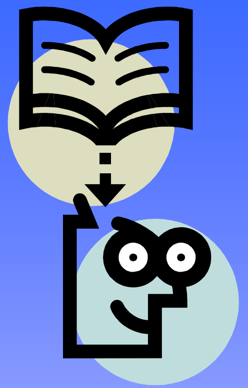
Operating statement

Cybernetix Case Study,
New ETMCC,
CSL model checking improvements

Ivan Zapreev

Chronology



- **Aug. 2004 – Current**

  MCC model checker

- **Jun. 2004 – Aug. 2004**

  Cybernetix Case Study

- **Mar. 2004 – Jun. 2004**

  Learning literature

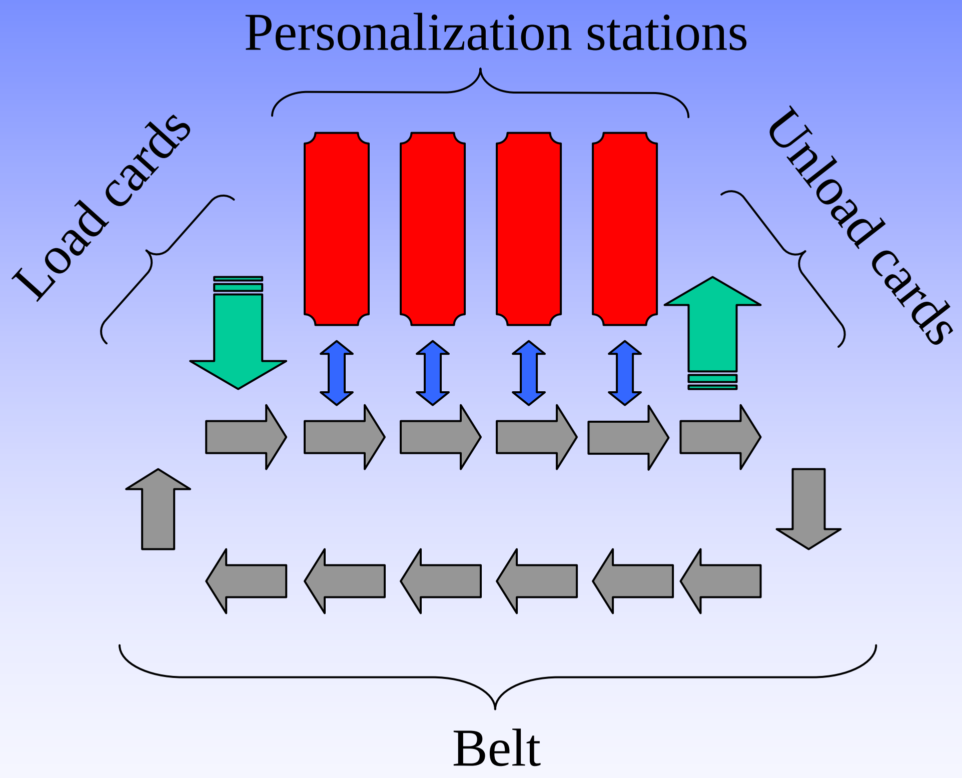
The Cybernetix Case Study. Probabilities and Non-determinism.

Ivan Zapreev

Outline

- The Cybernatix Case Study
- The main interest
- Considered Models
- Conclusions
- Future works

