UNIVERSITÀ
DEGLI STUDI
DI BERGAMO

Dipartimento
di Ingegneria Gestionale,
dell'Informazione e della Produzione

CTWedge

Combinatorial Testing Web Editor and Generator

Contatti:
Prof. Angelo Gargantini – angelo.gargantini@unibg.it
Dott. Andrea Bombarda – andrea.bombarda@unibg.it
Dott. Marco Radavelli

CTWedge in breve

- E’ un linguaggio per problemi CIT, con una semantica formale ed una grammatica definita con Xtext
- Offre un editor testuale integrato nell’IDE Eclipse
- Offre un insieme di tools per importare/esportare modelli CIT, e per generare test suites (utilizzando tool esterni)
- E’ un framework basato sull’Eclipse Modeling Framework (EMF)
- Integra un insieme di benchmarks ed esempi (Phone, Banking, HealthcareSystem, Concurrency, ...)

Può essere utilizzato in due modalità:

- Web-based: https://foselab.unibg.it/ctwedge
- Integrato in Eclipse: https://raw.githubusercontent.com/fmselab/ctwedge/master/ctwedge.updatesite

Linguaggio CTWedge

- CTWedge permette la definizione, all'interno dei modelli, di **parametri**, ciascuno dei quali con:
  - Nome
  - Tipo
    - Boolean
    - Ranges (ovvero intervalli interi), ad esempio [0..5]
    - Enumerativi, che possono essere liste di valori {valore1, valore2, valore3}

Sui vari parametri è poi possibile definire dei **constraint**, espressi in logica proposizionale (con i classici operatori logici), con la possibilità di usare uguaglianze ed operatori aritmetici.

I vari test che non soddisfano i constraint sono considerati non validi e non richiedono di essere forniti in output.

Esempio di modello e grammatica

```
/*
 * This is an example model
 */
Model Phone
 Parameters:
    emailViewer : Boolean
    textLines:  [ 25 .. 30 ]
    display : {16MC, 8MC, BW}

Constraints:
  # emailViewer => textLines > 28 #
```

GRAMMAR:

CitModel:
'Model' name=ID
//list of parameters
'Parameters' ':'
(parameters+=Parameter)+
// constraints
('Constraints' ':'
(constraints+=Constraint)+)?;

Visualizzazione delle test suites

- Le test suites vengono mostrate direttamente all'interno del Browser (o di Eclipse) e possono essere esportate in CSV

