# Testare bazată pe modele pentru sisteme reactive

## Abordări inteligente

**Annamária Szenkovits**

Conducător de doctorat: Prof. Dr. Horia F. Pop

Rezumatul tezei de doctorat

Facultatea de Matematică și Informatică
Universitatea Babeş-Bolyai Cluj-Napoca
str. Kogalniceanu 1, RO-400084

Julie 2017

# Cuprins

**1 Introducere** **2**

1.1 Contribuţii originale . . . 2

**2 Testare bazată pe modele: Fundamente și principalele rezultate actuale** **3**

2.1 Procesul testării bazate pe modele . . . 3

2.1.1 Notaţii de modelare . . . 4

2.2 Principalele caracteristici ale sistemelor reactive . . . 5

**3 Optimizarea procesului de testare cu metode inteligente** **5**

3.1 Limbaje și instrumente folosite . . . 7

3.1.1 Lustre: Limbajul din spatele limbajelor SCADE și Lutin . . . 7

3.1.2 Limbajul SCADE pentru modelarea sistemelor critice . . . 7

3.2 Lutin și modele de medii nedeterministe . . . 7

3.3 Testare bazată pe modele cu tehnici evolutive . . . 8

3.3.1 Componentele cadrului de testare . . . 9

3.3.2 Reprezentarea populației (parametrii optimizați) . . . 9

3.3.3 Optimizarea mediului cu DE . . . 10

3.3.4 Optimizare cu un algoritm DE adaptiv . . . 11

**4 Rezultate experimentale** **12**

4.1 O problemă din lumea reală: sisteme industriale din domeniul automaticii căilor ferate . . . 12

4.1.1 SUT: sistemul de protecție a trenului TBL1+ . . . 12

4.1.2 De la un mediu aleatoriu la unul realistic . . . 13

4.2 Rezultatele optimizarii cu DE . . . 13

4.3 Rezultatele optimizarii cu JADE . . . 13

**5 Concluzii și direcții viitoare de cercetare** **15**

**Cuvinte cheie.** testare bazată pe modele, testare automată, testare evolutivă, Differential Evolution, sisteme reactive

# 1 Introducere

Testarea este un pas esențial în ciclul de viață al dezvoltării software-ului. Este comun să se dedice cel puțin 50% din resursele proiectului acestui pas [Beizer, 1990]. Prin urmare, problema găsirii unor modalități eficiente de automatizare a diferitelor aspecte ale testării a primit o atenție sporită în comunitatea științifică [Ammann and Offutt, 2008].

Testarea este și mai importantă în cazul sistemele reactive, deoarece acestea sunt foarte des critice (de exemplu, în sistemele încorporate). Sistemele reactive prezintă provocări suplimentare în această privință, datorită buclei de reacție: Un sistem reactiv acționează asupra mediului său, care, la rândul său, acționează asupra sistemului. Secvențele de intrare realiste trebuie să fie generate în timp real, prin executarea sistemului într-un mediu simulat. Mai mult decât atât, mediile sunt deterministe, deoarece ele se pot schimba mult și, de asemenea, ele nu sunt perfect cunoscute. Lutin [Raymond et al., 2008b] este o limbă dedicată programării unor astfel de simulatoare de mediu; permite simularea sistemelor stohastice reactive. Programele Lutin efectuează o explorare aleatorie ghidată în spaţiul stărilor mediului sistemului alfat în testare (în engleză: System Under Test, SUT). Această explorare conține parametri care definesc probabilitățile ca diferite evenimente să aibă loc. Astfel de probabilități nu sunt întotdeauna ușor de definit. Ideea centrală a acestei teze este de a folosis algoritmi evolutivi pentru a optimiza automat acești parametri.

## 1.1 Contribuţii originale

Contribuţiile originale ale tezei se regăsesc în capitolele 3 şi 4, şi sunt prezentate în continuare.

- Se propune un framework de testare optimizat pentru sisteme reactive. În acest framework, se crează un model de mediu nedeterminist pe baza specificaţiilor sistemului. Acest model nedeterminist se foloseşte pentru stimularea sistemului testat, dar de asemenea pentru a genera date de testare în mod automat. Limbajul propus pentru crearea modelului de mediu este Lutin [Raymond et al., 2008a], un generator automat de teste pentru programe reactive dezvoltat de Verimag research lab, care permite desfăşurarea de explorări aleatoare ghidate în spaţiul stărilor mediului.
- Ghidarea (în generarea cazurilor de testare) poate fi realizată prin utilizarea anu-

mitor facilităţi ale limbajului Lutin care oferă posibilităţi efective de a parametriza procesul care generează datele de testare. Pentru a acoperi în mod eficient modelul şi pentru a acoperi regiuni interne specifice ale sistemului de testat parametrii mediului Lutin sunt setaţi adaptiv astfel încât cazurile de testare generate şi efectuate asupra sistemului de testat acoperă structura codului sistemului în măsură cât mai mare posibil.

