

# Elaborato finale in Software Testing

## **Realizzazione di uno strumento di testing combinatoriale**

Anno accademico 2020/2021

Candidato:

**Amato Paolo**

Matricola:

**M63001065**

## Sommario

Elaborato finale in Software Testing .....	1
1) Introduzione .....	3
2) Analisi dello strumento software .....	5
2.1) Traccia .....	5
2.2) Tecnologie e tools adoperati .....	6
2.3) Funzionamento .....	10
2.4) Prestazioni.....	28
2.5) Limitazioni.....	31
3) Il problema dell'oracolo .....	32
4) Assertion based testing .....	34
4.1) Costo .....	35
5) Model based testing .....	36
5.1) Costo .....	37
6) Riferimenti .....	38

## 1) Introduzione

Il testing combinatorio è una metodologia che può ridurre i costi e migliorare in maniera significativa l'efficacia del testing per molte applicazioni. L'intuizione chiave alla base di questa forma di test è che non tutti i parametri contribuiscono ad ogni guasto, e i dati empirici suggeriscono che quasi tutti i guasti del software sono causati da singoli parametri o da interazioni tra relativamente pochi parametri. Questa scoperta ha importanti implicazioni per il testing, in quanto suggerisce che il testing di combinazioni di parametri può fornire una capacità di rilevamento dei guasti altamente efficace.

Avendo capito che i guasti di un sistema software possono derivare dall'interazione di condizioni che potrebbero essere innocue se considerate individualmente, gli sviluppatori di software utilizzano da tempo il "test a coppie", in cui tutte le possibili coppie di valori dei parametri sono coperte da almeno un caso di test. Tuttavia, può anche accadere che ci siano guasti che vengono attivati da una inusuale – in molti casi non prevedibile – interazione combinatoria di più di due parametri.

Uno studio su di un dispositivo medico [1] ha rilevato un caso in cui la manifestazione di uno specifico guasto dipende da un'interazione a quattro vie tra i valori dei parametri. Successive indagini circa tale dispositivo hanno riscontrato una distribuzione di questo tipo: molti guasti innescati da particolari valori di specifici di parametri presi singolarmente, un numero minore ma comunque consistente di guasti dovuto ad un'interazione tra due parametri, e un numero progressivamente più piccolo di guasti innescati da interazioni a 3,4,5 e 6 vie.

Risultati del genere, riscontrabili anche in altri contesti, ci consentono di affermare che, per raggiungere un livello più elevato di garanzia del software, è necessario un testing combinatoriale che vada ad esercitare anche interazioni combinatorie di alto grado (a quattro vie o superiore).

Se si sa per esperienza che “t” o meno variabili sono coinvolte in difetti per un particolare tipo di applicazione, e si è in grado di generare una test suite di modo tale che tutte le t-ple di classi di equivalenza diverse siano coperte almeno una volta, allora si può avere una

fiducia ragionevolmente alta circa il corretto funzionamento dell'applicazione per la maggior parte del tempo di utilizzo di tale sistema software.

L'ingrediente principale per questa tipologia di testing è noto come “array di copertura”, un oggetto matematico in cui tutte le combinazioni t-way dei valori dei parametri sono coperte almeno una volta. La generazione di array di copertura per interazioni complesse (oltre il 2-way) è un problema in generale complicato, ma sono stati sviluppati algoritmi che consentono di generare array di copertura con una velocità di diversi ordini di grandezza superiore rispetto ai metodi precedenti, rendendo gli array di copertura fino a 6 vie trattabili per molte applicazioni.

Nuovi algoritmi, abbinati a processori veloci ed economici e alla possibilità di sfruttare la computazione basata su GPU, stanno rendendo i sofisticati test combinatori un approccio pratico che può soddisfare la promessa di test migliori del software a un costo inferiore.

Resta ovviamente da affrontare il cosiddetto problema dell'oracolo, del quale si discute in seguito.

## 2) Analisi dello strumento software

In questa sezione della tesina vengono approfonditi il funzionamento dello strumento realizzato, i dettagli implementativi ad esso relativi, gli strumenti già esistenti e le tecnologie di cui l'applicazione stessa fa uso.

### 2.1) Traccia

Realizzazione di uno strumento di testing combinatoriale:

- Nel contesto di testing di unità per classi scritte in linguaggio Java.
- Con eventuale utilizzo di uno strumento per la generazione del dizionario dei dati.
- In combinazione con strumenti whitebox per la valutazione della copertura.
- In combinazione con tecniche di riduzione della test suite generata.

Tale strumento non si occuperà in maniera automatica della generazione dell'oracolo per la test suite minimizzata. Ho approfondito tale problema, in linea teorica, negli ultimi capitoli del documento ("Il problema dell'oracolo", "Assertion based testing", "Model based testing").

## 2.2) Tecnologie e tools adoperati

- Linguaggio C#, ambiente di sviluppo Visual Studio.

Per evitare di incorrere in problemi nella compilazione del progetto o nell'esecuzione dell'applicazione, è consigliato disporre di una versione del .NET framework uguale o superiore alla 4.7.2. Ad ogni modo, dalla versione otto di Windows il framework è divenuto parte integrante del sistema operativo e viene generalmente aggiornato in maniera automatica.

- CTWedge.

Si tratta di un tool per la produzione automatica di test combinatori reso disponibile dall'università di Bergamo. Estende il framework CitLab realizzato da Paolo Valvassori.

CitLab è un laboratorio per il testing combinatoriale con le seguenti caratteristiche:

1) Un ricco linguaggio astratto con una semantica formale e precisa per specificare problemi combinatori.

2) Una sintassi concreta con una grammatica ben definita che consente ai professionisti di scrivere modelli e ai ricercatori di condividere esempi e benchmark scritti in una notazione "comune".

3) Un framework basato su Eclipse Modeling Framework (EMF) che fornisce strumenti e supporto in fase di esecuzione per produrre (automaticamente) un set di classi Java per modelli combinatori, insieme a un set di classi di adattatori e librerie di utilità che consentono di manipolare problemi combinatori in applicazioni Java utilizzando semplici API. Ciò consente agli sviluppatori di accedere a modelli combinatori all'interno dei loro programmi e strumenti.

4) Un editor integrato nell'IDE Eclipse per la gestione di problemi combinatori.

L'editor fornisce agli utenti tutte le funzionalità previste in un ambiente di programmazione moderno come l'evidenziazione della sintassi, il completamento del codice, il controllo degli errori in fase di esecuzione, le correzioni rapide e la visualizzazione struttura.

5) Un framework per l'introduzione di nuovi algoritmi di generazione di test che possono essere aggiunti a CITLAB come plugin. Ciò consente ai ricercatori di sviluppare tecniche di nuova generazione e di inserirle nel framework senza l'onere

di definire una grammatica, un parser, un visitatore di un albero di sintassi astratto e così via.

6) etc...

Quando si fa uso di CTWedge bisogna descrivere i parametri del test parametrizzato mediante un modello. Esempio:

```
1 Model Divisione
2 Parameters:
3   dividendo: [ 0..140 ]
4   divisore: {0,5,55,128,98}
5   risultato : {0}
6
7 Constraints:
```

È possibile inserire, in tale modello, anche vincoli relativi ai singoli parametri o ad interazioni tra due o più parametri.

Completato il modello, si seleziona il motore di generazione (ACTS o CASA), si sceglie il valore di k (es. 3-way), e il tool fornisce in uscita un file csv contenente le sequenze di valori di input create secondo la strategia impostata.

- Javac.

Si tratta del compilatore Java principale incluso nel Java Development Kit di Oracle Corporation. Il compilatore accetta codice sorgente conforme alla specifica del linguaggio Java (JLS) e produce bytecode Java conforme alla specifica JVMMS (Java Virtual Machine Specification).

La versione presente sul mio PC è la 11.0.7, versioni precedenti potrebbero causare problemi dato l'utilizzo di Junit 5.

- Junit 5.

Si tratta della nuova versione di Junit rilasciata a partire dal 2017, per applicazioni sviluppate con Java 8 e successive.

La creazione dello strumento software presentato in questa sede è stata resa possibile grazie ad una delle novità introdotte in questa versione di Junit, ossia i test parametrizzati.

I test parametrizzati consentono di eseguire un test più volte con argomenti diversi.

Sono dichiarati proprio come i normali metodi `@Test` ma usano invece l'annotazione `@ParameterizedTest`.

E' necessario dichiarare almeno una sorgente che fornirà gli argomenti per ogni invocazione e quindi utilizzare gli argomenti nel metodo di test.

Esempio di test parametrizzato:

```
public class Numbers {  
    public static boolean isOdd(int number) {  
        return number % 2 != 0;  
    }  
}  
  
@ParameterizedTest  
@ValueSource(ints = {1, 3, 5, -3, 15, Integer.MAX_VALUE}) // six numbers  
void isOdd_ShouldReturnTrueForOddNumbers(int number) {  
    assertTrue(Numbers.isOdd(number));  
}
```

Lo strumento consente, come sarà spiegato successivamente, di impostare un tempo massimo per l'esecuzione di ciascun caso di test. L'annotazione @Timeout, per qualche motivo, non funziona con i test parametrizzati.

Utilizzo la seguente alternativa:

```
static void assertTimeoutPreemptively(Duration timeout, Executable executable)  
    Asserts that execution of the supplied executable completes before the given timeout is exceeded.
```

- Jacoco.

Si tratta di una libreria di copertura del codice gratuita per Java, che è stata creata dal team di EclEmma sulla base delle lezioni apprese dall'utilizzo e dall'integrazione di librerie esistenti per molti anni.

