Modeling of Sensor Networks Using XRM

http://projects.lrde.epita.fr/Xrm

Akim Demaille Benoît Sigoure Sylvain Peyronnet
<FirstName.Name@lrde.epita.fr>

EPITA Research and Development Laboratory (LRDE)

Nov. 17th, 2006

Modeling of Sensor Networks Using XRM

Motivation

APMC

Randomized Approximation Scheme

$$p = \mathrm{Prob}\left[\psi\right]$$

$$\mathrm{Prob}\Big[(p-\varepsilon) \leq A \leq (p+\varepsilon)\Big] \geq 1-\delta$$

Definition (FPRAS)

Fully polynomial randomized approximation scheme with time complexity $\mathrm{poly}(\log(1/\delta), |\psi|, (1/\varepsilon))$

Randomized Approximation Scheme

$$p = \mathrm{Prob}\,[\psi]$$

$$\mathrm{Prob}\Big[\,(p - \varepsilon) \leq A \leq (p + \varepsilon)\,\Big] \geq 1 - \delta$$

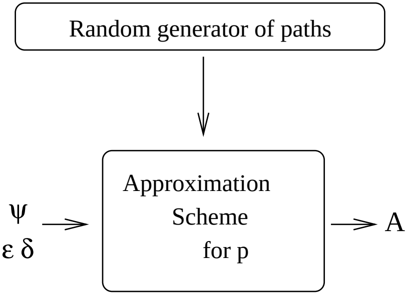
Definition (FPRAS)

Fully polynomial randomized approximation scheme with time complexity poly(log(1/δ), | ψ |, (1/ε))

Randomized Approximation Scheme

$$p = \mathrm{Prob}\,[\psi]$$

$$\mathrm{Prob}\Big[\,(p-\varepsilon) \leq A \leq (p+\varepsilon)\,\Big] \geq 1-\delta$$



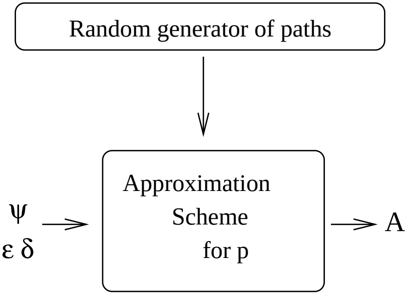
Definition (FPRAS)

Fully polynomial randomized approximation scheme with time complexity $\mathrm{poly}(\log(1/\delta), |\,\psi\,|, (1/\varepsilon))$

Randomized Approximation Scheme

$$p = \mathrm{Prob}\,[\psi]$$

$$\mathrm{Prob}\Big[\,(p-\varepsilon) \leq A \leq (p+\varepsilon)\,\Big] \geq 1-\delta$$



Definition (FPRAS)

Fully polynomial randomized approximation scheme with time complexity $\mathrm{poly}(\log(1/\delta), |\,\psi\,|, (1/\varepsilon))$

Our algorithm

Algorithm (Generic approximation)

input: ψ, *diagram*, ϵ, δ
Let $P := 0$
Let $N := \frac{1}{2}\log(\frac{2}{\delta})/\epsilon^2$
For i from 1 *to N do*

1. *Generate a random path* σ *of depth k*
2. *If* ψ *is true on* σ *then* $P := P + 1$

Return $A = P/N$

Theorem

This algorithm is an additive FPRAS for $\mathrm{Prob}[\psi]$

Our algorithm

Algorithm (Generic approximation)

input: ψ, *diagram*, ϵ, δ
Let P $:= 0$
Let N $:= \frac{1}{2}\log(\frac{2}{\delta})/\epsilon^2$
For i from 1 *to N do*

1. *Generate a random path* σ *of depth k*
2. *If* ψ *is true on* σ *then* $P := P + 1$

Return $A = P/N$

Theorem

This algorithm is an additive FPRAS for $\mathrm{Prob}[\psi]$

Approximate Probabilistic Model Checker

APMC [Hérault et al., 2004]