The Cybernatix Case Study



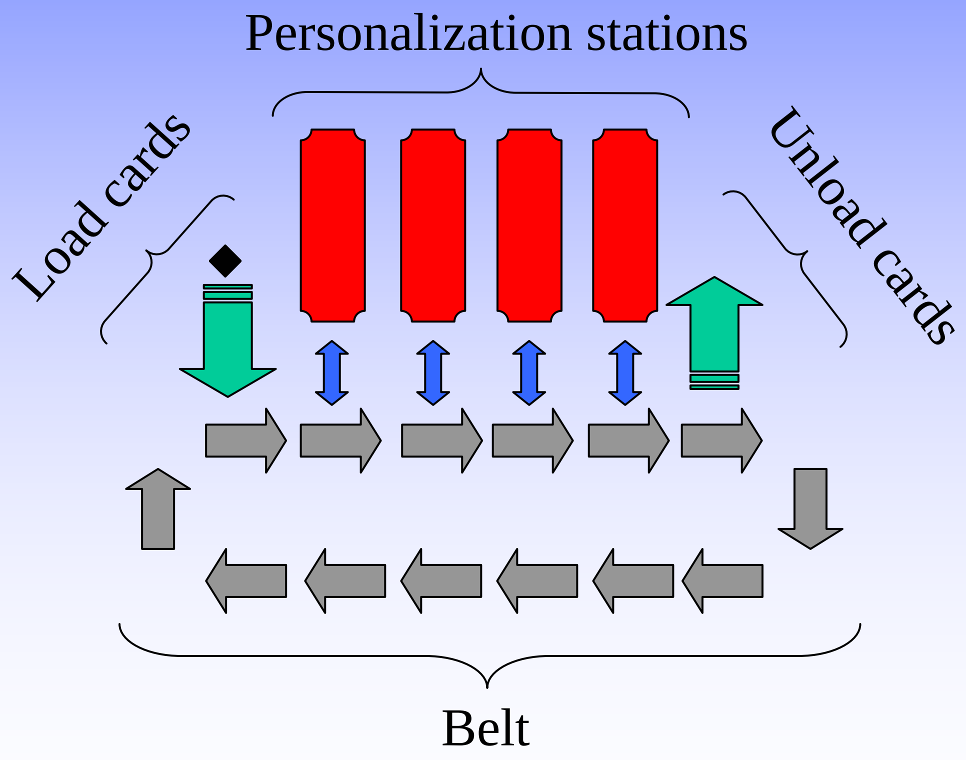
Main interests

- Involve *probabilistic* and/or *non-deterministic* failures
- Types of failures
  - A card can be broken
  - A personalization station can be broken

P(M of N cards are broken) = ?

Super Single Mode

- Do everything as fast as you can, and leave free space for personalized cards.
- Give uniform loading of stations



Considered models

A.S. ⇨ 1. Each card can be broken with a constant probability.

2. Each card can be broken with an increasing probability. Weibull distribution.

3. Station breaks with a constant probability and breaks [0,…,K] cards. Non determinism.

4. Station breaks with a constant probability and breaks a constant number of cards.

A.S. ⇨ 5. Station can break while working, needs constant time to be repaired.

6. A simple DTMC, there is a probability to break and a probability to be repaired

Type I discrete Weibull distribution

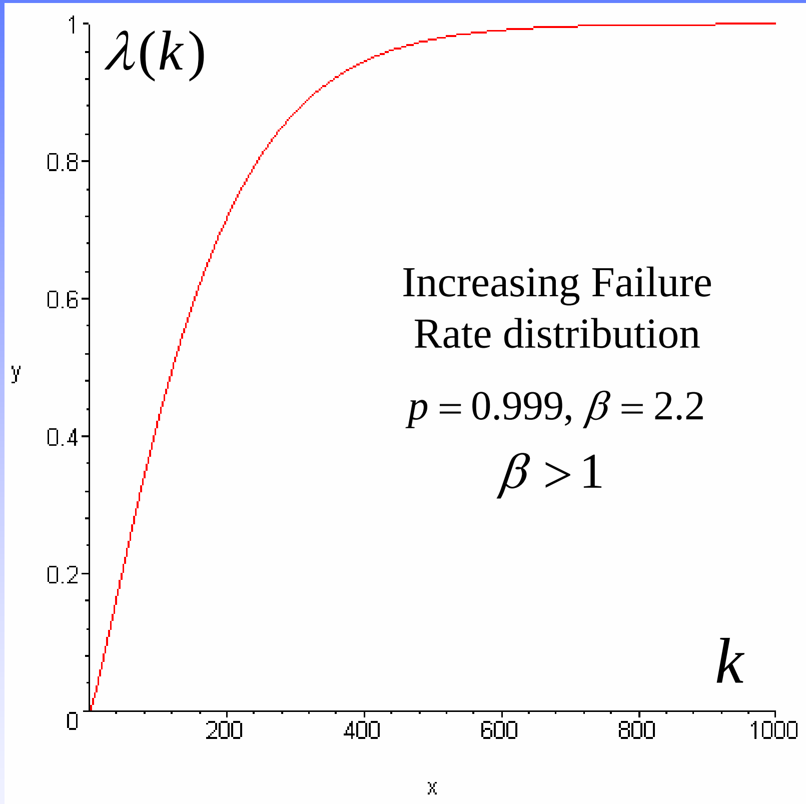
$R(k) = p^{k^{\beta}}$ - reliability function

