Probabilistic Model Checking

Marta Kwiatkowska
Gethin Norman
Dave Parker

University of Oxford

Part 11 – Advanced Topics

Overview

- Probabilistic model checking technology…
  - formulated, implemented and evaluated
  - usable and useful!
- Scalability challenge
  - state-space explosion has not gone away…
- Some approaches to tackle the problem
  - parallelisation
  - statistical model checking
  - abstraction
  - model reductions
  - more…

Parallelisation

- Parallelisation of probabilistic model checking
  - distribution of storage/computation costs
  - of growing importance, e.g. multicore architectures
- Ease of distribution depends on computation task
  - reachability? numerical computation?
- Potentially promising for symbolic approaches – less I/O
  - compactness enables storage of the full matrix at each node
  - approaches using Kronecker [Kemper et al.] and MTBDDs
- Here
  - focus on steady-state solution for CTMCs
  - use wavefront techniques

Numerical solution for CTMCs

- Recall, steady-state probability distribution
  - can be obtained by solving linear equation system:

$$\underline{\pi}^{C} \cdot \mathbf{Q} = \underline{0} \quad \text{and} \quad \sum_{s \in S} \underline{\pi}^{C}(s) = 1$$

where Q is infinitesimal generator matrix of C (C irreducible)

- We consider the more general problem of solving:

$$\mathbf{A} \cdot \underline{x} = \underline{b}$$ where **A** is n×n matrix, $\underline{b}$ vector of length n

- Numerical solution techniques
  - direct, not feasible for very large models
  - iterative stationary (Jacobi, Gauss-Seidel), memory efficient
  - projection methods (Krylov, CGS, …), fastest convergence, but require several vectors

Gauss-Seidel

- Computes one matrix row at a time
- Updates i^{th} element using most up-to-date values
- Computation for a single iteration, $n \times n$ matrix:

1. for $(0 \leq i \leq n-1)$
2. $\quad \underline{x}_i := (\underline{b}_i - \sum_{0 \leq j \leq n-1,\, j \neq i} A_{ij} \cdot \underline{x}_j) \,/\, A_{ii}$

- Can be reformulated in block form, $N \times N$ blocks, length M

1. for $(0 \leq p \leq N-1)$
2. $\quad \underline{v} := \underline{b}_{(p)}$
3. $\quad$ for each block $A_{(pq)}$ with $q \neq p$
4. $\qquad \underline{v} := \underline{v} - A_{(pq)}\, \underline{x}_{(q)}$
5. $\quad$ for $(0 \leq i \leq M-1, i \neq j)$
6. $\qquad \underline{x}_{(p)i} := (\underline{v}_i - \sum_{0 \leq j \leq M} A_{(pp)ij} \cdot \underline{x}_{(p)j}\,) \,/\, A_{(pp)ii}$

computes one matrix block at a time

Parallelising Gauss-Seidel

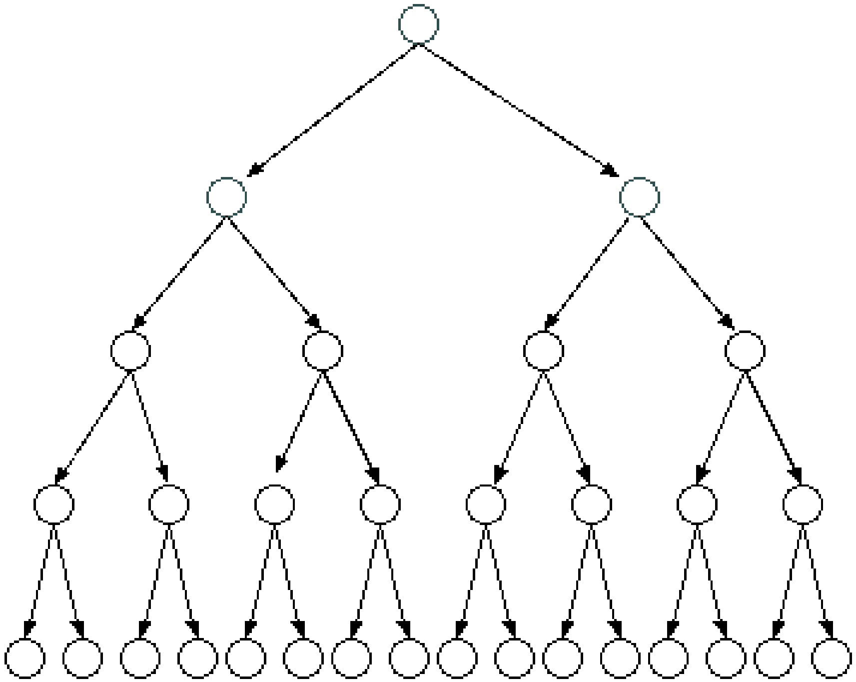
- Inherently sequential for dense matrices
  - uses results from current and previous iterations
- Permutation has no effect on correctness of the result
  - can be exploited to achieve parallelisation for certain sparse matrix problems, e.g. [Koester, Ranka & Fox 1994]
- The block formulation helps, although
  - requires row-wise access to blocks and block entries
  - need to respect computational dependencies
  - i.e. when computing vector block $\mathbf{x}_{(p)}$
    use values from current iteration for blocks $q < p$
    previous iteration for $q > p$
- Idea: propose to use wavefront techniques
  - extract dependency information

Symbolic techniques for CTMCs

- Explicit matrix representation
  - intractable for very large matrices
- Symbolic representations
  - exploit regularity to obtain compact matrix storage
  - also faster model construction, reachability, etc
  - sometimes also beneficial for vector storage
  - include Multi-Terminal Binary Decision Diagrams (MTBDDs), matrix diagrams and Kronecker representation
- Here, work with MTBDDs and derived structures
  - underlying data structure of the PRISM model checker
  - enhanced with caching-based techniques that substantially improve numerical efficiency

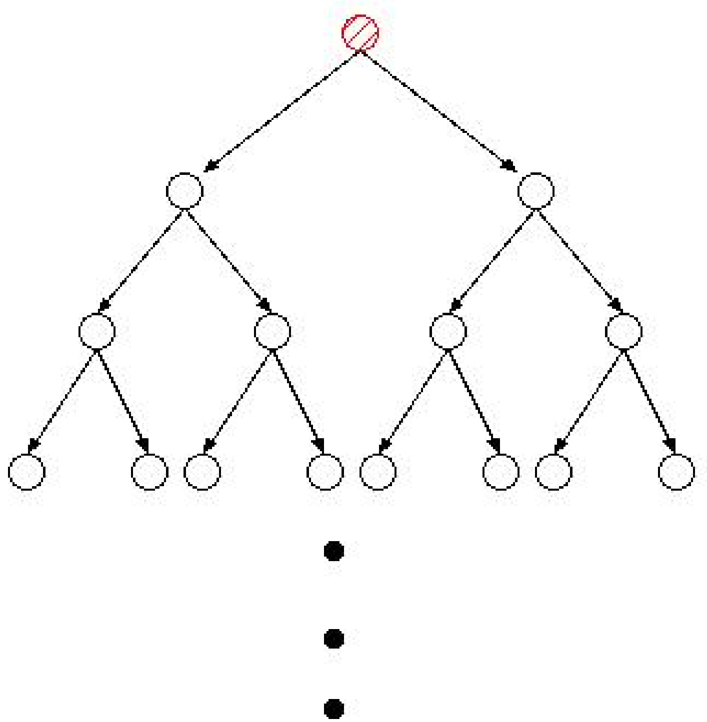
MTBDD data structures

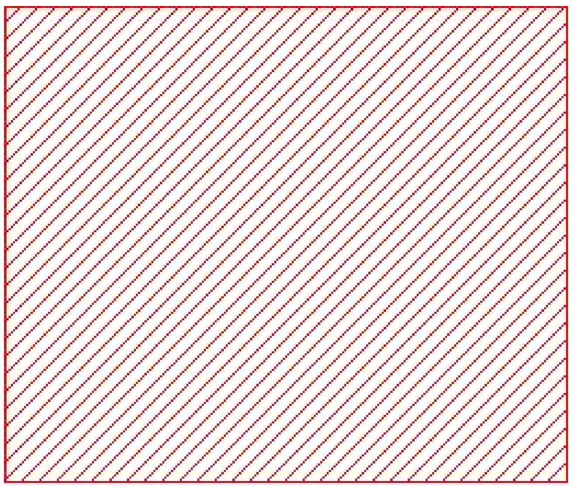
- Recursive, based on Binary Decision Diagrams (BDDs)
  - stored in reduced form (DAG), with isomorphic subtrees stored only once
  - exploit regularity to obtain compact matrix storage



