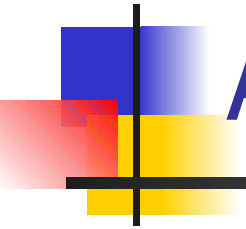
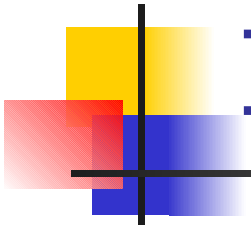


Probabilistic Verification of Discrete Event Systems using Acceptance Sampling

Håkan L. S. Younes Reid G. Simmons

Carnegie Mellon University

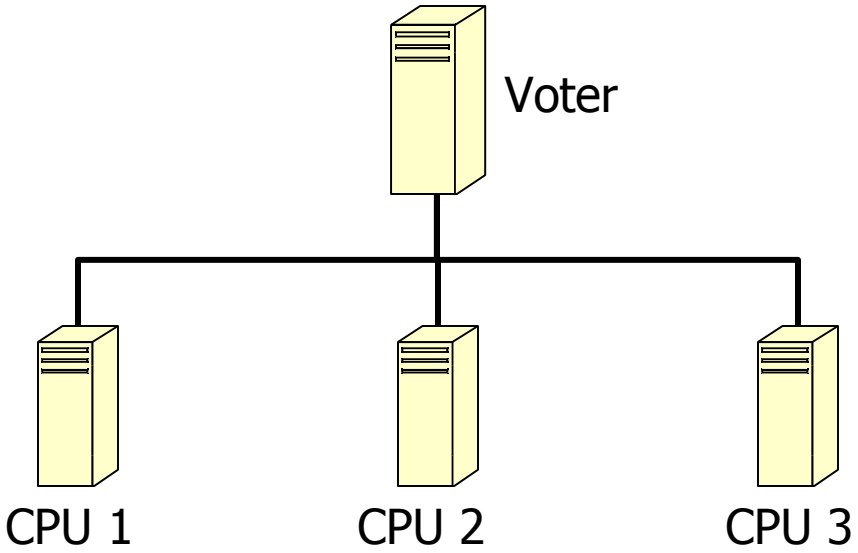


Introduction

- Verify properties of discrete event systems
- Model independent approach
- Probabilistic real-time properties expressed using CSL
- Acceptance sampling
- Guaranteed error bounds