$\lambda(k) = 1 - p^{k^{\beta} - (k-1)^{\beta}}$ - failure rate

$$p \in \left]0,1\right[, \beta > 0, k \in N^{*}$$

$P(k) = P(K = k)$ - probability of failure at demand k

$R(k) = P(K > k)$ - probability not to fail during k demands

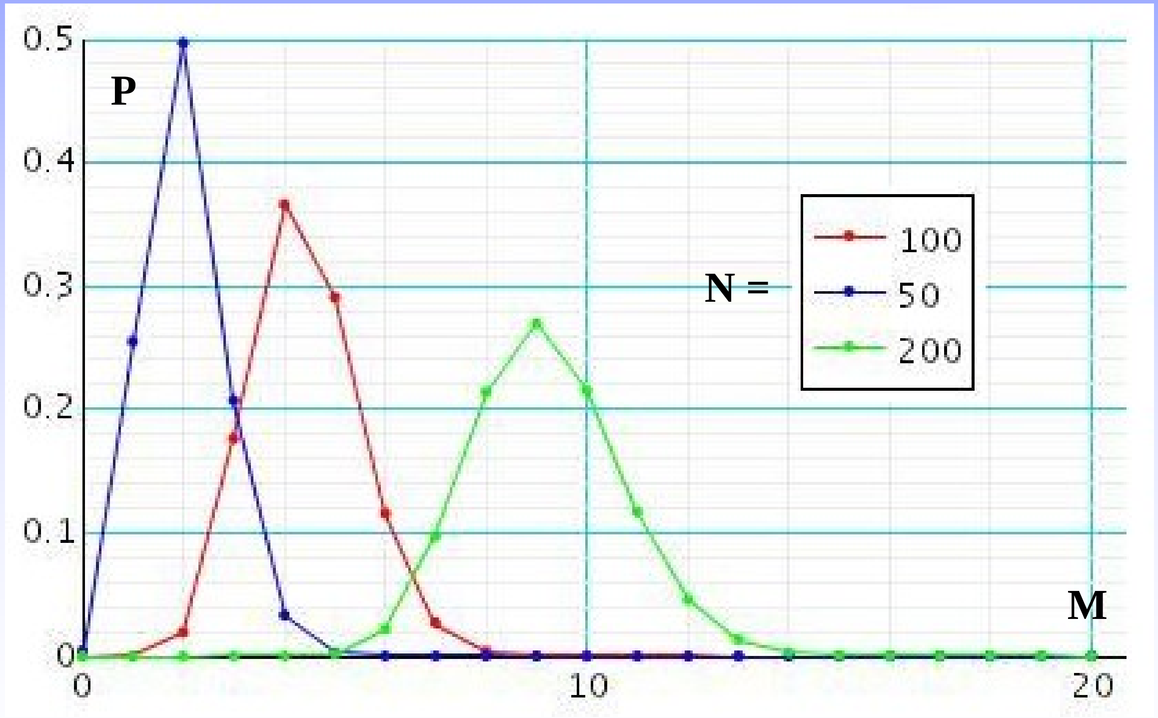


$$\lambda(k) = P(K = k \mid K \geq k) = \frac{P(k)}{R(k-1)}$$

- probability to fail at demand k if it did not fail before.

Simple Failure Model with Weibull distribution of failures

The model: Each card can be broken while personalization with the probability $\lambda(k)$ with $p = 0.999, \beta = 2.2$ where k is the number of cards, correctly personalized, by the given station since it broke card for the last time.



One station results generalization

- Uniform stations loading (SSM)
- Independent station failures

⇒

- ~~Time~~
- ~~Loading station~~
- ~~Unloading station~~
- ~~Belt~~
- ~~4 personalization stations~~

$$P_T^R\left(M \;\; of \;\; N \;\; cards \;\; are \;\; broken\right) = \sum_{M_1+\ldots+M_N=M} \prod_{i=1}^{R} P_T^1\left(M_i \;\; of \;\; \frac{N}{R} \;\; cards \;\; are \;\; broken\right)$$

N - the total amount of cards

R - the number of stations, is the divisor of N

$T \in \{\min, \max, \_\}$

P_T^1 - the probability for one station

Conclusions

- Different failure models were investigated,
- Analytical solutions were discovered,
- “One station” ⇒ “Any number of stations”;

Future works

Check whether the SSM still provides optimal throughput of good cards when probabilistic failures are involved.

Model:

- Non determinism on the level of scheduling
- Probabilistic breakings of personalization stations

Use Prism tool customized for finding an optimal schedule.

MCC model checker.
A reincarnation of ETMCC.

Ivan S Zapreev,

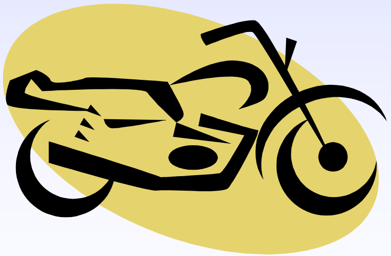
Maneesh Khattri

Outline

- Goals
- PRCTL & CSRL
- Details
- ETMCC vs. MCC
- Conclusions
- Future works

Goals

- Develop a unified framework for **PCTL**, **CSL**, **PRCTL** and **CSRL,**
- Make it work faster than ETMCC,
  - Use more efficient data structures,
  - Use improved algorithms for **CSL,**
    - Steady state detection,
    - Faster until operators,
    - Faster BSCCs search,
    - ETC….;



PRCTL & CSRL

PRCTL:

$$\phi ::= true \mid a \mid \phi \wedge \phi \mid \neg \phi \mid L_{\triangleright\triangleleft p}[\phi] \mid \mathrm{P}_{\triangleright\triangleleft p}[\phi\, U_J^I\, \varphi] \mid$$

$$\mathrm{E}_J^n(\phi) \mid \mathrm{E}_J(\phi) \mid C_J^n(\phi) \mid \mathrm{Y}_J^n(\phi)$$

$$n \in \mathrm{N},\ I \subseteq \mathrm{N} \cup \{\infty\},\ p \in [0,1],\ J \subseteq R_{\geq 0}$$