Matrices as MTBDDs

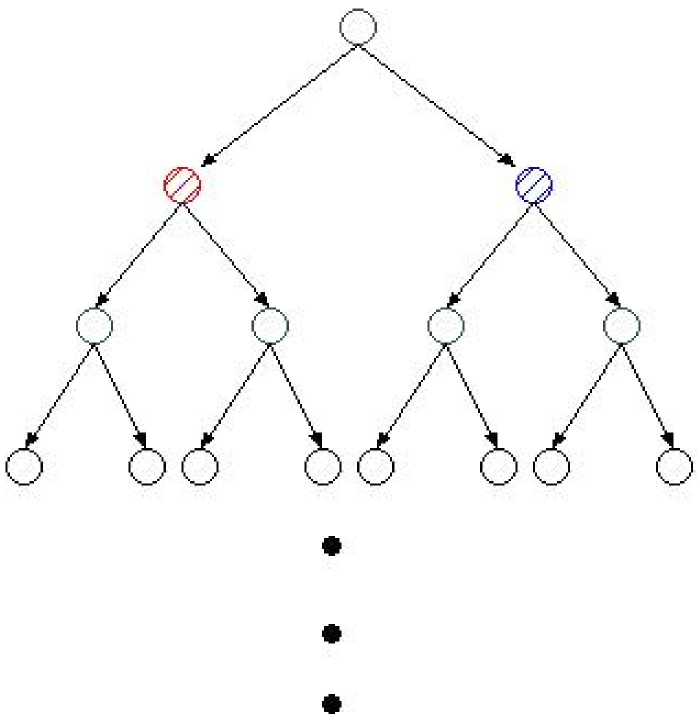
- Representation
  - root represents the whole matrix
  - leaves store matrix entries, reachable by following paths from the root node

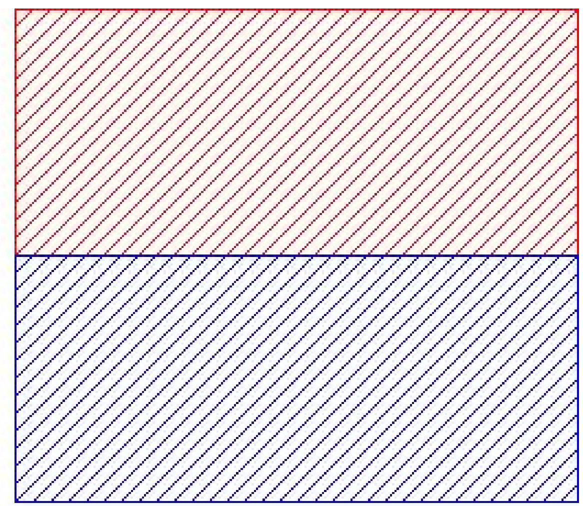




Matrices as MTBDDs

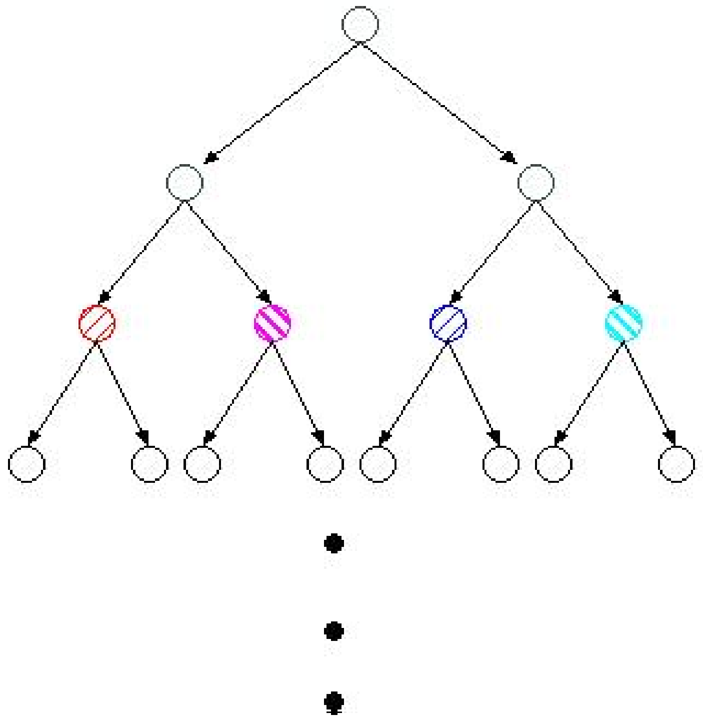
- Recursively descending through the tree
  - divides the matrix into submatrices
  - one level, divide into two submatrices

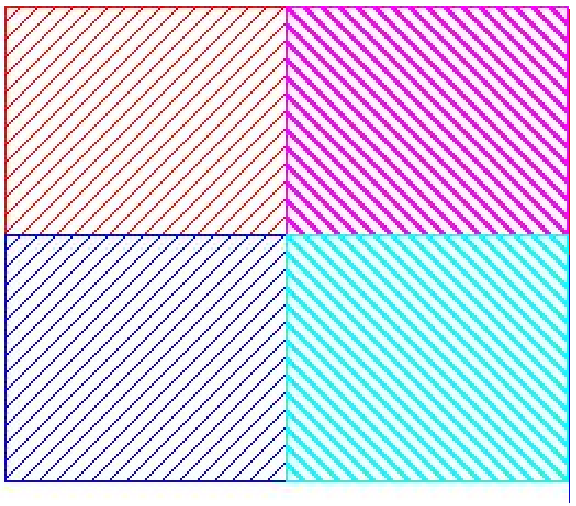




Matrices as MTBDDs

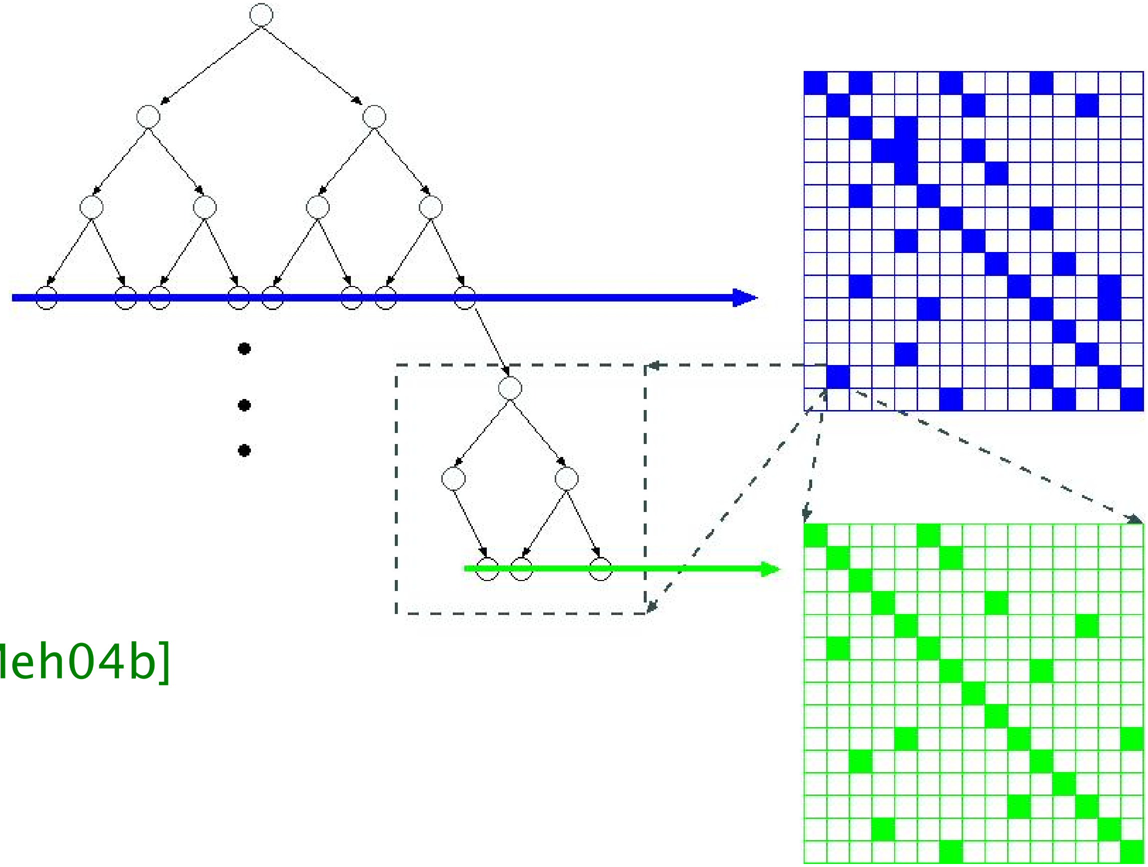
- Recursively descending through the tree
  - provides a convenient block decomposition
  - two levels, divide into four blocks





