Formal verification of complex dynamical systems
An approach via formal abstractions

Alessandro Abate
Department of Computer Science, University of Oxford

Labex DigiCosme
Spring School 2016 on Hybrid Systems
May 10, 2016

[Key references will appear here]
Outline

1. Formal abstractions for verification of complex dynamical systems

2. Formal verification of stochastic hybrid systems
   - Analysis and control synthesis problems
   - Verification via formal abstractions

3. Case study: demand response in energy networks

4. Formal verification of max-plus linear models
   - Analysis and control synthesis problems
   - Verification via formal abstractions
Outline

1. Formal abstractions for verification of complex dynamical systems

2. Formal verification of stochastic hybrid systems
   - Analysis and control synthesis problems
   - Verification via formal abstractions

3. Case study: demand response in energy networks

4. Formal verification of max-plus linear models
   - Analysis and control synthesis problems
   - Verification via formal abstractions
Formal verification in a nutshell

• is a **model-based**, automated, quantitative **verification** technique
• asserts quantitative properties over **given model** of system
• is juxtaposed to **validation** (via simulation, testing)

• industrial impact for **verification** of **protocols, circuits, hardware, software**
Formal verification via **model checking**

- is a *model-based*, automated, quantitative **verification** technique
- asserts quantitative properties over **given model** of system
- is juxtaposed to **validation** (via simulation, testing)

```
model M

property p

model checker:
does M verify p?

yes

no: counterexample
```
Challenge: formal verification of complex models

VS
Challenge: formal verification of complex models

vs

verification via formal abstractions
Formal abstractions for verification of complex models

| concrete complex model | property, specification, cost or reward |

A. Abate, Oxford
Formal abstractions for verification of complex models

\[ \varepsilon \text{-quantitative abstraction} \]

| concrete complex model | property, specification, cost or reward |

---

A. Abate, Oxford
Formal abstractions for verification of complex models

abstract simple model

ε-specification

ε-quantitative abstraction

concrete complex model

property, specification, cost or reward
Formal abstractions for verification of complex models

Abstract simple model $\epsilon$-specification

$\epsilon$-quantitative abstraction

Concrete complex model property, specification, cost or reward

Automatic verification control synthesis
Formal abstractions for verification of complex models

abstract simple model

$\epsilon$-specification

model checking

automatic verification

control synthesis

$\epsilon$-quantitative abstraction

concrete complex model

property, specification, cost or reward
Formal abstractions for verification of complex models

abstract simple model

$\epsilon$-specification

$\epsilon$-quantitative abstraction

model checking

automatic verification

control synthesis

$\epsilon$-spec holds yes/no

policy $\mu_\epsilon \rightarrow \epsilon$-spec

concrete complex model

property, specification, cost or reward

A. Abate, Oxford
Formal abstractions for verification of complex models

- abstract simple model
- \( \epsilon \)-specification

\( \epsilon \)-quantitative abstraction

- concrete complex model
  - property, specification, cost or reward

\( \epsilon \)-spec holds yes/no policy \( \mu_\epsilon \rightarrow \epsilon \)-spec

Refine back
Formal abstractions for verification of complex models

- **Abstract simple model**
  - $\epsilon$-specification

- **Model checking**
  - Automatic verification
  - Control synthesis

- **$\epsilon$-spec holds yes/no policy $\mu_\epsilon \rightarrow \epsilon$-spec**

- **$\epsilon$-quantitative abstraction**

- **Concrete complex model**
  - Property, specification, cost or reward

- **Spec holds yes/no policy $\mu \rightarrow \text{spec}$ (correct by design)**
Formal abstractions for verification of complex models

Abstract simple model \[ \epsilon \text{-specification} \]

\[ \epsilon \text{-spec holds yes/no} \]

Policy \[ \mu \epsilon \rightarrow \epsilon \text{-spec} \]

\[ \epsilon \text{-quantitative abstraction} \]

Refine back

Concrete complex model

Property, specification, cost or reward

Spec holds yes/no

Policy \[ \mu \rightarrow \text{spec} \]

Correct by design

If not, tune \( \epsilon \)
Outline

1. Formal abstractions for verification of complex dynamical systems

2. Formal verification of stochastic hybrid systems
   - Analysis and control synthesis problems
   - Verification via formal abstractions

3. Case study: demand response in energy networks

4. Formal verification of max-plus linear models
   - Analysis and control synthesis problems
   - Verification via formal abstractions
Formal abstractions for verification of complex models

- Abstract simple model
  - $\epsilon$-specification

- $\epsilon$-quantitative abstraction

- Concrete complex model
  - Property, specification, cost or reward

- Model checking
- Automatic verification
- Control synthesis

$\epsilon$-spec holds yes/no policy $\mu_\epsilon \rightarrow \epsilon$-spec

- Refine back

Spec holds yes/no policy $\mu \rightarrow \text{spec}$ (correct by design)

If not, tune $\epsilon$
Formal abstractions for verification of dtSHS

- **dtMC**
- **dtMDP**
- **ε-specification**

**PRISM**
- **MRMC**
- prob. model checking
- dynamic programming

ε-spec holds
- policy max/min ε-spec

- adaptive, sequential abstractions
- approximate probabilistic bisimulations

- **dtSHS**
- **PCTL automata**

spec holds
- policy max/min spec

refine back
Stochastic hybrid (discrete/continuous) systems

PRISM
MRMC
prob. model
checking
dynamic
programming

$\epsilon$-spec holds
policy max/min $\epsilon$-spec

dtMC
$\epsilon$-specification
$\epsilon$-spec holds
policy max/min $\epsilon$-spec

adaptive, sequential abstractions
approximate probabilistic bisimulations
refine back

dtMDP

PCTL automata
spec holds policy max/min spec

A. Abate, Oxford
Stochastic hybrid (discrete/continuous) systems

- discrete-time models

**finite-space Markov chain**

\((Z, \mathcal{T})\)

\(Z = (z_1, z_2, z_3)\)

\(\mathcal{T} = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & \cdots & \cdots \\ \cdots & \cdots & \cdots \end{bmatrix}\)

\(P(z_1, \{z_2, z_3\}) = p_{12} + p_{13}\)

**uncountable-space Markov process**

\((S, T_s)\)

\(S = \mathbb{R}^2\)

\(T_s(x|s) = \frac{e^{-\frac{1}{2}(x-m(s))^T\Sigma^{-1}(s)(x-m(s))}}{\sqrt{2\pi|\Sigma(s)|^{1/2}}}\)

\(P(s, A) = \int_A T_s(dx|s), \quad A \in \mathcal{B}(S)\)
Stochastic hybrid (discrete/continuous) systems

- discrete-time models

**finite-space Markov chain**

\((Z, \mathcal{T})\)

\(Z = (z_1, z_2, z_3)\)

\[\begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & \cdots & \cdots \\ \cdots & \cdots & \cdots \end{bmatrix}\]

\[P(z_1, \{z_2, z_3\}) = p_{12} + p_{13}\]

**uncountable-space Markov process**

\((S, T_s)\)

