Automated Verification of Cyber-Physical Systems

A.A. 2025/2026

Corso di Laurea Magistrale in Informatica

Statistical Model Checking

Igor Melatti

Università degli Studi dell'Aquila

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Statistical Model Checking

- In the recent literature, there is not full consensus, as it may refer to:
  - estimate the probability of a BLTL property of a stochastic system
    - in a stochastic system, transitions between states are not deterministic, but probabilistic
  - estimate the probability of an LTL property of a deterministic system
    - thus, input is the same of a classical model checking problem
    - but output is "probabilistic"
    - also referred to as *quantitative model checking* or *monte-carlo model checking*

Some Background

- In our context, a *random variable* is a function from some event space Ω to $\mathbb{R}$
  - $X : \Omega \to \mathbb{R}$
- Suppose we have a probability $\mathbb{P}$ defined on 2^{Ω}
  - thus, $\mathbb{P}$ is defined on sets of events $E \subseteq \Omega$
  - recall the Kolmogorov axioms:
    - $\forall E \subseteq \Omega.\ \mathbb{P}(E) \in [0, 1]$
    - $\mathbb{P}(\Omega) = 1$
    - $\forall I \subseteq \mathbb{N} : (E_i \in 2^{\Omega} \wedge \forall i \neq j \in I.\ E_i \cap E_j = \varnothing) \to \mathbb{P}(\cup_{i=1}^{\infty} E_i) = \sum_{i=1}^{\infty} \mathbb{P}(E_i)$
- The *mean* of a random variable, also called *expected value*, is defined as $\mu_X = \mathbb{E}[X] = \sum_{\omega \in \Omega} p(\omega) X(\omega)$
  - here $p(\omega) = \mathbb{P}(\{\omega\})$
  - by the axioms above, $p(\omega) \in [0, 1]$ and $\sum_{\omega \in \Omega} p(\omega) = 1$
  - if Ω is continuously infinite, then an integral should be used instead

Some Background

- A *Bernoulli random variable* Z on Ω is s.t. $Z : \Omega \to \{0, 1\}$
  - i.e., for each outcome, Z can say yes/no
    - think about paths on a system, with Z saying correct/incorrect...
  - we simply write Z instead of $Z(x)$
  - given $\mathbb{P}$ on 2^{Ω}, we define $p_Z = \mathbb{P}(Z = 1)$ and $q_Z = \mathbb{P}(Z = 0) = 1 - p_Z$
  - if $\Omega = \{0, 1\}$ and $Z(x) = x$, then $\mu_Z = \mathbb{E}[Z] = Z(1)p_Z + Z(0)q_Z = p_Z$

Some Background

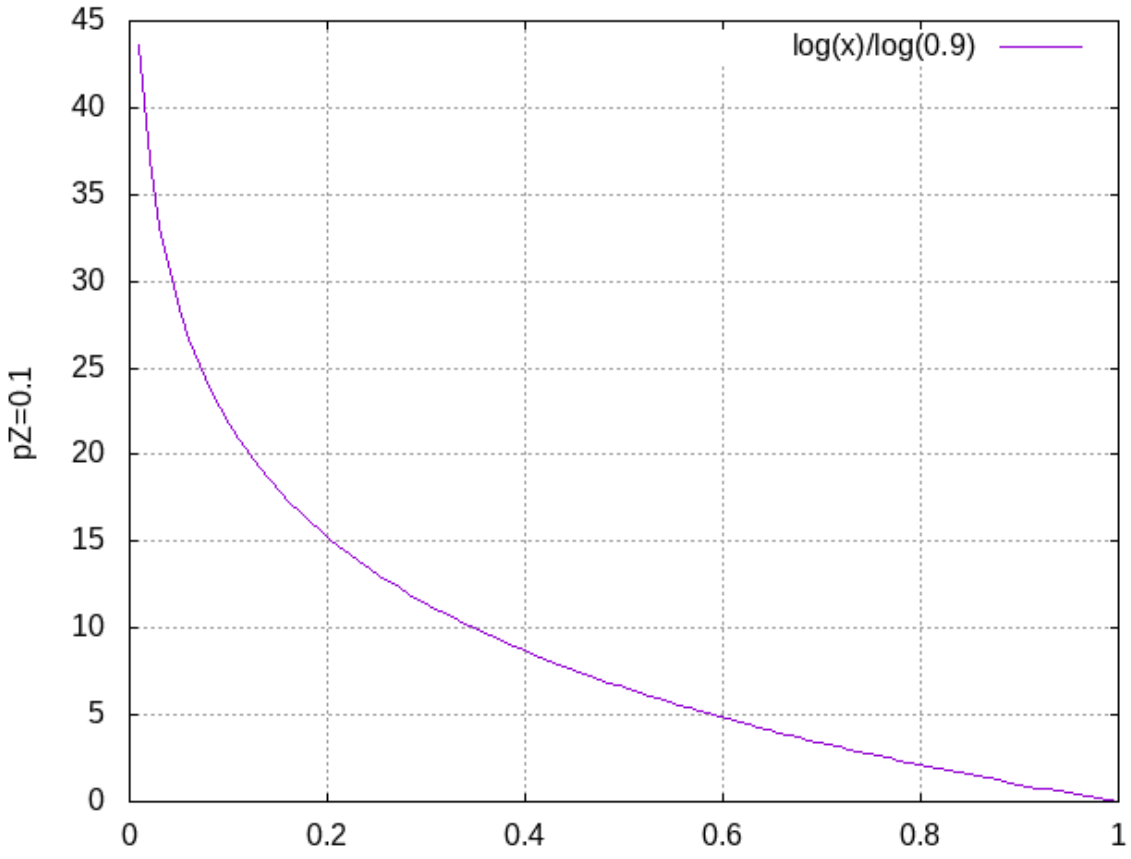
- A *Bernoulli process* consists in repeatedly running independent trials on a Bernoulli variable Z
  - either finite or infinite sequence of trials
  - "independent" means that the probability of outcome $o_1 \dots o_n$ is $\prod_{i=1}^{n} p(o_i)$
  - if there are k outcomes such that $o_i = 1$, then $\mathbb{P}(\{o_1 \dots o_n\}) = p_Z^k q_Z^{n-k} = p_Z^k (1 - p_Z)^{n-k}$
- We can define a geometric random variable X_Z s.t. $X : \Omega^\infty \to \mathbb{N}$
  - $X_Z(\omega) = n$ iff $Z = 1$ for the first time after exactly n independent trials (with probability p_Z)
- Thus, $\mathbb{P}(X_Z = N) = q_Z^{N-1} p_Z$
  - as a consequence, $\mathbb{P}(X_Z \leq N) = \sum_{n=1}^{N} q_Z^{n-1} p_Z = 1 - q_Z^N$
  - complementary to the probability of having N failures...

Some Background

- Suppose now that, in some way, you know the value of p_Z
- How many trials would we need to see $Z = 1$?
- Well, in these terms, you would need infinitely many trials
  - special case 1: you can't see $Z = 1$ if $p_Z = 0$
  - special case 2: you see $Z = 1$ after 1 trial if $p_Z = 1$
  - we are interested in $0 < p_Z < 1$
  - strictly speaking, this is actually true only for finite Ω
- Let's relax a bit: how many trials would we need to see $Z = 1$ *with a given confidence* $1 - \delta$?
  - e.g.: I want to be 90% sure, so $\delta = 0.1$
- We have $\mathbb{P}(X_Z \leq N) = 1 - (1 - p_Z)^N \geq 1 - \delta$
  - solving N as a function of δ and p_Z, we have $N \geq \frac{\log(\delta)}{\log(1-p_Z)}$
  - note that both numerator and denominator are negative, as $0 < \delta, p_Z < 1$

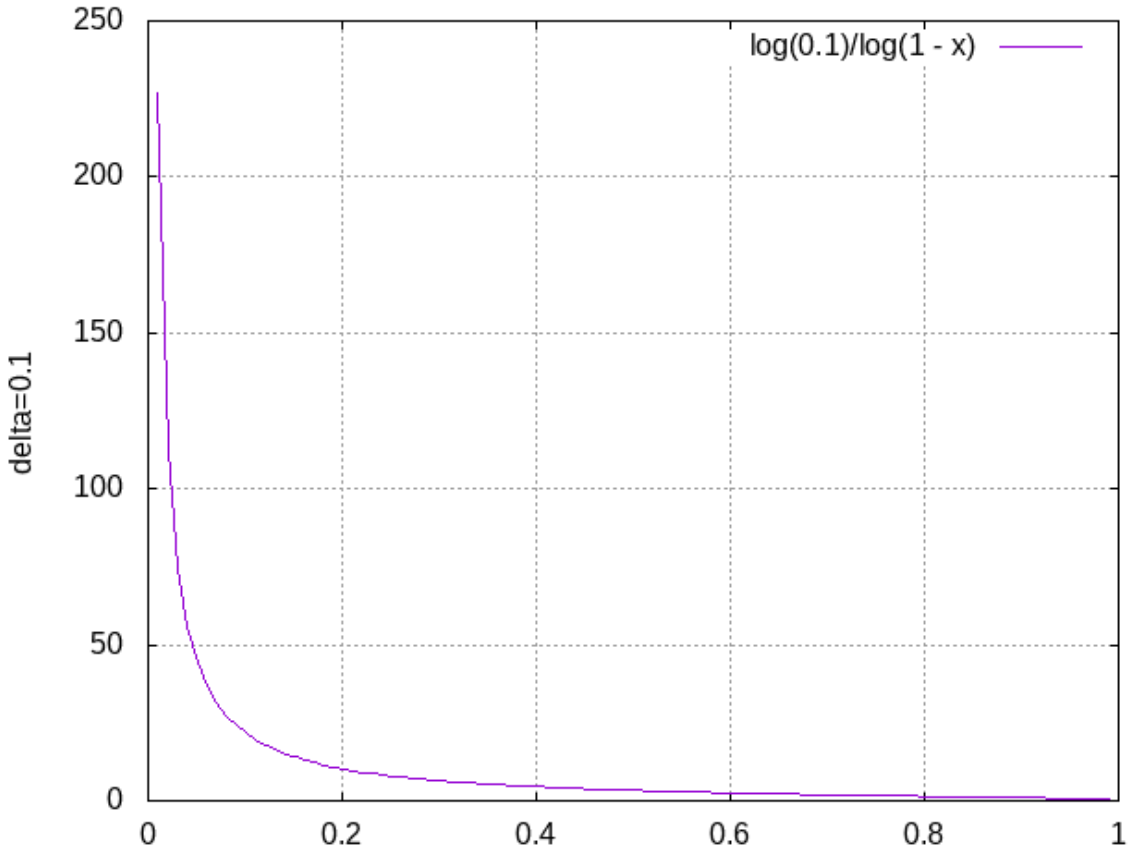
Some Background

- For fixed p_Z, N decreases with δ
  - i.e., increases with $1 - \delta$
  - you are ok if you are less confident? you can try less
  - you want to be more confident? you have to try more



Some Background

- For fixed δ, N decreases with p_Z
  - you want to detect something with big probability? you can try less
  - you want to detect something with small probability? you have to try more



Some Background

- ... But we do not know p_Z
  - indeed, it is exactly what we want to estimate by making trials
- Computing the precise value p_Z may be difficult, let us simply ask if it is big enough
  - that is, we expect the experiment outcome to be nearly always 0
  - we may choose some value $\varepsilon > 0$ of interest and test if $p_Z \geq \varepsilon$
  - ε is our *error margin*, $H_0 \equiv (p_Z \geq \varepsilon)$ is the *null hypothesis*
- We have that $M = \frac{\log(\delta)}{\log(1-\varepsilon)} \geq \frac{\log(\delta)}{\log(1-p_Z)} = N$
- Recalling the steps before, we have $\mathbb{P}(X_Z \leq M) \geq \mathbb{P}(X_Z \leq N) \geq 1 - \delta$
- Thus: $p_Z \geq \varepsilon$ *implies* $\mathbb{P}\left(X_Z \leq \frac{\log(\delta)}{\log(1-\varepsilon)}\right) \geq 1 - \delta$
  - using conditional probabilities and putting M back in, we have $\mathbb{P}\left(X_Z \leq M \mid p_Z \geq \varepsilon\right) \geq 1 - \delta$

Some Background

- Suppose we want to decide if $H_0 \equiv p_Z \geq \varepsilon$ holds (*hypothesis testing*)
- We perform $M = \left\lceil \frac{\log(\delta)}{\log(1-\varepsilon)} \right\rceil$ trials on Z
  - if we never see $Z = 1$, then we reject H_0
  - otherwise, we accept H_0
- There are 4 possible "higher outcomes"
  - *type-I error*: H_0 is rejected, but $p_Z \geq \varepsilon$ holds
  - *type-II error*: H_0 is accepted, but $p_Z < \varepsilon$ holds
  - we were right in rejecting/accepting H_0 (2 cases)
- The probability of a type-I error is denoted by α, the probability of a type-II error is β
  - generally speaking, they could be dependent on each other
- We have $\alpha = \mathbb{P}(X > M \mid H_0) = 1 - \mathbb{P}(X \leq M \mid H_0) \leq \delta$
  - since $\mathbb{P}(X \leq M \mid H_0) \geq 1 - \delta$

"Easy" Statistical Model Checking

- Ω is a set of simulation outputs, given some stochastic input variations
  - typically, $|\Omega| = \infty$
- $Z : \Omega \to \{0, 1\}$ is a *monitor* which tells if a simulation is ok or not
  - we have some property ϕ to be verified
  - for a simulation $\omega \in \Omega$, $Z(\omega) = 1$ iff $\omega \not\models \phi$
  - for most simulations ω, $Z(\omega) = 0$ since some "simple" testing has already been done
  - easy errors have already been detected
- p_Z is the probability that we have an error in our system!
- $p_Z = 0$ if:
  - there are no errors
  - there are errors, but their probability is zero
    - e.g., there are only a finite amount of ω s.t. $Z(\omega) = 1$

UNIVERSITÀ DEGLI STUDI DELL'AQUILA

DISIM
Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

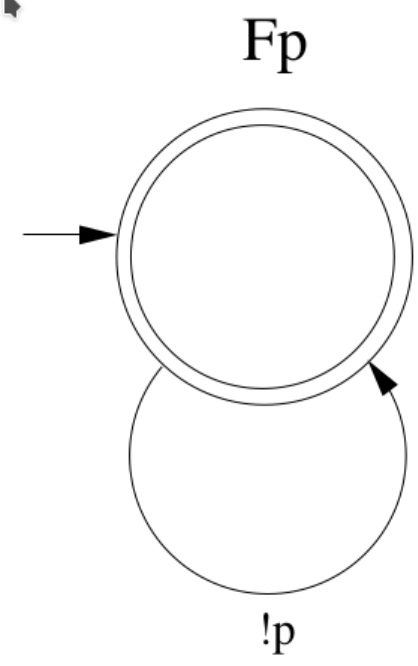
"Easy" Statistical Model Checking

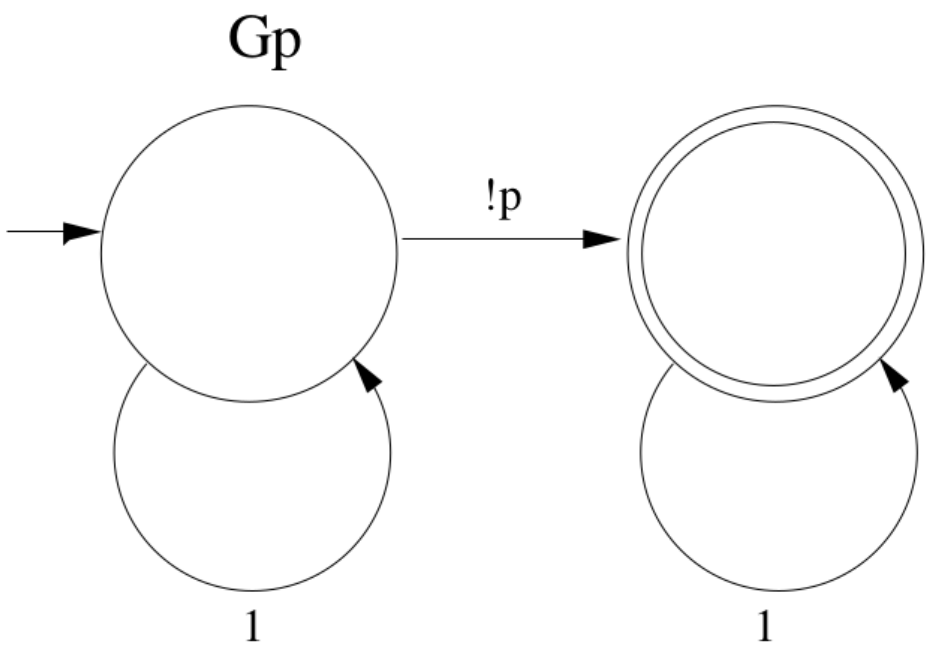
- Null hypothesis: "the probability that there is an error is $\geq \varepsilon$"
- Run M simulations, if you see an error you are happy
  - we are always correct in accepting the hypothesis, no type-II errors
  - or better: we do not care about type-II errors, as having a single ω s.t. $Z(\omega) = 1$ is enough for verification purposes
- Otherwise, you say there are no errors
- The probability you are wrong is less than δ
  - or, you are at least $1 - \delta$ confident you are right when you say there are no errors
- But all this holds only if $p_Z \geq \varepsilon$
  - will be back on this

LTL Monte-Carlo Model Checking

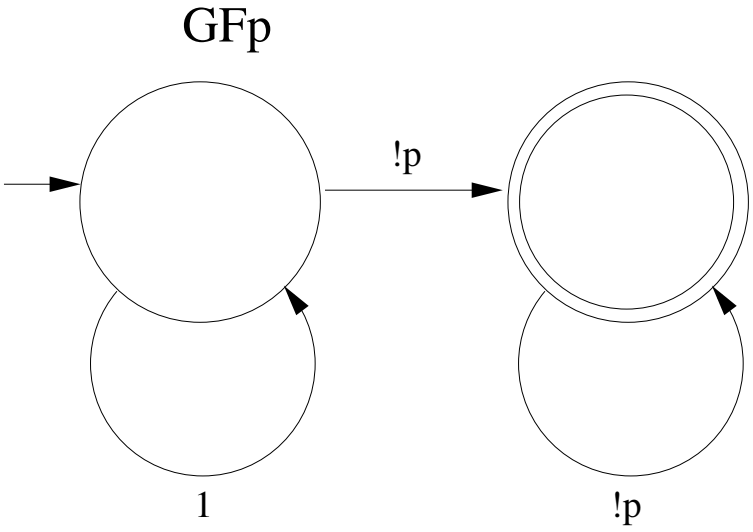
- Grosu, Smolka: "Monte Carlo Model Checking", Proc. of TACAS 2005
- In LTL Monte-Carlo Model Checking, the first part of the input is as in standard LTL Model Checking:
  - a Kripke structure $\mathcal{S} = \langle S, I, R, L \rangle$
  - an LTL formula φ
  - let us say that we directly have the Büchi Automaton $B = B_{\neg\varphi} \times B_{\mathcal{S}}$
    - as it is computed by explicit on-the-fly model checkers like SPIN
- Then, we also have two additional inputs: $0 < \delta, \varepsilon < 1$
- Output as in standard LTL Model Checking:
  - either PASS...
  - ... or FAIL with a counterexample

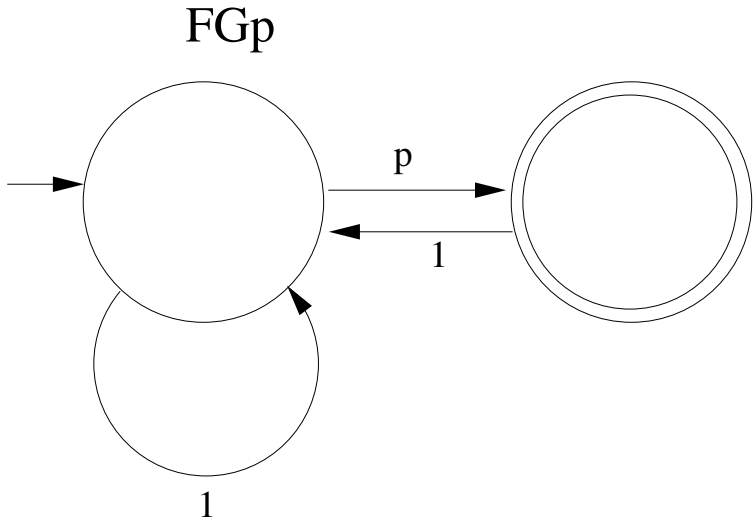
LTL (Counter)Examples





LTL (Counter)Examples





Nested DFS for LTL Model Checking

```
DFS(KS_BA 𝒮𝒜, state (s,q), bool n, state a) {
  let 𝒮𝒜 = ⟨S_A, I_A, R_A, L_A⟩;
  foreach (s',q') ∈ S_A s.t. ((s,q),(s',q')) ∈ R_A {
    if (n ∧ (s,q) == a)
      exit reporting error;
    if ((s',q',n) ∉ T) {
      T = T ∪ {(s',q',n)};
      DFS(𝒮𝒜, (s',q'), n, a);
      if (¬n ∧ (s',q') is accepting) {
        DFS(𝒮𝒜, (s',q'), true, (s',q'));
} } } }

LTLMC(KS 𝒮, LTL φ) {
  𝒜 = BA_from_LTL(φ); T = ∅;
  let 𝒮 = ⟨S, I, R, L⟩, 𝒜 = ⟨Σ, Q, Δ, Q_0, F⟩;
  foreach s ∈ I, q ∈ Q_0
    DFS(𝒮 × 𝒜, (s,q), false, null);
}
```

LTL Monte-Carlo Model Checking

- If FAIL with a counterexample σ is returned, then for sure we have an error in our model
  - that is, $\mathcal{S} \not\models \varphi$ holds
  - σ is a counterexample showing that $\mathcal{S} \not\models \varphi$
- Otherwise, it may still be the case that, notwithstanding the PASS result, $\mathcal{S} \not\models \varphi$
- However, the probability that $\mathcal{S} \not\models \varphi$ is less than δ
  - indeed, this does only work with a huge assumption (which involves the remaining input ε), as we will see
  - however, the huge assumption could be made reasonable
- How is this achieved? Exactly through the steps outlined in the previous slides!