- model checker
- probabilistic systems
- approximate
- resilient to state explosion
- distributed
- hence well suited for large models such as sensor networks

Approximate Probabilistic Model Checker

APMC [Hérault et al., 2004]

- model checker
- probabilistic systems
- approximate
- resilient to state explosion
- distributed
- hence well suited for large models such as sensor networks

PRISM and Reactive Modules

Model-checking, (Reactive) Modules and PRISM

APMC uses PRISM's parser

PRISM
: is a probabilistic model checker [Kwiatkowska et al., 2002]
  - PRISM language
  - Based on Reactive Modules' syntax
  - Widely used

Reactive Modules
: is a formalism [Alur and Henzinger, 1996]
  - Concurrent systems description
  - Model-checking

Model-checking, (Reactive) Modules and PRISM

APMC uses PRISM's parser

PRISM is a probabilistic model checker [Kwiatkowska et al., 2002]

- PRISM language
- Based on Reactive Modules' syntax
- Widely used

Reactive Modules is a formalism [Alur and Henzinger, 1996]

- Concurrent systems description
- Model-checking

The PRISM language

Probabilistic transition system

```
module process
  x1 : [0..1];
  []  x1=x5 -> 0.5 : (x1'=0) + 0.5 : (x1'=1);
  [] !x1=x5 -> (x1'=x5);
endmodule
```

The PRISM language

Module renaming

```
module process1
  x1 : [0..1];
  [] x1=x5 -> 0.5 : (x1'=0) + 0.5 : (x1'=1);
  [] !x1=x5 -> (x1'=x5);
endmodule

// Add further processes through renaming.
module process2 = process1[x1=x2, x5=x1] endmodule
module process3 = process1[x1=x3, x5=x2] endmodule
module process4 = process1[x1=x4, x5=x3] endmodule
module process5 = process1[x1=x5, x5=x4] endmodule
```

Several limitations

- Does not scale
  $N = 100$
- Not scriptable
  $N \in \{1, 2, 3, 5, 10, 15, 100, 1000\}$
- Not flexible
  What if some modules are different
  - variable renaming inappropriate
  - code duplication — error prone

Several limitations

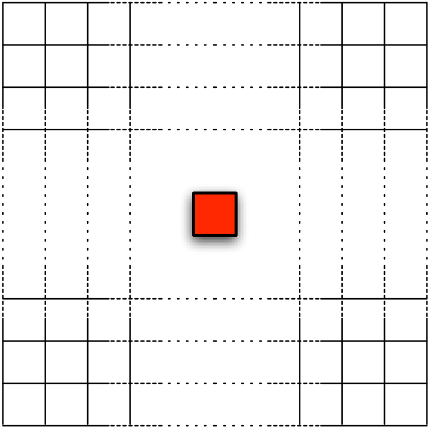
- Does not scale
  $N = 100$
- Not scriptable
  $N \in \{1, 2, 3, 5, 10, 15, 100, 1000\}$
- Not flexible
  What if some modules are different
  - variable renaming inappropriate
  - code duplication — error prone

Several limitations

- Does not scale
  $N = 100$
- Not scriptable
  $N \in \{1, 2, 3, 5, 10, 15, 100, 1000\}$
- Not flexible
  What if some modules are different
  - variable renaming inappropriate
  - code duplication — error prone

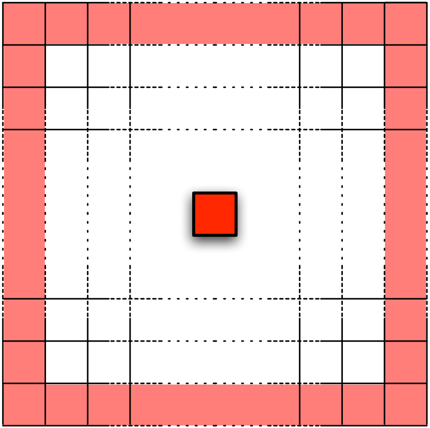
A simplistic sensor network

Sensor networks



The sensor in the middle broadcasts the alert.
Its code must be different.

