UNIVERSITÀ DEGLI STUDI DI PARMA
Dipartimento di Matematica e Informatica

Corso di Laurea in Informatica

Analisi e progettazione delle regole di traduzione per l'esecuzione del linguaggio a vincoli Charme tramite JSetL

Relatore:
Chiar.mo Prof. Gianfranco Rossi

Candidato:
Alessio Bortolotti

Anno Accademico 2013/2014

Ai miei genitori,
Monica e Mauro.
A mio fratello,
Matteo.

Ringraziamenti

Vorrei ringraziare la mia famiglia, in particolare i miei genitori e mio fratello Matteo, che mi hanno spronato a tener duro nei momenti di debolezza; senza i loro consigli e le loro parole di incoraggiamento non sarei arrivato a questo traguardo della mia vita.
Insieme a loro ringrazio Jacopo e Sebastian, coloro che più di tutti mi hanno sopportato durante questi anni e con i quali ho condiviso le fatiche e le soddisfazioni del percorso universitario e con loro ringrazio gli altri compagni di corso.
Vorrei ringraziare poi il professor Gianfranco Rossi per la pazienza che ha avuto ed il tempo che mi ha dedicato durante tutto il periodo della tesi.
Infine ma non per importanza ringrazio tutti i miei amici che mi hanno sempre supportato e il cui affetto mi ha sempre aiutato nei momenti di difficoltà.

Indice

Introduzione

Nel 1989 la ditta *Bull* presenta il linguaggio di programmazione a vincoli *Charme* dedicato agli ambienti industriali.
Alcune ditte lo utilizzano per la creazione di programmi per la risoluzione di problemi caratterizzati da vincoli, per esempio ottimizzazione del carico di camion o la gestione dei turni all'interno della ditta.
Il supporto al linguaggio e il linguaggio stesso viene però a mancare verso la fine degli anni '90 dato che il linguaggio viene acquisito da una ditta che non ha interessi nel tenere attivo il progetto.
Le ditte si sono quindi trovate ad avere programmi utilizzati al proprio interno scritti in un linguaggio che di fatto è sparito.
La seguente tesi si propone di analizzare e descrivere una metodologia per poter tradurre per adesso manualmente, ma in futuro automaticamente, il codice Charme in codice Java con l'utilizzo della libreria JSetL per la risoluzione dei vincoli.

Capitolo 1

Il linguaggio Charme

1.1 Breve panoramica su Charme

Charme è un linguaggio imperativo avente i vincoli come istruzioni. Il suo utilizzo principale riguarda l'ambito industriale ed è stato creato per la scrittura di codice adatto alla risoluzione di vincoli.

1.2 I dati

1.2.1 I tipi di dato atomici

Il linguaggio Charme riconosce due tipi di oggetti atomici:

- Numerici
  Viene considerato oggetto numerico ogni elemento appartenente all'insieme:

  $$\{x \in Z \mid MININT \leq x \leq MAXINT\}$$

  dove:
  MININT e MAXINT sono due valori interi che variano in base all'elaboratore.
  Un oggetto numerico con valore positivo può essere preceduto da +.
  Un oggetto numerico con valore negativo deve essere preceduto da -.

- Simbolici
  Viene considerato oggetto simbolico:

  – ogni sequenza di caratteri alfanumerici, con l'aggiunta del carattere "_", che inizia con una lettera minuscola;
  – ogni sequenza di caratteri racchiusa tra apici.

1.2.2 Le costanti

In Charme sono presenti tre tipi di costanti:

- Numeri interi.
  Viene considerato numero intero ogni sequenza di cifre.
- Stringhe.
  Viene considerata stringa ogni sequenza di caratteri racchiusa tra virgolette.
- Numeri reali.
  Viene considerato numero reale ogni oggetto della forma:
  <ParteIntera>.<ParteDecimale>[<Esponente>]
  dove:
  - <ParteIntera>: numero intero
  - <ParteDecimale>: numero intero
  - <Esponente>: è un numero intero opzionale, preceduto da e o E che rappresenta l'esponende da assegnare a 10
    esempio:
    10.02e5; $\iff 10.02 * 10^5$

1.2.3 Le variabili semplici

Il nome di una variabile semplice è composto da una sequenza di caratteri alfanumerici che inizia con una lettera maiuscola o un "_".
Una variabile semplice può contenere al proprio interno ogni oggetto Charme per esempio: una stringa, un intero, un simbolo, un array, ecc.
Una volta dichiarata tramite le keywords local o global oppure tramite assegnamento una variabile semplice si dice istanziata.
In Charme le variabili semplici sono non tipizzate, quindi il tipo di dato contenuto non è vincolato ma può mutare a seconda del valore assegnato.

1.2.4 Le D-Var

Le D-Var sono variabili a cui è associato un dominio di valori che rappresenta tutti i possibili valori che la D-Var può assumere.

Sintassi della dichiarazione di una D-Var:
<NomeVariabile> in <Dominio>
con:

- <NomeVariabile>: può essere una variabile semplice.

- in: indica che <Dominio> viene assegnato come dominio a <NomeVariabile>.
- <Dominio>: deve essere un dominio valido.

Ad una D-Var può anche essere assegnato come dominio l'insieme di tutti gli interi; per fare ciò la si dichiara come:

- <nomeVariabile> in INT
  oppure
- int <nomeVariabile>

1.2.5 I Domini

Vengono identificati due tipi di domini:

- Dominio numerico:
  contiene al proprio interno soltanto valori numerici.
- Dominio simbolico:
  contiene al proprio interno soltanto valori simbolici.

In Charme non viene accettato come dominio un insieme contenente sia valori numerici che simbolici.
Un dominio intero può essere definito da:

- un intervallo:
  $min...max$
  dove:
  - min è un numero intero
  - max è un numero intero
  - $min \leq max$
- una lista:
  $[< N1 >, < N2 >, ..., < Nn >]$
  dove:
  - $\forall i \in [1, n]$, $< Ni >$ è un numero intero
  - gli Ni possono essere in qualsiasi ordine e possono anche non essere distinti.

1.2.6 Gli Array

Gli array sono la struttura dati base di Charme.
Gli elementi di un array possono essere un oggetto Charme di qualsiasi, anche array.
Un array può essere multi-dimensionale.
Per accedere ad un elemento di un array si utilizza un indice. In Charme un indice può essere sia un numero intero che un simbolo.

Per accedere agli elementi di un array si possono utilizzare due metodi equivalenti:

- V[i] restituisce l'elemento corrispondente all'indice i del vettore V
- V^[i] restituisce l'elemento corrispondente all'indice i del vettore V

Sono presenti due modi basilari in cui definire un array mono-dimensionale:

- Implicitamente
  L'array (in questo caso chiamato vettore), viene specificato assegnado ad una variabile un insieme di valori che rappresentano gli elementi del vettore.
  L'insieme dei valori può essere specificato in due modi:

  1 tramite una lista di elementi
  <nomeVettore> = [<e1>,<e2>,...,<en];
  dove:

  * <nomeVettore> è un nome non ancora utilizzato.

  Esempio:

  $$V = [a, b, c, d]$$
  $$V[1] \implies a$$
  $$V[2] \implies b$$
  $$V[3] \implies c$$
  $$V[4] \implies d$$

  2 tramite un intervallo
  <nomeVettore> = min...max;
  dove:

  * min è un intero
  * max è un intero
  * $min \leq max \implies$ tutti gli interi compresi tra min e max
  * $min > max \implies$ vettore vuoto

Esempio:

$$V = 5..10;$$
$$V[1] \implies 5$$
$$V[3] \implies 7$$
$$V[6] \implies 10$$

- Esplicitamente
  Per dichiarare esplicitamente un array si utilizza la keyword array:

  $$array < nomeArray >::< vettoreDimensioni >$$

  dove:

  - <nomeArray> è una variabile semplice non ancora utilizzata;
  - <vettoreDimensioni> è una lista che contiene valori tutti distinti che rappresentano gli indici con i quali si può accedere agli elementi dell'array; dato che un array potrebbe essere multi-dimensionale gli elementi di <vettoreDimensioni> potrebbero essere a loro volta array.

  Esempi:

  $$array\ V1 :: [1, 2, 5, 6]$$
  $$array\ V2 :: [a, f, h, index]$$

  È possibile inizializzare un array direttamente un insieme di valori
  Esempio:

  $$< vettore >= [1, 5, a, 2] :: [1, 2, 5, 6]$$
  $$< vettore > [1] \implies 1$$
  $$< vettore > [2] \implies 5$$
  $$< vettore > [5] \implies a$$
  $$< vettore > [6] \implies 2$$

  Una sintassi un po' più completa:
  $< nomeArray >::< vettoreDimensioni >$ of $[< Domain >]$

  - <nomeArray> è una variabile semplice non ancora utilizzata;
  - <vettoreDimensioni> è una lista che contiene elementi tutti distinti e rappresentano gli indici con i quali si può accedere agli elementi dell'array.

– <Domain> è un dominio valido;
La presenza di <Domain> forza ogni elemento di <nomeArray> ad essere una D-Var a cui viene associato come dominio <Dominio>.

Esempio:
$V :: [a, f, h, index]$ of $[1, 2, 3]$
$arr :: [3, 2, 5, 100]$ of 1..10

1.2.7 Variabili locali, variabili globali e rispettivi scope

- Variabili locali
  Una variabile locale può essere definita sia definendo semplicemente una variabile semplice o una D-Var, sia facendo precedere alla definizioni la keyword *local*.
  Una variabile locale è visibile nel blocco in cui è definita e in tutti i suoi sotto-blocchi, non è invece visibile nei blocchi più esterni.

- Variabili globali
  Una variabile può essere dichiarata globale facendo precedere alla dichiarazione le keyword *global* o *memory*.
  Le variabili globali sono visibili dal punto in cui sono dichiarate fino alla fine del codice.

1.2.8 Espressioni Numeriche

Un'espressione numerica è un'espressione aritmetica contenente:

- variabili semplici che rappresentano numeri interi o numeri reali;
- D-Var con dominio numerico;
- costanti intere o reali;

1.2.9 Convenzioni Sintattiche

- Una variabile deve iniziare con una lettera maiuscola o con "_".
- Una procedura deve iniziare con una lettera minuscola.

1.3 Le istruzioni

In Charme un'istruzione può essere un'istruzione semplice,un costrutto(istruzione strutturata) o un blocco di istruzioni.
Un'istruzione semplice può essere:

- una definizione di variabile

- un'imposizione di un vincolo
- una chiamata a procedura

1.3.1 Constraint

Un constraint è una relazione che deve essere valida per la risoluzione del problema.
Relazioni:

- Uguaglianza:

$$A = B$$

  dove: A e B sono variabilili semplici, D-Var o espressioni numeriche.
  Il constraint risulta vero se e solo se il valore di A e quello di B sono uguali o se i due domini coincidono.

- Disuguaglianza:

$$A! = B$$

  dove: A e B sono variabilili semplici, D-Var o espressioni numeriche.
  Il constraint risulta vero se e solo se il valore di A e quello di B sono diversi o se i due domini sono disgiunti.

- operatori di confronto:
  - $<$

$$A < B$$

    dove: A e B sono variabilili semplici, D-Var o espressioni numeriche.
    Il constraint risulta vero se e solo se il valore di A è minore di quello di B o se:

$$\forall a \in dom(A).\forall b \in dom(B).a < b$$

  - $\leq$

$$A \leq B$$

    dove: A e B sono variabilili semplici, D-Var o espressioni

numeriche.
Il constraint risulta vero se e solo se il valore di A è minore o uguale a quello di B o se:

$$\forall a \in dom(A).\forall b \in dom(B).a \leq b$$

– $>$

$$A > B$$

dove: A e B sono variabilili semplici, D-Var o espressioni numeriche.
Il constraint risulta vero se e solo se il valore di A è maggiore di quello di B o se:

$$\forall a \in dom(A).\forall b \in dom(B).a > b$$

– $\geq$

$$A \geq B$$

dove: A e B sono variabilili semplici, D-Var o espressioni numeriche.
Il constraint risulta vero se e solo se il valore di A è maggiore o uguale a quello di B o se:

$$\forall a \in dom(A).\forall b \in dom(B).a \geq b$$

- all_in
  Sintassi:

```
A all_in Dom;
```

  dove:

  – A è un array generico
  – Dom è un dominio

  Semantica:

$$\forall a \in A.dom(a) \equiv Dom$$

  La stessa cosa vale anche per gli array multi-dimensionali.