LTL Monte-Carlo Model Checking

- Recall that a (non-deterministic) Büchi Automaton (BA) is a 5-tuple $B = \langle \Sigma, Q, \Delta, Q_0, F \rangle$ where:
  - Σ is the *alphabet*, i.e., a finite set of symbols
  - Q is the finite set of states, $Q_0 \subseteq Q$ are the initial states and $F \subseteq Q$ are the final states
  - $\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation
- We suppose that $B = B_{\neg\varphi} \times B_{\mathcal{S}}$ is the Cartesian product of the Kripke structure $\mathcal{S}$ and the Büchi automaton generated from φ using known algorithms
  - e.g., as it is implemented in SPIN
- A *lasso* of B is a sequence $\sigma = q_0 x_0 q_1 \ldots q_n$ s.t.:
  - $\forall 0 \leq i < n.\ (q_i, x_i, q_{i+1}) \in \Delta$
  - $\exists 0 \leq k \leq n :\ \forall 0 \leq i, j < n-1.\ q_i \neq q_j \wedge q_n = q_k$
- A lasso is *accepting* if $\exists k \leq i \leq n :\ q_i \in F$

LTL Monte-Carlo Model Checking

- We may easily define a probability distribution on the finite runs σ of B:
  - $\mathbb{P}(q_0) = \frac{1}{|Q_0|}$
  - $\mathbb{P}(q_0 x_0 q_1 \dots q_{n-1} x_{n-1} q_n) = \mathbb{P}(q_0 x_0 q_1 \dots q_{n-1}) \frac{1}{|\Delta(q_{n-1})|}$
  - being $\Delta(q) = \{(q, x, q') \mid (q, x, q') \in \Delta\}$
  - that is: each time we have a (non-deterministic) choice, we choose one uniformly at random
- Such probability is well-defined: we may extend it to obtain a (discrete) probability space $(2^L, \mathbb{P})$
  - being $L = \{\sigma \mid \sigma \text{ is a lasso in } B\}$
  - furthermore, $L \supseteq L_a = \{\sigma \mid \sigma \text{ is an accepting lasso in } B\}$
  - $L_n = L \setminus L_a$ is the set of non-accepting lassos

LTL Monte-Carlo Model Checking

- Given $(2^L, \mathbb{P})$, our Bernoulli variable Z is defined by:
  - take one lasso σ from L, following the rules defined by $\mathbb{P}$
  - that is: make a random walk (see the algorithm below)
  - $Z = 1$ iff $\sigma \in L_a$ is accepting
- From a theoretical point of view, since $|L| < \infty$, we would be tempted to say that $p_Z = \frac{|L_a|}{|L|}$
- But this is not true, since lassos do not have the same probability, according to $\mathbb{P}$
- Thus, $p_Z = \mathbb{P}(Z = 1) = \sum_{\lambda_a \in L_a} \mathbb{P}(\lambda_a)$
  - not actually useful for computation: L_a requires generating L, which may run out of computational resources

LTL Monte-Carlo Model Checking

```
MC²(KS_BA 𝒮𝒜, double ε, double δ) {
  for i in 1..⌈log(δ)/log(1−ε)⌉
    if (SampleLasso(𝒮𝒜) == (1, σ))
      return (FAIL, σ);
  return PASS;
}
SampleLasso(KS_BA 𝒮𝒜 = ⟨Σ, Q, Δ, Q0, F⟩) {
  (i, f, H, q) = (0, 0, ∅, pick_unif_random(Q0);
  while(H(q) = ⊥) {
    H(q) = i + 1; i = i + 1;
    if (q ∈ F) f = i;
    q = pick_unif_random(Δ(q));
  }
  if (H(q) ≤ f) return (1, getCurrLasso(H));
  else return (0, ⊥);
}
```

Nested DFS for LTL Model Checking

```
DFS(KS_BA 𝒮𝒜, state (s,q), bool n, state a) {
  let 𝒮𝒜 = ⟨S_A, I_A, R_A, L_A⟩;
  foreach (s',q') ∈ S_A s.t. ((s,q),(s',q')) ∈ R_A {
    if (n ∧ (s,q) == a)
      exit reporting error;
    if ((s',q',n) ∉ T) {
      T = T ∪ {(s',q',n)};
      DFS(𝒮𝒜, (s',q'), n, a);
      if (¬n ∧ (s',q') is accepting) {
        DFS(𝒮𝒜, (s',q'), true, (s',q'));
} } } }

LTLMC(KS 𝒮, LTL φ) {
  𝒜 = BA_from_LTL(φ); T = ∅;
  let 𝒮 = ⟨S, I, R, L⟩, 𝒜 = ⟨Σ, Q, Δ, Q_0, F⟩;
  foreach s ∈ I, q ∈ Q_0
    DFS(𝒮 × 𝒜, (s,q), false, null);
}
```

LTL Monte-Carlo Model Checking

- Standard LTL Model Checking requires both time and space to be at least $O(|S|)$
  - easily billion of states, often unaffordable for real-world systems
- Here, time is $O(MD)$ and space is $O(D)$
  - being D the diameter of S, i.e., the length of the longest lasso starting from an initial state
  - $M = \left\lceil \frac{\log(\delta)}{\log(1-\varepsilon)} \right\rceil$ as usual
- No type-II errors: if we find a counterexample, we are happy
- Given the discussion on the background, if the answer is PASS, then the probability that an error is present but came undetected through the M trials is less than δ
- However, this is only true if we *assume* that $p_Z > \varepsilon$

LTL Monte-Carlo Model Checking: $p_Z \geq \varepsilon$

- Recall that $Z = 1$ iff, making a random walk on the given BA, I find an accepting lasso
  - recall also that an accepting lasso is "bad", i.e., the property does not hold in the system
- Thus, we are saying that the probability that, among all lassos I can find with a random walk, the probability that it is accepting is at least ε
- There are two cases:
  - $p_Z \geq \varepsilon$, then all what we have said before is ok: the probability that a counterexample exists is less than δ
  - the real problem is: what if $p_Z < \varepsilon$?
  - e.g., if there are not errors in the system, we have $p_Z = 0 < \varepsilon$
  - by recalling the actual definition of p_Z, we still have a "good" result: the probability of extracting an error within the systems behaviors is less than ε
  - thus in both cases we have a bound (ε or δ) on the error, though defined in two different ways

LTL Monte-Carlo Model Checking: Experimental Results

- Results for classical systems: dining philosophers and Needham-Schroeder protocol
  - for dining philosophers, two properties: one is false (with counterexample), one is false
  - Needham-Schroeder is the bugged version (with counterexample)
  - $\delta = 0.1, \varepsilon = 0.00183 \rightarrow M = 1257$
- Columns meaning:
  - `ph`: number of philosophers
  - `mr`: parameter in the Needham-Schroeder protocol
    - the bigger the value, the bigger the number of states
  - `entr`: number of entries in the hash table (RAM usage...)
  - `mxl`: max length of a lasso
  - `cxl`: length of the counterexample found
  - `M`: number of trials to find a counterexample

Unfair Dining Philosophers

Two different properties, both false: deadlock and starvation

	DDFS		MC^2			
ph	time	entr	time	mxl	cxl	M
4	0.02	31	0.08	10	10	3
8	1.62	511	0.20	25	8	7
12	3:13	8191	0.25	37	11	11
16	>20:0:0	–	0.57	55	8	18
20	–	oom	3.16	484	9	20
30	–	oom	35.4	1478	11	100
40	–	oom	11:06	13486	10	209

	DDFS		MC^2			
ph	time	entr	time	mxl	cxl	M
4	0.17	29	0.02	8	8	2
8	0.71	77	0.01	7	7	1
12	1:08	125	0.02	9	9	1
16	7:47:0	173	0.11	18	18	1
20	–	oom	0.06	14	14	1
30	–	oom	1.12	223	223	1
40	–	oom	1.23	218	218	1

Fair Dining Philosophers

Two different properties, both true: deadlock and starvation

	DDFS		MC^2		
ph	time	entr	time	mxl	avl
4	0:01	178	0:20	49	21
6	0:03	1772	0:45	116	42
8	0:58	18244	2:42	365	99
10	16:44	192476	7:20	720	234
12	–	oom	21:20	1665	564
16	–	oom	3:03:40	7358	3144
20	–	oom	19:02:00	34158	14923

	DDFS		MC^2		
ph	time	entr	time	mxl	avl
4	0:01	538	0:20	50	21
6	0:17	9106	0:46	123	42
8	7:56	161764	2:17	276	97
10	–	oom	7:37	760	240
12	–	oom	21:34	1682	570
16	–	oom	2:50:50	6124	2983
20	–	oom	22:59:10	44559	17949

Unfair Needham-Schroeder

	DDFS		MC^2			
mr	time	entr	time	mxl	cxl	M
4	0.38	607	1.68	87	87	103
8	1.24	2527	11.3	208	65	697
16	5.87	13471	10.2	223	61	612
24	18.7	39007	3:06	280	44	12370
32	36.2	85279	2:54	269	63	11012

	DDFS		MC^2			
mr	time	entr	time	mxl	cxl	M
40	1:11	158431	1:46	325	117	7818
48	2:03	264607	1:45	232	25	6997
56	3:24	409951	6:54	278	133	28644
64	5:18	600607	7:12	347	32	29982
72	–	oom	11:53	336	63	43192

LTL Quantitative Model Checking

- Grosu, Smolka: "Quantitative Model Checking", Proc. of ISOLA 2004
- Input is the same as before: a KS $\mathcal{S}$, an LTL formula φ, $0 < \delta, \varepsilon < 1$
  - again, let's say we have $B = B_{\neg\varphi} \times B_{\mathcal{S}}$
- Output is the same: PASS or (FAIL, counterexample)
- FAIL is FAIL as before
- Much easier interpretation for PASS: as we will see, with confidence $1 - \delta$ we have a bound ε on the probability of $\mathcal{S} \not\models \varphi$

LTL Quantitative Model Checking

- Let Z be a random variable with values in $[0, 1]$
  - thus, Z is generally *not* a Bernoulli variable
  - but Bernoulli variables are a special case, so the methodology discussed below can be applied
- Recall that the *mean* of Z is $\mu_Z = \mathbb{E}[Z] = \sum_{\omega \in \Omega} p(\omega) Z(\omega) \in [0, 1]$
  - recall that, if Z is a Bernuolli variable, $\mu_Z = p_Z$
- The purpose here is exactly to compute μ_Z
- The exact value cannot be directly computed, so let us say we output $\tilde{\mu}_Z$ instead
- The methodology proposed here ensures that $\mathbb{P}(\mu_Z(1-\varepsilon) \leq \tilde{\mu}_Z \leq \mu_Z(1+\varepsilon)) \geq 1 - \delta$
  - so again, ε is a tolerance and δ is a confidence on the result
  - typically, they should be close to zero
  - often, this is called a (ε, δ)-approximation

OAA: Optimal Approximation Algorithm

- Dagum, Karp, Luby, Ross: "An Optimal Algorithm for Monte Carlo Estimation". *SIAM Journal on Computing, 29(5):1484–1496*, 2000.
- We have Z as a random variable in $[0, 1]$: how do we compute an (ε, δ)-approximation $\tilde{\mu}_Z$ of μ_Z?
- Idea: perform N independent trials of Z, collect results $Z_1, \ldots, Z_n$ and then output $\tilde{\mu}_Z = \frac{\sum_{i=1}^{N} Z_i}{N}$
- Straightforward problem: how to choose N, so as we have an (ε, δ)-approximation?

OAA: Optimal Approximation Algorithm

- We may employ an algorithm which *dynamically adjusts* the value of N on the basis of the results obtained so far
- In doing so, we use an auxiliary function SRA (Stopping Rule Algorithm)
- We also suppose to have a procedure $\mathcal{P}_Z$ which performs an experiment on Z and returns the corresponding value in $[0, 1]$
  - of course, different calls to $\mathcal{P}_Z$ will return different values
- Big limitation: $\mu_Z > 0$, or SRA does not terminate

OAA: Optimal Approximation Algorithm

```
OAA(procedure_for_Z P_Z, double ε, double δ) {
```

$\hat{\mu}_Z = \mathtt{SRA}(\mathcal{P}_Z, \min\{\frac{1}{2}, \sqrt{\varepsilon}\}, \frac{\delta}{3})$;

$\Upsilon = 2(1+\sqrt{\varepsilon})(1+2\sqrt{\varepsilon})\left(1+\frac{\log(3)-\log(2)}{\log(2)-\log(\delta)}\right)\frac{4(e-2)(\log(2)-\log(\delta))}{\varepsilon^2}$;

$N = \frac{\varepsilon\Upsilon}{\hat{\mu}_Z}$;

$S = \frac{1}{2}\sum_{i=1}^{N}(\mathcal{P}_Z() - \mathcal{P}_Z())^2$;

$\rho_Z = \max\left\{\frac{S}{N}, \varepsilon\hat{\mu}_Z\right\}$;

$N = \frac{\rho_Z\Upsilon}{\hat{\mu}_Z^2}$;

$\tilde{\mu}_Z = \frac{1}{N}\sum_{i=1}^{N}\mathcal{P}_Z()$;

return $\tilde{\mu}_Z$;

```
}
```

SRA: Stopping Rule Algorithm

```
SRA(procedure_for_Z P_Z, double ε, double δ) {
   Υ = 1 + (1 + ε) 4(e−2)(log(2)−log(δ)) / ε² ;
   N = 1;
   S = 0;
   while (S ≤ Υ) {
      N = N + 1;
      S = S + P();
   }
   return μ̂_Z = S/N;
}
```

LTL Quantitative Model Checking

- Leveraging on OAA, we use the almost same framework used for Monte-Carlo Model Checking
  - Bernoulli variable Z s.t. $Z = 1$ iff, making a random walk, you detect an non-accepting lasso
  - note that we reversed the previous definition: we will be back on this
  - Z is a special case of the random variables of OAA, so we may apply OAA to Z
  - also the probability space $(2^L, \mathbb{P})$ is the same
- The subroutine `SampleLasso` is the same as above

LTL Quantitative Model Checking

- OAA* is a modified version of OAA: as soon as $Z = 0$ for some trial, exit with probability 0
  - i.e., if a counterexample has been found
- Thus, OAA* returns either 0 (in the previous case) or 1 (otherwise)

```
QMC(KS_BA 𝒮𝒜, double ε, double δ) {
  p̃_Z = OAA*(SampleLasso(𝒮𝒜), ε, δ);
  if (p̃_Z == 0) {
    σ = extract the accepting lasso from the
        last trial;
    return (FAIL, σ);
  }
  else
    return PASS;
}
```

LTL Quantitative Model Checking

- Why $Z = 1$ if we find a "good" lasso?
  - instead of $Z = 1$ if we find a counterexample, as it was for MC^2?
- Recall that OAA only works if $p_Z = \mu_Z > 0$, otherwise SRA does not terminate
- With the current definition, $p_Z > 0$ means "there is at least a good lasso"
  - with the MC^2 definition, $p_Z > 0$ means "there is at least a counterexample": could easily be false!
- Even if "there is at least a good lasso" is false, QMC terminates as OAA* immediately exit after the first trial...

LTL Quantitative Model Checking

- Recall that $\mathbb{P}(\mu_Z(1-\varepsilon) \leq \tilde{\mu}_Z \leq \mu_Z(1+\varepsilon)) \geq 1-\delta$
- If $Z = 1$ for all trials, $\tilde{\mu}_Z = 1$
- Thus, $1 = \tilde{\mu}_Z \geq \mu_Z(1-\varepsilon)$ always holds
- What remains is $\mathbb{P}(\mu_Z(1+\varepsilon) \geq 1) \geq 1-\delta$
  - better: $\mathbb{P}(\mu_Z \geq \frac{1}{1+\varepsilon}) \geq 1-\delta$
- If we recall that $\mu_Z = p_Z = 1 - q_Z$ we have that $\mathbb{P}(q_Z \leq 1 - \frac{1}{1+\varepsilon}) = \mathbb{P}(q_Z \leq \frac{\varepsilon}{1+\varepsilon}) \geq 1-\delta$
- Actually, for small ε, $\frac{\varepsilon}{1+\varepsilon} \approx \varepsilon$, thus we are saying that $\mathbb{P}(q_Z \leq \varepsilon) \geq 1-\delta$!
  - q_Z is the probability that, making a random walk, we find a counterexample
  - much better than the obscure assumption of MC^2

LTL Quantitative Model Checking

- QMC seems extremely better than MC^2
- So why MC^2 has been published as an improvement of QMC one year later?
- Because the OAA methodology requires much more steps
- For MC^2, the worst-case number of trials is $M = \frac{\log(\delta)}{\log(1-\varepsilon)}$
- For QMC, we can show that worst-case number of trials is bound by $N = O(4\frac{\log(2)-\log(\delta)}{\varepsilon})$
  - recall that $\log(\delta) < 0$
  - $N > 5M$
  - e.g., $\delta = 0.1, \varepsilon = 0.0018 \rightarrow M = 1257$ for MC^2
  - but $N = 1257$ with $\delta = \varepsilon = 0.1$ for QMC
- RAM space is $O(D)$ for both

Unfair Dining Philosophers

	DDFS		QMC			
ph	time	entr	time	mxl	cxl	N
4	0.02	31	0.08	10	10	3
8	1.62	511	0.20	25	8	7
12	3:13	8191	0.25	37	11	11
16	>20:0:0	–	0.57	55	8	18
20	–	oom	3.16	484	9	20
30	–	oom	35.4	1478	11	100
40	–	oom	11:06	13486	10	209

	DDFS		QMC			
ph	time	entr	time	mxl	cxl	N
4	0.17	29	0.02	8	8	2
8	0.71	77	0.01	7	7	1
12	1:08	125	0.02	9	9	1
16	7:47:0	173	0.11	18	18	1
20	–	oom	0.06	14	14	1
30	–	oom	1.12	223	223	1
40	–	oom	1.23	218	218	1