Sensor networks



The neighborhoods on the edges are different. Their code for sensing alerts is different.

Possible solution

Generate PRISM code with scripts

- Use shell/M4/Ruby/Perl/Python/FooBar scripts
- You need to know a scripting language
- Waste of time
- Bugs in your script will be hard to debug
- Your attention is distracted from your first objective
- No standard

Or extend the language...

Possible solution

Generate PRISM code with scripts

- Use shell/M4/Ruby/Perl/Python/FooBar scripts
- You need to know a scripting language
- Waste of time
- Bugs in your script will be hard to debug
- Your attention is distracted from your first objective
- No standard

Or extend the language...

Possible solution

Generate PRISM code with scripts

- Use shell/M4/Ruby/Perl/Python/FooBar scripts
- You need to know a scripting language
- Waste of time
- Bugs in your script will be hard to debug
- Your attention is distracted from your first objective
- No standard

Or extend the language...

Possible solution

Generate PRISM code with scripts

- Use shell/M4/Ruby/Perl/Python/FooBar scripts
- You need to know a scripting language
- Waste of time
- Bugs in your script will be hard to debug
- Your attention is distracted from your first objective
- No standard

Or extend the language...

Possible solution

Generate PRISM code with scripts

- Use shell/M4/Ruby/Perl/Python/FooBar scripts
- You need to know a scripting language
- Waste of time
- Bugs in your script will be hard to debug
- Your attention is distracted from your first objective
- No standard

Or extend the language...

Possible solution

Generate PRISM code with scripts

- Use shell/M4/Ruby/Perl/Python/FooBar scripts
- You need to know a scripting language
- Waste of time
- Bugs in your script will be hard to debug
- Your attention is distracted from your first objective
- No standard

Or extend the language...

Possible solution

Generate PRISM code with scripts

- Use shell/M4/Ruby/Perl/Python/FooBar scripts
- You need to know a scripting language
- Waste of time
- Bugs in your script will be hard to debug
- Your attention is distracted from your first objective
- No standard

Or extend the language...

eXtended Reactive Modules

eXtended Reactive Modules

- an extended version of the PRISM language.
  - For loops
  - If statements
  - Functions to factor code
- a compiler generates PRISM
  - Consistency of the generated code is ensured by the compiler
  - Type-checking is possible
  - Meta-programming: code partially generated and evaluated at compile time

eXtended Reactive Modules

- an extended version of the PRISM language featuring:
  - For loops
  - If statements
  - Functions to factor code
- a compiler generates PRISM
  - Consistency of the generated code is ensured by the compiler
  - Type-checking is possible
  - Meta-programming: code partially generated and evaluated at compile time

eXtended Reactive Modules

- an extended version of the PRISM language featuring:
  - For loops
  - If statements
  - Functions to factor code
- a compiler generates PRISM
  - Consistency of the generated code is ensured by the compiler
  - Type-checking is possible
  - Meta-programming: code partially generated and evaluated at compile time

eXtended Reactive Modules

- an extended version of the PRISM language featuring:
  - For loops at the meta-level
  - If statements at the meta-level
  - Functions to factor code at the meta-level
- a compiler generates PRISM
  - Consistency of the generated code is ensured by the compiler
  - Type-checking is possible
  - Meta-programming: code partially generated and evaluated at compile time

The package

1. Motivation
2. eXtended Reactive Modules
   - The package
   - Features
   - Sensor Networks
3. Conclusion and perspectives

eXtended Reactive Modules

Built thanks to Stratego/XT [Visser, 2004]

Stratego a language designed for program transformations

SDF Syntax Definition Formalism
Modular definitions make it easy to:

- Extend grammars
- Embed a grammar into another

SGLR Scannerless Generalized LR parser

- Parser generator
- Supports ambiguities
- Provides several disambiguation filters

eXtended Reactive Modules

Built thanks to Stratego/XT [Visser, 2004]

Stratego a language designed for program transformations

SDF Syntax Definition Formalism
Modular definitions make it easy to:

- Extend grammars
- Embed a grammar into another

SGLR Scannerless Generalized LR parser

- Parser generator
- Supports ambiguities
- Provides several disambiguation filters

eXtended Reactive Modules

Built thanks to Stratego/XT [Visser, 2004]

Stratego a language designed for program transformations

SDF Syntax Definition Formalism
Modular definitions make it easy to:

- Extend grammars
- Embed a grammar into another

SGLR Scannerless Generalized LR parser

- Parser generator
- Supports ambiguities
- Provides several disambiguation filters

Tools

- 4 parsers

PRISM	
XRM	extended PRISM
PCTL	properties to model-check
XPCTL	PCTL extended with XRM embeddings

- 4 pretty-printers
- 1 compiler: xrm-front
  - XRM → PRISM
  - XPCTL → PCTL

Tools

- 4 parsers

PRISM	
XRM	extended PRISM
PCTL	properties to model-check
XPCTL	PCTL extended with XRM embeddings

- 4 pretty-printers
- 1 compiler: xrm-front
  - XRM → PRISM
  - XPCTL → PCTL

Tools

- 4 parsers

PRISM	
XRM	extended PRISM
PCTL	properties to model-check
XPCTL	PCTL extended with XRM embeddings

- 4 pretty-printers
- 1 compiler: xrm-front
  - XRM → PRISM
  - XPCTL → PCTL

Features

Meta-For loops (1/2)

Meta-for loops help describing large systems.

Writing sensor networks with XRM

```
const int width = 100;
const int height = 100;

for x from 0 to width - 1 do
  for y from 0 to height - 1 do
    module sensor[x][y]
      status[x][y] : [0..MAX_STATE] init SENSE;
      // Commands of the module go here.
    end module
  end
end
```

Meta-For loops (1/2)

For loops are unrolled by xrm-front.

Writing sensor networks with XRM

```
const int width = 100;
const int height = 100;

for x from 0 to width - 1 do
  for y from 0 to height - 1 do
    module sensor[x][y]
      status[x][y] : [0..MAX_STATE] init SENSE;
      // Commands of the module go here.
    end module
  end
end
```

Meta-For loops (2/2)

Meta foreach loops.

Shell-like meta-for loop

```
module xrm
  x : [0..1]  init 0;
  y : [0..10] init 0;
  z : [0..1]  init 0;
  for i in x, 1+2, y do
    [] y=i -> y' = y+1;
  end
end module
```

Meta-If statements

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
```

Arrays

- Large modules require many variables
- Multi-dimensional array declarations
- Subscripts must statically evaluate to positive integers

XRM Arrays

```
const int N = 4, M = 2;

module foo
  // multi-dimensional "sparse" array
  x[0..10][0,2,5..7] : [0..1] init 0;

  [] x[N][M]=0 -> (x[N][M]'=1);
end module
```

Arrays

- Large modules require many variables
- Multi-dimensional array declarations
- Subscripts must statically evaluate to positive integers

XRM Arrays

```
const int N = 4, M = 2;

module foo
  // multi-dimensional "sparse" array
  x[0..10][0,2,5..7] : [0..1] init 0;

  [] x[N][M]=0 -> (x[N][M]'=1);
end module
```

Arrays

- Large modules require many variables
- Multi-dimensional array declarations
- Subscripts must statically evaluate to positive integers

XRM Arrays

```
const int N = 4, M = 2;

module foo
  // multi-dimensional "sparse" array
  x[0..10][0,2,5..7] : [0..1] init 0;