- Pentru a realiza setarea adaptivă a mediului Lutin, se propune o abor- dare care implică două metode evolutive: Differential Evolution [Storn and Price, 1997] şi Differential Evolution adaptiv [Zhang and Sanderson, 2009]. Parametrii mediului Lutin sunt folosiţi pentru a crea membrii populaţiei, iar funcţia de fitness a unui individ este reprezentată de gradul de acoperire a sistemului testat. Sunt folosite două metrici de acoperire: MC/DC şi decision coverage.
- Pentru a evalua eficienţa framework-ului de testare propus, au fost efectu- ate experimente asupra unor probleme generate artificial precum şi asupra unor probleme din lumea reală, anume sisteme industriale din domeniul automaticii căilor ferate.

# 2 Testare bazată pe modele: Fundamente și principalele rezultate actuale

În acest capitol, sunt prezentate pașii principali ai procesului de testare bazată pe modele, precum și beneficiile acestei tehnici de testare față de alte metode de testare (de exemplu, tesare manuală, teste bazate pe script-uri, testarea capture/replay). În plus, sunt descrise cele mai utilizate tehnici de modelare. Totodată, se rezumă principalele caracteristici și comportamente ale sistemelor reactive, cu accent pe cele mai importante probleme care apar atunci când vrem să efectuăm testarea pe astfel de sisteme. În cele din urmă, formulăm principalele întrebări abordate în această teză.

## 2.1 Procesul testării bazate pe modele

Ideea principală a testării bazate pe modele este obținerea unui model bazat pe abstractizările sistemului testat și/sau mediul său și apoi generarea de cazuri de testare bazate

pe acest model. Procesul implică următorii pași principali:

1. Crearea unui model pentru SUT, mediul său sau ambele.
2. Generarea testelor abstracte pe baza modelului.
3. Concretizarea testelor abstracte, astfel încât acestea să poată fi executate.
4. Executarea testelor concretizate pe SUT.
5. Analizarea rezultatelor testelor. [Utting and Legeard, 2007]

### 2.1.1 Notaţii de modelare

În această secțiune, oferim o prezentare generală a celor mai frecvent utilizate tipuri de notații de modelare. În plus, subliniem la ce grup de tehnici de modelare aparțin SCADE și Lutin.

Potrivit [Utting et al., 2012], principalele notații pentru modelare pot fi grupate după cum urmează.

- **Notații pre/post (sau bazate pe stări)**: Sistemul este modelat ca o colecție de variabile care reprezintă starea internă a sistemului la un moment dat. Pe lângă variabile, operatorii care pot modifica variabilele sunt de asemenea descriși. În loc să folosim codul limbajului de programare pentru a defini operatorii, se folosesc precondițiile și postcondițiile. Exemple de notații pre/post includ mașinile abstracte din B [Abrial, 1996], UML Object Constraint Language (OCL) [Warmer and Kleppe, 2003], Java Modeling Language (JML) [Leavens and Cheon], VDM [Jones, 1990] [Fitzgerald et al., 2005] și Z.
- **Notații bazate pe tranzații**: După cum sugerează și numele, aceste notații descriu tranzițiile dintre diferitele stări ale sistemului. Notațiile bazate pe tranziții sunt, de obicei, notații grafice care folosesc noduri și arcuri, e.g.: mașinile cu stări finite, diagrame de stări (e.g., UML State Machines, Simulink State flow charts), labeled transition systems, și automate I/O (input/output).
- **Notații funcționale**: În cazul notațiilor funcționale, sistemul este reprezentat ca o colecție de funcții matematice.

- **Notații operaționale**: Atunci când se utilizează notații operaționale, sistemul este modelat ca un set de procese executabile, rulând în paralel. Aceste notații sunt potrivite în special pentru descrierea sistemelor distribuite și a protocoalelor de comunicare, de exemplu: notații Petri.

- **Notații statistice**: SUT este reprezentat ca un model de probabilitate al evenimentelor și valorilor de intrare. Deși aceste notații sunt potrivite pentru modelarea evenimentelor și a valorilor lor de intrare, ele sunt slabe la estimarea rezultatului așteptat al SUT. Pentru modelarea profilurilor de utilizare preconizate, una dintre metodele cele mai de succes sunt lanțurile Markov.

- **Notații bazate pe fluxuri de date**: În loc să modeleze fluxul de control al sistemului, notațiile bazate pe fluxul de date reprezintă fluxul de date prin sistem. De exemplu, Lustre și diagramele bloc utilizate în Matlab Simulink și SCADE utilizează notații bazate pe fluxul de date.

## 2.2 Principalele caracteristici ale sistemelor reactive

Textit Sistemele reactive sunt sisteme care au un comportament ciclic și interacționează permanent cu mediul lor. Pornind de la o intrare inițială, ei vor continua să interacționeze cu mediul lor pe parcursul execuției lor. Termenul *sistem reactiv* a fost introdus prima dată de David Harel și Amir Pnueli [Harel and Pnueli, 1985].

Pentru a descrie comportamentul sistemelor reactive, sunt disponibile două modele principale. Așa cum ilustrează figurile 1 și 2, putem vorbi despre sisteme reactive ale căror reacții sunt declanșate de către evenimente, precum și despre sisteme reactive unde reacțiile se execută odată la o unitate de timp dat.

