CONSTRAINT PROGRAMMING & PLANNING

Agostino Dovier

Università di Udine
Dipartimento di Matematica e Informatica

Udine, Marzo 2011

Sommario

Constraint Logic Programming

- Il linguaggio di base è Prolog, provvisto di librerie di vincoli. Useremo principalmente la libreria sui domini finiti. Potete usare B Prolog, SWI Prolog, GNU Prolog, Eclipse (liberi) o SICStus (a pagamento, ma abbiamo vecchie licenze)
- La tecnica di programmazione di base è la

  constraint & generate.
- La forma generale di un programma è

```
solve(X1,...Xn) :-
     constrain(X1,...,Xn),
     labeling([parametri],[X1,...,Xn]).
```

- `labeling` è built-in. Bisogna definire i vincoli e scegliere dei buoni parametri per la ricerca (per problemi di dimensione “media” spesso funziona bene `[ff]`).
- Prima di entrare nella metodologia di programmazione, studiamo il linguaggio.

CLP(X)

- *Constraint logic programming* su un dominio di computazione X, in breve: $CLP(X)$.
- Concilia la dichiaratività di Prolog con metodi di computazione orientati a specifici domini.
- Il meccanismo inferenziale, goal-driven, è infatti quello della SLD-risoluzione: da Prolog si ereditano (con opportuni adeguamenti) i concetti di clausola, goal, derivazione di successo o di fallimento, risposta, ecc..
- Per una parte dei simboli del linguaggio una particolare interpretazione su un prefissato dominio del discorso viene fissata.
- Simboli di predicato ad hoc su tali domini sono trattati in modo particolare (con constraint solver).
- Anche $=$ va trattato in modo opportuno.

CLP(X)

Sintassi

- Consideriamo un linguaggio del primo ordine su $\langle \Pi, \mathcal{F}, \mathcal{V} \rangle$.
- L'insieme Π dei simboli di predicato e' partizionato in due: $\Pi = \Pi_C \cup \Pi_P$ con $\Pi_C \cap \Pi_P = \emptyset$.
- Π_P: simboli predicativi definiti nel programma
- Π_C è un insieme di simboli di predicato di constraint (che non possono occorrere nelle teste delle regole).
- Π_C contiene "$=$".
- Un *constraint primitivo* è un letterale sull'alfabeto $\langle \Pi_C, \mathcal{F}, \mathcal{V} \rangle$ (true e false denotano due particolari constraint primitivi)
- Un *constraint* è una congiunzione di constraint primitivi (o un insieme più grande di formule che le contengono).

CLP(X)

Sintassi

- Se $p \in \Pi_P$, e $t_1, \ldots, t_n$ sono termini, allora l'atomo $p(t_1, \ldots, t_n)$ è detto *atomo di programma*.
- Un *goal CLP* è: $\leftarrow \bar{B}$, dove $\bar{B}$ è una congiunzione di atomi di programma e di constraint primitivi.
- Un *fatto CLP* è un atomo $p(t_1, \ldots, t_n)$, dove $p \in \Pi_P$ e ogni t_i è un termine.
- Una *regola CLP* è una clausola della forma

$$p(t_1, \ldots, t_n) \leftarrow \bar{B}$$

  dove $\leftarrow \bar{B}$ è un goal CLP, $p \in \Pi_P$, e t_i sono termini.
- Una regola CLP non è necessariamente una clausola di Horn:

$$p(X, Y) \leftarrow X \neq Y, X < Z, q(X, Z)$$

  equivale a:

$$p(X, Y) \vee (X = Y) \vee \neg(X < Z) \vee \neg q(X, Z)$$

CLP(X)

Semantica

- Uno *stato* e' una coppia $\langle G \mid C \rangle$ dove G è un goal CLP e C è un constraint (anche detto *constraint store*).
- Uno stato è detto di *successo* se ha la forma $\langle \leftarrow \Box | C \rangle$ e vale solve$(C) \neq$ false.
- Uno stato è di *fallimento* se ha la forma $\langle G | C \rangle$ e:
  - solve$(C) =$ false, oppure
  - G è una congiunzione di atomi di programma e non vi è nessuna regola la cui testa abbia simboli di predicato di programma occorrenti nel goal.

CLP(X)

Semantica

- Sia P un programma e G_1 un goal. Un *passo di derivazione CLP* $\langle G_1 \mid C_1 \rangle \Rightarrow \langle G_2 \mid C_2 \rangle$ è definito nel seguente modo:
- sia $G_1 = \leftarrow L_1, \ldots, L_m$ con $m \geq 1$
- sia, p.es. L_1, il letterale selezionato. Allora:
  - se L_1 è un constraint primitivo, si pone $C_2 = L_1 \wedge C_1$.
  - Inoltre se solve$(C_2) =$ false, allora si pone $G_2 = \leftarrow \square$, altrimenti si pone $G_2 = \leftarrow L_2, \ldots, L_n$.
  - se invece $L_1 = p(t_1, \ldots, t_n)$ è un atomo di programma e $p(s_1, \ldots, s_n) \leftarrow \bar{B}$ è una rinomina di una regola di P, allora si pone $G_2 = \leftarrow t_1 = s_1, \ldots, t_n = s_n, \bar{B}, L_2, \ldots, L_n$ e $C_2 = C_1$.

