Generazione automatica ed esecuzione di casi di test per la libreria Java JSetL

Candidato: Francesco Vetere
Relatore: Prof. Gianfranco Rossi

Università di Parma
Dipartimento di Scienze Matematiche, Fisiche e Informatiche
Corso di Laurea in Informatica

26 Settembre 2019

Struttura della presentazione

- Obiettivi
- Contesto
  - Testing
  - JSetL
  - {log}
- Approccio utilizzato
- Lo strumento realizzato
  - Architettura
  - Esempi d'uso
  - Analisi dei tempi
- Conclusioni e sviluppi futuri

Obiettivi

Obiettivi

- Testing sistematico ed esaustivo dei *vincoli* della libreria Java JSetL.
- Testing affrontato dal punto di vista della soddisfacibilità: per ogni vincolo, istanziato su una particolare combinazione di argomenti, si vuole ottenere un risultato (booleano) dal solver di JSetL.

Testing

Categoria di testing

- Black-box testing

  Utente non obbligato a conoscere i dettagli implementativi della libreria oggetto del testing.

- Equivalence Class Partitioning

  Spazio dei possibili input suddiviso per classi di equivalenza: test vector costruito scegliendo da ogni classe un valore campione.

Fase del processo di testing interessata

- Unit testing

  Testing relativo ai moduli di base di un sistema software (Framework JUnit 4).

JSetL

Che cos'è

- Libreria Java che combina il paradigma O-O con il paradigma logico a vincoli.
- JSetL implementa il linguaggio $L_{\mathcal{SET}}$, linguaggio logico basato su vincoli insiemistici (nella sua successiva estensione $L_{\mathcal{BR}}$, vengono trattati anche vincoli relazionali).

Esempi di vincoli insiemistici in JSetL

Vincolo	JSetL	Significato
Uguaglianza	`A.eq(B)`	$A = B$
Disgiunzione	`A.disj(B)`	$A \cap B = \varnothing$
Unione	`A.union(B,C)`	$C = A \cup B$
Intersezione	`A.inters(B,C)`	$C = A \cap B$
Differenza	`A.diff(B,C)`	$C = A \setminus B$

{log}

Che cos'è

- Lo stesso linguaggio $L_{\mathcal{SET}}$ è implementato in ambiente Prolog attraverso {log}.
- Il linguaggio $L_{\mathcal{SET}}$ è dunque il nucleo comune a JSetL e {log}.

Confronto tra vincoli insiemistici in JSetL/{log}

Vincolo	JSetL	{log}	Significato
Uguaglianza	`A.eq(B)`	`A = B`	$A = B$
Disgiunzione	`A.disj(B)`	`disj(A,B)`	$A \cap B = \varnothing$
Unione	`A.union(B,C)`	`un(A,B,C)`	$C = A \cup B$
Intersezione	`A.inters(B,C)`	`inters(A,B,C)`	$C = A \cap B$
Differenza	`A.diff(B,C)`	`diff(A,B,C)`	$C = A \setminus B$