\(S = \mathbb{R}^2\)

\[T_s(x|s) = \frac{e^{-\frac{1}{2}(x-m(s))^T \Sigma^{-1}(s)(x-m(s))}}{\sqrt{2\pi |\Sigma(s)|}^{1/2}}\]

\[P(s, A) = \int_A T_s(dx|s), \quad A \in \mathcal{B}(S)\]

\(\Rightarrow\) discrete-time, **stochastic hybrid system**

- \((S, T_s)\)

\(S = \bigcup_{q \in Q} (\{q\} \times \mathbb{R}^n), Q\) a discrete set of modes

\(T_s : S \times S \rightarrow [0, 1]\) specifies the dynamics of process at any hybrid point \((q, x)\)
Stochastic hybrid (discrete/continuous) systems

- countable set of **discrete** modes
- uncountable **continuous** domains,
  with mode-dependent dynamics, transitions, resets
- **stochasticity** everywhere
Stochastic hybrid (discrete/continuous) systems

Definition

A discrete-time stochastic hybrid system is a pair $(S, T_s)$, where

- $S = \bigcup_{q \in Q} (\{q\} \times \mathbb{R}^n)$, $Q$ a discrete set of modes
- $T_s : S \times S \rightarrow [0, 1]$ specifies the dynamics of process at point $s = (q, x)$:

$$T_s(ds' | s) = \begin{cases} 
T_x(dx' | (q, x))T_q(q | (q, x)), & \text{if } q' = q \text{ (no transition)} \\
T_r(dx' | (q, x), q')T_q(q' | (q, x)), & \text{if } q' \neq q \text{ (transition)} 
\end{cases}$$

- initial state $\pi : S \rightarrow [0, 1]$
Stochastic hybrid (discrete/continuous) systems

- Gaussian processes with nonlinear drift and diffusion
  \[ s(k + 1) = f(s(k)) + g(s(k))\eta(k), \quad \eta(\cdot) \sim \mathcal{N}(0, 1) \]

affine case with additive noise: \[ s(k + 1) = As(k) + B + \eta(k) \]

- control-dependent models \((u \in \mathcal{U})\)
  \[ T_s : S \times \mathcal{U} \times S \to [0, 1] \]

[AA et al - Automatica 08]
Stochastic hybrid systems in energy networks

- **modes**: ON vs OFF ($m$)
- **continuous dynamics** over temperature ($\theta$)
- **uncertainty**: weather, occupancy ($w$)

thermostatically-controlled load (TCL):

$$\theta(t + 1) = a \theta(t) + (1 - a)(\theta_a - m(t) \text{RP}_{rate}) + w(t)$$

$$m(t + 1) = \begin{cases} 
0, & \theta(t) < \theta_s - \delta/2 \\
1, & \theta(t) > \theta_s + \delta/2 \\
m(t), & \text{else}
\end{cases}$$
Stochastic hybrid systems in energy networks

- **modes**: ON vs OFF ($m$)
- **continuous dynamics** over temperature ($\theta$)
- **uncertainty**: weather, occupancy ($w$)

thermostatically-controlled load (TCL):

$$\theta(t + 1) = a \theta(t) + (1 - a)(\theta_a - m(t) RP_{rate}) + w(t)$$

$$m(t + 1) = \begin{cases} 
0, & \theta(t) < \theta_s - \frac{\delta}{2} \\
1, & \theta(t) > \theta_s + \frac{\delta}{2} \\
m(t), & \text{else}
\end{cases}$$

- provide aggregate model of large populations of TCL
  (use in demand-response schemes)

FP7 STREP “MoVeS”, [S. Soudjani, AA - ECC13, TCST14]
Outline

1. Formal abstractions for verification of complex dynamical systems

2. Formal verification of stochastic hybrid systems
   - Analysis and control synthesis problems
   - Verification via formal abstractions

3. Case study: demand response in energy networks

4. Formal verification of max-plus linear models
   - Analysis and control synthesis problems
   - Verification via formal abstractions
Analysis and control synthesis problems

PRISM
MRMC

prob. model checking

dynamic programming

\(\epsilon\)-spec holds
policy max/min \(\epsilon\)-spec

\(\epsilon\)-spec holds
policy max/min \(\epsilon\)-spec

refine back

adaptive, sequential abstractions
approximate probabilistic bisimulations

\(\epsilon\)-specification

\(\epsilon\)-spec holds
policy max/min spec

A. Abate, Oxford slide 13 /50
Analysis and control synthesis problems

reachability
(safety/invariance)

reach-avoid
(constrained reachability)

sequential reachability
(trajecotry planning)

∞-horizon objectives
(invariance, persistence)
Probabilistic safety: characterisation

- **Probabilistic safety** is *the probability that the execution, started at s, stays in S (safe set) during the time horizon* $[0, N]$:

$$P_s(S) = P_s(s_k \in S, \forall k \in [0, N])$$
Probabilistic safety: characterisation

- **probabilistic safety** is the probability that the execution, started at \( s \), stays in \( S \) (safe set) during the time horizon \([0, N]\):

\[
\mathcal{P}_s(S) = P_s(s_k \in S, \forall k \in [0, N])
\]

- consider realization \( s_k \in S, k \in [0, N] \) – then

\[
\prod_{k=0}^{N} 1_{S}(s_k) = \begin{cases} 
1, & \text{if } \forall k \in [0, N]: s_k \in S \\
0, & \text{otherwise}
\end{cases}
\]

\[
\Rightarrow \mathcal{P}_s(S) = P_s \left( \prod_{k=0}^{N} 1_{S}(s_k) = 1 \right) = E_s \left[ \prod_{k=0}^{N} 1_{S}(s_k) \right]
\]

- select \( \theta \in [0, 1] \) – probabilistic safe/invariant set with safety level \( \theta \) is

\[
S(\theta) = \{ s \in S : \mathcal{P}_s(S) \geq \theta \}
\]

[AA et al. - Automatica 08]
Probabilistic safety: computation

- computation of $\mathcal{P}_s(S)$ (and thus of $S(\theta)$) via dynamic programming: sequential update, backward in time, of multi-stage value function

$$V_k(s) : [0, N] \times S \rightarrow \mathbb{R}^+,$$

accounting for current and expected future rewards – in particular

$$V_N(s) = 1_S(s), \quad V_k(s) = \int_S V_{k+1}(x) T_s(dx|s)$$

$$V_0(s) = \mathcal{P}_s(S) \Rightarrow S(\theta)$$

- control dependent models: find optimal policy $\mu = (u_0, \ldots, u_k, \ldots)$, optimising recursively over

$$V_k(s, u_k) : [0, N] \times S \times \mathcal{U} \rightarrow \mathbb{R}^+$$
Computing probabilistic safety: issues

• issues
  1. non-standard (e.g., multiplicative, discontinuous) value function
  2. hybrid state space
  3. (possibly) continuous control space

⇒ **solution of DP** is seldom analytical
Computing probabilistic safety: issues

- issues
  1. non-standard (e.g., multiplicative, discontinuous) value function
  2. hybrid state space
  3. (possibly) continuous control space

