Some Reflections on Fault Tolerance Issues

Fevzi Belli (*belli@upb.de)*
Univ. Paderborn, Germany, and IzTech, Izmir

Abstract
Notions such as *detecting*, *correcting*, and *self-correcting* in their application to a well-defined class of faults are described, analyzed, and operationalized. Examples include a multi-storey shelving system and a sequential adder which are analyzed and extended to become (partly) self-correcting.

Outline

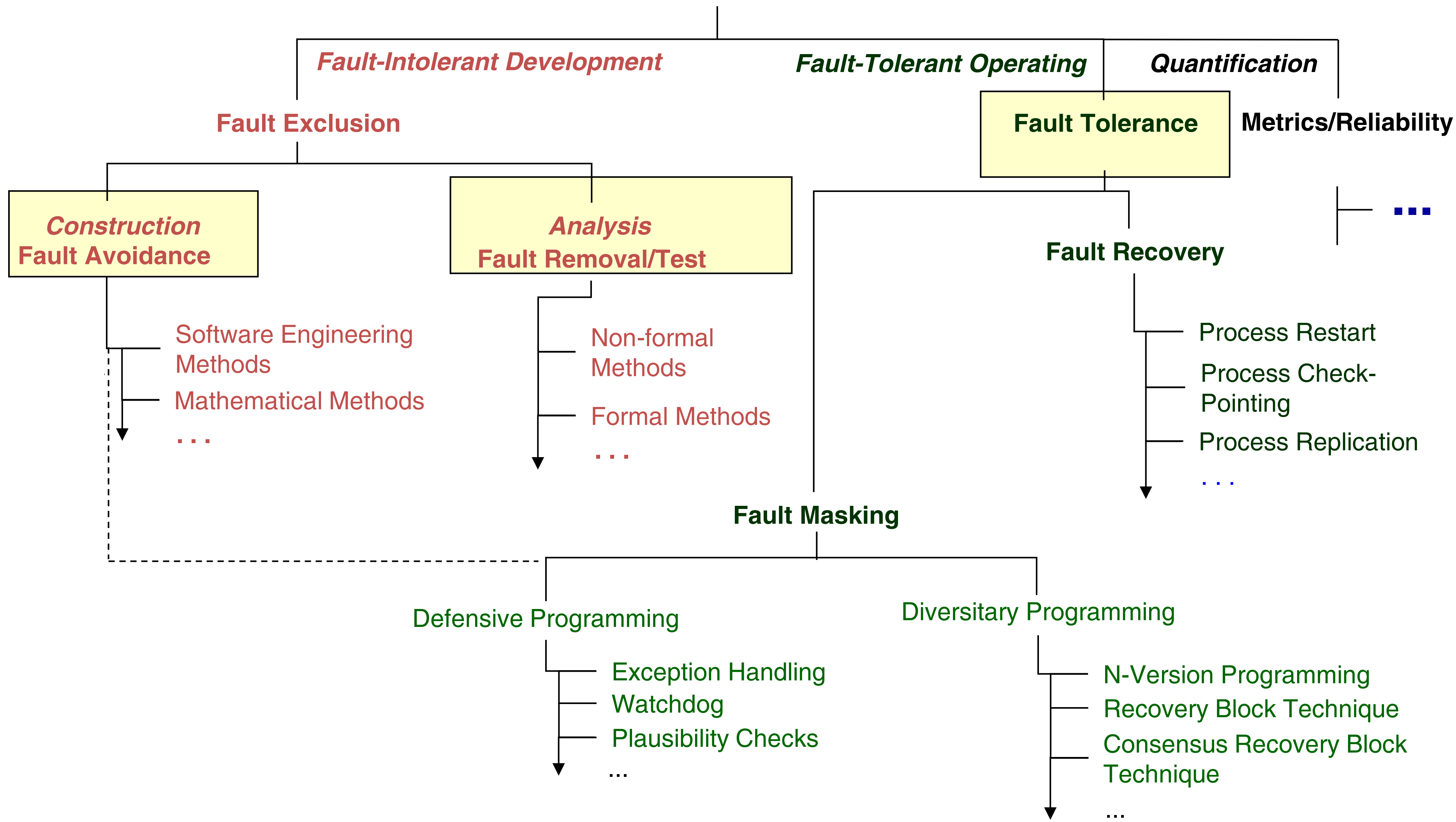
1. Introduction

2. Modeling, Analysis , *Toward Self-Testing and Self-Correcting*

3. Examples

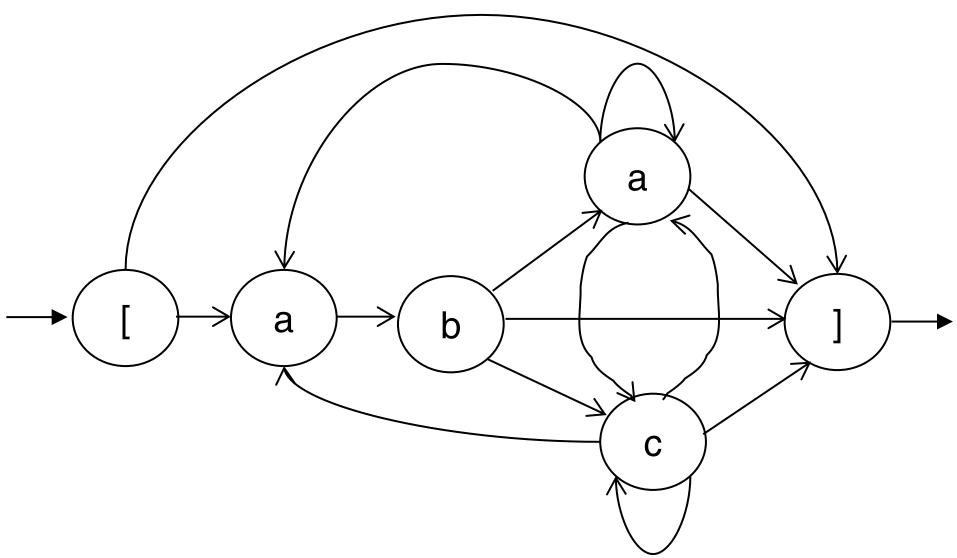
1. Introduction

Fault Handling



Reminder:
Regular Expressions (RegEx), Finite-State Automata (FSA), and Event Algebra

