Variabili di tipo "insieme"

- *In alcune applicazioni è utile avere variabili che rappresentano degli **insiemi***
  - *Una variabile con dominio finito (libreria fd) ha come dominio un insieme finito di interi*
    - *Es:* X::[1,2,3] *può assumere il valore 1, 2, oppure 3*
  - *Una variabile **"insieme finito"** ha come dominio **l'insieme delle parti** di un insieme finito di interi*
    - *Es: il dominio di X è l'insieme dei sottoinsiemi di* {1,2,3}:
      {{}, {1}, {2}, {3}, {1,2}, {1,3}, {2,3}, {1,2,3}}

Insiemi

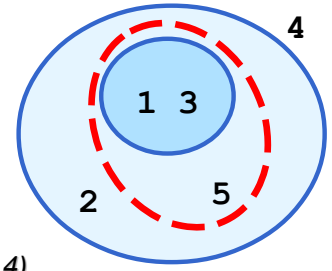
- *ECLiPSe ha varie librerie per gli insiemi (conjunto, fd_sets, cardinal, ic_sets); descriveremo* **`fd_sets`**
- *Per definire una variabile di tipo set, si usa la sintassi*

```
SetVar :: LowerBound..UpperBound
```

- *Dove* **`LowerBound`** *e* **`UpperBound`** *sono due liste di interi*
- *Ad esempio*

```
S :: [1,3]..[1,2,3,5]
```

- *significa che la variabile* **`S`**
  - *contiene sicuramente gli elementi 1 e 3*
  - *può contenere anche gli elementi 2 e 5*
  - *(ma non può contenere, ad esempio, il valore 4)*



Rappresentazione approssimata

- Si noti che questa rappresentazione è una approssimazione
- Se dovessi rappresentare in memoria tutti i valori nel dominio, dovrei usare almeno 1 bit per ogni valore
  - Sottoinsiemi di N elementi → 2^N bit
- Es. se voglio rappresentare l'insieme dei sottoinsiemi di {1,2,3} che hanno da 1 a 2 elementi

  {1}, {2}, {3}, {1,2}, {1,3}, {2,3}
- Il lower bound (intersezione) è:
- L'upper bound (unione) è:

Vincoli sugli insiemi

```
X in S
```
($X \in S$)

- `S::[]..[1,2,3,4,5,6], 3 in S`

```
yes, S = S{[3] \/ ([] .. [1, 2, 4, 5, 6])
```

```
X notin S
```
($X \notin S$)

- … `5 notin S`

```
S = S{[3] \/ ([] .. [1, 2, 4, 6])
```

- In entrambi i casi, **`X`** può essere una variabile **`fd`**.

Vincoli sugli insiemi

- `intersection(S1,S2,S3)` ⟺ $S1 \cap S2 = S3$
- `union(S1,S2,S3)` ⟺ $S1 \cup S2 = S3$
- `S1 disjoint S2` ⟺ $S1 \cap S2 = \emptyset$
- `S1 subset S2` ⟺ $S1 \subseteq S2$
- `difference(S1,S2,S3)` ⟺ $S1 \setminus S2 = S3$
- `symdiff(S1,S2,S3)` ⟺ $S3 = S1 \setminus S2 \cup S2 \setminus S1$
- `#(S,C)` ⟺ $|S| = C$

cardinalità dell'insieme. Si noti che `C` *è una variabile con dominio finito, su cui posso imporre vincoli della libreria* `fd`.

Definizione domini

- *Sintassi standard:*

```
SetVar :: []..[1,2,3,4,5,6,7]
```

Vale solo per dichiarare 1 variabile (non per liste di variabili!)

- *In alternativa, se il lower bound è l'insieme vuoto*

```
SetVar subset [1,2,3,4,5,6,7]
```

- *Se il lower bound è vuoto e l'upper bound contiene tutti gli interi compresi in un intervallo [Min,Max]*

`intset(SetVar,Min,Max)` *es:* `intset(S,1,7)`

- *Per dichiarare una lista di N variabili set, tutte da* `[]` *a* `[Min, … , Max]`*:*

```
intsets(Lista,N,Min,Max)
```

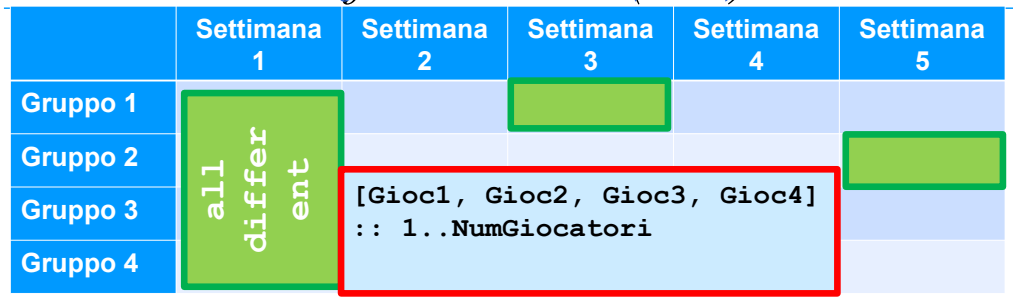

Altri vincoli

- `all_disjoint(+Sets)`
  - *Sets is a list of integers sets which are all disjoint*
- `all_union(+Sets, ?SetUnion)`
  - *SetUnion is the union of all the sets in the list Sets*
- `all_intersection(+Sets, ?SetIntersection)`
  - *SetIntersection is the intersection of all the sets in the list Sets*