- all_diff
  Sintassi:

```
all_diff A;
```

dove:
A è un array generico
Semantica:

$$\forall a_i, a_j \in A, i \neq j \implies a_i \neq a_j$$

La stessa cosa vale anche per gli array multi-dimensionali.

- distribute

  Sintassi:

```
distribute(<number>,<Array>,<valuesArray>);
```

  dove:

  - *<number>* è un valore intero
  - *<Array>* è un array di D-Var con dominio solo numerico o solo simbolico
  - *<valuesArray>* è un array monodimensionale i cui valori devono essere o solo numerici o solo simbolici

  Semantica: *<number>* elementi di *<Array>* devono avere il valore uguale ad uno di quelli presenti in *<valuesArray>*.

- distribution

  Sintassi:

```
distribution(<number>,<Array>,<valuesArray>,<
    quotasArray>);
```

  dove:

  - *<number>* è un valore intero
  - *<Array>* è un array di D-Var con dominio solo numerico o solo simbolico
  - *<valuesArray>* è un array monodimensionale i cui valori devono essere o solo numerici o solo simbolici
  - *<quotasArray>* deve essere un array monodimensionale con dimensione uguale a *<valuesArray>* contenente valori numerici

Semantica: <*number*> elementi di <*Array*> devono avere il valore uguale ad uno di quelli presenti in <*valuesArray*> in più:
$\forall i \in [1, length(< valuesArray >)], < valuesArray > [i]$ deve essere presente $< quotasArray > [i]$ volte all'interno di <*Array*>

- Generate

```
generate <oggettoCharme>
```

dove <*oggettoCharme*> può essere:

  - D-Var
  - Array monodimensionale
  - Array bidimensionale
  - Array tridimensionale

Questa funzione attiva il processo di labeling sull'oggetto <*oggettoCharme*>.

- find_all

```
find_all <blocco>
```

dove:
<*blocco*> è un qualsiasi blocco Charme.
Questa funzione forza il risolutore non a trovare soltanto la prima soluzione bensì a trovarle tutte, eper ciascuna viene eseguito il codice presente in <*blocco*>.

- minimize

```
minimize D;
```

dove:
D è una D-Var.
questa istruzione permette di risolvere il problema cercando di minimizzare il valore di *D*.

1.3.2 Chiamata di una procedura

La sintassi di una chiamata a procedura è:
$< nomeProcedura >$[(*parametro_1,parametro_2,...,parametro_n*)]
dove:
<*nomeProcedura*> è una una sequenza di caratteri alfanumerici che inizia con una lettera maiuscola o un "_" che rappresenta il nome di una procedura precedentemente definita 1.5

Se la funzione non prevede parametri le parentesi tonde vengono omesse.

1.3.3 Blocchi

In Charme un blocco ha la seguente forma sintattica:

```
{
        istruzione_1;
        istruzione_2;
        ...
        istruzione_n
}
```

dove istruzione_i può essere:

- un'istruzione semplice (terminante con ;)
- un blocco
- un costrutto

Una variabile dichiarata in un blocco è visibile all'interno del blocco dal punto in cui è dichiarata fino al termine del blocco e nei suoi sotto blocchi.

1.4 I costrutti (istruzioni strutturate)

1.4.1 for

Il ciclo for in Charme si presenta nella forma:

```
for <Variabile> in <Array> do <Istruzione>
```

dove:

- <Variabile> viene chiamata variabile di iterazione, ad ogni iterazione del ciclo assume il valore successivo all'interno dell'array e può essere utilizzata all'interno di <Istruzione> per indicare l'elemento di <Array> attualmente selezionato; <Variabile> è una variabile locale al costrutto for e nasconde una possible variabile dichiarata all'esterno del corpo del for.
- <Array> è un array di elementi;
- <Istruzione> può essere una istruzione semplice, un costrutto oppure un blocco di istruzioni;

  Se un' istruzione nel ciclo fallisce, il ciclo for fallisce e quindi la risoluzione del problema fallisce.

Esempi:

```
for I in 1..5 do print I;
```

Risultato:
I = 1
I = 2
I = 3
I = 4
I = 5

```
A :: [2..4];
for E in 2..4 do
{
        A[E] = E * E;
        print A[E];
}
```

Risultato:
A[2] = 4
A[3] = 9
A[4] = 16

```
A :: [2..4];
for E in 2..4 do
{
        A[E] = E ;
        A[3] != 3;
}
```

Risultato:
alla seconda iterazione la seconda istruzione fallisce quindi tutto il ciclo fallisce ed il problema non ha soluzione;

1.4.2 while

Il ciclo while in Charme si presenta nella forma:

```
while <Condizioni> do <Istruzione>
```

dove:

- <Condizioni> è un'istruzione semplice,un costrutto o un qualsiasi blocco Charme;
  se <Condizioni> fallisce ogni volta, il costrutto while termina con successo;

- <Istruzione> può essere un'istruzione semplice oppure un blocco di istruzioni;

  <Istruzione> viene eseguito ogni volta che <Condizioni> ha successo, se <Istruzione> fallisce sempre il costrutto while fallisce.

Esempi:

```
while fail; do print 10;
```

Risultato:
print 10 non viene mai eseguito ma il ciclo while ha successo;

```
while {} do fail;
```

Risultato:
il ciclo è infinito ma, dato che il corpo fallisce, fallisce sempre

1.4.3 if-then-else

Il costrutto if-then-else in Charme si presenta nella forma:

```
if <Condizioni>
then <Successo>
[else <Fallimento>]
```

dove:

- <Condizioni> è un'istruzione semplice o un qualsiasi blocco Charme;
- <Successo> corpo eseguito se <Condizioni> ha successo, può essere un'istruzione semplice o un blocco;
- <Fallimento> corpo eseguito se il ramo else è presente e se <Condizioni> fallisce, può essere un'istruzione semplice o un blocco;

La parte else <Fallimento> è opzionale e può essere omessa, in questo caso se <Condizioni> fallisce il costrutto if-the-else non fa niente e termina con successo
Esempi:

```
while fail; do print 10;
```

Risultato:
print 10 non viene mai eseguito ma il ciclo while ha successo;

1.5 Le procedure

In Charme una procedura viene definita con la seguente sintassi:

```
define <nomeProcedura> [<parametriFormali>] <block>
```

dove:

- <nomeProcedura> è una una sequenza di caratteri alfanumerici che inizia con una lettera maiuscola o un "_" non ancora utilizzata.

- <parametriFormali> è facoltativo, è una sequenza di variabili semplici racchiuse tra parentesi tonde e separate da una virgola.
- <block> può essere sia un blocco vuoto che un insieme di istruzioni racchiuso tra parentesi graffe.

Esempi:

```
define printA(A)
{
        print A;
}
```

```
define doNothing
{}
```

```
define printOne
{
        print 1;
}
```

1.6 Gli eventi

1.6.1 when_known

Sintassi:

```
when X do <block>
```

dove:

- X può essere:
  - una variabile semplice
  - un array
- <block> è un blocco

Semantica:

- se X è una variabile semplice *<block>* viene eseguito soltanto quando X risulta *bound* *
- se X è un array, *<block>* viene eseguito soltanto nel momento in cui tutte le variabili di X risultano *bound*.

*Una variabile risulta *bound* quando il suo dominio contiene soltanto un valore

Capitolo 2

JSetL

2.1 Introduzione a JSetL

JSetL è una libreria Java, sviluppata presso il Dipartimento di Matematica e Informatica, che unisce il paradigma di programmazione orientata agli oggetti di Java con i concetti dei linguaggi CLP, come variabili logiche, liste, risoluzione di vincoli e non determinismo;il suo utilizzo principale è il supporto alla programmazione dichiarativa a vincoli.

2.2 I tipi di dato utilizzati

2.2.1 IntLVar

Le *IntLVar* sono variabili logiche alle quali viene assegnato un dominio di valori interi che rappresenta l'insieme di valori che la singola variabile può assumere. Due sintassi principali:

```
IntLVar nome = new IntLVar();
```

semantica:
viene creata una variabile IntLVar con associato un dominio contenente tutti gli interi.

```
IntLVar nome = new IntLVar(a,b);
```

dove:

- a è un numero intero
- b è un numero intero
- $a \leq b$

semantica: viene creata una variabile IntLVar con associato il dominio

$$\{x \in Z | a \leq x \leq b\}$$

quando il dominio di una IntLVar viene ristretto ad un singolo elemento la variabile si dice *bound* in tutti gli altri casi viene definita *unbound*.

2.2.2 MultiInterval

La classe *MultiInterval* rappresenta un insieme di interi $M \subset Z$ definito da $n > 0$ intervalli $I_1, I_2, ..., I_n$, $I_\alpha \backslash \emptyset$ tali che:

- $M = I_1 \cup I_2 \cup ... \cup I_n$
- $I_1 \prec I_2 \prec ... \prec I_n$

2.3 I Constraints

2.3.1 Cos'è un constraint

Un constraint rappresenta un'operazione che può essere applicata ad una variabile logica o ad un insieme di quest'ultime.
Un constraint in JSetL è un'espressione che può avere una delle seguenti forme:

- (*constraints atomici*)
  - il constraint vuoto
  - $e_0.op(e_1, ..., e_n)$ oppure $op(e_0, e_1, ..., e_n)$ with $n = 0, ..., 3$

  dove *op* è il nome del constraint e $e_i (0 \leq i \leq 3)$ sono espressioni il cui dipo dipenda da *op*.
- (*constraints composti*)
  - c1.**and**(c2) (congiunzione)
  - c1.**or**(c2) (disgiunzione)
  - c1.**impliesTest**(c2) (implicazione)

2.3.2 Come creare un nuovo constraint

La creazione di un nuovo Constraint in JSetL richiede una procedura ben definita che richiede 4 passi:

1 Creare una nuova classe che deve estendere la classe *NewConstraintsClass*

```
public class <nomeClasse> extends NewConstraintsClass {

    public <nomeClasse>(SolverClass slv)
    {
        super(slv);
    }

    \\Metodi derivati dai passi successivi


}
```

2 Bisogna creare un metodo pubblico che deve avere la seguente struttura

```
    public Constraint <nomeConstraint>(parametro_1,
        parametro_2, ... , parametro_n) {
        return new Constraint("<nomeConstraint>",
            parametro_1, parametro_2, ... , parametro_n
            );
    }
```

3 Deve implementare il metodo *user_code* come segue

```
protected void user_code(Constraint c) throws Failure,
    NotDefConstraintException
    {
        if (c.getName().equals("<nomeConstraint>")) <
            nomeConstraint>(c);
        else throw new NotDefConstraintException();
    }
```

4 Bisogna infine creare un metodo privato che rappresenta il vero e proprio constraint

```
private void <nomeConstraint>(Constraint c) {
        \\Corpo del constraint
        return;
    }
```

2.4 La classe SolverClass

La classe *SolverClass* contiene il risolutore di vincoli che permette la risoluzione del problema.
In questo elaborato vengono utilizzati principalmente 2 metodi della classe:

- ```
  void add(Constraint c)
  ```

  che permette d'inserire un nuovo vincolo(*constraint*) nel solver.

- ```
  void solve()
  ```

  che fa partire il meccanismo di risoluzione di un vincolo memorizzato nel solver.
  Nel caso in cui non si trovasse una soluzione il metodo genera un'eccezione del tipo *failure exception*.