A two-layer structure from MTBDDs

- Block decomposition, store as two sparse matrices
  - enables fast row-wise access to blocks and block entries



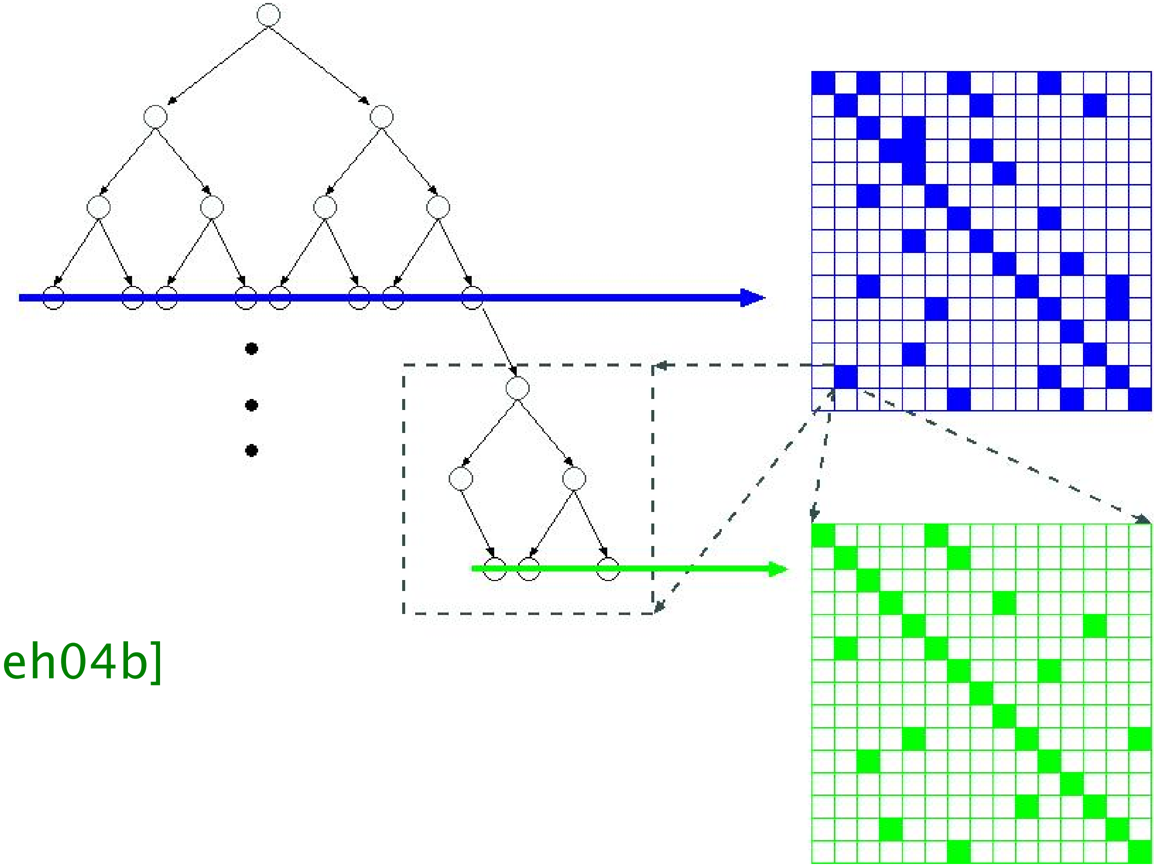
[Par02, Meh04b]

Wavefront techniques

- An approach to parallel programming, e.g. [Joubert et al ’98]
  - divide computation into tasks, form a schedule
- A schedule contains several wavefronts
  - each wavefront comprises algorithmically independent tasks
  - i.e. correctness is not affected by execution order
- The execution is carried out from one wavefront to another
  - tasks assigned according to the dependency structure
  - each wavefront contains tasks that can be executed in parallel
- Our approach
  - tasks are determined by matrix blocks
  - fast extraction of dependency information from MTBDD matrix

A two-layer structure from MTBDDs

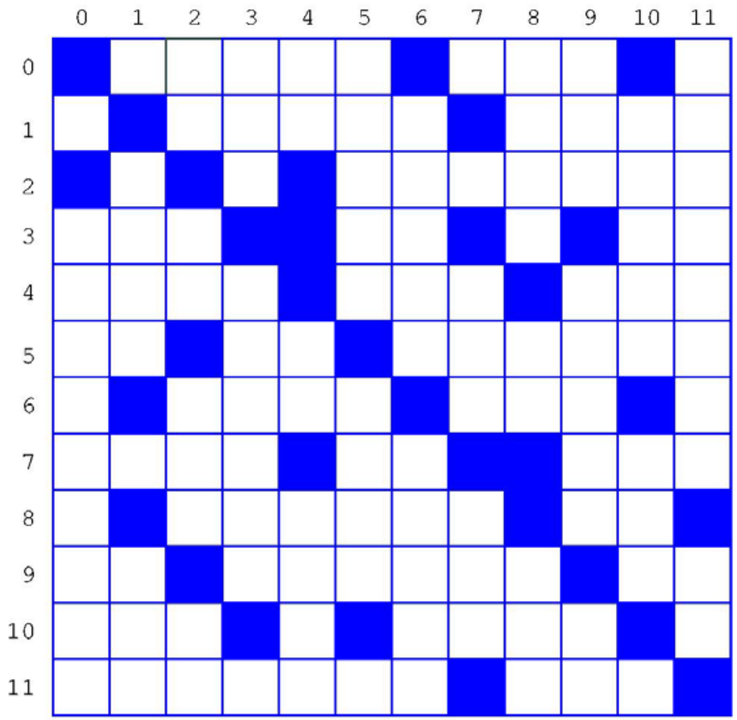
- Block decomposition, store as two sparse matrices
  - enables fast row-wise access to blocks and block entries

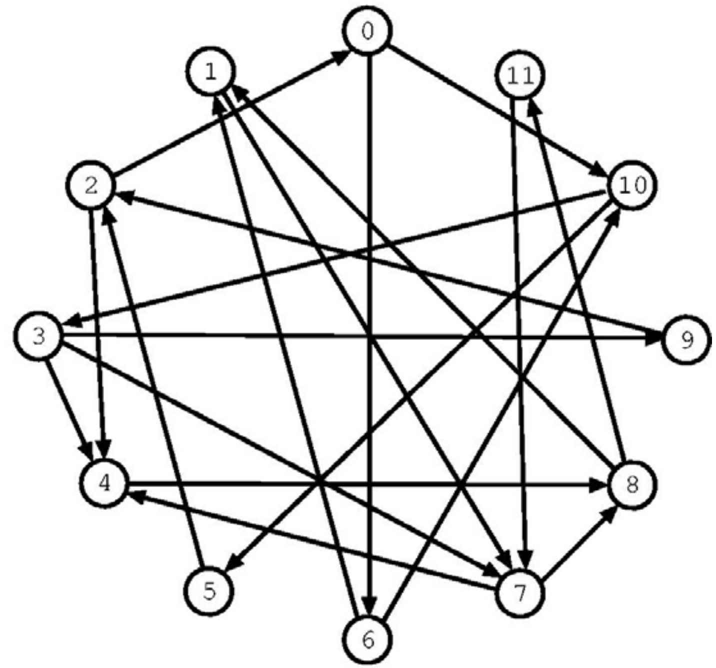


[Par02,Meh04b]