- A regular expression T consists of symbols connected by the operations
  - *Sequence* („."(dot) that can be left out: concatenation),
  - *Selection* („+": exclusive or),
  - *Iteration* („*": arbitrarily often repetition, Kleene´s star operation);
  - „+": at least one occurrence of …)
  - λ : empty event; ω : = λ* (empty word)
- **Example:** T= [(ab(a+c)*)*]; a,b, c: interpreted as *events*
- Regular expressions (RegEx) can be represented graphically by FSA, or simplified, by *Event Sequence Graphs* (*ESG*).
- **Example:**



- Merging inputs and states leads to more efficient algorithms for analysis and test.
- The result is a simplified version of the state transition diagram (STD) of the FSA that we call an *Event Sequence Graph (ESG)* based on [Myhill].

Myhill, J., “Finite Automata and the Representation of Events”, Wright Air Devel. Command, TR 57-624, pp. 112-137 (1957)

State Transition Diagram (STD) of the FSA

Event Sequence Graph (ESG) of the FSA

- As regular expression: Both are represented as ***`ab`*****.**

Event-Based Modeling – The Idea

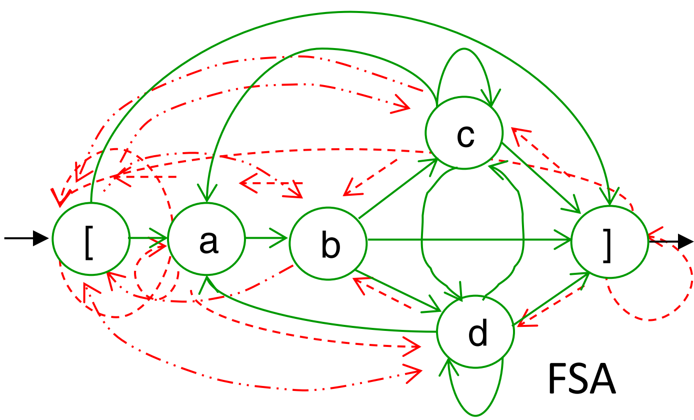
— Example: an elementary ***click – copy/delete – paste*** application

[: start (entry menu);]: finish (exit menu); b: copy an object;
c: delete an object; d: paste an object

[(ab(c+d)*)*] (RegEx)

- *Legal event sequences* (*ES*) of length 2:
  [a, [ ], ab, bc, bd, b], ca, cc, cd, c], dc, dd, d]

- *Illegal (faulty) event sequences (FES)* of length 2:
  [b, [c, [d, [[, ac, ad, a[, ba, bb, b[, cb, …



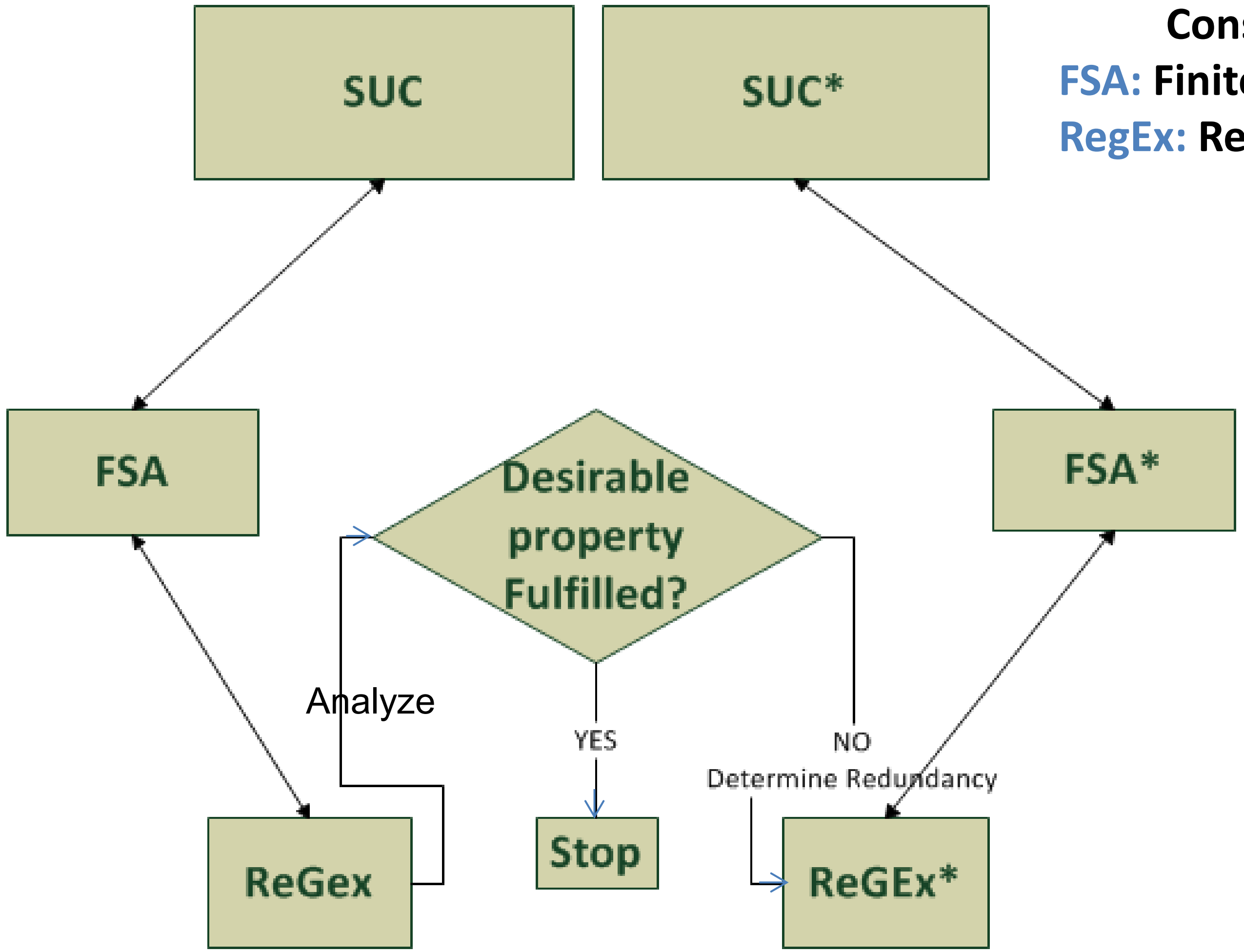
➢ *Complete ES/FES* are paths ("walks") through the graph.
CES: [ab], [abc], [abcd], …; CFES: [a, [ac, [ad, [aba, …