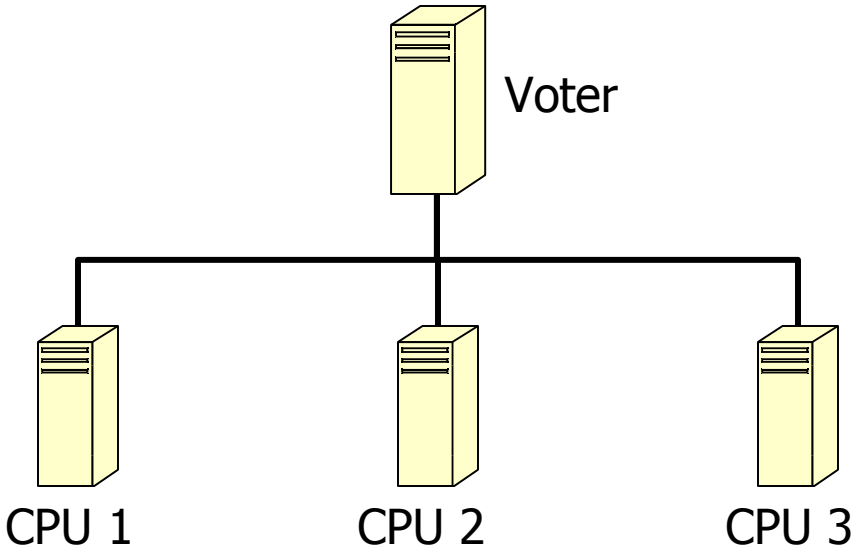
DES Example

- Triple modular redundant system
  - Three processors
  - Single majority voter

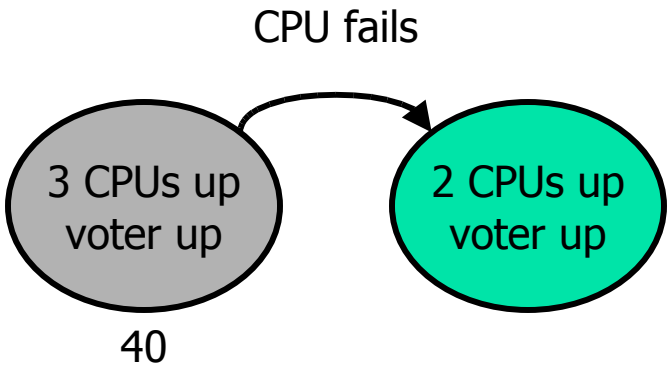


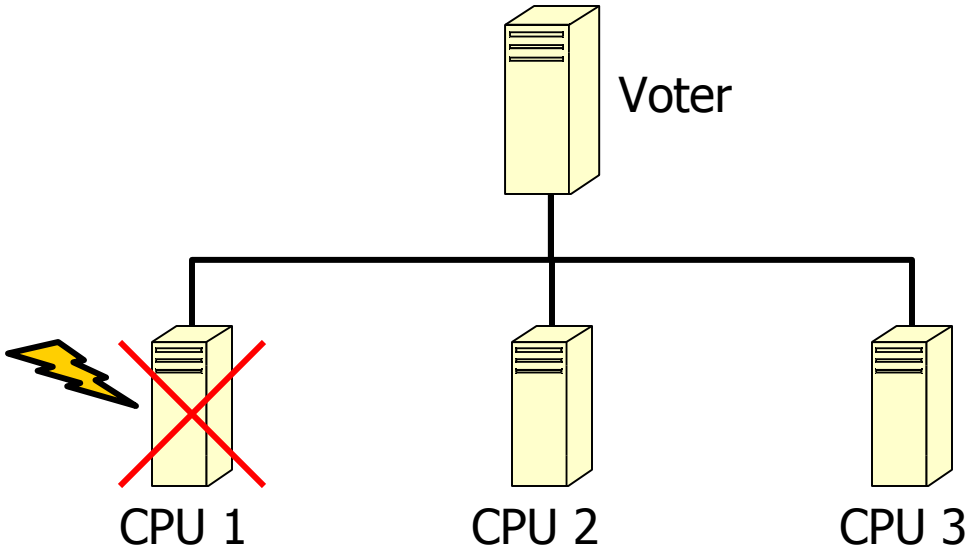
DES Example

3 CPUs up
voter up

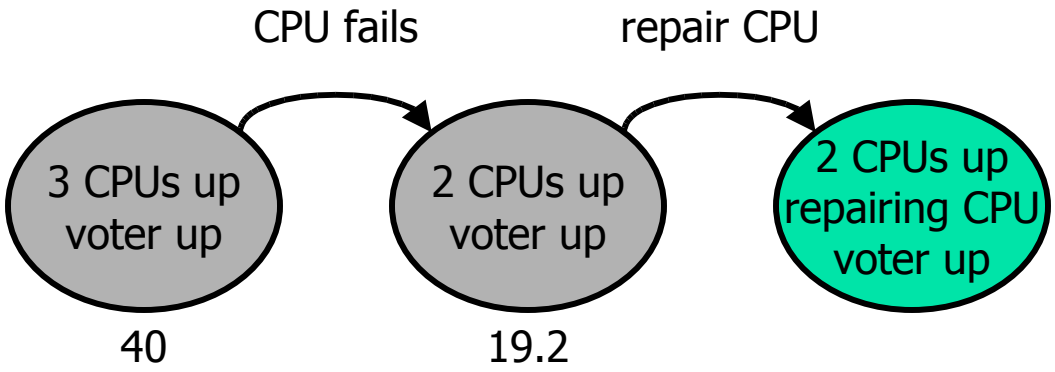


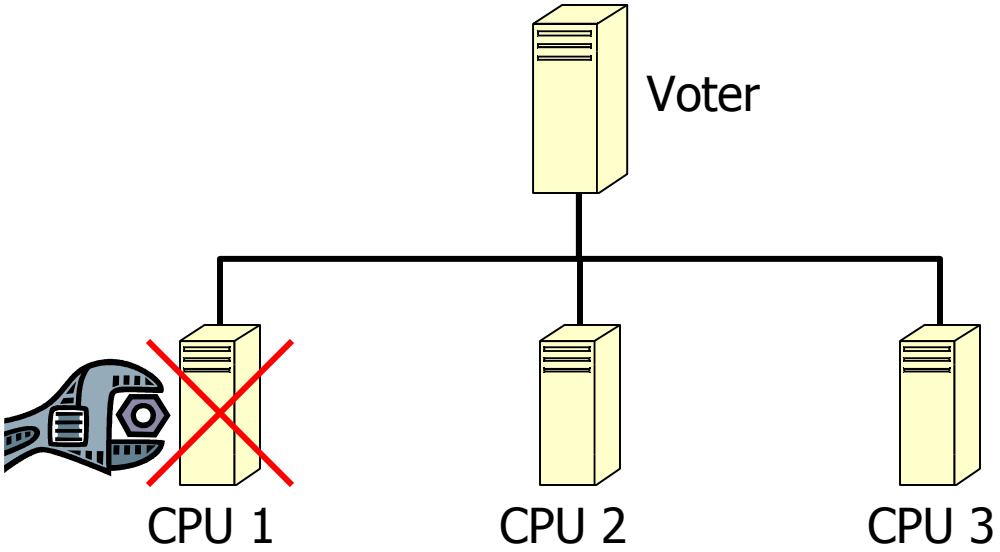
DES Example


CPU fails
3 CPUs up
voter up
2 CPUs up
voter up

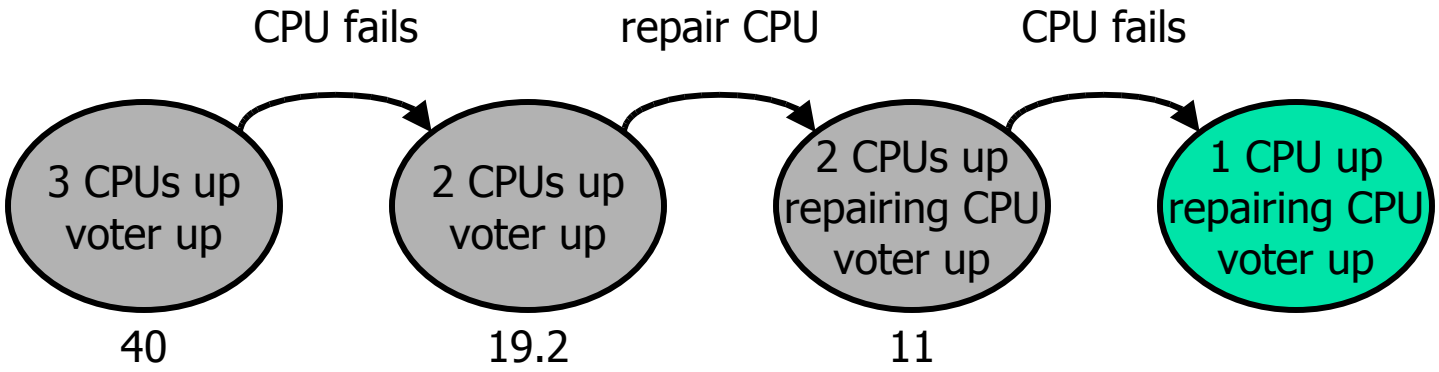

Voter
CPU 1
CPU 2
CPU 3

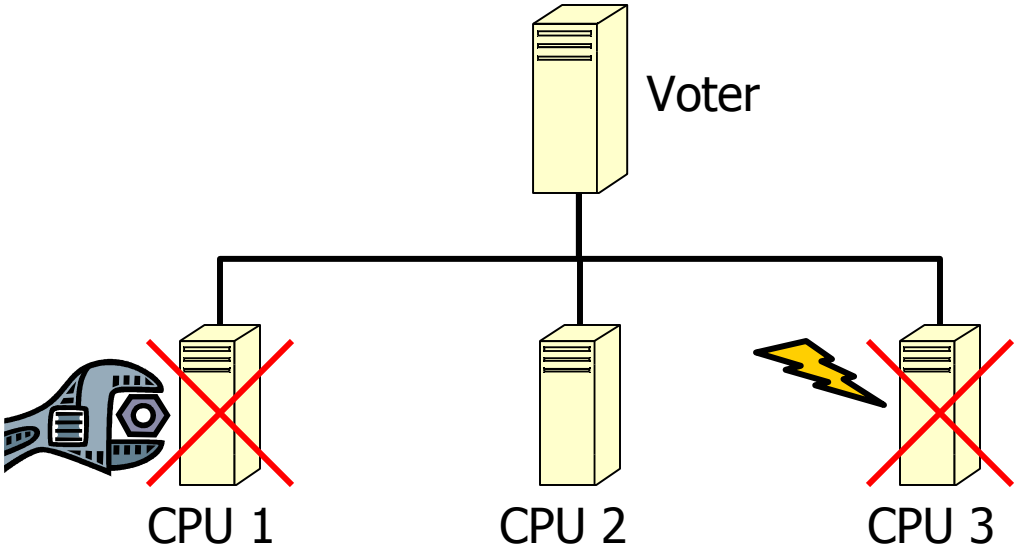
DES Example


CPU fails
repair CPU
3 CPUs up
voter up
2 CPUs up
voter up
2 CPUs up
repairing CPU
voter up
40
19.2

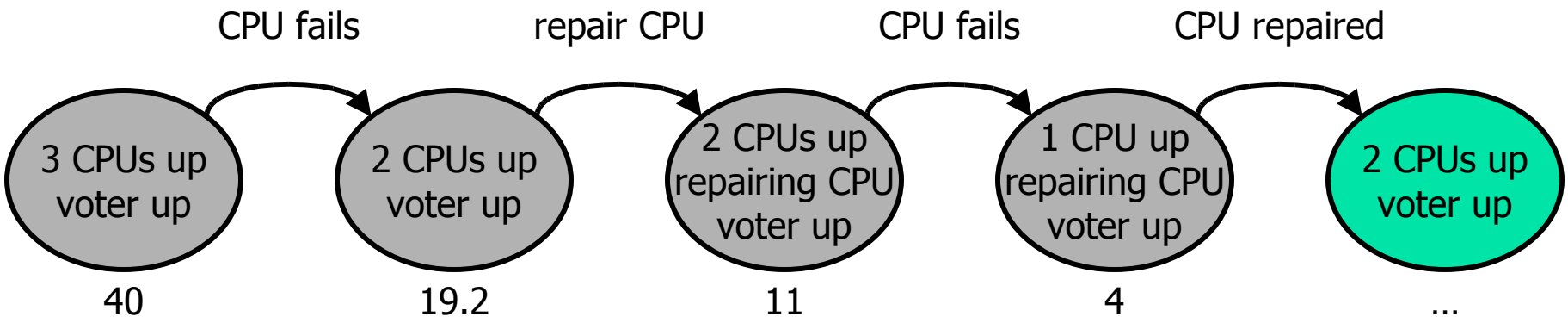

Voter
CPU 1
CPU 2
CPU 3

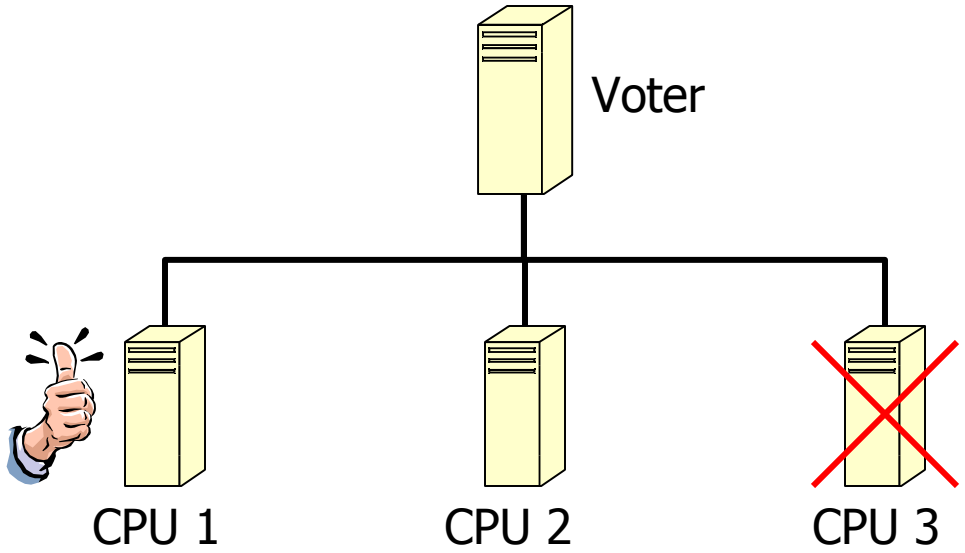
DES Example


CPU fails
repair CPU
CPU fails
3 CPUs up
voter up
2 CPUs up
voter up
2 CPUs up
repairing CPU
voter up
1 CPU up
repairing CPU
voter up
40
19.2
11