Fair Dining Philosophers

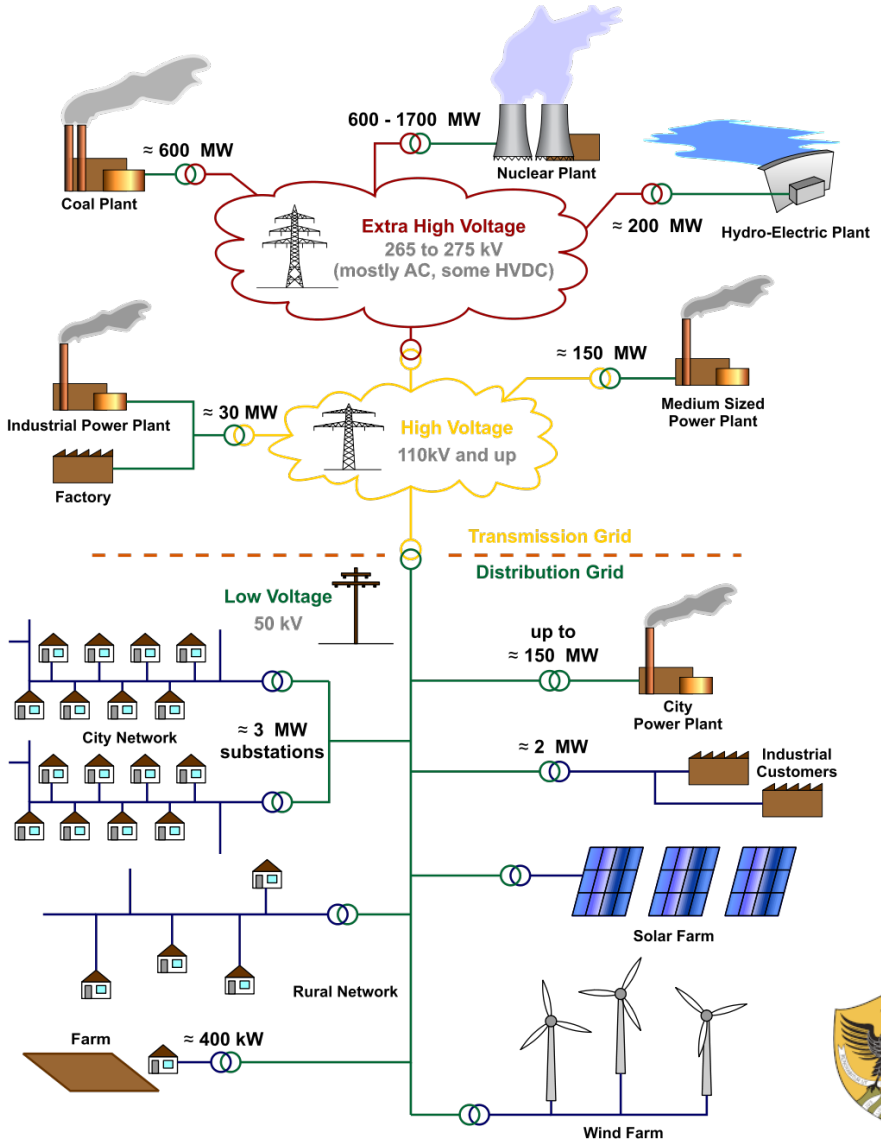
	DDFS		QMC		
phi	time	entries	time	mxl	avl
4	0:01	178	0:20	49	21
6	0:03	1772	0:45	116	42
8	0:58	18244	2:42	365	99
10	16:44	192476	7:20	720	234
12	–	oom	21:20	1665	564
14	–	oom	1:09:52	2994	1442
16	–	oom	3:03:40	7358	3144
18	–	oom	6:41:30	13426	5896
20	–	oom	19:02:00	34158	14923

	DDFS		QMC		
phi	time	entries	time	mxl	avl
4	0:01	538	0:20	50	21
6	0:17	9106	0:46	123	42
8	7:56	161764	2:17	276	97
10	–	oom	7:37	760	240
12	–	oom	21:34	1682	570
14	–	oom	1:09:45	3001	1363
16	–	oom	2:50:50	6124	2983
18	–	oom	8:24:10	17962	7390
20	–	oom	22:59:10	44559	17949

Combining QMC and MC^2: a Case Study

- Mancini, Mari, Melatti, Salvo, Tronci, Gruber, Hayes, Prodanovic, and Elmegaard. "Parallel Statistical Model Checking for Safety Verification in Smart Grids." In Proc. SmartGridComm 2018.
- EDN: Electric Distribution Network, also called "grid"
  - brings to residential houses, commercial buildings and industries the electricity they need
  - till some decades ago, simply based on demands
- Smart grid: usage of computational services to improve electricity distribution
  - e.g.: electricity usage is measured and then rendered in a web app

Combining QMC and MC^2: a Case Study



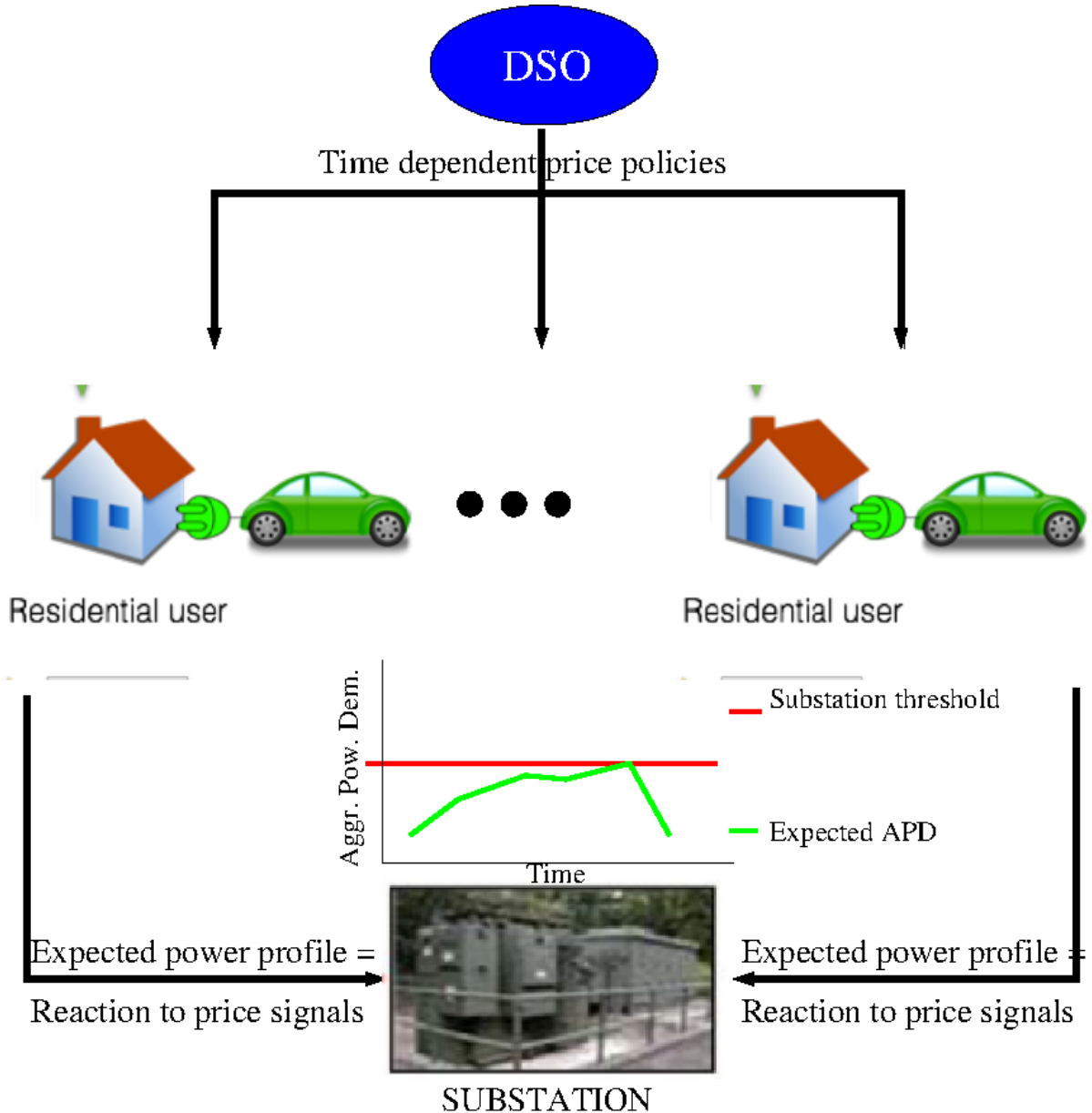
UNIVERSITÀ DEGLI STUDI DELL'AQUILA

DISIM
Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Combining QMC and MC^2: a Case Study

- Distribution System Operators (DSOs) and energy retailers compute price tariffs for residential users
- Expected Power Profiles (EPPs): how residential users will respond to price tariffs
- DSOs compute price tariffs so that EPPs do not threat substations safety
  - in each t, Aggregated Power Demand (APD) must be below the substation safety power threshold (e.g., 400 kW)
  - DSOs main goal is to achieve *peak shaving*

Problem at a Glance



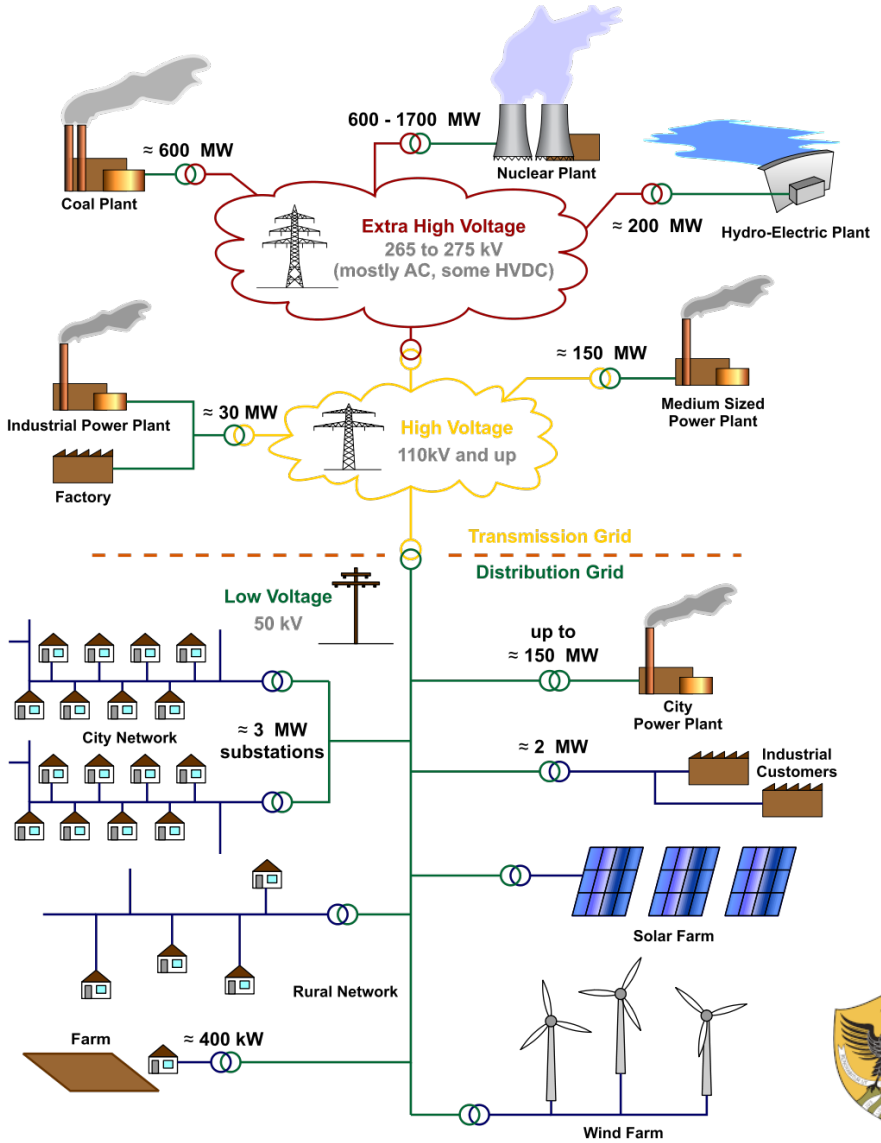
Autonomous Demand Response

- Residential users may or may not follow their corresponding Expected Power Profiles (EPPs)
  - there may be automatic tools to enforce EPPs
  - implemented on small devices on users premises
  - still, there is no guarantee, due to unexpected needs, bad forecasts, limited computational resources, etc.

Problem

Given that users may deviate from EPPs with a given probability distribution, what is the resulting probability distribution for the aggregated power demand (APD)?

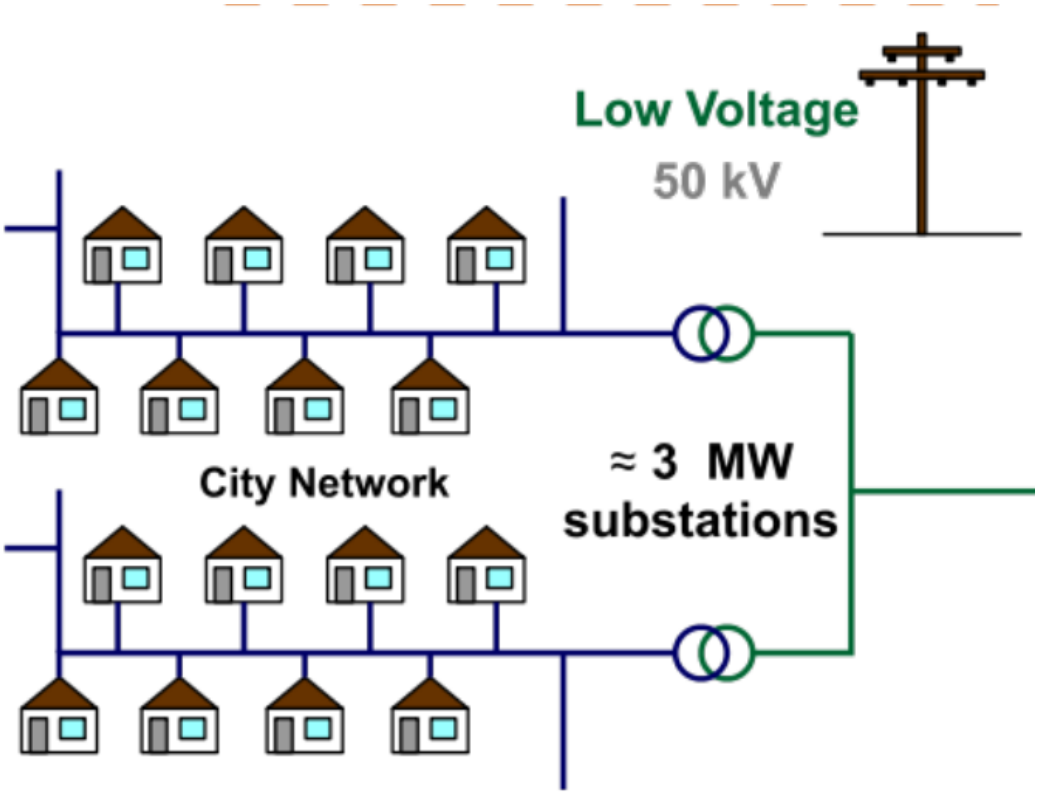
Combining QMC and MC^2: a Case Study



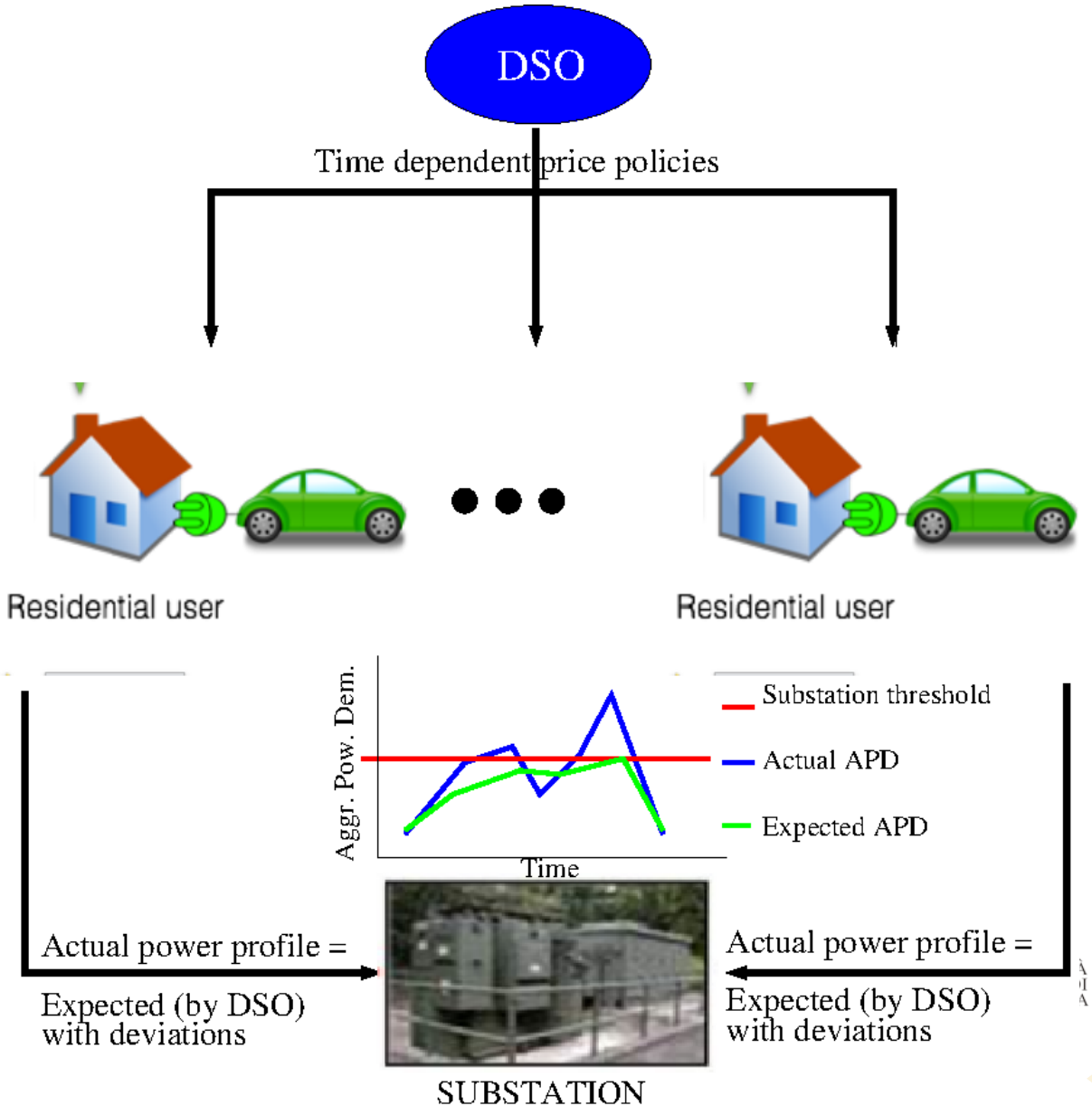
UNIVERSITÀ DEGLI STUDI DELL'AQUILA

DISIM
Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Combining QMC and MC^2: a Case Study



Problem at a Glance



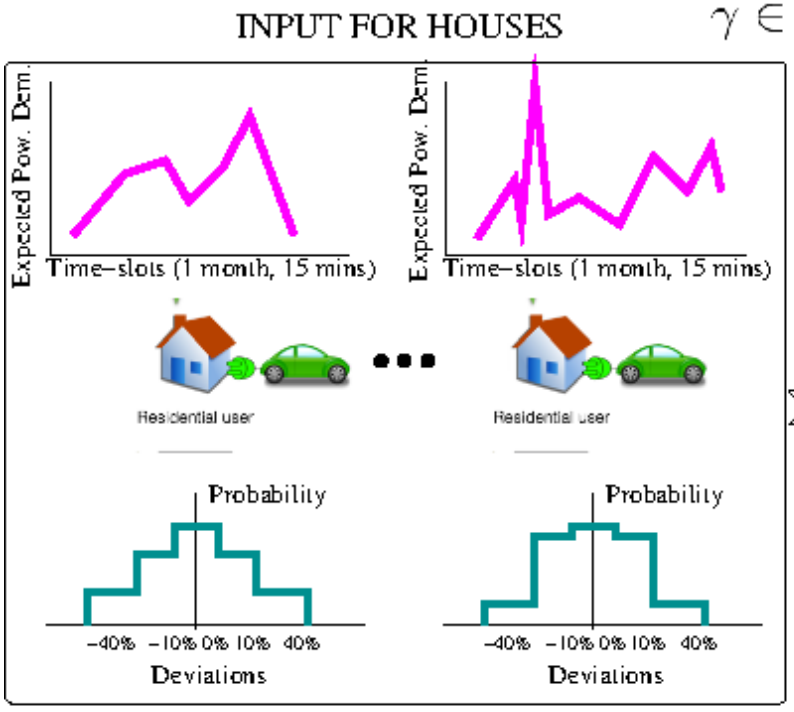
APD-Analyser

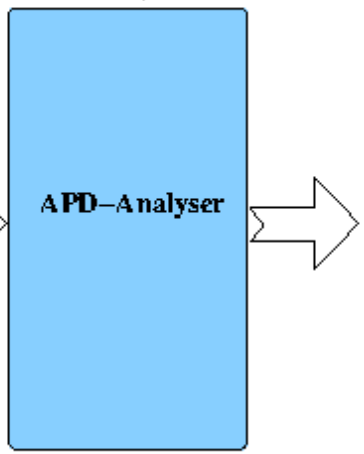
- We present the APD-Analyser tool
  - APD: Aggregated Power Demand