CpyPst -> [A
A -> {ab{c I d}}] (RegGr)

— Test **Coverage**/Optimization problem

- *Positive tests:* Construct a set of *CES* as test sequences of minimal total length to cover all ES of length *n (n=2,3,4,...).*
- *Negative tests:* Construct a minimal set of sub-complete paths (sub-walks) as of minimal total length to cover all FES of length *n=2.*
- This problem is a special form of the *Chinese Postman Problem*.

Event-Algebraic Modeling, Analysis and Extension for *Fault Tolerance*



SUC: System under Consideration
FSA: Finite State Automaton
RegEx: Regular Expression

2. Modeling, Analysis - *Toward Self-Testing and Self-Correcting*

Hypotheses for correcting faults in an *faulty event sequence* (*FES*):

. . . .   s   u   t . . .

I (insert)   v   . . . . .   (*I-Hypo*: ***insert*** v between s and u, which was wrongly *omitted*)

R (replace)   . . z   . . .   (*R-Hypo*: ***replace*** u by z, which was wrongly *corrupted*)

D (delete)   . . X   . . . .   (*D-Hypo*: ***delete*** u, which was wrongly *inserted*)

- s, u, t, v, z: symbols of an alphabet interpreted as *events* (user commands/inputs, system prompts) represented as *nodes* of an FSA.

- The hypotheses can be extended from one single error to *n* errors:
  - I^n*-errors*: *n* elements have to be *inserted*.
  - D^n*-errors*: *n* elements have to be *omitted*.
  - R^n*-errors*: *n* elements have to be *replaced*.

- Appropriate combination of these hypotheses represents arbitrary types of faults.

- Example: "*an event has been (illegally) forgotten, or inserted, or two transitions have been interchanged*" can be represented by $I + D^2 + R$

“PQ”-Fault Tolerance

- A hypothesis P$\in\{$I,R,D$\}$ that can be applied exactly to one position is called *P-detecting*.
- The position where the correction will be carried out is called *correction position*, the involved symbol is called *correcting symbol*.
- If exactly one correcting symbol can be involved in a P-Correction, then we have a *P-correcting FES*.
- If the hypotheses P,Q$\in\{$I,R,D$\}$ exclude each other, than we have a *PQ-independent* FES.
- If a FES is P-detecting and P-correcting, and moreover if all hypotheses P and Q are pairwise independent, then we have a *PQ-**self**-correcting* (*PQ-fault-tolerant*) FES.
- Generalization of the hypotheses through considering faults caused by n symbols (n>1) extends fault handling.

Event Analysis – FSA for Forwards *Scanning* of a RegEx

Example: $T = [(ba(b+c^*)^*(a+b)^*]$

- Index T for **tracing the contextual position**

  $T' = [^1(b^1a^1(b^2+c^1)^*)^*(a^2+b^3)^*]^1$

- Determine the states and state transitions of an FSA that accepts T'.

  (a)i=:j (input **a** transfers the automaton from state **i** into state j)

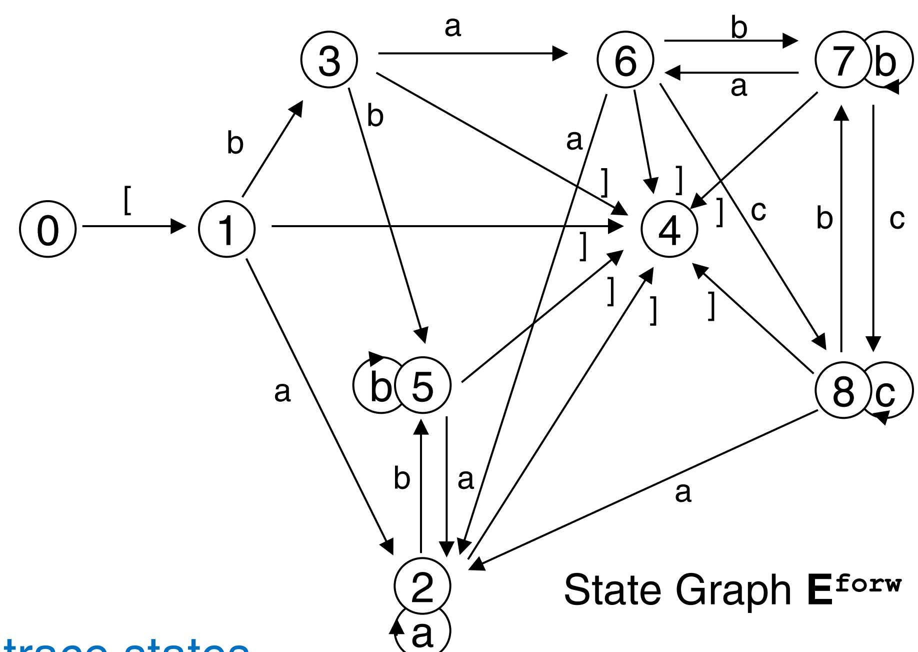
initial state = 0
$(a)1 = (a^2)1 =: 2 \equiv a^2$
$(])1 = (]^1)1 =: 4 \equiv\,]^1$
$(b)2 = (b^3)2 =: 5 \equiv b^3$
(a)3 = ... etc.

