An improved test generation approach from extended finite state machines using genetic algorithms

Raluca Lefticaru, Florentin Ipate

University of Bucharest, Romania

SEFM 2012, Thessaloniki, Greece

Outline

- Problem formulation: EFSM test generation
- Motivation: why **metaheuristics** in **EFSM testing?**
- Previous approaches
- Proposed strategy: decomposing long paths into independent subpaths
- Fitness functions: ***al+nbl*** versus ***ICF***
- Empirical evaluation
- Conclusions and future work

Test Data Generation for State-based Testing

Problem Formulation

Given: some **paths** in the state machine (obtained according to a certain coverage criteria).

Find: input parameter values to trigger the given paths.

Approach: Use metaheuristic search techniques (such as **genetic algorithms**) to find for each path (sequence of methods) the input values for the parameters, which satisfy the corresponding guards (pre-conditions)

Questions:

Encoding: $x = (x_1, x_2, \dots, x_n)$
Fitness function: $f(x) = ?$
Search technique: which one, tunings?

Motivation

- State-based testing
  - test selection methods for FSMs: W, Wp methods etc.
  - **coverage criteria** for state machines diagrams
- Automatic generation of test cases from EFSMs, state machines diagrams, X-machines etc. is not straightforward!
- Dranidis, Bratanis, Ipate . *JSXM: A Tool for Automated Test Generation*. SEFM 2012. http://www.jsxm.org/
- Constraints (guards) on transitions → **NP-complete problem!**
- **Idea:** *Why not to use the power of **evolutionary algorithms** or other **metaheuristics** to solve state-based testing problems?*
- This answer is given by new field of **Search-based Software Engineering**

Search-Based Software Engineering (SBSE)

- Software engineering problems → ***optimization problems*** + **metaheuristic search techniques** (**GAs**, PSO, SA).
- Term of **SBSE**: first coined by *Harman* and *Jones* (2001).
- Applications: **software testing**, requirements engineering, automated maintenance, service-oriented SE, compiler optimization, quality assessment, project planning and cost estimation.

Repository of publications on SBSE:
http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/

M. Harman, B.F. Jones. **Search-based software engineering**. *Information and Software Technology*, 2001.

Search-based testing from state-based specifications

Kalaji, Hierons, Swift. Generating feasible transition paths for testing from an extended finite state machine (EFSM). *ICST 2009*

Yano, Martins, Sousa. Generating feasible test paths from an executable model using a multi-objective approach. *ICST 2010*

Yano, Martins, Sousa. MOST: a multi-objective search-based testing from EFSM. *ICST 2011*

Derderian, Hierons, Harman, Guo. Generating feasible input sequences for extended finite state machines (EFSMs) using genetic algorithms. *GECCO 2005*

Kalaji, Hierons, Swift. An integrated search-based approach for automatic testing from extended finite state machine (EFSM) models. *Information and Software Technology*, 2011

Zhao, Harman, Li. Empirical study on the efficiency of search based test generation for EFSM models. *ICST 2010*

Lefticaru, Ipate. Functional search-based testing from state machines. *ICST 2008*

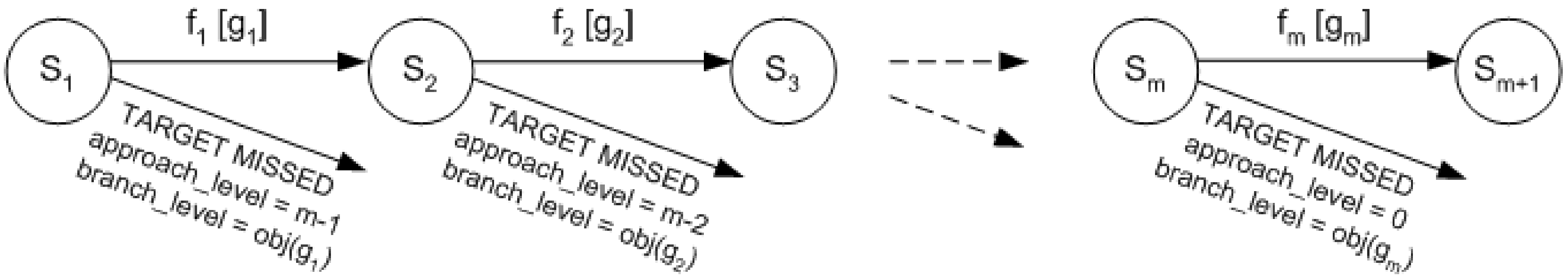
Lefticaru, Ipate. Automatic state-based test generation using genetic algorithms. *SYNASC 2007*