# 3 Optimizarea procesului de testare cu metode inteligente

În cele ce urmează, vom descrie pe scurt limbile de modelare și instrumentele utilizate pentru a crea frameworkul de testare. Apoi, prezentăm abordările propuse în care am folosit

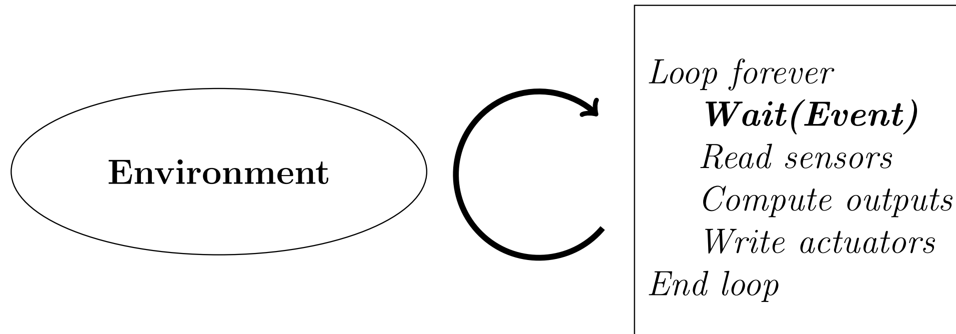


Figura 1: Modelul sistemelor reactive bazate pe evenimente. Un automat de cafea este un bun exemplu pentru un sistem reactiv bazat pe evenimente. Clientul poate executa anumite comenzi cum ar fi alegerea băuturilor sau ingredientelor și introducerea banilor, la care automatul răspunde corăspunzător.

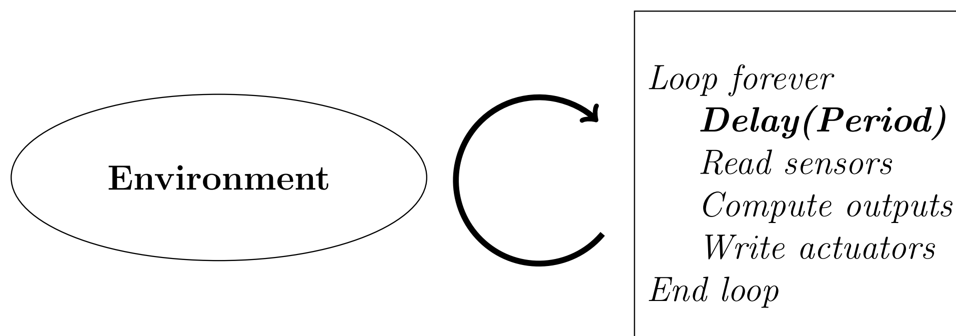


Figura 2: Modelul sistemelor reactive bazate pe evenimente. De exemplu, un termostat care trebuie să păstreze temperatura mediului înconjurător într-un interval dat este un sistem reactiv ale căror reacții sunt declanșate in fiecare unitate de timp. În fiecare unitate de timp, termostatul citește temperatura mediului prin intermediul senzorilor, actualizează modelul său intern, apoi decide dacă mărește sau scade temperatura.

```
node Never (A: bool) returns (never_A: bool);
let
never_A = not(A) -> not(A) and pre(never_A);
tel
```

Figura 3: Exemplu de cod Lustre care demonstrează utilizarea operatorului *pre*.

diferiți algoritmi de căutare pentru a optimiza generarea intrărilor de test în frameworkul de testare.

## 3.1 Limbaje și instrumente utilizate pentru generarea automată a testelor pentru sisteme reactive

### 3.1.1 Lustre: Limbajul din spatele limbajelor Scade și Lutin

Lustre este un limbaj funcțional structurat pe așa-numite **noduri**. Un nod reprezintă un program sau un subprogram și operează pe **fluxuri**: o secvență finită sau infinită de valori dintr-un anumit tip. Un program Lustre are un comportament ciclic, astfel că la iterația a $n$-a a programului, toate fluxurile din program își iau valoarea a $n$-a. Un nod generează unul sau mai mulți parametri de ieșire bazați pe unul sau mai mulți parametri de intrare. Toți acești parametri sunt fluxuri. Figura 3 prezintă un exemplu de nod Lustre.

### 3.1.2 Limbajul Scade pentru modelarea sistemelor critice

SCADE este un limbaj de modelare grafică, folosit pe scară largă pentru sistemele critice, în special pentru proiectarea sistemelor avionice și feroviare. Scade utilizează în esență două notații de modelare diferite: fluxuri de date și mașini de stări (pentru o descriere mai detaliată a diferitelor notații de modelare, vezi secțiunea 2.1.1). Se pot defini operatori SCADE combinând fluxurile de date cu mașinile de stare.

## 3.2 Lutin și modele de medii nedeterministe

Lutin este un generator automat de teste pentru sisteme reactive care se concentrează pe testarea funcțională.