⇒ solution of DP is seldom analytical

- numerical solutions are needed

⇒ problem # 1: difference between real solution and computed solution (crucial for correct-by-design controller synthesis)

⇒ problem # 2: Bellman’s curse of dimensionality (state/control space explosion)
Alternative approaches: distributed- and approximate-
Outline

1. Formal abstractions for verification of complex dynamical systems

2. Formal verification of stochastic hybrid systems
   - Analysis and control synthesis problems
   - Verification via formal abstractions

3. Case study: demand response in energy networks

4. Formal verification of max-plus linear models
   - Analysis and control synthesis problems
   - Verification via formal abstractions
Dynamical properties as temporal specifications

- PRISM
- MRMC

\[ \text{prob. model checking} \]
\[ \text{\( \epsilon \)-spec holds} \]
\[ \text{policy max/min \( \epsilon \)-spec} \]

- dtMC
- dtMDP

\[ \text{\( \epsilon \)-specification} \]

- adaptive, sequential abstractions
- approximate probabilistic bisimulations

- dtSHS
- PCTL automata

\[ \text{spec holds} \]
\[ \text{policy max/min spec} \]

refine back
Dynamical properties as temporal specifications

- recall properties and objectives discussed above . . .
- temporal specifications of finite-state systems expressed via modal logics
- PCTL used to express dynamical properties of finite-state dtMC \((\mathcal{Z}, \mathcal{T})\)

**PCTL formulae are defined over states (via labels)**

If \(\phi\) is a path formula and \(p \in [0, 1]\), then \(\Psi = P^\sim p[\phi]\) is a state formula

Example:

\[ P \geq 0.5 \land U \leq 10 \Delta \]

Model checking of PCTL formula:

**input:** dtMC \((\mathcal{Z}, \mathcal{T})\), PCTL formula \(\Psi\)

**output:** \(\text{Sat}(\Psi) = \{ z \in \mathcal{Z} : z \models \Psi \}\)

from dtSHS to dtMC: how to? Formal abstractions
Dynamical properties as temporal specifications

- recall properties and objectives discussed above . . .
- temporal specifications of finite-state systems expressed via modal logics
- PCTL used to express dynamical properties of finite-state dtMC \((Z, T)\)

- PCTL formulae are defined over states (via labels)
- if \(\varphi\) is a path formula and \(p \in [0, 1]\), then \(\Psi = P_{\sim p}[\varphi]\) is a state formula
- example: \(P_{\geq 0.5}[\Gamma \Upsilon \leq 10 \Delta]\)

- model checking of PCTL formula:
  - input: dtMC \((Z, T)\), PCTL formula \(\Psi\)
  - output: \(\text{Sat}(\Psi) = \{z \in Z : z \models \Psi\}\)

- from dtSHS to dtMC: how to? Formal abstractions
Abstraction algorithm

- approximate dtSHS \((S, T_s)\) as dtMC \((\mathcal{Z}, \mathcal{T})\), where
  - \(\mathcal{Z} = \{z_1, z_2, \ldots, z_p\}\) – finite set of abstract states
  - \(\mathcal{T} : \mathcal{Z} \times \mathcal{Z} \rightarrow [0, 1]\) – transition probability matrix
Abstraction algorithm

- approximate dtSHS \((S, T_s)\) as dtMC \((Z, \mathcal{T})\), where
  - \(Z = \{z_1, z_2, \ldots, z_p\}\) – finite set of abstract states
  - \(\mathcal{T} : Z \times Z \rightarrow [0, 1]\) – transition probability matrix

- algorithm:

<table>
<thead>
<tr>
<th>input: dt-MP ((S, T_s))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 select finite partition (S = \bigcup_{i=1}^{p} S_i)</td>
</tr>
<tr>
<td>2 and representative points (z_i \in S_i)</td>
</tr>
<tr>
<td>3 define finite state space (Z := {z_i, i = 1, \ldots, p})</td>
</tr>
<tr>
<td>4 and transition probability matrix: (\mathcal{T}(z_i, z_j) = T_s(S_j</td>
</tr>
</tbody>
</table>

  | output: dtMC \((Z, \mathcal{T})\) |
Abstraction algorithm

- approximate dtSHS \((S, T_s)\) as dtMC \((Z, \mathcal{T})\), where
  - \(Z = \{z_1, z_2, \ldots, z_p\}\) – finite set of abstract states
  - \(\mathcal{T} : Z \times Z \to [0, 1]\) – transition probability matrix

- algorithm:

  **input**: dt-MP \((S, T_s)\)
  1 select finite partition \(S = \bigcup_{i=1}^{p} S_i\)
  2 and representative points \(z_i \in S_i\)
  3 define finite state space \(Z := \{z_i, i = 1, \ldots, p\}\)
  4 and transition probability matrix: \(\mathcal{T}(z_i, z_j) = T_s(S_j \mid z_i)\)

  **output**: dtMC \((Z, \mathcal{T})\)
Abstraction algorithm

- approximate dtSHS \((S, T_s)\) as dtMC \((Z, \mathcal{T})\), where
  - \(Z = \{z_1, z_2, \ldots, z_p\}\) – finite set of abstract states
  - \(\mathcal{T} : Z \times Z \rightarrow [0, 1]\) – transition probability matrix

algorithm:

<table>
<thead>
<tr>
<th>input: dt-MP ((S, T_s))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 select finite partition (S = \bigcup_{i=1}^{p} S_i)</td>
</tr>
<tr>
<td>2 and representative points (z_i \in S_i)</td>
</tr>
<tr>
<td>3 define finite state space (Z := {z_i, i = 1, ..., p})</td>
</tr>
<tr>
<td>4 and transition probability matrix: (\mathcal{T}(z_i, z_j) = T_s(S_j \mid z_i))</td>
</tr>
</tbody>
</table>

| output: dtMC \((Z, \mathcal{T})\) |
Abstraction algorithm

- approximate dtSHS \((S, T_s)\) as dtMC \((\mathcal{Z}, \mathcal{T})\), where
  - \(\mathcal{Z} = \{z_1, z_2, \ldots, z_p\}\) – finite set of abstract states
  - \(\mathcal{T} : \mathcal{Z} \times \mathcal{Z} \rightarrow [0, 1]\) – transition probability matrix

algorithm:

\begin{verbatim}
input: dt-MP \((S, T_s)\)
1 select finite partition \(S = \bigcup_{i=1}^{p} S_i\)
2 and representative points \(z_i \in S_i\)
3 define finite state space \(\mathcal{Z} := \{z_i, i = 1, \ldots, p\}\)
4 and transition probability matrix: \(\mathcal{T}(z_i, z_j) = T_s(S_j \mid z_i)\)
output: dtMC \((\mathcal{Z}, \mathcal{T})\)
\end{verbatim}
Model checking probabilistic safety via formal abstractions

- safety set $S \subset \mathcal{S}$, time horizon $N$, safety level $\theta$
Model checking probabilistic safety via formal abstractions

- safety set $S \subset S$, time horizon $N$, safety level $\theta$
- $\delta$-abstract $(S, T_s)$ as dtMC $(Z, \mathcal{F})$, so that $S \rightarrow S_\delta$
- compute approximation error $f(\delta, N)$