Jacoco funziona in questo modo:

- 1) Istrumenta l'applicazione solo al tempo della valutazione della copertura.
- 2) Registra tutte le esecuzioni di ogni istruzione del bytecode.
- 3) Associa le istruzioni di bytecode alle corrispondenti istruzioni nel codice sorgente.
- 4) Genera report della copertura delle righe di codice sorgente. Per la generica linea di codice sorgente, il valore di copertura è pari ad 1 se tutte le istruzioni del corrispondente bytecode sono state coperte, 0 se nessuna istruzione del bytecode corrispondente è stata coperta, valore frazionario compreso tra 0 e 1 se il bytecode corrispondente è stato coperto solo parzialmente.

Una riga di codice sorgente con valore di copertura 1 viene colorata di verde.


Una riga di codice sorgente con valore di copertura 0 viene colorata di rosso.











Una riga di codice sorgente con valore di copertura frazionario viene colorata di giallo.

Quanto appena detto è fondamentale da tenere a mente, verrà sfruttato nella procedura di minimizzazione della test suite.

Esempio di report di copertura prodotta da Jacoco:

 JaCoCo

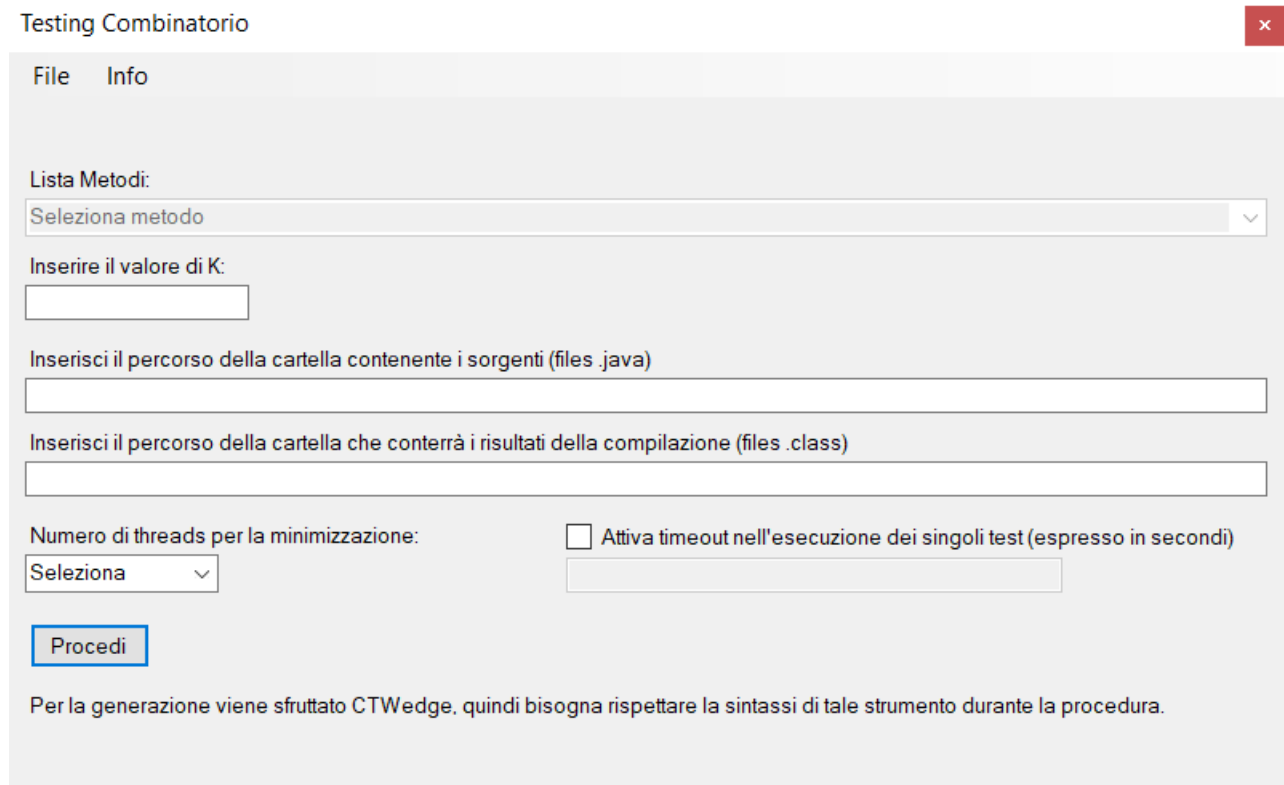
### JaCoCo

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
 <a href="#">org.jacoco.examples</a>	<div><div></div></div>	58%	<div><div></div></div>	64%	24	53	97	193	19	38	6	12
 <a href="#">org.jacoco.core</a>	<div><div></div></div>	97%	<div><div></div></div>	93%	106	1,378	114	3,311	20	710	2	136
 <a href="#">org.jacoco.agent.rt</a>	<div><div></div></div>	77%	<div><div></div></div>	84%	31	121	62	310	21	74	7	20
 <a href="#">jacoco-maven-plugin</a>	<div><div></div></div>	90%	<div><div></div></div>	81%	35	183	44	407	8	110	0	19
 <a href="#">org.jacoco.cli</a>	<div><div></div></div>	97%	<div><div></div></div>	100%	4	109	10	275	4	74	0	20
 <a href="#">org.jacoco.report</a>	<div><div></div></div>	99%	<div><div></div></div>	99%	4	572	2	1,345	1	371	0	64
 <a href="#">org.jacoco.ant</a>	<div><div></div></div>	98%	<div><div></div></div>	99%	4	163	8	429	3	111	0	19
 <a href="#">org.jacoco.agent</a>	<div><div></div></div>	86%	<div><div></div></div>	75%	2	10	3	27	0	6	0	1
Total	1,351 of 27,196	95%	143 of 2,125	93%	210	2,589	340	6,297	76	1,494	15	291

La versione da me utilizzata è la 0.8.6, la più recente disponibile al momento della stesura di questo documento.

## 2.3) Funzionamento

All'avvio dell'applicazione, viene mostrata la seguente schermata.



The screenshot shows a window titled "Testing Combinatorio" with a red close button in the top right corner. The window has a menu bar with "File" and "Info". Below the menu bar, there is a section titled "Lista Metodi:" with a dropdown menu labeled "Seleziona metodo". Below this is a label "Inserire il valore di K:" followed by a text input field. Then, there is a label "Inserisci il percorso della cartella contenente i sorgenti (files .java)" followed by a text input field. Below that is a label "Inserisci il percorso della cartella che conterrà i risultati della compilazione (files .class)" followed by a text input field. At the bottom, there are two options: "Numero di threads per la minimizzazione:" with a dropdown menu labeled "Seleziona", and a checkbox labeled "Attiva timeout nell'esecuzione dei singoli test (espresso in secondi)" followed by a text input field. A "Procedi" button is located below these options. At the very bottom, there is a note: "Per la generazione viene sfruttato CTWedge, quindi bisogna rispettare la sintassi di tale strumento durante la procedura."

La prima cosa da fare è caricare la classe Java contenente il metodo da testare.

A questo scopo, cliccare su “File” e poi su “Carica Classe Java”.

Testing Combinatorio ×

File

Info

Carica Classe Java

Lista Metodi:

Seleziona metodo ▼

Inserire il valore di K:

Inserisci il percorso della cartella contenente i sorgenti (files .java)

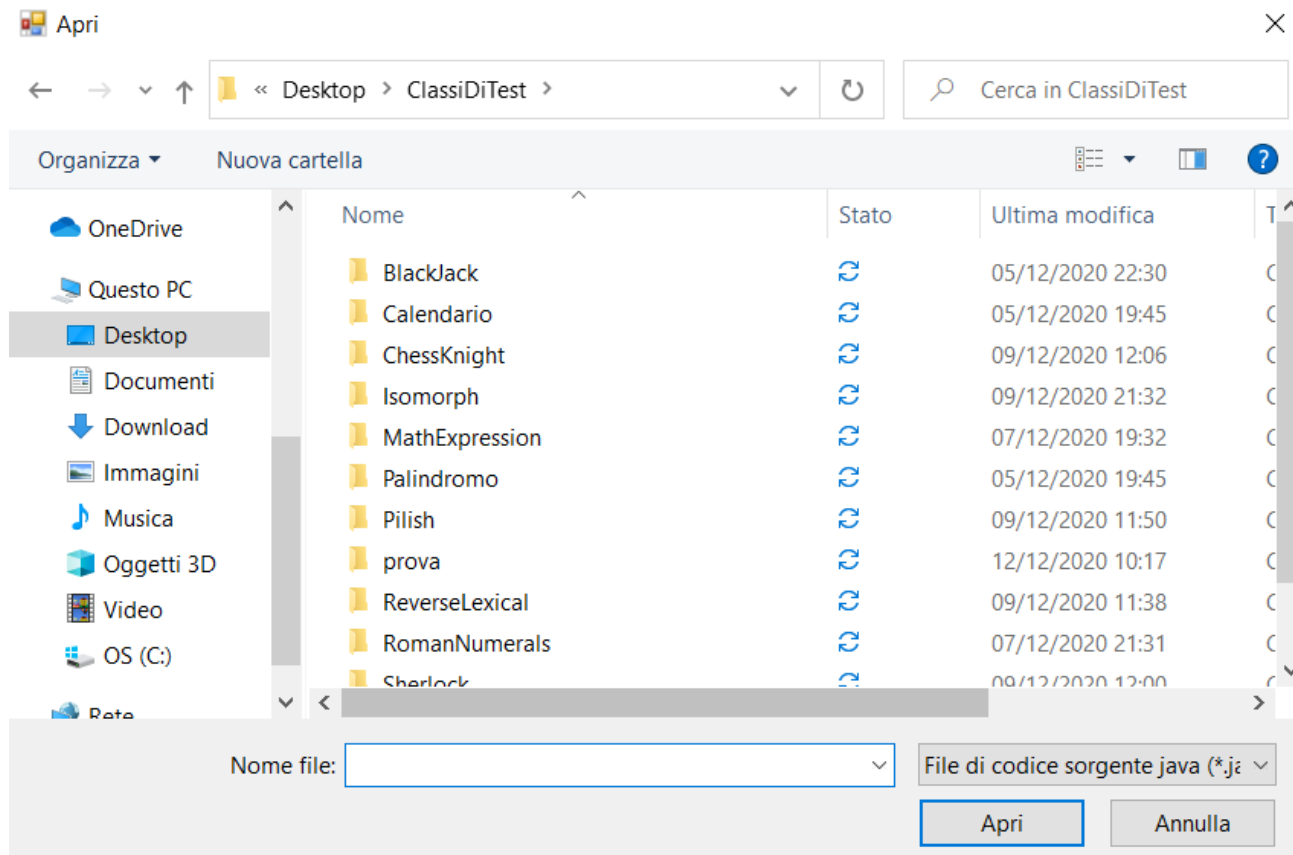
Inserisci il percorso della cartella che conterrà i risultati della compilazione (files .class)

