Approximation and verification

Sylvain Peyronnet

December 7^{th}, 2010

Research interests

10 years of research

- Approximation and model checking of probabilistic systems
- Uniform generation of executions in large systems

Research interests

10 years of research

- Approximation and model checking of probabilistic systems
- Uniform generation of executions in large systems
- Adversarial information retrieval on the web (not presented)
- Large-scale problems and probabilistic algorithms (not presented)
- Grid computing (not presented)
- Architecture driven compressive sensing algorithms (not presented)

General context

The big picture

system

Does the system satisfy the requirement?

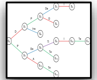
system requirement

General context

The big picture

system

succinct representation of the model



explicit model

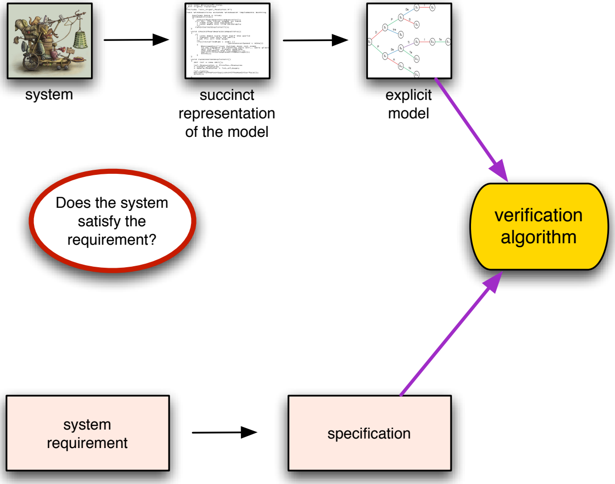
Does the system satisfy the requirement?