$([)0 = ([^1)0 =: 1 \equiv [^1$
$(b)1 = (b^1 + b^3)1 =: 3 \equiv b^1 + b^3$
$(a)2 = (a^2)2 = a^2 \equiv 2$
$([)2 = (]^1)2 = (]^1) = 4 \equiv\,]^1$
Final State = 4

- Construct the automaton **E^forw**

	z	[	a	b	c	]
-	0	1				
$[^1$	1		2	3		4
a^2	2		2	5		4
$b^1 + b^3$	3		6	5		4
$]^1$	4					
b^3	5		2	5		4
$a^1 + a^2$	6		2	7	8	4
$b^1 + b^2 + b^3$	7		6	7	8	4
c^1	8		2	7	8	4

State Table **E^forw**



State Graph **E^forw**

- Scan T *forwards* through **E^forw**, trace states.

$T^{forw} = [^1(b^{3+7}a^6(b^7+c^8)^*)^*(a^{2+6}{}_{+}b^{3+5+7})^*]^4$

Coding: Simultaneous Forwards and Backwards Scanning of RegEx

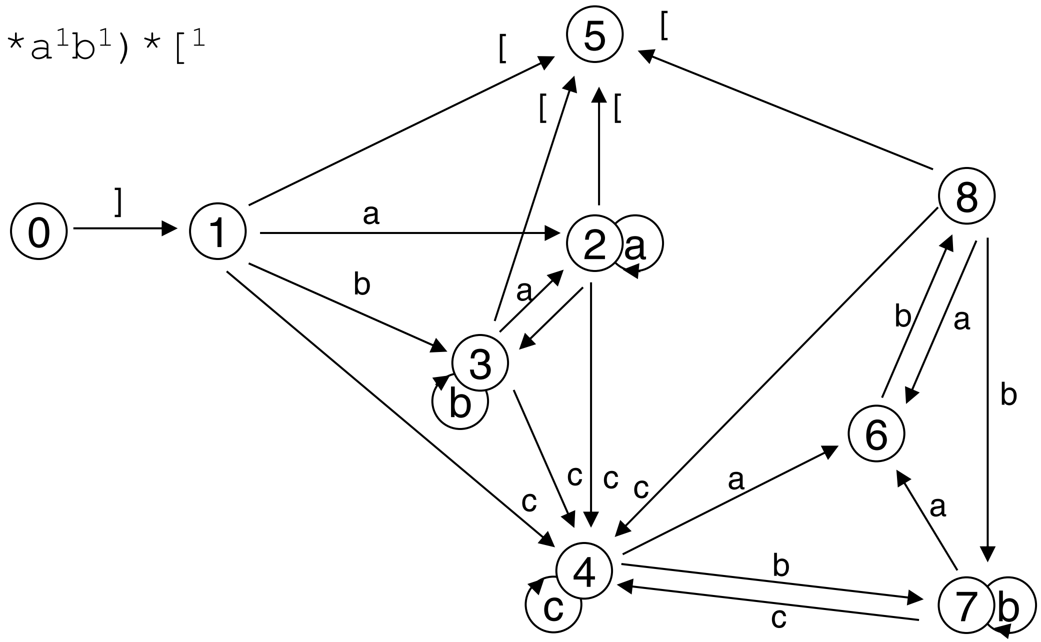
- Reverse (*mirror*) *of* T'

  $T^{mirr} =]^1(b^3+a^2)*((c^1+b^2)*a^1b^1)*[^1$

- Construct the automaton $\mathbf{E_{back}}$

	z*	[	a	b	c
-	0				
$]^1$	1	5	2	3	4
$a^1 + a^2$	2	5	2	3	4
$b^1 + b^2 + b^3$	3	5	2	3	4
c^1	4		6	7	4
$[^1$	5				
a^1	6			8	
b^2	7		6	7	4
b^1	8	5	6	7	4

State Table $\mathbf{E_{back}}$



State Graph $\mathbf{E_{back}}$

- Scan T^{mirr} *forwards* through E_{back}, trace states.

  $T^{mirr\_forw} =]^1(b^3+a^2)((c^4+b^{3+7})*a^{2+6}b^{3+8})*[^5$

- Reverse (*mirror*) T^{mirr_forw}, subscribe the indices.

  $T^{mirr\ forw\ mirr} = [_5(b_{3+8}a_{2+6}(b_{3+7}+c_4)*)*(a_2+b_3)*]_1 =: T_{back}$

- Note forwards and backwards indices simultaneously for *Coding* T.

  $$T^{forw}_{back} = [^1_5(b^{3+7}_{3+8}\ a^6_{2+6}(b^7_{3+7}+c^8_4)*)*(a^{2+6}_2+b^{3+5+7}_3)*]^4_1$$

- T^{forw}_{back} delivers all information needed for a complete fault analysis and redundancy determination for fault tolerance.

Characteristic Features of T

- *Compability* ***Relation C:***

  C= э { (i,j) э S(E^{forw}),
  sj э S(E_{back}) } => isj, iCj.