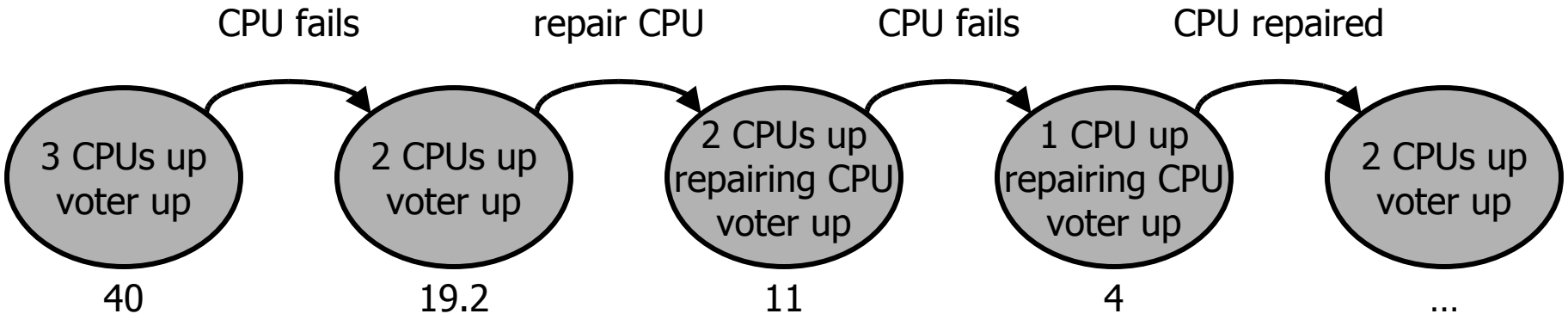

Voter
CPU 1
CPU 2
CPU 3

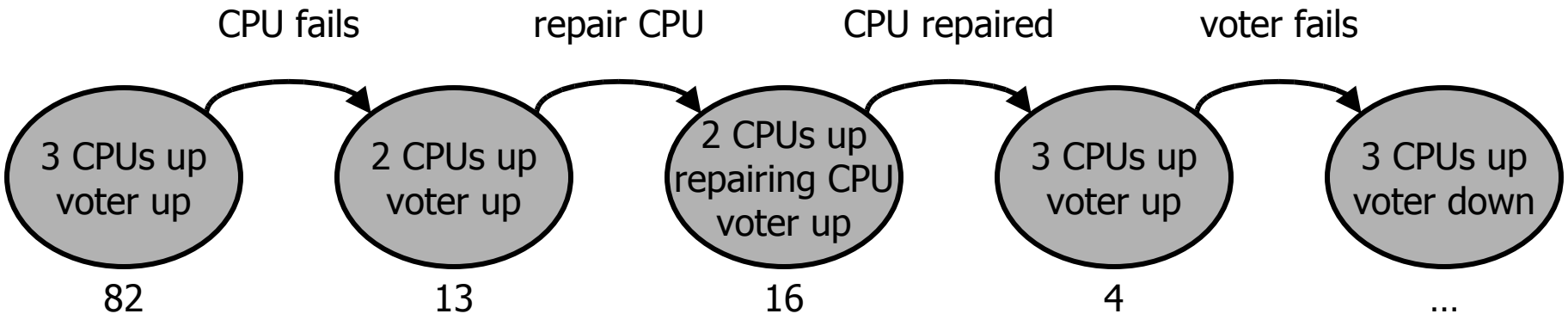
DES Example

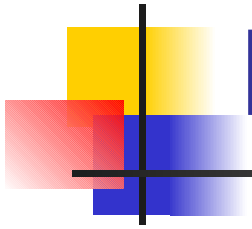

CPU fails
repair CPU
CPU fails
CPU repaired
3 CPUs up
voter up
2 CPUs up
voter up
2 CPUs up
repairing CPU
voter up
1 CPU up
repairing CPU
voter up
2 CPUs up
voter up
40
19.2
11
4
...


Voter
CPU 1
CPU 2
CPU 3

Sample Execution Paths







Properties of Interest

- Probabilistic real-time properties
  - "The probability is at least 0.1 that the voter fails within 120 time units while at least 2 CPUs have continuously remained up"

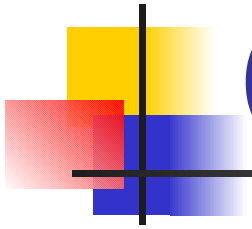
Properties of Interest

- Probabilistic real-time properties
  - "The probability is at least 0.1 that the voter fails within 120 time units while at least 2 CPUs have continuously remained up"

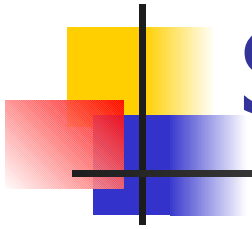
CSL formula:

$Pr_{\geq 0.1}$("2 CPUs up" $\vee$ "3 CPUs up" $U^{\leq 120}$ "voter down")

Continuous Stochastic Logic (CSL)



- State formulas
  - Truth value is determined in a single state
- Path formulas
  - Truth value is determined over an execution path



State Formulas

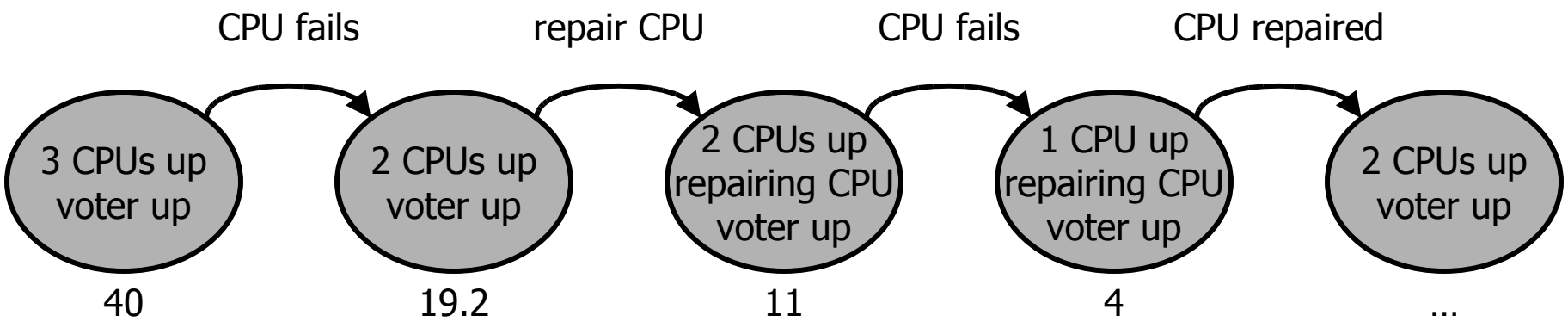
- Standard logic operators: $\neg\varphi$, $\varphi_1 \wedge \varphi_2$ ...
- Probabilistic operator: $Pr_{\geq\theta}(\rho)$
  - True iff probability is at least θ that ρ holds

Path Formulas

- Until: $\varphi_1 \; U^{\leq t} \; \varphi_2$
  - Holds iff φ_2 becomes true in some state along the execution path before time t, and φ_1 is true in all prior states

Verifying Real-time Properties

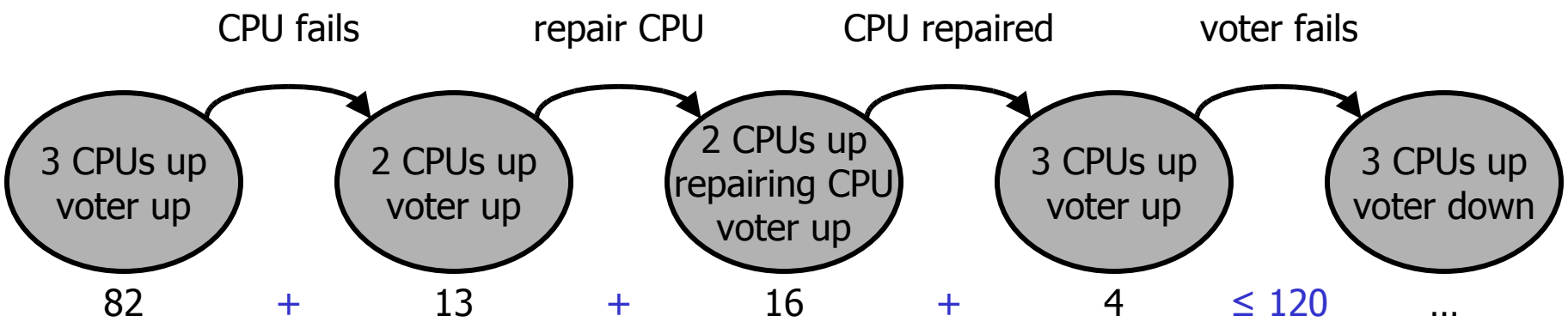
- “2 CPUs up” $\vee$ “3 CPUs up” $U^{\leq 120}$ “voter down”



False!

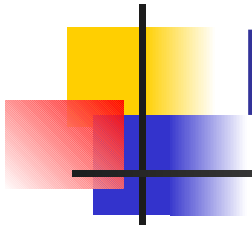
Verifying Real-time Properties

- "2 CPUs up" $\vee$ "3 CPUs up" $U^{\leq 120}$ "voter down"

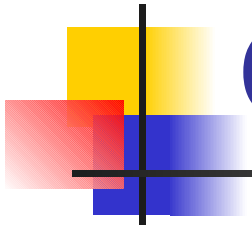


True!

Verifying Probabilistic Properties

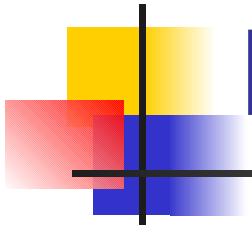


- "The probability is at least 0.1 that ρ"
  - Symbolic Methods
    - Pro: Exact solution
    - Con: Works only for restricted class of systems
  - Sampling
    - Pro: Works for any system that can be simulated
    - Con: Uncertainty in correctness of solution



Our Approach

- Use simulation to generate sample execution paths
- Use sequential acceptance sampling to verify probabilistic properties