Kalaji, Hierons, Swift. Automatic generation of test sequences form EFSM models using evolutionary algorithms. BURA *Technical Report*, 2008.

Generating feasible transition paths

Generating test data for feasible paths in the state machine

Fitness Function Calculation: *al+nbl*



Fitness Calculation: *al + nbl*

$$fitness = approach\_level + normalized\_branch\_level$$

$$approach\_level \in \{0,1,\ldots,m-1\}$$

$$normalized\_branch\_level \in [0,1]$$

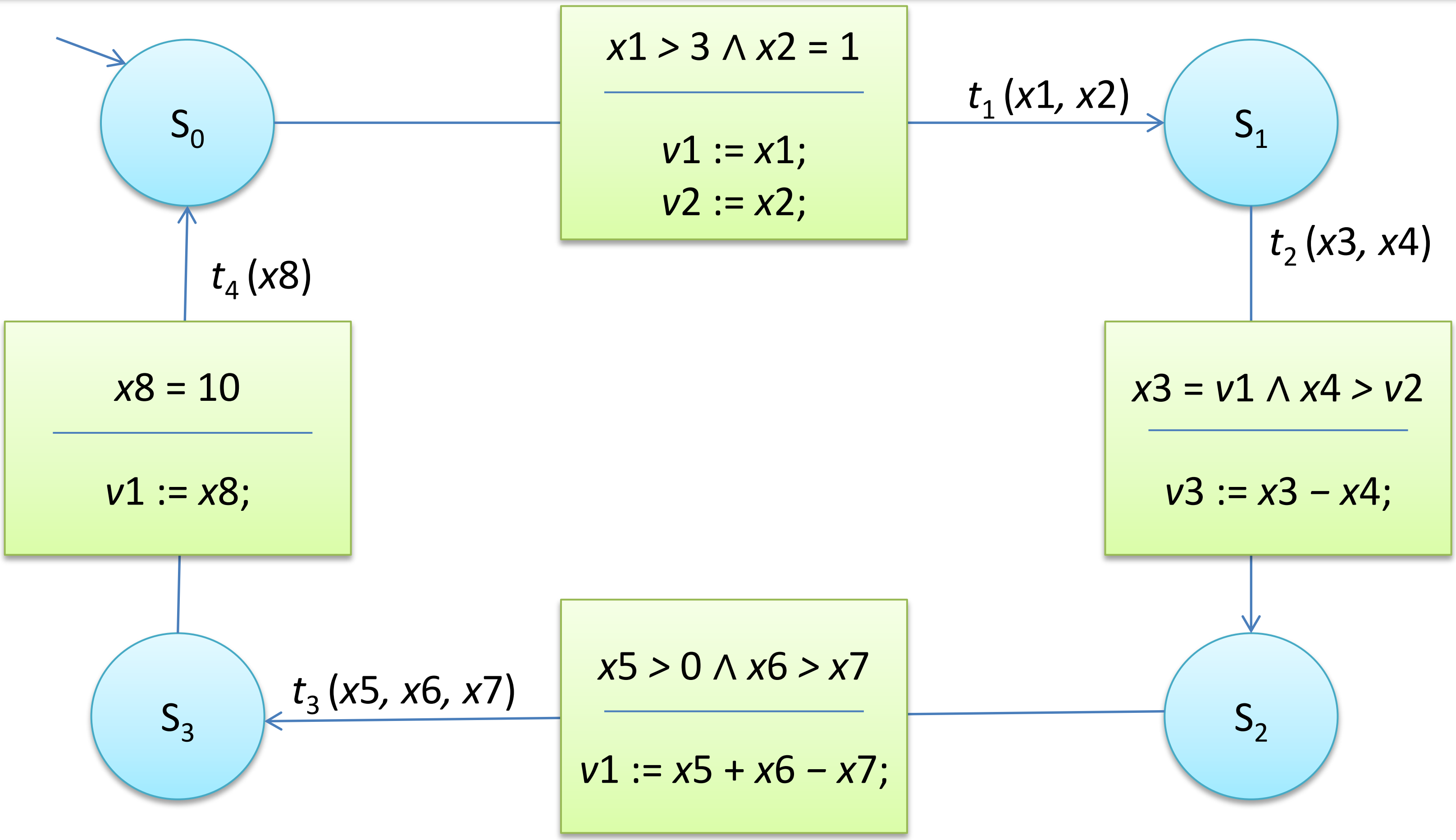
R. Lefticaru, F. Ipate. Automatic state-based test generation using genetic algorithms. In *SYNASC 2007*