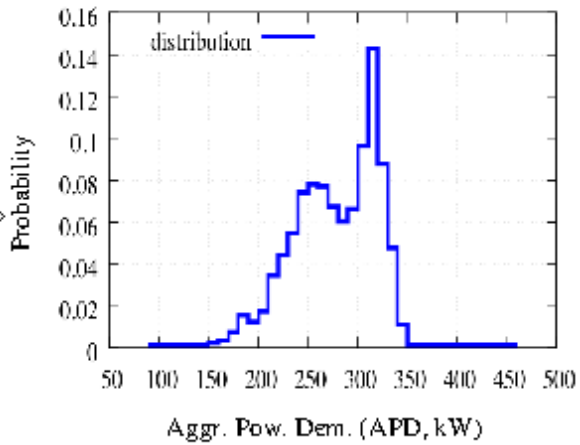
- Main goal: compute the probability distribution for the APD
  - given probability distributions on each residential user
    Expected Power Profile (EPP)

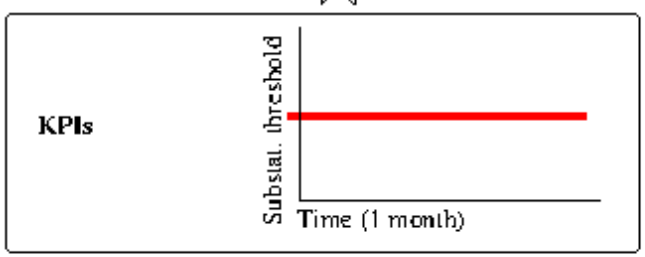
APD-Analyser: Input and Output

$0 < \varepsilon < 1$: tolerance
$0 < \delta < 1$: numerical accuracy
$\gamma \in \mathbb{R}^+$: discretisation for APD









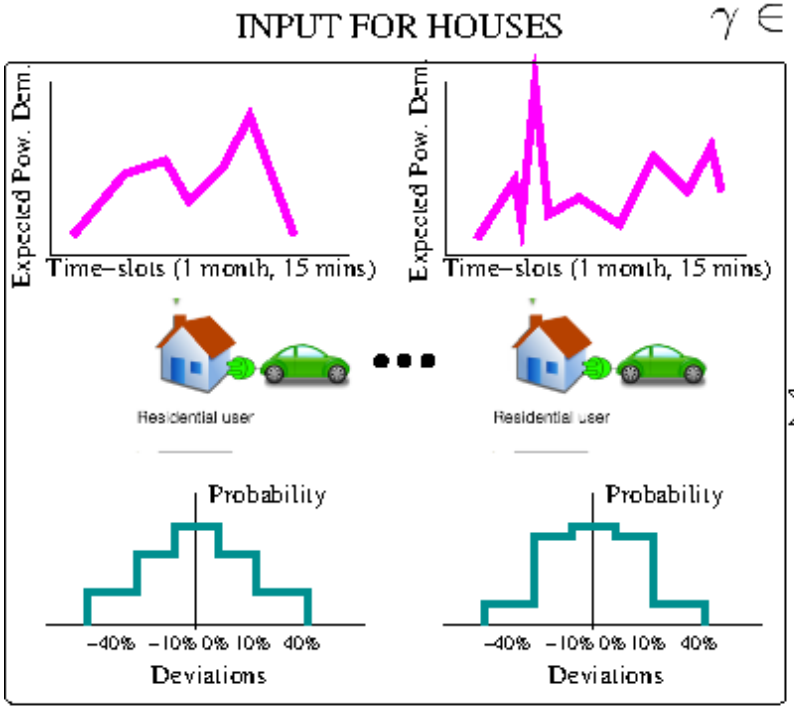
INPUT FOR SUBSTATION

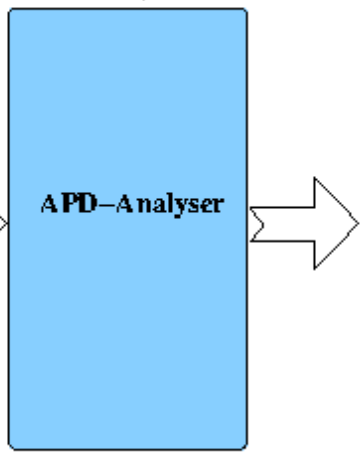
APD-Analyser: Input

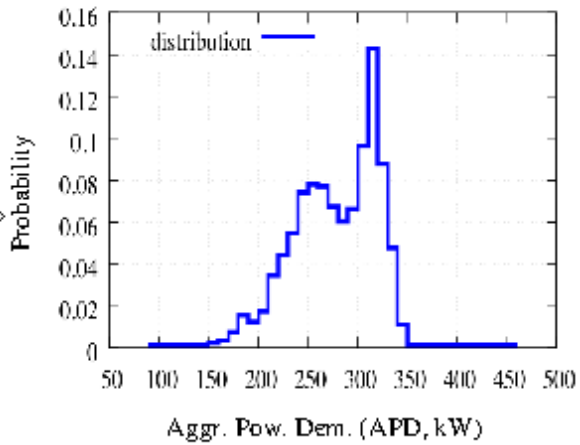
- Set of residential users U connected to the same substation
- Set of time-slots T (e.g., one month with 15 minutes step)
- Expected Power Profiles (EPP)
  - one for each user $u \in U$: for each time-slot $t \in T$, the expected power demand of u in t
  - $p_u : T \rightarrow \mathbb{R}$
- A probabilistic model for users deviations from EPPs
  - a real function $dev_u : \mathbb{R} \rightarrow [0, 1]$, for each user $u \in U$
  - $\int_{-\infty}^{+\infty} dev_u(x)dx = 1$
  - $\int_a^b dev_u(x)dx =$ probability that actual power demand of u in any time-slot $t \in T$ is in $[(1 + a)p_u(t), (1 + b)p_u(t)]$
  - e.g.: $\int_{-0.02}^{0.02} dev_u(x)dx =$ probability that actual power demand of u in any time-slot $t \in T$ deviates at most by 2% from EPP of u

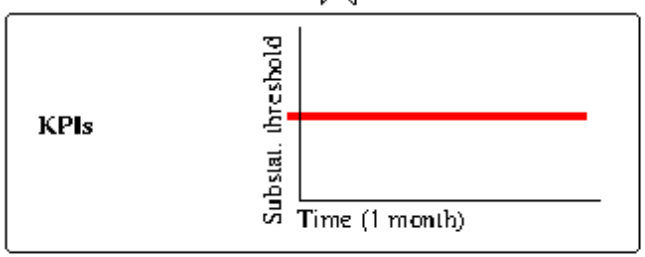
APD-Analyser: Input and Output

$0 < \varepsilon < 1$: tolerance
$0 < \delta < 1$: numerical accuracy
$\gamma \in \mathbb{R}^+$: discretisation for APD









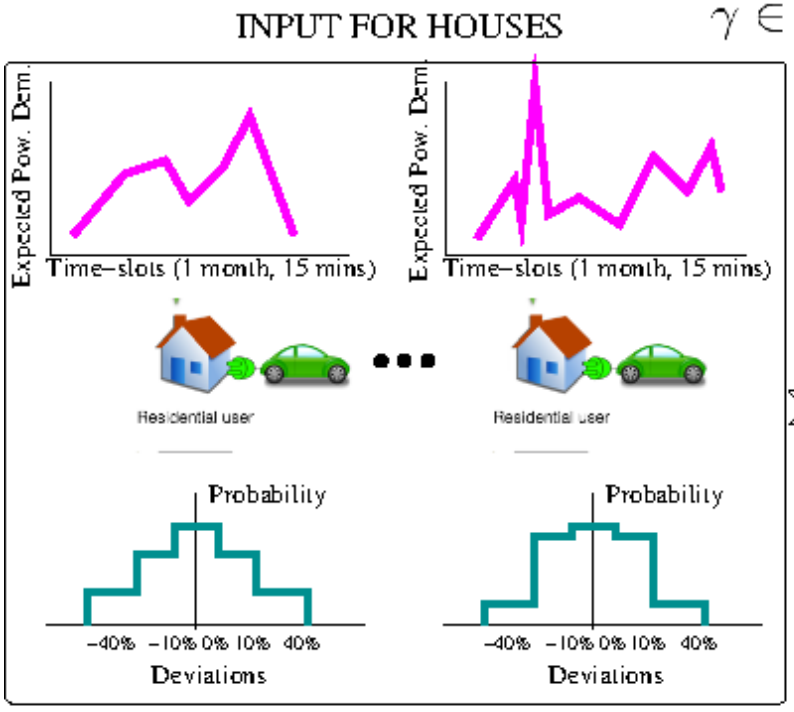
INPUT FOR SUBSTATION

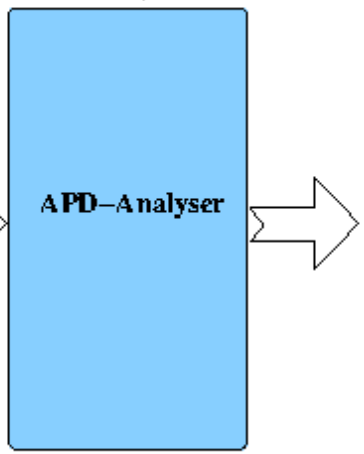
APD-Analyser: Input

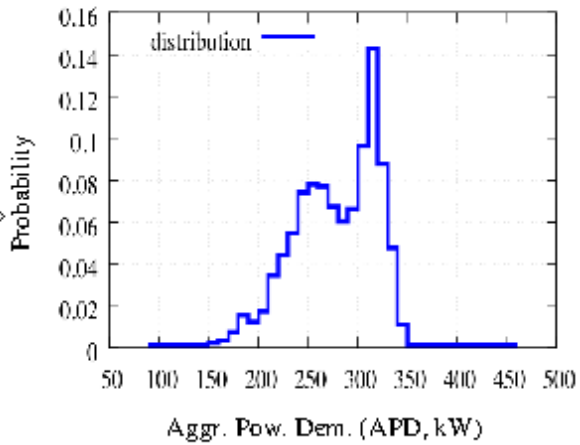
- Substation safety requirements
  - $p_s : T \to \mathbb{R}$
  - for each $t \in T$, DSO wants the APD to be below $p_s(t)$
  - that is, $\forall t \in T \to \sum_{u \in U}[(1 + \text{deviation}_u)p_u(t)] \leq p_s(t)$
- Key Performance Indicators (KPIs)
  - e.g., probability distribution that $p_s(t)$ is exceeded in any $t \in T$
- Parameters
  - $0 < \delta, \varepsilon < 1$: as for output probability distributions, the values must be correct up to tolerance ε with statistical confidence δ
    - $\Pr[(1 - \varepsilon)\mu \leq \tilde{\mu} \leq (1 + \varepsilon)\mu] \geq 1 - \delta$
    - μ: (unknown) correct value, $\tilde{\mu}$: computed value
  - $\gamma \in \mathbb{R}^+$: discretisation step for output probability distribution

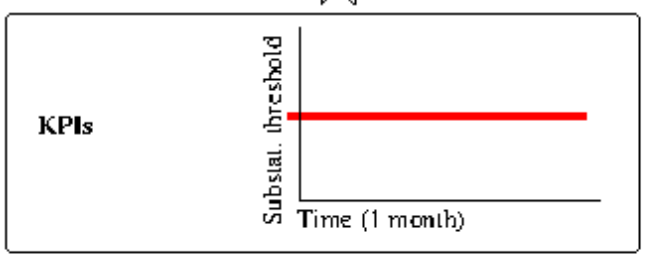
APD-Analyser: Input and Output

$0 < \varepsilon < 1$: tolerance
$0 < \delta < 1$: numerical accuracy
$\gamma \in \mathbb{R}^+$: discretisation for APD









INPUT FOR SUBSTATION

APD-Analyser: Output

- Probability distribution for APD resulting from EPPs disturbed with given probabilistic disturbance model
  - easy to evaluate KPIs once such distribution is computed
  - formally: $\Psi(W)$ is the probability that APD takes a value in interval W in any time-slot $t \in T$
- Exactly computing Ψ is infeasible, thus we compute $\tilde{\Psi}$ as a (ε, δ) approximation of a γ-discretisation of the APD
- For each γ-discretised value $w = \mathrm{APD}_{min} + k\gamma$, we compute $\tilde{\Psi}(w)$ s.t., with confidence at least $1 - \delta$:
  - if $\tilde{\Psi}(w) = \perp \notin [0, 1]$ then $\Psi([w, w + \gamma)) < \varepsilon$
  - otherwise, $\tilde{\Psi}([w, w + \gamma))$ is within $(1 \pm \varepsilon)\Psi(w)$

APD-Analyser: Algorithm

- *Monte-Carlo model checking*
  - goal: estimate the mean of a 0/1 random variable Z_w
  - $Z_w = 1$ iff, taken at random a $t \in T$, the value of the APD is in $[w, w + \gamma)$, when EPPs are perturbed using deviations model dev_u
  - then, the mean is exactly our $\tilde{\Psi}(w)$
- Method: perform N independent experiments (samples) for Z_w, and then the mean of Z_w is $\frac{\sum_{i=1}^{N} \hat{Z}_i}{N} \in [0, 1]$
  - Optimal Approximation Algorithm (OAA) by Dagum & al. (2000) + Monte-Carlo Model Checking (MCMC) by Grosu & Smolka (2005)
  - *sequential analysis*: use outcomes of previous experiments to compute N
  - the value of N is automatically adjusted, at run-time, while performing the samples
  - so that the desired tolerance ε is achieved with desired accuracy δ

DISIM
Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Optimal Approximation Algorithm (OAA)

1. Phase 1
   1. Perform $N_1 = f_1(\varepsilon, \delta)$ experiments $\hat{Z}_{1,1}, \ldots, \hat{Z}_{1,N_1}$
   2. Compute mean of successful experiments $\hat{\mu}_Z = \frac{1}{N_1} \sum_{i=1}^{N_1} \hat{Z}_{1,i}$
2. Phase 2
   1. Perform $2N_2 = 2f_2(\varepsilon, \delta, \hat{\mu}_Z)$ experiments $\hat{Z}_{2,0}, \ldots, \hat{Z}_{2,2N_2-1}$
   2. Compute $S = \frac{1}{N_2} \sum_{i=0}^{N_2-1} \frac{|\hat{Z}_{2,2i} - \hat{Z}_{2,2i+1}|}{2}$
3. Phase 3
   1. Perform $N_3 = f_3(\varepsilon, \delta, \hat{\mu}_Z, S, N_2)$ experiments $\hat{Z}_{3,1}, \ldots, \hat{Z}_{3,N_3}$
   2. Return mean of successful experiments $\tilde{\mu}_Z = \frac{1}{N_3} \sum_{i=1}^{N_3} \hat{Z}_{1,i}$
      - It holds that $\Pr[(1-\varepsilon)\mu_Z \leq \tilde{\mu}_Z \leq (1+\varepsilon)\mu_Z] \geq 1 - \delta$

OAA + Monte-Carlo Model Checking (MCMC)

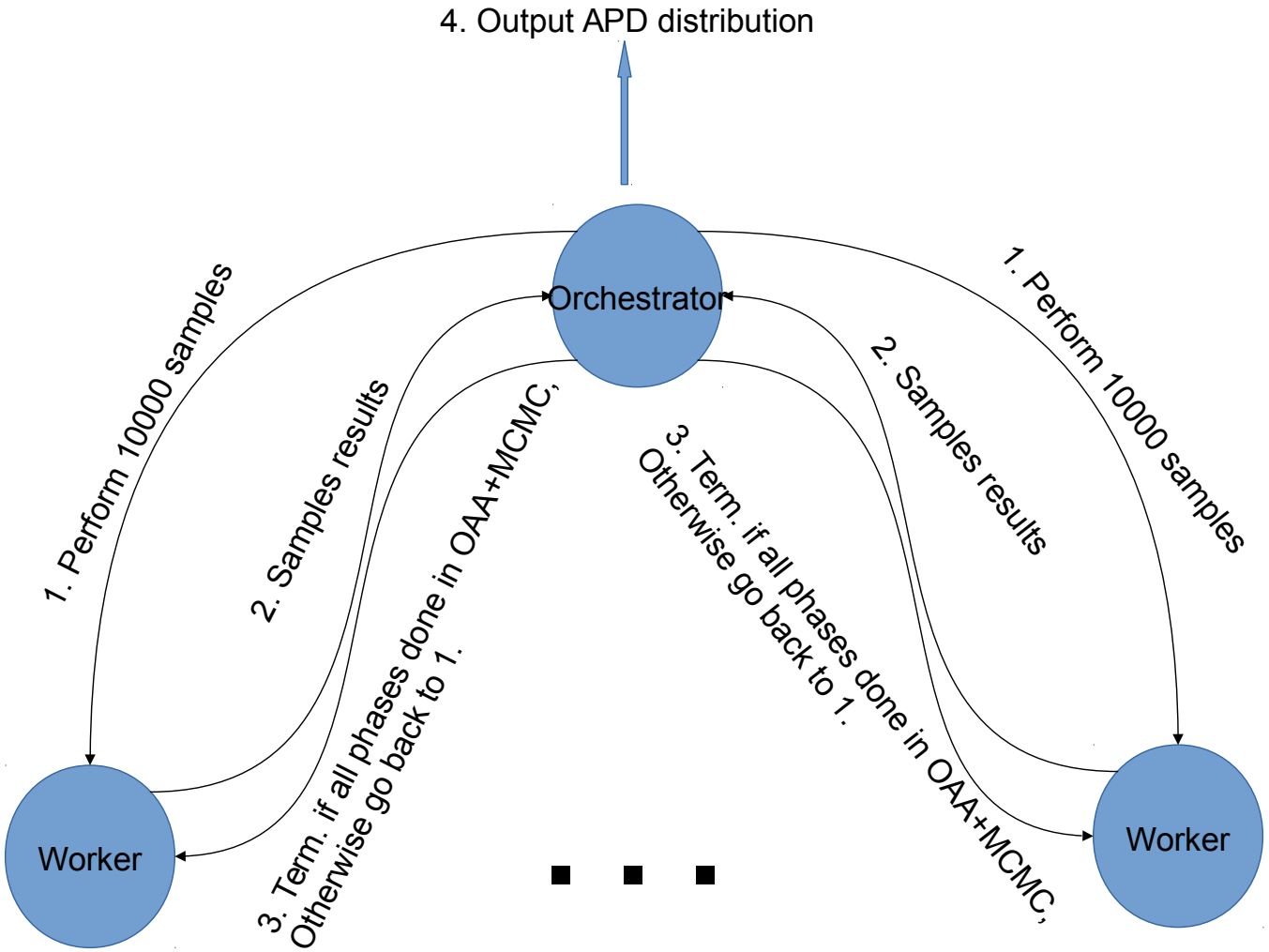
- Correct phase 1 using statistical hypothesis testing
- If $\sum_{i=1}^{M} \hat{Z}_{1,i} = 0$ for $M = f_4(\varepsilon, \delta) = \left\lceil \frac{ln(\delta)}{ln(1-\varepsilon)} \right\rceil$, terminate the computation
- Return $\tilde{\mu}_Z = 0$
- It holds that $\Pr[\mu_Z < \varepsilon] \geq 1 - \delta$

APD-Analyser: HPC Algorithm

- $N = N_1 + N_2 + N_3$ can be prohibitively high
  - easily order of 10^9 in our experiments
  - OAA+MCMC to be run for each different value of w
  - if performed with a sequential algorithm, order of 1 month for the computation time
- We re-engineer the OAA to be run on a HPC infrastructure, i.e., a cluster (distributed memory)
  - main obstacle: value of N depends on samples outcomes! To be computed at run-time
- One *orchestrator* node instructs *worker* nodes to perform given number of samples
  - worker nodes perform samples in parallel and send results to the orchestrator
  - the orchestrator keeps track of phases of each worker and of different values of w
- As a result, less than 2 hours of computation with 89 workers