Generated Test Suite

Generated using ACTS strength=2 ignoringConstraints: false

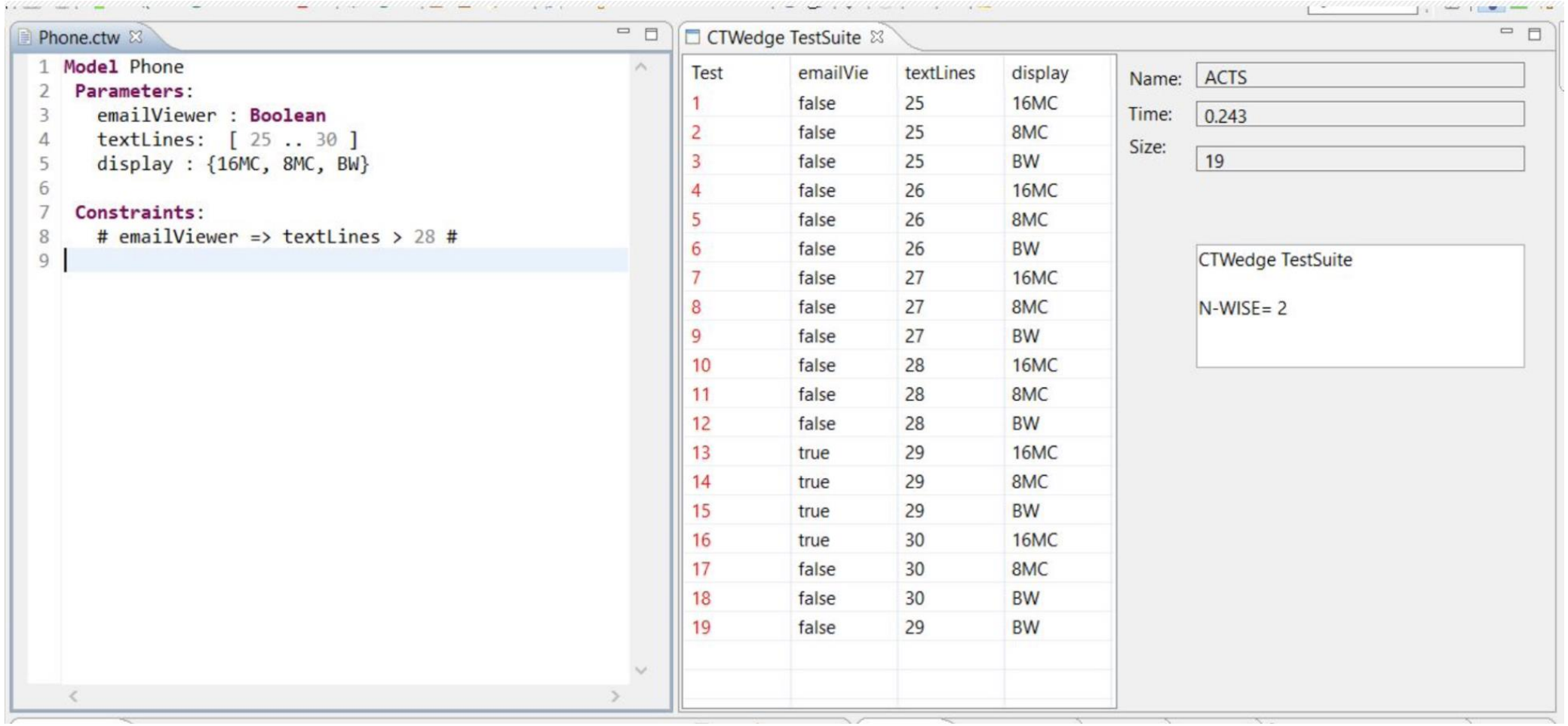
Download CSV Separator: Semicolon (;) - DEFAUl

#	emailViewer	textLines	display
1	false	25	16MC
2	false	25	8MC
3	false	25	BW
4	false	26	16MC
5	false	26	8MC
6	false	26	BW
7	false	27	16MC
8	false	27	8MC
9	false	27	BW
10	false	28	16MC
11	false	28	8MC
12	false	28	BW
13	true	29	16MC

Utilizzo di CTWedge in Eclipse (1)

- Per utilizzare CTWedge in Eclipse è necessario (dopo aver fatto l'installazione con l'Update-Site) seguire questi passaggi:
    - Creare un progetto vuoto
    - Creare un file, all'interno progetto, con estensione .ctw che rappresenterà il modello
    - Completare il modello come desiderato
    - Click-Destro sul file e selezionare Run As -> Run configurations -> CTWedge generator -> Run

Utilizzo di CTWedge in Eclipse (2)



Esercizi

Esercizio 1 – Equazioni di secondo grado

Equazioni di secondo grado

Dato un metodo che restituisce il numero di soluzioni per una equazione di secondo grado:

static public int numSolutions(int a, int b, int c){....}

- implementa il metodo
- Fai input domain modeling: modella l'insieme degli input in modo da coprirli secondo diversi criteri di copertura. Partiziona l'insieme in triple in modo:
  - INTERFACE-BASED: considera una partizione per a, per b e per c e prendi dei valori significativi (però senza considerare il problema) per ognuna delle partizioni.
  - FUNCTIONALITY-BASED: dividi il dominio in partizioni a seconda del risultato che ti aspetti dal metodo che devi testare. Prendi un rappresentate per partizione e scrivi i casi di test in JUnit.
- Applica il combinatorial testing al precedente modello (solo interface-based) usando CTWedge, e sulla base della test suite restituita (provare con strength 2 e 3), scrivi i casi di test in Junit per ogni caso di test.