⇒ probabilistic safe set

$$S(\theta) = \{s \in S : \mathbb{P}_s(S) \geq \theta\}$$
$$= \{s \in S : (1 - \mathbb{P}_s(S)) \leq 1 - \theta\}$$
Model checking probabilistic safety via formal abstractions

- safety set $S \subset S$, time horizon $N$, safety level $\theta$
- $\delta$-abstract $(S, T_s)$ as dtMC $(Z, \mathcal{F})$, so that $S \rightarrow S_\delta$
- compute approximation error $f(\delta, N)$

$\Rightarrow$ probabilistic safe set

$$S(\theta) = \{s \in S : P_s(S) \geq \theta\}$$
$$= \{s \in S : (1 - P_s(S)) \leq 1 - \theta\}$$

can be related to

$$Z_\delta(\theta + f(\delta, N)) \doteq \text{Sat} \left( P_{\leq 1-\theta-f(\delta,N)} \left( \text{true } \mathcal{U}^{\leq N} \neg S_\delta \right) \right)$$
$$= \{z \in Z : z \models P_{\leq 1-\theta-f(\delta,N)} \left( \text{true } \mathcal{U}^{\leq N} \neg S_\delta \right) \}$$

[AA et al. - EJC 11]
Quantification of abstraction error $f(\delta, N)$

- approximation via $\delta$-partitioning of safe set $S$

\[
\text{error is } |P_s(S) - P_{z_i}(S_\delta)| \leq \max_{i=1,\ldots,p} N_{\delta_i} \sum_{j=1,\ldots,p} h(i,j) = f(\delta, N),
\]

$\delta_i = \text{diam}(S_i)$, $\delta = \max_{i=1,\ldots,p} \delta_i$

Error is linear in $N$, $\delta_i$ and depends on local constants $h(i,j)$ → local tuning

[AA et al. - EJC 11, S. Soudjani, AA - QEST 11, TAC 13]
Quantification of abstraction error $f(\delta, N)$

- approximation via $\delta$-partitioning of safe set $S$

- under Lip-continuity assumptions on density of kernel $T_s$, $h(i, j), \quad i, j = 1, \ldots, p$

- for any $z^i \in S_\delta, \forall s : s \wedge z^i \in S^i$, error is

$$|\mathcal{P}_s(S) - \mathcal{P}_{z^i}(S_\delta)| \leq \max_{i=1,\ldots,p} N\delta_i \sum_{j=1,\ldots,p} h(i, j) \doteq f(\delta, N),$$

$$\delta_i = \text{diam} (S^i), \quad \delta = \max_{i=1,\ldots,p} \delta_i$$

[AA et al. - EJC 11, S. Soudjani, AA - QEST 11, TAC 13]
Quantification of abstraction error $f(\delta, N)$

- approximation via $\delta$-partitioning of safe set $S$

- under Lip-continuity assumptions on density of kernel $T_s$,

\[ h(i, j), \quad i, j = 1, \ldots, p \]

- for any $z^i \in S_\delta$, $\forall s : s \land z^i \in S^i$, error is

\[
|\mathcal{P}_s(S) - \mathcal{P}_{z^i}(S_\delta)| \leq \max_{i=1,\ldots,p} N\delta_i \sum_{j=1,\ldots,p} h(i, j) \overset{\text{def}}{=} f(\delta, N),
\]

\[
\delta_i = \text{diam} \left( S^i \right), \quad \delta = \max_{i=1,\ldots,p} \delta_i
\]

error is linear in $N, \delta_i$ and depends on local constants $h(i, j) \to$ local tuning

[AA et al. - EJC 11, S. Soudjani, AA - QEST 11, TAC 13]
Use of abstraction error $f(\delta, N)$
Use of abstraction error $f(\delta, N)$

[S. Soudjani, AA - QEST 11, HSCC 12, ATVA12, SIAM 13; I. Tkachev, AA - HSCC 13]
Use of abstraction error $f(\delta, N)$

[S. Soudjani, AA - QEST 11, HSCC 12, ATVA12, SIAM 13; I. Tkachev, AA - HSCC 13]
Use of abstraction error $f(\delta, N)$
Use of abstraction error $f(\delta, N)$

[S. Soudjani, AA - QEST 11, HSCC 12, ATVA12, SIAM 13; I. Tkachev, AA - HSCC 13]
Use of abstraction error $f(\delta, N)$

[S. Soudjani, AA - QEST 11, HSCC 12, ATVA12, SIAM 13; I. Tkachev, AA - HSCC 13]
Use of abstraction error $f(\delta, N)$

- error generalization
  - discontinuous and partially degenerate kernels
  - ill-conditioned kernels (different time scales, e.g. biology)
  - structured dynamics (sparse variable coupling, pathways)

- error bound refinement
  - tradeoff time/memory

[S. Soudjani, AA - QEST 11, HSCC 12, ATVA12, SIAM 13; I. Tkachev, AA - HSCC 13]
Characterization of $\infty$-horizon properties

- consider target set $T$; invariant set $S = T^c = S \setminus T$; $\forall s \in S$:

$$P_s(\forall n \geq 0 : s_n \in S) \iff 1 - P_s(\text{true} \cup T)$$

[I. Tkachev, AA - CDC 11, HSCC 12, CDC 12, TCS 13, SPL 14]
Characterization of $\infty$-horizon properties

- consider target set $T$; invariant set $S = T^c = S \setminus T$; $\forall s \in S$:
  \[ P_s(\forall n \geq 0 : s_n \in S) \iff 1 - P_s(\text{true} \cup T) \]

- existence and computation of absorbing sets $B$
  \[ \forall x \in B, T_s(B|x) = 1 \]

- use of coupling techniques for exponential error bounds in time

[I. Tkachev, AA - CDC 11, HSCC 12, CDC 12, TCS 13, SPL 14]
Model checking general specifications

- generalizations to
  - any PCTL formula (→ reachability, reach-avoid)
  - DFA and Büchi automata
- boils down to probabilistic reachability computations on hybrid spaces

[AA et al. - HSCC 11; D’Innocenzo, AA, J.-P. Katoen - HSCC 12; I. Tkachev et al. - HSCC13]
Controller synthesis over quantitative specifications

Abstract model

Concrete model
Controller synthesis over quantitative specifications

Abstract model
correct-by-construction policy $u_k$

Concrete model
Controller synthesis over quantitative specifications

Abstract model

correct-by-construction policy $u_k$

⇓ refine to

Concrete model

construct certified policy $u$

based on abstract policy $u_k$
Controller synthesis over quantitative specifications

Abstract model
- correct-by-construction policy $u_k$
- \Downarrow \text{refine to}

Concrete model
- construct certified policy $u$
  - based on \textbf{abstract} policy $u_k$

* extensions to \textbf{partially observed} models
FAUST$^2$: Software for formal abstractions

