minima sul numero di contatti

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.
- ESO può essere considerata come la base logica formale per tutti i linguaggi per la modellazione a vincoli

CSP: Constraint Satisfaction Problem

- Quando vengono assegnati dei dati di input (fissando un' estensione per i simboli di predicato nell'insieme R) si ottiene un classico problema CSP, 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.
- Una soluzione di un CSP è un'assegnazione di valori del dominio alle variabili, che soddisfa tutti i vincoli.

CSP e SAT

- I problemi CSP possono essere divisi in due gruppi:
  - CSP finiti, quelli con un dominio finito di valori
  - CSP infiniti, quei problemi con un dominio infinito di valori (programmazione lineare);
- SAT è un problema CSP finito, e ogni CSP può essere riducibile a SAT.
- Esempio: problema SAT modellato come 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

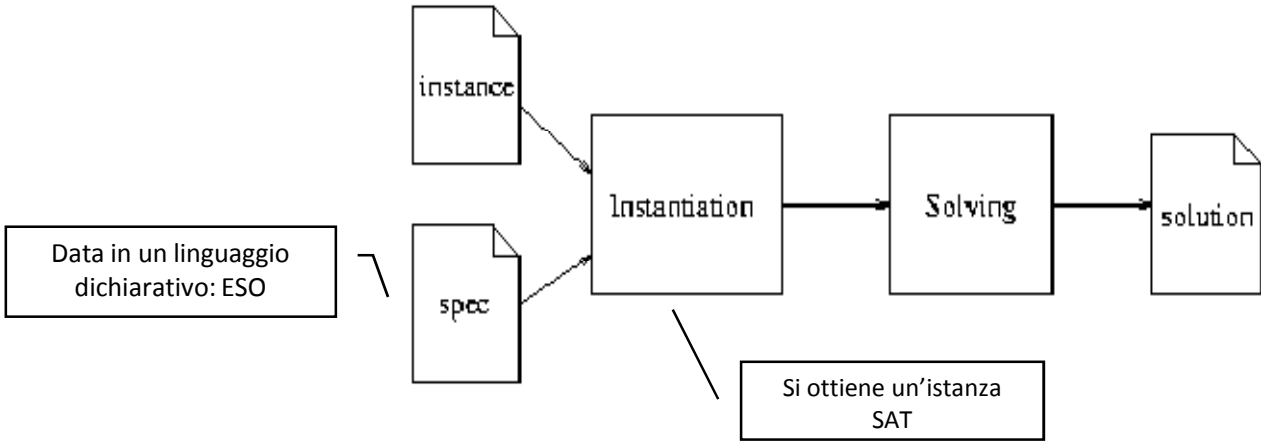
Quindi: $(x_1 \vee x_2 \vee x_3) = \text{TRUE}$

$(\neg x_1 \vee x_2 \vee x_3) = \text{TRUE}$

$(x_1 \vee \neg x_2) = \text{TRUE}$

ESO vs CSP-SAT

- 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.



Considerazioni

- Il modello ottenuto spesso non è efficiente e richiede una riformulazione per accelerare il processo di soluzione.
- 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.

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.
- Il riconoscimento delle dipendenze funzionali permette di risparmiare spazio e tempo durante la fase di branching: 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' conveniente, ritardare il più possibile la scelta di b come variabile di branching.
- Riconoscere quante più possibili dipendenze funzionali permette di diminuire di gran lunga la profondità dell'albero di ricerca

Dipendenze funzionali

- Il problema di trovare le dipendenze funzionali sulle specifiche ESO si riduce al verificare le proprietà semantiche della formula al primo ordine.

  Data una specifica di problemi su uno schema di input $\mathcal{R}$

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

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

  $$<\Sigma, \Pi> \neq <\Sigma', \Pi'> \text{ e}$$

  $$\Sigma, \Pi, I \models \phi \text{ e}$$

  $$\Sigma', \Pi', I \models \phi$$

  Allora $\Sigma \neq \Sigma'$.