CSRL:

$$\phi ::= true \mid a \mid \phi \wedge \phi \mid \neg \phi \mid L_{\triangleright\triangleleft p}[\phi] \mid \mathrm{P}_{\triangleright\triangleleft p}[\phi\, U_J^T\, \varphi]$$

$$T \subseteq R_{\geq 0},\ p \in [0,1],\ J \subseteq R_{\geq 0}$$

Details

- Data structures:
  - Sparse Matrix special representation,
  - Fast Matrix Vector multiplication,
  - Linear memory allocation,
  - Predecessor sets,
- Algorithms:
  - Direct search only for required BSCCs,
  - Efficient algorithms for bounded until,
  - Efficient algorithms for unbounded until,
  - Collapse $\varphi \,\&\, \neg\phi \wedge \neg\varphi$ states,
  - On the fly steady state detection,
  - Store transient state probabilities of reaching BSCCs,
  - Bisimulation minimization;

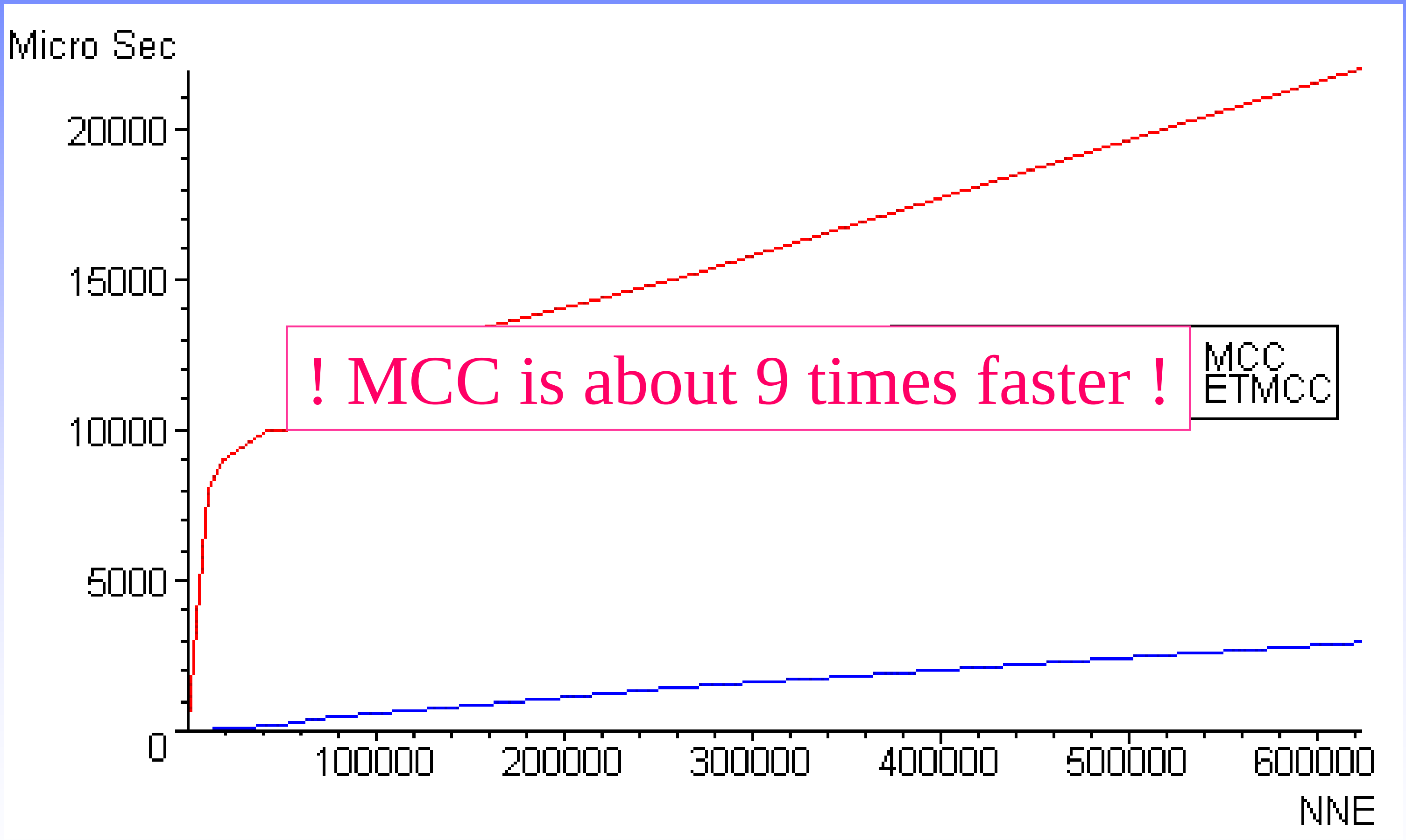
Data Structure

- Make states absorbing
- Compute Uniformized DTMC from CTMC

$$A = \begin{bmatrix} 0.5 & 0.15 & 0.0 \\ 0.25 & 0.0 & 0.75 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \Rightarrow \begin{bmatrix} Ncols = 2 \\ Diag = 0.5 \\ Column \to [2] \\ Value \to [0.15] \\ \hline Ncols = 2 \\ Diag = 0 \\ Column \to [1, 3] \\ Value \to [0.25, 0.75] \\ \hline Ncols = 0 \\ Diag = 0 \\ Column \to NULL \\ Value \to NULL \end{bmatrix}$$

ETMCC vs. MCC

Matrix vector multiplication:

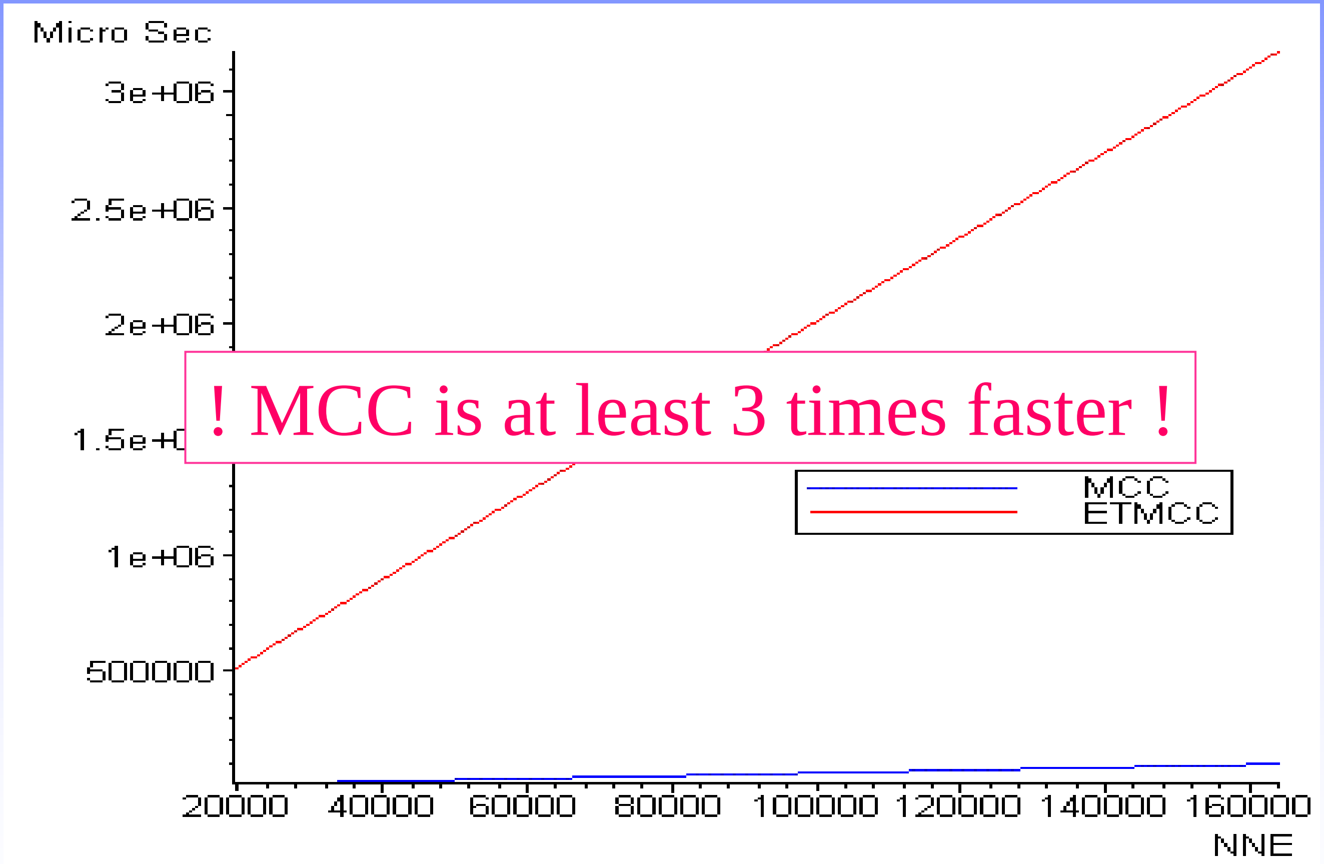


ETMCC vs. MCC

The Cluster Computing example

$$\mathrm{P}_{\triangleright\triangleleft p}\left(\phi\, U^{\leq t}\varphi\right)$$