Numero di threads per la minimizzazione: Seleziona ▼ ☐ Attiva timeout nell'esecuzione dei singoli test (espresso in secondi)

Procedi

Per la generazione viene sfruttato CTWedge, quindi bisogna rispettare la sintassi di tale strumento durante la procedura.

Così facendo, viene mostrata una OpenFileDialog che chiede di selezionare un file con estensione “.java”.



(Tutte le cartelle qui visibili sono relative a classi prelevate da Internet, utilizzate per il testing dell'applicazione. Tali classi sono disponibili nel repository di progetto.)

L'applicazione è dotata di un piccolo analizzatore statico per classi Java da me realizzato, che si basa sull'utilizzo delle espressioni regolari.

Completata la lettura del file che rappresenta la classe, la seguente funzione consente l'estrazione dei metodi contenuti nella classe:

```
1 public static string[] GetMethods(string sourceCode) {
2
3     var regex = new Regex(@"((public|private|protected|static|final|native|synchronized|abstract|transient)+\s)+[\$_\w\<\>\w\s\[\]]*\s*[\$_\w]+\(([\^)]*)?\s*");
4
5     var matches = regex.Matches(sourceCode).Cast < Match > ().Select(m => m.Value).ToArray();
6     for (var i = 0; i < matches.Length; i++) {
7
8         var len = matches[i].Length;
9         var arr = matches[i].Split(' ');
10        var found = false;
11        foreach (var str in arr) {
12            if (str != "private") continue;
13            matches[i] = "-.-";
14            found = true;
15        }
16
17        if (found) {
18            continue;
19        }
20
21        matches[i] = matches[i].Replace("\r", "").Replace("\n", "").Replace("\t", "");
22    }
23    return matches;
24 }
```

Vengono opportunamente etichettati e in seguito filtrati i metodi con visibilità privata, in quanto non invocabili dalla classe di test che viene successivamente generata.

Per semplicità, da ora in poi si fa riferimento alla classe “RomanNumeral”.

Questa classe gestisce la conversione da numeri arabi a numeri romani e viceversa.

Si procede selezionando il metodo da testare dalla lista dei metodi. Supponiamo di selezionare “public String toRoman(int arabic)”.

Nella seconda casella di testo bisogna inserire il valore k per la strategia k-way da utilizzare.

Nella terza e nella quarta casella di testo si inseriscono, rispettivamente, il percorso della cartella contenente i files sorgente (estensione .java) e il percorso della cartella in cui inserire i risultati della compilazione (estensione .class).

Se lo si desidera, è possibile inserire un time-out per l’esecuzione dei singoli casi di test spuntando la casella “Attiva time-out nell’esecuzione dei singoli test” e inserendo nella casella di testo sottostante il valore di tale time-out.

Bisogna inoltre selezionare il numero di threads (1,2 o 3) da utilizzare nel processo di minimizzazione della test suite.

Ho riscontrato, almeno sulla mia macchina, che sfruttando il parallelismo le prestazioni migliorano quando il numero di casi di test è elevato.

In questo esempio, che sfrutto per mostrare il funzionamento dello strumento, la configurazione usata è la seguente.

Come riportato anche nell'applicazione, la generazione dei test combinatori avviene per mezzo di CTWedge.

Cliccando su procedi, verrà mostrato un InputBox per ogni parametro di ingresso del metodo oggetto di test e verrà chiesto di inserire l'insieme dei possibili valori per tale parametro. Nel fare ciò, bisogna rispettare le regole che caratterizzano un tipico modello di CTWedge.

Analoga cosa va fatta anche per il valore di ritorno del metodo.

Parametro n. 1 ✕

Inserisci l'insieme dei possibili valori per il parametro "arabic" di tipo "int"

Valore di ritorno ✕

Inserisci un dizionario dei dati per il valore di ritorno di tipo "String"

Viene poi richiesto di inserire eventuali vincoli a completamento del modello.

Inserisci qui i vincoli — □ ✕

La creazione del modello CTWedge viene effettuata mediante il seguente metodo:

```
1 private StringBuilder CreaModello(string metodo) {
2     var parametri = JavaClassAnalyzer.GetParameters(metodo).ToList();
3     var count = 1;
4     var sb = new StringBuilder();
5     sb.AppendLine("Model " + Path.GetFileNameWithoutExtension(openFileDialogClass.FileName));
6     sb.AppendLine("Parameters:");
7     foreach(var parametro in parametri) {
8         var t = JavaClassAnalyzer.AnalizzaParametro(parametro, count);
9         var nome = t.Item3;
10        var tipo = t.Item2;
11        var title = "Parametro n. " + count;
12        var prompt = @"Inserisci l'insieme dei possibili valori per il parametro "" + nome + @"" di tipo "" + tipo + @""";
13        var dizionario = Interaction.InputBox(prompt, title);
14        sb.AppendLine(nome + " : " + dizionario);
15        count++;
16    }
17
18    var returnType = JavaClassAnalyzer.GetReturnType(metodo);
19
20    var n = "risultato";
21    var tl = "Valore di ritorno";
22    var p = @"Inserisci un dizionario dei dati per il valore di ritorno di tipo "" + returnType + @""";
23    var d = Interaction.InputBox(p, tl);
24    sb.AppendLine(n + " : " + d);
25
26    sb.AppendLine("Constraints:");
27    var vincoli = "";
28    ShowInputDialog(ref vincoli, "Inserisci qui i vincoli");
29    sb.AppendLine(vincoli);
30    return sb;
31 }
```

Il metodo AnalizzaParametro prende in ingresso la firma di un metodo e restituisce una coppia del tipo (tipo parametro, nome parametro).

```
1 public static Tuple < int,
2     string,
3     string > AnalizzaParametro(string parametro, int count) {
4     parametro = parametro.Trim().Replace("final ", "");
5     var attributi = new List < string > ();
6     attributi = parametro.Split(' ').ToList();
7     var tipo = attributi[0];
8     var nome = attributi[1];
9     return Tuple.Create(count, tipo, nome);
10
11 }
```

Il metodo GetReturnType prende in ingresso la firma di un metodo e restituisce il tipo del valore di ritorno del metodo.



```

1 public static string GetReturnType(string metodSig) {
2     var str = metodSig.Split(new[] {
3         '(',
4         ')',
5     },
6     StringSplitOptions.RemoveEmptyEntries);
7     var arr = str[0].Split(' ');
8     var len = arr.Length;
9     return arr[len - 2];
10 }

```

Al termine della creazione del modello, viene inviata una richiesta http a

<https://foselab.unibg.it/ctwedge/> per recuperare il file csv contenente i casi di test.

Il metodo che si occupa di questo è InviaRichiesta.

```

1 private static string InviaRichiesta(string modello, string k) {
2     var request = (HttpWebRequest) WebRequest.Create("https://foselab.unibg.it/ctwedge/generator/");
3
4     var postData = "model=" + Uri.EscapeDataString(modello);
5     postData += "&strength=" + Uri.EscapeDataString(k);
6     postData += "&generator=" + Uri.EscapeDataString("ACTS");
7     postData += "&ignConstr" + Uri.EscapeDataString("false");
8     var data = Encoding.ASCII.GetBytes(postData);
9
10    request.Method = "POST";
11    request.ContentType = "application/x-www-form-urlencoded";
12    request.ContentLength = data.Length;
13
14    using(var stream = request.GetRequestStream()) {
15        stream.Write(data, 0, data.Length);
16    }
17
18    var response = (HttpWebResponse) request.GetResponse();
19
20    var responseString = new StreamReader(response.GetResponseStream()).ReadToEnd();
21    return responseString;
22 }

```

Appena ricevuta la risposta, viene eseguito questo metodo:

```
1 private static void SalvaCsv(string CTanswer) {  
2     var path = Directory.GetCurrentDirectory() + @"\data.csv";  
3  
4     dynamic array = JsonConvert.DeserializeObject(CTanswer);  
5  
6     if (CTanswer.Contains(".csv")) {  
7         using(var client = new WebClient()) {  
8             string name = array.result;  
9  
10            var url = "https://foselab.unibg.it/ctwedge/results/?name=" + name;  
11  
12            var answer = client.DownloadString(url);  
13            while (answer == "" || answer.Contains("Please")) answer = client.DownloadString(url);  
14            client.DownloadFile(url, Directory.GetCurrentDirectory() + @"\data.csv");  
15            var lines1 = File.ReadAllLines(path);  
16            File.WriteAllLines(path, lines1.Skip(1).ToArray());  
17        }  
18  
19        return;  
20    }  
21  
22    string csv = array.result.ToString();  
23    csv = csv.Replace(';', ',');  
24    // string path = Directory.GetCurrentDirectory() + "/data.csv";  
25    File.WriteAllText(path, csv);  
26    var lines = File.ReadAllLines(path);  
27    File.WriteAllLines(path, lines.Skip(1).ToArray());  
28 }
```

Questo metodo serve a gestire il fatto che, nel caso in cui la risposta sia di “piccole dimensioni”, CTWedge la restituisce direttamente in http, altrimenti bisogna scaricare un file csv appositamente generato dal tool.