Un *nod* Lutin este un program bazat pe fluxuri de date care transformă o secvență de tupluri de intrare (alcătuite din valori booleene, întregi sau reale) într-o secvență de tupluri

```
node choice () returns(x:int) =
loop {
    |3: x = 42
    |1: x = 1
}
```

Figura 4: Nod Lutin care generează o secvență infinită de numere întregi cu o probabilitate de 0.75 pentru valoarea 42 și 0.25 pentru 1.

de ieșire, exact așa cum SCADE sau diagramele de bloc Simulink. Principala diferență dintre un nod Lutin și un nod SCADE sau Simulink este că cele din urmă sunt alcătuite dintr-un set de ecuații care au (datorită unor restricții sintactice) o singură soluție, în timp ce un nod Lutin este alcătuit dintr-un set de constrângeri (liniare) care poate avea orice număr de soluții.

Pentru a exprima diferite scenarii, Lutin are și structuri de control bazate pe operatori regulari: secvența (**fby**) și alegerea (|). De exemplu, nodul din Fig. 4 va genera o secvență infinită de numere întregi cu o probabilitate de 0.75 pentru valoarea 42 și 0.25 pentru 1.

## 3.3 Testare bazată pe modele cu tehnici evolutive

Algoritmii evolutivi (EA) sunt instrumente puternice de optimizare, pot fi ușor adaptate la probleme specifice de optimizare și au aplicabilitate bună în diferite domenii, cum ar fi ingineria, robotica, biologia, economia etc. EA sunt inspirați de evoluția biologică, utilizând mecanisme precum reproducerea, mutația, recombinarea și selecția.

În clasa algoritmilor evolutivi, Algoritmii Genetici (GA) sunt una dintre cele mai cunoscute și cele mai des folosite tehnici de rezolvare a problemelor de optimizare [Inza et al., 1999; Sagarna et al., 2003]. GA sunt o metodă de căutare bazată pe populație și implică următorii pași principali:

**Algorithm 1** GEA

1: Se creează un set de indivizi sau candidați la soluția problemei de optimizare. (Acest set este denumit populație.)
2: Indivizii promițători sunt alese din populație pe baza unei funcții de fitness.
3: O populație nouă este creată pe baza indivizilor selectate folosind operatorii mutație și recombinare.

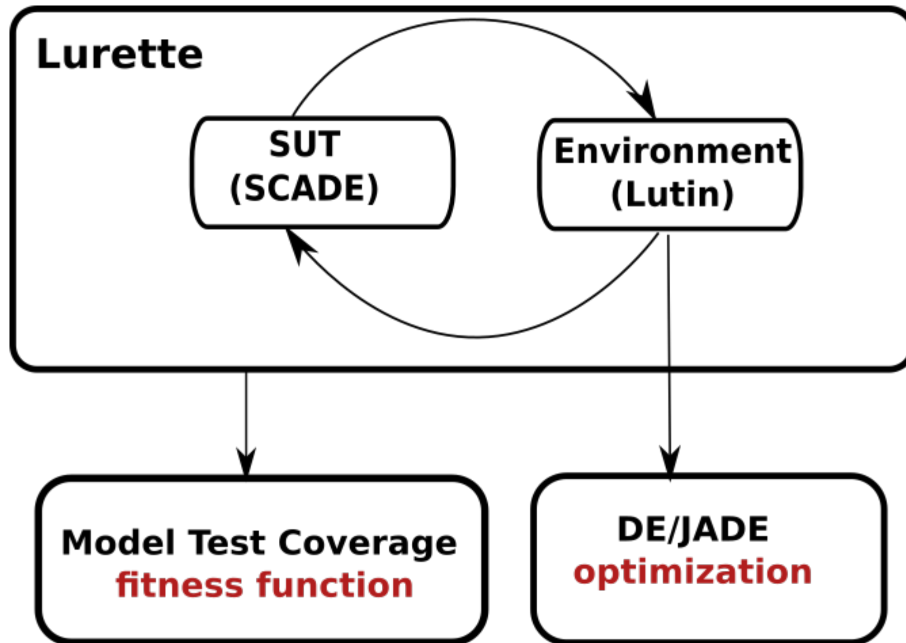


Figura 5: Principalele componente ale cadrului de testare: **sut** sub forma unui cod C code generat dintr-un model SCADE cu ajutorul lui SCADE Suite KCG, modelul **mediului** în Lutin, respectiv **analizorul mtc**. SUT și mediul sunt în interacțiune continuă una cu cealaltă și au un comportament ciclic. Executarea lui SUT și a mediului se face prin intermediul instrumentului Lurette. Optimizarea bazată pe DE/JADE este adăugată pentru a calcula anumite părți ale mediului Lutin și pentru a îmbunătăți generarea automată de teste.

### 3.3.1 Componentele cadrului de testare

Pentru a investiga modul în care metodele evolutive sunt aplicabile sistemelor reactive, am propus extinderea instrumentului Lutin cu un modul de testare evolutivă. Cadrul de testare propus a constat din următoarele componente, așa cum este ilustrat și în Fig. 5: SUT, modelul mediului și analizorul SCADE MTC, instrumentul care măsoară acoperirea atinsă prin testare.