system requirement

specification

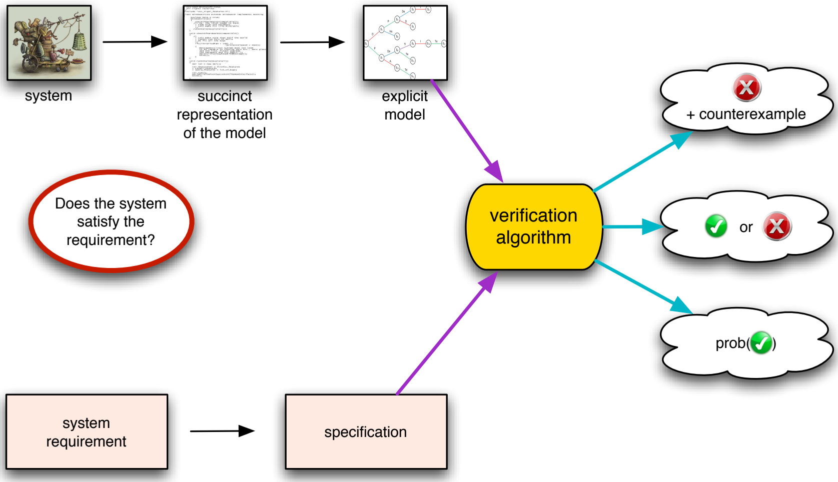
General context

The big picture



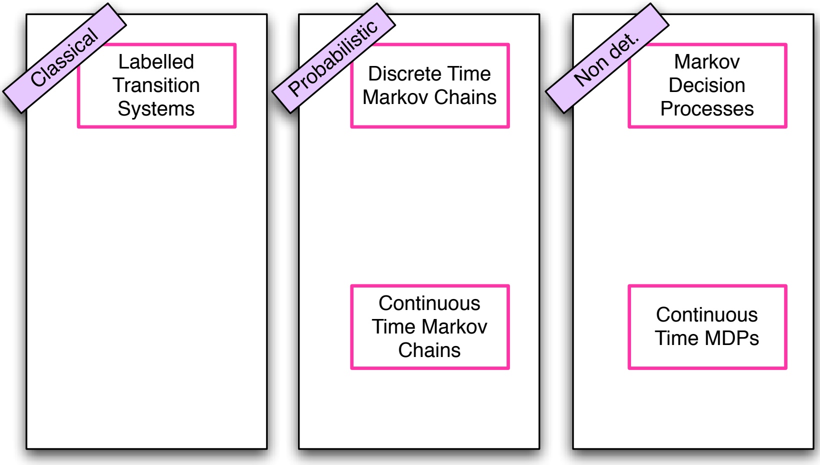
General context

The big picture



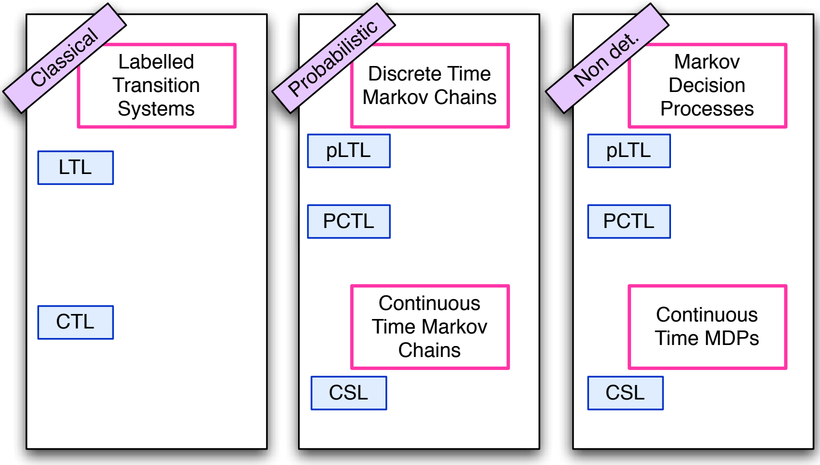
General context

The big picture



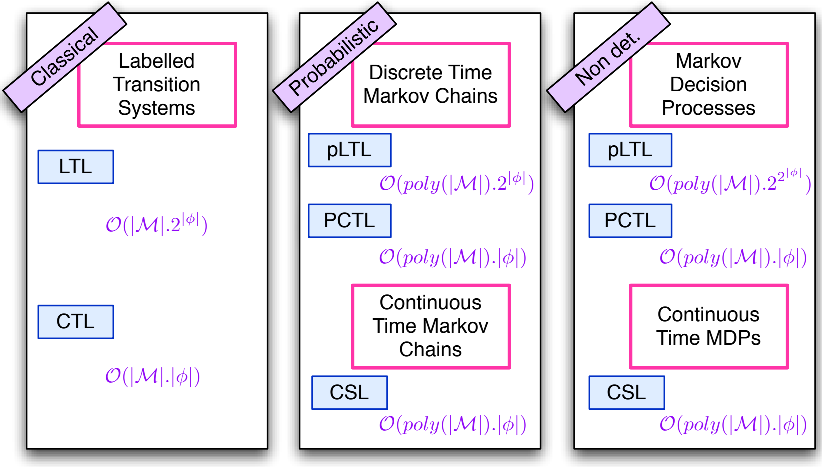
General context

The big picture



General context

The big picture



General context

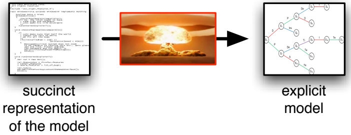
The big picture

LOOKS LIKE THERE IS NO PROBLEM

General context

What's the problem?

The problem is the **state space explosion phenomenon**



• A state-of-the-art probabilistic model checker can handle systems whose size is at most $\approx 10^{11}$ states (PRISM FAQ)

• Several methods are aimed to push the memory wall: symbolic MC, abstraction, composition, etc.

General context

What's the problem?

The pro

My work focuses on approximation methods insensitive to the state space explosion

omenon

of the model

• A state-of-the-art probabilistic model checker can handle systems whose size is at most $\approx 10^{11}$ states (PRISM FAQ)

• Several methods are aimed to push the memory wall: symbolic MC, abstraction, composition, etc.

General context

What's the problem?

The pro omenon

My work focuses on approximation methods insensitive to the state space explosion

of the model

Space complexity sub-linear (log) or independent of the size of the model

Time complexity is not the issue, but the lower is the better

MC, abstraction, composition, etc.

General context

Probabilistic systems

- Can verification be approximated?
- Approximate verification algorithms
  - Additive (absolute) error
  - Multiplicative (relative) error
- Perspectives

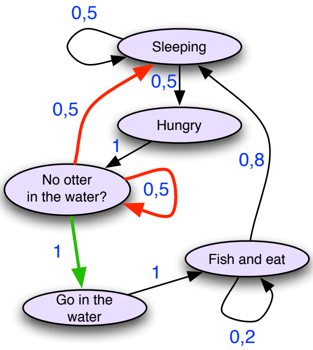
Non probabilistic systems

Approximate Probabilistic Model Checker (APMC)

- Architecture
- Distributed and parallel APMC
- APMC input language
- Perspectives

Conclusion

Discrete Time Markov Chains (DTMCs)



The fishing otter model

- $\mathcal{M} = (S, P, s_0, L)$
  - S: states
  - $P : S \times S \rightarrow [0, 1]$ probabilistic transitions
  - $s_0 \in S$ initial state
  - L labelling function
- execution or path: sequence of states according to P
- $\phi \in LTL$, $Prob(\phi)$ is the measure of paths satisfying ϕ in $\mathcal{M}$

Can verification be approximated? [P01]

Can verification be approximated? [P01]

Randomized Approximation Scheme (RAS)
A randomized algorithm that takes as input the model, the specification ϕ, two real numbers $\varepsilon, \delta > 0$ and produces a value $\tilde{p}$ such that:

$$Pr\big(|\tilde{p} - prob(\phi)| < \varepsilon.prob(\phi)\big) \;\geq\; 1 - \delta \text{ (multiplicative error)}$$