Lo step successivo consiste nella creazione della classe di test. Il metodo “SalvaClasseTest” assolve a questo compito.

```

1 private void SalvaClasseTest(string metodo) {
2     var parametri = JavaClassAnalyzer.GetParameters(metodo);
3     var methodName = JavaClassAnalyzer.GetMethodName(metodo);
4     var returnType = JavaClassAnalyzer.GetReturnType(metodo);
5     var className = lblNomeClasse.Text.Substring(13);
6     var classObjName = "obj" + className;
7     var sb = new StringBuilder();
8     sb.AppendLine(@"import static org.junit.Assert.*;");
9     sb.AppendLine(@"import org.junit.jupiter.api.*;");
10    sb.AppendLine(@"import org.junit.jupiter.api.Test;");
11    sb.AppendLine(@"import org.junit.jupiter.params.ParameterizedTest;");
12    sb.AppendLine(@"import org.junit.jupiter.params.provider.CsvFileSource;");
13    sb.AppendLine(@"import org.junit.jupiter.params.provider.CsvSource;");
14    sb.AppendLine(@"import java.time.Duration;");
15    sb.AppendLine("");
16    sb.AppendLine("class Test" + className + " {");
17
18    sb.AppendLine("\t" + className + " " + classObjName + ";");
19    sb.AppendLine("\t@BeforeEach");
20    sb.AppendLine("\tvoid init() {");
21
22    var inizializzazione = "\t\t" + classObjName + " = new " + className + "();";
23    // sb.AppendLine("\t\t" + classObjName + " = new " + className + "();");
24
25    var dialogResult = MessageBox.Show("Vuoi modificare lo scheletro di invocazione del costruttore relativo alla classe considerata? (Necessario nel caso in cui il costruttore preveda dei parametri)",
26
27    if (dialogResult == DialogResult.Yes) ShowInputDialog(ref inizializzazione, "Modifica costruttore");
28
29    sb.AppendLine(inizializzazione);
30    sb.AppendLine("\t\tassertNotNull(" + classObjName + ");");
31    sb.AppendLine("\t");
32    sb.AppendLine("");
33    sb.AppendLine("\t@ParameterizedTest");
34    sb.AppendLine("\t" + @"@CsvFileSource(resources = ""data.csv"");");
35    sb.AppendLine("\tvoid TestMethod(");
36    foreach (var t in parametri) sb.Append(t + ",");
37
38    sb.AppendLine(returnType + " risultato_atteso) {");
39
40    if (cbTimeout.Checked) sb.AppendFormat("\t\torg.junit.jupiter.api.Assertions.assertTimeoutPreemptively(Duration.ofSeconds({0}), () -> {{", timeout); //timeout
41
42    sb.AppendLine("");
43    sb.Append("\t\t" + returnType + " risultato_effettivo = " + classObjName + "." + methodName + "(");
44
45    for (var i = 0; i < parametri.Length; i++) {
46        var attributi = JavaClassAnalyzer.AnalizzaParametro(parametri[i], 0);
47        if (i == parametri.Length - 1) sb.Append(attributi.Item3);
48        else sb.Append(attributi.Item3 + ",";
49    }
50
51    sb.AppendLine("");
52
53
54    if (returnType.Equals("float", StringComparison.OrdinalIgnoreCase) || returnType.Equals("double", StringComparison.OrdinalIgnoreCase)) {
55        var precision = Interaction.InputBox("Il tipo di ritorno è " + returnType + ". Inserisci l'errore assoluto massimo accettato per il confronto tra risultato atteso e risultato ottenuto:", "Attenzione");
56        sb.AppendLine("\t\tassertTrue(\"Errore\", Math.abs(risultato_atteso-risultato_effettivo)<" + precision + ");");
57    }
58    else {
59        sb.AppendLine("\t\tassertEquals(risultato_atteso,risultato_effettivo);");
60    }
61
62    if (cbTimeout.Checked) sb.AppendLine("\t\t}); //timeout
63
64    sb.AppendLine("\t}");
65    sb.AppendLine("");
66
67    var path = Directory.GetCurrentDirectory() + "/Test" + className + ".java";
68    var sourceCode = sb.ToString();
69    File.WriteAllText(path, sourceCode);
70 }

```



Si parte dunque da uno scheletro di base, che comprende un metodo “init”, annotato con @BeforeEach e usato per inizializzare un oggetto della classe, e un metodo “TestMethod”, annotato con @ParameterizedTest, che conterrà il test vero e proprio.

Tale scheletro viene opportunamente modificato in base al contesto.

Viene chiesto se si vuole modificare esplicitamente lo scheletro di invocazione del costruttore, cosa necessaria nel caso in cui il costruttore della classe preveda dei parametri.

Se il tipo di ritorno del metodo è double oppure float, viene chiesto di inserire l’errore assoluto massimo accettato per il confronto tra risultato atteso e risultato ottenuto.

Nel caso in cui sia stato specificato un time-out e abilitato l’inserimento dello stesso, l’intero codice del metodo di test viene passato come secondo argomento di “assertTimeoutPreemptively”.

Questo è il risultato di quanto detto fino ad ora applicato all'esempio considerato:

```
1  import static org.junit.Assert.*;
2  import org.junit.jupiter.api.*;
3  import org.junit.jupiter.api.Test;
4  import org.junit.jupiter.params.ParameterizedTest;
5  import org.junit.jupiter.params.provider.CsvFileSource;
6  import org.junit.jupiter.params.provider.CsvSource;
7  import java.time.Duration;
8
9  class TestRomanNumeral {
10     RomanNumeral objRomanNumeral;
11     @BeforeEach
12     void init() {
13         objRomanNumeral = new RomanNumeral();
14         assertNotNull(objRomanNumeral);
15     }
16
17     @ParameterizedTest
18     @CsvFileSource(resources = "data.csv")
19     void TestMethod(int arabic,String risultato_atteso) {
20
21         String risultato_effettivo = objRomanNumeral.toRoman(arabic);
22         assertEquals(risultato_atteso,risultato_effettivo);
23     }
24 }
25
```

Testing Combinatorio

File Info

Nome classe: RomanNumeral

Lista Metodi:

public String toRoman(int arabic)

Inserire il valore di K:

2

Inserisci il percorso della cartella contenente i sorgenti:

C:\Users\amato\OneDrive\Desktop\ClassiDiTest\RomanNumeral

Inserisci il percorso della cartella che conterrà i risultati:

C:\Users\amato\OneDrive\Desktop\ClassiDiTest\RomanNumeral

Numero di threads per la minimizzazione:

2

Procedi

test (espresso in secondi)

Classe di test generata con successo!

OK

Per la generazione viene sfruttato CTWedge, quindi bisogna rispettare la sintassi di tale strumento durante la procedura.

Dopo la creazione della classe, si passa a valutare la copertura di codice ottenuta con la test suite originale, ossia non minimizzata.