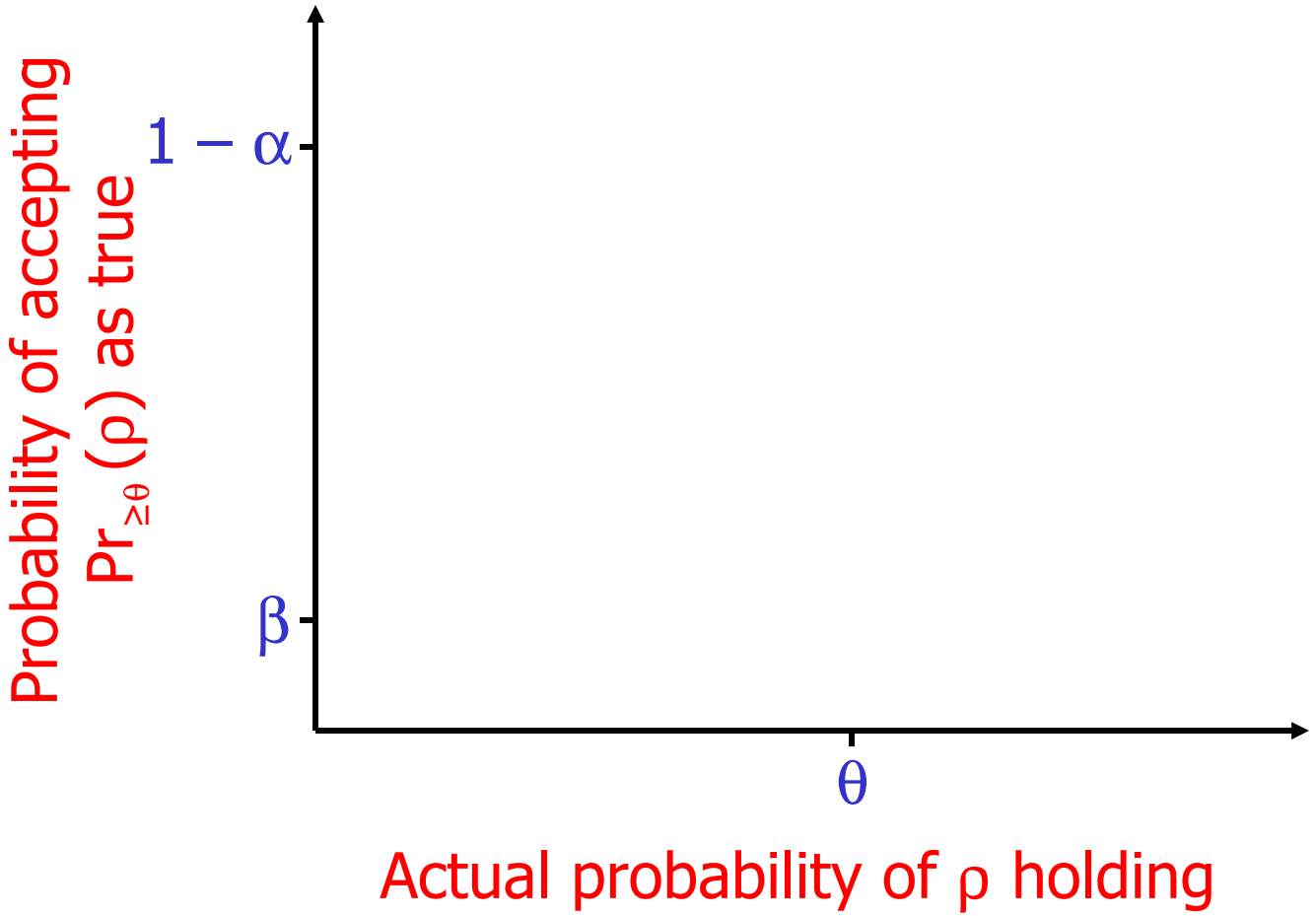
Error Bounds

- Probability of false negative: $\leq \alpha$
  - We say that φ is false when it is true
- Probability of false positive: $\leq \beta$
  - We say that φ is true when it is false