- C-Table of T"

i [j	i a j	i b j	i c j	i] j
i [5	2 a 2	3 b 3	8 c 4	4] i
	6 a 2	3 b 8		
	6 a 6	5 b 3		
		7 b 3		
		7 b 7		
		7 b 8		

- *Right and Left context* ***relations*** of T".

$r" := r^{forw}$; $l = l_{back}$

l"	s"	"r"	l,,	s,,	r,,
---	$[^1$	$a^2+b^3+]^4$	---	$[_5$	$a_2+b_3+b_8+]_1$
$[^1+a^2+a^6+b^5+c^8$	a^2	$a^2+b^5+]^4$	$[_5+a_2+b_3+c_4$	a_2	$a_2+b_3+]_1$
b^3+b^7	a^6	$a^2+b^7+c^8+]^4$	b_8	a_6	$b_7+b_8+c_4$
$[^1$	b^3	$a^6+b^5+]^4$	$[_5+a_2+b_3+c_4$	b_3	$a_2+b_3+]_1$
$a^2+b^3+b^5$	b^5	$a^2+b^5+]^4$	$a_6+b_7+c_4$	b_7	$b_7+b_8+c_4$
$a^6+b^7+c^8$	b^7	$a^6+b^7+c^8+]^4$	$[_5+a_6+b_7+c_4$	b_8	a_6
$a^6+b^7+c^8$	c^8	$a^2+b^7+c^8+]^4$	$a_6+b_7+c_4$	c_4	$a_2+b_3+b_7+b_8+c_4+]_1$
$[^1+a^2+a^6+b^3+b^5+b^7+c^8$	1^4	---	$[_5+a_2+b_3+c_4$	$]_1$	---

Analysis of an event sequence (ES)

- Coding of an ES by E^{forw}_{back}

$$ES^{forw}_{back} = [^1_f b^3_7 b^5_7 b^5_8 a^2_6 c^e_4 a^e_2 b^e_3 b^e_3 a^e_2],$$

ES is a faulty ES, i.e., FES, with e and f as faulty states of E^{forw} and E_{back}, respectively.

- Ambiguities hinder Self-Correction.

<-------- Correction Area -------->

. . . $s^j_j u^k_i t^m_n$ . . . $v^a_b x^c_d y^g_h$ . . .

R: . . . . $z^{i'}_{n'}$ . . . . $k^{a'}_{h'}$

Two R corrections.

- Within a *correction area*, a fault detection (if any) will be carried out only between two faulty symbols.

$k^{forw}_{back} = b^3_7 \; b^5_7 \; b^5_8 a^2_6 c^e_4$

I . . . . . a^6_6 . . . . (as $a^6 \in r^{forw}(b^3) \wedge a_6 \in l_{back}(b_7)$)

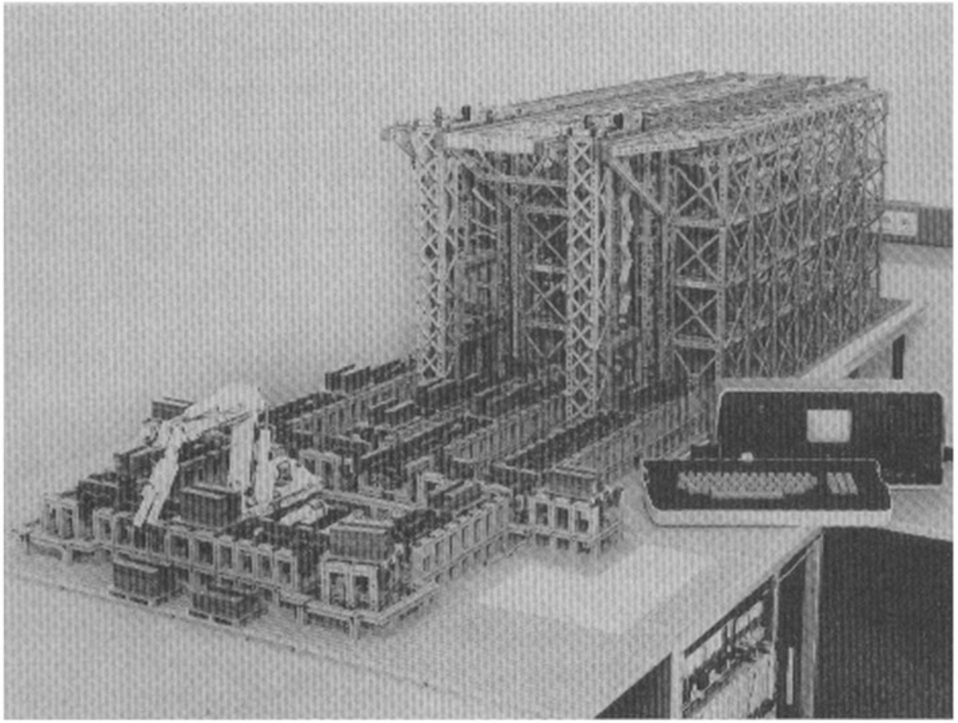
R . . . . a^6_6 . . . . $a^2_2 + b^5_3$ (analogous)

D . . . . x (analogous)