Capitolo 3

Regole di traduzione da Charme a JSetL

3.1 Introduzione alla traduzione

Come abbiamo visto nel secondo capitolo il linguaggio Charme gestisce due tipi di valori:

- Numerici
- Simbolici

Il focus di questa tesi è stato rivolto esclusivamente ai valori numerici.
Questo significa in particolare che non sono state ricercate regole di traduzione e di gestione dei domini simbolici e delle relative variabili semplici.
Inoltre è stato scelto di restringere il numero di dimensioni degli array a 3.
Questo comporta che tutti gli array ad 1, 2 o 3 dimensione vengono tradotti e gestiti correttamente mentre quelli con un numero maggiore di dimensioni non sono stati studiati.

3.2 Struttura del programma finale

Il programma java ha la seguente struttura:

```
package <NomePackage>;

import JSetL.*;
import JSetL.lib.*;

public class <NomeProgramma> {
/*
```

```
        In questa zona vengono definite tutte le variabili e
             le enumerazioni utilizzate nel programma.

*/

        static SolverClass solver;
        static MetaOps meta;
        static LArrayOps arrayOps;

static void Initialize() throw Exception{


        solver = new SolverClass();
        meta = new MetaOps(solver);
        arrayOps = new LArrayOps(solver);

        /*
                Qui verranno inizializzate le variabili
                     precedentemente definite.
        */
}

/*
        In questa zona saranno inserite le traduzioni delle
             procedure definite nel codice Charme
*/

public static void main(String[] argv) throws Exception{
        initialize();
        /*
                resto del codice necessario per gestire la
                     risoluzione del problema e per stampare
                      o salvare la soluzione
        */
}
```

3.3 Traduzione delle variabili

3.3.1 Variabili semplici

```
        Var = I;
```

dove:

- Var è un nome di variabile
- I è un intero

Traduzione:

```
Integer Var = I;
```

3.3.2 D-Var

```
D in Dom
```

dove:

- D è un nome di variabile
- Dom è un intero

Traduzione:

```
IntLVar D = new IntLVar(Dom);
```

dove: $\overline{Dom}$ è la traduzione del dominio *Dom*

3.3.3 Array

- Array monodimensionali con intervallo come dimensione e indice

  ```
  array A :: min..max [of Dom];
  ```

  traduzione: Viene prima creata la variabile

  ```
  IntLVar[] A;
  ```

  poi viene generato l'array:

  - Se il dominio non è presente

    ```
    A = LArray.generateArray(new MultiInterval(
        min,max));
    ```

  - Se il dominio è presente
    - Se il dominio è un intervallo *minD..maxD*

      ```
      A = LArray.generateArray(new
          MultiInterval(min,max),new
          MultiInterval(minD,maxD));
      ```

    - Se il dominio è una lista *[<eD1>, ..., <eDn>]*

      1 Verrà creata l'enumerazione e la sua istanziazione che assumeremo venga chiatama *enumDomInst*;

2
```
A = LArray.generateArray(new
    MultiInterval(min,max),
    enumDomInst);
```

- Array monodimensionali con enumerazione come dimensione e indice

```
array A :: [<e1>,<e2>,...,<en>];
```

con: <e1>...<en> valori interi o letterali
Traduzione:
viene prima creata la variabile e l'Enum

```
IntLVar[] A;
public enum enum_i
{
        e1(0), e2(1),.., en(n-1);
        int pos;
        private enum_i(int pos)
        {
                this.pos = pos;
        }
        public Integer getOrdinal()
        {
                return pos;
        }
}

enum_i enum_iInst = enum_i.e1;
```

poi viene generato l'array:

  - Se il dominio non è presente

```
A = LArray.generateArray(enum_iInst);
```

  - Se il dominio è presente
    * Se il dominio è un intervallo *minD..maxD*

```
A = LArray.generateArray(enum_iInst,new
    MultiInterval(minD,maxD));
```

    * Se il dominio è una lista *[<eD1>, ..., <eDn>]*
      1 Verrà creata l'enumerazione e la sua istanziazione che assumeremo venga chiatama *enumDomInst*;
      2
```
A = LArray.generateArray(enum_iInst,
    enumDomInst);
```

- Array bididimensionali
  Indichiamo con O_1 e O_2 due oggetti ciascuno dei quali può essere una lista o un intervallo, indichiamo poi con $\overline{O_1}$ e $\overline{O_2}$ le rispetive traduzioni.

```
array A :: [O1 ,O2] [of Dom];
```

  Traduzione:
  Viene prima creata la variabile

```
IntLVar[][] A;
```

  Poi viene generato l'array

  – Se il dominio non è presente

```
A = LMatrix2D.generateArray(O1, O2);
```

  – Se il dominio è presente

    ∗ Se il dominio è un intervallo *minD..maxD*

```
A = LMatrix2D.generateArray(O1, O2, new
    MultiInterval(minD,maxD));
```

    ∗ Se il dominio è una lista *[<eD1>, ..., <eDn>]*

      1 Verrà creata l'enumerazione e la sua istanziazione che assumeremo venga chiatama *enumDomInst*;

      2

```
A = LMatrix2D.generateArray(O1, O2,
    enumDomInst);
```

- Array tri-dimensionali Si rimanda al caso precedente per l'analisi della sintassi

```
array A :: [O1, O2, O3] [of Dom];
```

  Traduzione:
  Viene prima creata la variabile

```
IntLVar[][][] A;
```

  Poi viene generato l'array

  – Se il dominio non è presente

```
A = LMatrix3D.generateArray(O1, O2,O3);
```

  – Se il dominio è presente

* Se il dominio è un intervallo *minD..maxD*

```
A = LMatrix3D.generateArray(O1, O2,O3,
    new MultiInterval(minD,maxD));
```

* Se il dominio è una lista *[<eD1>, ..., <eDn>]*

  1 Verrà creata l'enumerazione e la sua istanziazione che assumeremo venga chiatama *enumDomInst*;

  2
```
A = LMatrix3D.generateArray(O1, O2,
    O3, enumDomInst);
```

3.3.4 Domini

In Charme sono presenti 2 metodologie per dichiarare un dominio:

- Intervalli
- Liste di elementi

Analizziamoli separatamente:

1) Intervalli:
   Sintassi

   $$<\text{nomeDominio}> = min...max;$$

   dove:

   - <nomeDominio> è un nome di variabile non ancora utilizzato
   - $min \in N, max \in N$
   - $min \leq max$

   Traduzione:

```
static final MultiInterval <nomeDominio> = new
    MultiInterval(min,max);
```

2) Liste di elementi:
   Sintassi:

   $$<\text{nomeDominio}> = [< E1 >, < E2 >, .., < En >];$$

   dove:

   - <nomeDominio> è un nome di variabile non ancora utilizzato

– $\forall i, j \in [1, n], i \neq j \implies < Ei > \neq < Ej >$

Traduzione: La traduzione consiste nella creazione di una enumerazione e nell'istanziazione di un elemento del tipo enumerativo appena creato;

```
        public enum <nomeDominio>
        {
                <E1>(0), <E2>(1) ,.., <En>(n-1);
                int pos;
                private <nomeDominio>(int pos)
                {
                        this.pos = pos;
                }
                public Integer getOrdinal()
                {
                        return pos;
                }
        }

<nomeDominio> <nomeDominio>Inst = <nomeDominio>.<E1>;
```

Esempio:

In Charme

```
Family = [Children, Daughter_Son,Daughter_Son];
```

In JSetL

```
public enum Family
        {
                Children(0),
                Daughter_Son(1),
                Daughter_Son(2);

                int pos;
                private Family(int pos)
                {
                        this.pos = pos;
                }
                public Integer getOrdinal()
                {
                        return pos;
                }
        }

Family FamilyInst = Family.Children;
```

3.3.5 Intervalli

Sintassi:

$$min...max$$

dove:

- $min \in N, max \in N$
- $min \leq max$

Traduzione:

```
new MultiInterval(min,max)
```

3.4 Espressioni

In questa sezione si userà la seguente notazione:

- V per indicare una variabile semplice (con valore numerico) o una costante intera
- D per indicare una D-Var
- E per indicare un'espressione (che può contenere interi, variabili semplici o D-Var)
- $\overline{e}$ per indicare la traduzione di un'espressione e
- $\overset{\sim}{.}$ per indicare l'analisi e la traduzione di espressioni contenenti elementi Charme ed IntLVar
- I per indicare una IntLVar

Si fa notare che la traduzione di un'espressione può portare a due risultati:

1 ad una IntLVar
se l'espressione contiene almeno una D-Var, infatti le operazioni nella classe IntLVar restituiscono IntLVar;

2 ad un valore numerico
se tutti i termini dell'espressione sono numerici.

Operazioni:

- Casi base:
  - $\overline{D} = D$
  - $\overline{V} = V$

  - $\widetilde{I} = I$
  - $\widetilde{V} = V$
  - $\widetilde{D} = D$

- addizione
  - $\overline{D + V} =$

```
D.sum(V)
```

  - $\overline{V + D} =$

```
D.sum(V)
```

  - $\overline{D1 + D2} =$

```
D1.sum(D2)
```

  - $\overline{D + E} =$

```
D.sum(E);
```

  - $\overline{E + D} =$

```
D.sum(E);
```

  - $\overline{E + V} =$

$$\widetilde{\overline{E} + V}$$

  - $\overline{E1 + E2} =$

$$\widetilde{\overline{E1} + \overline{E2}}$$

  - $\widetilde{I + V} =$

```
(I).sum(V)
```

  - $\widetilde{V + I} =$

```
(I).sum(V)
```

  - $\widetilde{I1 + I2} =$

```
(I1).sum(I2)
```

– $\overline{V1 + V2} =$

```
V1 + V2
```

- sottrazione

– $\overline{D - V} =$

```
D.sub(V)
```

– $\overline{V - D} =$

```
(D.mul(-1)).sum(V)
```

– $\overline{D1 - D2} =$

```
D1.sub(D2)
```

– $\overline{D - E} =$

```
D.sub(E);
```

– $\overline{E - D} =$

```
(D.mul(-1)).sum(E)
```

– $\overline{E - V} =$

$$\widetilde{\overline{E} - V}$$

– $\overline{E1 - E2} =$

$$\widetilde{\overline{E1} - \overline{E2}}$$

– $\widetilde{I - V} =$

```
(I).sub(V)
```

– $\widetilde{V - I} =$

```
((I).mul(-1)).sum(V)
```

– $\widetilde{I1 - I2} =$

```
(I1).sub(I2)
```

- $\overline{V1 - V2} =$

```
V1 - V2
```

- prodotto
  - $\overline{D * V} =$

```
D.mul(V)
```

  - $\overline{V * D} =$

```
D.mul(V)
```

  - $\overline{D1 * D2} =$

```
D1.mul(D2)
```

  - $\overline{D * E} =$

```
D.mul(E)
```

  - $\overline{E * D} =$

```
D.mul(E)
```

  - $\overline{E * V} =$

$$\widetilde{\overline{E} * V}$$

  - $\overline{E1 * E2} =$

$$\widetilde{\overline{E1} * \overline{E2}}$$

  - $\widetilde{I * V} =$

```
(I).mul(V)
```

  - $\widetilde{V * I} =$

```
(I).mul(V)
```

  - $\widetilde{I1 * I2} =$

```
(I1).mul(I2)
```

- $\overline{V1 * V2} =$

```
V1 * V2
```

- divisione
  - $\overline{D/V} =$

```
D.div(V)
```

  - $\overline{V/D} =$

    Per prima cosa bisogna aggiungere la definizione della variabile:

```
static IntLVar tDi;
```

    dove i è un indice incrementale(infatti potrebbe capitare di avere più situazioni simili a questa).
    successivamente nella funzione initialize si aggiunge l'istruzione:

```
tDi = new IntLVar(V);
```