CLP(X)

Semantica

- Una *derivazione per uno stato* S_0 è una sequenza massimale di passi di derivazione che hanno S_0 come primo stato:

$$S_0 \Rightarrow S_1 \Rightarrow \cdots$$

- Una *derivazione per un goal CLP G* è una derivazione per lo stato $\langle G \,|\, \text{true} \rangle$.
- Una *derivazione* (di lunghezza finita) $S_0 \Rightarrow \cdots \Rightarrow S_n$ è detta di *successo* se S_n è uno stato di successo. In tal caso la *risposta calcolata* è definita essere simplify$(C_n, FV(S_0))$.
- Una *derivazione* $S_0 \Rightarrow \cdots \Rightarrow S_n$ è invece è di *fallimento* se S_n è uno stato di fallimento.

CLP(X)

Semantica

- Si basa su due funzioni solve e simplify.
- Per esse non viene data una definizione rigida: la loro specifica dipende dal dominio in cui vengono valutati i constraint.
- Di solito il test solve(C) $\neq$ false non e' un realizzato da un constraint solver completo in quanto potrebbe essere troppo oneroso.
- Viene realizzato invece da un'algoritmo di propagazione efficiente (eventualmente il fallimeno si ha nella ricerca della singola soluzione).

CLP(X)

Semantica

- C'è implicita la nozione di albero SLD ereditato da Prolog.
- E' importante avere dimestichezza con l'albero SLD per ottimizzare il labeling con euristiche.
- E' possibile ripetere i teoremi di punto fisso di Prolog, partendo pero' da interpretazioni non di Herbrand per i termini.
- Ad esempio in $CLP(FD)$ il termine $2 + 3$ va interpretato allo stesso modo di 5 e $3 + 2$. Sono pero' termini diversi dal punto di vista sintattico.
- Chi vuole approfondire, non ha che da chiedere.

CLP(FD) in BProlog

Domain

La libreria in SICStus prolog viene caricata con:

```
:- use_module(library(clpfd)).
```

Negli altri Prolog è implicitamente caricata. Per assegnare domini a una variabile `A` o a una lista di variabili `L`:

```
A in 1..20
```

```
domain(L,-5,12)
```

(in B domain è accettato anche per variabili singole). In B e altri linguaggi si puo' anche:

```
L ::  1..20
```

Per portabilita', si può definire:

```
:- op(100,xfx, ::).
L ::  A..B :- domain(L,A,B).
```

CLP(FD) in BProlog

Domain

Per domini che non siano intervalli:

```
X in {1,2,5,10,23}
```

in SICStus, oppure

```
X in [1,2,5,10,23]
```

in B Prolog.
Operatori matematici

```
#=    #<    #>    #=<    #>=    #\=
```

Simboli di funzione

```
+    -    *    /    mod    min    max    abs
```

CLP(FD) in BProlog

Propagation

Consideriamo il goal

```
?- domain([A,B,C],1,4), A #< B, B #< C.
```

ad esso viene risposto:

```
A in [1..2],
B in [2..3],
C in [3..4]
```

Altro goal:

```
?- domain([A,B,C],1,5), A #< B, B # 2, B #< C.
```

ad esso viene risposto:

```
A in [1..3],
B in [2,4]
C in [3..5]
```

CLP(FD) in BProlog

Propagation

Consideriamo il goal

```
?- domain([A,B,C],1,4), A #< B, B #< C.
```

ad esso viene risposto:

```
A in [1..2],
B in [2..3],
C in [3..4]
```

Altro goal:

```
?- domain([A,B,C],1,5), A #< B, B # 2, B #< C.
```

ad esso viene risposto:

```
A in [1..3],
B in [2,4]
C in [3..5]
```

CLP(FD) in BProlog

Reified Constraints

Supponiamo di avere un certo numero di constraint, ad esempio cinque, e che si desideri che almeno tre di essi siano verificati nella soluzione:

```
?- constraint1 #<=> B1,
   constraint2 #<=> B2,
   constraint3 #<=> B3,
   constraint4 #<=> B4,
   constraint5 #<=> B5,
   B1+B2+B3+B4+B5 #>= 3
```

I vari B_i sono visti come constraint booleani e quindi con valori possibili 0 o 1.

CLP(FD) IN BPROLOG

TABLE CONSTRAINTS

Se disponiamo di una relazione *n*-aria (una *n*-upla di variabili che può assumere solo un certo insieme di valori) o che non ne può prendere altri, si può usare il cosiddetto Table Constraint (o combinatorial constraint). In certe situazioni è molto efficace (propagazione veloce ed effettiva). Esempi:

```
?- (A,B,C) in [ (0,1,0),(0,1,1) ].
```

```
?- (X,Y,Z,T) notin [ (0,0,0,0), (1,1,1,1) ].
```

Notate l'uso di parentesi tonde e quadre. In SICStus c'è il built-in table che fa lo stesso. Si usano però solo quadre.

CLP(FD) in BProlog

Labeling

Per istanziare una singola variabile V:

```
indomain(V)
```

Per avere particolari valori del dominio:

```
fd_max(Var,Max)
fd_min(Var,Min)
```

(guardate anche altre opzioni sul manuale). Per istanziare tutte le variabili:

```
labeling(Opzioni, Lista)
```

Vediamo le opzioni sul manuale. Sono più o meno le stesse nei vari Prolog.