Reactive Probabilistic Programming Semantics with Mixed Nondeterministic/Probabilistic Automata

Albert Benveniste       Jean-Baptiste Raclet

INRIA Rennes and IRIT Toulouse

September 18, 2020
What is Probabilistic Programming?

*Bringing the inference algorithms and theory from statistics combined with formal semantics, compilers, and other tools from programming languages to build efficient inference evaluators for models and applications from Machine Learning.* [...] *Probabilistic programming is a tool for statistical modeling.* (Fabiana Clemente)
Basic issues in probabilistic paradigms

Approaches
- Statisticians and AI people
- Reactive Programming: ProbZelus

ReactiveBayes minilanguage

Factor Graphs + constraints = Mixed Systems

Putting dynamics: Mixed Automata
- Preliminaries to Mixed Automata
- Mixed Automata

ReactiveBayes and its semantics

Discussion and Comparisons
- Probabilistic Automata
- (Non reactive) Probabilistic Programming
- Reactive Probabilistic Programming

Limitations of Mixed Automata and Fixes

Conclusion
Basic issues in probabilistic paradigms

- Specifying a probabilistic system
  - A distribution (Bernoulli, Gaussian . . .)
  - A probabilistic dynamics (Markov Chain, Cyber Physical System subject to noise, Safety analysis . . .)

- Estimating, learning, inferring
  - Model parameters
  - Black-box dynamics (deep learning)

- Statistical decision and classification
Basic issues in probabilistic paradigms

- **Specifying a probabilistic system**
  - A distribution (Bernoulli, Gaussian...) 
  - A probabilistic dynamics (Markov Chain, Cyber Physical System subject to noise, Safety analysis...) 

- **Estimating, learning, inferring**
  - Model parameters 
  - black-box dynamics (deep learning) 

- **Statistical decision and classification**

- **Issues**
  - Blending probabilities and nondeterminism 
  - Modularity in the above tasks
Requirements on Probabilistic Programming

- Probabilistic programming: offer a high-level language for the
  - specification
  - estimation
  - decision/detection/classification

of systems involving a mix of proba and nondeterminism
Requirements on Probabilistic Programming

- Probabilistic programming: offer a high-level language for the
  - specification
  - estimation
  - decision/detection/classification
  of systems involving a mix of proba and nondeterminism

- Supporting important nontrivial constructions:
  - Conditioning: $\pi(A \mid B) = \text{def} \frac{\pi(A \cap B)}{\pi(B)}$ provided that $\pi(B) > 0$
  - Modularity in specification, estimation, and decision:
    - Factor Graphs & Bayesian Networks
      (generalizations of Bayes rule $P(X, Y) = P(X)P(Y \mid X)$)
    - Parallel composition
Requirements on Probabilistic Programming

- Probabilistic programming: offer a high-level language for the
  - specification
  - estimation
  - decision/detection/classification
  of systems involving a mix of proba and nondeterminism

- Supporting important nontrivial constructions:
  - Conditioning: \( \pi(A \mid B) = \frac{\pi(A \cap B)}{\pi(B)} \) provided that \( \pi(B) > 0 \)
  - Modularity in specification, estimation, and decision:
    - Factor Graphs & Bayesian Networks
      (generalizations of Bayes rule \( P(X, Y) = P(X)P(Y \mid X) \))
    - Parallel composition

- Hosting libraries of algorithms for estimation and decision

- Providing a layered language for supporting all of this
Requirements on Probabilistic Programming

- **Factor Graphs**: nondirected

- **Bayesian Networks**: directed, for causal reasoning
Advantages of a layered language

3 layers, each one specifying:

- a probabilistic system
  - semantics, equivalence, rewriting rules

- a statistical problem (probability of some property, sampling, estimating, detecting, classifying,...)
  - semantics, equivalence, rewriting rules

- algorithms for solving statistical problems
  - operational semantics
Basic issues in probabilistic paradigms

**Approaches**
- Statisticians and AI people
- Reactive Programming: ProbZelus

*ReactiveBayes minilanguage*

**Factor Graphs + constraints = Mixed Systems**

**Putting dynamics: Mixed Automata**
- Preliminaries to Mixed Automata
- Mixed Automata

*ReactiveBayes and its semantics*

**Discussion and Comparisons**
- Probabilistic Automata
  - (Non reactive) Probabilistic Programming
- Reactive Probabilistic Programming

**Limitations of Mixed Automata and Fixes**

**Conclusion**
Existing approaches by statisticians and AI people

Pragmatic proposals by statisticians and AI people

- **BUGS** [Spiegelhalter 1994]: a software package for Bayesian inference using Gibbs sampling. The software has been instrumental in raising awareness of Bayesian modelling among both academic and commercial communities internationally, and has enjoyed considerable success over its 20-year life span. 2009

- **Stan** [Carpenter 2017]: Stan is a probabilistic programming language for specifying statistical models. A Stan program imperatively defines a log probability function over parameters conditioned on specified data and constants. As of version 2.14.0, Stan provides full Bayesian inference for continuous-variable models through Markov chain Monte Carlo methods such as the No-U-Turn sampler, an adaptive form of Hamiltonian Monte Carlo sampling. Penalized maximum likelihood estimates are calculated using optimization methods such as the limited memory Broyden-Fletcher-Goldfarb-Shanno algorithm.