Esempio: Social Golfer

- *32 giocatori di golf ogni settimana giocano in gruppi di 4 persone, per 10 settimane*
- *Vogliono far sì che ogni settimana ogni giocatore sia in un gruppo in cui non ha mai incontrato nessuno*

Social Golfer in CLP(FD)

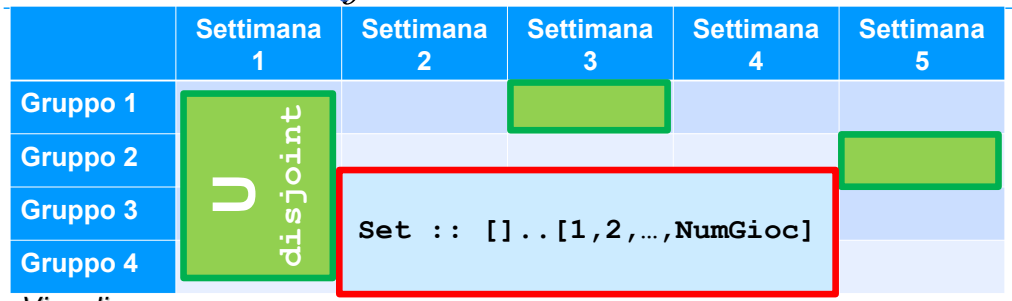


Vincoli:

- *Per ogni colonna:* `alldifferent(Lgiocatori)`
- *Per ogni coppia di celle in settimane diverse:*

  se due giocatori erano insieme nella prima settimana, allora non possono essere insieme nella seconda

Social Golfer con Set



Vincoli:

- *In ogni cella:* `#(Set,4)`
- *Per ogni colonna:* $\bigcup_i Set_i$ = [1,2,3…,NumGioc]
  - *Per ogni coppia di celle nella colonna:* `Set1 disjoint Set2`
- *Per ogni coppia di celle in settimane diverse:*

  `intersection(Set1,Set2,Int), #(Int,Card), Card::0..1`

Vincolo weight

`weight(?Set, ++Weights, ?Tot)`

- *dove*
  - `Set` *è una variabile di tipo set*
  - `Weights` *è un termine ground, sintassi* `w(ElencoValori)`
  - `Tot` *è una variabile con dominio finito*
- *Impone che il peso totale degli elementi nell'insieme* `Set` *sia pari a* `Tot`
- *Esempio:*

  1 2 3 4 5 6

  `S::[2]..[2,3,5],weight(S,w(1,4,3,7,5,2),T)`

  `Yes, T:: 4..12`

lista <->termine

- *Il vincolo* `weight` *richiede un termine* `w(ElencoPesi)`
- *Se ho i pesi in una lista, come li trasformo in un termine?*
- *Per trasformare liste in termini e viceversa, si può usare l'operatore*

  `Termine =.. Lista`

- *La* `Lista` *conterrà il funtore come primo elemento, poi a seguire gli argomenti*
- *Es:* `p(a,2,X) =.. L.`

  `yes, L = [p, a, 2, X]`
- *Es:* `T =.. [w,1,3,4].`

  `yes, T = w(1,3,4)`

Collegamento con variabili fd

`membership_booleans(?Set, ?BoolArr)`

- **`BoolArr`** *is an array of booleans describing* **`Set`**

```
?- Set :: [2]..[1,2,3],
membership_booleans(Set,BoolArr),
1 notin Set.

Set = Set{[2] \/ ([] .. [3]):Card{[1, 2]}}

BoolArr = [](0, 1, B3{[0, 1]})
```


Search con gli insiemi

`insetdomain(Set,CardSel,ElemSel,Order)`

- *dove*
  - **`Set`** *è una variabile* **`fd_set`**
  - *Gli altri parametri decidono l'ordine in cui vengono provati gli assegnamenti. Se si vuole lasciare il default, lasciarli variabili.*
- *Es:* **`S::[]..[1,2,3], insetdomain(S,_,_,_).`**

```
yes, S=[1,2,3] ;
yes, S=[1,2]   ;
yes, S=[1,3]   ;
yes, S=[1]     ;
yes, S=[2,3]
```


Esercizio

- Una donna vuole invitare *N* amici a cena.
- Per *N*(*N*-1)/6 giorni vuole organizzare delle cene, ciascuna con esattamente 3 invitati in modo tale che gli stessi 2 amici non si trovino due (o più) volte a cena insieme
- In altre parole, in ciascuno dei giorni ci devono essere 3 invitati e l'intersezione degli invitati di due giorni diversi può essere al massimo di 1 persona