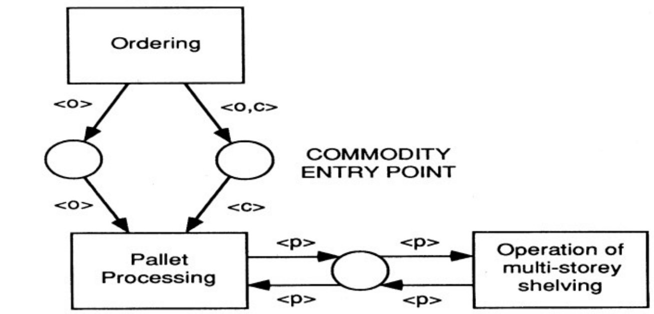
T = [(ba(b+c*)*(a+b)*] is

- **not R-detecting,**
- **not R-correcting,**
- **not IR-/ID-/RD-independent.**

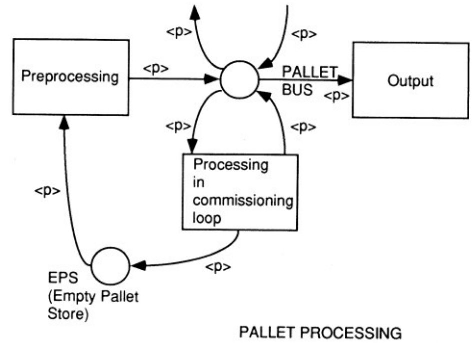
3. Example I: An Interactive System



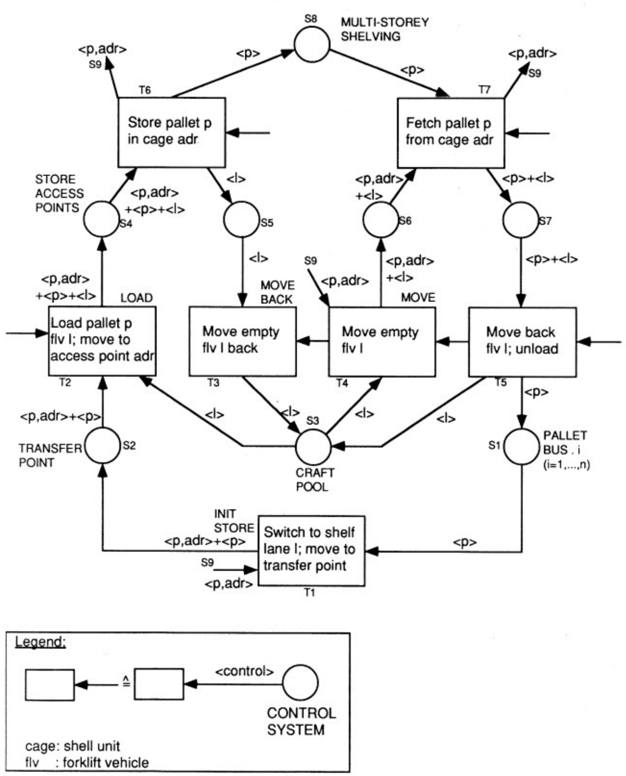
General view of the model of system under consideration (SUC) –
an automated commodity storage and multi-storey shelving system
(Hochschule Bremerhaven)



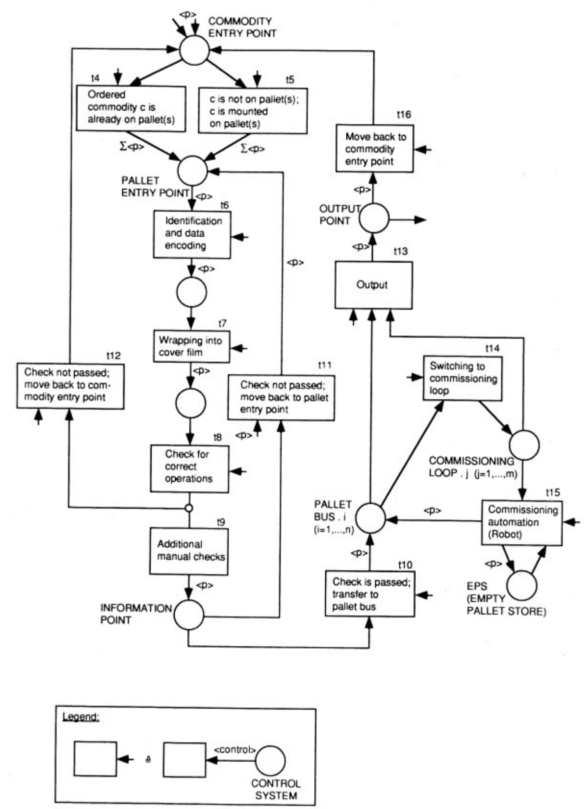
(a) The PrT net model of SUC (topmost hierarchical level)



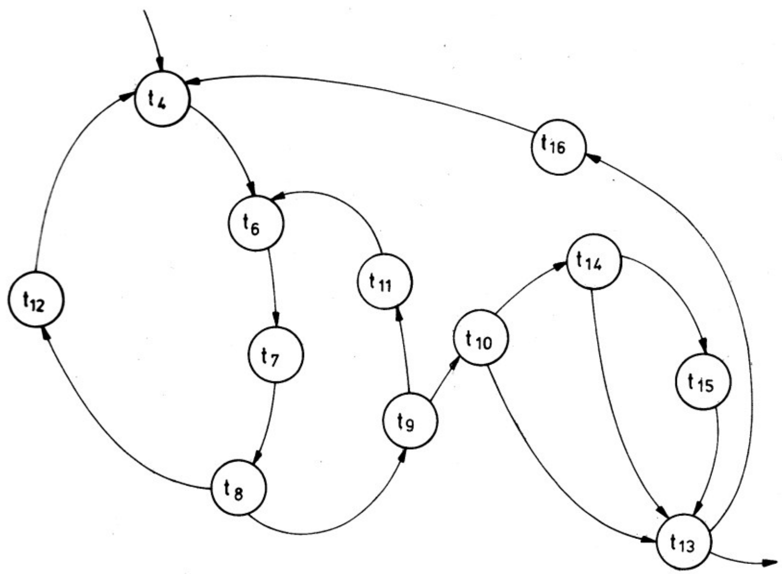
(c) Refinement of the transition „pallet processing“ (see (a))



(b) Refinement of the transition „multistorey shelving“ (see (a))