$\Longrightarrow$ Stesso significato, sintassi differenti

Approccio utilizzato

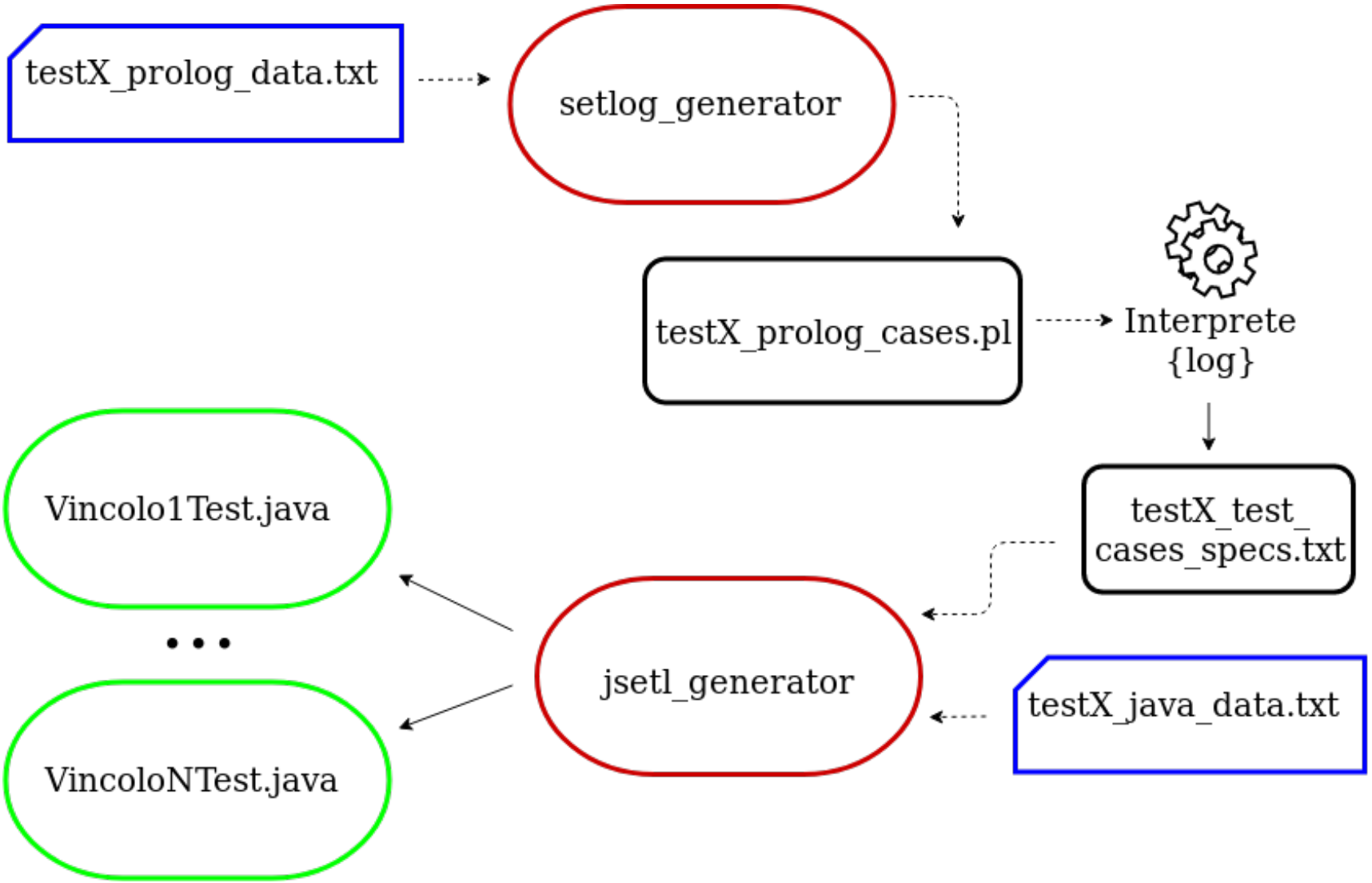
Approccio

- Progettazione e sviluppo di uno strumento software *ad hoc* per il testing dei vincoli della libreria JSetL.
- Lo strumento software realizzato sfrutta il risolutore di {log} per ricavare i risultati dei singoli test, per poi riportarli su JSetL senza ricalcolarli.

Vantaggi

- Utente non è costretto a conoscere in anticipo i risultati di soddisfacibilità dei vincoli.
- Testing indiretto di {log}: le due implementazioni devono mantenere coerenza.

Architettura



Esempi d'uso

Collaudo dello strumento realizzato

- Data la complessità del processo di generazione, è stato inserito uno script (Linux/Windows) che esegue in automatico tutti i comandi necessari ad ogni step.
- Lo strumento realizzato è stato collaudato attraverso vari test set di prova.
- Esempio: test1

  Scopo: testing dei vincoli `eq`, `disj` e `union` su diverse tipologie di insiemi logici e loro combinazioni.