http://www.cs.bham.ac.uk/dxp/prism/cluster.html



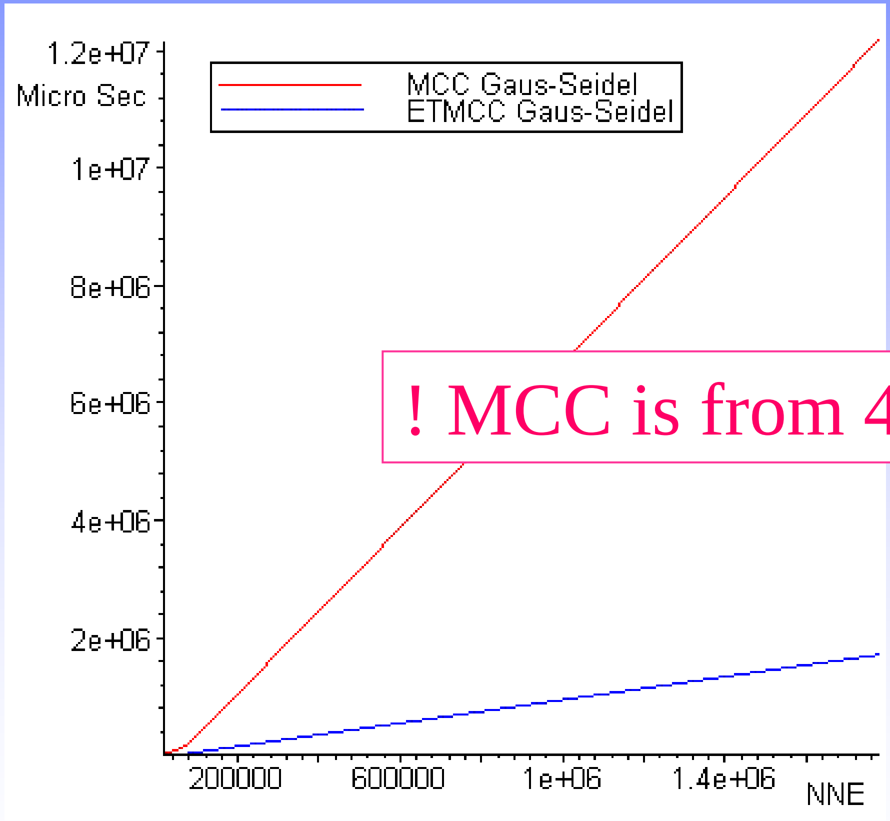
! MCC is at least 3 times faster !

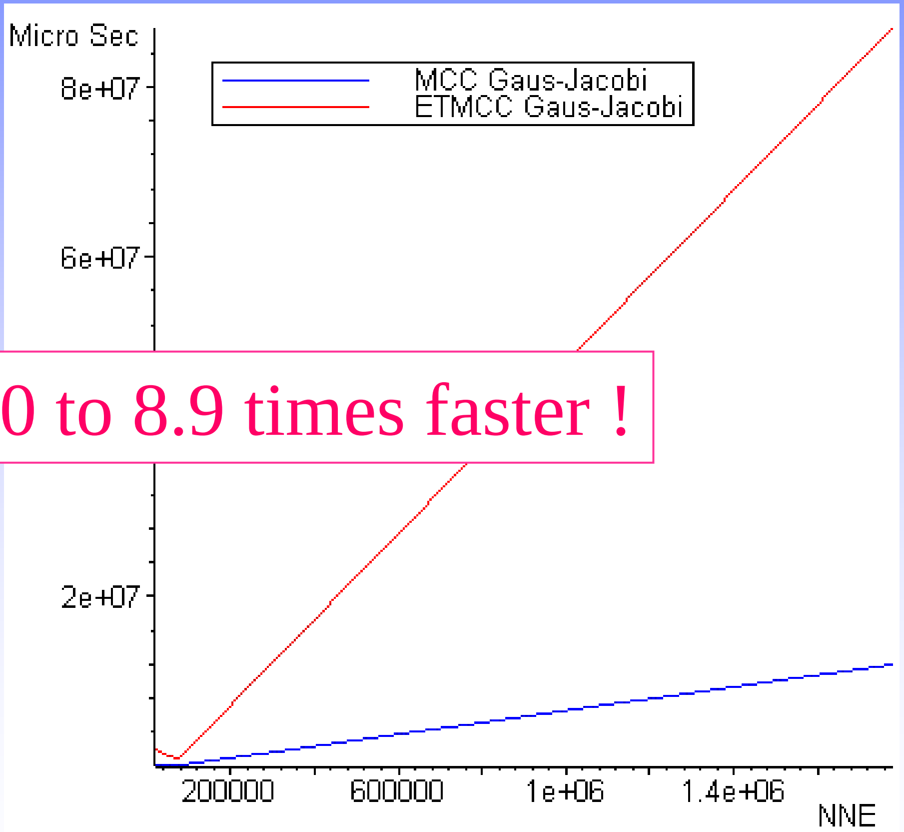
ETMCC vs. MCC

The Cluster Computing example

$S_{\triangleright\triangleleft p}(\phi)$

http://www.cs.bham.ac.uk/dxp/prism/cluster.html





! MCC is from 4.0 to 8.9 times faster !

Conclusions

- The PCTL, CSL, PRCTL logics are supported,
- The implementation is several times faster,
- There are still ways for further improvements;

Future works

- Incorporate CSRL logic model checking,
- Work on future improvements of algorithms for CSL,
  - On the fly steady state detection,
  - Store transient state probabilities of reaching BSCCs,
  - Collapse $\varphi \,\&\, \neg\phi \wedge \neg\varphi$ states,
  - Involve bisimulation minimization;

CSL model checking improvements

Ivan S Zapreev,

Maneesh Khattri

Outline

- CSL logic
- The $\phi U^{\leq t}_{\lhd p} \varphi$ operator
- Krylov Subspaces
- Matrix exponent estimate
- The $\phi U^{\leq t}_{\lhd p} \varphi$ estimate
- Conclusions
- Future works