### 3.3.2 Reprezentarea populației (parametrii optimizați)

Am parametrizat operatorii de alegere Lutin și am folosit DE pentru optimizarea acestor parametri. Am restricționat fiecare parametru la intervalul de la 1 la 100. Pentru operatorii de alegere cu două ramuri, am atribuit valoarea $p$ la prima ramură și $100 - p$ la a

```
-- pressing and releasing of the button
node button () returns (bac: bool) =
  loop {
    |p: bac = true
    |100-p: bac = false
  }
```

Figura 6: Simularea apăsării unui buton. Simularea este realizată cu operatorul de alegere aleatoriu al lui Lutin. Parametrul $p$ atribuit ramurilor este optimizat cu DE.

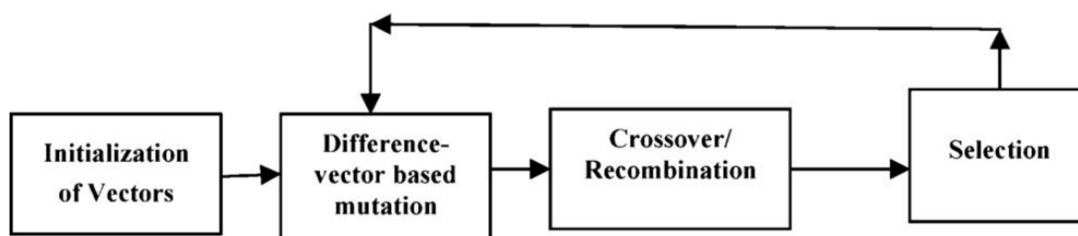


Figura 7: Pașii principali ai algoritmului DE.

doua (e.g. Fig. 6).

### 3.3.3 Optimizarea mediului cu DE

Evoluția diferențială (DE) [Storn and Price, 1997] este un algoritm stochastic, bazat pe populație, cu rezultate bune în multe probleme de optimizare din lumea reală, chiar și în medii non-continue sau dinamice (Unele aplicații și descrieri pot fi găsite în [Das and Suganthan, 2011]).

DE este foarte popular datorită simplității, vitezei și robusteții sale. Pașii principali ai algoritmului sunt arătați în figura 7.

**Operatori.** Populația inițială a fost selectată aleatoriu din intervalele de parametri admisibile. În fiecare iterație, pentru fiecare individ $l$ din populație, un descendent $O[l]$ a fost creat utilizând schema prezentată în Algoritmul 3, unde $U(0, x)$ este un număr distribuit uniform între 0 și $x$, $CR$ denotă raportul crossover, $dim$ este numărul de parametri ai problemei, în timp ce $F$ este factorul de scalare.

The initial population was selected randomly from the admissible parameter ranges. During each iteration, for each individual $l$ from the population, an offspring $O[l]$ was

created using the scheme presented in Algorithm 3, where $U(0,x)$ is a uniformly distributed number between 0 and $x$, $CR$ denotes the crossover ratio, $dim$ is the number of parameters of the problem, while $F$ is the scaling factor.

Principalii pași ai algoritmului sunt descriși în Algoritmul 2.

**Algorithm 2** DE [Mihoc et al., 2016]

1: Randomly generate initial population $P_0$ of solutions;
2: **while** (not termination condition) **do**
3:     **for** each $l = \{1, ..., population\ size\}$ **do**
4:         create offspring $O[l]$ from parent $l$;
5:         **if** $O[l]$ is better then parent $j$ **then**
6:             $O[l]$ replaces parent $j$;
7:         **end if**
8:     **end for**
9: **end while**

**Algorithm 3** DE - the DE*/rand/1/bin* scheme [Mihoc et al., 2016]

***Create offspring*** $O[l]$ **from parent** $P[l]$

1: $O[l] = P[l]$
2: randomly select parents $P[i_1]$, $P[i_2]$, $P[i_3]$, where $i_1 \neq i_2 \neq i_3 \neq i$
3: $n = U(0, dim)$
4: **for** $j = 0$; $j < dim \wedge U(0,1) < CR$; $j = j + 1$ **do**
5:     $O[l][n] = P[i_1][n] + F * (P[i_2][n] - P[i_3][n])$
6:     $n = (n + 1) \bmod dim$
7: **end for**

### 3.3.4 Optimizare cu un algoritm DE adaptiv

Deși DE este un algoritm simplu și eficient, performanța acestuia depinde de parametrii de control (probabilitatea de mutație și de încrucișare) [Gämperle et al., 2002]. Pentru a rezolva problema găsirii unei setări optime de parametri, s-au introdus unele mecanisme care, conform [Eiben et al., 1999], pot fi clasificate în trei clase: control determinist de parametri, control adaptiv de parametri și controlul auto-adaptiv de parametri.

Algoritmul JADE [Zhang and Sanderson, 2009] aparține clasei a doua, are un control adaptiv de parametri, implementează o strategie de mutație ”$DE/curent - to - pbest$” [Zhang and Sanderson, 2009]. Totodată, JADE folosește și o arhivă care are următoarele două funcții: (i) să ofere informații despre direcția de progres, (ii) să îmbunătățească diversitatea populației.