PRISM
MRMC
prob. model
checking
dynamic
programming

$\epsilon$-spec holds
policy max/min $\epsilon$-spec

$\epsilon$-specification

$\epsilon$-spec holds
policy max/min $\epsilon$-spec

refine back

spec holds
policy max/min spec

adaptive, sequential abstractions
approximate probabilistic bisimulations

PRISM MRMC
prob. model checking
dynamic programming

$\epsilon$-specification

$\epsilon$-spec holds
policy max/min $\epsilon$-spec

adaptation, sequential abstractions
approximate probabilistic bisimulations

$\epsilon$-spec holds
policy max/min $\epsilon$-spec

refine back

spec holds
policy max/min spec

PCTL automata

$\epsilon$-spec holds
policy max/min $\epsilon$-spec

refine back

spec holds
policy max/min spec
FAUST²: Software for formal abstractions

- sequential, adaptive formal abstractions – anytime algorithm
- from MATLAB/Simulink model to MRMC/PRISM input
- scales to larger models than alternative grid-based techniques

http://sourceforge.net/projects/faust2

[S. Soudjani, AA - SIAM 13, TACAS 15]
Approximate probabilistic bisimulations

- PRISM
- MRMC
  - prob. model checking
- $\epsilon$-specification
  - $\epsilon$-spec holds
    - policy max/min $\epsilon$-spec

- dtMC, dtMDP
  - $\epsilon$-specification
    - adaptive, sequential abstractions
    - approximate probabilistic bisimulations

- dtSHS
  - PCTL automata
  - spec holds
    - policy max/min spec

- refine back
Approximate probabilistic bisimulations

\[ T_z = \begin{bmatrix} 0.6 & 0.3 & 0.1 \\ 0.4 & 0.1 & 0.5 \\ 0.6 & 0.3 & 0.1 \end{bmatrix} \]

\[ \bar{T}_z = \begin{bmatrix} 0.7 & 0.3 \\ 0.9 & 0.1 \end{bmatrix} \]

- \((\bar{S}, \bar{T}_z)\) is a "lumped" version of \((S, T_z)\)
- exact probabilistic bisimulation as model abstraction

\[ [\text{Larsen & Skou, 91}] \]
Approximate probabilistic bisimulations

\[
T_z = \begin{bmatrix}
0.6 & 0.3 & 0.1 \\
0.4 & 0.1 & 0.5 \\
0.6 & 0.3 & 0.1 \\
\end{bmatrix}
\]

\[
\tilde{T}_z = \begin{bmatrix}
0.7 & 0.3 \\
0.9 & 0.1 \\
\end{bmatrix}
\]

- \((\tilde{S}, \tilde{T}_z)\) is a “lumped” version of \((S, T_z)\)
- exact probabilistic bisimulation as model abstraction  
  \[\text{[Larsen & Skou, 91]}\]
- now consider \(\tilde{T}_z = \begin{bmatrix}
0.6 + \delta_1 & 0.3 - \delta_1 & 0.1 \\
0.4 & 0.1 & 0.5 \\
0.6 + \delta_2 & 0.3 - \delta_2 & 0.1 \\
\end{bmatrix}\)
- \(\Rightarrow\) approximate probabilistic bisimulation  
  \[\text{[AA - ENTCS 13; I. Tkachev, AA - HSCC 13]}\]
Outline

1. Formal abstractions for verification of complex dynamical systems

2. Formal verification of stochastic hybrid systems
   - Analysis and control synthesis problems
   - Verification via formal abstractions

3. Case study: demand response in energy networks

4. Formal verification of max-plus linear models
   - Analysis and control synthesis problems
   - Verification via formal abstractions
Demand response in energy networks

- consider dynamical behaviour of temperature of single TCL, controlled via temperature setpoint

⇒ provide aggregate model for large-dimensional population of $n_p$ TCL

- control power consumption (TCL in ON mode) via temperature setpoint:

$$y_{total}(t) = \sum_{i=1}^{n_p} P_i m_i(t)$$

D. Callaway, ECC13 plenary talk
Aggregate TCL population model in the literature

- based on state-space partition: divide TCL dead band into bins
- introduce population state $X$ (TCL portion with temperature in specific bin)
- build dynamics $X(t + 1) = AX(t) + Bu(t)$
  - $A$ depends on TCL temperature dynamics over bins
  - $u$ affects portion of TCL in each bin

---

Figure 1: State bin transition model. [Koch et al., 2011]
Aggregate TCL population model in the literature

- based on state-space partition: divide TCL dead band into bins
- introduce population state $X$ (TCL portion with temperature in specific bin)
- build dynamics $X(t+1) = AX(t) + Bu(t)$
  - $A$ depends on TCL temperature dynamics over bins
  - $u$ affects portion of TCL in each bin

- aggregation procedure leading to model $X$
  1. is deterministic
  2. hinges on restrictive assumptions on TCL dynamics
  3. leads to unclear dependence of model precision on number of bins

[Koch et al., 2011]
Approximate dynamics of single TCL (dtSHS) by Markov chain (dtMC)

2 Consider population of TCL by taking cross product of Markov chains

3 Define labels of product Markov chain to be the number of TCL within an abstract state (bin)

4 Construct bisimulation of product Markov chain based on labels, obtain dynamics of aggregated model
Aggregate TCL population model via abstractions

1. approximate dynamics of single TCL (dtSHS) by Markov chain (dtMC)
2. consider population of TCL by taking cross product of Markov chains
3. define labels of product Markov chain to be the number of TCL within an abstract state (bin)
4. construct bisimulation of product Markov chain based on labels, obtain dynamics of aggregated model

★ quantify error (on total output power) introduced by the abstraction

[S. Soudjani, AA - ECC13, TCST14]
Aggregate TCL population model via abstractions

- average of 50 Monte Carlo simulations for a population of $n_p = 10^3$ TCL
- all TCL initialised in OFF mode and at set point $\theta_s$
- comparison: “deterministic” (state-of-art) vs formal abstractions

$\sigma = 0.003$

$\sigma = 0.03$

[S. Soudjani, AA - ECC13, TCST14]
Aggregate TCL population model via abstractions

- average of 50 Monte Carlo simulations for a population of $n_p = 10^3$ TCL
- heterogeneous population, with $\sigma = 0.03$
- comparison: “deterministic” (state-of-art) vs formal abstractions

$C \in [8, 12]$

$C \in [2, 18]$

[S. Soudjani, AA - ECC13, TCST14]
Aggregate TCL population model via abstractions

- centralised control of population of TCLs via setpoint changes
  1. perform state estimation from total power consumption
  2. devise closed-loop control scheme for power tracking
- again, error of procedure can be tuned

[ S. Soudjani, AA - ECC13, TCST14 ]
Aggregate TCL population model via abstractions

- centralised control of population of TCLs via setpoint changes
  1. perform state estimation from total power consumption
  2. devise closed-loop control scheme for power tracking
- again, error of procedure can be tuned

- tracking of a piece-wise constant reference signal via set-point regulation

[S. Soudjani, AA - ECC13, TCST14]
Aggregate TCL population model via abstractions

- centralised control of population of TCLs via setpoint changes
  1. perform state estimation from total power consumption
  2. devise closed-loop control scheme for power tracking
- again, error of procedure can be tuned