CSL logic

The syntax of CSL:

$$\phi ::= true \mid a \mid \phi \wedge \phi \mid \neg\phi \mid S_{\triangleright\triangleleft p}[\phi] \mid \mathrm{P}_{\triangleright\triangleleft p}[\varphi]$$

$$\varphi ::= \mathrm{X}\phi \mid \phi \bigcup^{\leq t} \phi \mid \phi \bigcup \phi$$

Where

$$p \in [0,1]\,,\; t \in R_{\geq 0},\; \triangleright\triangleleft \in \{\geq, \leq\}$$

The $\phi \, U^{\le t}_{\triangleright\triangleleft p} \, \varphi$ operator

The $\mathrm{Prob}\left(\phi \, U^{\le t}_{\triangleright\triangleleft p} \, \varphi\right)$ can be computed as:

- The solution:

$$\mathrm{Prob}^{\mathrm{M}}\left(\phi \ \mathrm{U}^{\le \mathrm{t}} \ \varphi\right) = \sum_{s''} \pi^{M[\neg\phi\vee\varphi]}(s, s'', t)$$

$$\pi^{M[\neg\phi\vee\varphi]}(s, s'', t) = \mathrm{Prob}^{\mathrm{M}[\neg\phi\vee\varphi]}\left(s, \Diamond^{[t,t]} at_{s''}\right)$$

- Using uniformisation:

$$\mathrm{Prob}\left(\phi \, U^{\le t}_{\triangleright\triangleleft p} \, \varphi\right) = e^{q\cdot t\cdot(P-I)} \cdot \overrightarrow{i_{\varphi}} = \sum_{k=0}^{\infty} e^{-qt} \frac{(q\cdot t)^k}{k!} P^k \cdot \overrightarrow{i_{\varphi}}$$

Krylov Subspaces

The probability

$$\pi^{M[\neg\phi\vee\varphi]}\left(s, s'', t\right) = \text{Prob}^{\text{M}[\neg\phi\vee\varphi]}\left(s, \Diamond^{[t,t]} at_{s''}\right)$$

is just a transient probability and is obtained as a solution of a differential equation. The solution is given in the for of matrix exponent.

Krylov-based algorithm:

Computes matrix exponential multiplied by vector at once as an approximation of *w(t)* by mapping the solution onto a much smaller subspace.

$$w(t) = e^{t\cdot A}\cdot\vec{v}$$

Matrix exponent estimate

Unfortunately this estimate
does not work ☹

$$P_{\min} = \begin{pmatrix} \min_k\left(p_{k,1}\right) & \cdots & \min_k\left(p_{k,N}\right) \\ \vdots & \ddots & \vdots \\ \min_k\left(p_{k,1}\right) & \cdots & \min_k\left(p_{k,N}\right) \end{pmatrix} \qquad P_{\max} = \begin{pmatrix} \max_k\left(p_{k,1}\right) & \cdots & \max_k\left(p_{k,N}\right) \\ \vdots & \ddots & \vdots \\ \max_k\left(p_{k,1}\right) & \cdots & \max_k\left(p_{k,N}\right) \end{pmatrix}$$

$$\mathrm{Prob}\left(\phi\, U^{\le t}_{\rhd\lhd p}\, \varphi\right) = e^{q\cdot t\cdot(P-I)} \cdot \vec{i}_{\varphi} = e^{-qt} \cdot e^{q\cdot t\cdot P} \cdot \vec{i}_{\varphi}$$

$$P = \left(p_{i,j}\right) \quad \& \quad \sum_{j=1}^{N} p_{i,j} = 1$$

$$\mathrm{Prob}\left(\phi\, U^{\le t}_{\rhd\lhd p}\, \varphi\right) \le e^{-q\cdot t} \cdot I \cdot \vec{i}_{\varphi} + P_{\max} \cdot \left(1 - e^{-q\cdot t}\right) \cdot \vec{i}_{\varphi}$$

$$e^{-q\cdot t} \cdot I \cdot \vec{i}_{\varphi} + P_{\min} \cdot \left(1 - e^{-q\cdot t}\right) \cdot \vec{i}_{\varphi} \le \mathrm{Prob}\left(s, \phi\, U^{\le t}_{\rhd\lhd p}\, \varphi\right)$$

The $\text{Prob}\left(\phi\, U_{\triangleright\triangleleft p}^{\leq t}\, \varphi\right)$ estimate

Idea 1: In DTMC consider all transitions leading to φ state only once and collect probability only of some paths. This will give us some Min_{φ} estimate.

Idea 2: In DTMC compute Min estimate of reaching $\neg\phi \wedge \neg\varphi$ state. This will give us max estimate $Max_{\varphi} = 1 - Min_{\neg\phi\wedge\neg\varphi}$

Idea 3: The ideas 1 & 2 give Min and Max estimate for DTMC. The estimates for CTMC can be gathered with the using the formula & Fox-Glynn algorithm:

$$\text{Prob}\left(\phi\, U_{\triangleright\triangleleft p}^{\leq t}\, \varphi\right) = \sum_{k=L_{\varepsilon}}^{U_{\varepsilon}} e^{-qt} \frac{(q \cdot t)^{k}}{k!} P^{k} \cdot \vec{i_{\varphi}}$$