Objective Functions used for Branch Level

Classical Tracey's objective functions

Relational predicate or logical connective	Objective function obj
$a = b$	if $abs(a-b) = 0$ then 0 else $abs(a-b) + K$
$a \neq b$	if $abs(a-b) \neq 0$ then 0 else K
$a < b$	if $a - b < 0$ then 0 else $(a-b) + K$
$a \leq b$	if $a - b \leq 0$ then 0 else $(a-b) + K$
$a > b$	if $b - a < 0$ then 0 else $(b-a) + K$
$a \geq b$	if $b - a \leq 0$ then 0 else $(b-a) + K$
Boolean	if $TRUE$ then 0 else K
$a \wedge b$	$obj(a) + obj(b)$
$a \vee b$	$min(obj(a), obj(b))$
a xor b	$obj((a \wedge \neg b) \vee (\neg a \wedge b))$
$\neg a$	Negation is moved inwards and propagated over a

EFSM Example



Path: [t_1(x1,x2), t_2(x3, x4), t_3(x5,x6,x7), t_4(x8)]

Chromosome: (x1, x2, … , x8)

Fitness Function Evaluation: *al* + *nbl*

Require: Target path p, containing the transitions $t_1, t_2, \ldots, t_m$, corresponding guards $g_1, g_2, \ldots, g_m$, chromosome $x = (x_1, \ldots, x_n)$

Ensure: The fitness value of the chromosome $x = (x_1, \ldots, x_n)$ for the given path.

Create an instance of the EFSM in the initial configuration.

$aproach\_level \leftarrow m - 1$

for $i = 1 \to m$ **do**

  {for every transition t_i in the sequence p}

  if **not** g_i **then**

    Calculate $obj(g_i)$

    return $aproach\_level + norm(obj(g_i))$

  else

    $aproach\_level \longleftarrow aproach\_level - 1$

    Apply transition t_i with the corresponding values from $(x_1, \ldots, x_n)$

  end if

end for

return 0

Stopping at the **first unsatisfied guard!**

Give a chance to other blocks to be evaluated, too!

Who's better?