Fully Polynomial RAS (FPRAS)
Running time is polynomial in $|\phi|$, in the size of the succinct representation, in $\frac{1}{\varepsilon}$ and in $\log(\frac{1}{\delta})$

Can verification be approximated? [P01]

Can verification be approximated? [P01]

Not efficiently! [LP05, LP08]

Theorem [LP08]
If there is a FPRAS for the problem of computing $Prob[\phi]$ for LTL formula ϕ, then $RP = NP$

This is unlikely! non approximability of pLTL

Can verification be approximated? [P01]

Not efficiently! [LP05, LP08]

Theorem [LP08]
If there is a FPRAS for the problem of computing $Prob[\phi]$ for LTL formula ϕ, then $RP = NP$

This is unlikely! non approximability of pLTL
Can we however design an algorithm that works for an expressive fragment of LTL?

Can verification be approximated? [P01]

Not efficiently! [LP05, LP08]

Theorem [LP08]
If there is a FPRAS for the problem of computing $Prob[\phi]$ for LTL formula ϕ, then $RP = NP$

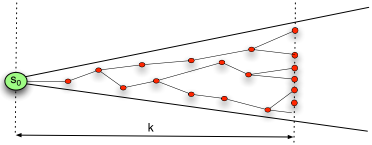
This is unlikely! non approximability of pLTL
Can we however design an algorithm that works for an expressive fragment of LTL?

YES WE CAN

Relaxation on the fragment but also on the type of error

Approximate verification algorithms

Additive error



We consider **bounded properties**
i.e. we approximate $Prob_k(\phi)$

Input: $\mathcal{D}, k, \phi, \varepsilon, \delta$
Output: approx. of $Prob_k[\phi]$

$N := \ln(\frac{2}{\delta})/2\varepsilon^2$
$A := 0$
For $i = 1$ to N do
$B := \big(RandomPath(\mathcal{D}, k) \models \phi\big)$
$A := A + B$
EndFor
Return $Y = A/N$

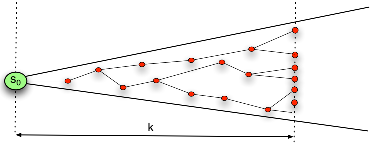
We use a *succinct representation* $\mathcal{D}$ s.t. $|\mathcal{D}| = polylog(|\mathcal{M}|)$

Additive error:
$Pr\big(|Y - prob_k(\phi)| < \varepsilon\big) \ \geq \ 1 - \delta$

Running time is polynomial in $|\mathcal{D}|$ and $|\phi|$

Approximate verification algorithms

Additive error



We consider **bounded properties**
i.e. we approximate $Prob_k(\phi)$

Input: $\mathcal{D}, k, \phi, \varepsilon, \delta$
Output: approx. of $Prob_k[\phi]$
$N := \ln(\frac{2}{\delta})/2\varepsilon^2$
$A := 0$
For $i = 1$ to N do
$B := \big(RandomPath(\mathcal{D}, k) \models \phi\big)$
$A := A + B$
EndFor
Return $Y = A/N$

We use a *succinct representation* $\mathcal{D}$ s.t. $|\mathcal{D}| = polylog(|\mathcal{M}|)$

Additive error:
$Pr\big(|Y - prob_k(\phi)| < \varepsilon\big) \ \geq \ 1 - \delta$

Running time is polynomial in $|\mathcal{D}|$ and $|\phi|$

This is a FPRAS for $Prob_k(\phi)$ [LP02, HLMP04]

Approximate verification algorithms

Additive error cont'd

Extension to more expressive subset [HLMP04, LP05, LP08]

- Monotone formulas (this includes reachability)
  - if true on a path at depth k, still true for $k' > k$
- Essentially Positive Fragment (EPF)
  - negation in front of atomic propositions only
  - EPF formulas are monotone
- Anti monotone formulas through negation (this includes safety)

How to proceed?

Approximate verification algorithms

Additive error cont'd

Extension to more expressive subset [HLMP04, LP05, LP08]
How to proceed?

Iterate the previous algorithm until k is large enough

Approximate verification algorithms

Additive error cont'd

Extension to more expressive subset [HLMP04, LP05, LP08]
How to proceed?

Iterate the previous algorithm until k is large enough
How large?

k **is such that** $O(k^{m_2-1}|\lambda_2|^k) \leq \varepsilon$ (Perron-Frobenius)
where m_2 is the multiplicity of λ_2, second eigenvalue of P

Approximate verification algorithms

Additive error cont'd

Extension to more expressive subset [HLMP04, LP05, LP08]
How to proceed?