# 4 Rezultate experimentale

## 4.1 O problemă din lumea reală: sisteme industriale din domeniul automaticii căilor ferate

### 4.1.1 SUT: sistemul de protecție a trenului TBL1+

Sistemul pe care am efectuat experimentele este legat de sistemul TBL1+, un sistem de protecție a trenurilor compatibil cu sistemul european de control al trenurilor, utilizat în Belgia și pe linia de cale ferată East Rail din Hong Kong. Principalul său rol este de a asigura o funcționare sigură în cazul unei greșeli cauzate de oameni. Specificația problemei a fost propusă de Siemens. Am folosit codul C cu aproximativ 17000 de linii de cod generate cu generatorul de cod KCG din modelul SCADE al sistemului.

Sistemul este alcătuit dintr-un baliză aflat pe sol care emite un semnal electromagnetic. Acest semnal este recepționat de o antenă aflată sub locomotivă. Dacă trenul se apropie de un semnal roșu, acest sistem de asistență aprinde o lumină în cabină. Mecanicul locomotivei trebuie să confirme apoi că a primit avertizarea prin apăsarea unui buton. Dacă mecanicul nu face acest lucru, atunci frâna de siguranță a trenului este activată automat.

Pe lângă funcția de verificare a vigilenței menționată mai sus, sistemul TBL1+ are o funcție de verificare a restricției de viteză. Această funcționalitate este activată de o baliză situată la 300 de metri în sus de la un semnal roșu. Dacă trenul se deplasează cu o viteză mai mare de 40 km/h înaintea semnalului roșu, sistemul TBL1+ declanșează frâna de siguranță. Cu toate acestea, frâna poate fi dezactivată de către mecanic după 20 de secunde prin apăsarea butonului de confirmare, dacă pericolul nu mai există.

### 4.1.2 De la un mediu aleatoriu la unul realistic

Am creat trei modele Lutin diferite pentru mediul sistemului TBL1+. În toate cele trei versiuni, modelul de mediu avea următoarele componente principale: viteza trenului, codurile transmise de către balizele aflate la sol, respectiv butonul de confirmare pentru frâna de siguranță. Am creat următoarele trei modele de mediu:

1. **Model de mediu aleatoriu**: Acesta este un model de mediu foarte simplu, unde am presupus că nu avem cunoștințe despre modul în care tbl funcționează. Viteza trenului ia valori aleatorii de la o iterație la alta, iar starea butoanelor (apăsate sau eliberate) este de asemenea generată aleatoriu utilizând o distribuție uniformă.

2. **Model de mediu realist**: În cea de-a doua versiune a modelului de mediu, am presupus că avem mai multe cunoștințe despre TBL1+ ca în primul caz și cunoaștem posibilele valori valide care pot fi transmise prin balize. Cu toate acestea, codurile sunt încă selectate aleatoriu cu distribuție uniformă.

3. **Model de mediu realist optimizat cu de/jade**: Cel de-al treilea model de mediu este varianta în care a fost adăugată optimizarea cu DE și JADE.

## 4.2 Rezultatele optimizarii cu DE

Am rulat DE cu două setări de parametri diferite. Aceste setări sunt rezumate în Tabelul 1. Bazat pe [Liu and Lampinen, 2002], pentru unele probleme o valoare mai mică de $F$ este o setare bună, prin urmare, am folosit o valoare mică pentru $F$ pentru ambele setări. Am rulat 10 experimente independente pentru ambele setări. Rezultatele medii și deviația standard sunt descrise în tabelul 4, precum și rezultatele obținute cu mediul simplu Lutin, unde s-a folosit distribuția uniformă.

## 4.3 Rezultatele optimizarii cu JADE

În cazul testării optimizate cu JADE, am folosit aceleași modele de mediu Lutin ca și pentru experimentele cu DE. Am folosit parametrii recomandați pentru JADE: 0,05 pentru p (determină gradul de ”greediness” pentru strategia de mutație) și 0,1 pentru c (controlează rata de adaptare a parametrilor). Am executat cinci rulări independente pentru fiecare

| Parameter | Setare 1 | Setare 2 |
|---|---|---|
| Dim. populație | 50 | 50 |
| Factor de scalare F | 0.1 | 0.25 |
| Raport crossover (CR) | 0.9 | 0.75 |

Tabela 1: Setările experimentale pentru DE.

| Parametri | Setare 1 *avg.* ± *stdev.* | Setare 2 *avg.* ± *stdev.* | Mediu realistic simplu Lutin |
|---|---|---|---|
| 3 | $70.03 \pm 0.45$ | $70.51 \pm 0.32$ | 62.70 |
| 4 | $69.86 \pm 0.36$ | $70.07 \pm 0.45$ | 62.70 |
| 5 | $70.51 \pm 0.40$ | $70.72 \pm 0.12$ | 62.70 |
| 6 | $71.10 \pm 0.24$ | $71.28 \pm 0.23$ | 62.70 |
| 7 | $71.52 \pm 1.17$ | $72.74 \pm 0.22$ | 62.70 |
| 8 | $69.75 \pm 0.68$ | $70.17 \pm 0.25$ | 62.70 |

Tabela 2: **Ratele de acoperire mtc mc/dc** (valorile medii și deviația standard) obținute cu mediul Lutin optimizat cu DE, respectiv mediul Lutin simplu. Setările 1 și 2 pentru DE sunt descrise în tabelul 1. Numărul de parametri se referă la numărul de parametri optimizate în codul Lutin.