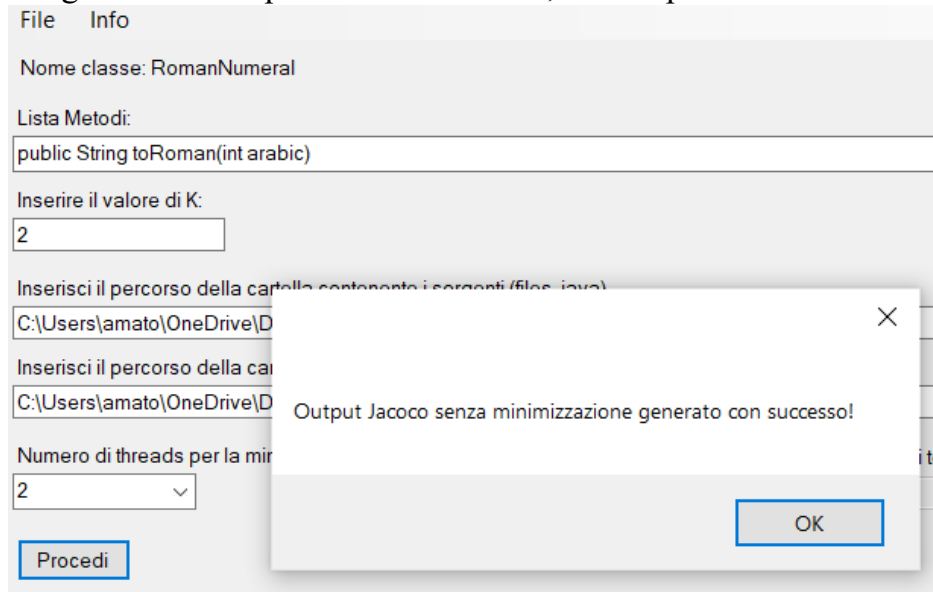
A tale scopo, viene eseguito il seguente metodo.

```
1 private void GeneraOutputJacocoNoMin() {
2     var className = lblNomeClasse.Text.Substring(13);
3     var testClassName = "Test" + className;
4     var srcPath = txtSrc.Text;
5     var binPath = txtBin.Text;
6     percorsoSrc = srcPath;
7     percorsoBin = binPath;
8     File.Delete(Directory.GetCurrentDirectory() + @"\jacoco.exec");
9     File.Delete(binPath + @"\jacoco.exec");
10    var currentUser = Environment.UserName;
11    File.Copy(Directory.GetCurrentDirectory() + @"\\" + testClassName + ".java",
12        srcPath + @"\\" + testClassName + ".java", true);
13    File.Copy(Directory.GetCurrentDirectory() + @"\data.csv", binPath + @"\data.csv", true);
14    var cartelle = GetClassPath(srcPath);
15    var command = @"/C javac -classpath C:\javatools\*;";
16    command += srcPath + @"\*";
17    foreach (var cart in cartelle) command += cart;
18
19    command += " " + srcPath + @"\*.java ";
20    command += "-d " + binPath;
21    new ShellCommand(command).ExecuteCommand();
22    command =
23        @"/C java -javaagent:C:\javatools\jacocoagent.jar -jar "C:\javatools\junit.jar" --classpath="C:\javatools\jacocoagent.jar;" +
24        binPath + @" " --scan-classpath";
25    new ShellCommand(command).ExecuteCommand();
26    var jacocoPath = Directory.GetCurrentDirectory();
27    File.Copy(jacocoPath + @"\jacoco.exec", binPath + @"\jacoco.exec", true);
28    command = @"/C java -jar "C:\javatools\jacococli.jar" report "" + binPath + @"\jacoco.exec" +
29        @"" --classfiles "" + binPath + @"" --sourcefiles "" + srcPath +
30        @"" --html "" + Directory.GetCurrentDirectory() + @"\coperturaNoMin" + @"";
31    new ShellCommand(command).ExecuteCommand();
32    command = @"/C java -jar "C:\javatools\jacococli.jar" report "" + binPath + @"\jacoco.exec" +
33        @"" --classfiles "" + binPath + @"" --sourcefiles "" + srcPath +
34        @"" --xml "" + Directory.GetCurrentDirectory() + @"\coperturaNoMin\jacoco.xml" + @"";
35    new ShellCommand(command).ExecuteCommand();
36 }
```

In sostanza:

1. La classe di test viene compilata, insieme alla classe oggetto di test e a tutte le classi da cui essa dipende.
2. La classe di test viene eseguita con Junit, e al tempo stesso Jacoco si occupa di misurare la copertura del codice

3. Jacoco genera sia il report in formato html, sia il report in formato xml.



Il report di copertura prodotto viene inserito nella cartella “coperturaNoMin”.

```
29.  ◆      if (arabic < 1)
30.          throw new NumberFormatException("Value of RomanNumeral must be positive.");
31.  ◆      if (arabic > 3999)
32.          throw new NumberFormatException("Value of RomanNumeral must be 3999 or less.");
33.      num = arabic;
34.
35.
36.      String roman = ""; // The roman numeral.
37.      int N = num;        // N represents the part of num that still has
38.                          // to be converted to Roman numeral representation.
39.  ◆      for (int i = 0; i < numbers.length; i++) {
40.  ◆          while (N >= numbers[i]) {
41.              roman += letters[i];
42.              N -= numbers[i];
43.          }
44.      }
45.      return roman;
46.
47.  }
```

In questo caso, si è ottenuta una copertura del 100% per il metodo testato (sia per le istruzioni che per i branches).

Si procede poi con la minimizzazione della test suite.

Si tenga presente quanto segue:

- Ciascun caso di test viene eseguito separatamente e ne viene valutata la copertura (attraverso il parsing del report in formato xml), secondo quanto appena visto con la test suite di partenza. Si considera la copertura delle linee di codice, non quella dei branches. Di conseguenza si ha che, una linea di codice sorgente con valore di

copertura pari ad 1 o pari ad un numero frazionario, viene considerata coperta. Una linea di codice sorgente con valore di copertura pari a 0 viene considerata non coperta.

- Valutata la copertura delle LOC per ciascun caso di test, viene creata la matrice di copertura da usare nel processo di minimizzazione. Questa matrice riporta sulle righe i casi di test e sulle colonne gli indici delle linee di codice.

Quanto ora illustrato viene gestito dal seguente metodo.

```
1 private List < TestCase > CreaMatrice() {  
2     var testSuite = new List < TestCase > ();  
3  
4     var className = lblNomeClasse.Text.Substring(13) + ".java";  
5     var methodName = JavaClassAnalyzer.GetMethodName(cbMetodi.SelectedItem.ToString());  
6  
7     var csvlines = File.ReadAllLines(Directory.GetCurrentDirectory() + @"\data.csv");  
8     File.WriteAllLines(Directory.GetCurrentDirectory() + @"\dataNoMin.csv", csvlines);  
9  
10    var count = 0;  
11  
12    var options = new ParallelOptions {  
13        MaxDegreeOfParallelism = Convert.ToInt32(cbParallelismo.SelectedItem.ToString())  
14    };  
15  
16    Parallel.For(0, csvlines.Length, options, index =>{  
17        var csvline = csvlines[index];  
18        var id = index;  
19        var valore = csvline;  
20        GeneraCoverageTestCase(csvline, id);  
21        var jacocoXML = File.ReadAllText(Directory.GetCurrentDirectory() + @"\temp\" + id + @"\jacoco.xml ");  
22        var lines = JacocoReportAnalyzer.ParseXml(jacocoXML, className, methodName);  
23        var tc = new TestCase(id, valore, lines);  
24        testSuite.Add(tc);  
25    });  
26  
27  
28  
29    return testSuite;  
30 }
```

La minimizzazione viene effettuata dal metodo “MinimizzaTestSuite”, che a sua volta sfrutta il metodo “Minimizza” della classe “Minimizzazione”.

```
1 public static List<int> Minimizza(int[, ] matrice) {
2     var numRighe = matrice.GetLength(0);
3     var numColonne = matrice.GetLength(1);
4     var soluzione = new List<int> ();
5     var indiciRighe = new List<int> (Enumerable.Range(0, numRighe).ToArray());
6     var indiciColonne = new List<int> (Enumerable.Range(0, numColonne).ToArray());
7     var stop = false;
8     for (var i = 0; i < numRighe; i++)
9     for (var j = i; j < numRighe; j++) {
10         if (i == j) continue;
11         var equal = true;
12         for (var k = 0; k < numColonne; k++)
13             if (matrice[i, k] != matrice[j, k]) equal = false;
14         if (equal) indiciRighe.Remove(j);
15     }
16     if (indiciRighe.Count == 1) {
17         soluzione.Add(indiciRighe.First());
18         stop = true;
19     }
20     while (!stop) {
21         if (indiciRighe.Count == 0 || indiciColonne.Count == 0) {
22             stop = true;
23             continue;
24         }
25         var essentialFound = false;
26         var essentialRowId = 0;
27         foreach (var t in indiciColonne) {
28             var onescount = 0;
29             var temp = new List<int> ();
30             foreach (var t1 in indiciRighe.Where(t1 => matrice[t1, t] == 1)) {
31                 onescount++;
32                 temp.Add(t1);
33             }
34             if (onescount != 1) continue;
35             essentialFound = true; //riga essenziale trovata
36             essentialRowId = temp[0];
37             break;
38         }
39         var tempCol = indiciColonne.ToArray().ToList();
40         if (essentialFound) {
41             foreach (var t in indiciColonne.Where(t => matrice[essentialRowId, t] == 1)) tempCol.Remove(t);
42             indiciColonne = tempCol.ToArray().ToList();
43             indiciRighe.Remove(essentialRowId);
44             soluzione.Add(essentialRowId);
45         }
46         var rigaeliminata = false;
47         var lstRighe = (from indiceRiga in indiciRighe
48             let id = indiceRiga
49             let colonneCoperte = indiciColonne.Where(indiceColonna => matrice[indiceRiga, indiceColonna] == 1).ToList()
50             select new RigaMatrice(indiceRiga, colonneCoperte)).ToList();
51         foreach (var rigaRoot in lstRighe) {
52             var lstRigheTemp = lstRighe.ToArray().ToList();
53             lstRigheTemp.Remove(lstRighe.Single(r => r.Id == rigaRoot.Id));
54             foreach (var rigaChild in lstRigheTemp.Where(rigaChild => ContainsAllItems(rigaRoot.IDcolonneCoperte, rigaChild.IDcolonneCoperte) && rigaRoot.IDcolonneCoperte.Count > rigaChild.IDcolonneCoperte.Count))
55                 indiciRighe.Remove(rigaChild.Id);
56             rigaeliminata = true;
57         }
58     }
59     var colonnaaeliminata = false;
60     var lstColonne = (from indiceColonna in indiciColonne
61         let id = indiceColonna
62         let righeCoperte = indiciRighe.Where(indiceRiga => matrice[indiceRiga, indiceColonna] == 1).ToList()
63         select new ColonnaMatrice(indiceColonna, righeCoperte)).ToList();
64     foreach (var colonnaRoot in lstColonne) {
65         var lstColonneTemp = lstColonne.ToArray().ToList();
66         lstColonneTemp.Remove(lstColonne.Single(r => r.Id == colonnaRoot.Id));
67         foreach (var colonnaChild in lstColonneTemp.Where(colonnaChild => ContainsAllItems(colonnaRoot.IDrigheCoperte, colonnaChild.IDrigheCoperte) && colonnaRoot.IDrigheCoperte.Count > colonnaChild.IDrigheCoperte.Count))
68             indiciColonne.Remove(colonnaRoot.Id);
69         colonnaaeliminata = true;
70     }
71 }
72 if (essentialFound || rigaeliminata || colonnaaeliminata) continue; {
73     var obj = lstRighe.Where(r => r.IDcolonneCoperte.Count == lstRighe.Max(m => m.IDcolonneCoperte.Count));
74     var riga = obj.ElementAt(0);
75     var rowId = riga.Id;
76     foreach (var idColonna in riga.IDcolonneCoperte) indiciColonne.Remove(idColonna);
77     indiciRighe.Remove(rowId);
78     soluzione.Add(rowId);
79 }
80 }
81 return soluzione;
82 }
```