Iterate the previous algorithm until k is large enough
How large?

k is such that $O(k^{m_2-1}|\lambda_2|^k) \leq \varepsilon$ (Perron-Frobenius)
where m_2 is the multiplicity of λ_2, second eigenvalue of P

This is a RAS for $Prob(\phi)$, logspace complexity

Approximate verification algorithms

Additive error cont'd

previous bound is, in practice, intractable [LP08]

Need of a practical stopping criterion

Approximate verification algorithms

Additive error cont'd

previous bound is, in practice, intractable [LP08]

Need of a practical stopping criterion

- Let $p = Prob[\psi] = lim_{k \to \infty} Prob_k[\psi]$
- ψ is a monotone formula, so $q = 1 - p = Prob[\neg\psi]$ can be approximated by a slight modification of the MC^2 algorithm (Grosu & Smolka 05)
- Use concurrently the two algorithms
- If $|\hat{p}_k - (1 - \hat{q}_l)| \leq \varepsilon/3$ then $\hat{p}_k$ is an ε-approximation of p

Approximate verification algorithms

Additive error cont'd

Extension to CTMCs [HLP06]

- CTMC $= (S, I, R, L)$
  - S set of states, I initial states, $R : S \times S \to \mathbb{R}_+$ rate matrix, and L labelling function
- Rate of transition from a state $s \in S$: $\lambda(s) = \sum_{s' \in S} R(s, s')$

Probability of the transition from $s \to s'$ within t time units:

$$Prob(s \to s') = \frac{R(s, s')}{\lambda(s)} \left(1 - e^{-t \cdot \lambda(s)}\right)$$

Extension to CTMCs [HLP06]

- CTMC $= (S, I, R, L)$
  - S set of states, I initial states, $R : S \times S \rightarrow \mathbb{R}_+$ rate matrix, and L labelling function
- Rate of transition from a state $s \in S$: $\lambda(s) = \sum_{s' \in S} R(s, s')$

Probability of the transition from $s \rightarrow s'$ within t time units:

$$Prob(s \rightarrow s') = \frac{R(s, s')}{\lambda(s)}\left(1 - e^{-t \cdot \lambda(s)}\right)$$

Same algorithm as for DTMCs, only a modification of the path generation process:

- **Choose state** j **with probability** $P(i, j) = R(i, j)/\lambda(i)$
- $s := j$ **and** $t := t - \ln(random_{[0,1]}/R(i, j))$

Randomized Approximation Scheme with multiplicative error

- RAS with relative error for DTMC
- Prob. version of the algorithm of Grosu & Smolka 2005
- Not fully polynomial even for bounded properties
- Theoretical foundations coming from Karp, Luby and Madras 1989, and Dagum, Karp, Luby and Sheldon 2000.

If the random variables $X_1, X_2, \ldots, X_N$ are iid according to X, $S = \sum_{i=1}^{N} X_i$, and $N = 4(e-2).\ln(\frac{2}{\delta}).\rho/(\epsilon.\mu)^2$, then:

$$Pr\big(\mu(1-\epsilon) \leq S/N \leq \mu(1+\epsilon)\big) \geq 1-\delta$$

where $\rho = max(\sigma^2, \epsilon\mu)$ is a parameter used to optimize the number N of experiments and σ^2 is the variance of X.

Approximate verification algorithms

Multiplicative error

Randomized Approximation Scheme with multiplicative error

1. Output an (ϵ, δ)-approximation $\hat{\mu}$ of μ after expected number of experiments proportional to $4(e-2).\ln(\frac{2}{\delta})/\epsilon^2\mu$
2. Use $\hat{\mu}$ to set the number of experiments to produce $\hat{\rho}$ that is within a constant factor of ρ with probability at least $(1-\delta)$,
3. Use $\hat{\mu}$ and $\hat{\rho}$ to set the number of experiments and runs the experiments to produce an (ϵ, δ)-approximation of μ.

RAS (with multiplicative error) for computing $p = Prob_k[\phi]$

Not an *FPRAS* as the expected number of experiments can be exponential for small values of μ

Approximate verification algorithms

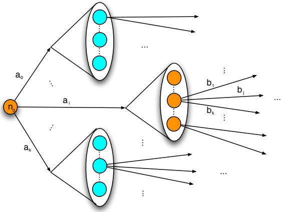
Summary

	DTMC		
	Bounded	**Monotone**	**LTL**
additive error	✓ FPRAS	✓ RAS	✓ RAS
multiplicative error	✓ RAS	✓ RAS	✗ FPRAS $\Downarrow$ RP=NP

- Extension to CTMCs
- Practical stopping criterion

Perspectives: approx. verification for MDPs with rewards

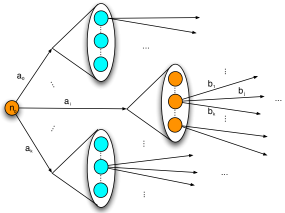
Statement of the problem



- $\mathcal{M} = (S, A, P, R)$
  - S: states
  - A: set of actions
  - $P(s, a)$: transition probabilities from s under action a
  - $R(s, a)$: reward for executing action a from state s