| Parametri | Setare 1 *avg.* ± *stdev.* | Mediu aleatoriu | Mediu realistic simplu Lutin |
|---|---|---|---|
| 3 | $75.48 \pm 0.00$ | 21.29 | 66.45 |
| 4 | $75.48 \pm 0.00$ | 21.29 | 66.45 |
| 5 | $76.06 \pm 0.50$ | 21.29 | 66.45 |
| 6 | $76.16 \pm 0.47$ | 21.29 | 66.45 |
| 7 | $79.39 \pm 0.47$ | 21.29 | 66.45 |

Tabela 3: Acoperirea la nivel de ramificaţie (valorile medii și deviația standard) obținute cu mediul Lutin optimizat cu DE, respectiv mediul Lutin simplu. Setările 1 și 2 pentru DE sunt descrise în tabelul 1. Numărul de parametri se referă la numărul de parametri optimizate în codul Lutin.

| Parametri | JADE <br> *avg.* $\pm$ *stdev.* | Mediu realistic simplu Lutin |
|---|---|---|
| 5 | $71.82 \pm 0.00$ | 62.70 |
| 6 | $71.43 \pm 0.60$ | 62.70 |
| 7 | $71.27 \pm 0.00$ | 62.70 |

Tabela 4: Ratele de acoperire MTC MC/DC (rezultatele medii și deviația standard) obținute cu mediul jade - optimizat și cu mediul simplu Lutin.

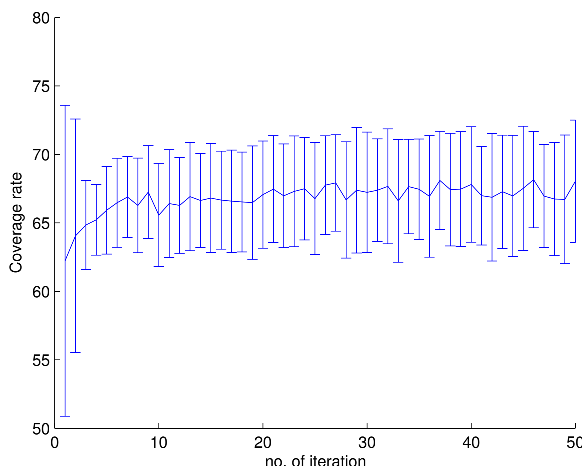


Figura 8: Media populației și deviația standard într-o rulare cu JADE, pentru 7 parametri.

problemă. Figura 8 prezintă evoluția ratelor de acoperire în 108 iterații. Se prezintă valorile medii și deviația standard.

Tabelul 4 prezintă rezultatele obținute pentru algoritmul JADE și pentru valorile implicite. Așa cum ne-am așteptat, JADE are rezultate semnificativ mai bune. Comparând rezultatele obținute cu [Szenkovits et al., 2016], avem asemănări, dar în cazul optimizării cu JADE, nu a fost necesară ajustarea parametrilor, ca într-un algoritm ”tradițional” DE.

# 5 Concluzii și direcții viitoare de cercetare

În această teză, am propus o metodă de testare bazată pe modele optimizată pentru sisteme reactive. Am folosit limbajul Lutin pentru a crea un model de mediu executabil, nedeterminist pentru SUT. Deoarece mediul de execuție al unui sistem reactiv este deseori

nespecificat sau se poate schimba mult, natura nedeterministă a limbajului Lutin îl face potrivit pentru a modela medii realiste. Crearea unui model de mediu eficient necesită adesea cunoștințe specifice domeniului.

Am propus o abordare în care am lăsat doi algoritmi inteligenți să calculeze parametrii modelului de mediu Lutin și să optimizeze explorarea aleatoră ghidată a spațiului de stare al mediului. Mai exact, am folosit doi algoritmi evolutivi: DE și JADE. Am folosit parametrii mediului Lutin pentru a crea membrii populației. La sfârșitul fiecărei serii de iterații de o anumită lungime au fost selectați cei mai buni indivizi ai populației, iar modelele de mediu asociate acestor indivizi au fost actualizate.

Funcțiile de fitness utilizate au fost MC/DC și decision coverage pentru SUT. Am testat metodele DE și JADE pe sistemul TBL1+, un sistem de protecție a trenului. Am măsurat mai întâi rata de acoperire obținută cu mediul Lutin simplu, în care nu a fost adăugată nicio optimizare, apoi a fost comparată cu rezultatele obținute cu mediile optimizate cu DE și JADE. În ambele cazuri, rata de acoperire a lui SUT a fost îmbunătățită în medie cu 9%, fără a avea nevoie de cunoștințe suplimentare de domeniu. Deoarece vorbim despre un sistem complex din lumea reală, putem considera aceste rezultate semnificative. Chiar dacă rata de acoperire al lui SUT putea fi îmbunătățită atât cu DE, cât și cu JADE, JADE are un beneficiu major față de DE: jade își calculează parametrii de control adaptiv, în timp ce în DE, acești parametri au fost setate prin metoda ”trial and error”.