    In questo modo viene assegnato come unico valore possibile per la variabile tDi il valore V.

    poi dove devo inserire il comando aggiungo:

```
tDi.div(D)
```

  - $\widetilde{V/I} =$

    Per prima cosa viene aggiunta la definizione della variabile:

```
static IntLVar tDi;
```

    dove i è un indice incrementale(infatti potrebbe capitare di avere più situazioni simili a questa).
    successivamente nella funzione initialize si aggiunge l'istruzione:

```
tDi = new IntLVar(V);
```

    In questo modo viene impostato come unico valore possibile per la variabile tDi il valore V.

    poi dove devo inserire il comando aggiungo:

```
tDi.div(I)
```

  - $\widetilde{I/V} =$

```
(I).div(V)
```

- $\widetilde{I1/I2} =$

```
I1.div(I2)
```

- $\overline{D1/D2} =$

```
D1.div(D2)
```

- $\overline{D/E} =$

```
D.div(E)
```

- $\overline{E/D} =$

$$\widetilde{\overline{E}/D}$$

- $\overline{E/V} =$

$$\widetilde{\overline{E}/V}$$

- $\overline{E1/E2} =$

$$\widetilde{\overline{E1}/\overline{E2}}$$

- $\widetilde{I/E} =$

```
(I).div(E)
```

- $\overline{V1/V2} =$

```
V1 / V2
```

3.5 Costrutti

Notazione:

- B per indicare un blocco
- $\overline{B}$ per indicare la traduzione del blocco B

3.5.1 for

Per tradurre il ciclo for bisogna distinguere in base alla tipologia di oggetto su cui il ciclo opera:

- un intervallo:
  In Charme abbiamo:

```
for Var in min..max do B
```

La traduzione sarà:

```
for(Integer Var : new MultiInterval(min,max))
{
        B
}
```

- una lista:
  In Charme abbiamo:

```
for <Variabile> in [<N1>,<N2>,...,<Nn>] do B
```

La traduzione sarà:

```
for(Integer I : new Integer[]{<N1>,<N2>,...,<Nn
    >})
{
        B
}
```

3.5.2 while

In Charme abbiamo:

```
while B1 do B2
```

Per poter procedere alla traduzione di questo ciclo dobbiamo tradurre B1 in una funzione che ritorna un booleano a seconda che tutti i vincoli siano o meno verificati, questo perchè in charme esiste la possibilità di avere nello condizioni non soltanto vincoli ma anche istruzioni mentre in java questo non è possibile.

Quindi il procedimento sarà:
per prima cosa creare una funzione della forma

```
boolean fCond_i(parametro_1,...,parametro_n)
{
        B1
}
```

Una volta creata la funzione nel codice inseriremo:

```
while (fCond_i(parametro_1,...,parametro_n))
{
        (*$\overline{B2}$*)
}
```

3.5.3 if-then-else

In Charme abbiamo:

```
if
        B1
then
        B2
else
        B3
```

Per poter procedere alla traduzione di questo costrutto bisogna tradurre B1 in una funzione che restituisca un booleano a seconda che tutti i vincoli siano o meno verificati; questo perchè in Charme esiste la possibilità di avere nelle condizioni non soltanto vincoli ma anche istruzioni mentre in java questo non è possibile.

Quindi il procedimento sarà:

per prima cosa creare una funzione della forma

```
boolean fCond_i(parametro_1,...,parametro_n)
{
        B1
}
```

Una volta creata la funzione nel codice inseriremo:

```
if (fCond_i(parametro_1,...,parametro_n))
{
        B2
}
else
{
        B3
}
```

3.6 Istruzioni

Notazione utilizzata:

- V per indicare una variabile semplice (con valore numerico) o un numero intero
- D per indicare una D-Var o una IntLVar(derivata da una traduzione di un'espressione)
- E per indicare un'espressione (che può contenere interi, variabili semplici o D-Var)

- $\dot{-}$ per indicare la traduzione di un'espressione
- $\dot{\sim}$ per indicare l'analisi e la traduzione di espressioni contenenti elementi Charme ed IntLVar
- I per indicare un IntLVar
- A per indicare il nome di un array
- Dom per indicare un Dominio Charme
- $\overline{Dom}$ per indicare la traduzione di Dom
- Pi indica il parametro i_esimo
- NP per indicare il nome di una procedura

3.6.1 =

- $\overline{D = V} =$

```
solve.add(D.eq(V));
```

- $\overline{V = D} =$

```
solver.add(D.eq(V));
```

- $\overline{D1 = D2} =$

```
solver.add(D1.eq(D2));
```

- $\overline{D = E} =$

```
solver.add((D).eq$\overline{E}$);
```

- $\overline{E = D} =$

```
solver.add((D).eq$\overline{E}$);
```

- $\overline{E = V} =$

$$\widetilde{\overline{E} = V}$$

- $\overline{E1 = E2} =$

$$\widetilde{\overline{E1} = \overline{E2}}$$

- $\widetilde{I = V} =$

```
solver.add((I).eq(V));
```

- $\widetilde{V = I} =$

```
solver.add((I).eq(V));
```

- $\widetilde{I1 = I2} =$

```
solver.add(I1.eq(I2));
```

- $\overline{V1 = V2} =$

```
IntLVar tVi = new IntLVar(V1);
solver.add(tVi.eq(V2));
```

3.6.2 !=

- $\overline{D! = V} =$

```
solve.add(D.neq(V));
```

- $\overline{V! = D} =$

```
solver.add(D.neq(V));
```

- $\overline{D1! = D2} =$

```
solver.add(D1.neq(D2));
```

- $\overline{D! = E} =$

```
solver.add(D.neq(E));
```

- $\overline{E! = D} =$

```
solver.add(D.neq(E));
```

- $\overline{E! = V} =$

$$\widetilde{\overline{E} = V}$$

- $\overline{E1! = E2} =$

$$\widetilde{\overline{E1} = \overline{E2}}$$

- $\widetilde{I! = V} =$

```
solver.add((I).neq(V));
```

- $\widetilde{V! = I} =$

```
solver.add((I).neq(V));
```

- $\widetilde{I1! = I2} =$

```
solver.add(I1.neq(I2));
```

- $\overline{V1! = V2} =$

```
IntLVar tVi = new IntLVar(V1);
solver.add(tVi.neq(V2));
```

3.6.3 <

- $\overline{D < V} =$

```
solve.add(D.lt(V));
```

- $\overline{V < D} =$

```
solver.add(D.lt(V));
```

- $\overline{D1 < D2} =$

```
solver.add(D1.lt(D2));
```

- $\overline{D < E} =$

```
solver.add(D.lt($\overline{E}$));
```

- $\overline{E < D} =$

```
solver.add((D).lt($\overline{E}$));
```

- $\overline{E < V} =$

$$\widetilde{\overline{E} = V}$$

- $\overline{E1 < E2} =$

$$\widetilde{\overline{E1} = \overline{E2}}$$

- $\widetilde{I < V} =$

```
solver.add((I).lt(V));
```

- $\widetilde{V < I} =$

```
solver.add((I).lt(V));
```

- $\widetilde{I1 < I2} =$

```
solver.add(I1.lt(I2));
```

- $\overline{V1 < V2} =$

```
IntLVar tVi = new IntLVar(V1);
solver.add(tVi.lt(V2));
```

3.6.4 $\leq$

- $\overline{D \leq V} =$

```
solve.add(D.le(V));
```

- $\overline{V \leq D} =$

```
solver.add(D.le(V));
```

- $\overline{D1 \leq D2} =$

```
solver.add(D1.le(D2));
```

- $\overline{D \leq E} =$

```
solver.add((D).le(E));
```

- $\overline{E \leq D} =$

```
solver.add((D).le(E));
```

- $\overline{E \leq V} =$

$$\widetilde{\overline{E} = V}$$

- $\overline{E1 \leq E2} =$

$$\widetilde{\overline{E1} = \overline{E2}}$$

- $\widetilde{I \leq V} =$

```
solver.add((I).le(V));
```

- $\widetilde{V \leq I} =$

```
solver.add((I).le(V));
```

- $\widetilde{I1 \leq I2} =$

```
solver.add(I1.le(I2));
```

- $\overline{V1 \leq V2} =$

```
IntLVar tVi = new IntLVar(V1);
solver.add(tVi.le(V2));
```

3.6.5 >

- $\overline{D > V} =$

```
solve.add(D.gt(V));
```

- $\overline{V > D} =$

```
solver.add(D.gt(V));
```

- $\overline{D1 > D2} =$

```
solver.add(D1.gt(D2));
```

- $\overline{D > E} =$

```
solver.add((D).gt(E));
```

- $\overline{E > D} =$

```
solver.add((D).gt(E));
```

- $\overline{E > V} =$

$$\widetilde{\overline{E} = V}$$

- $\overline{E1 > E2} =$

$$\widetilde{\overline{E1} = \overline{E2}}$$

- $\widetilde{I > V} =$

```
solver.add((I).gt(V));
```

- $\widetilde{V > I} =$

```
solver.add((I).gt(V));
```

- $\widetilde{I1 > I2} =$

```
solver.add(I1.gt(I2));
```

- $\overline{V1 > V2} =$

```
IntLVar tVi = new IntLVar(V1);
solver.add(tVi.gt(V2));
```

3.6.6 $\geq$

- $\overline{D \geq V} =$

```
solve.add(D.ge(V));
```

- $\overline{V \geq D} =$

```
solver.add(D.ge(V));
```

- $\overline{D1 \geq D2} =$

```
solver.add(D1.ge(D2));
```

- $\overline{D \geq E} =$

```
solver.add((D).ge(\overline{E});
```

- $\overline{E \geq D} =$

```
solver.add((D).ge(\overline{E});
```

- $\overline{E \geq V} =$

$$\widetilde{\overline{E} = V}$$

- $\overline{E1 \geq E2} =$

$$\widetilde{\overline{E1} = \overline{E2}}$$

- $\widetilde{I \geq V} =$

```
solver.add((I).ge(V));
```

- $\widetilde{V \geq I} =$

```
solver.add((I).ge(V));
```

- $\widetilde{I1 \geq I2} =$

```
solver.add(I1.ge(I2));
```

- $\overline{V1 \geq V2} =$

```
IntLVar tVi = new IntLVar(V1);
solver.add(tVi.ge(V2));
```

3.6.7 in

Come visto nel capito riguardante charme per assegnare un dominio ad una variabile D-Var esistono principalmente due metodi:

- Usando gli intervalli;
- Usando le liste di elementi;

Usando gli intervalli:
Sintassi:

```
D in min..max
```

dove: min e max sono due interi e $min \geq max$
Traduzione:

```
solver.add(D.dom(new MultiInterval(min,max)));
```

Usando le liste:
Sintassi:

```
D in [<N1>,<N2>,..,<Nn>]
```

dove:

- X è una IntLVar già inizializzata
- $\forall i \in [1, N]$. Ni è un intero

Inizialmente viene tradotta la lista in un MultiInterval (per la traduzione si rimanda al paragrafo riguardante i Domini) che indicheremo qui con $\overline{Dom}$ e successivamente applicheremo la seguente regola:
Traduzione:

```
solver.add(X.dom(Dom));
```

3.6.8 all_in

Sintassi:

```
A all_in Dom
```

Analiziamo la traduzione in base al tipo di array:

- Array mono-dimensionale

```
LArray.allIn(A, Dom);
```

- Array a due dimensioni

```
LMatrix2D.allIn(A, Dom);
```

- Array a tre dimensioni

```
LMatrix3D.allIn(A, Dom);
```

Ovviamente $\overline{Dom}$ deve essere già tradotto opportunamente in un MultiInteval.