Dependency graph from MTBDD

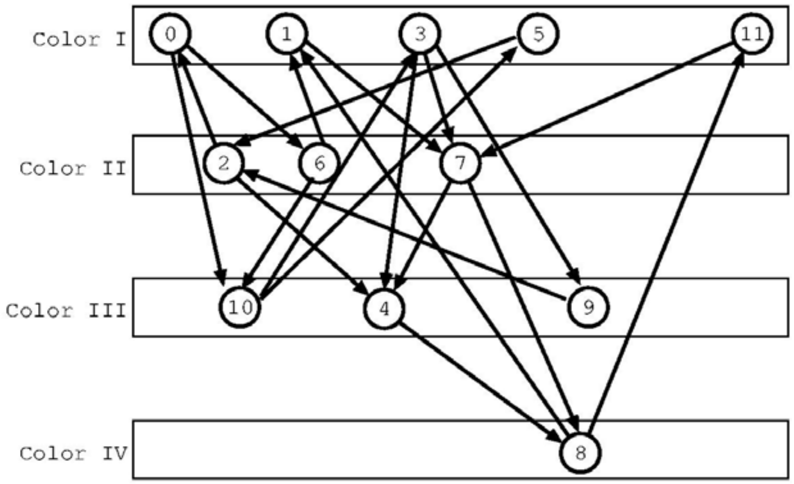
- By traversal of top levels of MTBDD, as for top layer





Generating a wavefront schedule

- By colouring the dependency graph…



- Can generate a schedule to compute in waves from one colour to another

Implementation

- Symbolic approach particularly well suited to wavefront parallelisation of Gauss-Seidel
  - can store full matrix at each node
  - hence reduced communication costs, since only vector blocks need to be exchanged
- Runs on Ethernet and Myrinet-enabled PC cluster [ZPK05a]
  - use MPI (the MPICH implementation)
  - prototype extension for PRISM
  - various optimisations, load-balancing, etc
- Evaluated on a range of benchmarks
  - good overall speedup
  - within PRISM, currently only steady-state

Experimental results: models

- Parameters and statistics of models
  - Include Kanban 9,10 and FMS 13, previously intractable
  - All compact, requiring less than 1GB

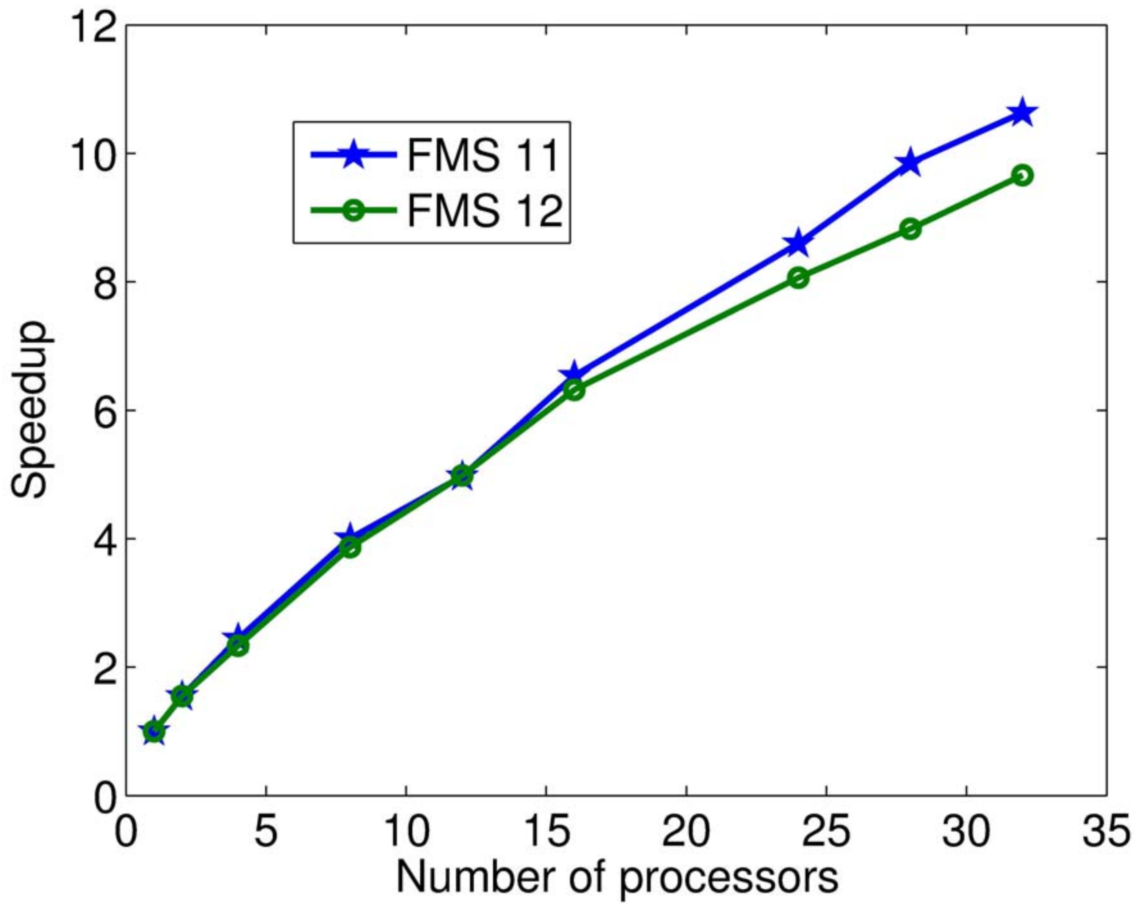
Model	States	Transitions	Blocks (N)	Size (MB)	
				MTBDD	Sparse
FMS (K=11)	54,682,992	518,030,370	1,365	297	6,137
FMS (K=12)	111,414,940	1,078,917,632	1,820	558	12,772
FMS (K=13)	216,427,680	2,136,215,172	2,380	1,005	25,273
Kanban (K=7)	41,644,800	450,455,040	120	18	5,314
Kanban (K=8)	133,865,325	1,507,898,700	165	43	17,767
Kanban (K=9)	384,392,800	4,474,555,800	220	95	52,674
Kanban (K=10)	1,005,927,208	12,032,229,352	286	195	141,535
Polling (K=20)	31,457,280	340,787,200	308	65	4,020
Polling (K=21)	66,060,288	748,683,264	324	141	8,820
Polling (K=22)	138,412,032	1,637,875,712	340	307	19,272

Experimental results: time

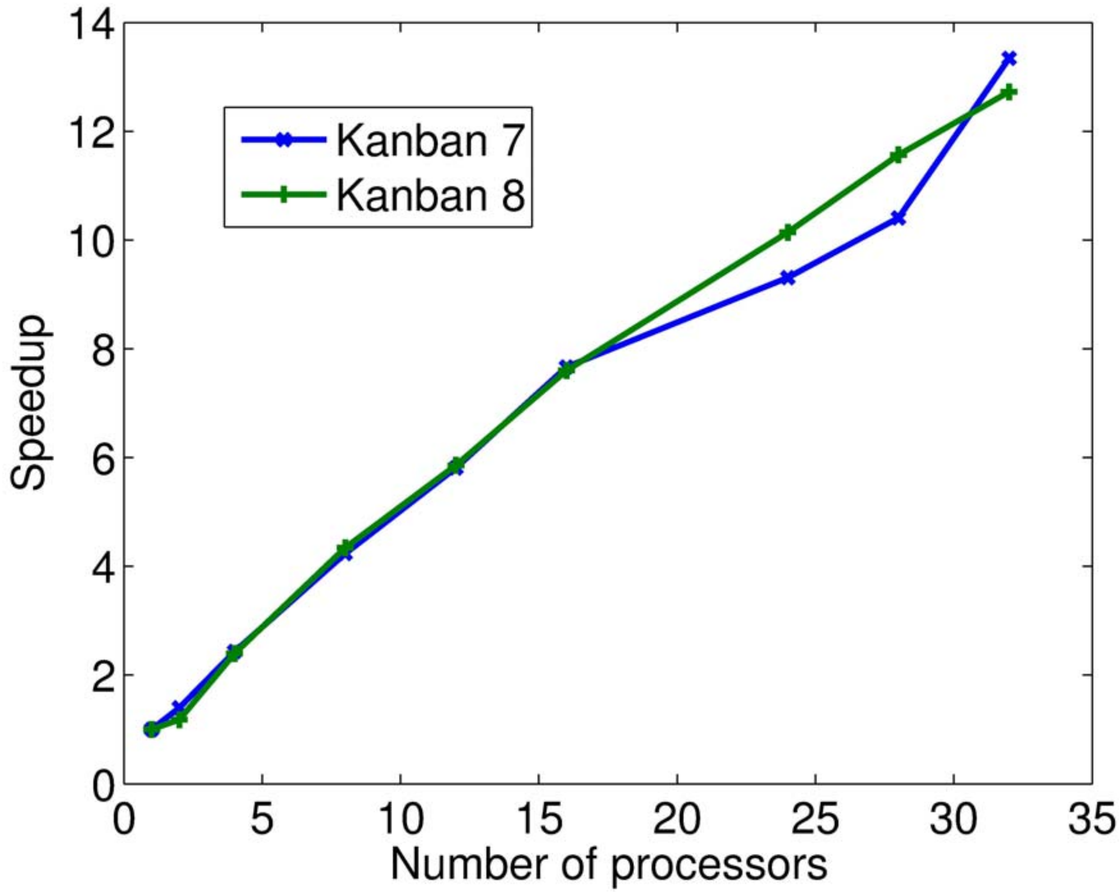
- Total execution times (in seconds) with 1 to 32 nodes
  - Termination condition maximum relative difference 10^{-6}
  - Block numbers selected to minimise storage