Esempio: test1

test1_prolog_data.txt

```
//regole definite da utente
$$$ user rules $$$

//insiemi o relazioni del test set
$$$ args $$$
vuoto${}${}${}
var$S1$S2$S3
chiuso${X1,Y1}${X2,Y2}${X3,Y3}
aperto${A1/B1}${A2/B2}${A3/B3}

//vincoli del test set
$$$ constraints $$$
eq$2$infix$=
disj$2$prefix$disj
union$3$prefix$un
```

Esempio: test1

test1_java_data.txt

```
//codice globale
$$$ global $$$

//codice locale
$$$ local $$$

//insiemi o relazioni del test set
$$$ args $$$
vuoto$LSet E1 = LSet.empty()$...
var$LSet V1 = new LSet("S1")$...
chiuso$LSet C1 = LSet.empty().ins(new LVar("X1")).ins(new
    LVar("Y1"))$...
aperto$LSet A1 = new LSet("B1").ins(new LVar("A1"))$...

//vincoli del test set
$$$ constraints $$$
eq$2$infix
disj$2$infix
union$3$infix
```

Esempio: test1

DisjTest.java

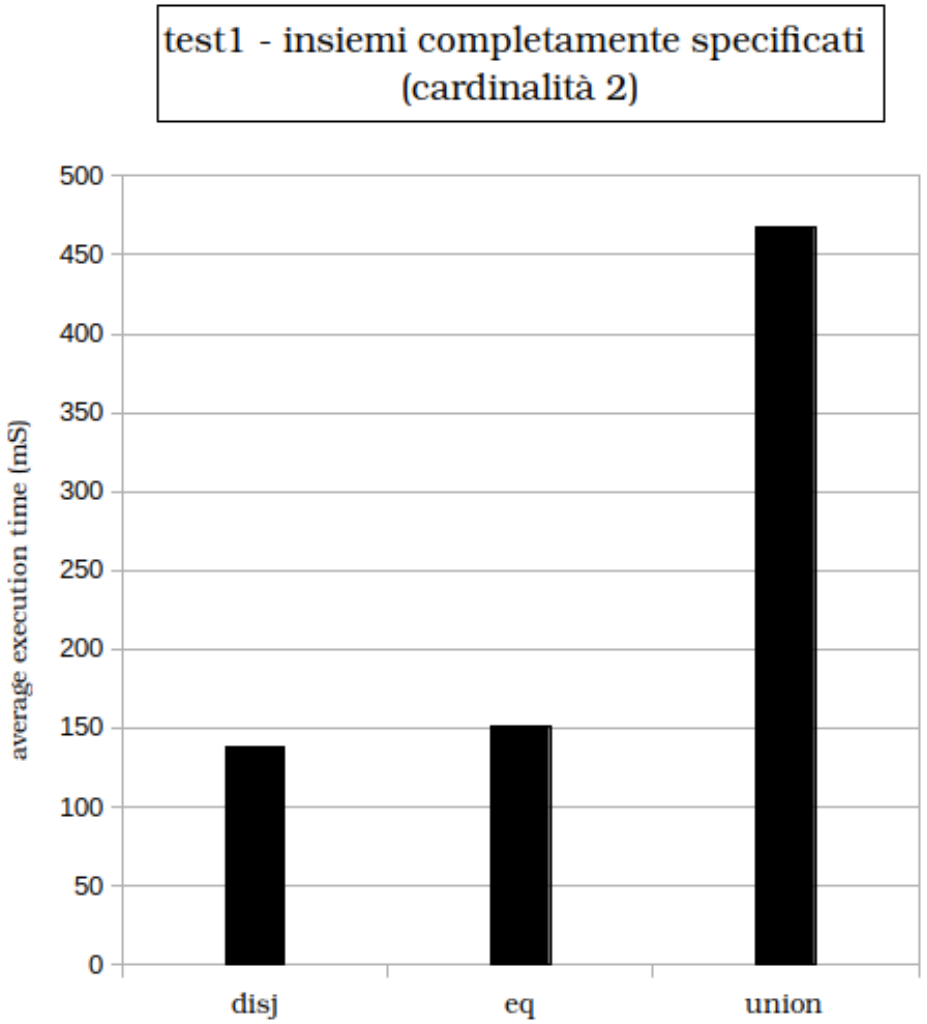
```
public class DisjTest {
    //...

    @Test
    public void testDisj_Vuoto_Vuoto() {
        Solver solver = new Solver();
        LSet E1 = LSet.empty();
        LSet E2 = LSet.empty();
        solver.add(E1.disj(E2));
        assertTrue(solver.check());
    }
    //...
}
```

- Se n è l'arità di un vincolo ed m sono i diversi argomenti specificati nei file di input, verranno generati n^m metodi di test