Perspectives: approx. verification for MDPs with rewards

Statement of the problem



- $\mathcal{M} = (S, A, P, R)$
  - S: states
  - A: set of actions
  - $P(s, a)$: transition probabilities from s under action a
  - $R(s, a)$: reward for executing action a from state s

Actions are chosen by adversaries

Choices of an adversary define a strategy

One want to minimize/maximize the total reward given by a strategy

We want to approximate the probability of a LTL behavior ϕ under best (and worst) strategy π

$$Prob_\pi(\phi)$$

Perspectives: approx. verification for MDPs with rewards

- Usually, randomized (or simple) adversaries are considered
- **Finding a quasi-optimal strategy can be done through sampling algorithm for all mighty adversaries**
- Seems to be hard (or most?) for randomized adversaries
- Seems to apply (*i.e.* to have a meaning) only for properties on rewards
- What about infinite MDPs?

Approximation?

Random walks versus uniform generation

Approximation for non probabilistic systems

Random walks in probabilistic systems use local random choices, this is legit since these choices are truly in the system

Approximation?

Random walks versus uniform generation

Approximation for non probabilistic systems

Random walks in probabilistic systems use local random choices, this is legit since these choices are truly in the system
In testing, model checking or simulation of non probabilistic systems, isotropic exploration is often used

Local choices from a node are made uniformly according to the out degree of the node

Approximation?

Random walks versus uniform generation

Approximation for non probabilistic systems

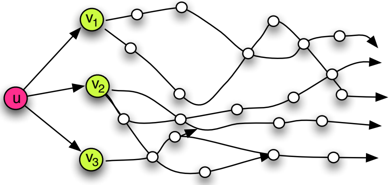
Random walks in probabilistic systems use local random choices, this is legit since these choices are truly in the system
In testing, model checking or simulation of non probabilistic systems, isotropic exploration is often used

Local choices from a node are made uniformly according to the out degree of the node

The topology has an impact on the coverage, this is BAD
Need for an uniform exploration!

Uniform generation of paths

Uniform generation of paths of fixed length in a graph [DGGLP06, GDGLOP08]



$$l_u(k) = \#paths_k(u)$$

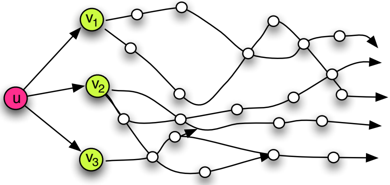
$paths_k(u)$: length k, starting at u

Condition for path uniformity:

$$Prob(choose\ v_i) = \frac{l_{v_i}(k-1)}{l_u(k)}$$

Uniform generation of paths

Uniform generation of paths of fixed length in a graph [DGGLP06, GDGLOP08]



$$l_u(k) = \#paths_k(u)$$

$paths_k(u)$: length k, starting at u

Condition for path uniformity:

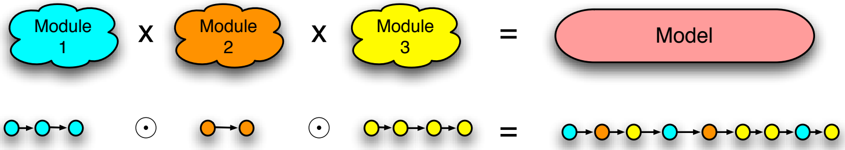
$$Prob(choose\ v_i) = \frac{l_{v_i}(k-1)}{l_u(k)}$$

This brute force method works for small models
Computing all these prob. is too expensive for large models!

Uniform generation of paths

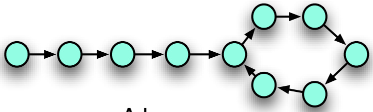
compositional approach

- Large models are constructed through a concurrent composition of modules
- Use uniform generation of paths in modules to generate almost uniformly paths in the model



- Lot of technicalities: how to choose the length of subpaths, how to shuffle, etc. **But it works!**
- Other methods and improvements: Oudinet PhD thesis (2010)

Counting and Generating Lassos in Directed Graphs [ODGLP]



A lasso

Problems:

• Counting elementary cycles is not easy unless $P = NP$

• Finding all elementary cycles: no polynomial time algorithm

- **The fundamental problem is still the state space explosion**
- **Seems to be feasible for reducible flowgraphs**
- **Extension to broader class of graphs is not trivial**