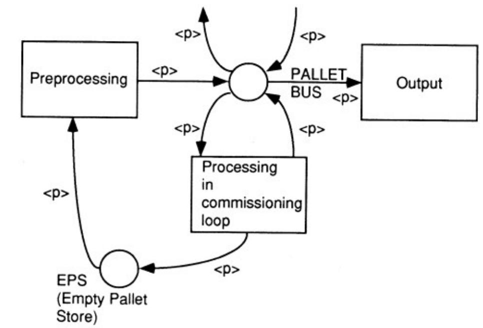


(d) Refinement of the transition „pallet processing“ (see (a))

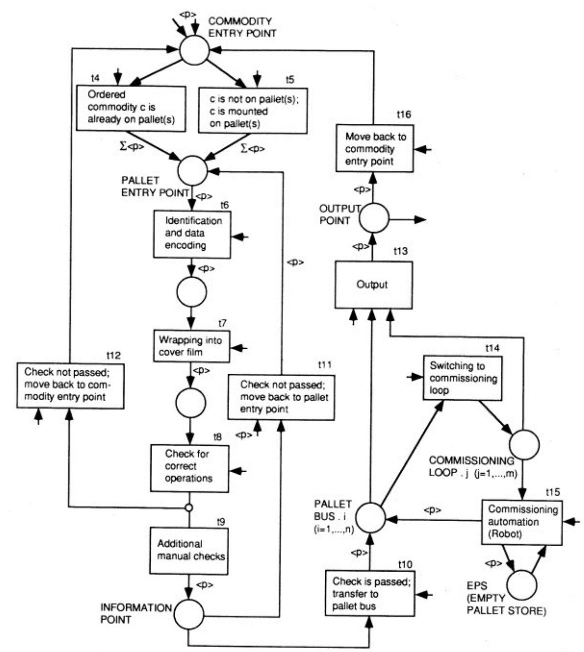


(a)

ESG as instantaneous, sequential record of (d)

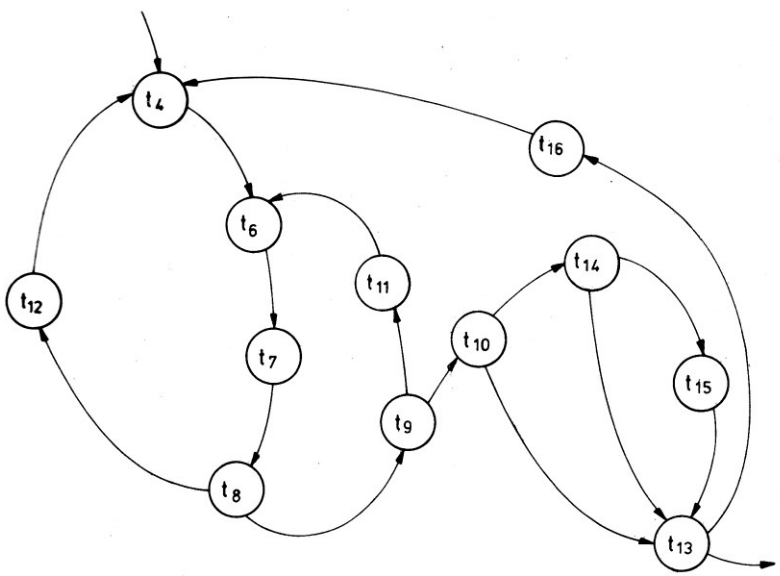


(c) Refinement of the transition „pallet processing“ (see (a))

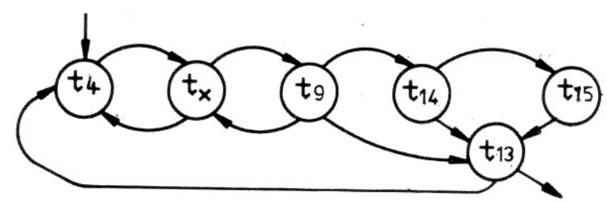


(d) Refinement of the transition „pallet processing“ (see (a))

$tx \coloneqq t6\,t7\,t8.$	Merging the Events
$t8$: includes the adjacent state $t12$ $t9$: includes the adjacent states $t10$ and $t11$ $t13$: includes the adjacent state $t16$	Further Mergings for simplifiying ESG
$T = \langle t4tx\langle(t4+t9)tx\rangle t9(t13+t14t13+t14t15t13)\rangle$	Regular Expression of the simplified ESG
$T^{forw}_{back} = \langle t4^8_3 tx^7_4 \langle (t4^5_8 + t9^6_6) tx^4_7 \rangle t9^6_6 (t13^7_3 + t14^8_5 t13^7_3 + t14^8_5 t15^9_4 t13^3_7)\rangle.$	Not R-Correcting!
$tx \coloneqq tx\ end$	Extension for R-Correcting
$\hat{T} = \langle t4tx\ end\langle(t4+t9)tx\ end\rangle t9(t13+t14t13+t14t15t13)\rangle.$	Extended Term for R-Correcting

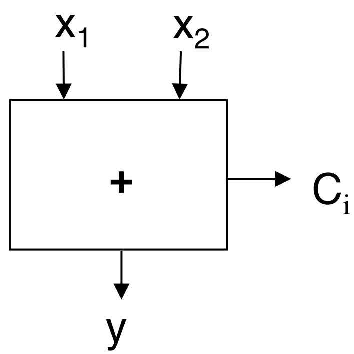


(a)

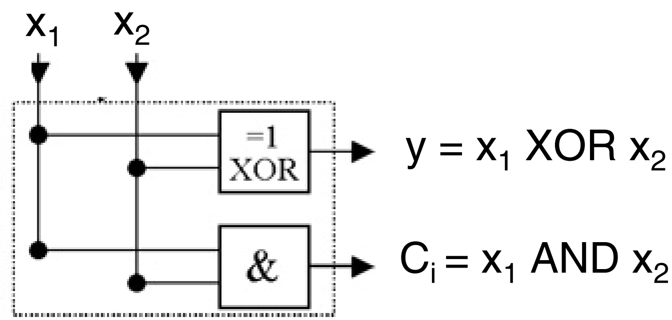


(b)