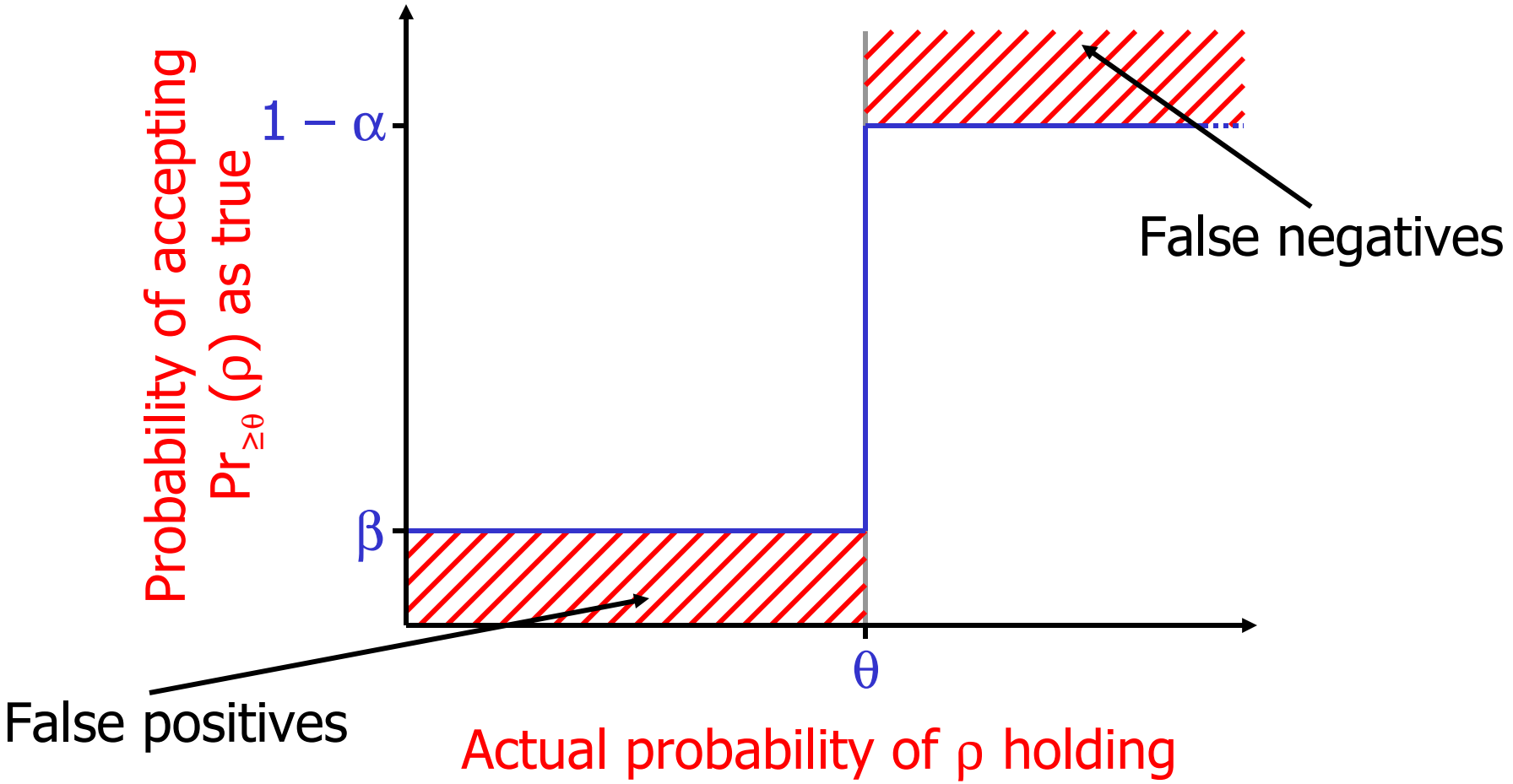
Acceptance Sampling

- Hypothesis: $Pr_{\geq\theta}(\rho)$

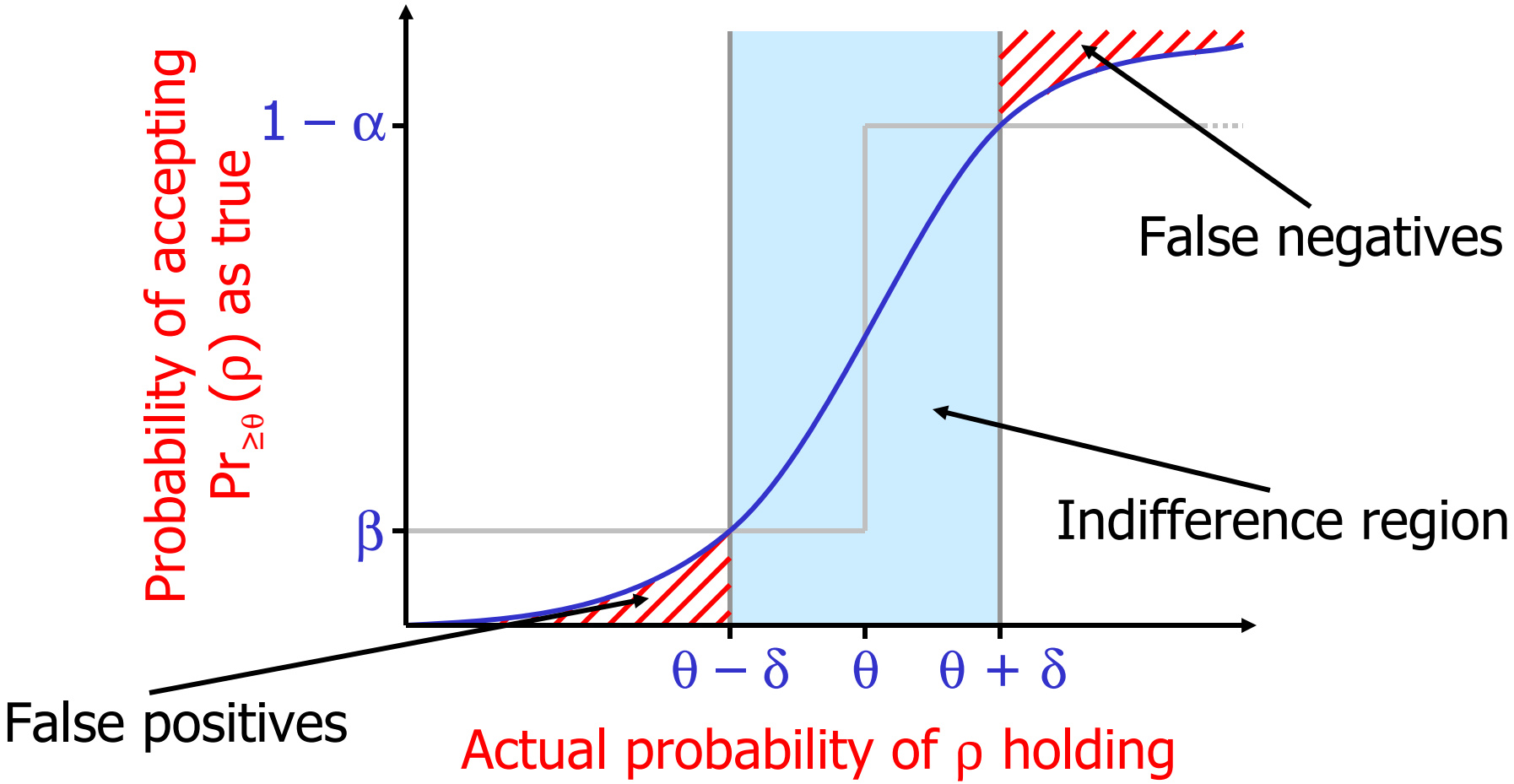
Performance of Test



Ideal Performance

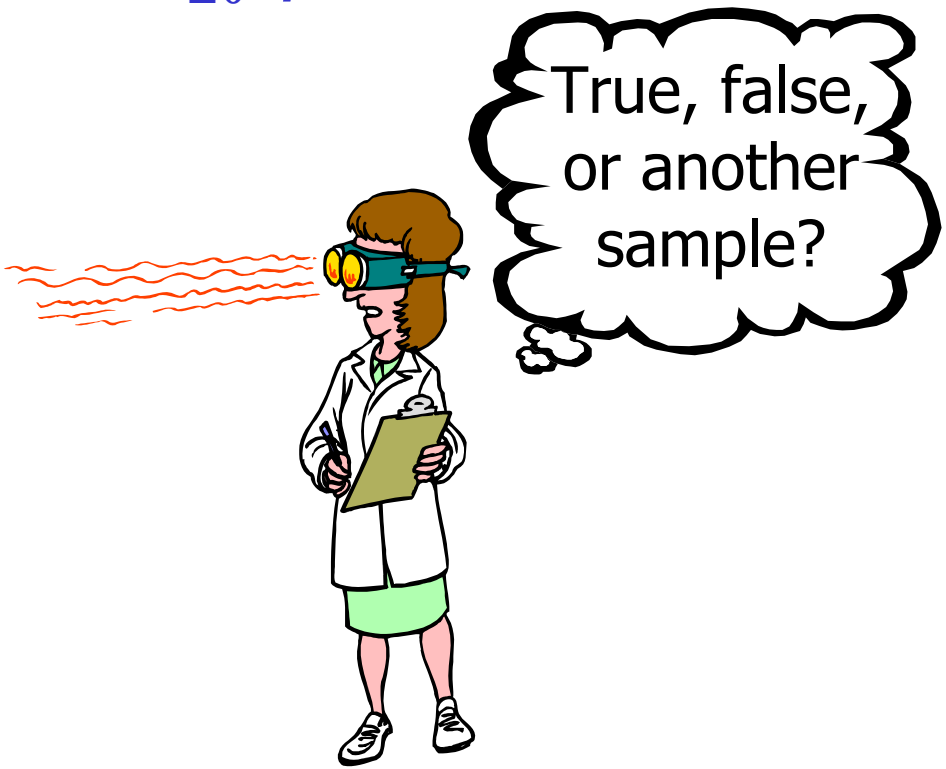


Actual Performance

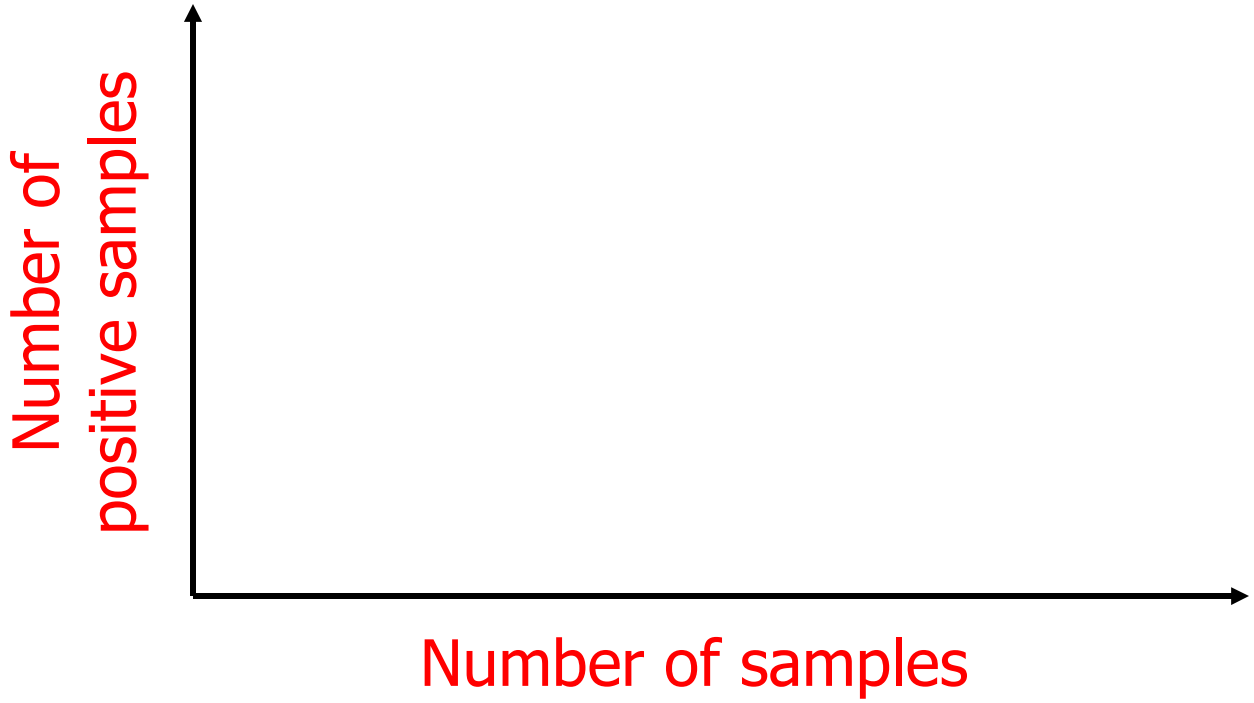


Sequential Acceptance Sampling

- Hypothesis: $Pr_{\geq\theta}(\rho)$

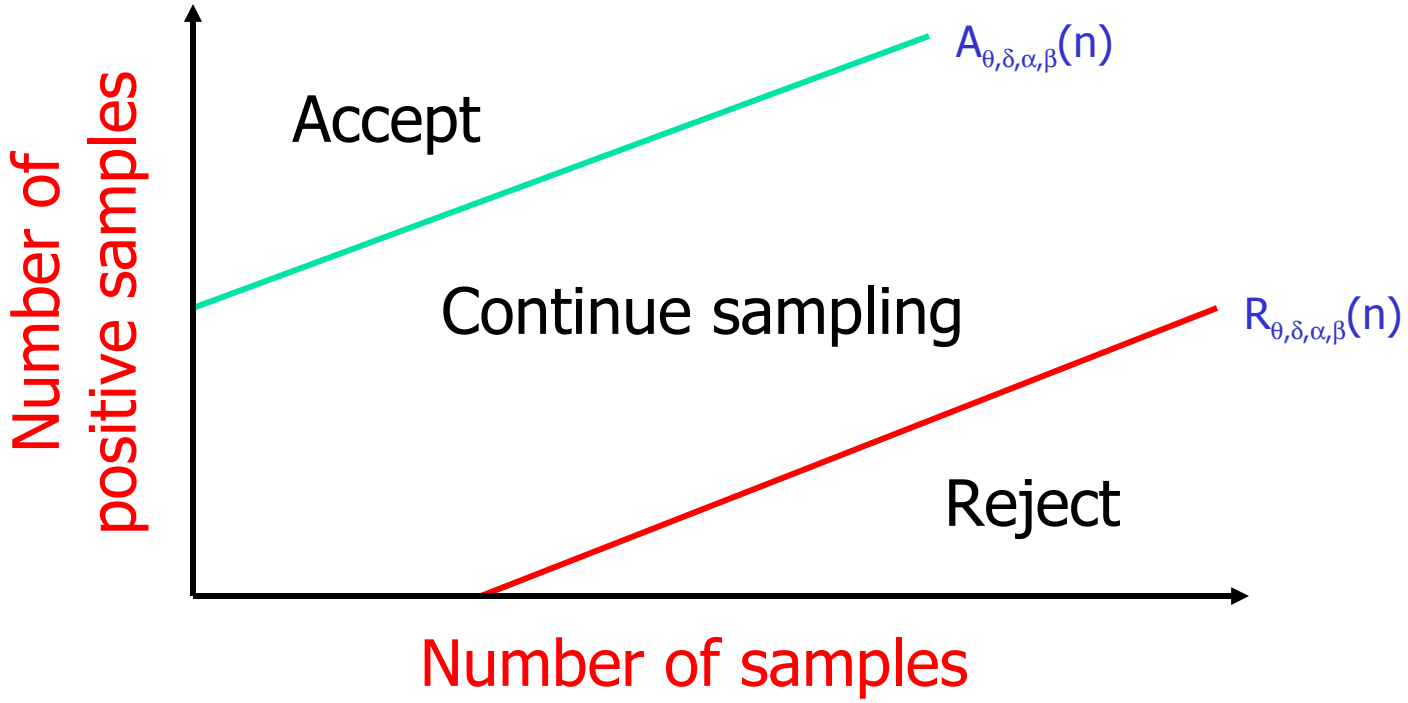


Graphical Representation of Sequential Test



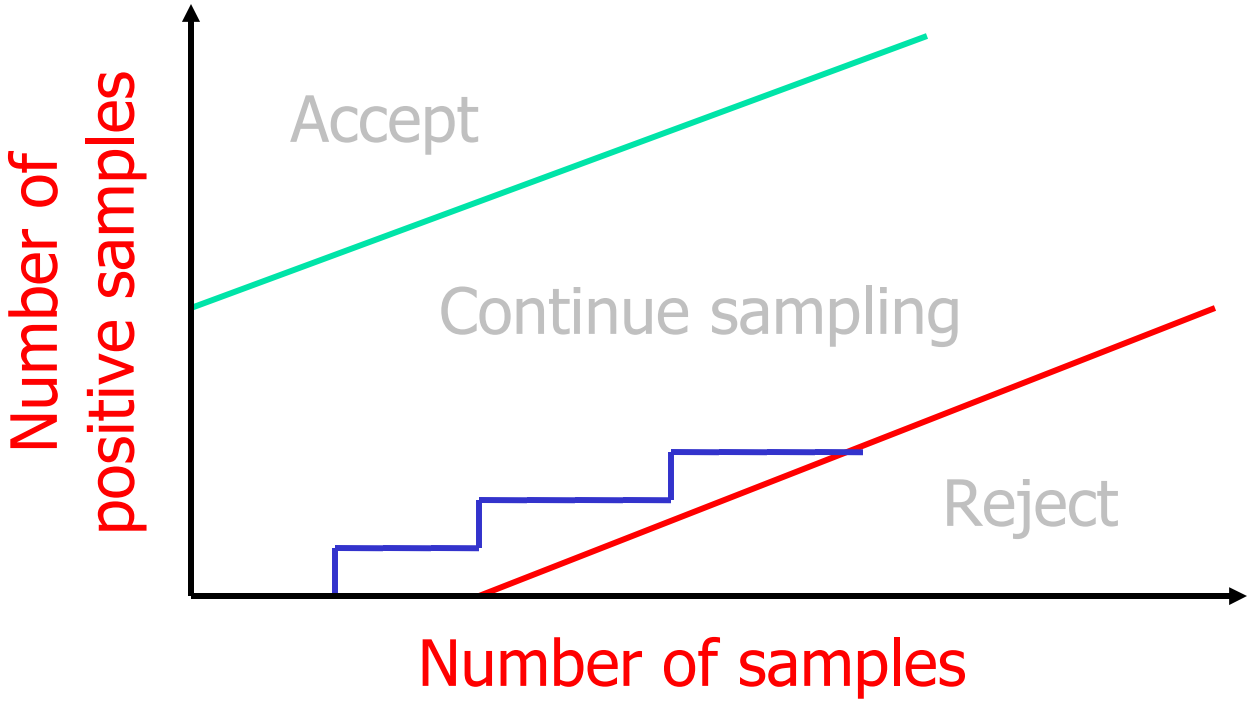
Graphical Representation of Sequential Test

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



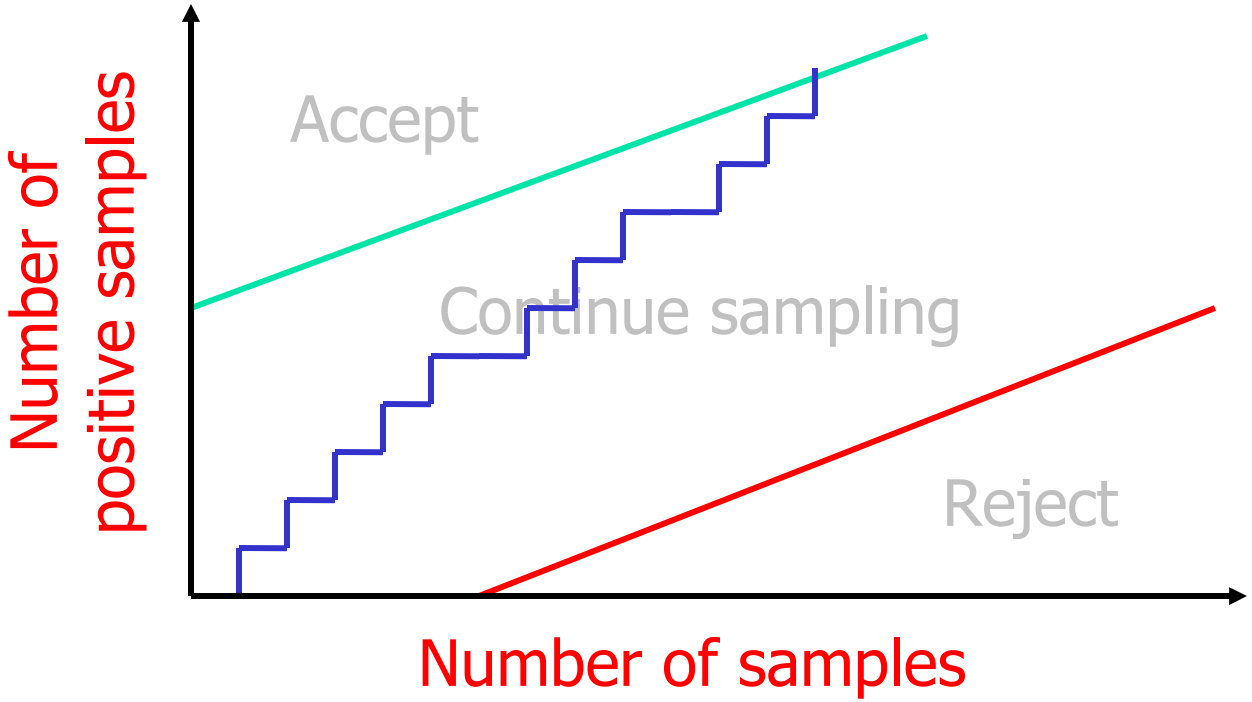
Graphical Representation of Sequential Test

- Reject hypothesis

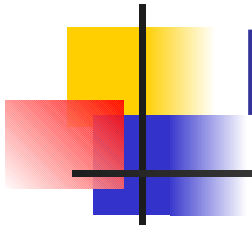