Example for the $\mathrm{Prob}\left(\phi\, U^{\leq t}_{\rhd\lhd p}\, \varphi\right)$ estimate

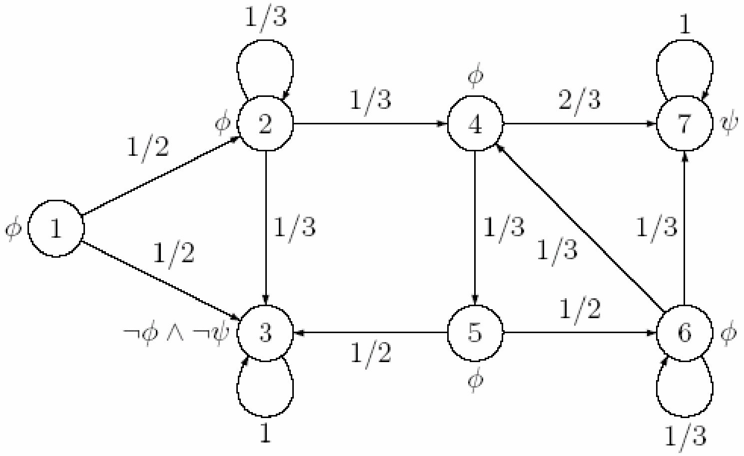


Figure 1: The Initial Example.

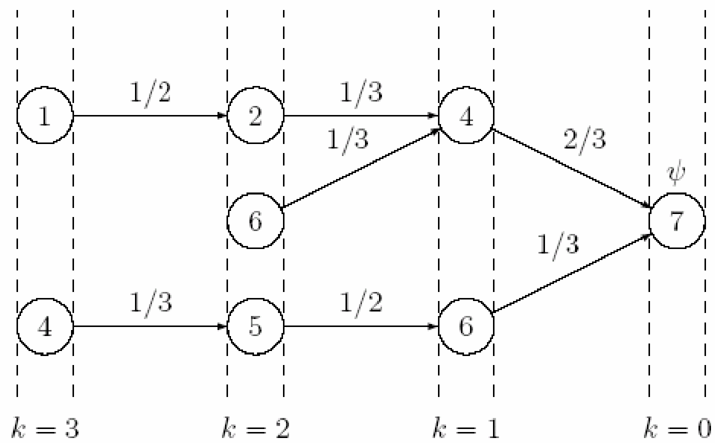


Figure 2: Some of the finite paths.

k=0:
P^0_{min}=(0, 0, 0, 0, 0, 0, 1)
P^0 =(0, 0, 0, 0, 0, 0, 1)

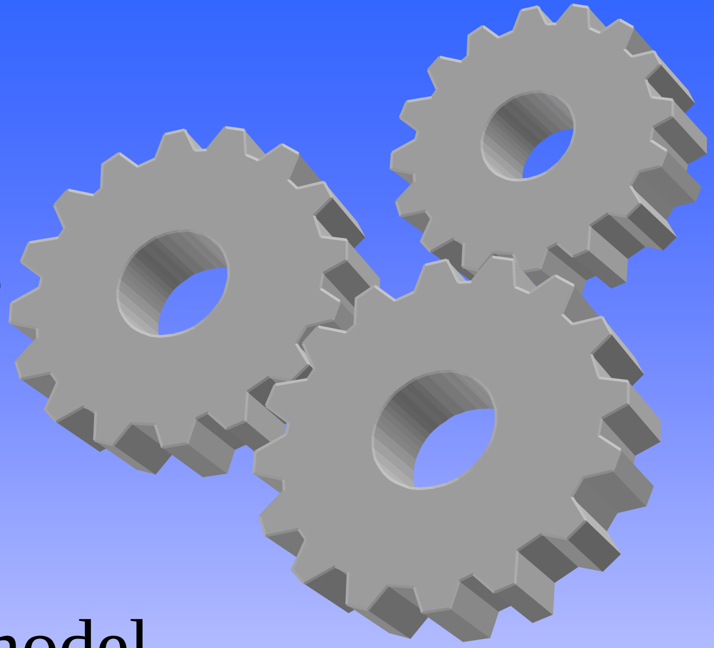
k=1:
P^1_{min}=(0, 0, 0, 2/3, 0, 1/3, 1)
P^1 =(0, 0, 0, 2/3, 0, 1/3, 1)

k=2:
P^2_{min}=(0, 2/9, 0, 2/3, 1/6, 5/9, 1)
P^2 =(0, 2/9, 0, 2/3, 1/6, 2/3, 1)

k=3:
P^3_{min}=(1/9, 2/9, 0, 13/18, 5/18, 5/9, 1)
P^3 =(1/9, 8/27, 0, 13/18, 1/3, 7/9, 1)

k=5:
P^5_{min}=(1/9, 2/9, 0, 13/18, 5/18, 5/9, 1)
P^5 =(55/324, 181/486, 0, 43/54, 5/12, 47/54, 1)

Conclusions



Several possible ways of model checking improvement are under consideration, the work is in progress!

General Conclusions

- The Cybernetix Case Study,
- The CSL model checking improvements,
- The new version of ETMCC;