APD-Analyser: HPC Implementation Sketch

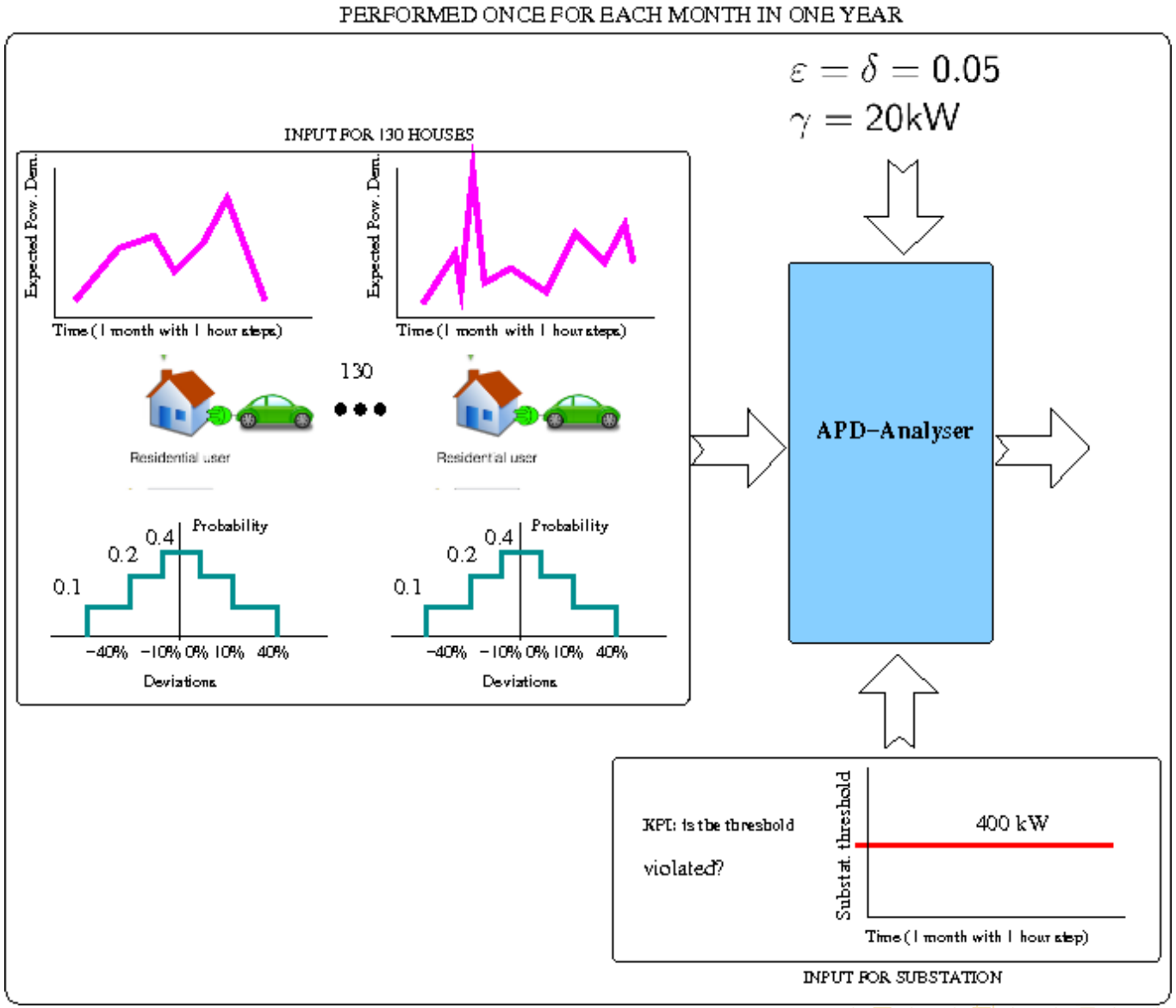
Different workers may be in different phases and different w



Experimental Evaluation: Case Study

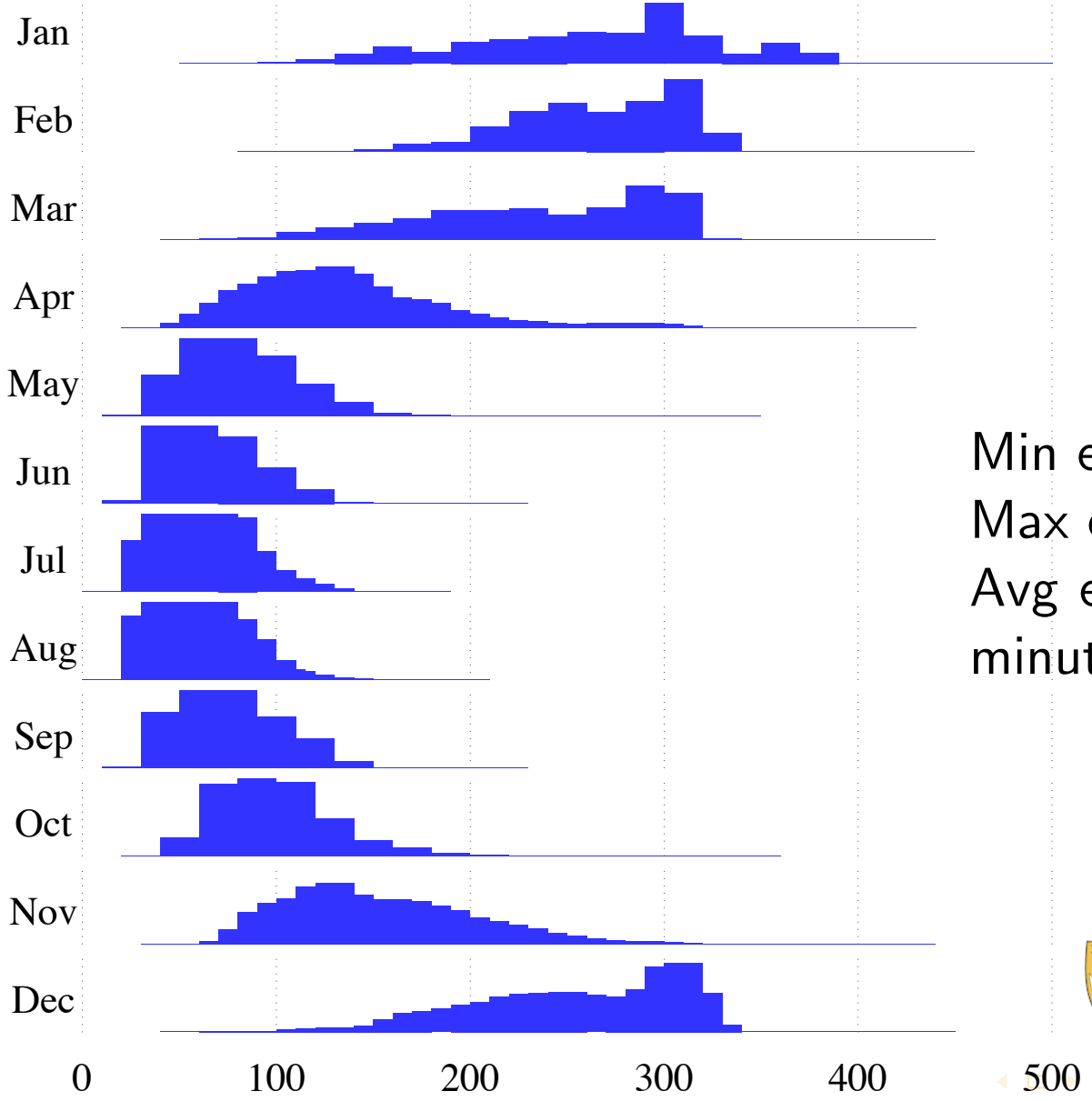
- 130 houses in Denmark, all connected to the same substation
- EPPs computed by using methodologies from the literature
  - starting point: historical data collected on those houses for one year (SmartHG FP7 project)
  - computed as shifts within given flexibilities so as to collaboratively respond to price policies
- Very liberal deviation model: up to $\pm 40\%$ variations with 10% probability, up to $\pm 20\%$ variations with 20% probability
- We want to compute the APD for each month of the year
  - by using time-slots of 1 day, we have $5^{30 \times 130}$ overall number of deviations

Experimental Evaluation: Case Study



DISIM
Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Experimental Results



Min exec time: 4782 secs
Max exec time: 6448 secs
Avg exec time: 1 hour, 28 minutes and 7 seconds

UNIVERSITÀ DEGLI STUDI DELL'AQUILA

DISIM
Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Experimental Results: HPC Scalability

# workers	samples/sec	speedup	efficiency
1	*5924.89*	1×	100%
20	79275.028	13.38×	66.90%
40	162578.98	27.44×	68.60%
60	257791.96	43.51×	72.52%
80	335823.24	56.68×	70.85%

$$\text{speedup} = \frac{s_k}{s_1}, \text{ efficiency} = \frac{s_k}{k s_1}$$

Conclusions

- We presented the HPC-based tool APD-Analyser
- Main purpose: support DSOs in analysing effects of price policies on aggregated power demand (APD) at substation level
  - especially for highly-fluctuating and individualised price policies
- From expected power profiles disturbed by probabilistic deviations (input) to probability distribution for APD (output)
- As a result, APD-Analyser enables safety assessment of price policies in highly dynamic ADR schemas

Statistical Model Checking for Everything

- Zuliani, Platzer, Clarke: "Bayesian Statistical Model Checking with Application to Stateflow/Simulink Verification", Formal Methods in System Design vol. 43, 2013
- In the works above, it was necessary to have some simple language defining the system
  - e.g., Promela of SPIN, though they use a different language
  - needed to perform the Cartesian product of the property and the system itself
  - and also to actually make a random walk of the system
  - actually, such a limitation is not difficult to overcome, but it is presented in this way
  - especially ok for systems already expressed in the language, but which went out of resources
- Here, we directly use simulators
  - Simulink, but conceptually also Modelica

Some Background

- As before, we have to define our probability space; this time is not easy
- Given a set X, a *σ-algebra* on X is $\mathcal{Y} \subseteq 2^X$ s.t. $\mathcal{Y}$ is closed for complements, countable unions and countable intersections, i.e.:
  - $\forall Y \in \mathcal{Y}.\ \bar{Y} \in \mathcal{Y}$, with $\bar{Y} = \{x \in X \mid x \notin Y\}$ being the complement of Y
  - $\forall I \subseteq \mathbb{N}$ s.t. $Y_i \in \mathcal{Y}$:
    - $\forall i, j \in I.\ Y_i \cup Y_j \in \mathcal{Y}$
    - $\forall i, j \in I.\ Y_i \cap Y_j \in \mathcal{Y}$
- Example: $\mathcal{Y}_1 = 2^X$ and $\mathcal{Y}_2 = \{\varnothing, X\}$ are always σ-algebras
- Example: for $X = \{a, b\}$, $\mathcal{Y} = \{\{a\}, \{b\}, \{a, b\}\}$ is not a σ-algebra since $\{a\} \cap \{b\} = \varnothing \notin \mathcal{Y}$

Some Background

- If $X \subseteq \mathbb{R}^n$, the *Borel set* on X, denoted by $\mathcal{B}(X)$, is the smallest σ-algebra of X which contains all open sets of X
  - recall that a set $A \subseteq \mathbb{R}^n$ is open iff, for all $a \in A$, there exists a n-dimensional ball (border excluded) centered in a which is contained in A
  - that is, $\exists \varepsilon > 0 : \ \forall x \in \mathbb{R}^n. \ |a - x| < \varepsilon \Rightarrow x \in A$
- The pair $(X, \mathcal{B}(X))$ is called *measureable space*
- Thus, $\mathcal{B}(X)$ retains all open sets already in X and ensures that intersection, union and complementation are in $\mathcal{B}(X)$

Some Background

- A *stochastic kernel* on $(X, \mathcal{B}(X))$ is a function $K : X \times \mathcal{B}(X) \to [0, 1]$ s.t.:
  - for all $x \in X$, the function $K_x : \mathcal{B}(X) \to [0, 1]$ defined by $K_x(B) = K(x, B)$ is a probability measure on $\mathcal{B}$
    - that is, the three Kolmogorov axioms are true
    - note that K_x actually takes subsets
  - for all $B \in \mathcal{B}(X)$, the function $K_{\mathcal{B}} : X \to [0, 1]$ is a measurable function on X
    - we are less interested on this point

Some Background

- Since each state is a point $x \in X \subseteq \mathbb{R}^n$, execution traces are sequences $\sigma \in X^\omega$
  - for finite (terminated) runs, we may add a loop on the last state (*stuttering*)
- We want to define probabilities on traces, thus $\Omega = (\mathbb{R}^n)^\omega$
- Usually, we define the probability on $(\Omega, 2^\Omega)$
- For these types of Ω, we are happy with something contained in 2^Ω, namely $\mathcal{F}$ as the cylindric σ-algebra built on Ω
  - essentially, such sequences behave "well"

Some Background

- We suppose to have a stochastic kernel K defined on $(\Omega, \mathcal{F})$
- Together with an initial state $x \in X$, this defines a probability on $(\Omega, \mathcal{F})$ of a Markov process $\{X_t \mid t \in \mathbb{N}\}$
  - $\mathbb{P}(X_1 \in B) = 1$ if $x \in B$ and 0 otherwise;
  - $\mathbb{P}(X_{i+1} \in B) = K(x_i, B)$
    - where $B \subseteq \Omega$ is any "event", i.e., a subset of executions in Ω
    - K defines the outgoing transitions probability
    - meaning of $K(x, B)$: probability of arriving in B starting from an initial state x
- Thus, if we are able to define a K, we have a probability space for our SMC methodology

Discrete Time Hybrid Automaton

- Giving a precise semantics of Simulink (or Modelica) is difficult, but the following definition is quite close
- A *Discrete Time Hybrid Automaton* (DTHA) is defined as $\mathcal{D} = \langle Q, E, n, q_0, x_0, \Phi, J \rangle$ where:
  - n is the dimension of the state space, which is understood to be $\mathbb{R}^n$
  - (Q, E) is a directed graph
    - Q is a set of *locations*, E is a set of *control switches* or *modalities*
  - (q_0, x_0) is the *starting state*, $(q_0, x_0) \in Q \times \mathbb{R}^n$
  - $\Phi = \{\phi_q : \mathbb{R}^+ \times \mathbb{R}^n \to \mathbb{R}^n \mid q \in Q\}$
  - $J = \{j_e : \mathbb{R}^n \to \mathbb{R}^n \mid e \in E\}$

Discrete Time Hybrid Automaton Semantics

- The *transition relation* δ of a DTHA $\mathcal{D}$ defines when we go from a state in $Q \times \mathbb{R}^n$ to another
  - not simple as for Kripke structures, where one step is one step: here, also time passing is important
- 2 underling ideas:
  - time only passes within locations, handled by Φ
  - jumps within locations happen in time 0, defined by E with conditions given by J
  - either the time pass within a location, or a jump between locations is performed
- $\delta \in Q \times \mathbb{R}^n \times (\mathbb{R}^+ \dot{\cup} E) \times Q \times \mathbb{R}^n$

Discrete Time Hybrid Automaton Semantics

- $\delta \in Q \times \mathbb{R}^n \times (\mathbb{R}^+ \dot{\cup} E) \times Q \times \mathbb{R}^n$
- $(q, x, t, q, x') \in \delta \equiv (q, x) \rightarrow_t (q, x')$ iff $x' = \phi_q(t, x)$
  - note that q does not change
- $(q, x, e, q', x') \in \delta \equiv (q, x) \rightarrow_e (q', x')$ iff $x' = j_e(x) \wedge e = (q, q')$
  - note that time does not pass
- $\Delta : Q \times \mathbb{R}^n \rightarrow (\mathbb{R}^+ \dot{\cup} E)$ is the *simulation function*
  - decides if, in a given state, a location jump or a time pass has to be performed
  - if time passes, decides how much
  - unified notation $(q, x) \rightarrow_{\Delta(q,x)} (q', x')$

Discrete Time Hybrid Automaton Semantics

- δ may be non-deterministic
  - in a given state (q, x), both some j_e and ϕ_q could be enabled
  - even if only ϕ_q is enabled, many values for t may apply
- Δ is deterministic; in Simulink:
  - if both a discrete and a continuous transition can be taken, take the discrete one
  - if continuous, stay for the maximum time allowed before a location change
  - an ordering on outgoing edge is always available, so the first one is selected when multiple edges are present

Discrete Time Hybrid Automaton Semantics

- A *trace* is a sequence $\sigma = (s_0, t_0), \ldots, (s_i, t_i), \ldots$ s.t.
  - $s_0 = (q_0, x_0)$
  - $\forall i \geq 0.\ s_i \in Q \times \mathbb{R}^n,\ t_i \in \mathbb{R}^+$
  - $\forall i \geq 0.\ s_i \rightarrow_{\Delta(s_i)} s_{i+1}$
  - $\forall i \geq 0.\ t_i = \Delta(s_i)$ if $\Delta(s_i) \in \mathbb{R}^+$
  - $\forall i \geq 0.\ t_i = 0$ if $\Delta(s_i) \in E$
- At step $\sigma_i = (s_i, t_i)$, the global time is $\sum_{j=0}^{i-1} t_j$
- For an infinite trace σ, $\sum_{j=0}^{\infty} t_j = \infty$
  - there must be finitely many location switches in finite time

Probabilistic Discrete Time Hybrid Automaton Semantics

- For a set X, let $D(X) = \{f \mid f \text{ is a probability density function on } X\}$
  - for $X = \{x_1, \ldots, x_n\}$, $f(t) = \sum_{i=1}^{n} p_i \delta(t - x_i)$, for any choice of $p_i \in [0, 1]$ s.t. $\sum_{i=1}^{n} p_i = 1$
  - here δ is the Dirac function, i.e. $\delta(0) = 1, \forall x \neq 0.\ \delta(x) = 0$
  - otherwise, for continuous X, $f(t)$ is s.t. $\int_a^b f(x)dx \in [0, 1]$ for any $[a, b] \subseteq X$ and $\int_X f(x)dx = 1$
- A *probabilistic transition function* Π for a DTHA $\mathcal{D}$ is a function $\Pi : Q \times \mathbb{R}^n \to D(\{0, 1\}) \times D(\mathbb{R}^+) \times D(E)$
  - since Π returns 3 values, we will denote its components by $\Pi(s) = \langle \Pi_a(s), \Pi_c(s), \Pi_d(s) \rangle$
  - the following must be true: $\forall (q, q') \in E, r \in Q, x \in \mathbb{R}^n.\ q \neq r \to \Pi_d(r, x)(q, q') = 0$
  - i.e., if an edge is taken, it must start from the current state

Probabilistic Discrete Time Hybrid Automaton Semantics

- Informally, a probabilistic transition function Π has the goal of definining a (possibly non-uniform) "random walk" on a DTHA
  - suppose we are in a state $s = (q, x)$
  - both a location change and a continuous move may be taken? choose at random with probability $\Pi_a(s)$
  - if a location change must take place, choose one at random with probability $\Pi_d(s)$
  - if time must pass, decide how much with probability $\Pi_c(s)$

Probabilistic Discrete Time Hybrid Automaton Semantics

- Thus, $K((q,x),B) = p_a \sum_{e \in E(B,q,x)} \Pi_d(q,x)(e) + (1 - p_a) \int_0^\infty \Pi_c(q,x)(t) I_B(q, \phi_q(t,x)) dt$
  - B is a Borel set over $Q \times \mathbb{R}^n$
  - $p_a = \Pi_a(q,x)(0)$
    - arbitrary choice, could also have been $p_a = \Pi_a(q,x)(1)$
  - $E \supseteq E(B,q,x) = \{(q,q') \in E \mid (q', j_{(q,q')}(x)) \in B\}$
    - to be well-defined, we must stay in the same Borel set
  - I_B is the indicator function of B, i.e. $I_B(q,x) = 1$ iff $(q,x) \in B$, and 0 otherwise
    - again, to be well-defined, we must stay in the same Borel set
- It may be shown that K is a stochastic kernel, so probability is well-defined over infinite traces

Probabilistic DTHA in Simulink/Stateflow

- n is the number of variables in a Simulink/Stateflow model
  - some of them may be discrete, but $\mathbb{R}^n$ is for sure a superclass
- Q corresponds to "states" of Stateflow and E are states transitions
- Simulink only perform deterministic transitions, so probability density function output by Π all consists in just one point being defined
- Differently from the Grosu & Smolka works, here we cannot provide a deterministic model and let the methodology turn it probabilistic
  - the user must define something probabilistic
  - typically done by introducing probabilistic blocks in the design
  - Uniform Random Number block

Bayes Theorem

- Conditional probability: $\mathbb{P}(A|B)$ is the probability of event A, under the assumption that event B already occured
  - by definition, $\mathbb{P}(A|B) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}$
- Which is the relationship between $\mathbb{P}(A|B)$ and $\mathbb{P}(B|A)$?
- The well-known Bayes Theorem states that $\mathbb{P}(A|B) = \frac{\mathbb{P}(B|A)\mathbb{P}(A)}{\mathbb{P}(B)}$
  - $\mathbb{P}(A_i|B) = \frac{\mathbb{P}(B|A_i)\mathbb{P}(A_i)}{\sum_{j=1}^{n} \mathbb{P}(B|A_j)\mathbb{P}(A_j)}$, if $\cup_{j=1}^{n} A_j = \Omega$
- Here we need a more refined version of the Bayes Theorem
- First of all, the conditional probability density function of a Bernoulli random variable X and a random variable U with values in $(0, 1)$ is $f(x_i|u) = u^{x_i}(1-u)^{1-x_i}$
  - then, $f(x_i = 1|u) = u$ and $f(x_i = 0|u) = 1 - u$