3.6.9 all_diff

Sintassi:

```
all_diff A
```

Analizziamo la traduzione in base al tipo di array:

- Array mono-dimensionale

```
LArray.allDiff(A);
```

- Array a due dimensioni

```
LMatrix2D.allDiff(A);
```

- Array a tre dimensioni

```
LMatrix3D.allDiff(A);
```

3.6.10 if-possible

Si è scelto di creare una regola per tradurre il caso in cui if-possible viene utilizzato con due Constraint (insieme di vincoli) al posto di due blocchi (che possono contenere anche codice).
Sintassi:

```
if possible C1
then C2
```

Traduzione:

```
solver.add(meta.ifThen(C1, C2));
```

3.6.11 distribute

In Charme:

```
distribute(<number>,<Array>,<valuesArray>);
```

Traduzione:

```
solver.add(meta.distribute(<number>, <Array>, <valuesArray>)
    );
```

3.6.12 distribution

In Charme:

```
distribution(<number>,<Array>,<valuesArray>,<quotasArray>);
```

Traduzione:

```
solver.add(meta.distribution(<number>, <Array>, <valuesArray
    >, <quotasArray>));
```

```
        solve.add(D.eq(V));
```

3.6.13 Generate

Sintassi:

```
generate <oggettoCharme>;
```

Traduzione:
analiziamo la traduzione in base al tipo di *<oggettoCharme>*:

- D-Var

  ```
  solver.add(<oggettoCharme>.label());
  ```

- Array monodimensionale

  ```
  solver.add(LArray.labelArray(<oggettoCharme>));
  ```

- Array bidimensionale

  ```
  solver.add(LMatrix2D.labelMatrix(<oggettoCharme>));
  ```

- Array monodimensionale

  ```
  solver.add(LMatrix3D.labelMatrix(<oggettoCharme>));
  ```

3.6.14 find_all

Sintassi:

```
find_all B
```

dove:
B è un blocco Charme
Traduzione:
nella traduzione al posto di cercare una soluzione tramite

```
solver.solve();
```

si utilizza il seguente codice:

```
solver.solve();
do {
        B
} while (solver.nextSolution());
```

3.6.15 minimize

```
minimize D;
```

Traduzione:

```
solver.minimize(x)
```

3.6.16 Chiamata di una procedura

Sono presenti due casi:

- senza parametri
  Sintassi

  ```
  NP;
  ```

  Traduzione

  ```
  NP();
  ```

- con parametri
  Sintassi

  ```
  NP(P1,...,Pn);
  ```

Traduzione

```
NP(P1 , ... , Pn) ;
```

dove $\overline{Pi}$ sarà:

- Pi stesso se Pi è un valore numerico o un nome di variabile
- la traduzione di Pi se Pi è una lista

3.7 Procedure

Notazione utilizzata:

- NP indicherà un nome di procedura
- Pi, con $i \in N$, indicherà il parametro *i_ esimo*
- B indicherà un blocco;
- $\overline{B}$ indicherà la traduzione di un blocco
- $\overline{Pi}$ indicherà la traduzione del parametro *i_ esimo*

Come abbiamo già visto la definizione di una procedura in Charme ha come sintassi:

```
define NP [P1,...,Pn]  B
```

La traduzione cambierà a seconda della presenza o meno di parametri formali:

- senza parametri formali

```
void NP ()
{

        B

}
```

- con parametri formali
  In questa situazione avremo come sintassi in charme

```
define NP (P1,P2,...,Pn)  B
```

La traduzione sarà:

```
void NP (P1,P2,...,Pn)
{

        B

}
```

dove $\overline{Pi}$ sarà :

- se Pi è un D-Var

```
IntLVar pi
```

- se Pi è una variabile semplice

```
Integer pi
```

3.8 Eventi

3.8.1 when-known

Notazione utilizzata:

- Var per indicare una variabile
- B per indicare un plocco
- $\overline{Var}$ per indicare la traduzione della variabile

Sintassi:

```
when_known [Var1, ..., Varn] do B
```

Traduzione:

```
solver.add(meta.when_known(new IntLVar[]{Var1,...,
    Varn}, new WhenKnownBody() {
@Override public void when_true() {
B
}
@Override
public void when_false() {

}
@Override
public boolean control() {
return true;
}
}));
```

Caso completo: In Charme

```
when_known [Var1, ..., Varn] do
if B1
then B2
else B3
```

Traduzione:

```
solver.add(meta.when_known(new IntLVar[]{Var1,...,
    Varn}, new WhenKnownBody() {
@Override public void when_true() {
B2
}
@Override
public void when_false() {
B2
}
@Override
public boolean control() {
// in questo caso B3 viene tradotto in una funzione
    che restituisce true se tutti i vincoli vengono
     rispettati
return B3;
}
}));
```

Capitolo 4

Estensione del framework JSetL

In questo capitolo sono presentate e spiegate le funzionalità aggiunte a JSetL necessarie per poter emulare alcuni meccanismi presenti nel linguaggio Charme, tutte le librerie, visto il focus rivolto sui domini interi, sono rivolte al tipo di dato IntLVar.

4.1 Operazioni su array

Come già visto nella sezione del capito 1 dedicata, in Charme sono presenti operazioni avanzate su array che in Java e JSetL non sono presenti.
Quindi il framework è stato arricchito con delle classi dedicate alla manipolazione di array di *IntLVar* a 1, 2 e 3 dimensioni.

4.1.1 La classe LArray

Questa classe manipola, emulando il comportamento di Charme, gli array monodimensionali.

- ```
  public static IntLVar[] generateArray(Object arr)
  ```

  dove:
  arr può essere:

  - MultiInterval
  - Istanziazione di un Enum
  - Array
  - Un intero

  Questo metodo restituisce un array monodimensionale di oggetti IntLVar a cui non è assegnato un dominio quindi per definizione il
  ```

loro dominio è $[minint, maxint]$.
Nel caso di MultiInterval, Enum, Array l'array restituito avrà tanti elementi quanti ne ha il paramentro, mentre per quanto riguarda l'intero esso stesso rappresenta la dimensione, ovviamente dovrà essere un numero positivo strettamente maggiore di 0.

- ```
  public static IntLVar[] generateArray(Object arr,
      MultiInterval domain)
  ```

  dove:

  - *arr* è già stato visto prima
  - *domain* è un MultiInterval che contiene tutti i valori che rapresentano il dominio di ogni oggetto appartenente all'array.

  Questo metodo restituisce un array monodimensionale, ogni elemento dell'array ha come dominio *domain*.

- ```
  public static IntLVar[] generateArray(Object arr,
      MultiInterval domain, SolverClass sol)
  ```

  dove:

  - *arr* e *domain* sono già stati analizzati precedentemente.
  - *sol* è l'oggetto SolverClass utilizzato per la risoluzione del problema.

  Questo metodo restituisce un array monodimensionale, il dominio però non viene assegnato direttamente alle variabili ma viene inserito un vincolo all'interno di *sol*.

- ```
  public static Vector<IntLVar> arrayToVector(IntLVar[]
      Vect)
  ```

  Il metodo restituisce un oggetto della classe *Vector* contenente una copia di ogni variabile dell'array *Vect*.

- ```
  public static Constraint allIn(IntLVar[] array,
      MultiInterval m)
  ```

  dove:

  - *array* è un array di variabili IntLVar

– m è un MultiInterval rappresentante un dominio

Questo metodo restituisce un Constraint rappresentante il vincolo:

$$\forall a \in array, a \in m$$

- ```
  public static Constraint allDiff(IntLVar[] array)
  ```

  Il metodo restituisce un Constraint rappresentante il vincolo:

  $$\forall i, j \in [0, array.length() - 1], i \neq j \implies array[i] \neq array[j]$$

- ```
  public static Constraint labelArray(IntLVar[] array)
  ```

  Il metodo restituisce un Constraint che, quando viene analizzato, attiva il meccanismo di labeling su ogni variabile di *array*

- ```
  public static Constraint domArray(IntLVar[] a,
      MultiInterval r)
  ```

  È un metodo equivalente a *allIn*

- ```
  public static Constraint sumArray(IntLVar[] a, IntLVar
      v)
  ```

  Il metodo restituisce un Constraint contenente il vincolo:

  $$v = \sum_{i=0}^{array.length()-1} a_i$$

  dove:

  $$\forall i \in [0, array.length() - 1], a_i \in array$$

4.1.2 La classe LMatrix2D

Questa classe manipola, emulando il comportamento di Charme, gli array a due dimensione. Metodi della classe:

- ```
  public static IntLVar[][] generateArray(Object arr,
      Object arr2)
  ```

  dove:

  - *arr* è un oggetto rappresentante il numero di righe della matrice *.
  - *arr2* è un oggetto rappresentante il numero di colonne della matrice

  Il metodo restituisce una matrice a 2 dimensioni di oggetti IntLVar ognuno dei quali ha come dominio $[minint, maxint]$

- ```
  public static IntLVar[][] generateArray(Object arr,
      Object arr2, MultiInterval domain)
  ```

  arr e *arr2* sono già stati analizzati precedentemente, per quanto riguarda il metodo esso restituisce una matrice bidimensione in cui ad ogni oggetto è stato assegnato il dominio *domain*.

- ```
  public static IntLVar[][] generateArray(Object arr,
      Object arr2, MultiInterval domain, SolverClass sol)
  ```

  arr e *arr2* sono già stati analizzati precedentemente, per quanto riguarda il metodo esso restituisce una matrice bidimensione, e a sol sono stati aggiunti i constraint che forzano il dominio degli oggetti ad essere uguale a *domain*.

- ```
  public static IntLVar[] extractArray(IntLVar[][] Mat,
      int x, int y)
  ```

  dove:

  - *Mat* è una matrice n x m
  - $x \in [-1, n] \wedge y \in [-1, m]$

*Per i possibili valori assegnabili e l'analisi di questi ultimi rimandiamo alla classe *LArray*

- $x = -1 \iff y \neq -1 \land y = -1 \iff x \neq -1$

Il metodo restituisce un array monodimensionale contenente:

- se $x = -1 \land y \neq -1$
  l'array contiene tutti gli elementi della colonna y
  esempio:

```
IntLVar[][] matrix = LMatrix2D.generateArray(4, 4);
Integer count = 0;
for(Integer i = 0; i < 4; ++i)
        for(Integer e = 0; e < 4; ++e)
                matrix[i][e].setValue(++count);
/* matrix:
        1  | 2  | 3  | 4  |
        5  | 6  | 7  | 8  |
        9  | 10 | 11 | 12 |
        13 | 14 | 15 | 16 |
*/

IntLVar[] array = LMatrix2D.extractArray(matrix,
    -1, 2);
/* array:
        3 | 7 | 11 | 15 |
*/
```

- se $y = -1 \land x \neq -1$
  l'array contiene tutti gli elementi della riga x
  esempio:

```
IntLVar[][] matrix = LMatrix2D.generateArray(4, 4);
Integer count = 0;
for(Integer i = 0; i < 4; ++i)
        for(Integer e = 0; e < 4; ++e)
                matrix[i][e].setValue(++count);
/* matrix:
        1  | 2  | 3  | 4  |
        5  | 6  | 7  | 8  |
        9  | 10 | 11 | 12 |
        13 | 14 | 15 | 16 |
*/

IntLVar[] array = LMatrix2D.extractArray(matrix,
    2,-1);
/* array:
        9 | 10 | 11 | 12 |
*/
```

- ```
  public static IntLVar[] matrixToArray(IntLVar[][]
      matrix)
  ```

  Il metodo restituisce un array monodimensionale contenente tutti gli elementi della matrice *matrix*.

- ```
  public static Constraint allIn(IntLVar[][] matrix,
      MultiInterval domain)
  ```

  Il metodo è equivalente a quello presente nella classe *LArray* quindi per la spiegazione rimando al capitolo precedente.