La minimizzazione viene effettuata in questo modo:

1. Si eliminano dalla matrice di partenza tutte le righe uguali.
2. Fin quando la matrice non diventa completamente vuota, in successione si applicano:
3. Criterio di essenzialità
4. Criterio di dominanza delle righe
5. Criterio di dominanza delle colonne



6. Qualora in una iterazione non si riesca ad applicare nessuno dei tre seguenti criteri, viene aggiunta all'insieme delle soluzioni la riga che, tra tutte quelle ancora presenti nella matrice, è quella con il numero maggiore di 1.

Al termine della minimizzazione viene mostrato un messaggio di riepilogo.

×

-----  
La test suite non minimizzata ha 9 casi di test  
-----

La test suite minimizzata ha 3 casi di test  
-----

Tempo totale impiegato per la minimizzazione: 11,3693891 secondi  
-----

"coperturaNoMin" è la cartella contenente il report di copertura prodotto da Jacoco prima della minimizzazione.

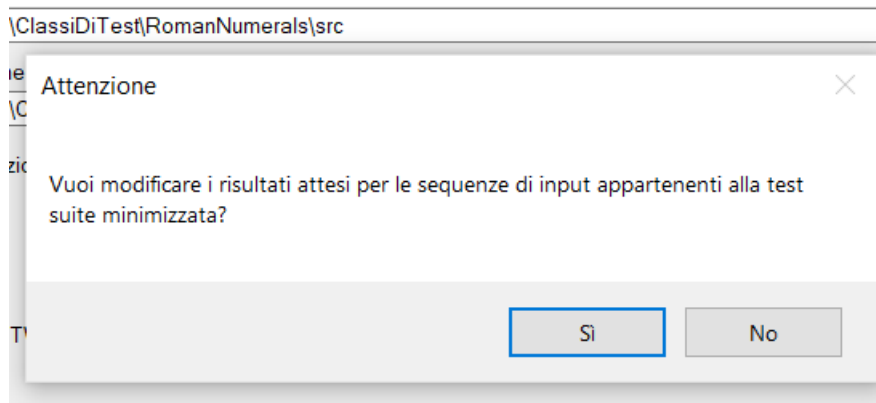
"dataNoMin.csv" è il file contenente l'insieme degli input prima della minimizzazione.

-----  
"coperturaConMin" è la cartella contenente il report di copertura prodotto da Jacoco dopo la minimizzazione.

"dataMin.csv" è il file contenente l'insieme degli input dopo la minimizzazione.  
-----

OK

Viene poi chiesto se si vuole modificare i risultati attesi per i casi di test presenti nella test suite minimizzata.



Test suite non minimizzata:

```
1 -5000, "prova"
2 -500, "prova"
3 -50, "prova"
4 -5, "prova"
5 0, "prova"
6 5, "prova"
7 50, "prova"
8 500, "prova"
9 5000, "prova"
10
```

Test suite minimizzata:

```
1 -5000, "prova"
2 5000, "prova"
3 5, "prova"
4
```

```
25.     public String toRoman(int arabic) {
26.         // Constructor. Creates the Roman number with the int value specified
27.         // by the parameter. Throws a NumberFormatException if arabic is
28.         // not in the range 1 to 3999 inclusive.
29.         if (arabic < 1)
30.             throw new NumberFormatException("Value of RomanNumeral must be positive.");
31.         if (arabic > 3999)
32.             throw new NumberFormatException("Value of RomanNumeral must be 3999 or less.");
33.         num = arabic;
34.
35.
36.         String roman = ""; // The roman numeral.
37.         int N = num;        // N represents the part of num that still has
38.                             // to be converted to Roman numeral representation.
39.         for (int i = 0; i < numbers.length; i++) {
40.             while (N >= numbers[i]) {
41.                 roman += letters[i];
42.                 N -= numbers[i];
43.             }
44.         }
45.         return roman;
46.
47.     }
```

In questo caso, anche la test suite minimizzata garantisce una copertura totale sia delle linee di codice sia dei branches. In generale, può accadere, dato l'obiettivo di copertura fissato (LOC), che la coverage dei branches nella test suite minimizzata sia inferiore alla coverage dei branches nella test suite non minimizzata.

Per mettere alla prova la robustezza dello strumento prodotto nella sua interezza, l'ho utilizzato su classi diverse, più o meno complesse, eventualmente dipendenti da altre classi, realizzate da sviluppatori diversi, secondo stili diversi.

In tutti i casi considerati, sia l'analizzatore statico, sia l'intero processo di generazione della test suite, valutazione della copertura, minimizzazione e nuova valutazione della copertura si sono comportati correttamente.

Di seguito viene riportata, a titolo di esempio per le funzionalità dello strumento non messe in evidenza dall'esempio precedente, la classe di test generata per testare il metodo "eval" della classe "Expr", presente nel repository.

```
1  import static org.junit.Assert.*;
2  import org.junit.jupiter.api.*;
3  import org.junit.jupiter.api.Test;
4  import org.junit.jupiter.params.ParameterizedTest;
5  import org.junit.jupiter.params.provider.CsvFileSource;
6  import org.junit.jupiter.params.provider.CsvSource;
7  import java.time.Duration;
8
9  class TestExpr {
10     Expr objExpr;
11     @BeforeEach
12     void init() {
13         objExpr = new Expr();
14         assertNotNull(objExpr);
15     }
16
17     @ParameterizedTest
18     @CsvFileSource(resources = "data.csv")
19     void TestMethod(String definition, double variable, double risultato_atteso) {
20         org.junit.jupiter.api.Assertions.assertTimeoutPreemptively(Duration.ofSeconds(10), () -> {
21             double risultato_effettivo = objExpr.eval(definition, variable);
22             assertTrue("Errore", Math.abs(risultato_atteso - risultato_effettivo) < 1e-8);
23         });
24     }
25 }
26
```

Qui è possibile osservare l'utilizzo di un time-out per l'esecuzione di ciascun caso di test e, inoltre, il confronto del risultato atteso con il risultato effettivo andando a considerare la valutazione dell'errore assoluto e la verifica che tale errore sia inferiore ad un certo valore.

## 2.4) Prestazioni

In questa sezione della tesina vengono effettuate alcune considerazioni circa le prestazioni dello strumento.

Il collo di bottiglia in questo senso è rappresentato dall'esecuzione dei casi di test presi singolarmente e dalla corrispondente valutazione della copertura. Questa procedura, detto n il numero di casi di test della test suite sottoposta a minimizzazione, richiede di eseguire n volte i seguenti passi:

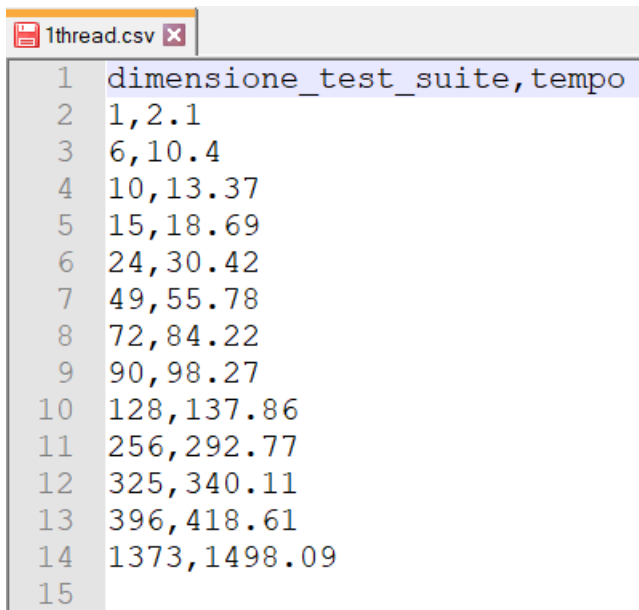
1. Utilizzare Junit per eseguire la classe di test, e al tempo stesso impostare Jacoco come "java agent" per misurare la copertura delle linee di codice.
2. Fare il parsing del report xml ottenuto per discernere le linee di codice coperte da quelle non coperte con tale caso di test.

L'operazione più onerosa è la prima e grava principalmente sulla CPU. Tra l'altro, bisogna tener conto che il tempo necessario a completare suddetta operazione è ovviamente dipendente anche dal tipo di metodo che si va a testare e dalla sua complessità computazionale.