În viitor, ne propunem să creștem în continuare acoperirea MC/DC și să ajungem cât mai aproape posibil de 100 %. Propunem să încercăm și alte tehnici evolutive.

Obținerea unui grad mare de acoperire pentru SUT este deosebit de importantă pentru sistemele critice, ca și în cazul sistemul TBL1+, unde diferite standarde reglează procesul de testare. În mod ideal, modelele trebuie verificate prin metode de analiză statică. În realitate însă, nu este întotdeauna posibil să se verifice totul în mod static, astfel încât verificarea dinamică ca și testarea poate fi utilizată cu succes în combinație cu analiza statică. În cadrul de testare propus, procesul de generare a testelor este complet automatizat cu Lutin și diferite metode inteligente, deci explorarea aleatoră ghidată se poate face fără supraveghere, mărind șansele pentru găsirea unor teste mai ”interesante”.

# Bibliografie

J.-R. Abrial. *The B-book: Assigning Programs to Meanings*. Cambridge University Press, New York, NY, USA, 1996. ISBN 0-521-49619-5.

P. Ammann and J. Offutt. *Introduction to Software Testing*. Cambridge University Press, New York, NY, USA, 1 edition, 2008. ISBN 0521880386, 9780521880381.

B. Beizer. *Software Testing Techniques (2Nd Ed.)*. Van Nostrand Reinhold Co., New York, NY, USA, 1990. ISBN 0-442-20672-0.

S. Das and P. N. Suganthan. Differential evolution: a survey of the state-of-the-art. *Evolutionary Computation, IEEE Transactions on*, 15(1):4–31, 2011.

A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *Evolutionary Computation, IEEE Transactions on*, 3(2):124–141, 1999.

J. Fitzgerald, P. G. Larsen, P. Mukherjee, N. Plat, and M. Verhoef. *Validated Designs For Object-oriented Systems*. Springer-Verlag TELOS, Santa Clara, CA, USA, 2005. ISBN 1852338814.

R. Gämperle, S. D. Müller, and P. Koumoutsakos. A parameter study for differential evolution. *Advances in intelligent systems, fuzzy systems, evolutionary computation*, 10:293–298, 2002.

D. Harel and A. Pnueli. Logics and models of concurrent systems. chapter On the Development of Reactive Systems, pages 477–498. Springer-Verlag New York, Inc., New York, NY, USA, 1985. ISBN 0-387-15181-8. URL http://dl.acm.org/citation.cfm?id=101969.101990.

I. Inza, P. Larranaga, R. Etxeberria, and B. Sierra. Feature subset selection by bayesian network-based optimization. 1999.

C. B. Jones. *Systematic Software Development Using VDM (2Nd Ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1990. ISBN 0-13-880733-7.

G. T. Leavens and Y. Cheon. The java modeling language (jml) home page. http://www.eecs.ucf.edu/~leavens/JML//index.shtml. [Online; accessed May-2017].

J. Liu and J. Lampinen. On setting the control parameter of the differential evolution method. In *Proc. 8th Int. Conf. Soft Computing MENDEL 2002*, pages 11–18, 2002.

T. D. Mihoc, R. I. Lung, N. Gaskó, and M. Suciu. Approximation of (k,t)-robust equilibria. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 805–811, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4206-3. doi: 10.1145/2908812.2908877. URL http://doi.acm.org/10.1145/2908812.2908877.

P. Raymond, Y. Roux, and E. Jahier. Lutin: A language for specifying and executing reactive scenarios. *EURASIP J. Emb. Sys.*, 2008, 2008a. URL http://dblp.uni-trier.de/db/journals/ejes/ejes2008.html#RaymondRJ08.

P. Raymond, Y. Roux, and E. Jahier. Lutin: A language for specifying and executing reactive scenarios. *EURASIP J. Emb. Sys.*, 2008, 2008b.

R. Sagarna, J. A. Lozano, and P. M. Lardiazabal. On the performance of estimation of distribution algorithms applied to software testing. 2003.

R. Storn and K. Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

A. Szenkovits, N. Gaskó, and E. Jahier. *Applications of Evolutionary Computation: 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30 – April 1, 2016, Proceedings, Part I*, chapter Environment-Model Based Testing with Differential Evolution in an Industrial Setting, pages 819–830. Springer International Publishing, Cham, 2016. ISBN 978-3-319-31204-0. doi: 10.1007/978-3-319-31204-0_52. URL http://dx.doi.org/10.1007/978-3-319-31204-0_52.

M. Utting and B. Legeard. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007. ISBN 0123725011,

9780080466484.

M. Utting, A. Pretschner, and B. Legeard. A taxonomy of model-based testing approaches. *Softw. Test. Verif. Reliab.*, 22(5):297–312, Aug. 2012. ISSN 0960-0833. doi: 10.1002/stvr.456. URL http://dx.doi.org/10.1002/stvr.456.

J. Warmer and A. Kleppe. *The Object Constraint Language: Getting Your Models Ready for MDA*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2 edition, 2003. ISBN 0321179366.

J. Zhang and A. C. Sanderson. Jade: adaptive differential evolution with optional external archive. *Evolutionary Computation, IEEE Transactions on*, 13(5):945–958, 2009.