Esercizi

Esercizio 2 – Rush Hour

Rush Hour

Nel RushHour una griglia 6x6 contiene 6 macchine numerate da 1 a 6 (ogni macchina per semplicità ha dimensione 1x1). L'obiettivo è portare - tramite spostamenti in celle libere adiacenti - la macchina numero 1 (rossa) all'uscita della griglia, che corrisponde alla cella con indici (3,6) (o 2,5 se le coordinate iniziano da 0) indicata dallo sfondo scuro. La configurazione iniziale del gioco è quella indicata.

Il programma Java ha un metodo **moveCar(int row, int col, int dir)** che muove l'auto sulla griglia (1-Nord, 2-Est, 3-Sud, 4-Ovest) solo se la cella di destinazione è all'interno della griglia e libera da auto. C'è anche il metodo **redCarAtExit** che dice se la macchina rossa ha raggiunto l'uscita.

	1	2	3	4	5	6
1				6		
2						2
3			1			3
4						4
5		5				
6						

Applica il combinatorial testing al sistema semplificato con solo 3 auto in modo di avere tutte le configurazioni possibili. Prova a generare la copertura pairwise delle posizioni. Includi i necessari constraints. Se riesci aggiungi anche una variabile che indica se l'auto rossa è nella posizione di uscita.

Esercizi

Esercizio 3 – Magazzino

Magazzino (1)

Un magazzino contiene 5 tipi di prodotti, e al massimo 100 unità di ogni prodotto. Si possono aggiungere al massimo 10 unità alla volta per uno stesso prodotto, e solo fino a raggiungere il livello massimo.

In Java scrivi una classe **Magazzino** che implementa (tramite un array di interi) il sistema sopra, con 3 metodi:

- **boolean insert(int productIndex, int addQuantity)** : aggiunge il valore addQuantity (al massimo 10 prodotti) al prodotto identificato da productIndex, e ritorna true se l'aggiunta viene eseguita, false altrimenti. L'aggiunta non viene eseguita se l'indice di prodotto è errato, o se la quantità aggiunta non è corretta (non compresa tra 1 e 10), o se la nuova quantità che si otterrebbe con l'aggiunta supera le 100 unita.
- **boolean isFull(int productIndex)** : dice se un certo prodotto ha raggiunto il massimo
- **boolean isFull()** : restituisce se il magazzino è completamente pieno (tutti i prodotti sono la massimo)

Magazzino (2)

Rappresenta in combinatorial testing la chiamata del metodo **insert**, considerando come variabili del problema: **productIndex** e **addQuantity** (che sono i parametri del metodo), **returnedValue** (che è il valore ritornato), **nproductsOld** (il numero di prodotti prima della chiamata), **nproductsNew** (il numero di prodotti dopo la chiamata).

Introduci i domini opportuni e i vincoli tra i parametri. In questo modo hai anche l'oracolo.

Applica il combinatorial testing. Prova a generare la copertura pairwise.

Infine, prova a tradurre una riga dal tuo test in un test Junit per la classe Magazzino.

Esercizi

Esercizio 4 – Tipo triangolo

Tipo triangolo

Dato un metodo che restituisce il tipo di un triangolo, sotto forma di stringa (ISOSCELE, SCALENO, EQUILATERO) , considerando le tre lunghezze dei lati

static public String triangleType(int lato1, int lato2, int lato3){....}

- implementa il metodo
- Fai input domain modeling: modella l'insieme degli input in modo da coprirli secondo diversi criteri di copertura. Partiziona l'insieme in triple in modo:
  - INTERFACE-BASED: considera una partizione per a, per b e per c e prendi dei valori significativi (però senza considerare il problema) per ognuna delle partizioni.
  - FUNCTIONALITY-BASED: dividi il dominio in partizioni a seconda del risultato che ti aspetti dal metodo che devi testare. Prendi un rappresentate per partizione e scrivi i casi di test in JUnit.
- Applica il combinatorial testing al precedente modello usando CTWedge, e sulla base della test suite restituita (provare con t = 2), scrivi i casi di test in Junit per ogni caso di test.