3. Example II: Serial Adder



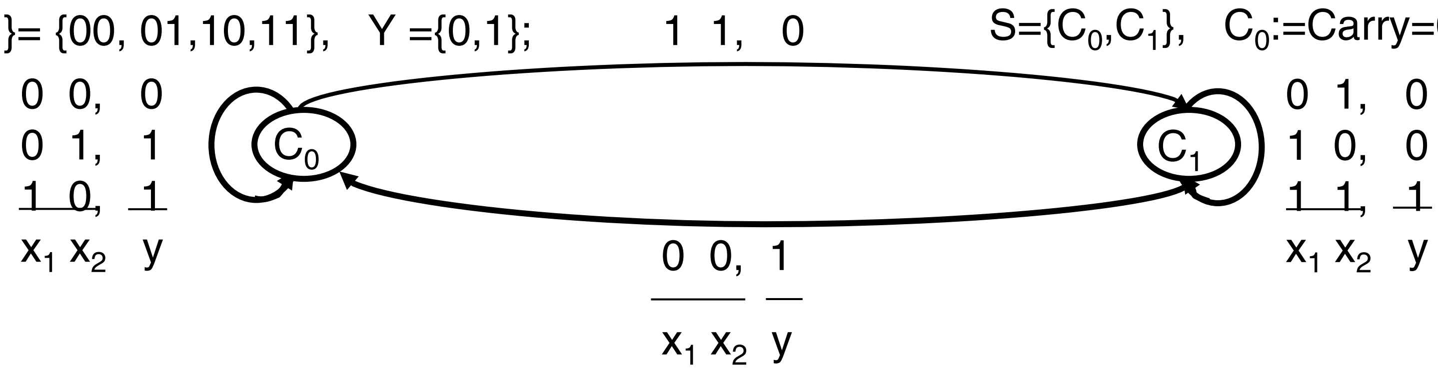
x_1	x_2	y	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



State Diagram (Mealy Automaton)

$X=\{x_1x_2\}= \{00, 01,10,11\}$, $Y =\{0,1\}$;

$S=\{C_0,C_1\}$, C_0:=Carry=0, C_1:=Carry=1



RegEx of Serial Adder: Plus = (000+011+101+110(010+100+111)*001)*

Coded RegEx: Plus“ = $(\,0^{1}_{1}0^{3}_{3}0^{7}_{7}+0^{1}_{1}1^{4}_{4}1^{8}_{8}+1^{2}_{2}0^{5}_{5}1^{8}_{8}+1^{2}_{2}1^{6}_{6}0^{9}_{9}(0^{10}_{10}1^{6}_{6}0^{9}_{9}+1^{11}_{11}0^{12}_{12}0^{9}_{9}+1^{11}_{11}1^{12}_{12}1^{14}_{14})^{*}0^{10}_{10}0^{5}_{5}1^{8}_{8}\,)^{*}$

$\ldots\ 0^{9}\ \ 1^{11}_{14}\ \ 1^{13}_{18}\ \ 1^{14}_{\uparrow 2}\ \ 1^{11}_{\uparrow 6}\ \ 1^{13}_{\uparrow}\ \ 0_{9}\ \ 1_{11}\ \ldots$

R: $\ldots\ldots\ldots\ \ 0^{10}\ \ 0^{12}\ \ 1^{14}$

Fault-Detection / Correction on a faulty sequence

- **Serial adder is not R-self-correcting.**
- **(Obviously) o_9 is the only faulty event that stems from an operational fault.**
- **Simple operational faults are (obviously) R-correcting.**

Summary

* **What is this all about?**

Modeling, Analysis, and Extension for Fault Tolerance.

- **Whether/How can the knowledge be used in daily work?**

(i) Fault handling incl. testing, in sequential systems – both hard- and software.

(ii) Figuratively speaking, the goal: "Define me a fault, and I will construct a test case to detect it – and tell you how to make the system immune to this fault"

* **What are the next problems to be solved?**

(i) Optimizing the redundancy for self-fault detection/correction,

(ii) extending/combining fault classes, improving scalability,

(iii) considering distributiveness and concurrency.

- **Who/Whose work could help?**

(i), (iii) Theoretical computer science, OR (multi-variate optimization),

(ii) Empirical research, circuit theory/practice.

* **Impacts on other work?**

(i) Software/Hardware co-design and co-validation,

(ii) Developing and validation of high assurance, safe systems.

REFERENCES

(1) J. A. Brzozowski, *Regular Expression Techniques for Sequential Circuits*, PhD Dissertation, University of Princeton, 1962.

(2) F. Belli, *Extension of Regular Languages for Self-Detecting and Self-Correction of Syntactic Errors* (in German), *R. Oldenburg Verlag*, Munich, 1978.

(3) F. Belli, Großpietsch, K.-E., Specification of Fault-Tolerant System Issues by Predicate/Transition Nets and Regular Expressions – Approach and Case Study, *IEEE Transactions on Software Engineering*, Vol. 17, 513-526, 1991.

(4) F. Belli, Takeshi Endo, A., Linschulte, M., Simao, A., A Holistic Approach to Model-Based Testing of Web Service Compositions, *Software: Practice and Experience*, vol.44, no.2, pp. 201–234, 2013.

(5) F. Belli, Beyazıt, M., Takeshi Endo, A., Mathur, A., Fault Domain-Based Testing in Imperfect Situations: A Heuristic Approach and Case Studies, *Software Quality Journal*, pp. 1–17; DOI10.1007/s11219-014-9242-6, 2014.

(6) F. Belli, Beyazıt, M., Exploiting Model Morphology for Event-Based Testing, to appear in *IEEE Transactions on Software Engineering*, 2015.