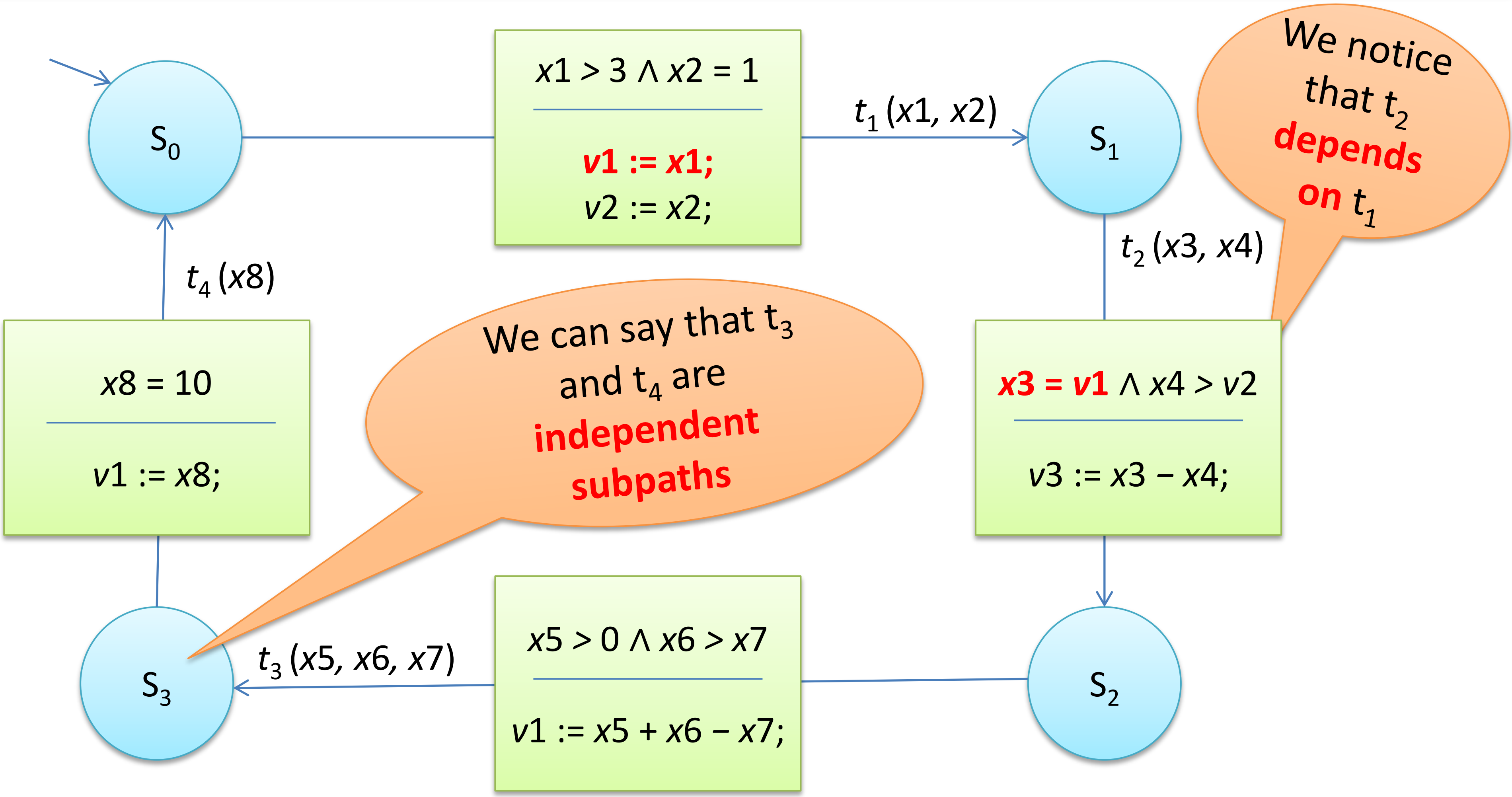
Chromosome1:

(x1, x2, x3, x4, x5, x6, x7, x8)

Chromosome2:

(y1, y2, y3, y4, y5, y6, y7, y8)

Establishing Dependencies in EFSM



Independent sub-paths: [[t_1(x1,x2), t_2(x3, x4)] , [t_3(x5,x6,x7)], [t_4(x8)]]

An Improved Fitness Function for EFSM Testing

- **Main idea:** defining **dependency relations** between transitions, decomposing long paths into **independent sub-paths**
- For given paths, that can be logically decomposed into independent sub-paths, a new function, ***ICF***, was designed, which takes into account the independent components:

$$fitness(x) = \sum_{i=1}^{subpaths\_no} fitness(subpath_i)$$

- **Why?** The **fitness function** that guides the search is ***crucial*** for the success of a genetic algorithm! An improvement in the fitness function will reduce the duration of the generation process and increase the success chances of the search algorithm.