Bayes Theorem

- Our refined version of the Bayes Theorem states that $f(u|x_1 \dots x_n) = \frac{f(x_1 \dots x_n|u)g(u)}{\int_0^1 f(x_1 \dots x_n|v)g(v)dv}$
  - u is the unknown probability that we have an error in our system
  - x_i are "observations" of u: we make a simulation and see if it fails or not
  - g is the probability *prior* distribution of u
    - prior as opposed to *posterior* $f(u|x_1 \dots x_n)$: without having taken samples
    - we will assume it to have a given shape
  - since we assume observations to be independent, $f(x_1 \dots x_n|u) = \prod_{i=1}^{n} f(x_i|u)$
- We want to know p as the probability of the posterior $f(u|x_1 \dots x_n)$
- We use the *posterior Bayes estimator* of p

Bayes Theorem

- From the Bayes theorem it follows that $\int_{t_0}^{t_1} f(u|x_1 \dots x_n)du = F_{(x+\alpha,n-x+\beta)}(t_1) - F_{(x+\alpha,n-x+\beta)}(t_0)$ where:
  - $t_0, t_1 \in (0,1)$
  - $x = \sum_{i=1}^{n} x_i$ is the number of successes in the n trials
  - $\alpha, \beta \in \mathbb{R}^+$ are given parameters
  - $F_{A,B}(t) = \int_0^t g_{A,B}(u)du$ is a Beta distribution function
  - g above is the prior density, here we assume it to be $g_{A,B}(u) = \frac{u^{A-1}(1-u)^{B-1}}{\int_0^1 t^{A-1}(1-t)^{B-1}dt}$
  - thus, $F_{A,B}(t) = \frac{\int_0^t u^{A-1}(1-u)^{B-1}du}{\int_0^1 t^{A-1}(1-t)^{B-1}dt}$
  - F may be easily made explicit, or simply computed using mathematical tools like MATLAB
- When sampling from a Bernoulli distribution with a Beta prior of parameters α, β, it is known that the mean of the posterior is $\hat{p} = \frac{x+\alpha}{n+\alpha+\beta}$

The Algorithm for BSMC

- BSMC: Bayes-based Statistical Model Checking
- The input is as follows:
  - $\mathcal{S}$ as the simulator model for the system to be verified
    - may be black-box, Simulink, Modelica or proprietary
    - must have some probabilistic behaviour, i.e., 2 consecutive simulations may have different results
  - φ as the BLTL property to be verified
    - Bounded LTL: all **U** operators must be bounded, i.e., they are of the form $\mathbf{U}^{\leq t}$, with $t > 0$
    - hence, also **F** and **G** must be bounded too
  - $\alpha, \beta \in \mathbb{R}^+$ as the parameters for the prior Beta distribution
  - $\delta \in (0, 1)$ as the desired size of the output interval
  - $c \in (\frac{1}{2}, 1)$ as the desired interval coverage coefficient

The Algorithm for BSMC

- The output is as follows:
  - (t_0, t_1) such that $t_1 - t_0 = \delta$
  - $\hat{p}$ as the estimate of the probability p that $\mathcal{S} \models \varphi$
- It holds that:
  - (t_0, t_1) is a $100c$ *Bayesian interval estimate*
  - $\hat{p} \in (t_0, t_1)$
    - usually at half interval, but with some adjustments
- Thus, we want δ to be small
  - implies our output interval is narrow, and the estimate is accurate
- We want c to be high
  - implies we are confident on the estimate
- Needless to say, the smaller δ and the higher c, the higher computation time required
- All this assuming no errors are found, otherwise FAIL and ok

The Algorithm

```
BSMC(ProbModel 𝒮, BLTL property φ,
     double α, β, c, δ) {
  (n, x) = (0, 0);
  do {
    σ = simulate(𝒮, time(φ));
    n = n + 1; if (σ ⊨ φ) x = x + 1;
    p̂ = (x+α)/(n+α+β); (t0, t1) = (p̂ − δ/2, p̂ + δ/2);
    if (t1 > 1) (t0, t1) = (1 − δ, 1);
    if (t0 < 0) (t0, t1) = (0, δ);
    γ = F_(x+α,n−x+β)(t1) − F_(x+α,n−x+β)(t0);
  } while (γ < c);
  return ⟨(t0, t1), p̂⟩;
}
```

BLTL Logic, Formally

$$\Phi ::= p \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid (\Phi) \mid \Phi_1 \, \mathbf{U}^{\leq t} \, \Phi_2$$

- $t \in \mathbb{Q}^+$ is a time
- Atomic propositions p are of the form $y \sim v$, being y a variable in the model, $\sim \in \{<, >, \leq, \geq, =\}$ and $v \in \mathbb{Q}$
- Some other derived operators:
  - of course true, false, OR and other propositional logic connectors
  - future (or eventually): $\mathbf{F}^{\leq t}\Phi = \mathrm{true} \, \mathbf{U}^{\leq t} \, \Phi$
  - globally: $\mathbf{G}^{\leq t}\Phi = \neg(\mathrm{true} \, \mathbf{U}^{\leq t} \, \neg\Phi)$
- As for LTL, $\mathcal{S} \models \varphi$ when, for all executions σ of $\mathcal{S}$, σ satisfies φ
- For a given σ, $\sigma \models \varphi$ iff $\sigma, 0 \models \varphi$

BLTL Logic, Formally

- To define when $\sigma, i \models \varphi$, a recursive definition over the recursive syntax of BLTL is provided
  - recall that $\sigma = (s_0, t_0), \ldots, (s_i, t_i), \ldots$
  - at step $\sigma_i = (s_i, t_i)$, the global time is $\sum_{j=0}^{i-1} t_j$
- $\sigma, i \models y \sim v$ iff $\sigma(i)(y) \sim v$
- $\sigma, i \models \Phi_1 \wedge \Phi_2$ iff $\sigma, i \models \Phi_1 \wedge \sigma, i \models \Phi_2$
- $\sigma, i \models \neg\Phi$ iff $\sigma, i \not\models \Phi$
- $\sigma, i \models \Phi_1 \, \mathbf{U}^{\leq t} \, \Phi_2$ iff
  $\exists k \geq i : \ \sigma, k \models \Phi_2 \wedge \forall i \leq j < k.\ \sigma, j \models \Phi_1$ and $\sum_{j=i}^{k-1} t_j \leq t$
- Note this is different from the bounded semantics of LTL used in Bounded Model Checking
- But somewhat similar to CSL

On BSMC Algorithm

- Crucial steps in BSMC algorithm:
  - `simulate`, i.e., invoking our simulator, whatever it is
  - evaluating $\sigma \models \varphi$
- Does `simulate` actually returns σ?
  - typically, simulators output is a log with lines $(t_i, v_{i1}, \ldots, v_{in})$
  - being $v_{i1}, \ldots, v_{in}$ the values at time t_i for each of the n variables used in the simulator model
  - usually, state locations may be inferred from $v_{i1}, \ldots, v_{in}$
  - usually, $t_{i+1} = t_i + \Delta t$ for a fixed (and small) $\Delta t > 0$
  - thus, a simple postprocess computation may translate the log in an execution $\sigma = (s_0, t_0), \ldots, (s_i, t_i), \ldots$
  - this also allows to compute $\sigma(i)(y)$ for any variable y

On BSMC Algorithm

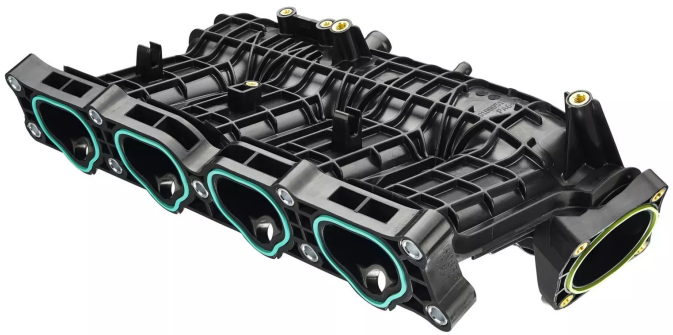
- The first 2 inputs of the BSMC algorithm are straightforward
  - if I want to verify something, of course I need a model and a property
- We may understand δ, c: they control accuracy and confidence of the result
  - the more accuracy/confidence is required, the longer the computation
- What about α, β?
  - informally, it is a measure of the "weight" we *believe* passes and fails should have
  - if none is known, it is probably good to choose a *uniform* Beta distribution, i.e., $\alpha = \beta$
  - e.g., $\alpha = \beta = 1$

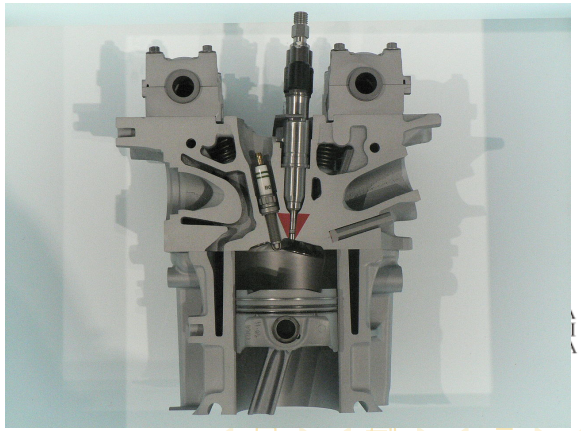
On BSMC Algorithm

- It may be proven that BSMC nearly always terminates:
  - for all possible valid inputs, BSMC terminates with probability 1
  - no, this does not imply that BSMC always terminates (we are in an infinite space)
  - but it is enough for practical applications
- It may be proven that errors on BSMC output are unlikely
  - let our null hypothesis be $p \in (t_0, t_1)$
  - both type-I and type-II errors are bounded by $\frac{\pi_0(1-c)}{c(1-\pi_0)}$
    - recall: type-I is saying that $p \notin (t_0, t_1)$ when instead $p \in (t_0, t_1)$
    - recall: type-II is saying that $p \in (t_0, t_1)$ when instead $p \notin (t_0, t_1)$
  - c is the coverage input as in BSMC
  - π_0 is the actual (prior) probability that $p \in (t_0, t_1)$

BSMC Results

- Case study: Fault-Tolerant Fuel Control System
  - for details, see `http://www.mathworks.com/help/simulink/examples/modeling-a-fault-tolerant-fuel-control-system.html`
- Gasoline engine (e.g., used in avionics), must provide power for vehicle operations
- This model focuses on a critical parameter: the air/fuel rate, which must be kept close to a reference value, i.e., 14.6
  - air is pumped away by intake manifold, fuel is pumped in by injectors





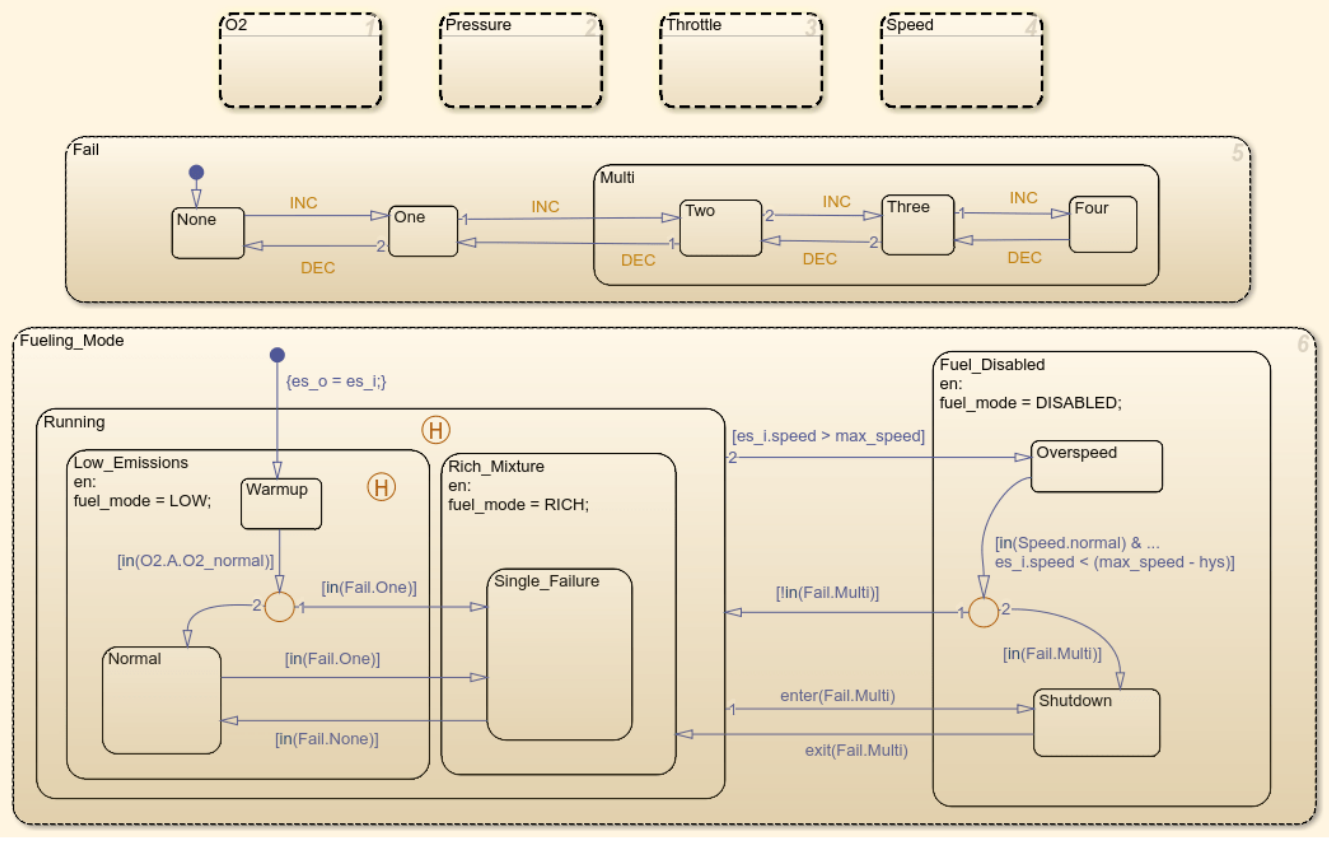
BSMC Results

- The model uses sensors for some key measurements: EGO (exhaust gas residual oxygen), engine speed, throttle, pressure
- If all sensors works well, it is rather easy to control the actuators so that the air/fuel ratio is 14.6
  - the actuator is on the fuel rate
  - i.e., the controller may adjust the air/fuel fraction by changing the denominator
- But sensors may fail: the controller is able to detect such failures and adjust fuel rate
  - by increasing it
- If more than one sensor fail, the engine is shut down
- One a sensor fail, it resumes its operation after one second

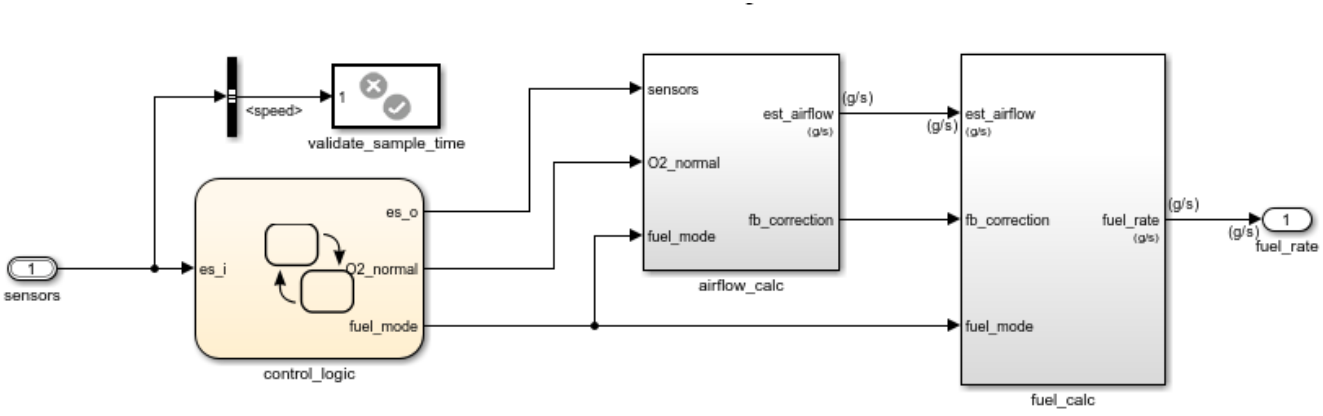
BSMC Results

- We need a stochastic system, thus sensor failures are made probabilistic
  - independent Poisson processes with different arrival rates: $\mathbb{P}(N(t) = n) = \frac{\lambda^n t^n}{n! e^{\lambda t}}$
  - MATLAB already has functions to do this
- The other parts of the system are deterministic
  - there should be the throttle command as input, but it is replaced by a triangular deterministic input

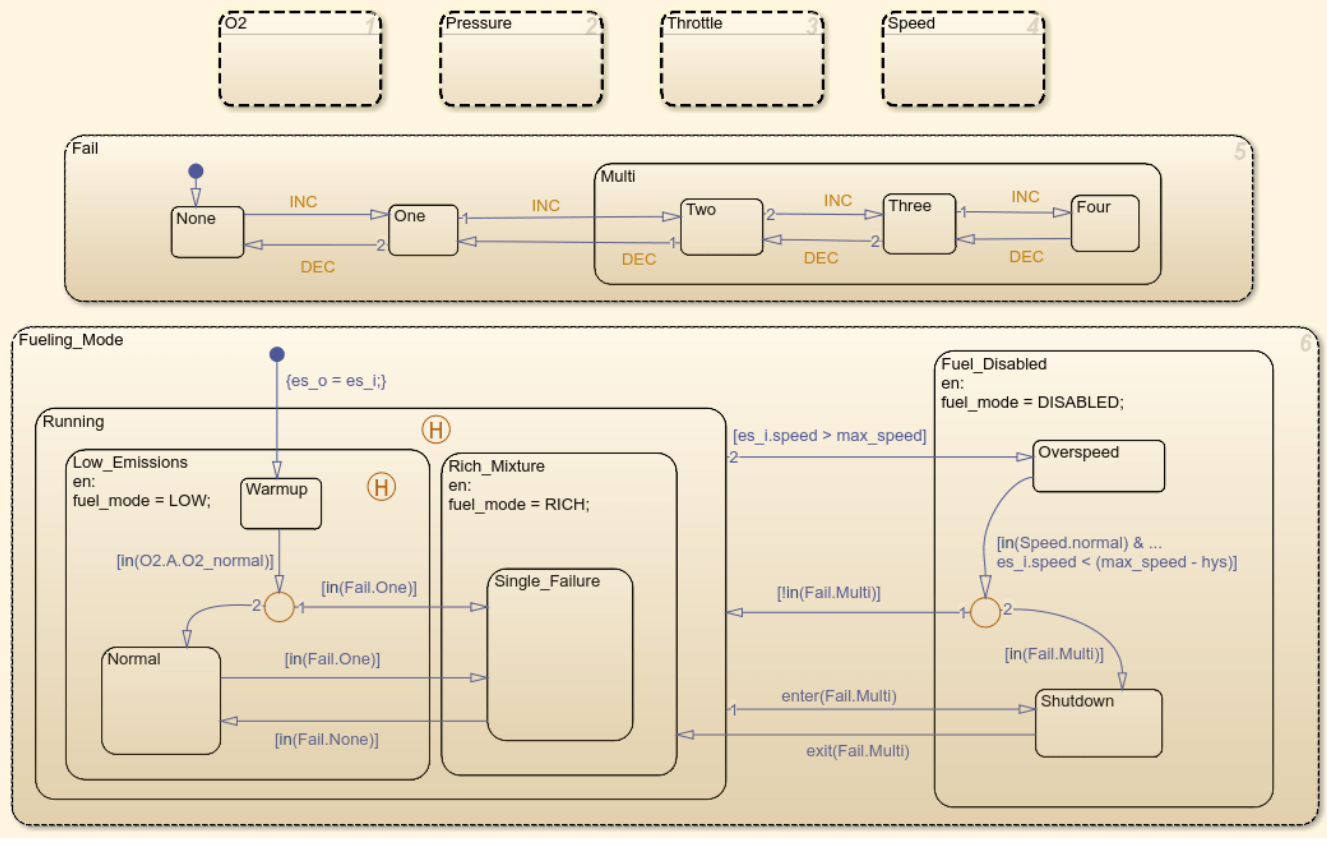
BSMC Results



BSMC Results



BSMC Results



BSMC Results

- BLTL formula to be checked: $\neg \mathbf{F}^{100}\mathbf{G}^{1}\mathrm{FuelFlowRate} = 0$
  - it must not happen that, within 100 seconds, the fuel flow rate becomes zero for 1 second
  - referred as (15) in the tables following
- Different experiments varying:
  - $(\delta, c) \in \{0.05, 0.01\} \times \{0.99, 0.999\}$ (4 possible pairs)
  - fault rates for sensors in $\{(3,7,8),(10,8,9),(20,10,20),(30,30,30)\}$
- The higher the rates, the higher the error probability...
- The C-H bound is also computed: how many experiments should be done with the Chernoff-Hoeffding methodology
  - C-H bound only depends on δ, c, whilst the given algorithm dynamically adjusts the number of samples