  [] x[N][M]=0 -> (x[N][M]'=1);
end module
```

Builtins

Random values

builtins

```
module sample
  x : [0..51] init 0;
  [] true -> x'=static_rand(42);
  [] true -> x'=rand(42);
end module
```

Builtins

Random values

Generated code

```
module sample
  x : [0..51] init 0;
  [] true -> x'=<random value>;
  [] true -> x'=__rand_0;
endmodule
module __rand_0
  __rand_0 : [0..42];
  [] true -> 1/43:(__rand_0'=0) + 1/43:(__rand_0'=1) +
             1/43:(__rand_0'=2) + ...
             ... + 1/43:(__rand_0'=42);
endmodule
```

Formulas

Extentions

Parametric Formulas

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

Recursive Formulas

```
formula fact (int n) = n <= 1 ? 1 : n * fact (n - 1);
```

Formulas

Extentions

Parametric Formulas

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

Recursive Formulas

```
formula fact (int n) = n <= 1 ? 1 : n * fact (n - 1);
```

Formulas

Extentions

Statements in Formulas

```
formula tick = t' = t + 1;
```

Macros

```
formula incr (exp var) = var' = var + 1;
```

Formulas

Extentions

Statements in Formulas

```
formula tick = t' = t + 1;
```

Macros

```
formula incr (exp var) = var' = var + 1;
```

eXtended PCTL

PCTL Probabilistic Computational Tree Logic.
Properties specifications.

XPCTL = PCTL + XRM extensions.

- Meta-code
- Arrays
- Parameterized formulas
- Possibly embedded in the XRM sources

Sensor Networks

Basic formulas

Earing the neighbors

```
// Whether (x, y) are valid sensor coordinates.
formula valid (int x, int y) =
  0 <= x & x < X & 0 <= y & y < Y;

// Whether (x, y) is valid, and broadcasting.
formula broadcasts (int x, int y) =
  valid (x, y) & s[x][y] = BROADCASTS;

// Whether a neighbor of (x, y) is broadcasting.
formula ears (int x, int y) =
    broadcasts (x - 1, y) | broadcasts (x + 1, y)
  | broadcasts (x, y - 1) | broadcasts (x, y + 1);
```

Basic formulas

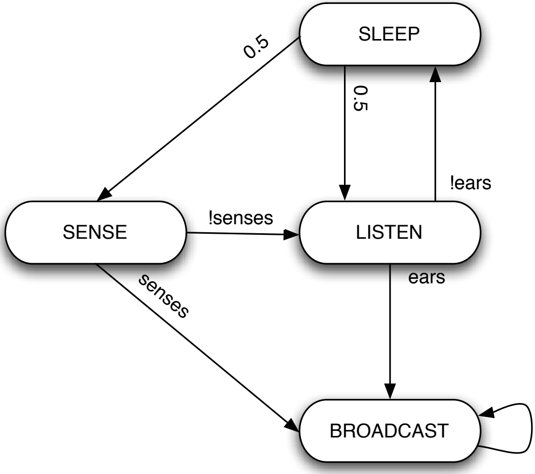
Consuming

```
// Consume c units of power.
formula consume (int x, int y, int c) =
  b[x][y]' = b[x][y] < c ? 0 : b[x][y] - c;

// Reach state st if energy allows it.
formula set_state (int x, int y, int st) =
  s[x][y]' = (0 <= b[x][y]) ? st : OFF;

// Transition to the next state st.
formula transition (int x, int y, int st) =
  t' = t+1
  & consume(x, y, cost (s[x][y]))
  & set_state(x, y, st);
```

Node behavior



Node implementation

Embedded in two for loops

```
module sensor[x][y]

  s[x][y] : [0..4] init SENSE;
  b[x][y] : [0..POWER] init static_rand (0, POWER);

  [] s[x][y] = SEN
     -> 1:   transition(x, y, senses(x, y) ? BRO : LIS);
  [] s[x][y] = LIS
     -> 1:   transition(x, y, ears(x, y)   ? BRO : SLE);
  [] s[x][y] = SLE
     -> 0.5: transition(x, y, SEN)
      + 0.5: transition(x, y, LIS);
  [] s[x][y] = BRO
     -> 1:   transition(x, y, BRO);
end
```

Properties

Embedded Properties

They share all the formulas and constants etc.