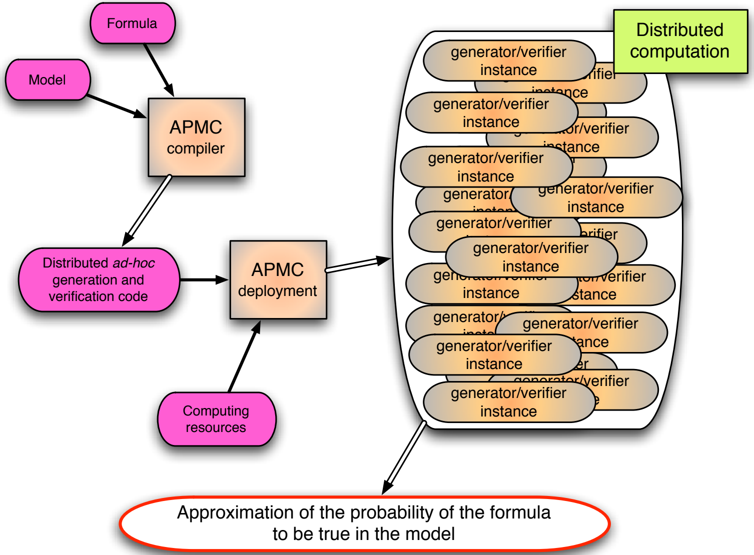
APMC

Approximate Probabilistic Model Checker (APMC)

- Implement techniques previously presented today
- First prototype by Herault, Lassaigne, Magniette and Peyronnet **[HLMP04]**
- First scalable version by Guirado, Herault, Lassaigne and Peyronnet **[GHLP05]**
- Since then, a real team work (around 20 collaborators) for development and case studies

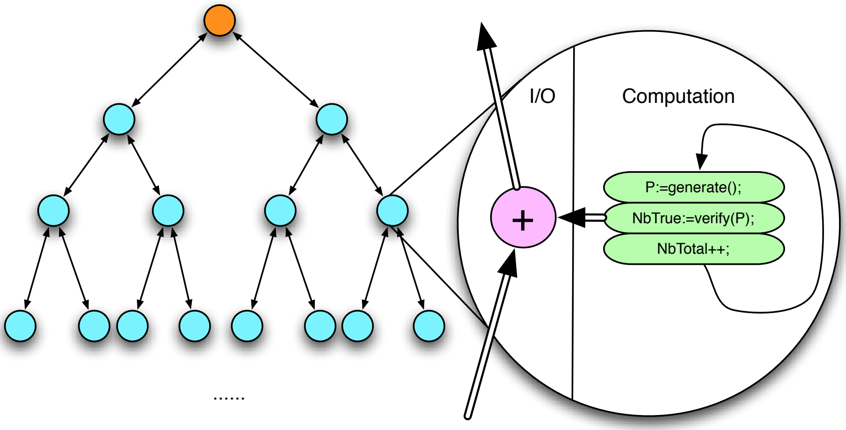
APMC

Architecture



APMC

Distributed architecture, from the past...



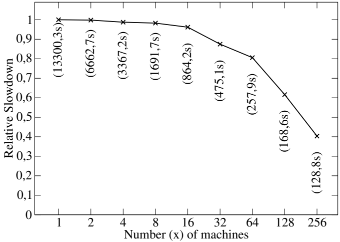
APMC

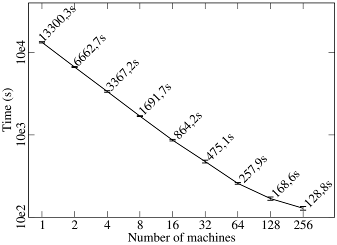
Distributed architecture, from the past...

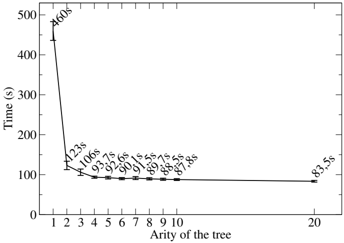
Old experiments on an old platform **[GHLP05]**

Don't look at the numbers, look only at the behavior!

Model is the 160 dining philosophers







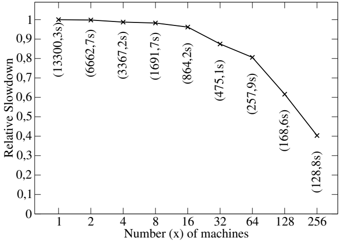
APMC

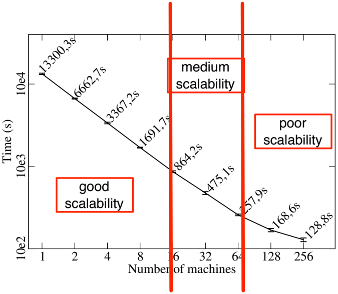
Distributed architecture, from the past...

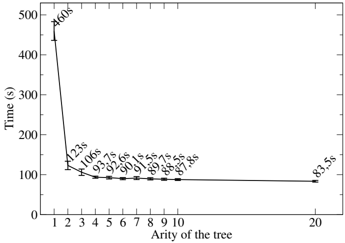
Old experiments on an old platform **[GHLP05]**

Don't look at the numbers, look only at the behavior!