- tracking of a piece-wise constant reference signal via set-point regulation
Aggregate TCL population model via abstractions

- centralised control of population of TCLs via setpoint changes
  1. perform state estimation from total power consumption
  2. devise closed-loop control scheme for power tracking

- again, error of procedure can be tuned

- tracking of a piece-wise constant reference signal via set-point regulation, by SMPC scheme (with constraints)

[Graphs depicting total power consumption, reference power signal, and set-point over time]

[S. Soudjani, AA - ECC13, TCST14]
Formal abstractions for verification of complex models

- stochastic hybrid systems (today)
- max-plus linear models (discrete events)
  http://sourceforge.net/projects/verisimpl
- timed automata (continuous-time)
- ordinary differential equations (ODE)
Outline

1. Formal abstractions for verification of complex dynamical systems

2. Formal verification of stochastic hybrid systems
   - Analysis and control synthesis problems
   - Verification via formal abstractions

3. Case study: demand response in energy networks

4. Formal verification of max-plus linear models
   - Analysis and control synthesis problems
   - Verification via formal abstractions
Formal abstractions for verification of MPL models

- **LTS**: LTL, safe LTL
  - **abstractions**: bisimulations
  - **TRANS**: transient or steady-state

- **MPL**: transient or steady-state
  - **deterministic (detm.)**: possible for (exists policy) property yes/no
  - **refine back**: (forall policies) property yes

- **SPIN**: model checking
  - (exists policy) spec yes/no
  - (forall policies) spec yes
Introduction to MPL systems

- LTS
  - LTL
    - safe LTL

- MPL
  - transient or steady-state

- SPIN
  - model checking
    - (∃ policy) spec yes/no
      - (∀ policies) spec yes

- refinements back
  - (∃ policy) property yes/no
    - (∀ policies) property yes

- abstractions
  - bisimulations
    - simulations

A. Abate, Oxford slide 33/50
Introduction to MPL systems

- **Max-Plus-Linear (MPL) systems** are event-driven models
- Applications: railway scheduling, planning of production lines, network calculus

\[ x(k) \] is the time of \( k \)-th event, \( k \in \mathbb{N} \cup \{0\} \)
- Timing updates: maximization \((\oplus)\) and addition \((\otimes)\) operations

→ Max-plus algebra

- \( \varepsilon = -\infty \), \( \mathbb{R}_\varepsilon = \mathbb{R} \cup \{\varepsilon\} \), \( \alpha, \beta \in \mathbb{R}_\varepsilon \)
- \( \alpha \oplus \beta := \max(\alpha, \beta) \), \( \alpha \otimes \beta := \alpha + \beta \), and matrix operations
Max-plus algebra

- $\epsilon = -\infty$, $\mathbb{R}_\epsilon = \mathbb{R} \cup \{\epsilon\}$
- $\alpha, \beta \in \mathbb{R}_\epsilon$, $A, B \in \mathbb{R}_\epsilon^{m \times p}$, $C \in \mathbb{R}_\epsilon^{p \times n}$
- $\alpha \oplus \beta \overset{\text{def}}{=} \max(\alpha, \beta)$
- $\alpha \otimes \beta \overset{\text{def}}{=} \alpha + \beta$
Max-plus algebra

- $\epsilon = -\infty$, $\mathbb{R}_{\epsilon} = \mathbb{R} \cup \{\epsilon\}$
- $\alpha, \beta \in \mathbb{R}_{\epsilon}$, $A, B \in \mathbb{R}_{\epsilon}^{m \times p}$, $C \in \mathbb{R}_{\epsilon}^{p \times n}$

- $\alpha \oplus \beta \overset{\text{def}}{=} \max(\alpha, \beta)$

- $\alpha \otimes \beta \overset{\text{def}}{=} \alpha + \beta$

- $[A \oplus B]_{i,j} \overset{\text{def}}{=} [A]_{i,j} \oplus [B]_{i,j}$, for $i = 1, \ldots, m$ and $j = 1, \ldots, p$

- $[A \otimes C]_{i,j} \overset{\text{def}}{=} \bigoplus_{k=1}^{p} [A]_{i,k} \otimes [C]_{k,j}$, for $i = 1, \ldots, m$ and $j = 1, \ldots, n$
Max-plus-linear models

Definition (Autonomous MPL model)

\[ x(k + 1) = A \otimes x(k), \]

where \( A \in \mathbb{R}^{n \times n} \) and \( k \in \mathbb{N} \cup \{0\} \)

Example

A simple railway model [Heidergott, 06]

\[
\begin{bmatrix}
2 & 5 \\
3 & 3 \\
\end{bmatrix} \otimes \begin{bmatrix}
x_1(k+1) \\
x_2(k+1) \\
\end{bmatrix} = \begin{bmatrix}
\max\{2 + x_1(k), 5 + x_2(k)\} \\
\max\{3 + x_1(k), 3 + x_2(k)\} \\
\end{bmatrix}
\]

[Baccelli et al., 92]
Max-plus-linear models

Definition (Autonomous MPL model)

\[ x(k+1) = A \otimes x(k), \]
where \( A \in \mathbb{R}^{n \times n} \) and \( k \in \mathbb{N} \cup \{0\} \)

Example

A simple railway model [Heidergott, 06]

\[ x(k+1) = \begin{bmatrix} 2 & 5 \\ 3 & 3 \end{bmatrix} \otimes x(k), \quad \begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} \max\{2+x_1(k), 5+x_2(k)\} \\ \max\{3+x_1(k), 3+x_2(k)\} \end{bmatrix} \]

Definition (Non-autonomous MPL model)

\[ x(k+1) = A \otimes x(k) \oplus B \otimes u(k), \]
where \( B \in \mathbb{R}^{n \times m} \) and \( u \in \mathbb{R}^{m} \) (synthesis = scheduling)

[Baccelli et al., 92]
Outline

1. Formal abstractions for verification of complex dynamical systems

2. Formal verification of stochastic hybrid systems
   - Analysis and control synthesis problems
   - Verification via formal abstractions

3. Case study: demand response in energy networks

4. Formal verification of max-plus linear models
   - Analysis and control synthesis problems
   - Verification via formal abstractions
Classical analysis of MPL models

LTS
LTL
safe LTL

abstractions
bisimulations
simulations

MPL
transient or steady-state

SPIN
model checking

(∃ policy) spec yes/no
(∀ policies) spec yes

refine back

(∃ policy) property yes/no
(∀ policies) property yes

determ.
Classical analysis of MPL models

- study of transient and periodic regimes, of asymptotics
- classical analysis based on algebraic or geometric properties

Definition

1. **max-plus eigenvector**  \( x \in \mathbb{R}^n \):  \( A \otimes x = \lambda \otimes x \Rightarrow x(k + 1) = \lambda \otimes x(k) \)

2. **cycles on precedence graph** \( \Rightarrow \) periodic regime with period \( c \):
   \[ \forall k \geq k_0, x(k + c) = \lambda^{\otimes c} \otimes x(k) \]