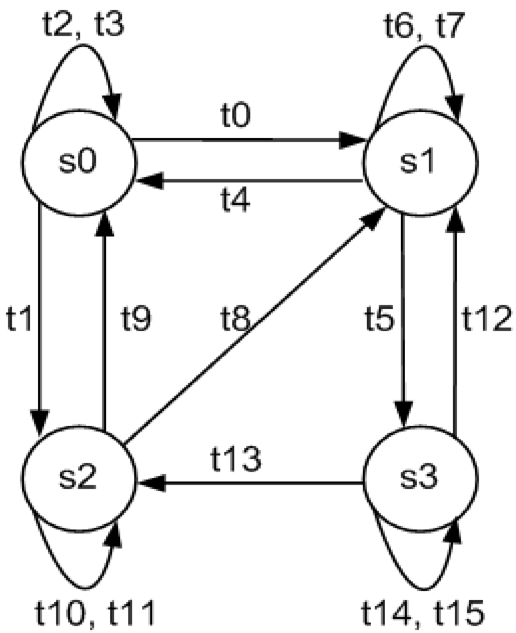
Extended Finite State Machines (EFSM)

- An ***Extended Finite State Machine* (EFSM)** is a 6-tuple (S, s_0, V, I, O, T) where:
  - S is a non empty ***set of logical states***;
  - $s_0 \in S$ is the ***initial state***;
  - V is the finite set of ***internal variables***;
  - I and O are the set of ***input and output interactions***, respectively;
  - T is the ***finite set of transitions***.
- A ***transition*** $t \in T$ has ***start*** and ***end states,*** may have associated input parameters, a ***guard*** and a ***computational block*** which consists of assignments and output statements.

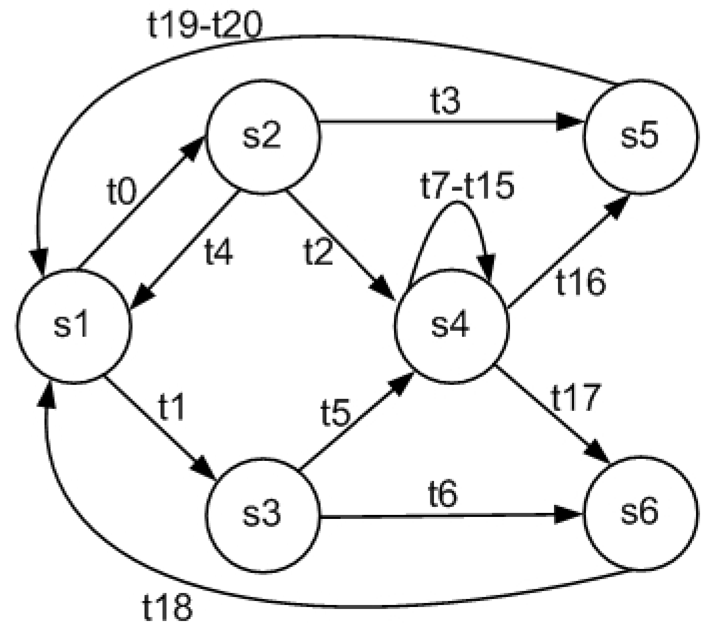
Dependency Relations on EFSMs

- A transition t ***depends on the internal variable*** $v_i \in V$ if the value of v_i is evaluated by its guard.
- A transition t ***modifies the internal variable*** $v_i \in V$ if v_i appears on the left hand side of an assignment operation of its sequence of atomic operations.
- Let $[\, t_1 , t_2 , \ldots , t_i , \ldots , t_j , \ldots t_p \,]$ be a **path** in the EFSM. The transition t_j ***directly depends on the transition*** t_i if there exists an internal variable $v_k \in V$ such as t_i modifies the internal variable v_k, t_j depends on v_k and there is no other transition t_r, $i < r < j$ that modifies the internal variable v_k.
- Result: a ***dependency graph*** between transitions.
- Compute the ***independent subpaths*** (connected components in the dependency graph).
- ***ICF fitness function***: takes into account the fitness of the independent components