- As part of **TensorFlow** open source platform for machine learning
Existing approaches by statisticians and AI people

- Programming languages for specifying
  - **Factor Graphs**: nondirected

![Factor Graph Example]

  \[
  p(x_1, x_2, x_3) = f_1(x_1, x_2) f_2(x_2, x_3) f_3(x_1, x_3) f_4(x_3)
  \]

- **Bayesian Networks**: directed, for causal reasoning

![Bayesian Network Example]

- Emphasis is on algorithms for performing **Bayesian inference**; Decentralized algorithms with local computations only (Metropolis, MCMC, . . . ) in order to scale up
A conservative extension of Lucid Synchrone synchronous language with probabilistic primitives:

- `x = sample d`: declares random variable $X$ with distribution $d$
- `observe(d, y)`: estimates likelihood of $y$ wrt distribution $d$
- `infer(m, obs)`: infers distribution of outputs of model $m$ based on observations of $obs$
Basic issues in probabilistic paradigms

Approaches
Statisticians and AI people
Reactive Programming: ProbZelus

*ReactiveBayes minilanguage*

Factor Graphs + constraints = Mixed Systems

Putting dynamics: Mixed Automata
Preliminaries to Mixed Automata
Mixed Automata

*ReactiveBayes and its semantics*

Discussion and Comparisons
Probabilistic Automata
(Non reactive) Probabilistic Programming
Reactive Probabilistic Programming

Limitations of Mixed Automata and Fixes

Conclusion
Example of programming we would like to support

S1:

```
system System(y0)
  observed u
  and init y = y0
  and v = u + pre y
  and y = if fail then v + noise else v
```

S2:

```
system Noisegen(var)
  init noise = 0.0
  and noise = 0.9 * pre noise + w
  and w ~ normal(0, var)
```

S3:

```
system Failure(p)
  init backup = false
  and fail = rootfail ∧ ¬ pre backup
  and rootfail ~ Bernoulli(p)
```