Example

1. eigenspace (periodic regime with period 1 and \( \lambda = 4 \)):
   \[
   \begin{bmatrix}
   1 \\
   0
   \end{bmatrix}, \begin{bmatrix}
   5 \\
   4
   \end{bmatrix}, \begin{bmatrix}
   9 \\
   8
   \end{bmatrix}, \begin{bmatrix}
   13 \\
   12
   \end{bmatrix}, \begin{bmatrix}
   17 \\
   16
   \end{bmatrix}, \begin{bmatrix}
   21 \\
   20
   \end{bmatrix}, \begin{bmatrix}
   25 \\
   24
   \end{bmatrix}, \begin{bmatrix}
   29 \\
   28
   \end{bmatrix}, \begin{bmatrix}
   33 \\
   32
   \end{bmatrix}, \begin{bmatrix}
   37 \\
   36
   \end{bmatrix}, \begin{bmatrix}
   41 \\
   40
   \end{bmatrix}, \begin{bmatrix}
   45 \\
   44
   \end{bmatrix}, \ldots
   \]

2. periodic regime with period \( c = 2 \) (transient \( k_0 = 3 \)):
   \[
   \begin{bmatrix}
   4 \\
   0
   \end{bmatrix}, \begin{bmatrix}
   6 \\
   7
   \end{bmatrix}, \begin{bmatrix}
   12 \\
   10
   \end{bmatrix}, \begin{bmatrix}
   15 \\
   15
   \end{bmatrix}, \begin{bmatrix}
   20 \\
   18
   \end{bmatrix}, \begin{bmatrix}
   23 \\
   23
   \end{bmatrix}, \begin{bmatrix}
   28 \\
   26
   \end{bmatrix}, \begin{bmatrix}
   31 \\
   31
   \end{bmatrix}, \begin{bmatrix}
   36 \\
   34
   \end{bmatrix}, \begin{bmatrix}
   39 \\
   39
   \end{bmatrix}, \begin{bmatrix}
   44 \\
   42
   \end{bmatrix}, \begin{bmatrix}
   47 \\
   47
   \end{bmatrix}, \ldots
   \]
Outline

1. Formal abstractions for verification of complex dynamical systems

2. Formal verification of stochastic hybrid systems
   - Analysis and control synthesis problems
   - Verification via formal abstractions

3. Case study: demand response in energy networks

4. Formal verification of max-plus linear models
   - Analysis and control synthesis problems
   - Verification via formal abstractions
Labeled transition system (LTS)

- LTS
  - LTL safe LTL

- MPL
  - transient or steady-state

- SPIN
  - model checking
  - (∃ policy) spec yes/no
  - (∀ policies) spec yes

- refinements
  - refine back
  - (∃ policy) property yes/no
  - (∀ policies) property yes

- abstractions
  - bisimulations
  - simulations

A. Abate, Oxford
Labeled transition system (LTS)

- set of states $S = \{1, 2, 3, 4\}$
- set of inputs $Act = \{\alpha, \beta\}$
- transitions $\longrightarrow = \{(1, \alpha, 4), (4, \alpha, 3), \ldots\}$
- set of outputs $AP = \{a, b\}$ and output map $L(1) = \emptyset, L(2) = \{b\}, \ldots$

- labels can be defined over states or transitions
- LTS can be deterministic vs non-deterministic
- LTS can be infinite vs finite

[Baier & Katoen, 08]
Finite LTS as abstractions of MPL models

\[
\begin{align*}
\text{LTS} & \quad \text{LTL} \quad \text{safe LTL} \\
\text{MPL} & \quad \text{transient or steady-state}
\end{align*}
\]

SPIN

- model checking
  - (∃ policy) spec yes/no
  - (∀ policies) spec yes

abstractions

bisimulations simulations

refine back

- determin.
  - (∃ policy) property yes/no
  - (∀ policies) property yes

procedure: need to compute

- states
- transitions
- labels
LTS states: partitioning of state space

- state space $\mathbb{R}^n$ is partitioned in finitely many polytopic regions
- partition is not arbitrary, it is adapted to underlying dynamics
- obtained state-space partition defines states of LTS
- partition can be possibly refined (determinization – more later)

Example

we obtain a total of 5 regions:

- $R_1 = \{ x \in \mathbb{R}^2 : x_1 - x_2 < 0 \}$
- $R_2 = \{ x \in \mathbb{R}^2 : x_1 - x_2 = 0 \}$
- $R_3 = \{ x \in \mathbb{R}^2 : x_1 - x_2 > 3 \}$
- $R_4 = \{ x \in \mathbb{R}^2 : x_1 - x_2 = 3 \}$
- $R_5 = \{ x \in \mathbb{R}^2 : 0 < x_1 - x_2 < 3 \}$
Difference-bound matrices (DBM)

**Definition (DBM)**

A difference-bound matrix in $\mathbb{R}^n$ is the finite intersection of sets defined by

$$x_i - x_j \simeq_{i,j} \alpha_{i,j},$$

where $\simeq_{i,j} \in \{<, \leq\}$, $\alpha_{i,j} \in \mathbb{R} \cup \{+\infty\}$, for $1 \leq i \neq j \leq n$

- DBM allow **compact matrix representation**
- DBM are **easy to manipulate** (projections, emptiness and inclusion check)

[Dill, 90]
Difference-bound matrices (DBM)

Definition (DBM)

A difference-bound matrix in $\mathbb{R}^n$ is the finite intersection of sets defined by

$$x_i - x_j \simeq_{i,j} \alpha_{i,j},$$

where $\simeq_{i,j} \in \{<, \leq\}$, $\alpha_{i,j} \in \mathbb{R} \cup \{+\infty\}$, for $1 \leq i \neq j \leq n$

- DBM allow compact matrix representation
- DBM are easy to manipulate (projections, emptiness and inclusion check)
- closure: image/inverse image of DBM over MPL dynamics is again a DBM

[Dill, 90]
LTS transitions: one-step reachability

- consider any two TS states (partitioning regions) \( R, R' \)
- \( R \rightarrow R' \) iff there exists a \( x(k) \in R \) such that \( x(k+1) \in R' \): check

\[
R' \cap \{x(k+1) : x(k) \in R\} \neq \emptyset
\]
LTS transitions: one-step reachability

- consider any two TS states (partitioning regions) $R, R'$
- $R \rightarrow R'$ iff there exists a $x(k) \in R$ such that $x(k + 1) \in R'$: check

$$R' \cap \{x(k + 1) : x(k) \in R\} \neq \emptyset$$

- computation of transitions:
  - use region representation via DBM, DBM forward-mapping via PWA dynamics, DBM emptiness check
- transitions are stored in sparse Boolean matrix
LTS transitions, an example

Example

- determinism vs non-determinism of obtained TS
LTS transitions, an example

- **determinism vs non-determinism** of obtained TS
Relationship between LTS and MPL