Graphical Representation of Sequential Test

- Accept hypothesis

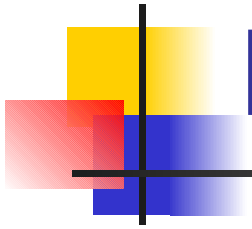


Verifying Probabilistic Properties



- Verify $Pr_{\geq\theta}(\rho)$ with error bounds α and β
  - Generate sample execution paths using simulation
  - Verify ρ over each sample execution path
    - If ρ is true, then we have a positive sample
    - If ρ is false, then we have a negative sample
  - Use sequential acceptance sampling to test the hypothesis $Pr_{\geq\theta}(\rho)$

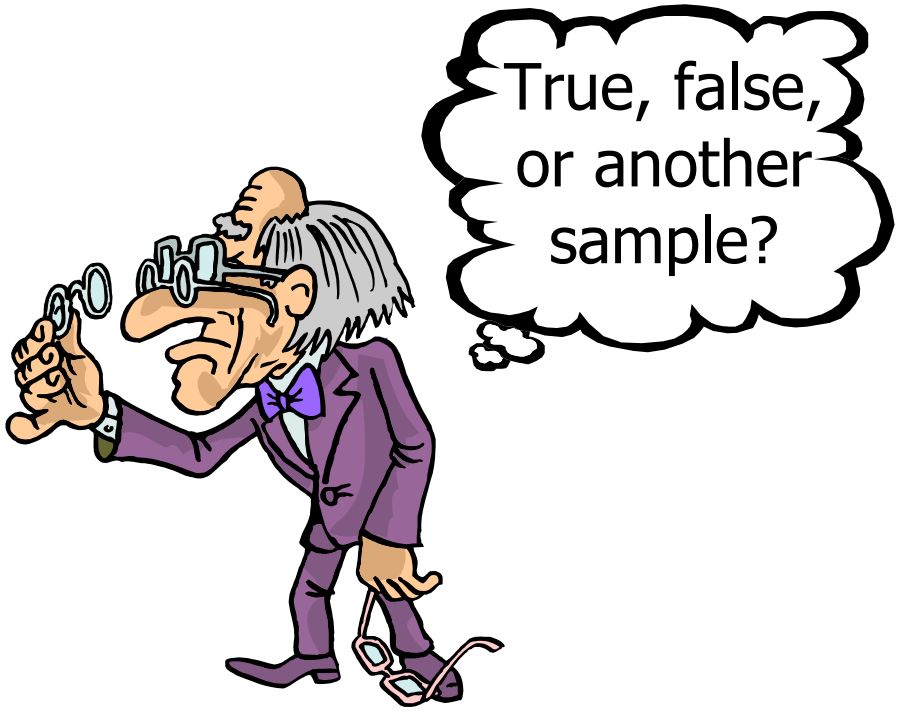
Verification of Nested Probabilistic Statements



- Suppose ρ, in $Pr_{\geq\theta}(\rho)$, contains probabilistic statements
  - Error bounds α' and β' when verifying ρ