```
properties
  for T from 0 to T_max step 100 do
    // Alarm triggered before instant T?
    P =? [ true U (t <= T & boundary_broadcasts) ];
  end
end
```

Use of eXtended Reactive Modules

- Shell + M4/m4sugar [Demaille et al., 2006]
  - 264 lines of M4 + 247 lines of Shell script.
  - Generates 1316 lines of PRISM + 25 lines of PCTL.
- eXtended Reactive Modules
  - 87 lines of XRM + 12 lines of XPCTL.
  - Generates 941 lines of PRISM + 25 lines of PCTL.

Use of eXtended Reactive Modules

- Shell + M4/m4sugar [Demaille et al., 2006]
  - 264 lines of M4 + 247 lines of Shell script.
  - Generates 1316 lines of PRISM + 25 lines of PCTL.
- eXtended Reactive Modules
  - 87 lines of XRM + 12 lines of XPCTL.
  - Generates 941 lines of PRISM + 25 lines of PCTL.

Use of eXtended Reactive Modules

- Shell + M4/m4sugar [Demaille et al., 2006]
  - 264 lines of M4 + 247 lines of Shell script.
  - Generates 1316 lines of PRISM + 25 lines of PCTL.
- eXtended Reactive Modules
  - 87 lines of XRM + 12 lines of XPCTL.
  - Generates 941 lines of PRISM + 25 lines of PCTL.

Conclusion and perspectives

Conclusion

- A quite complete and reliable way to model large systems
- Benefits from APMC's ability to handle large systems
- Passes 93% of the 616 tests of its test suite

Future work

- More complex modelings
- Type checking
- Bound checking
- Non-static array accesses
- Modularity through imports
- Optimizations
- C Back-end

Future work

- More complex modelings
- Type checking
- Bound checking
- Non-static array accesses
- Modularity through imports
- Optimizations
- C Back-end

Modeling of Sensor Networks Using XRM

Appendix

4 Appendix

Bibliography I

Alur, R. and Henzinger, T. A. (1996).
Reactive modules.
In *LICS*, pages 207–218.

Demaille, A., Peyronnet, S., and Hérault, T. (2006).
Probabilistic verification of sensor networks.
In *Proceedings of the Fourth IEEE International Conference on Computer Sciences, Research, Innovation and Vision for the Future (RIVF'06)*, Ho Chi Minh City, Vietnam.

Bibliography II

- Hérault, T., Lassaigne, R., Magniette, F., and Peyronnet, S. (2004).
  Approximate probabilistic model checking.
  In Steffen, B. and Levi, G., editors, *VMCAI*, volume 2937 of *Lecture Notes in Computer Science*, pages 73–84. Springer.

- Kwiatkowska, M. Z., Norman, G., and Parker, D. (2002).
  Probabilistic symbolic model checking with PRISM: A hybrid approach.
  In *Tools and Algorithms for Construction and Analysis of Systems*, pages 52–66.

Bibliography III

LRDE — EPITA Research and Developpement Laboratory (2005).
Transformers home page.
http://transformers.lrde.epita.fr.

stratego.
http://www.stratego-language.org.

Visser, E. (2004).
Program transformation with Stratego/XT: Rules, strategies, tools, and systems in StrategoXT-0.9.
In Lengauer, C. et al., editors, *Domain-Specific Program Generation*, volume 3016 of *Lecture Notes in Computer Science*, pages 216–238. Spinger-Verlag.

Bibliography IV

xrm-svn.
https://svn.lrde.epita.fr/svn/xrm/.