- *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.*

ESO vs OPL

- Uno dei linguaggi dichiarativi di modellazione a vincoli più usati è OPL, tale linguaggio è molto simile ad ESO.
- 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”;
  - 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, se si vuole seguire una strategia per ricerca diversa quella di default.
- 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 non è possibile esprimere le funzioni obiettivo.

ESO vs OPL: Esempio

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

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

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

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

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

Tecniche di Istanziazione

- Esistono traduttori che a partire da una specifica restituiscono un'istanza CNF.
- Abbiamo preso in considerazione Spec2Sat.
- Spec2Sat ha un modulo *parser* che riceve in ingresso il file contenente la specifica e costruisce una sua rappresentazione interna.
- Dalla rappresentazione interna all'istanza SAT vera e propria la maggior parte del lavoro è svolta dal file *translate.*

Tecniche di Istanziazione

- 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

- Spec2Sat prende in input, come specifica, un file scritto nel linguaggio NP-SPEC

Metaregole

- Definiscono sinteticamente un insieme di regole che descrive la tipologia dello spazio di ricerca
- Non aumentano il potere espressivo del linguaggio
- Si dividono in
    - Permutation
    - Partition
    - Subset
    - intFunction

NP-SPEC

- E' una estensione non deterministica del DATALOG
- Si può vedere come una variante sintattica di ESO
- Il potere espressivo di NP-SPEC permette di catturare NP
- Ogni specifica è formata da 2 sezioni:
  - una sezione *DATABASE,* che rappresenta l'istanza specifica del problema e dove vengono definite le costanti;
  - una sezione *SPECIFICATION,* che contiene la dichiarazione dello spazio di ricerca, che corrisponde alla definizione dei guessed predicate

DATABASE

- Questa sezione è usata per specificare l'istanza del problema.
- E' in corrispondenza con l'insieme di predicati R, che definisce lo schema per tutte le istanze di input del problema.
- L'estensione dei predicati dichiarati nella sezione *DATABASE* è sempre come specificato dall'insieme dei fatti.

SPECIFICATION

- Questa sezione è in corrispondenza con l'insieme S dei predicati guessed le cui estensioni codificano i punti dello spazio di ricerca per il problema.
- 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.

NP-SPEC: Esempio

Graph 3-coloring:

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).

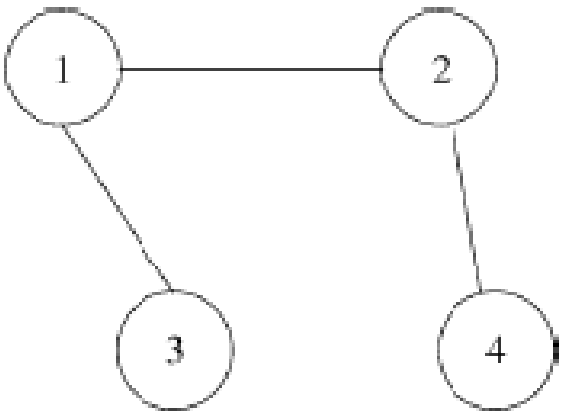
Translate

- Calcola :
  - le Metarules
  - i DBPredicates
  - l'Universo di Herbrand
- Costruisce il vocabolario 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),
  - una variabile per ogni altro tipo di predicato (variabili di tipo γ).

Esempio di vocabolario

- Graph 3-coloring

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



Translate

- richiama il metodo buildGraph() che costruisce il grafo G della specifica;
- che calcola l'insieme dei predicati *inline;*
- invoca il metodo *SimplifyRules()* che istanzia le regole con i predicati *inline*;
- calcola l'insieme *alive* per i predicati di DB e per le metarules;
- definisce un ordinamento topologico dei nodi del grafo;
- processa i predicati creando così l'insieme T delle clausole che costituiscono l'istanza SAT.

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.