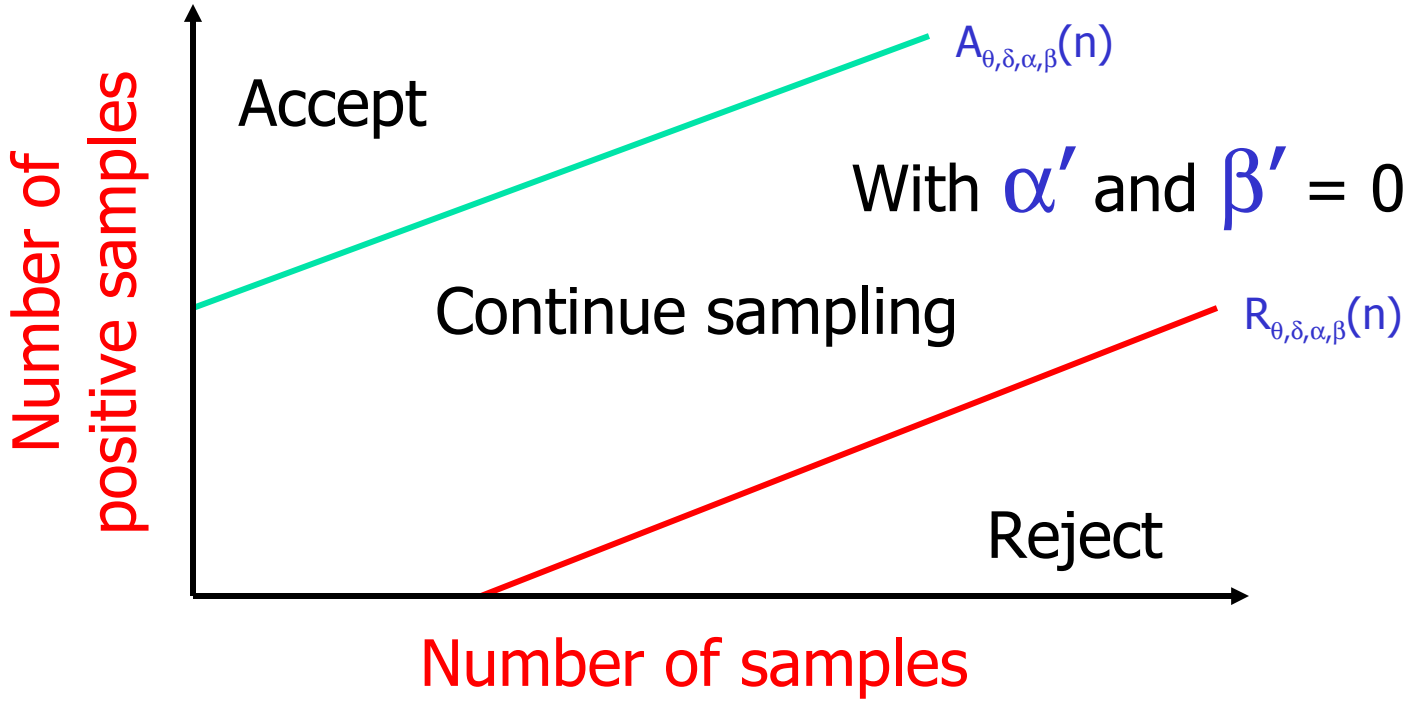
Verification of Nested Probabilistic Statements

- Suppose ρ, in $Pr_{\geq\theta}(\rho)$, contains probabilistic statements



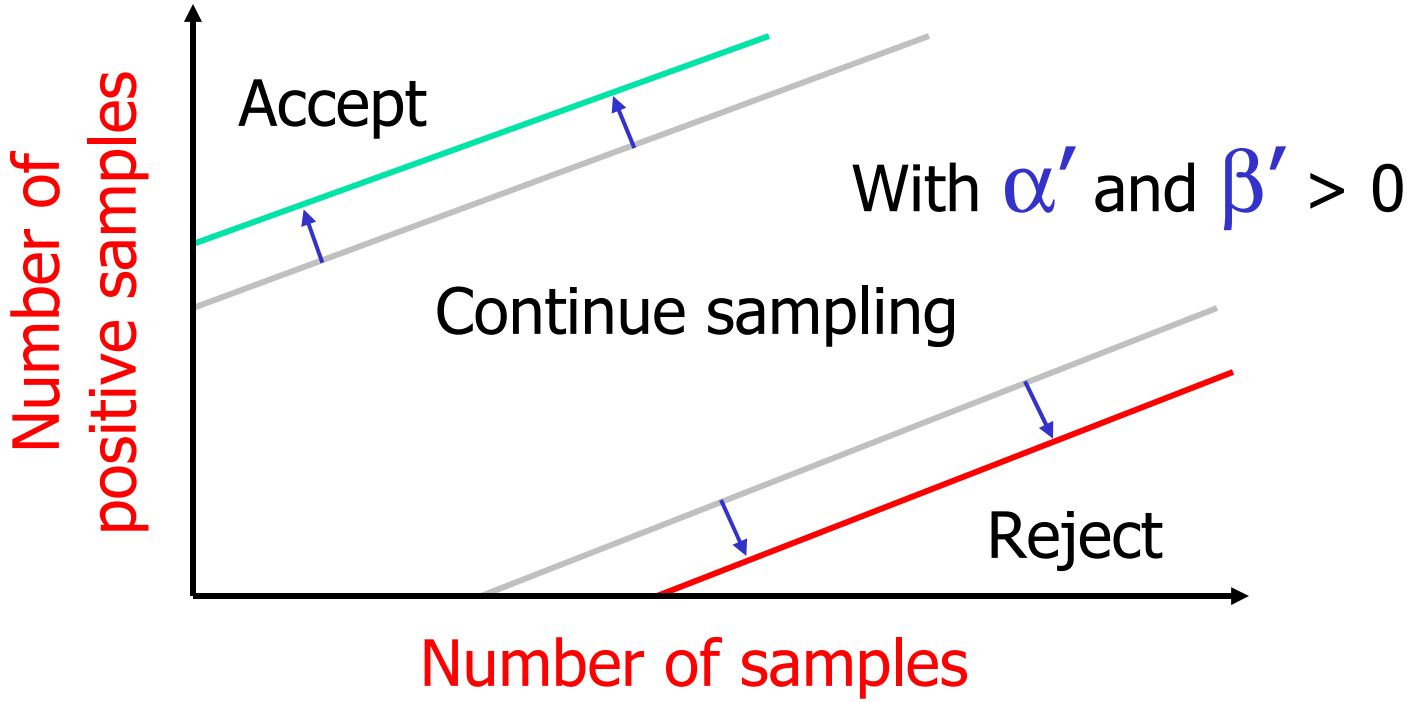
Modified Test

- Find an acceptance line and a rejection line given θ, δ, α, β, α', and β':



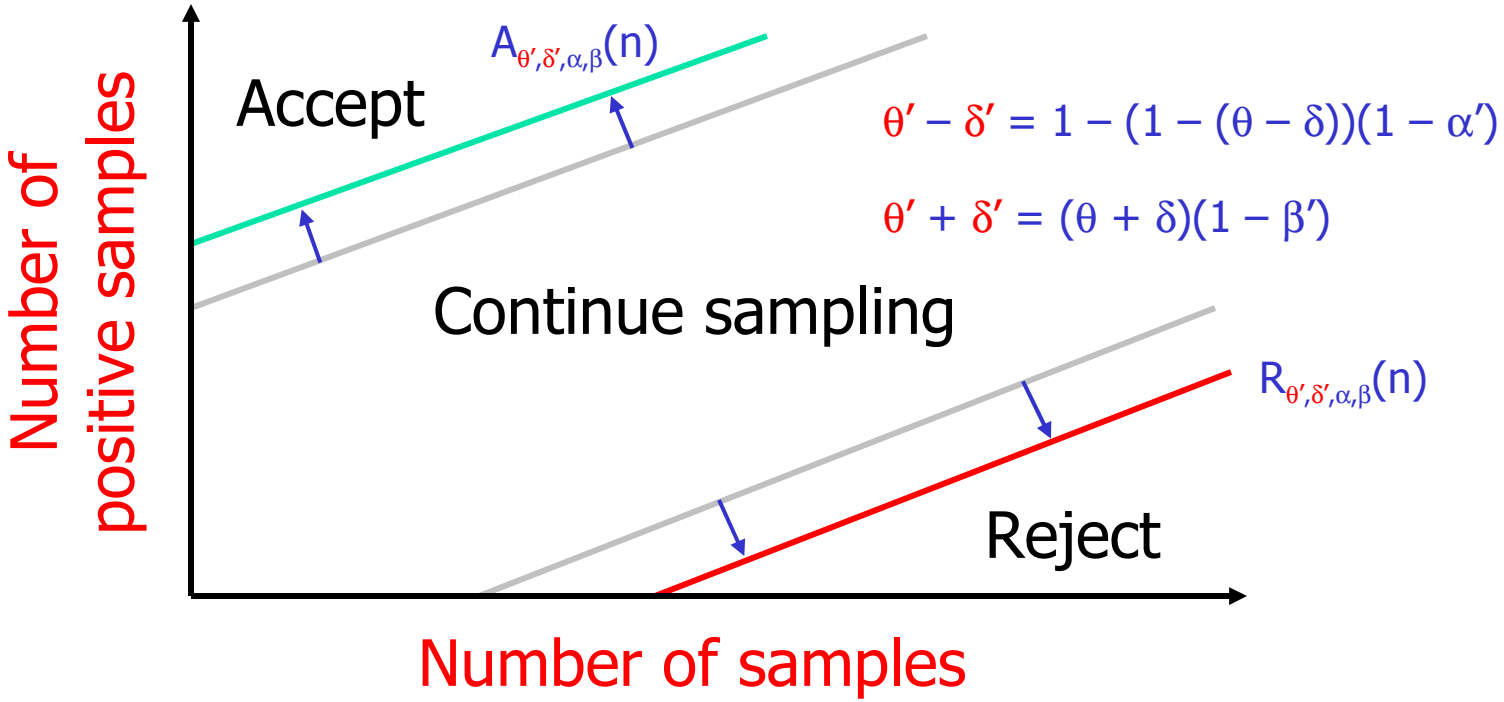
Modified Test

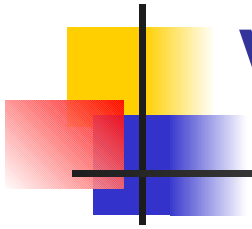
- Find an acceptance line and a rejection line given θ, δ, α, β, α', and β':



Modified Test

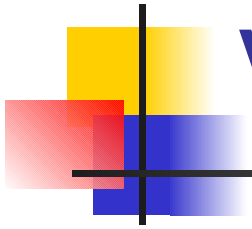
- Find an acceptance line and a rejection line given θ, δ, α, β, α', and β':





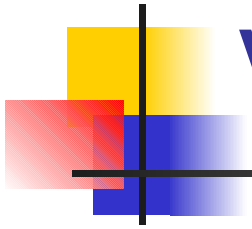
Verification of Negation

- To verify $\neg\varphi$ with error bounds α and β
  - Verify φ with error bounds β and α