- ```
  public static Constraint allDiff(IntLVar[][] matrix)
  ```

  matrix matrice n x m
  Il metodo restituisce un Constraint rappresentante il vincolo:

  $$\forall x_1, x_2 \in [0, n]$$

  $$\forall y_1, y_2 \in [0, m]$$

  $$x_1 \neq x_2 \vee y_1 \neq y_2 \implies matrix[x_1][y_1] \neq matrix[x_2][y_1]$$

- ```
  public static Constraint labelMatrix(IntLVar[][] matrix
      )
  ```

  Il metodo è equivalente a quello presente nella classe *LArray* quindi per la spiegazione rimando al capitolo precedente.

4.1.3 La classe LMatrix3D

Questa classe manipola, emulando il comportamento di Charme, gli array a tre dimensione.
Di seguito riporteremo soltanto l'elenco dei metodi in quanto sono equivalenti a quelli presentati nei due capitoli precedenti.

- ```
  public static IntLVar[][][] generateArray(
      Object arr, Object arr2, Object arr3)
  ```
  ```

- ```
  public static IntLVar[][][][] generateArray(Object
      arr, Object arr2, Object arr3, MultiInterval
      domain)
  ```

- ```
  public static IntLVar[][][][] generateArray(Object
      arr, Object arr2, Object arr3, MultiInterval
      domain, SolverClass sol) throws Exception
  ```

- ```
  public static IntLVar[][] extractArray2D(
      IntLVar[][][][] Mat, int x, int y, int z)
  ```

dove:

- *Mat* è una matrice del tipo n x m x l
- $x \in [-1, n] \wedge y \in [-1, m] \wedge z \in [-1, l]$
- $x = -1 \wedge y = -1 \iff z \neq -1$
- $x = -1 \wedge z = -1 \iff y \neq -1$
- $y = -1 \wedge z = -1 \iff x \neq -1$

Si definiscono due dimensioni in cui prelevare tutte le variabili (assegnandogli il valore -1) e invece si fissa un valore per l'ultima dimensione; il risultato di questa selezione è una matrice bidimensionale.

Esempio:

```
IntLVar[][][][] matrix = LMatrix3D.generateArray(4, 4,2);
Integer count = 0;

for(Integer f = 0; f < 2; ++f)
        for(Integer i = 0; i < 4; ++i)
                for(Integer e = 0; e < 4; ++e)
                        matrix[i][e][f].setValue(++
                            count);

/* matrix[all][all][0]
        1  | 2  | 3  | 4  |
        5  | 6  | 7  | 8  |
        9  | 10 | 11 | 12 |
        13 | 14 | 15 | 16 |

matrix[all][all][1]
        17 | 18 | 19 | 20 |
        21 | 22 | 23 | 24 |
```
```

```
        25 | 26 | 27 | 28 |
        29 | 30 | 31 | 32 |
*/

IntLVar[][] array = LMatrix3D.extractArray2D(matrix,
    -1, -1, 0);

/* array:
        1  | 2  | 3  | 4  |
        5  | 6  | 7  | 8  |
        9  | 10 | 11 | 12 |
        13 | 14 | 15 | 16 |
*/
```

- ```
  public static IntLVar[] extractArray(IntLVar
      [][][][] Mat, int x, int y, int z)
  ```

  dove:

  - *Mat* è una matrice del tipo n x m x l
  - $x \in [-1, n] \wedge y \in [-1, m] \wedge z \in [-1, l]$
  - $x = -1 \iff y \neq -1 \ \wedge z \neq -1$
  - $y = -1 \iff x \neq -1 \ \wedge z \neq -1$
  - $z = -1 \iff x \neq -1 \ \wedge y \neq -1$

  Si scelgono una dimensione per la quale prelevare tutti i valori, ciò viene fatto passando come parametro attuale il valore -1, mentre per le altre due dimensioni si fissa un valore; l'array risultante viene poi restituito dalla funzione.

  Esempio:

```
IntLVar[][][][] matrix = LMatrix3D.generateArray(4, 4,2);
Integer count = 0;

for(Integer f = 0; f < 2; ++f)
        for(Integer i = 0; i < 4; ++i)
                for(Integer e = 0; e < 4; ++e)
                        matrix[i][e][f].setValue(++
                            count);

/* matrix[all][all][0]
        1  | 2  | 3  | 4  |
        5  | 6  | 7  | 8  |
        9  | 10 | 11 | 12 |
        13 | 14 | 15 | 16 |
```
```

```
matrix[all][all][1]
        17 | 18 | 19 | 20 |
        21 | 22 | 23 | 24 |
        25 | 26 | 27 | 28 |
        29 | 30 | 31 | 32 |
*/

IntLVar[][] array = LMatrix3D.extractArray2D(matrix,
    -1, -1, 0);

/* array:
        19 | 23 | 27 | 31 |
*/
```

- ```
  public static IntLVar[] matrixToArray(IntLVar
      [][][][] matrix)
  ```

  Il metodo restituisce un array contenente tutte le variabili presenti nella matrice *Mat*.

- ```
  static Vector<IntLVar> matrixToVector(IntLVar
      [][][][] Mat)
  ```

  Il metodo restituisce un oggetto *Vector* contenente tutte le variabili della matrice *Mat*.

- ```
  static Constraint sumMatrix(IntLVar[][][][] Mat,
      IntLVar sum, SolverClass solver)
  ```

  Il metodo restituisce un Constraint rappresentante il vincolo : assumendo *Mat* di dimensioni $r * c * d$.

$$sum = \sum_{i=0}^{r} \sum_{e=0}^{c} \sum_{j=0}^{d} a_{iej}$$

- ```
  public static Constraint labelMatrix(IntLVar
      [][][][] matrix)
  ```

4.2 I nuovi constraint

4.2.1 La classe LArrayOps

Nella classe *LArrayOps* sono definiti tre constraint per la gestione degli array :

```
1 public Constraint ithElem(IntLVar[] a, IntLVar i,
      IntLVar x)
```

Il constraint risultante vincola x ad essere uguale alla i_esima variabile dell'array a dove la variabile i è a sua volta una variabile logica, avendo la variabile non un singolo valore ma in generale un dominio bisogna aspettare che quest'ultimo venga ridotto ad un singolo valore per poter effettuare la selezione.
Esempio:

```
SolverClass solver = new SolverClass();
LArrayOps arrayOps = new LArrayOps(solver);

IntLVar[] array = LArray.generateArray(4,new
     MultiInterval(1,10));

IntLVar a = new IntLVar(1);
IntLVar b = new IntLVar(2);
IntLVar i = new IntLVar();

//i = a + b
solver.add(i.eq(a.sum(b)));

/*per poter forzare la i_esima variabile di array ad
     avere il valore  5
dobbiamo utilizzare ithElem*/

IntLVar x = new IntLVar();
solver.add(arrayOps.ithElem(array, i, x));

solver.add(x.eq(5));

solver.solve();

for(Integer e = 0; e < 4; ++e)
         System.out.println("array["+e+"]="+array[e].
              getValue());
/*output:

array[0]=null
array[1]=null
array[2]=null
```

```
array[3]=5
\*/
\\i valori null sono dovuti al dominio non bound della
    variabile
```

```
public Constraint occurrence(IntLVar[] a, IntLVar v,
    IntLVar n)
```

Il metodo restituisce un constraint che risulterà soddisfatto se e solo se a conterrà esattamente n elementi uguale a v.

3

```
public Constraint scalarProdArray(IntLVar[] a, IntLVar
    [] b, IntLVar v)
```

Il metodo restituisce un constraint che rappresenta il vincolo:

$$v = \sum_{j=0}^{a.length()} a[i] * b[i]$$

•

```
public Constraint distribute(Object number, IntLVar[]
    array, Integer[] valuesArray)
```

Il metodo *distribute* genera un Constraint con il seguente significato:

$$\exists F \subseteq array. |F| = number \wedge (\forall f \in F : f \in valuesArray)$$

Esempio:

```
SolverClass solver = new SolverClass();
LArrayOps arrayOps = new LArrayOps(solver);
IntLVar[] array = LArray.generateArray(3,new
    MultiInterval(1,8));

solver.add(arrayOps.distribute(2, array, new Integer
    []{1,5,3}));
solver.add(LArray.labelArray(array));
solver.solve();

for(int i = 0 ; i < 3; ++i)
        System.out.println("array["+i+"]="+array[i].
            getValue());

/*output:

array[0]=1
array[1]=1
```

```
array[2]=2
*/
/*due delle variabili dell'array hanno un valore
    presente nel valuesArray*/
/*per effetto del labeling i valori delle due variabili
    vincolate sono uguali al primo disponibile, vale a
    dire 1, la terza prende il primo valore non
    presente nell'array dei valori quindi 2.
*/
```

- ```
  public Constraint distribution(Object number, IntLVar[]
      array, Integer[] valuesArray, Integer[]
      quotasArray)
  ```

Definita la funzione:
$count(I, v_i)$ = numero di volte che compare l'elemento v_i nell'insieme I

Il metodo *distribution* genera un Constraint con il seguente significato:

$$\exists F \subseteq array. |F| = number \wedge (\forall f \in F : f \in valuesArray) \wedge (\forall i \in [0, |V|-1] : \wedge$$

$$\wedge count(A, valuesArray[i]) = quotasArray[i])$$

Esempio:

```
SolverClass solver = new SolverClass();
LArrayOps arrayOps = new LArrayOps(solver);
IntLVar[] array = LArray.generateArray(6,new
    MultiInterval(1,5),solver);

solver.add(arrayOps.distribution(4, array, new Integer
    []{1,2,3},new Integer[]{2,1,1} ));
solver.add(LArray.labelArray(array));
solver.solve();

for(int i = 0 ; i < 6; ++i)
        System.out.println("array["+i+"]="+array[i].
            getValue());
/*output:

array[0]=1
array[1]=1
array[2]=2
array[3]=3
array[4]=4
array[5]=4
```
```

```
*/
il valore 1 compare esattamente 2 volte, i valori 2 e 3
     invece esattamente 1 volta
*/
```

4.2.2 La classe MetaOps

Nella classe *MetaOps* sono presenti metodi necessari per poter vincolare l'analisi o l'esecuzione di una sezione di codice, o di un constraint, a determinati eventi.

- ```
  public Constraint when_known(IntLVar[] variables,
      WhenKnownBody f)
  ```

  dove *WhenKnownBody* è l'interfaccia:

  ```
  public interface WhenKnownBody {

          public boolean control();
          public void when_true();
          public void when_false();
  }
  ```

  Il metodo *when_known* genera un Constraint che ha il seguente comportamento:
  l'analisi del constraint viene rimandata finchè tutte le variabili presenti nell'array *variables* non sono bound [†]; quando ciò si verifica viene eseguita la funzione $f.control()$ ora abbiamo due situazioni:

  - $f.control() = true$
    In questo caso viene eseguita la funzione $f.when_t true()$
  - $f.control() = false$
    In questo caso viene eseguita la funzione $f.when_f alse()$

  Esempio di utilizzo del constraint:

  ```
  IntLVar a = new IntLVar(new MultiInterval(1,10));
  SolverClass solver =new SolverClass();
  static MetaOps meta = new MetaOps(solver);
  solver.add(meta.when_known(new IntLVar[]{a}, new
      WhenKnownBody() {
          @Override public void when_true() {
          System.out.println("la varibile a ha valore
               pari");
  ```

[†]Una variabile si dice *bound* quando il suo dominio è un singoletto
  ```

```
        }
        @Override
        public void when_false() {
                System.out.println("la varibile a ha
                    valore dispari");
        }
        @Override
        public boolean control() {
        return (a.getValue() % 2) == 0;
        }
        }));
```

```
public Constraint ifThen(Constraint c1, Constraint c2)
```

Il metodo *ifThen* genera un Constraint con il seguente significato:
se l'insieme di vincoli *c1* è risolvibile allora aggiungi il Constraint *c2* alla soluzione del nostro problema.

- ```
  public Constraint delay(LVar x, BodyDelay b)
  ```

per la spiegazione del metodo delay rimando ai capito successivi.