Num. nodes	FMS			Kanban				Polling		
	K=11	K=12	K=13	K=7	K=8	K=9	K=10	K=20	K=21	K=22
1	15,990	35,637	O/M	4,683	19,417	O/M	O/M	8,764	14,195	45,485
2	10,349	22,986	O/M	3,351	16,419	O/M	O/M	6,451	10,834	37,713
4	6,548	15,264	O/M	1,925	8,099	34,755	O/M	4,906	6,301	21,553
8	3,991	9,212	O/M	1,106	4,474	16,271	O/M	2,123	3,463	11,287
12	3,218	7,148	O/M	806	3,314	11,452	45,206	1,488	2,433	8,338
16	2,446	5,642	12,544	611	2,555	9,522	29,674	1,153	1,807	5,929
24	1,860	4,419	9,657	503	1,915	6,741	20,560	769	1,335	4,546
28	1,623	4,038	8,173	450	1,679	5,753	18,599	736	1,203	3,491
32	1,504	3,689	7,693	351	1,526	5,134	15,750	650	858	3,086

Experimental results: FMS speed-up



Experimental results: Kanban speed-up



Overview

- Probabilistic model checking technology…
  - formulated, implemented and evaluated
  - usable and useful!
- Scalability challenge
  - state-space explosion has not gone away…
- Some approaches to tackle the problem
  - parallelisation
  - statistical model checking
  - abstraction
  - model reductions
  - more…

Approximate verification

- Approximate probabilistic model checking
  - sampling using Monte Carlo discrete-event simulation
  - performed at modelling language level
  - no need to build the probability/rates matrix
  - more easily extended to a wider range of properties
  - potentially huge number of samples for accurate answers

- Tool support:
  - APMC [LHP06] – PCTL/LTL for D/CTMCs, distributed version
  - also supported in PRISM (distributed version coming soon)

- Statistical hypothesis testing, acceptance sampling
  - “bounded” properties, e.g. $P_{<p}[\phi_1 \ U^{\leq t} \ \phi_2]$
  - see e.g. Ymer [YS02]

Statistical probabilistic model checking

- Numerical method
  - requires the solution of a linear equation system
  - highly accurate results
  - expensive for systems with many states
  - in practice, approximate since solution usually iterative

- Statistical method
  - work from the syntactic model description
  - low memory requirements
  - adapts to difficulty of problem (sequential)
  - expensive if high accuracy is required

Numerical solution method

- Recall to verify $P_{\geq p}$ [ϕ_1 $U^{[0,t]}$ ϕ_2] for CTMC C:
  - compute probability of being in a state satisfying ϕ_2 at time t in modified model $C[\phi_2][\neg\phi_1 \wedge \neg\phi_2]$

$$\underline{\text{Prob}}(\phi_1 \ U^{[0,t]} \ \phi_2) \ = \ \sum_{i=0}^{\infty} \left(\gamma_{q \cdot t, i} \cdot \left(P^{\text{unif}(C[\phi_2][\neg\phi_1 \wedge \neg\phi_2])} \right)^i \cdot \underline{\phi_2} \ \right)$$

  - using uniformisation, where $\gamma_{q \cdot t, i}$ are Poisson coefficients
  - $P_{\geq p}$ [ϕ_1 $U^{[0,t]}$ ϕ_2] holds in state s iff Prob(s, ϕ_1 $U^{[0,t]}$ ϕ_2)$\geq$p

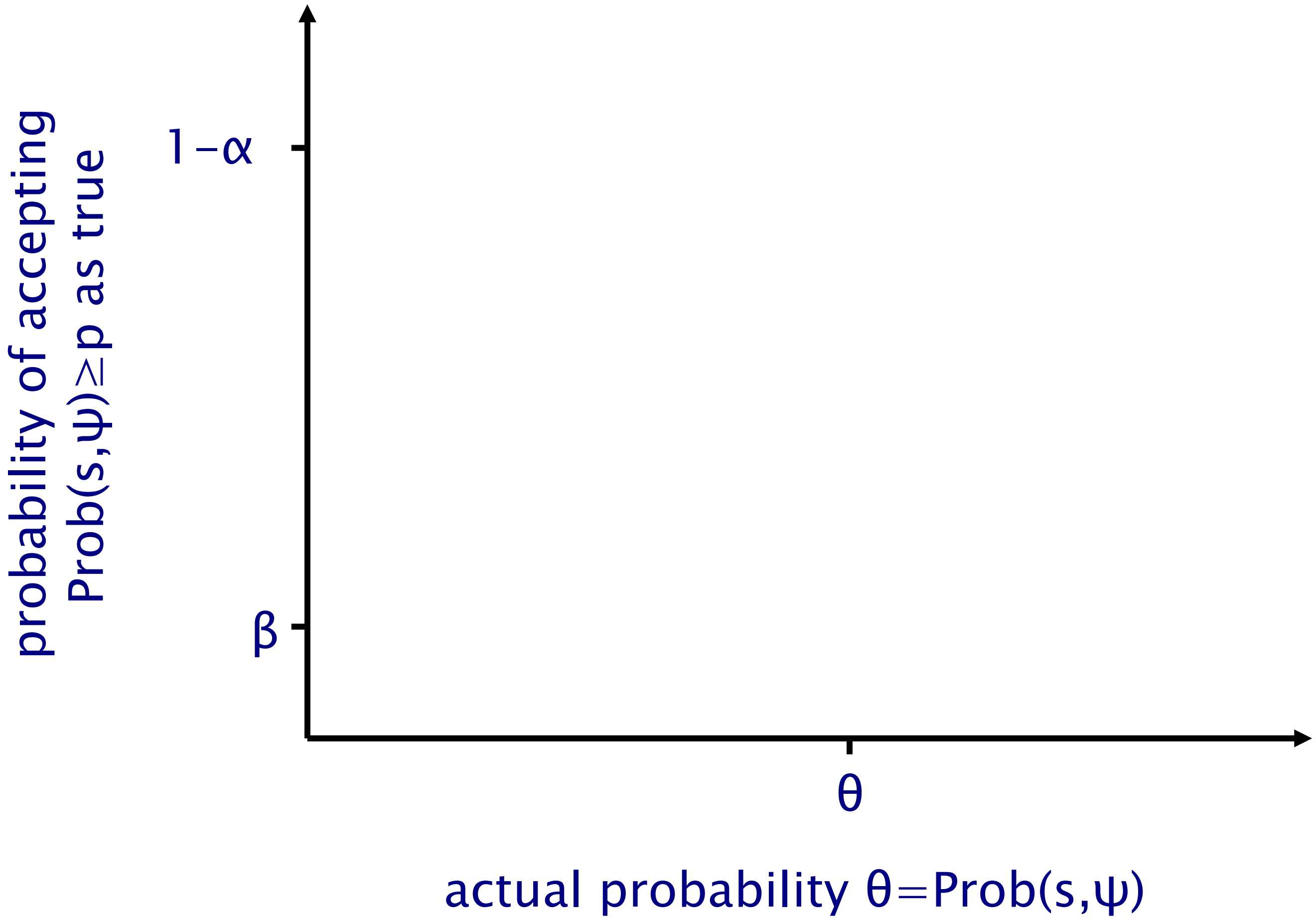
- Truncate the summation using Fox-Glynn with error ϵ
  - if computed probability$\geq$p, then Prob(s,ϕ_1 $U^{[0,t]}$ ϕ_2)$\geq$p
  - if computed probability$\leq$p$-\epsilon$, then Prob(s,ϕ_1 $U^{[0,t]}$ ϕ_2)$\leq$p
  - otherwise, we cannot tell if $P_{\geq p}$ [ϕ_1 $U^{[0,t]}$ ϕ_2] holds
  - complexity $O(q \cdot t)$ matrix-vector multiplications
  - but $\epsilon = 10^{-10}$ possible with no performance degradation