Verification of Conjunction

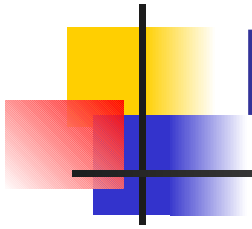
- Verify $\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n$ with error bounds α and β
  - Verify each φ_i with error bounds α and β/n



Verification of Path Formulas

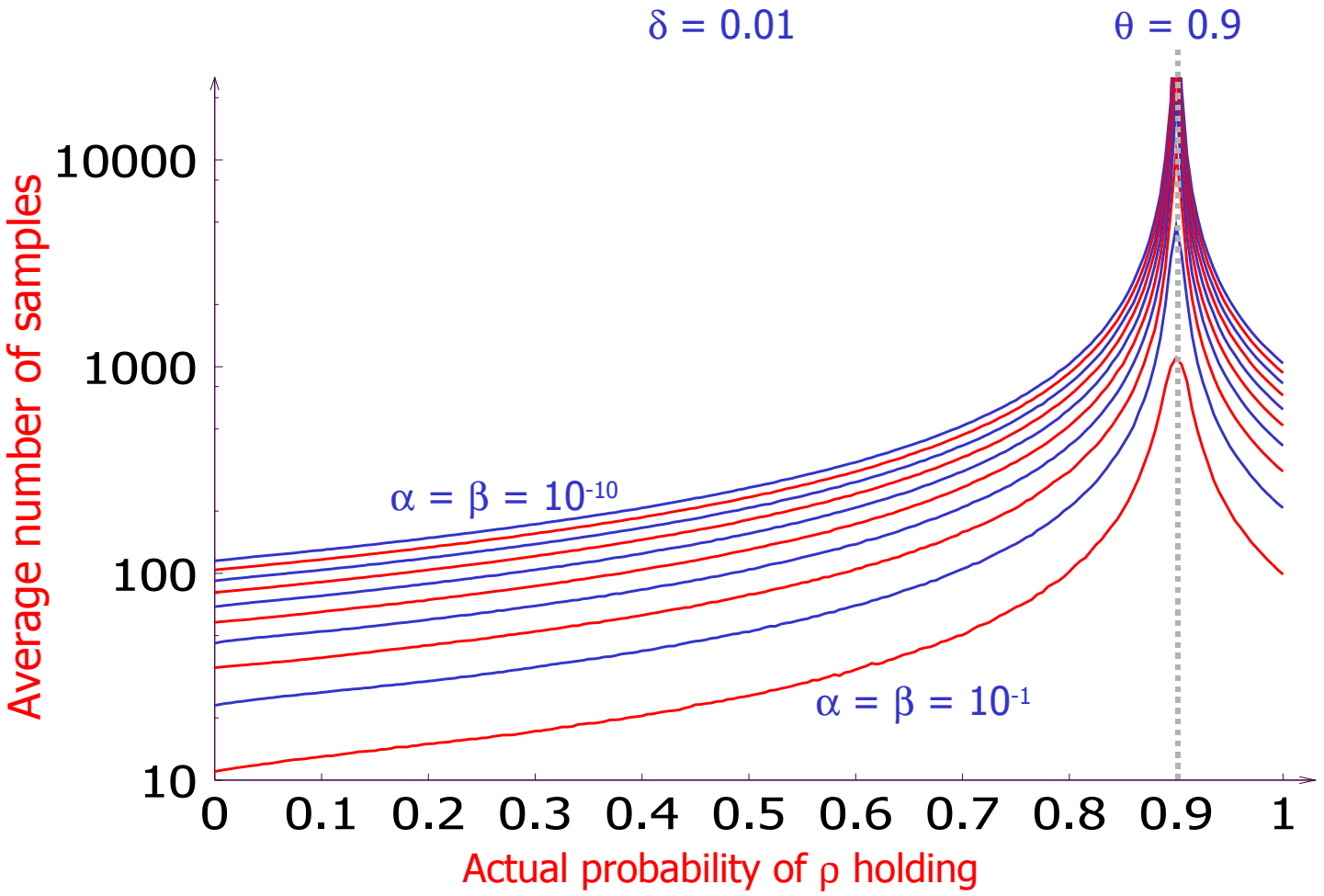
- To verify $\varphi_1 \, U^{\leq t} \, \varphi_2$ with error bounds α and β
  - Convert to disjunction
    - $\varphi_1 \, U^{\leq t} \, \varphi_2$ holds if φ_2 holds in the first state, or if φ_2 holds in the second state and φ_1 holds in all prior states, or …

More on Verifying Until

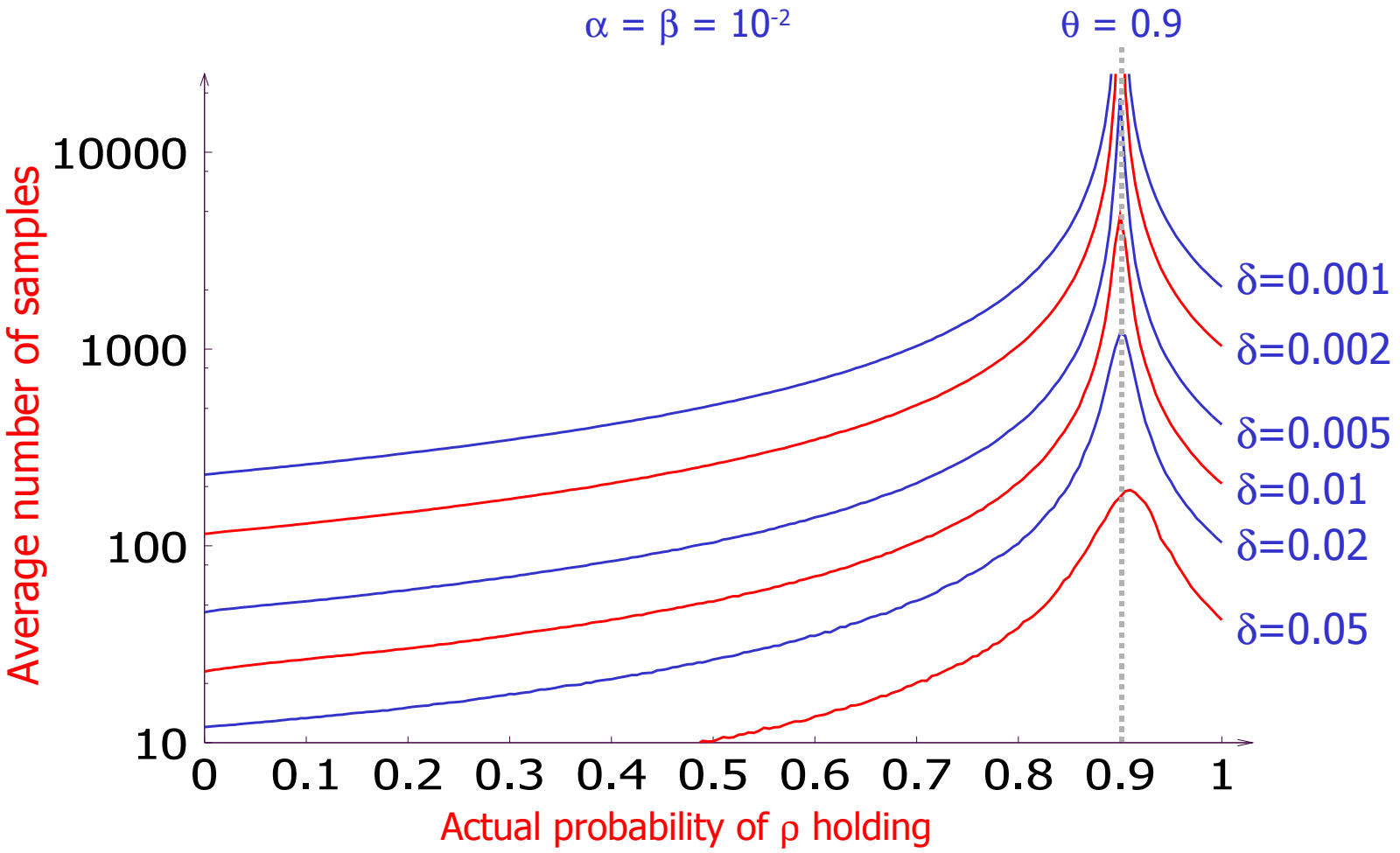


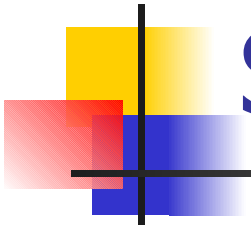
- Given $\varphi_1 \, U^{\leq t} \, \varphi_2$, let n be the index of the first state more than t time units away from the current state
- Disjunction of n conjunctions c_1 through c_n, each of size i
- Simplifies if φ_1 or φ_2, or both, do not contain any probabilistic statements

Performance



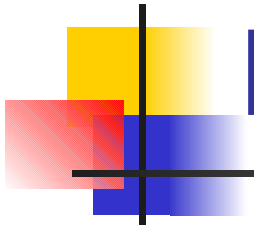
Performance





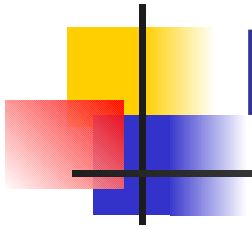
Summary

- Model independent probabilistic verification of discrete event systems
- Sample execution paths generated using simulation
- Probabilistic properties verified using sequential acceptance sampling
- Easy to trade accuracy for efficiency



Future Work

- Develop heuristics for formula ordering and parameter selection
- Use in combination with symbolic methods
- Apply to hybrid dynamic systems
- Use verification to aid controller synthesis for discrete event systems



ProVer: Probabilistic Verifier

http://www.cs.cmu.edu/~lorens/prover.html