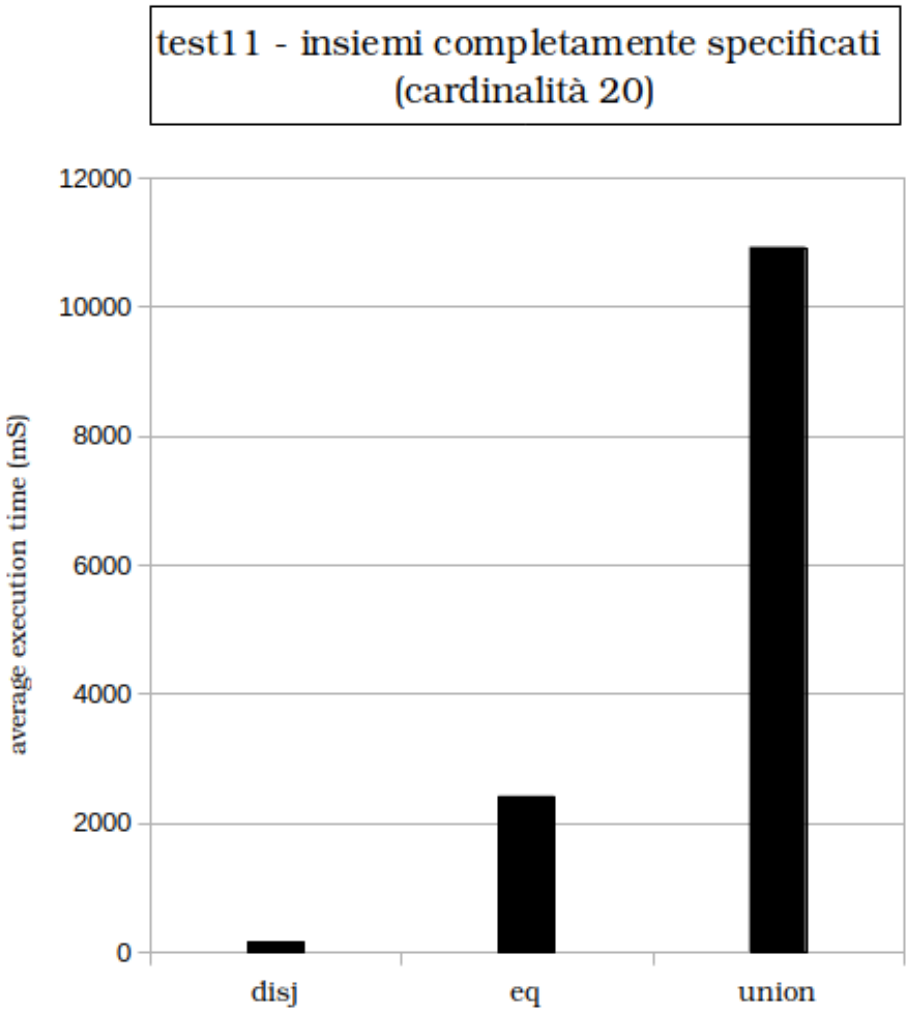
Analisi dei tempi di esecuzione

- Oltre al testing di soddisfacibilità, prevista (opzionalmente) anche l'analisi dei tempi di esecuzione.
- Utile per evidenziare in che modo i vincoli riescano a scalare all'aumentare della cardinalità degli insiemi.

test1


test1 - insiemi completamente specificati
(cardinalità 2)
average execution time (mS)
500
450
400
350
300
250
200
150
100
50
0
disj
eq
union

test11


test11 - insiemi completamente specificati
(cardinalità 20)
average execution time (mS)
12000
10000
8000
6000
4000
2000
0
disj
eq
union

Conclusioni

- In questo lavoro di tesi è stato affrontato il problema del testing dei vincoli di JSetL realizzando uno strumento software ad hoc avente il compito di generare test cases sistematici in maniera automatica.
- La presenza dello strumento fornisce la possibilità di testare la correttezza di qualsiasi vincolo presente nella libreria, nonché di misurare le prestazioni relative alla loro risoluzione.

Sviluppi futuri

- Esplorare un possibile approccio parallelo attraverso l'esecuzione multi-thread dei metodi di test (JUnit 5).
- Per i vincoli soddisfacibili, dimostrare che le soluzioni prodotte da {log} siano prodotte anche da JSetL.

Grazie per l'attenzione