BSMC Results

Table 3 Posterior mean/number of samples for estimating probability of (15) with uniform prior and $\delta = 0.05$, and sample size required by the Chernoff-Hoeffding bound [27]

		Interval coverage c	
		0.99	0.999
Fault rates	(3 7 8)	0.3569/606	0.3429/972
	(10 8 9)	0.8785/286	0.8429/590
	(20 10 20)	0.9561/112	0.9625/158
	(30 30 30)	0.9778/43	0.9851/65
	C-H bound	922	1382

Table 4 Posterior mean/number of samples when estimating probability of (15) with uniform prior and $\delta = 0.01$, and sample size required by the Chernoff-Hoeffding bound [27]

		Interval coverage c	
		0.99	0.999
Fault rates	(3 7 8)	0.3558/15205	0.3563/24830
	(10 8 9)	0.8528/8331	0.8534/13569
	(20 10 20)	0.9840/1121	0.9779/2583
	(30 30 30)	0.9956/227	0.9971/341
	C-H bound	23026	34539

Statistical Model Checking: Last Improvements

- Jegourel, Sun, Dong: "Sequential Schemes for Frequentist Estimation of Properties in Statistical Model Checking", ACM Transactions on Modeling and Computer Simulation, Vol. 29, No. 4, Article 25, 2019
- Main result: reduce the amount of samples needed to get the final answer
- In Bayesian SMC, the probability to estimate must be given by a prior random variable whose density is based on previous experiments and knowledge about the system
- This article focuses on "frequentist" estimation approaches to overcome this problem
  - the battle "Bayesian" vs. "frequentist" is frequent in Statistics

Theoretical Framework

- A *stochastic system* $\mathcal{S}$ is a set of interacting components in which the state is determined randomly w.r.t. a global probability distribution
  - this means that there must be something probabilistic, so that different runs may have different outcome
  - could also be a totally deterministic system, but with input probabilistically picked from an input space
- Let $(\Omega, \mathcal{F}, \mu)$ be the probability space induced by the system:
  - Ω is the set of finite paths of $\mathcal{S}$
  - $\mathcal{F}$ is a σ-algebra of Ω
  - μ the probability distribution defined over $\mathcal{F}$
- We consider properties φ that are violated or satisfied by an arbitrary execution of the system with probability 1 in finite time
  - SMC may also address the problem of verifying whether a property probability exceeds a threshold or not

UNIVERSITÀ DEGLI STUDI DELL'AQUILA

DISIM
Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Theoretical Framework: Notation

- $\mathcal{S} \models \mathbf{P}(\varphi) = \gamma$ iff $\mathbf{P}(\varphi) = \gamma$ in the probability space $(\Omega, \mathcal{F}, \mu)$
- $\mathcal{S} \models^{a}_{\varepsilon} \mathbf{P}(\varphi) = \gamma$ iff $\mathbf{P}(\varphi) \in [\gamma - \varepsilon, \gamma + \varepsilon]$
  - *absolute* margin of error
- $\mathcal{S} \models^{r}_{\varepsilon} \mathbf{P}(\varphi) = \gamma$ iff $\mathbf{P}(\varphi) \in [(1 - \varepsilon)\gamma, (1 + \varepsilon)\gamma]$
  - *relative* margin of error
- $\mathcal{S} \models^{a}_{\varepsilon,\delta} \mathbf{P}(\varphi) = \hat{\gamma}_n$ iff $\mathbf{P}(\mathbf{P}(\varphi) \in [\hat{\gamma}_n - \varepsilon, \hat{\gamma}_n + \varepsilon]) \geq 1 - \delta$
- $\mathcal{S} \models^{r}_{\varepsilon,\delta} \mathbf{P}(\varphi) = \hat{\gamma}_n$ iff
  $\mathbf{P}(\mathbf{P}(\varphi) \in [(1 - \varepsilon)\hat{\gamma}_n, (1 + \varepsilon)\hat{\gamma}_n]) \geq 1 - \delta$
  - $\hat{\gamma}_i$ for $i = 1, \ldots, n$ is a sequence of *estimates* of $\mathbf{P}(\varphi) = \gamma$
  - usually, you run $\mathcal{S}$ for i times and then the estimate $\hat{\gamma}_i$ is the mean on the i values
  - *two sided* bounds

Theoretical Framework

- $z : \Omega \to \{0, 1\}$, $z(\omega) = 1$ iff $\omega \models \varphi$
- If ω is extracted from Ω with some probability, then we have a Bernoulli variable Z described by z
  - recall that ω is a path of $\mathcal{S}$
- If the probability of extraction is μ, then $\mathcal{S} \models \mathbf{P}(\varphi) = \mathbf{E}_\mu[Z]$
  - by definition, $\gamma = \mathbf{E}_\mu[Z] = \int_\Omega z(\omega) d\mu(\omega)$
  - i.e., the average value γ is the integral of function z w.r.t. distribution μ over space Ω

Theoretical Framework

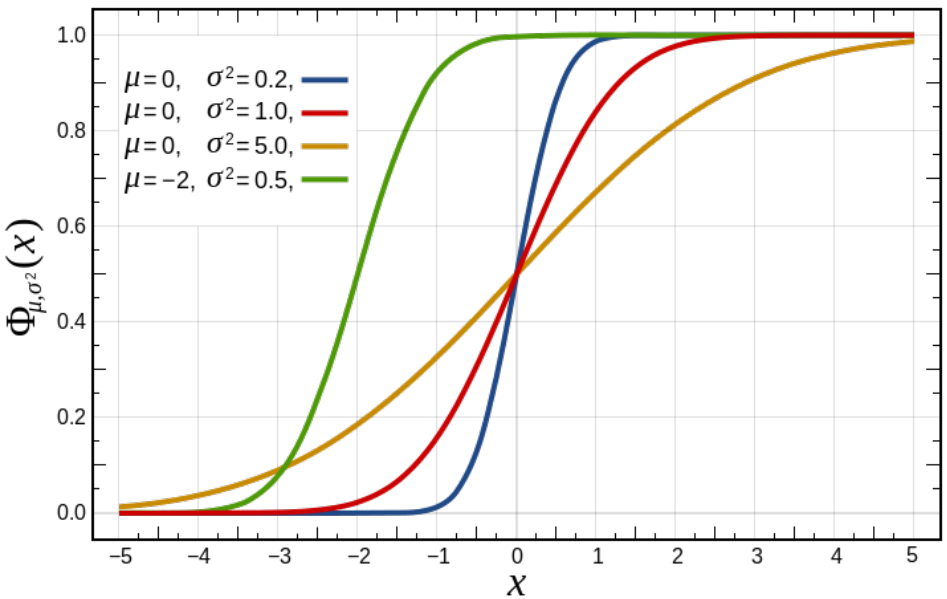
- A (Monte-Carlo) *estimator* runs $\mathcal{S}$ for n times, each time selecting a path $\omega_i \in \Omega$ with probability μ, and then computes $\hat{\gamma}_n = \frac{1}{n}\sum_{i=1}^{n} z(\omega_i) \approx \mathbf{E}_\mu[Z]$
  - path selection may be done offline, online or both
  - depending on $\mathcal{S}$ accepting inputs at the beginning only, at execution time only, or both
- Let $m = \sum_{i=1}^{n} z(\omega_i)$ be the number of successes
  - so, $\hat{\gamma}_n = \frac{m}{n}$
- Let $\sigma^2 = \gamma(1-\gamma)$ be the variance of Z

Theoretical Framework

- Let $I = [\ell_I, u_I]$ be a $(1-\delta)$-confidence interval for a probability γ based on n samples
  - that is, after n samples, we may say $\mathbf{P}(\gamma \in I) \geq 1-\delta$
  - ℓ_I, u_I are two random variables which depend on the number of successes among n samples
- Let $I(m,n)$ be the evaluation of I given m successes on n samples, then the coverage of γ is defined by $C(\gamma, I) = \mathbf{P}(\gamma \in I) = \sum_{m=0}^{n} [\gamma \in I(m,n)] \mathbf{P}(\sum_{i=1}^{n} z(\omega_i) = m)$
  - $[A] = 1$ if A is true, otherwise $[A] = 0$
  - note that $I(m,n)$ varies when n (and possibly m) grows
- Thus, we want $C(\gamma, I_n) \geq 1-\delta$, where I_n is an interval built after n samples
  - how to build it?

Theoretical Framework

- Let Φ be the standard normal distribution function
  - $\Phi_{\mu,\sigma}(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\frac{x-\mu}{\sigma}} e^{-\frac{t^2}{2}} dt$
  - $\Phi_{\mu,\sigma}(x) = \mathbf{P}(X \leq x)$, if X is a random variable with mean μ and variance σ^2 (CDF, Cumulative Distribution Function)
- Let $z_\delta = \Phi^{-1}(1-\delta)$
  - that is, $\mathbf{P}(X \leq z_\delta) = 1 - \delta$



Theoretical Framework

- Thanks to the central limit theorem, we could use the following as our I with $1-\delta$ confidence:

$$\left[\frac{m}{n} - z_{\frac{\delta}{2}}\frac{\sigma}{\sqrt{n}}, \frac{m}{n} + z_{\frac{\delta}{2}}\frac{\sigma}{\sqrt{n}}\right]$$

- Unfortunately, $\sigma = \gamma(1-\gamma)$, so, since we do not know γ, we do not know σ as well
- Thus, we replace $\frac{\sigma}{\sqrt{n}}$ above with the standard deviation of the n samples: $\sigma_n = \sqrt{\frac{m}{n}(1-\frac{m}{n})}$
- Hence, we have

$$I(m,n) = \left[\frac{m}{n} - z_{\frac{\delta}{2}}\sigma_n, \frac{m}{n} + z_{\frac{\delta}{2}}\sigma_n\right]$$

  - with $C(\gamma, I(m,n)) \geq 1-\delta$

UNIVERSITÀ DEGLI STUDI DELL'AQUILA

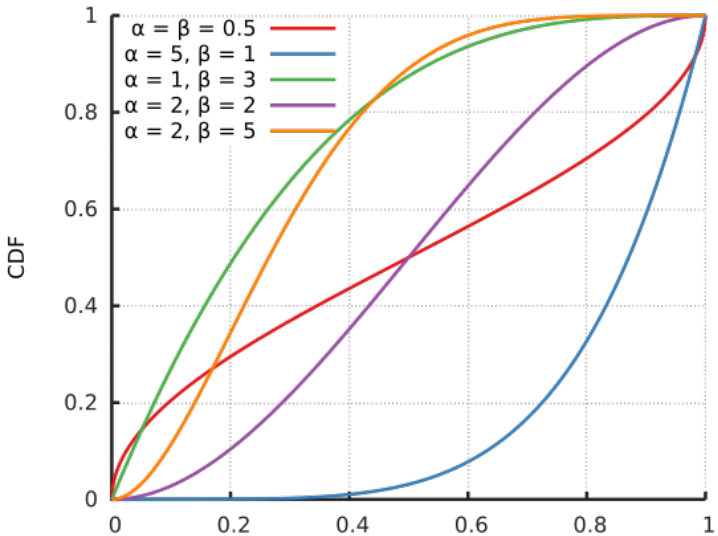
DISIM
Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Theoretical Framework

- An alternative interval providing the same (actually, more strict) guarantee is the Clopper-Pearson interval:

$$J = \left[\beta^{-1}(\frac{\delta}{2}, m, n-m+1), \beta^{-1}(1-\frac{\delta}{2}, m+1, n-m)\right]$$

  - being $\beta(x, a, b) = \int_0^x t^{a-1}(1-t)^{b-1}dt$



Theoretical Framework

- A further alternative interval providing the same guarantee is the Agresti-Coull interval: instead of

$$I(m,n) = \left[\frac{m}{n} - z_{\frac{\delta}{2}}\sigma_n, \frac{m}{n} + z_{\frac{\delta}{2}}\sigma_n\right]$$

we have

$$I(m,n) = \left[\frac{m+\frac{z_\delta^2}{2}}{n+z_\delta^2} - z_{\frac{\delta}{2}}\sigma_{n+z_\delta^2}, \frac{m+\frac{z_\delta^2}{2}}{n+z_\delta^2} + z_{\frac{\delta}{2}}\sigma_{n+z_\delta^2}\right]$$

  - especially useful when $m << n$

Theoretical Framework

- What about the number of samples? We typically want $|I(m,n)| \leq \varepsilon$ (for the absolute error...), so we should go on till when this happens
- Here are the most important upper bounds
- Okamoto (1958): for all $0 < \varepsilon < 1$ we have that

$$\mathbf{P}\left(\left|\frac{m}{n} - \gamma\right| > \varepsilon\right) \leq 2e^{-2n\varepsilon^2}$$

  - we want it to be $\leq \delta$, so we simply put $\delta = 2e^{-2n\varepsilon^2}$
  - hence, the minimum number of samples is $n = \left\lceil \frac{\log \frac{\delta}{2}}{-2\varepsilon^2} \right\rceil$
  - too rough, e.g., for $\delta = 0.1, \varepsilon = 0.001$ we have $n > 10^6$

Theoretical Framework

- Hoeffding (1963), rivisited: for all $0 < \varepsilon, \gamma < 1$ we have that

$$\mathbf{P}\left(\left|\frac{m}{n} - \gamma\right| > \varepsilon\right) \leq 2e^{-n\varepsilon^2 f(\gamma)}$$

being

$$f(\gamma) = \frac{\log \frac{1-\gamma}{\gamma}}{1-2\gamma}$$

  - note that $\gamma \in \{0, 1\}$ is not allowed
  - note that $f(\frac{1}{2})$ is undefined, so we set $f(\frac{1}{2}) = 2$
  - problem: γ is unknown, but this can be tackled in practice
  - better, but still improvable

Theoretical Framework

- Massart (1990): for all $0 < \gamma < 1, 0 < \varepsilon < \min\{\gamma, 1-\gamma\}$ we have that

$$\mathbf{P}\left(\left|\frac{m}{n} - \gamma\right| > \varepsilon\right) \leq 2e^{-n\varepsilon^2 h_a(\gamma,\varepsilon)}$$

being

$$h_a(\gamma,\varepsilon) = \begin{cases} \frac{9}{2(3\gamma+\varepsilon)(3-3\gamma-\varepsilon)} & \text{if } 0 < \gamma < \frac{1}{2} \\ \frac{9}{2(3\gamma+\varepsilon)(3-3\gamma+\varepsilon)} & \text{otherwise} \end{cases}$$

  - again, $\gamma \in \{0, 1\}$ is not allowed
  - furthermore, ε must be smaller than both the searched probability and its complement
  - problem: γ is unknown, but this can be tackled in practice

Theoretical Framework

- Hoeffding (before 1979), rivisited: for all $0 < \varepsilon, \gamma < 1$ we have that

$$\mathbf{P}\left(\left|\frac{m}{n} - \gamma\right| > \varepsilon\gamma\right) \leq 2e^{-\frac{n\varepsilon^2\gamma}{2+\varepsilon}}$$

- Massart (1990): for all $0 < \gamma < 1, 0 < \varepsilon < \frac{1-\gamma}{\gamma}$ we have that

$$\mathbf{P}\left(\left|\frac{m}{n} - \gamma\right| \geq \varepsilon\gamma\right) \leq 2e^{-n\varepsilon^2 h_r(\gamma,\varepsilon)}$$

being

$$h_r(\gamma,\varepsilon) = \begin{cases} \frac{9\gamma}{2(3\gamma+\varepsilon)(3-3\gamma-\gamma\varepsilon)} & \text{if } 0 < \gamma < \frac{1}{2} \\ \frac{9\gamma}{2(3\gamma-\varepsilon)(3-3\gamma+\gamma\varepsilon)} & \text{otherwise} \end{cases}$$

Theoretical Framework

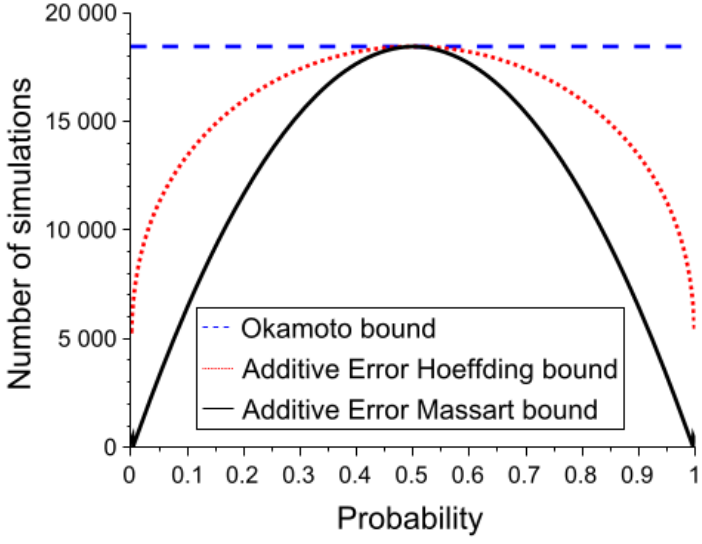


Fig. 1. Okamoto (dash), Hoeffding (dot), and Massart (plain) bounds with absolute error $\epsilon = 0.01$ and confidence parameter $\delta = 0.05$.

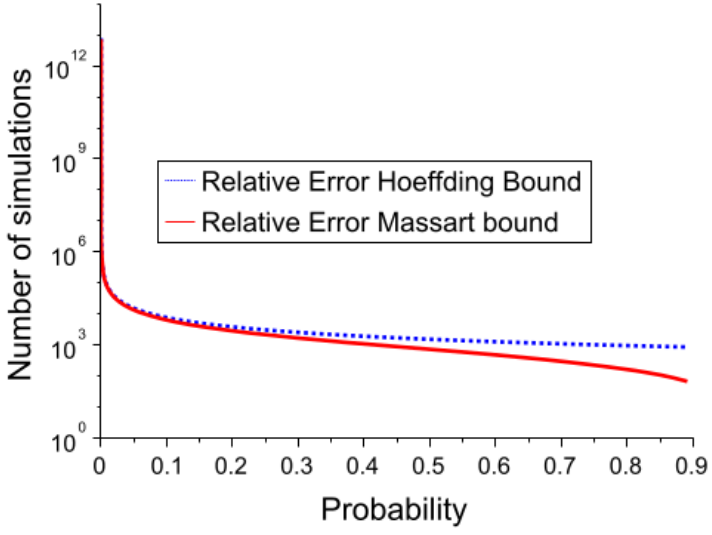


Fig. 2. Hoeffding (dot) and Massart (plain) bounds with relative error $\epsilon = 0.1$ and confidence parameter $\delta = 0.05$.