La minimizzazione vera e propria, ossia quella applicata alla matrice di copertura, impiega un tempo trascurabile rispetto al resto.

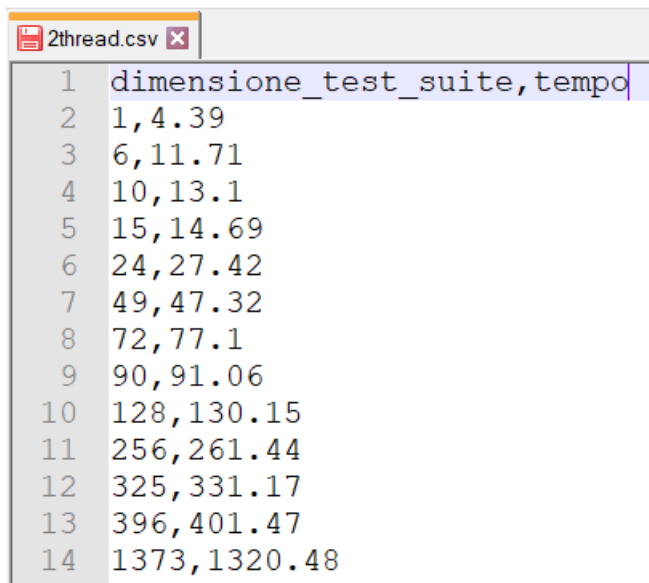
Ho introdotto la possibilità di sfruttare il multithreading per cercare di migliorare le prestazioni. Vengono ora mostrate delle immagini che consentono di visualizzare come varia il tempo necessario ad eseguire l'intera procedura di minimizzazione al variare della dimensione della test suite. Ho preso come riferimento il metodo "knightBFS" della classe "ChessKnight", disponibile nel repository di progetto.

Tempi di esecuzione in funzione del numero di casi di test con minimizzazione a singolo thread:



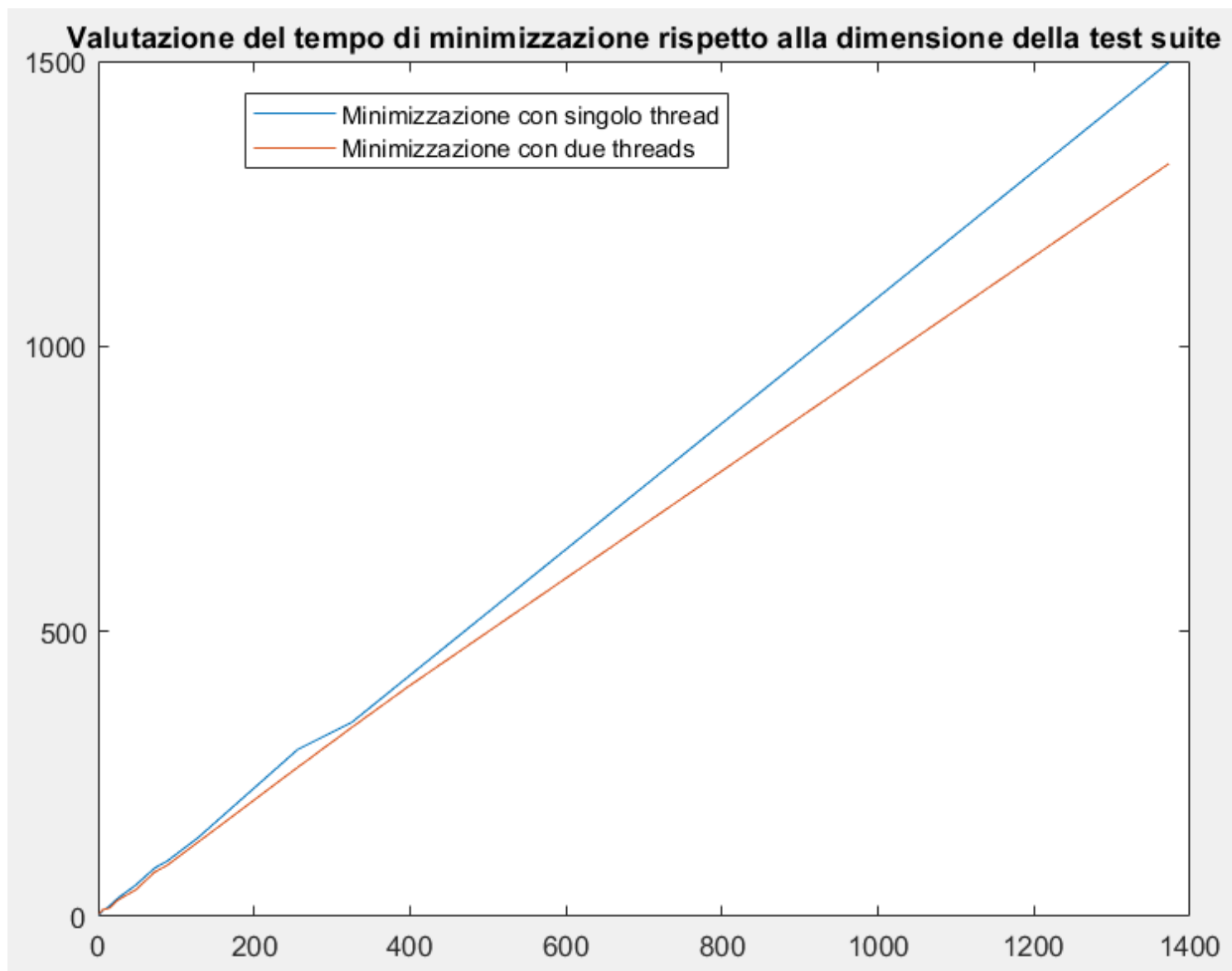
	dimensione_test_suite,tempo
1	1,2.1
2	6,10.4
3	10,13.37
4	15,18.69
5	24,30.42
6	49,55.78
7	72,84.22
8	90,98.27
9	128,137.86
10	256,292.77
11	325,340.11
12	396,418.61
13	1373,1498.09
14	
15	

Tempi di esecuzione in funzione del numero di casi di test con minimizzazione a due threads:



	dimensione_test_suite,tempo
1	1,4.39
2	6,11.71
3	10,13.1
4	15,14.69
5	24,27.42
6	49,47.32
7	72,77.1
8	90,91.06
9	128,130.15
10	256,261.44
11	325,331.17
12	396,401.47
13	1373,1320.48
14	
15	

Per ottenere tempi più veritieri, per ogni dimensione della test suite, la procedura è stata eseguita tre distinte volte prendendo come valore finale la media dei 3. Questo è stato fatto sia nel caso di singolo thread sia nel caso di due threads.



Il miglioramento derivante dal multithreading lo si ottiene soprattutto all'aumentare del numero di casi di test che costituiscono la test suite.

La macchina su cui ho eseguito i test presenta le seguenti specifiche:

- CPU i5-8400
- Intel SSD 660p
- 8 GB RAM

## 2.5) Limitazioni

- I parametri di ingresso del metodo da testare devono essere tutti primitivi (byte, short, int, long, float, double, boolean, char) o tipi riconducibili a quelli primitivi. Fa eccezione il tipo non primitivo “String”, anch’esso supportato.
- Se il costruttore della classe prevede uno o più parametri, il programma chiede esplicitamente all’utente di inserirli. Del resto, dai valori di tali parametri potrebbe dipendere il risultato di tutta la successiva esecuzione e non ho saputo gestire altrimenti questa situazione.
- La minimizzazione non è “perfetta”. Ci sono alcuni casi in cui, durante una qualche specifica iterazione del processo di minimizzazione, l’applicazione non riesce ad utilizzare nessuno dei tre criteri possibili (essenzialità, dominanza delle righe, dominanza delle colonne). In questo caso, da progetto, l’applicazione aggiunge alla soluzione la riga che, tra tutte quelle ancora presenti nella matrice di copertura, è quella con il numero maggiore di 1. Non è detto che questa sia sempre la scelta migliore, di conseguenza non è garantito sempre che, nel passaggio da test suite non minimizzata a test suite minimizzata, la copertura complessiva delle linee di codice sia la stessa.

### 3) Il problema dell'oracolo

Anche con algoritmi efficienti per produrre matrici di copertura, il problema dell'oracolo resta: si è in grado di generare automaticamente test combinatori, ma non si è in grado di fare altrettanto con i corrispondenti oracoli. Questi andrebbero inseriti a mano in tutti i casi di test, aumentando i costi del testing fino a renderlo non più conveniente.