Statistical solution method [YS02]

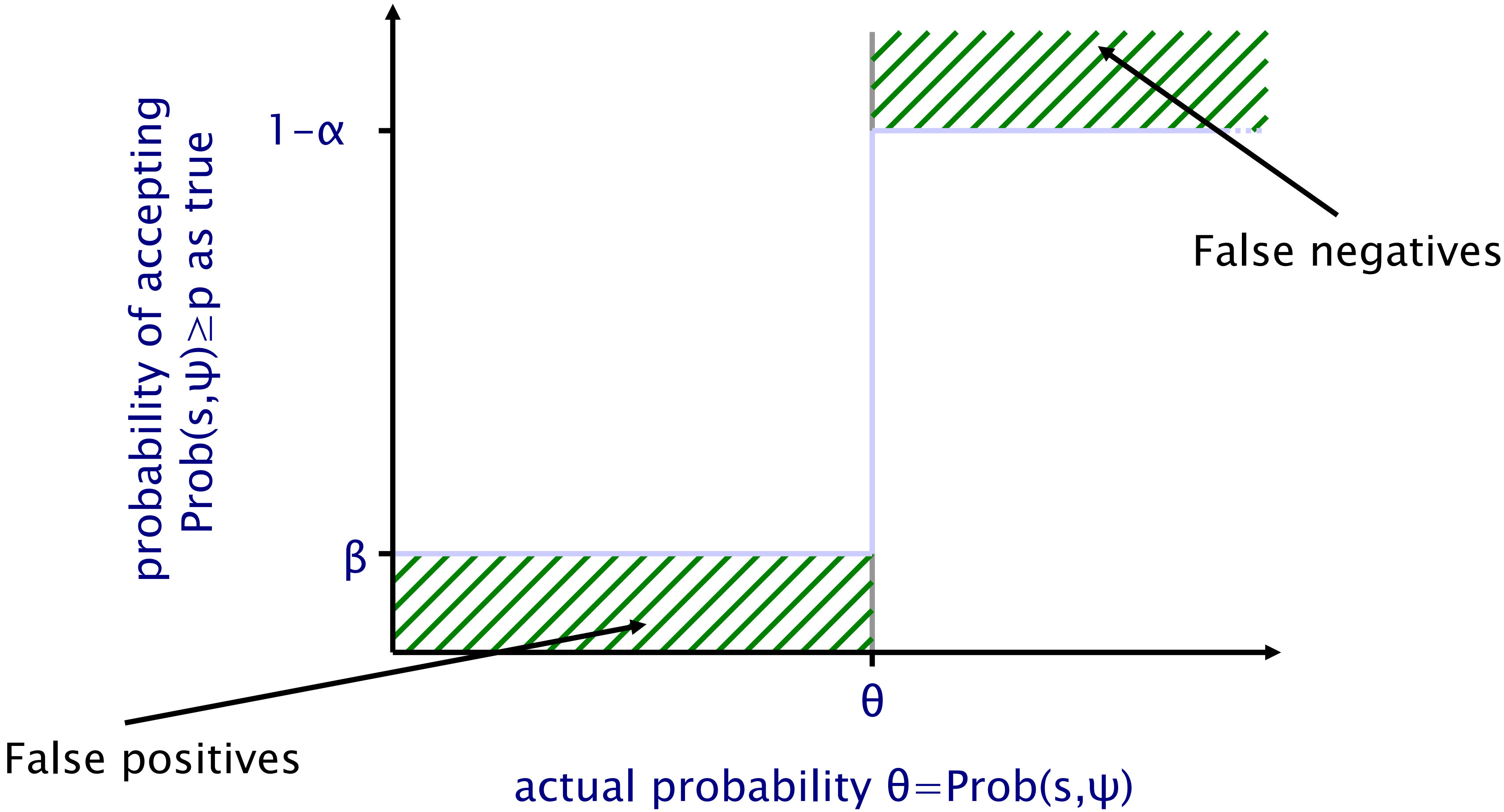
- Use discrete event simulation to generate sample paths
- Use sequential acceptance sampling to verify probabilistic properties, for path formula ψ
  - hypothesis: $Prob(s,\psi) \geq p$
- Choose error bounds α,β
- Probability of false negative: $\leq \alpha$
  - we say that $Prob(s,\psi) \geq p$ is false when it is actually true
- Probability of false positive: $\leq \beta$
  - we say that $Prob(s,\psi) \geq p$ is true when it is actually false

Not estimation!