Empirical Evaluation



(a) Book



(b) Class 2 transport protocol

4 states
2 internal variables
16 transitions

6 states
5 internal variables
20 transitions

Excerpt from Protocol EFSM

t	$s_s \rightarrow s_e$	Input declaration	Guards	Operations
t_0	$s_1 \rightarrow s_2$	t1(*dst_add*, prop_opt)	Nil	opt := prop_opt; R_credit := 0;
t_1	$s_1 \rightarrow s_3$	t2(*peer_add*, opt_ind, cr)	Nil	opt := opt_ind; S_credit := cr; R_credit := 0; TRsq := 0; TSsq := 0;
t_2	$s_2 \rightarrow s_4$	t2(opt_ind, cr)	opt_ind < opt	opt := opt_ind; S_credit := cr;
t_3	$s_2 \rightarrow s_5$	t3(opt_ind, *cr*)	opt_ind > opt	Nil
t_5	$s_3 \rightarrow s_4$	t5(accpt_opt)	accpt_opt < opt	opt := accpt_opt; TRsq := 0; TSsq := 0;
t_7	$s_4 \rightarrow s_4$	t7(*Udata, E0SDU*)	S_credit > 0	S_credit := S_credit -1; TSsq := (TSsq +1)mod128;
t_8	$s_4 \rightarrow s_4$	t8(Send_sq, *Ndata,E0TSDU)*	R_credit $\neq$ 0 AND Send_sq = TRsq	TRsq := (TRsq+1)mod128; R_credit := R_credit -1;
t_9	$s_4 \rightarrow s_4$	t9(Send_sq, *Ndata,E0TSDU*)	R_credit = 0 $\vee$ Send_sq $\neq$ TRsq	Nil
t_{10}	$s_4 \rightarrow s_4$	t10(cr)	Nil	R_credit := R_credit + cr;
t_{11}	$s_4 \rightarrow s_4$	t11(XpSsq, cr)	TSsq $\geq$ XpSsq $\wedge$ cr + XpSsq - TSsq $\geq$ 0 $\wedge$ cr +XpSsq - TSsq $\leq$ 15	S_credit := cr + XpSsq - TSsq;

Infeasible path example: [t_1 , t_5 , t_8 , …] because of *R_credit* internal variable modification

Experiment Settings and Results: ***al +nbl* *vs.* *ICF***

- EFSM models: Book, Protocol
- Large pool of **randomly generated paths**, for each model, having different lengths (from 6 to 35 transitions)
- GA applied for each path 100 times with *al+nbl*, 100 times with *ICF*
- GA settings: population size 20, *BestChromosomesSelector 0.8*, single point crossover, mutation rate 1/12
- H_0: *There is no difference in efficiency (number of generations needed by the GA to find a solution) between the two fitness functions,* al + nbl *and* ICF.
- Statistical tests: *t-test* with significance level $\alpha = 0.05$, *p*-values were also recorded to decide if tests were significant (confidence 95%) and very significant (confidence 99%).

Summary of Experimental Results

EFSM Model	Path Set	No. of Paths	Path Lengths	Stat. Signif.	ICF +	Alnbl +	Very Signif.	ICF +	Alnbl +
Book	T1	20	6 – 15	14	14	0	13	13	0
Book	T2	20	20	16	15	1	15	14	1
Book	T3	20	25	18	18	0	17	17	0
Book	T1–T3	60	6 – 25	48	47	1	45	44	1
Protocol	P1	20	6 – 15	0	0	0	0	0	0
Protocol	P2	20	20	6	6	0	3	3	0
Protocol	P3	20	25	4	4	0	2	2	0
Protocol	P4	20	30	6	6	0	4	4	0
Protocol	P5	20	35	7	7	0	4	4	0
Protocol	P1–P5	100	6 – 35	23	23	0	13	13	0
Book, Protocol	All	160	6 – 35	71	70	1	58	57	1

Conclusions and Future Work

- *ICF* obtains better results than ***al + nbl*** for the majority of the paths: it improves the success rate of the GA, especially for complex paths
- Easy to trigger paths: similar behaviour
- Complex paths: the differences are very significant (confidence 99%)

Future Work

- Analysing how often independent sub-paths occur in real world EFSMs
- Comparing the results obtained by GA with other metaheuristic search techniques.
- Adapt and integrate this search-based approach into existing tools, such as JSXM

Questions?