Model is the 160 dining philosophers







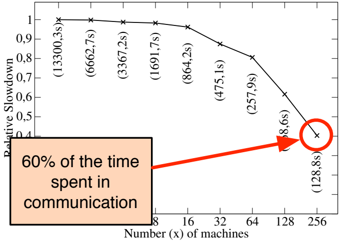
APMC

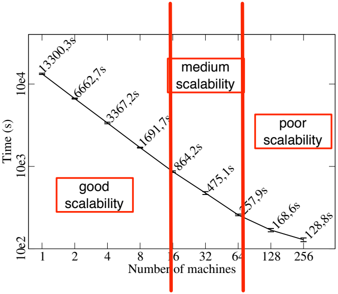
Distributed architecture, from the past...

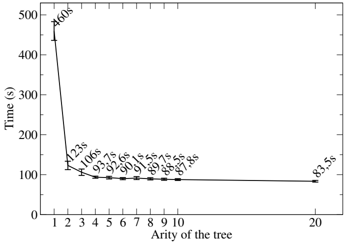
Old experiments on an old platform **[GHLP05]**

Don't look at the numbers, look only at the behavior!

Model is the 160 dining philosophers







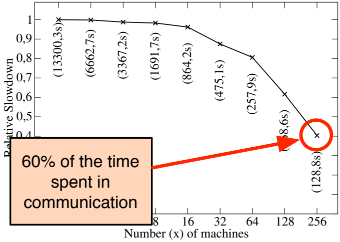
APMC

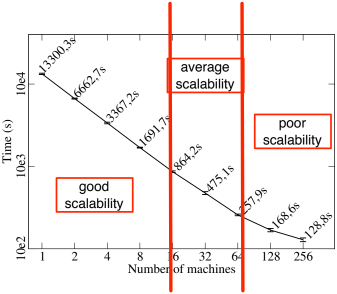
Distributed architecture, from the past...

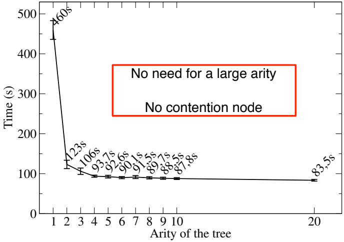
Old experiments on an old platform **[GHLP05]**

Don't look at the numbers, look only at the behavior!

Model is the 160 dining philosophers



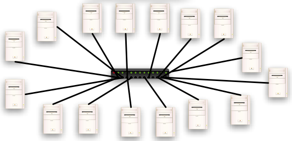




APMC

...to the future: architecture-driven parallel APMC

Cluster

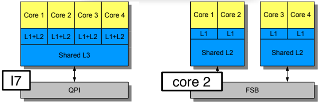


Machines connected through a network

Typical example: clusters of GRID5000

Communication using messages

SMP / multi-core



Processors or cores of the same machine

Typical example: most commercial processor

Communication using shared memory

Hybrid

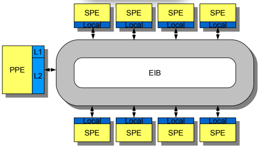
Cluster of SMP / multi-core machines

Typical example: clusters of GRID5000

Communication using messages between machines

Communication using shared memory between cores

CELL



Vector computing cores (SPEs), only 256 KB

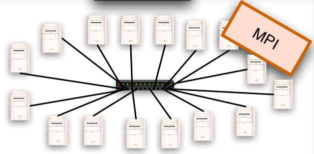
The PPE controls the SPEs

PS3 Cell theoretical peak performance:
150 GFLOPS (150% of a 3GHz Intel quad-core)

APMC

...to the future: architecture-driven parallel APMC

Cluster

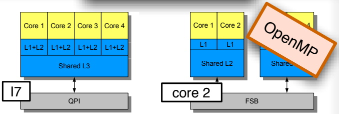


Machines connected through a network

Typical example: clusters of GRID5000

Communication using messages

SMP / multi-core



Processors or cores of the same machine

Typical example: most commercial processor

Communication using shared memory

Hybrid

MPI + OpenMP

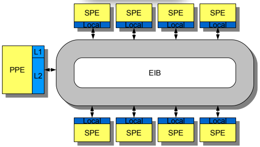
Cluster of SMP / multi-core machin

Typical example: clusters of GRID5000

Communication using messages between machines

Communication using shared memory between cores

CELL



Vector computing cores (SPEs), only 256 KB

The PPE controls the SPEs

PS3 Cell theoretical peak performance:
150 GFLOPS (150% of a 3GHz Intel quad-core)

APMC

...to the future: architecture-driven parallel APMC

MPI

- The parallelization scales perfectly
- No overhead in the use of BSP++
- A small slowdown due to the synchronization time that rises with the number of cores when used on a SMP machine

OpenMP

- The parallelization scales perfectly
- No overhead in the use of BSP++
- OpenMP slightly better than MPI on SMP machines (no surprise here)

Cluster, SMP and Hybrid obtained using BSP++ [HBEFP10]

Hybrid

- The parallelization scales perfectly
- No overhead in the use of BSP++
- Outperforms the MPI version by 50% to 10% depending on the models (in fact it depends on the ratio computation/communication)