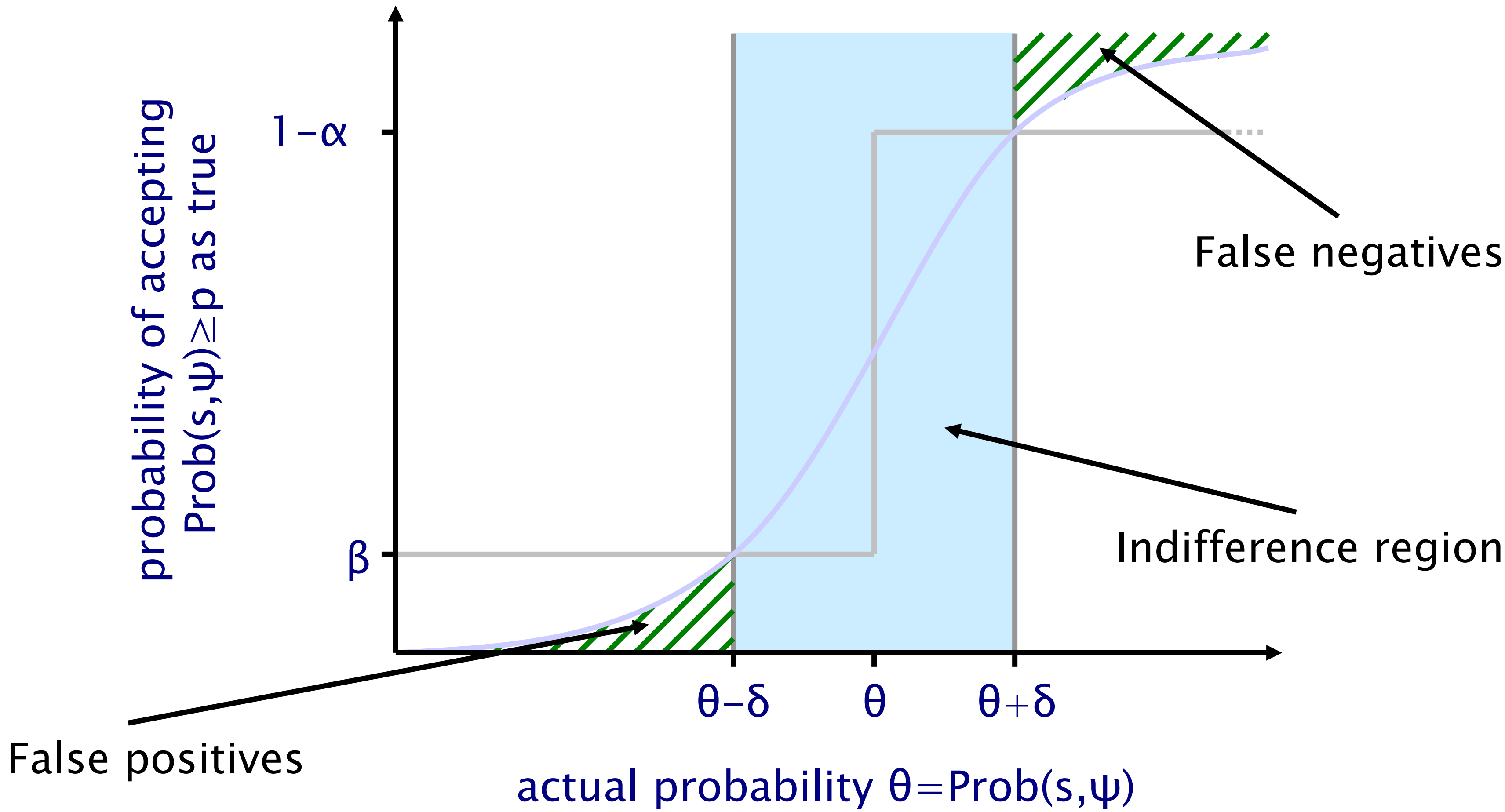
Performance of test



Ideal performance

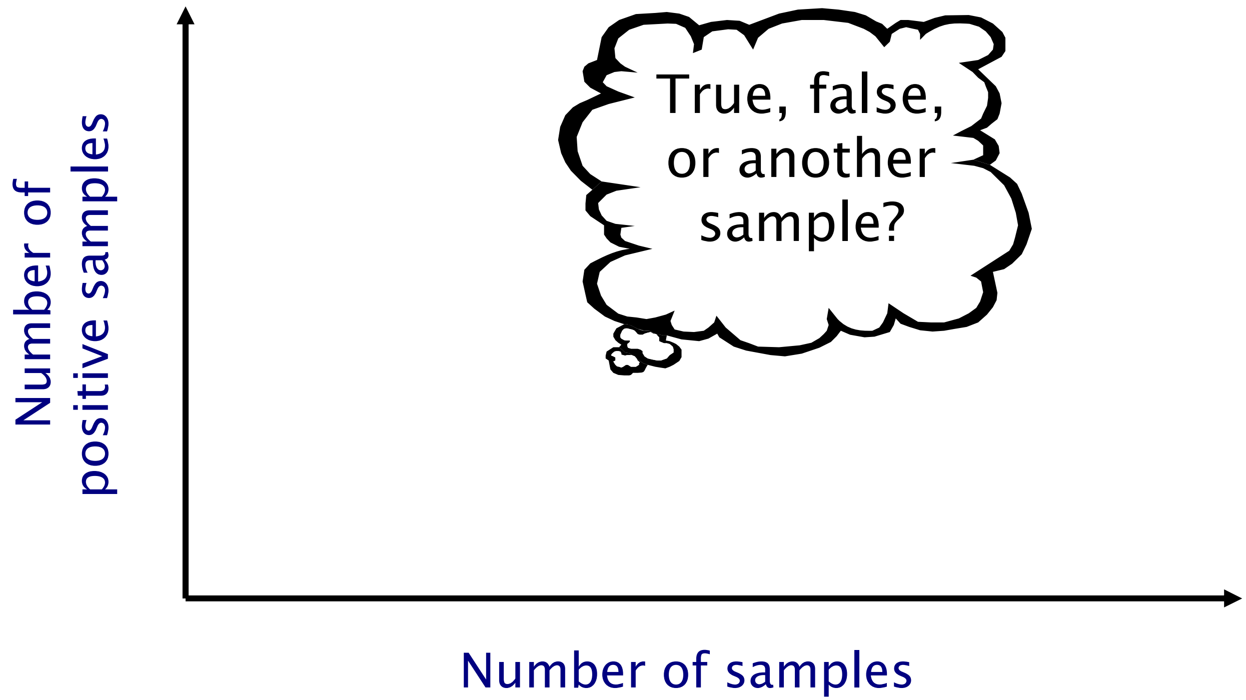


Actual performance



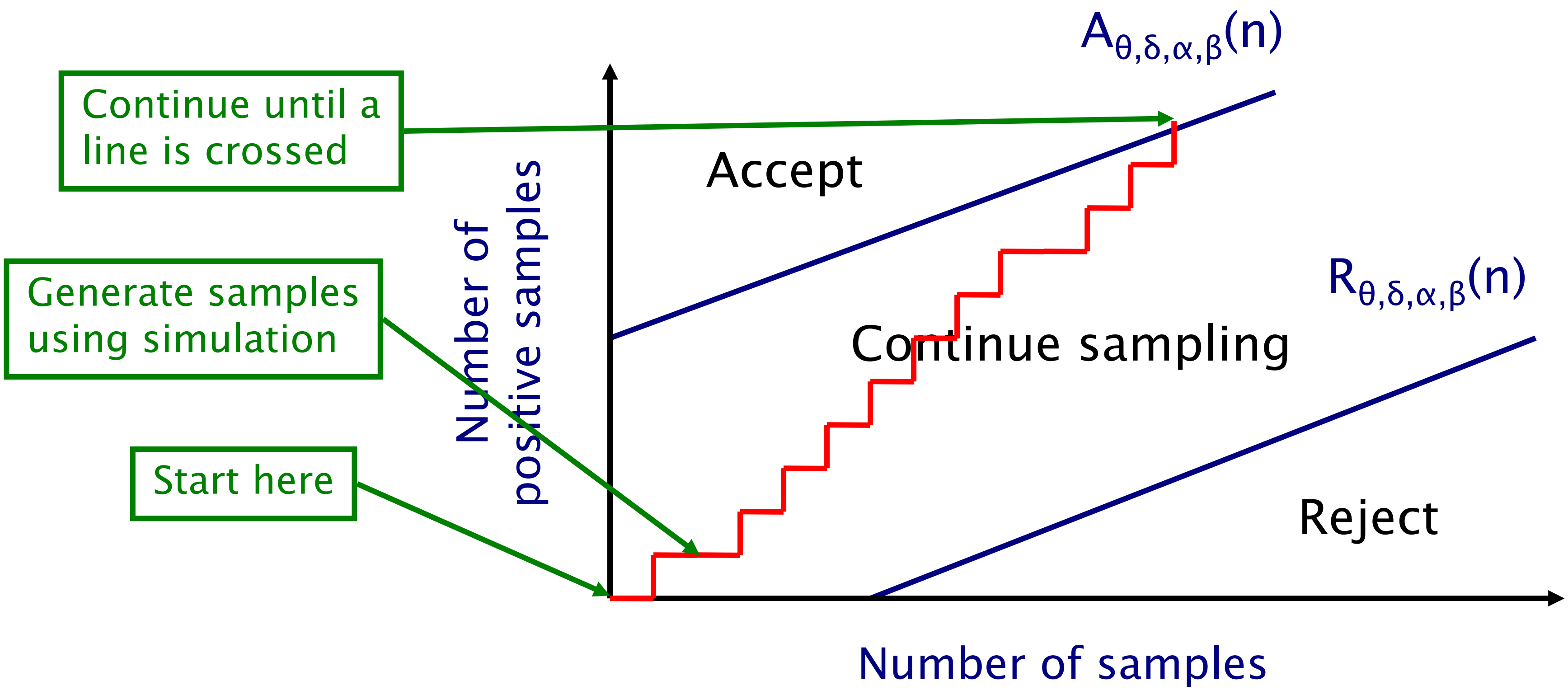
Sequential hypothesis testing

- Hypothesis: Prob(s,ψ)≥p



Sequential hypothesis testing

- We can find an acceptance line and a rejection line given θ, δ, α and β

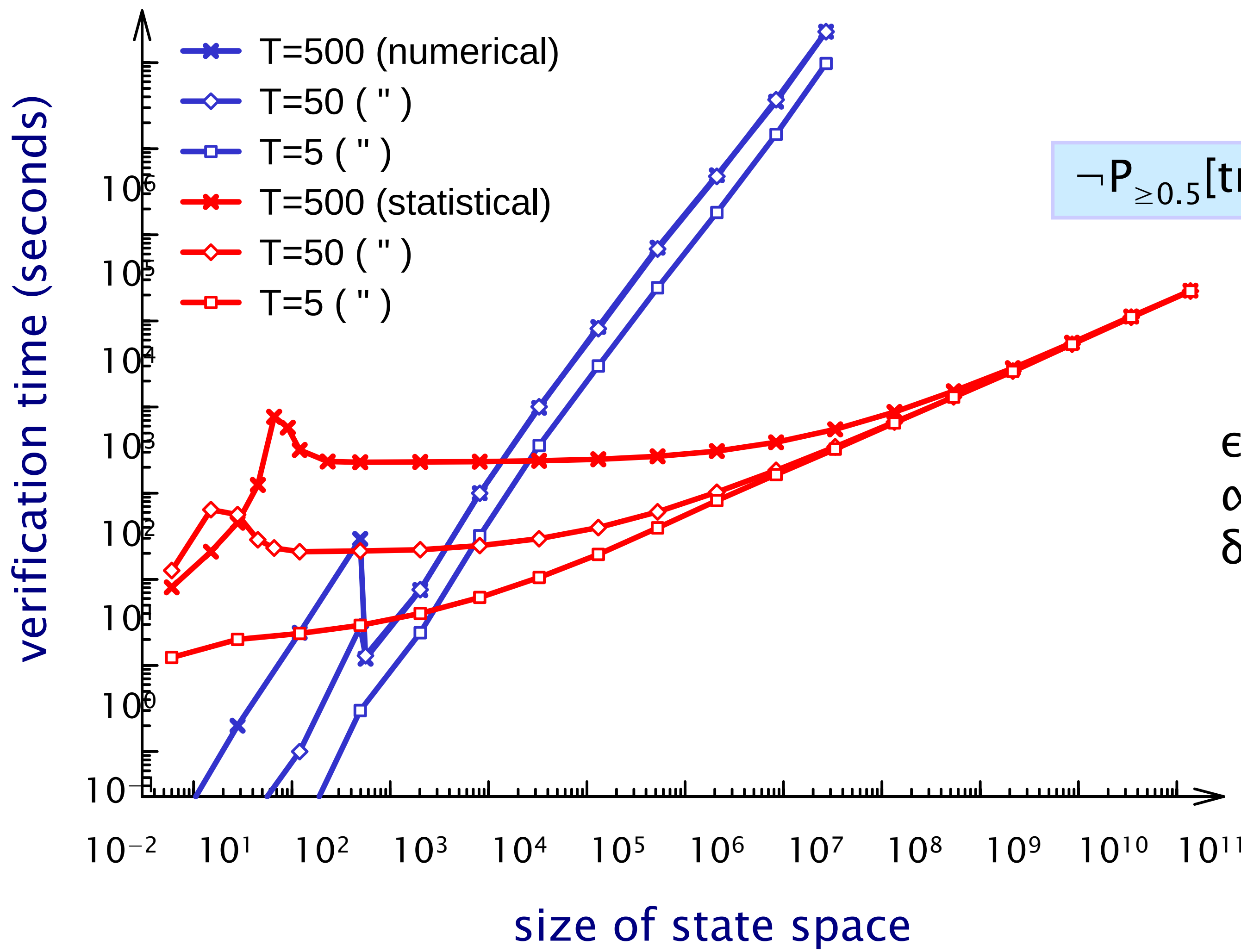


Verifying probabilistic properties

- Verify Prob(s,ψ)≥p with error bounds α and β
  - generate sample paths using simulation
  - verify ψ over each sample path
  - if ψ is true, then we have a positive sample
  - if ψ is false, then we have a negative sample
  - use sequential acceptance sampling to test the hypothesis

- Complexity of the method
  - number of samples: complex dependency on θ, δ, α and β
  - length of sample paths
    - expected length at most $q \cdot t$ (t time bound in ψ)
    - shorter paths if $\neg\phi_1 \vee \phi_2$ is satisfied early
  - no direct dependence on size of state space

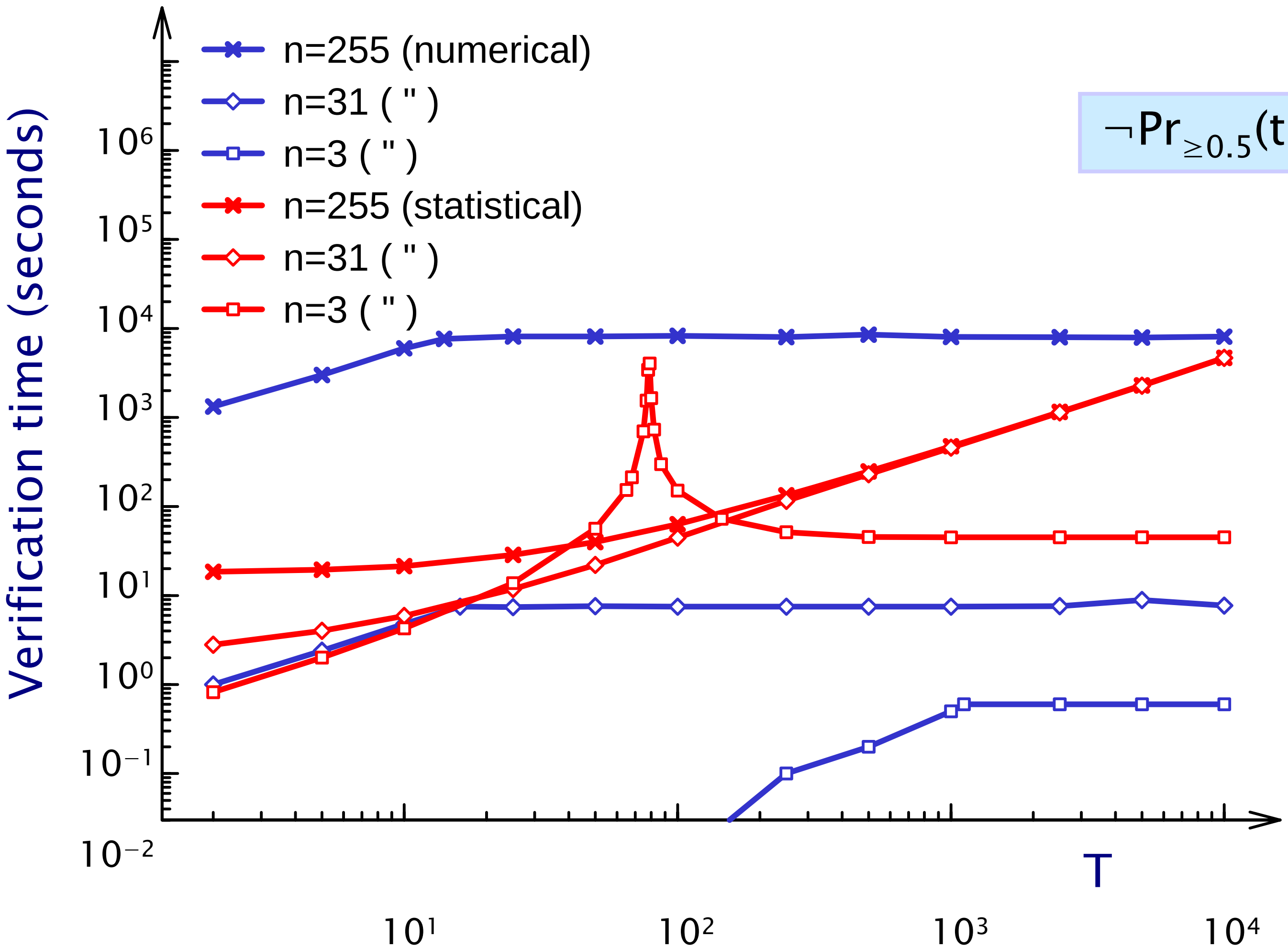
Tandem Queuing Network (results)



$\neg P_{\geq 0.5}[\text{true } U^{[0,T]} \text{ full}]$

$\epsilon = 10-6$
$\alpha = \beta = 10^{-2}$
$\delta = 0.5 \cdot 10^{-2}$

Tandem Queuing Network (results)



$\neg Pr_{\geq 0.5}(\text{true } U^{\leq T} \text{ full})$

$\varepsilon = 10{-}6$