Soluzioni sono possibili in contesti specifici:

- **Crash testing:** l'approccio più semplice e meno costoso consiste nell'eseguire tutti i casi di test prodotti automaticamente per verificare se una qualche determinata combinazione dei valori di input provoca un crash nell'applicazione oggetto di test o altri malfunzionamenti facilmente rilevabili. È poi possibile analizzare i dumps della memoria per determinare la causa del crash. Questo è più o meno quello che accade in alcuni tipi di "fuzz testing", che mandano valori casuali in ingresso al sistema sotto test. Va notato che sebbene il testing casuale puro riesca a coprire generalmente un'alta percentuale di combinazioni di t-way, la copertura del 100% delle combinazioni di solito richiede un set di test casuali molto più grande di un array di copertura.
- **Asserzioni:** una tecnica sempre più diffusa consiste nell'inserire affermazioni all'interno del codice per verificare la correttezza delle relazioni tra i dati, ad esempio, come precondizioni, postcondizioni o controlli di coerenza. Strumenti come il Java Modeling Language (JML) possono essere utilizzati per introdurre asserzioni molto complesse, incorporando efficacemente una specifica formale all'interno del codice. Le asserzioni rappresentano una forma eseguibile della specifica, fornendo così un oracolo per la fase di test. Con le asserzioni incorporate nel codice, esercitare il sistema software oggetto di test con tutte le combinazioni t-way può fornire una ragionevole garanzia che il codice funzioni correttamente su una gamma molto ampia di input. Questo approccio è stato utilizzato con successo per testare le smart card, con asserzioni JML incorporate che fungono da oracolo per i test combinatori. I risultati hanno mostrato che l'80-90% dei guasti può essere trovato in questo modo.
- **Model based test generation:** si utilizza un modello matematico del sistema sotto test e un simulatore o un verificatore di modelli per generare i risultati attesi per ogni combinazione di valori input. Se è possibile utilizzare un simulatore, i risultati attesi



possono essere generati direttamente dalla simulazione. I verificatori di modelli possono invece essere utilizzati, oltre che per generare casi di test, anche per dimostrare proprietà del sistema. Concettualmente, un verificatore di modelli si occupa di esplorare tutti gli stati di un modello di sistema per determinare se una proprietà rivendicata in un'istruzione di specifica è vera. Ciò che rende un verificatore di modelli particolarmente interessante è che se l'affermazione è falsa, il verificatore di modelli non solo lo riporta, ma fornisce anche un controesempio che mostra in maniera pratica la non veridicità della proprietà affermata. In effetti, questo è un test case completo, ovvero un insieme di valori di parametri e risultati attesi. È quindi semplice mappare questi valori in casi di test completi nella sintassi necessaria per il sistema oggetto di test.

- Testing di regressione: L'oracolo di ogni test automaticamente generato coincide con il risultato ottenuto eseguendo lo stesso test su di un altro sistema, rispetto al quale si sta valutando la non regressione.

## 4) Assertion based testing

Il testing automatico integrato è una caratteristica comune dei circuiti integrati in cui funzioni hardware e software aggiuntive consentono il controllo della correttezza delle operazioni svolte dal sistema mentre il sistema stesso è in funzione, in contrasto con l'uso di test applicati esternamente. Un concetto simile, quello delle asserzioni incorporate, è stato utilizzato per decenni nel mondo dell'ingegneria del software e i progressi nei linguaggi di programmazione e nelle prestazioni dei calcolatori hanno reso questo metodo ancora più utile e pratico. Può essere particolarmente efficace se abbinato alla tecnica di testing combinatoriale. Se il sistema è completamente esercitato con i test t-way e le asserzioni sono complete, si può avere una ragionevole certezza che il funzionamento del sistema sarà corretto per combinazioni fino a t-way di valori di input. Oltre alle funzionalità standard dei linguaggi di programmazione, è stata sviluppata una varietà di strumenti specializzati per rendere questo approccio più semplice ed efficace. Molti linguaggi di programmazione includono una funzione di asserzione che consente al programmatore di specificare proprietà che si presume siano vere in un punto particolare del programma.

Ad esempio, una funzione che include una divisione in cui un particolare parametro  $x$  verrà utilizzato come divisore potrebbe richiedere che questo parametro non sia mai zero. Questa funzione potrebbe includere l'istruzione C# `Assert.IsTrue(x != 0);` come prima istruzione eseguita. Si noti che un'asserzione non è la stessa cosa di un controllo di validità dell'input: quest'ultimo emette un messaggio di errore se l'input non è accettabile. Con un numero sufficiente di asserzioni derivate da una specifica, il programma può ottenere una proprietà di "auto-verifica".

Le asserzioni possono essere utili come una sorta di prova incorporata di proprietà importanti: se le asserzioni sono vere per tutte le esecuzioni del programma, ci si può aspettare che le proprietà codificate nelle asserzioni siano valide per il sistema considerato. Quindi, se le asserzioni formano una catena logica che implica una dichiarazione formale delle proprietà del programma, è possibile dimostrare la correttezza del programma rispetto a queste proprietà.

Si può trarre vantaggio da questo schema nei test combinatori dimostrando che le asserzioni valgono per tutte le combinazioni di input t-way. Ovviamente, questo tipo di verifica non corrisponde ad una dimostrazione di correttezza, quest'ultimo è un concetto più forte.

La qualità dei test basati sulle asserzioni con metodi combinatori dipende dalla forza delle asserzioni, oltre che dalla forza dell'interazione t-way.

Strumenti come JML per Java possono essere utilizzati per introdurre asserzioni complesse nel sorgente, incorporando efficacemente una specifica all'interno del codice.

#### 4.1) Costo

Quello delle asserzioni è in generale un approccio conveniente all'automazione dei test poiché le asserzioni vengono inserite direttamente attraverso il codice. Tipicamente, l'uso delle asserzioni è correlato a tassi di errore ridotti, ma la reale efficacia di questa metodologia dipende molto anche dal tipo di sistema che si ha davanti e dall'uso che se ne fa.

In molte applicazioni, le asserzioni vengono utilizzate in modo molto semplice, ad esempio per garantire che puntatori nulli non vengano passati a una funzione che li utilizzerà o che i parametri che possono essere usati come divisori siano diversi da zero.

Asserzioni più complesse possono fornire una maggiore sicurezza, ma esistono dei limiti alla loro efficacia. Ad esempio, le invarianti (proprietà che dovrebbero essere mantenute durante tutta l'esecuzione) generalmente non possono essere garantite senza inserire un'asserzione per ogni riga di codice.

Poiché le asserzioni devono essere eseguite per mostrare la presenza o l'assenza di una proprietà a un certo punto, è possibile che errori che impediscono il raggiungimento di un'asserzione non vengano rilevati.

Linguaggi specializzati per il controllo delle asserzioni come JML possono alleviare molti di questi problemi fornendo direttive del preprocessore per generare codice che implementa un controllo così complesso senza rendere il programma difficile da leggere.

## 5) Model based testing

A prima vista, il problema dell'oracolo per sistemi con calcoli e condizioni complessi può sembrare quasi irrisolvibile: come possiamo verificarne la correttezza senza implementare software altrettanto complessi, la cui correttezza deve anche essere verificata, portando ad una serie infinita di esercizi di verifica?

Le asserzioni possono aiutare ma non sempre sono sufficienti.

Uno dei modi più efficaci per produrre oracoli per i casi test consiste nell'utilizzare un modello del sistema sottoposto a test e generare test completi, inclusi sia i dati di input che i risultati attesi direttamente dal modello.

Usiamo il termine modello nello stesso modo in cui verrebbe utilizzato in altri rami dell'ingegneria: il modello incorpora aspetti del sistema che siamo interessati a studiare.

I modelli nel campo del testing di sistemi software possono essere utilizzati per verificare i calcoli o le prestazioni, ad esempio, ma non altre proprietà come la posizione di un particolare valore numerico su uno schermo (se includesse tutti i dettagli, il modello sarebbe equivalente al sistema stesso).

Per applicare il testing combinatoriale in questo contesto, vale quanto segue:

Strumenti utilizzati:

1. ACTS covering array generator
2. NuSMV, una variante del verificatore di modelli SMV ( symbolic model verifier )
3. È necessaria una specifica formale o semi-formale del sistema sotto test. Può essere sotto forma di una specifica logica formale, ma è possibile utilizzare anche tabelle di transizione di stato, tabelle decisionali, pseudocodice o linguaggio naturale strutturato, purché le regole non siano ambigue.

Task da svolgere:

1. Utilizzando ACTS, si costruisce una serie di test che copriranno tutte le combinazioni t-way dei valori dei parametri. La matrice di copertura specifica i dati di test, ogni riga della matrice può essere considerata come un insieme di valori di parametri per un singolo test.

2. Determinare quale output dovrebbe essere prodotto dal sistema sotto test per ogni insieme di valori dei parametri di input. Gli output provenienti da ACTS saranno incorporati nelle specifiche di SVM che possono essere processate dal verificatore di modelli NuSVM.

### 5.1) Costo

Il modello o la specifica formale possono essere costosi da produrre, ma una volta disponibili, è possibile generare, eseguire e analizzare un gran numero di test senza l'intervento umano. Questo può rappresentare un enorme risparmio sui costi, poiché il test di solito richiede il 50% o più del budget di sviluppo del software. Una proprietà interessante dell'approccio di model checking descritto è che la generazione di casi di test può essere eseguita in parallelo.

Per ogni riga di test della matrice di copertura, si esegue il model checker per determinare i risultati attesi per gli input forniti da quella riga e le diverse esecuzioni del model checker sono indipendenti l'una dall'altra.

Con l'ampia disponibilità di sistemi cloud, la generazione di test può essere eseguita molto rapidamente. Nella maggior parte dei casi, anche l'esecuzione dei test può essere effettuata in parallelo, sebbene in alcuni casi si potrebbe essere limitati da problemi pratici come la disponibilità di hardware specializzato.

## 6) Riferimenti

1. D.R. Wallace, D.R. Kuhn, Failure Modes in Medical Device Software: an Analysis of 15 Years of Recall Data, International Journal of Reliability, Quality, and Safety Engineering, Vol. 8, No. 4, 2001.
2. Combinatorial Testing: Theory and Practice: D. Richard Kuhn, Renee Bryce, Feng Duan, Laleh Sh. Ghandehari, Yu Lei, Raghu N. Kacker
3. Introduction to Combinatorial Testing: D. Richard Kuhn, Raghu N. Kacker, Yu Lei