4.3 La classe GenericFunction

4.3.1 Gestione domini

È necessario implementare tutti i meccanismi volti a gestire correttamente gli oggetti utilizzati per rappresentare un dominio:

- ```
  public static int getDimension(Object d) throws
      Exception
          {
                  if(d instanceof MultiInterval)
                  {
                          return ((MultiInterval)d).size
                              ();
                  }
                  else if( ((Class<?>)d.getClass()).
                      getEnumConstants() != null    )
                  {
                          return ((Class<?>)d.getClass())
                              .getEnumConstants().length;
                  }
                  else if(d.getClass().isArray()    )
                  {
  ```

```
                return  Array.getLength(d);
            }
            else if(d instanceof Integer)
                return (Integer)d;
            else
                throw new Exception("
                    unsupported element");
        }
```

d è un oggetto che ha uno dei seguenti tipi:

- MultiInterval
- Enum
- Array
- Integer

il metodo restituisce il numero di elementi che l'oggetto contiene, nel caso di *Integer* restituisce l'oggetto stesso.

- ```
public static MultiInterval getDomainFromEnum(Object e)
    throws Exception
      {
            if(((Class<?>)e.getClass()).
                getEnumConstants() == null)
                throw new Exception("
                    unsupported element");
            MultiInterval temp = new MultiInterval
                ();
            Class<?> c = e.getClass();
            Method[] allMethods = c.
                getDeclaredMethods();
            Method m=null;
            for(Method mt : allMethods)
            {
                String mname = mt.getName();
                if(mname.equals("getOrdinal"))
                {
                    m = mt;
                    break;
                }
            }
            if(m != null)
                for(Object o : e.getClass().
                    getEnumConstants())
                {
                    temp.add((Integer)m.
                        invoke(o, new
                        Object[]{}));
```
```

```
                }
            if(temp.size()==0)
                throw new Exception("
                    impossibile convertire l'
                    oggetto in un
                    multiintervallo");
            return temp;
        }
```

e è un oggetto del tipo Enum con le caratteristiche specificane nel capito 3.3.4. Il metodo restituisce un *MultiInterval* contenente tutti i valori degli elementi contenuti nell'enum ottenuti tramite l'invocazione del metodo *getOrdinal()*.

- ```
public static MultiInterval getDomain(Object dom)
    throws Exception
        {
            if(dom instanceof MultiInterval)
            {
                return (MultiInterval)dom;
            }
            else if(((Class<?>)dom.getClass()).
                getEnumConstants() != null   )
            {
                return getDomainFromEnum(dom);
            }
            else if(dom instanceof Integer[])
            {
                MultiInterval m = new
                    MultiInterval();
                for(Integer i : (Integer[])dom)
                    m.add(i);
                return m;
            }
            else
                throw new Exception("
                    unsupported element");
        }
```

*dom* è un oggetto che ha come tipo uno dei seguenti:

- MultiInterval
- Enum
- Array di Integer

Il metodo restituisce un MultiIntervallo rappresentante tutti i valori contenuti nell' oggetto passato.
```

Per la creazione di questi metodi vengono utilizzate le classi e i metodi forniti dal Package :

```
java.lang.reflect
```

Queste ultime infatti sono un insieme di classi che permettono l'interrogazione dinamica di una istanza di classe per avere informazioni riguardo i suoi metodi e su suio oggetti.

4.3.2 Metodi per il rollback

I seguenti due metodi servono per poter effettuare manualmente rollback, meccanismo indispensabile per poter verificare dei vincoli senza modificare lo stato interno delle variabili.

- ```
  public static Vector<IntLVar> saveAll(Vector<IntLVar>
      lVars)
  ```

  Permette di ottenere una copia dello stato di tutte le variabile appartenenti al vettore *lVars*

- ```
  public static void restoreAll(Vector<IntLVar> lVars,
      Vector<IntLVar> savedLVars)
  ```

  Permette di ripristinare lo stato delle variabili

4.3.3 Produttoria e sommatoria

I seguenti metodi invece sono delle utilities per facilitare la gestione di produttorie e sommatorie:

- ```
  public static IntLVar getSum(IntLVar[] array,
      SolverClass solver)
  ```

  Restituisce una IntLVar rappresentante la sommatoria di tutte le variabili appartenenti a *array*

- ```
  public static IntLVar getMul(IntLVar[] array1,IntLVar[]
       array2,SolverClass solver)
  ```

  Restituisce una IntLVar rappresentate il prodotto scalare tra le variabili di *array1* e *array2*

4.4 Generalizzazioni

delay

Come si può notare i costrutti *when_know* e *ithElem* sono esempi di come è necessario a volte rimandare l'esecuzione di un vincolo finchè certe precondizioni non sono rispettate, nel caso di *when_know* di rimandare il codice e l'eventuale controllo finchè un array di variabili logiche non risultano *bound* mentre nel caso di *ithElem* finchè un indice necessario per la selezione di un elemento di un array non risulta *bound*.
Si è quindi provveduto a creare un generico constraint che ha lo scopo di rimandare l'esecuzione di una porzione di codice finché un vincolo non risulta vero.

Sintassi:

```
public Constraint delay(LVar x, BodyDelay b)
```

dove:

- x
  è una variabile IntLVar sulla quale eseguire il controllo e l'azione definite in b

Capitolo 5

Esempio di programma tradotto

5.1 Send-More-Money

5.1.1 Charme

```
/*********************************************************
    */
/* CRYPTO ARITHMETIC PROBLEM
    */
/*
    */
/*              S E N D        Each letter stands
    */
/*            + M O R E        for a different digit
    */
/*            ---------
    */
/*            M O N E Y
    */
/*
    */
/*********************************************************
    */

Letters = [S,E,N,D,M,O,R,Y];

/* CONSTRAINTS */

Letters all_in 0..9;

R0              = M             ;
R1 +     S + M = O     + 10*R0;
```

```
R2 +     E + O = N     + 10*R1;
R3 +     N + R = E     + 10*R2;
         D + E = Y     + 10*R3;

S != 0;              /* no 0 on the left
    */
M != 0;
all_diff Letters; /*each letter is different from the others
    */

/* COMMENT */

printf("\n  SEND + MORE = MONEY ??\n\n");

/* GENERATION */

generate Letters ;

/* DISPLAY SOLUTION */

printf("  S E N D       %V %V %V %V\n", S,E,N,D);
printf("+ M O R E     + %V %V %V %V\n", M,O,R,E);
printf("-------     -------\n");
printf("M O N E Y     %V %V %V %V %V\n\n", M,O,N,E,Y);

/* AND EXIT */

quit;
```

5.1.2 Java + JSetL

```
package applications.charme;

import JSetL.*;
import JSetL.lib.LArray;

public class ExamplesBortolotti_SendMoreMoney {


        public static void main(String[] argv) throws
            NotValidDomainException, Exception
        {
                SolverClass solver = new SolverClass();

                //Letters = [S,E,N,D,M,O,R,Y];
                IntLVar S = new IntLVar("S");
                IntLVar E = new IntLVar("E");
                IntLVar N = new IntLVar("N");
```

```
IntLVar D = new IntLVar("D");
IntLVar M = new IntLVar("M");
IntLVar O = new IntLVar("O");
IntLVar R = new IntLVar("R");
IntLVar Y = new IntLVar("Y");

IntLVar Letters[] = {S,E,N,D,M,O,R,Y};

//Letters all_in 0..9;
for(IntLVar v : Letters)
{
        solver.add(v.in(new MultiInterval(0,
            9)));
}

/* CONSTRAINTS */

//R0              = M             ;
IntLVar R0 = M;
IntLVar R1 = new IntLVar();

IntLVar R2 = new IntLVar();
IntLVar R3 = new IntLVar();

//R1 +     S + M = O     + 10*R0;
solver.add(R1.sum(S.sum(M)).eq(O.sum(R0.mul
    (10))));

//R2 +     E + O = N     + 10*R1;
solver.add(R2.sum(E.sum(O)).eq(N.sum(R1.mul
    (10))));

//R3 +     N + R = E     + 10*R2;
solver.add(R3.sum(N.sum(R)).eq(E.sum(R2.mul
    (10))));

//D + E = Y     + 10*R3;
solver.add(D.sum(E).eq(Y.sum(R3.mul(10))));

//S != 0;             /* no 0 on the left
                        */
solver.add(S.neq(0));

//M != 0;
solver.add(M.neq(0));

//all_diff Letters; /*each letter is
    different from the others*/
solver.add(LArray.allDiff(Letters));
```

```
                /* COMMENT */
                System.out.println("\n  SEND + MORE = MONEY
                    ??\n\n");

                /* GENERATION */

                solver.add(LArray.labelArray(Letters));


                solver.solve();

                /* DISPLAY SOLUTION */
                System.out.println("  S E N D       "+S+" "+
                    E+" "+N+" "+D+"\n");
                System.out.println("+ M O R E     + "+M+" "+
                    O+" "+R+" "+E+"\n");
                System.out.println("-------     -------\
                    n");
                System.out.println("M O N E Y     "+M+" "+O+
                    " "+N+" "+E+" "+Y+"\n\n");

        }

}
```

Capitolo 6

Conclusioni

6.1 Riflessioni conclusive

Durante il lavoro di tesi è stato effettuato uno studio di fattibilità riguardante la possibilità di creare delle regole, che si potessero definire standard, per poter tradurre direttamente il codice Charme in codice Java con l'ausilio di JSetL.
Dallo studio si è visto che, per quanto riguada i domini numerici, ciò è veramente possibile.
Per verificare quanto appena detto abbiamo eseguito la traduzione di tutti gli esempi presenti nel manuale Charme [8] ed abbiamo verificato che il comportamento del codice è analogo a quello dichiarato all'interno del manuale.
Il risultato finale non è stato soltanto la creazione delle regole necessarie per poter effettuare una corretta traduzione dal codice Charme a Java + JSetL ma molto importante è stato anche l'estensione della libreria JSetL che ha portato ad un aumento della sua versatilità.

6.2 Sviluppi futuri

Sicuramente sarà necessario continuare il lavoro da noi intrapreso per poter terminare l'analisi di tutte le funzionalità di Charme che non siamo riusciti a trattare in questo studio, prima tra tutte l'analisi dei domini simbolici.
Una volta terminata l'analisi e la traduzione il passo successivo potrebbe essere quello di creare un pre-compilatore per permettere l'inserimento di porzioni di codice Charme direttamente nei file sorgente Java + JSetL così da avvicinare sempre più la sintassi del codice Java + JSetL a quella di Charme.
Un ulteriore miglioramento sarebbe la creazione di un compilatore che traduca automaticamente il file sorgente Charme in codice Java + JSetL sfruttando le regole definite in questa tesi; ma siamo scettici su questo

punto in quanto, per far ciò, sarebbe necessaria, quanto meno, la presenza di tutta la specifica della sintassi Charme che a tutt'oggi non siamo stati in grado di reperire.

Bibliografia

[1] Gianfranco Rossi, Elio Panegai, Elisabetta Poleo
JSetL: a Java library for supporting declarative programming in Java
Software Practice & Experience 2007; 37:115-149.

[2] F. Bergenti, L. Chiarabini, G. Rossi
Programming with Partially Specified Aggregates in Java
Computer Languages, Systems & Structures, Elsevier, vol. 37/4, 178-192, 2011.

[3] Gianfranco Rossi e Roberto Amadini
JSetL User's Manual
`http://cmt.math.unipr.it/jsetl/JSetLUserManual-v.2.3.pdf`

[4] JSetL Home Page
`http://cmt.math.unipr.it/jsetl.html`

[5] Bruce Eckel
Thinking in Java 4th Edition Vol. 1-3
Prentice-Hall, 2006.