LTS

<table>
<thead>
<tr>
<th>LTL safe LTL</th>
</tr>
</thead>
</table>

MPL

| transient or steady-state |

SPIN

model checking

(∃ policy) spec yes/no
(∀ policies) spec yes

abstractions

bisimulations simulations

determine

(∃ policy) property yes/no
(∀ policies) property yes

refine back
Theorem

- *TS simulates* the original MPL model
- *TS bisimulates* the MPL model if and only if it is *deterministic*
- every *irreducible* MPL model admits finite *deterministic TS abstraction*

- non-deterministic TS can be “determinized” by refining partitioning regions
- termination of refinement procedure does not hold in general
LTS labels

Definition

- **state labels:**
  all possible values of $x_i(k) - x_j(k)$, for $1 \leq i < j \leq n$
  time difference of **same-event variables**

- **transition labels:**
  all possible values of $x_i(k+1) - x_i(k)$, for $1 \leq i \leq n$
  time difference of **successive events**

- labels are **vectors of intervals**, can be represented as **DBM**
LTS labels, an example

Example

- LTS transition labels

\[
\begin{array}{c}
R'_1 \quad R'_2 \quad R'_3 \quad R'_4 \quad R'_5 \quad R'_6 \quad R'_7 \quad R'_8 \quad R'_9 \\
[3,3] \quad [3,3] \quad [6,∞) \quad [5,∞) \quad [3,3] \quad [5,5] \quad [5,5] \quad [3,3] \quad [4,4] \\
(2,2) \quad [6,6] \quad (2,3) \quad (5,6) \quad (4,5) \quad (3,4) \quad (3,4) \quad (4,5) \\
\end{array}
\]
Formal analysis of MPL models is now “very simple”

VeriSiMPL – Verification via biSimulation of MPL models

- LTS
  - LTL
  - safe LTL

- MPL
  - transient or steady-state

- SPIN
  - model checking

- (∃ policy) spec yes/no
  - (∀ policies) spec yes

- abstractions
  - bisimulations

- refine back

- (∃ policy) property yes/no
  - (∀ policies) property yes

A. Abate, Oxford
Formal analysis of MPL models is now “very simple”

VeriSiMPL – Verification via biSimulation of MPL models

- abstract MPL model as LTS (in MATLAB)
- export LTS abstraction (as PROMELA script) into SPIN model checker
- consider properties in LTL logic
- verify property via SPIN over LTS and export outcome back to MPL model

http://sourceforge.net/projects/verisimpl
Formal analysis of MPL models is now “very simple”

VeriSiMPL – Verification via biSimulation of MPL models

- coded in MATLAB, run over 12-core Intel Xeon, 3.47 GHz, 24 GB
- A randomly generated with elements taking values between 1 and 100
- 10 independent experiments per dimension – mean values are displayed:

<table>
<thead>
<tr>
<th>size of MPL model</th>
<th>time for generation of states</th>
<th>time for generation of transitions</th>
<th>time for generation of labels</th>
<th>total number of LTS states</th>
<th>total number of LTS transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.1 [s]</td>
<td>0.4 [s]</td>
<td>0.1 [s]</td>
<td>3.6</td>
<td>4.3</td>
</tr>
<tr>
<td>5</td>
<td>0.2 [s]</td>
<td>0.4 [s]</td>
<td>0.1 [s]</td>
<td>8.6</td>
<td>13.8</td>
</tr>
<tr>
<td>7</td>
<td>0.9 [s]</td>
<td>0.5 [s]</td>
<td>0.3 [s]</td>
<td>37.2</td>
<td>289.3</td>
</tr>
<tr>
<td>9</td>
<td>4.1 [s]</td>
<td>0.8 [s]</td>
<td>1.6 [s]</td>
<td>120.0</td>
<td>1.7·10^3</td>
</tr>
<tr>
<td>11</td>
<td>24.8 [s]</td>
<td>15.2 [s]</td>
<td>16.1 [s]</td>
<td>613.2</td>
<td>1.9·10^4</td>
</tr>
<tr>
<td>13</td>
<td>3.5 [m]</td>
<td>5.5 [m]</td>
<td>2.8 [m]</td>
<td>1.9·10^3</td>
<td>1.9·10^5</td>
</tr>
<tr>
<td>15</td>
<td>53.6 [m]</td>
<td>2.0 [h]</td>
<td>39.4 [m]</td>
<td>7.4·10^3</td>
<td>2.0·10^6</td>
</tr>
</tbody>
</table>

- bottleneck: generation of transitions
Computational benchmark for reachability analysis

- A randomly generated set of initial conditions is selected as the unit hypercube set of initial conditions is selected as the unit hypercube.

10 independent experiments per dimension – mean values are displayed:

<table>
<thead>
<tr>
<th>size of MPL model</th>
<th>time for generation of abstract TS</th>
<th>number of regions of abstract TS</th>
<th>time for generation of reach tube</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.09 [s]</td>
<td>5</td>
<td>0.09 [s]</td>
</tr>
<tr>
<td>10</td>
<td>4.73 [s]</td>
<td>700</td>
<td>8.23 [s]</td>
</tr>
<tr>
<td>19</td>
<td>67.07 [m]</td>
<td>3.48 \times 10^5</td>
<td>7.13 [h]</td>
</tr>
</tbody>
</table>

- Generation time for reach tube of 10-dimensional MPL model, different time horizons

- Comparison VeriSiMPL vs MPT (multi-parametric tool, ETH Zürich):

<table>
<thead>
<tr>
<th>time horizon</th>
<th>VeriSiMPL</th>
<th>MPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>11.02 [s]</td>
<td>47.61 [m]</td>
</tr>
<tr>
<td>40</td>
<td>17.94 [s]</td>
<td>1.19 [h]</td>
</tr>
<tr>
<td>60</td>
<td>37.40 [s]</td>
<td>2.32 [h]</td>
</tr>
<tr>
<td>80</td>
<td>51.21 [s]</td>
<td>3.03 [h]</td>
</tr>
<tr>
<td>100</td>
<td>64.59 [s]</td>
<td>3.73 [h]</td>
</tr>
</tbody>
</table>
Example

- automatically find MPL eigenspace: \( \bigvee_{\varphi \in L=AP} (\Box \varphi \land |\varphi| = 0) \)
Example

- automatically find MPL **periodic regime**: 
\[
\bigvee_{\varphi \in L} AP \Box (\varphi \land \Box^c \varphi)
\]
Outline

1. Formal abstractions for verification of complex dynamical systems

2. Formal verification of stochastic hybrid systems
   - Analysis and control synthesis problems
   - Verification via formal abstractions

3. Case study: demand response in energy networks

4. Formal verification of max-plus linear models
   - Analysis and control synthesis problems
   - Verification via formal abstractions
Formal abstractions for verification of complex models

abstract simple model

$\epsilon$-specification

$\epsilon$-quantitative abstraction

model checking

automatic verification

control synthesis

$\epsilon$-spec holds yes/no

policy $\mu_\epsilon \rightarrow \epsilon$-spec

refine back

concrete complex model

property, specification, cost or reward

spec holds yes/no

policy $\mu \rightarrow$ spec

(correct by design)

if not, tune $\epsilon$
Thanks for your attention!

For more info:

aabate@cs.ox.ac.uk
Selected key references

Additional references