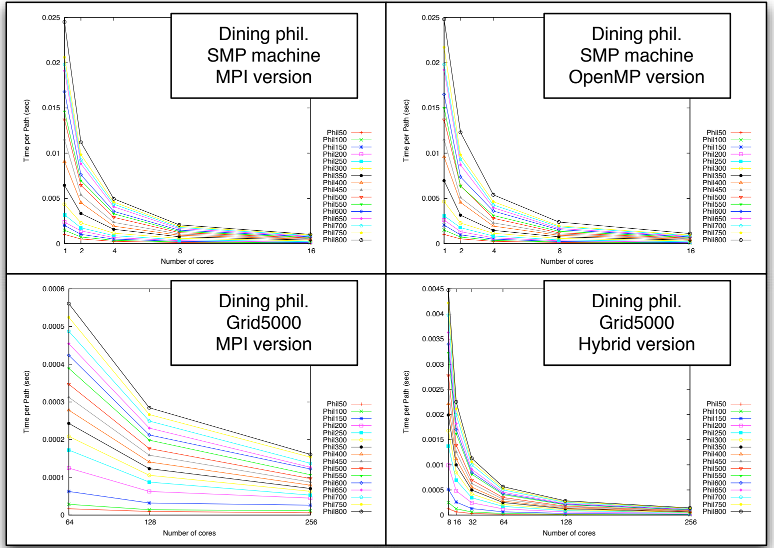
CELL

APMC CELL-ASSISTED [BHLP08]

- Does not scale so hit the memory wall!
- Not compatible with CTMCs
- Does not handle synchronization

APMC

...to the future: architecture-driven parallel APMC



APMC

...to the future: architecture-driven parallel APMC

Architecture-driven parallel APMC

The best strategy for building a parallel version of a sampling based probabilistic model checker is to use an hybrid architecture with an hybrid version of the code

The overhead due to the use of an automatic parallelization framework (here BSP++) is close to zero

APMC

The problem of the input

Can we use a more concise and user-friendly input language?

- Currently APMC uses PRISM input language
  - Not scriptable
  - Not flexible (What if some modules are a little bit different)
    - variable renaming inappropriate
    - code duplication — error prone
- APMC is not bound by the memory wall, the language can be fancy and more expressive
- We want to avoid scripting since it requires extra skills

eXtended Reactive Modules (XRM) [DSP06]

- an extended version of the PRISM language.

eXtended Reactive Modules (XRM) [DSP06]

- an extended version of the PRISM language featuring:
  - For loops
  - If statements
  - Functions to factor code
  - Built-in functions
  - Parametric and recursive formulas

eXtended Reactive Modules (XRM) [DSP06]

- an extended version of the PRISM language featuring:
  - For loops
  - If statements
  - Functions to factor code
  - Built-in functions
  - Parametric and recursive formulas
- a compiler generates PRISM language
  - Consistency of the generated code is ensured by the compiler
  - Type-checking is possible

APMC

eXtended Reactive Modules (XRM)

Conditional definition of a module

```
// Event location.
const int event_x = 5, event_y = 5;

for x from 0 to width − 1 do
  for y from 0 to height − 1 do
    module sensor[x][y]
      if x = event_x & y = event_y then
        // Broadcasting
      else
        // Listening
      end
    end module
  end
end
```

APMC

eXtended Reactive Modules (XRM)

Conditional definition of a module

```
// Event location.
const int event_x = 5, event_y = 5;

for x from 0 to width - 1 do
  for y from 0 to height - 1 do
    module sensor[x][y]
      if x = event_x & y = event_y then
       // Broadcasting
      else


    en
  end
end
```

Parametric Formulas

```
formula consume (int value) =
  battery' = battery < value ? 0 : battery - value;
```

Results on a case study [DSP06 versus DHP06]

Test on a simple model: basic sensor network

	DHP06	DSP06
Tools	Shell M4/m4sugar	XRM XPCTL
Size of the description	264 lines of M4 247 lines of Shell script	87 lines of XRM 12 lines of XPCTL
Size of the generated PRISM model	1346 lines of PRISM language 25 lines of PCTL	941 lines of PRISM language 25 lines of PCTL

Can we do something even more user-friendly?

- APMC works on top of a simulator
  - If we can control totally a real system, we don't need any input language
- VD-S is a virtualization platform that allows for the total control of any application running on top of it **[HLPQCJ09]**
- It can be used as a path generator for the APMC engine
- Adversaries can be defined externally to interact with the system under verification

New architectures?

- GPU (unlikely to be efficient)
- FPGA (as accelerator)
- CELL (as accelerator)
- etc.

Conclusion

- Complexity, lower bounds and efficient algorithms for approximate verification
- Efficient implementation showing good performances on huge systems
- Several technical improvements, including the consideration of the machines architecture

A lot of exciting perspectives, with many collaborators!