[6] Pellegrino Principe
Java 7 Guida Completa
Apogeo, 1995.

[7] Oplobedu, A., Marcovich, J., and Tourbier, Y. 1989
Charme: un langage industriel de programmation par contraintes, illustré par une application chez renault.
In Ninth Internetaional Workshop on Expert Systems and their Applications: General Conference, volume 1, pages 55-70, Avignon, May 1989.

[8] Arificial Intelligence Development
Charme Reference Manual
Bull, 1990.

[9] K.R. Apt and A. Schaerf
The Alma Project, or How First-Order Logic Can Help Us in Imperative

Programming.
E.-R. Olderog, B. Steffen, Eds.: Correct System Design. Springer-Verlag Lecture Notes in Computer Science 1710, pp. 89-113.

Appendices

Esempi completi di traduzione

.1 Ages

.1.1 Charme

```
Text = [
/********************************************************
      */
" ARRAY OF AGES                                          ",
" Rene and Leo have 3 daughters and 3 sons , all under 10    ",
" years old . Rene and Leo don ' t have twins .                 ",
" − Leo youngest child is a girl .                              ",
" − Rene ' s youngest daughter has just been born .             ",
" − The sum of ages of all the children is 60.                ",
" − In each family the sum of ages of the girls is equal to ",
"   the sum of ages of the boys .                              ",
" − In each family the sum of the square of ages of the      ",
"   girls is equal to the sum of the square of ages of the  ",
"   boys .                                                      ",
" What are the ages of all these children ?                  "
      ];
/********************************************************
      */

/* DATA */

Last_child = 3;
```

```
Children = 1..Last_child;
Fathers = [rene, leo];
Daughter_Son = [girl,boy];
Ages = 0..9;
array Family :: [Children, Daughter_Son, Fathers] of Ages;

/* CONSTRAINTS */

define constraints
{ for A_father in Fathers do
  { /* no twins
       */
    all_diff( Family[_,_,A_father]);

    /* Sum of ages of girls = Sum of ages of boys
         */
    sum(Family[_,girl,A_father]) =
      sum(Family[_,boy,A_father]);

    /* Sums of squares of ages are equal for girls and boys
         */
     scal_prod(Family[_,girl,A_father],Family[_,girl,
          A_father]) =
     scal_prod(Family[_,boy,A_father],Family[_,boy,A_father
          ]);
  }

 /* children are given by decreasing age
      */
 for A_father in Fathers do
   for A_kid in Daughter_Son do
     for I in 2..Last_child do
       Family[I,A_kid,A_father] < Family[I-1,A_kid,A_father
            ];

 /* the last leo's child is a girl
      */
 Family[Last_child,girl,leo] < Family[Last_child,boy,leo];

 /* the last rene's girl have just been to born
      */
 Family[Last_child,girl,rene] = 0;

 /* sum of ages is 60
      */
 sum(Family) = 60;
}

/* DISPLAY SOLUTION */
```

```
define pr_sol
{ for A_father in Fathers do
  { printf("\nChildren of %V: \n", A_father);
    for A_kid in Daughter_Son do
    { printf("%V \t: ",A_kid);
      for A_child in Children do
        printf("%V ", Family[A_child,A_kid,A_father]);
      printf("\n");
    }
  }
}

/* MAIN PROGRAM */

define main
{ for C in Text do printf("%s\n",C);
  constraints;

  /* generation
       */
  /* criteria [-1,-5] : choose the D_var with the least
       */
  /*          domain and then with the minimum of constraints
       */
  /* Order -2 : enumerate values in a dichotomous way
       */
  generation(Family,[-1, -5], -2);

  pr_sol;
  quit;
}

/* RUNNING */

main;
```

.1.2 Java + JSetL

```
package applications.charme;

import java.util.Iterator;

import JSetL.IntLVar;
import JSetL.MultiInterval;
import JSetL.SolverClass;
import JSetL.lib.LArray;
import JSetL.lib.LArrayOps;
```

```
import JSetL.lib.GenericFunctions;
import JSetL.lib.LMatrix2D;
import JSetL.lib.LMatrix3D;

public class ExamplesBortolotti_Ages {

        static SolverClass s = new SolverClass();
//      static GlobalConstraints globalConstraints = new
    GlobalConstraints(s);
        static LArrayOps arrayOps;

        static final int Last_child = 2;
        static final int Children = 3;
        static final MultiInterval Ages = new MultiInterval
            (0,9);
        static IntLVar[][][][] Family;

        enum Fathers{
                rene(0),
                leo(1);

                int pos;
                   Fathers(int p) {
                      pos = p;
                   }
                   int getOrdinal() {
                      return pos;
                   }
        }
        enum Daughter_Son{
                girl(0),
                boy(1);

                int pos;
                Daughter_Son(int p) {
                      pos = p;
                   }
                   int getOrdinal() {
                      return pos;
                   }
        }

        static void constraints() throws Exception {
                arrayOps = new LArrayOps(s);
                Fathers f = Fathers.leo;
                Daughter_Son d = Daughter_Son.boy;

                Family =LMatrix3D.generateArray(Children, d,
                      f, Ages, s);
```

```
            for(Fathers A_father:Fathers.values()){

                    s.add(LMatrix2D.allDiff(LMatrix3D.
                        extractArray2D(Family,-1,-1,
                        A_father.getOrdinal())));
                    s.add(GenericFunctions.getSum(
                        LMatrix3D.extractArray(Family
                        ,-1,Daughter_Son.girl.
                        getOrdinal(),A_father.
                        getOrdinal()), s).eq(
                        GenericFunctions.getSum(
                        LMatrix3D.extractArray(Family
                        ,-1,Daughter_Son.boy.getOrdinal
                        (),A_father.getOrdinal()), s)))
                        ;
        s.add(GenericFunctions.getMul(LMatrix3D.
            extractArray(Family,-1,Daughter_Son.girl.
            getOrdinal(),A_father.getOrdinal()),
            LMatrix3D.extractArray(Family,-1,
            Daughter_Son.girl.getOrdinal(),A_father.
            getOrdinal()), s).eq(GenericFunctions.
            getMul(LMatrix3D.extractArray(Family,-1,
            Daughter_Son.boy.getOrdinal(),A_father.
            getOrdinal()),LMatrix3D.extractArray(Family
            ,-1,Daughter_Son.boy.getOrdinal(),A_father.
            getOrdinal()), s)));
        }

    for (Fathers A_father : Fathers.values() )
        for( Daughter_Son A_kid : Daughter_Son.values() )
        {
                int I = 0;
                Iterator<Integer> iter = (new
                    MultiInterval(1, Last_child)).
                    iterator();
                while (iter.hasNext())
                {
                        I = iter.next();
                        s.add(Family[I][A_kid.getOrdinal
                            ()][A_father.getOrdinal()].
                            lt( Family[I-1][A_kid.
                            getOrdinal()][A_father.
                            getOrdinal()]));
                }
        }
    s.add(Family[Last_child][Daughter_Son.girl.getOrdinal
        ()][Fathers.leo.getOrdinal()].lt(Family[
        Last_child][Daughter_Son.boy.getOrdinal()][
```

```
            Fathers.leo.getOrdinal()]));
        s.add(Family[Last_child][Daughter_Son.girl.getOrdinal
            ()][Fathers.rene.getOrdinal()].eq(0));
        s.add(GenericFunctions.getSum(LMatrix3D.matrixToArray
            (Family), s).eq(60));
    }

    /* DISPLAY SOLUTION */
        static void display() {
                for (Fathers A_father : Fathers.values() ) {
                        System.out.print("\nChildren of
                            father " + A_father.getOrdinal
                            () + "\n");
                    for( Daughter_Son A_kid : Daughter_Son.
                        values() ){
                            System.out.print(A_kid.
                                getOrdinal() + ":\t");
                        for (int A_child = 0; A_child <
                            Children; A_child++)
                            System.out.print(Family[A_child
                                ][A_kid.getOrdinal()][
                                A_father.getOrdinal()] + "
                                ");
                        System.out.print("\n");
                }
        }

     }


    public static void main(String[] args) throws Exception
        {

        constraints();
            s.add(LMatrix3D.labelMatrix(Family));
        boolean ris = s.check();
        if(ris)
            display();
        else
            System.out.println("impossibile trovare una
                soluzione");

    }

}
```

.2 Queens

.2.1 Charme

```
define queens (N, Queens)
{ /* data structure: a vector of size N
     */
  /* Queens[I] = J means there is a queens column I line J
       */
  /* set also the constraint: exactly one queen by column
       */
  array Queens :: [1..N] of 1..N;

  /* No two queens of the same diagonal
       */
  for I in 1..N do
    for J in I+1..N do
    { Queens[I]-Queens[J] != I-J;
      Queens[I]-Queens[J] != J-I;
    }

  /* No two queens of the same line
       */
  all_diff Queens;
}

/* MAIN */

define main
{ init N;
  queens (N, Queens);
  draw_board N;
  display_when_known (N, Queens);
  find_all
  { generate Queens;
    count_sol N;
  };
  end;
}

/* RUN */

main;
```

.2.2 Java + JSetL

```
package applications.charme;

import java.util.Iterator;

import JSetL.*;
import JSetL.lib.LArrayOps;
import JSetL.lib.MetaOps;
import JSetL.lib.WhenKnownBody;
import JSetL.lib.LArray;

public class ExamplesBortolotti_Queens {

        static Integer N = 8;
        static IntLVar[] Queens;
        static SolverClass solver;
        static MetaOps metaOps;
        static LArrayOps arrayOps;
        public static void initialize() throws Exception
        {
                solver = new SolverClass();
                metaOps = new MetaOps(solver);
                arrayOps = new LArrayOps(solver);
                Queens = LArray.generateArray(new
                    MultiInterval(0,N-1), new MultiInterval
                    (0,N-1), solver);

        }

        public static void  queens (SolverClass solver)
            throws Exception
        {

                Iterator<Integer> ind1 = new MultiInterval
                    (1,N).iterator();
                while(ind1.hasNext())
                {
                        Integer I = ind1.next();
                        Iterator<Integer> ind2 = new
                            MultiInterval(I+1,N).iterator()
                            ;

                        while(ind2.hasNext())
                        {
                                Integer J = ind2.next();
                                solver.add(Queens[I-1].sub(
                                    Queens[J-1]).neq((I-1)
                                    -(J-1)));
                                solver.add(Queens[I-1].sub(
                                    Queens[J-1]).neq((J-1)
```

```
                                        -(I-1)));
                    }
                }
                solver.add(IntLVar.allDifferent(Queens));
        }

        public static void main(String[] argv) throws
            Exception
        {

                initialize();
                queens(solver);

                solver.add(metaOps.when_known(Queens, new
                    WhenKnownBody() {

                        @Override
                        public void when_true() {
                                Integer i;
                                for (i = 0; i < N; ++i) {
                                int queen = Queens[i].
                                    getValue();
                                Integer j = 0;
                                for (; j < queen; ++j)
                                        System.out.print(". ");
                                        System.out.print("@ ");
                                        for (j = queen + 1;
                                            j < N; ++j)
                                                System.out.
                                                    print(" . ");
                                                System.out.
                                                    println
                                                    ();
                                }
                                System.out.print("\n");
                        }

                        @Override
                        public void when_false() {
                                // DO NOTHING
                        }

                        @Override
                        public boolean control() {
                                return true;
                        }
```

```
                }));

                solver.add(LArray.labelArray(Queens));

                boolean result = solver.check();

                if(result)
                {
                        System.out.println("soluzione
                            trovata");
                }
                else
                System.out.println("impossibile trovare una
                    soluzione");
                return;
        }

}
```