Algorithms

```
SMC0(procedure_for_Z P_Z, double ε, double δ) {
  n = m = 0;
  do {
    n += 1;
    if (P_Z())
      m += 1;
  } while (n < log(δ/2) / (-2ε²));
  return m/n;
}
```

Ok for absolute error specification; too conservative (i.e., more samples than necessary)

Algorithms: UPPAAL-SMC

```
SMC1(procedure_for_Z P_Z, double ε, double δ) {
  n = m = 0;
  do {
    n += 1;
    if (P_Z())
      m += 1;
    [l, u] = [β^{-1}(δ/2, m, n − m + 1), β^{-1}(1 − δ/2, m + 1, n − m)];
  } while (u − l > 2ε);
  return (u−l)/2;
}
```

No guarantee that $\mathcal{S} \models^{a}_{\varepsilon,\delta} \mathbf{P}(\varphi) = \hat{\gamma}_n$!
Approximation may be biased; absolute error specification

Algorithms

```
SMC2(procedure_for_Z P_Z, double ε, double δ) {
  n = m = 0;
  do {
    n += 1;
    if (P_Z())
      m += 1;
  } while (n < 2 (log δ/2) / ε^2 (1/4 − (|m/n − 1/2| − 2/3))^2);
  return m/n;
}
```

Ok for $\mathbf{P}(\mathbf{P}(\varphi) \leq \hat{\gamma}_n + \varepsilon) \geq 1 - \delta$

The other bound is only conjectured; absolute error specification

Algorithms: OAA

SMC3(procedure_for_Z $\mathcal{P}_Z$, **double** ε, **double** δ) {

$\hat{\mu}_Z = \mathtt{SRA}(\mathcal{P}_Z, \min\{\frac{1}{2}, \sqrt{\varepsilon}\}, \frac{\delta}{3})$;

$\Upsilon = 2(1+\sqrt{\varepsilon})(1+2\sqrt{\varepsilon})\left(1+\frac{\log(3)-\log(2)}{\log(2)-\log(\delta)}\right)\frac{4(e-2)(\log(2)-\log(\delta))}{\varepsilon^2}$;

$N = \frac{\varepsilon\Upsilon}{\hat{\mu}_Z}$;

$S = \frac{1}{2}\sum_{i=1}^{N}(\mathcal{P}_Z() - \mathcal{P}_Z())^2$;

$\rho_Z = \max\left\{\frac{S}{N}, \varepsilon\hat{\mu}_Z\right\}$;

$N = \frac{\rho_Z\Upsilon}{\hat{\mu}_Z^2}$;

$S = \frac{1}{N}\sum_{i=1}^{N}\mathcal{P}_Z()$;

return $\tilde{\mu}_Z = \frac{S}{N}$;

}

SRA: Stopping Rule for OAA

```
SRA(procedure_for_Z 𝒫_Z, double ε, double δ) {
   ϒ = 1 + (1 + ε) 4(e−2)(log(2)−log(δ)) / ε² ;
   N = 1;
   S = 0;
   while (S ≤ ϒ) {
      N = N + 1;
      S = S + 𝒫();
   }
   return μ̂_Z = S/N;
}
```

Ok for any random variable in $[0, 1]$, thus not optimized for Bernoulli variables

Relative error

Algorithms: Watanabe

```
SMC4(procedure_for_Z P_Z, double ε, double δ) {
  n = m = 0;
  do {
    n += 1;
    if (P_Z())
      m += 1;
  } while (m < (3+3ε)/ε^2 log 2/δ);
  return m/n;
}
```

Surprising (what if the mean is close to 0???), but it works; relative error specification

The Ultimate SMC Algorithm (Absolute Error)

```
SMC(procedure_for_Z P_Z, double ε, double δ,
    double δ') {
  assert(δ' < δ);
  n = m = 0; ℓ = M = ⌈ log(2/δ) / (2ε²) ⌉;
  do {
    n += 1;
    if (P_Z()) m += 1;
    [l, u] = [β⁻¹(δ'/2, m, n − m + 1), β⁻¹(1 − δ'/2, m + 1, n − m)];
    if (1/2 ∈ [l, u]) ℓ = M;
    else if (u < 1/2) ℓ = ⌈ 1/(h_a(u,ε)ε²) log 2/(δ−δ') ⌉;
    else ℓ = ⌈ 1/(h_a(l,ε)ε²) log 2/(δ−δ') ⌉;
    ℓ = min{ℓ, M};
  } while (n < ℓ);
  return m/n;
}
```

UNIVERSITÀ DEGLI STUDI DELL'AQUILA

DISIM
Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

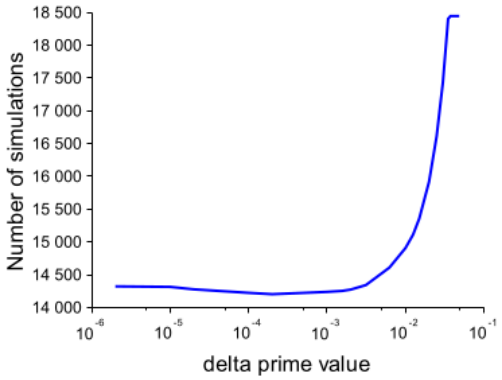
The Ultimate SMC Algorithm (Relative Error)

```
SMC(procedure_for_Z P_Z, double ε, double δ,
    double δ', double γ_min) {
  assert(δ' < δ);
  n = m = 0; ℓ = M = ⌈ log(2/δ) / (h_r(γ_min, ε) ε²) ⌉;
  do {
    n += 1;
    if (P_Z()) m += 1;
    [l, u] = [β⁻¹(δ'/2, m, n − m + 1), β⁻¹(1 − δ'/2, m + 1, n − m)];
    if (γ_min ≥ l) ℓ = M;
    else ℓ = ⌈ 1/(h_r(l, ε) ε²) log 2/(δ − δ') ⌉;
    ℓ = min{ℓ, M};
  } while (n < ℓ);
  return m/n;
}
```

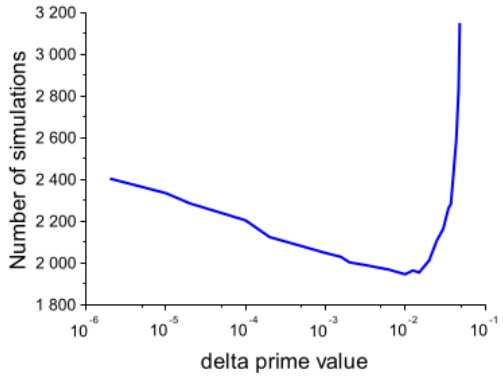
The Ultimate SMC Algorithm

- There are either one or two additional parameters
  - a δ' is present for both relative and absolute error algorithms
  - a γ_{min} is present for the relative error algorithm only
- Actually, γ_{min} is not a problem
  - we must guarantee that $\gamma_{min} \leq \gamma$
  - either there is not an error (OK!), or γ_{min} may be set, e.g., to the IEEE-754 precision
- Thus we focus on the parameter $\delta' < \delta$: given a δ, which value should we choose?
  - impossible to a-priori optimise δ': it also depends on the unknown γ
  - empirically, better to set it closer to 0 than to δ

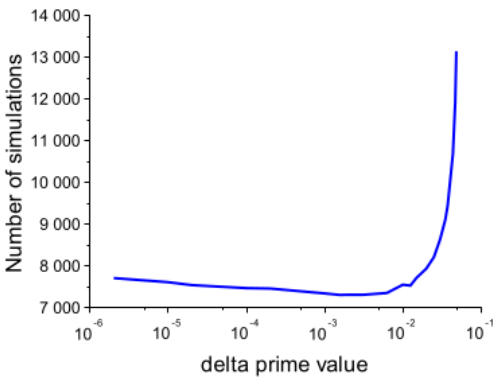
The Ultimate SMC Algorithm



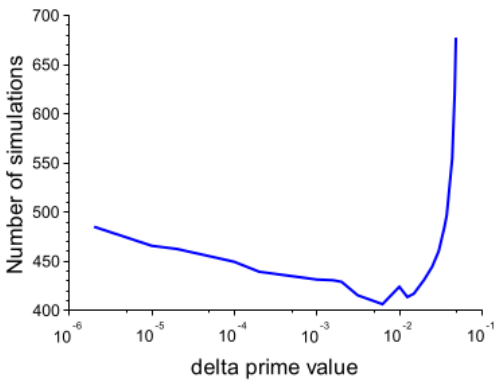
(a) Number of simulations for various δ', given an absolute error $\epsilon = 0.01$, confidence parameter $\delta = 0.05$ and $\gamma = 0.25$.



(b) Number of simulations for various δ', given an absolute error $\epsilon = 0.01$, confidence parameter $\delta = 0.05$ and $\gamma = 0.02$.



(c) Number of simulations for various δ', given relative error $\epsilon = 0.1$, confidence parameter $\delta = 0.05$ and $\gamma = 0.1$



(d) Number of simulations for various δ', given an absolute error $\epsilon = 0.1$, confidence parameter $\delta = 0.05$ and $\gamma = 0.7$.

Fig. 3. Number of simulations for δ'.

Experimental Results

Table 1. Sampling Size Gains over Standard PRISM Benchmarks

	Probability γ	APMC (ϵ, δ)	(AE) Gain	Dagum (ϵ, δ)	(RE) Gain
tandem	0.155132	$(0.01, 0.001)$	1.7	$(0.05, 0.001)$	5.18
polling	0.540786	$(0.001, 0.01)$	1	$(0.01, 0.01)$	3.65
cluster	5.160834×10^{-4}	$(10^{-4}, 0.05)$	399	$(0.2, 0.05)$	9

Gain is the ratio between the number of samples required by the most standard approaches (APMC in the absolute error case and Dagum et al. in the relative error case) and our algorithms. Gains > 1 imply that our algorithms require less samples.

Experimental Results

On a toy example

Table 3. Descriptive Statistics, Coverage, and Sample Size Average of the Absolute Error Algorithms with $\epsilon = 0.01$ and $\delta = 0.05$

γ	0.005	0.01	0.02	0.05	0.1	0.3	0.5
Coverage (simple)	1	0.965	**0.94**	0.96	0.965	0.975	**0.945**
$\hat{\gamma}$ min (simple)	0	0	**0.007**	**0.036**	**0.087**	0.288	**0.484**
$\hat{\gamma}$ max (simple)	0.013	0.021	0.029	0.062	**0.113**	**0.316**	**0.513**
$\bar{N}$ (simple)	518	729	1,107	2,172	3,777	8,278	9,703
Coverage (Chen)	1	0.98	1	0.995	1	0.995	0.995
$\hat{\gamma}$ min (Chen)	0	0	0.011	0.04	0.091	0.292	0.492
$\hat{\gamma}$ max (Chen)	0.01	0.017	0.028	0.059	0.107	0.31	0.511
$\bar{N}$ (Chen)	810	1,171	1,900	3,946	7,035	15,684	18,444
Coverage (new, $\delta' = 0.025$)	1	0.99	0.995	0.995	0.995	1	1
$\hat{\gamma}$ min (new)	0	0	0.01	0.039	0.089	0.291	0.491
$\hat{\gamma}$ max (new)	0.011	0.019	0.027	0.059	0.106	0.309	0.51
$\bar{N}$ (new)	831	1,229	2,064	4,474	8,161	18,434	18,445
Coverage (new, $\delta' = 0.001$)	1	1	0.995	0.995	0.99	0.99	1
$\hat{\gamma}$ min (new)	0	0.003	0.01	0.04	0.089	0.29	0.488
$\hat{\gamma}$ max (new)	0.009	0.015	0.028	0.058	0.114	0.311	0.508
$\bar{N}$ (new)	971	1,318	2,031	4,095	7,192	15,826	18,445

DELL AQUILA

DISIM
Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

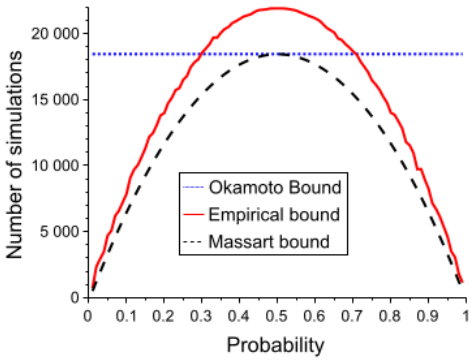
Experimental Results

On a toy example

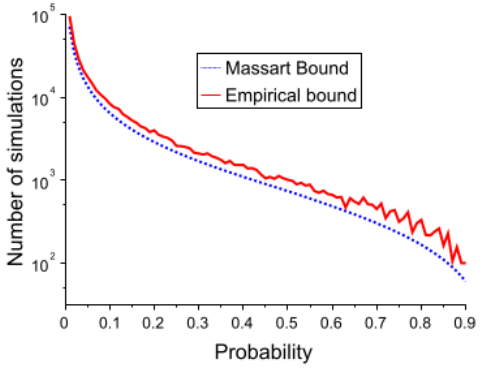
Table 4. Sample Size Average of the Relative Error Algorithms, Given ϵ and δ

γ	0.9	0.7	0.5	0.3	0.1	0.05	0.01	0.001
$\bar{N}$ Dagum, $(\epsilon, \delta) = (0.1, 0.01)$	1,871	4,402	9,056	19,703	74,064	152,757	803,572	8,124,356
$\bar{N}$ Dagum, $(\epsilon, \delta) = (0.1, 0.05)$	1,360	3,160	6,412	14,253	52,432	111,703	570,763	5,787,456
$\bar{N}$ Dagum, $(\epsilon, \delta) = (0.05, 0.05)$	3,162	8,912	19,244	43,276	163,084	346,269	1,800,585	18,208,080
$\bar{N}$ Dagum, $(\epsilon, \delta) = (0.05, 0.01)$	4,394	12,337	26,677	60,263	226,889	479,164	2,467,430	25,300,472
$\bar{N}$ W., $(\epsilon, \delta) = (0.1, 0.01)$	1,942	2,498	3,501	5,836	17,479	35,006	175,092	1,746,713
$\bar{N}$ W., $(\epsilon, \delta) = (0.1, 0.05)$	1,353	1,738	2,439	4,048	12,207	24,362	122,029	1,218,779
$\bar{N}$ W., $(\epsilon, \delta) = (0.05, 0.05)$	5,163	6,634	9,299	15,453	46,496	92,950	465,144	4,650,289
$\bar{N}$ W., $(\epsilon, \delta) = (0.05, 0.01)$	7,416	9,540	13,347	22,235	66,756	133,581	665,536	6,677,525
$\bar{N}$ New, $(\epsilon, \delta, \delta') = (0.1, 0.01, 0.005)$	202	623	1,373	3,043	11,365	23,812	122,426	1,236,491
$\bar{N}$ New, $(\epsilon, \delta, \delta') = (0.1, 0.05, 0.025)$	137	441	991	2,204	8,208	17,356	88,838	895,496
$\bar{N}$ New, $(\epsilon, \delta, \delta') = (0.05, 0.05, 0.025)$	476	1,631	3,737	8,473	32,175	67,850	348,706	3,515,688
$\bar{N}$ New, $(\epsilon, \delta, \delta') = (0.05, 0.01, 0.005)$	669	2,266	5,151	11,675	44,346	93,236	482,998	4,871,059
$\bar{N}$ New, $(\epsilon, \delta, \delta') = (0.1, 0.01, 0.001)$	204	583	1,273	2,822	10,484	21,930	112,880	1,135,687
$\bar{N}$ New, $(\epsilon, \delta, \delta') = (0.1, 0.05, 0.001)$	156	431	905	1,970	7,333	15,310	67,511	789,934
$\bar{N}$ New, $(\epsilon, \delta, \delta') = (0.05, 0.05, 0.001)$	471	1,489	3,296	7,422	27,951	58,724	301,258	3,043,438
$\bar{N}$ New, $(\epsilon, \delta, \delta') = (0.05, 0.01, 0.001)$	648	2,119	4,701	10,686	40,303	84,880	438,929	4,438,120

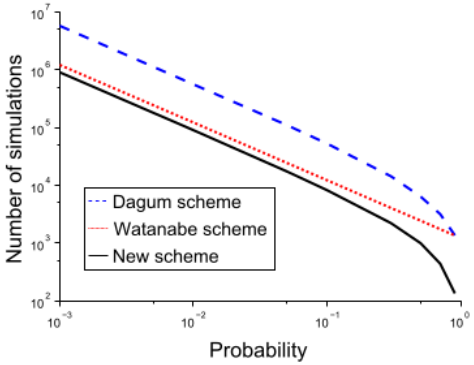
Experimental Results



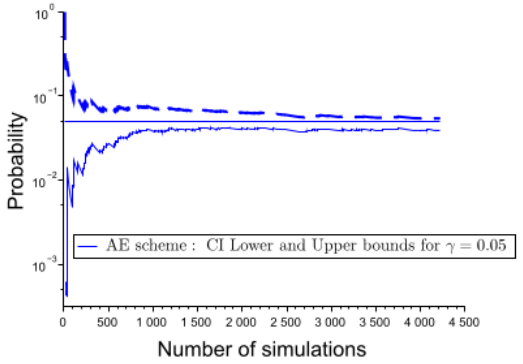
(a) Notional Okamoto (dot) and Massart (dash) bounds versus empirical results (absolute error $\epsilon = 0.01$ and confidence parameter $\delta = 0.05$).



(b) Massart bounds (dot) versus empirical bounds (relative error $\epsilon = 0.1$ and confidence parameter $\delta = 0.05$).



(c) Comparison between Dagum-and-al. (above), Watanabe (middle) and new (below) relative error algorithms ($\epsilon = 0.1$ and $\delta = 0.05$).



(d) Evolution of the confidence interval bounds for the Absolute Error algorithm with $\gamma = 0.05$.

Fig. 4. Experimental results with coverage parameter $\delta' = 0.025$.

A Better Stopping Rule Algorithm

- It may be shown that OAA is "optimal" w.r.t. the number of samples
- That is, any other stopping rule (ε, δ) approximation may be faster by at most a multiplicative constant
  - OAA: $\exists C : \mathbf{P}(T \leq C \frac{\log 2 - \log \delta}{\varepsilon^2 \mu^2} \max\{\sigma^2, \varepsilon\mu\}) \geq 1 - \delta$
  - where μ, σ are the mean and the variance of the samples
  - any other stopping rule:
    $\exists C' : \mathbf{P}(T \geq C' \frac{\log 2 - \log \delta}{\varepsilon^2 \mu^2} \max\{\sigma^2, \varepsilon\mu\}) \geq 1 - \delta$
  - recall that an (ε, δ) approximation is s.t.
    $\mathbf{P}(|\hat{\mu} - \mu| \leq \varepsilon|\mu|) \geq 1 - \delta$

A Better Stopping Rule Algorithm

- Adapted from Mnih, Szepesvári, Audibert: "Empirical Bernstein Stopping", Proc. of ICML 2008
- Slightly more general setting: we want to compute a set $\mathcal{K}$ of KPIs
  - Key Performance Indicator for some cyber-physical system
  - e.g.: average response time, throughput, ...
  - OAA is the same with $|\mathcal{K}| = 1$, and with the only KPI accepting values in $(0, 1]$
- We need two parameters for the algorithm: β, p
  - typical values $\beta = p = 1.1$
- $m = 1, 2$, i.e., there are two alternative versions
  - we pick $m = 2$

A Better Stopping Rule Algorithm

function Estimate$(\varepsilon, \delta, \beta, p, m, \mathcal{K}, \mathcal{S})$
  $\boldsymbol{R}, \boldsymbol{L}, \boldsymbol{U}, t, k \leftarrow \mathbf{0}, \mathbf{0}, \boldsymbol{\infty}, 1, 0;$
  $\boldsymbol{y} \leftarrow \text{run}(\mathcal{S});$
  foreach $i \in \mathcal{K}$ **do** $X_{1i} \leftarrow y_i;$
  while $\exists i \in \mathcal{K}\ (1+\varepsilon)L_i < (1-\varepsilon)U_i$ **do**
    $t \leftarrow t+1;$
    $\boldsymbol{y} \leftarrow \text{run}(\mathcal{S});$
    foreach $i \in \mathcal{K}$ **do**
      $X_{ti} \leftarrow y_i;$
      $R_i \leftarrow \max_t X_{ti} - \min_t X_{ti};$
    if $t > \lfloor \beta^k \rfloor$ **then**
      $k \leftarrow k+1;$
      $\alpha \leftarrow \frac{\lfloor \beta^k \rfloor}{\lfloor \beta^{k-1} \rfloor};$
      if $m = 1 \vee k = 1$ **then** $d_k \leftarrow \frac{1}{k(k+1)};$
      else $d_k \leftarrow \frac{\delta(p-1)}{p(\log_\beta k)^p};$
      $x \leftarrow \alpha \log \frac{3}{d_k};$
    foreach $i \in \mathcal{K}$ **do**
      $\bar{X}_{ti} \leftarrow \frac{1}{t}\sum_{j=1}^{t} X_{ji};$
      $\bar{\sigma}_{ti} \leftarrow \sqrt{\frac{1}{t}\sum_{j=1}^{t}(X_{ji} - \bar{X}_{ti})^2};$
      $c_{ti} \leftarrow \bar{\sigma}_{ti}\sqrt{\frac{2x}{t}} + 3R_i\frac{x}{t};$
      $L_i \leftarrow \max\{L_i, |\bar{X}_{ti}| - c_{ti}\};$
      $U_i \leftarrow \min\{U_i, |\bar{X}_{ti}| + c_{ti}\};$
  foreach $i \in \mathcal{K}$ **do** $r_i \leftarrow \frac{1}{2}\text{sgn}(\bar{X}_{ti})((1+\varepsilon)L_i + (1-\varepsilon)U_i);$
  return $\boldsymbol{r};$

A Better Stopping Rule Algorithm

- Just one phase; as for number of samples T, there exists a constant C s.t.
  - $\exists C : \mathbf{P}(T \leq C(\log \frac{1}{\delta} + \log \frac{R}{\varepsilon|\mu|}) \max\{\frac{\sigma^2}{\varepsilon^2\mu^2}, \frac{R}{\varepsilon|\mu|}\}) \geq 1 - \delta$
- Also works for negative-valued samples (and for variables with mean 0)