$\alpha = \beta = 10^{-2}$
$\delta = 0.5 \cdot 10^{-2}$

Overview

- Probabilistic model checking technology…
  - formulated, implemented and evaluated
  - usable and useful!
- Scalability challenge
  - state-space explosion has not gone away…
- Some approaches to tackle the problem
  - parallelisation
  - statistical model checking
  - abstraction
  - model reductions
  - more…

Some ongoing research areas

- Abstraction and refinement, see e.g. [DJJL01, KNP06a]
  - construct smaller, abstract model by removing information/variables not relevant to property being checked, iteratively refine abstraction if analysis fails
- Symmetry reduction [DM06, KNP06b]
  - exploit replication of identical components
- Partial order reduction, see e.g. [BGC04, DN04, GNB+06]
  - exploit commutativity of concurrently executed transitions
- Bisimulation quotient [KKZJ07]
  - exploit bisimilarity to obtain reduced model

Future topics

- Counterexamples for probabilistic model checking
  - compute tree-like counterexamples, see e.g. [HK07]
- Directed probabilistic model checking [AHL05]
  - explore the model state space using heuristics
- Predicate abstraction for probabilistic models
  - reduce possibly infinite-state systems
- Compositionality, see e.g. [dAHJ01, Che06, EKVY07]
  - analyse full model based on analysis of sub-components