S4:

```
system Sensor
  observed y

with parallel compositions of any of them
probabilistic statements x ~ ··· are private
Example of programming we would like to support

S1 : \[
\begin{aligned}
\text{system System}(y0) \\
\quad \text{observed } u \\
\quad \text{and init } y = y0 \\
\quad \text{and } v = u + \text{pre } y \\
\quad \text{and } y = \text{if fail then } v + \text{noise else } v
\end{aligned}
\]

\[
\forall n : \left\{ \begin{array}{lcl}
\text{observed}(u_n) \\
v_n &=& u_n + y_{n-1} \\
y_n &=& \text{if } \text{fail}_n \text{ then } (v_n + \text{noise}_n) \text{ else } v_n
\end{array} \right.
\]

- The semantics is a dynamical system with observed and unobserved signals; traces of observed signals are fixed; equations define relations.
- Intuition: signal $u_n$ can be seen as an input; $\text{fail}_n$, $\text{noise}_n$ as nondeterministic inputs (daemons).
Example of programming we would like to support

\[
\begin{align*}
\text{S1 : } & \quad \text{system System}(y_0) \\
& \quad \text{ observed } u \\
& \quad \text{ and init } y = y_0 \\
& \quad \text{ and } v = u + \text{pre } y \\
& \quad \text{ and } y = \text{if fail then } v + \text{noise else } v \\
\text{S4 : } & \quad \text{system Sensor} \\
& \quad \text{ observed } y
\end{align*}
\]

S1 and S4

\[
\forall n : \begin{cases}
\text{observed}(u_n, y_n) \\
\quad v_n = u_n + y_{n-1} \\
\quad y_n = \text{if fail}_n \text{ then } (v_n + \text{noise}_n) \text{ else } v_n
\end{cases}
\]

- The semantics is a dynamical system with observed and unobserved signals; traces of observed signals are fixed; equations define relations.

- Intuition: signal \( u_n \) can be seen as an input; \( \text{fail}_n, \text{noise}_n \) as nondeterministic inputs (daemons). Output \( y_n \) is measured.
Example of programming we would like to support

\[
S2 : \begin{cases}
\text{system} \text{ NoiseGen}(\text{var}) \\
\quad \text{init} \ \text{noise} = 0.0 \\
\quad \text{and} \ \text{noise} = 0.9 \times \text{pre noise} + w \\
\quad \text{and} \ w \sim \text{normal}(0, \text{var})
\end{cases}
\]

\[
\forall n : \begin{cases}
\text{observed}(\text{none}) \\
\quad \text{noise}_n = 0.9 \times \text{noise}_{n-1} + \text{w}_n \\
\quad \text{w}_n \sim \mathcal{N}(0, \text{var}) \text{ and } w \text{ i.i.d.}
\end{cases}
\]

- Prior distribution of $w$ is i.i.d. (independent identically distributed) with distribution as specified;
- $S2$: probabilistic model for noise it is a time series AR(1)
Example of programming we would like to support

Semantics of S1 and S2 and S3 and S4:

\[
\begin{align*}
S1 & : \\
& \begin{cases}
\text{system System}(y0) \\
\quad \text{observed } u \\
\quad \text{and init } y = y0 \\
\quad \text{and } v = u + \text{pre } y \\
\quad \text{and } y = \text{if fail then } v + \text{noise else } v \\
\end{cases}
\end{align*}
\]

\[
\begin{align*}
S2 & : \\
& \begin{cases}
\text{system Noisegen}(\text{var}) \\
\quad \text{init } \text{noise} = 0.0 \\
\quad \text{and } \text{noise} = 0.9 \times \text{pre } \text{noise} + \text{w} \\
\quad \text{and } \text{w} \sim \text{normal}(0, \text{var}) \\
\end{cases}
\end{align*}
\]

\[
\begin{align*}
S3 & : \\
& \begin{cases}
\text{system Failure}(p) \\
\quad \text{init } \text{backup} = \text{false} \\
\quad \text{and } \text{fail} = \text{rootfail} \wedge \neg \text{pre } \text{backup} \\
\quad \text{and } \text{rootfail} \sim \text{Bernoulli}(p) \\
\end{cases}
\end{align*}
\]

\[
\begin{align*}
S4 & : \\
& \begin{cases}
\text{system Sensor} \\
\quad \text{observed } y \\
\end{cases}
\end{align*}
\]
Example of programming we would like to support

Semantics of S1 and S2 and S3 and S4:

\[
\begin{align*}
\text{observed}(u, y) \\
\text{init}(u, y, \text{backup}) &= (u_0, y_0, \text{backup}_0) \\
v &= u + \text{pre } y \\
y &= \text{if } \text{fail } \text{then } v + \text{noise } \text{else } v \\
\text{noise} &= 0.9 \times \text{pre noise } + w \\
\text{fail} &= \text{rootfail } \land \neg \text{pre backup} \\
(w, \text{rootfail}) &\sim \mathcal{N}(0, \text{var}) \otimes \text{Bernoulli}(p)
\end{align*}
\]

- y not observed $\implies$ a scheduling meeting causality constraints is possible: Bayesian Network
- y observed $\implies$ distribution of $(w, \text{rootfail})$ is conditioned by $(u, y)$ having the values observed: not a Bayesian Network
Example of programming we would like to support

The bottom line:

- Combining constraints and probabilities defines (posterior) conditional distributions

- State variables are visible for interaction

- Probabilistic variables are private and made visible only through the constraints relating them to the state variables

- Putting all of this to maths…
Basic issues in probabilistic paradigms

Approaches
  Statisticians and AI people
  Reactive Programming: ProbZelus

ReactiveBayes minilanguage

Factor Graphs + constraints = Mixed Systems

Putting dynamics: Mixed Automata
  Preliminaries to Mixed Automata
  Mixed Automata

ReactiveBayes and its semantics

Discussion and Comparisons
  Probabilistic Automata
  (Non reactive) Probabilistic Programming
  Reactive Probabilistic Programming

Limitations of Mixed Automata and Fixes

Conclusion
Factor Graphs + constraints = Mixed Systems

[Benveniste, Levy, Fabre, Le Guernic, TCS 1995]
**Factor Graphs + constraints = Mixed Systems**

**Mixed System:** \( S = (\Omega, \pi, X, C) \), where:

- \((\Omega, \pi)\): private probability space (we address only discrete \( \Omega \))
- \(X\) set of **public** variables with domain \( Q = \prod_{x \in X} Q_x \) of states
- \(C \subseteq \Omega \times Q\) is a relation; write \( \omega C q \) iff \( (\omega, q) \in C \)

**Intuition:**

\[
\begin{align*}
\text{draw } \omega \text{ with } & \pi(\cdot \mid \exists q. C) \\
\end{align*}
\]
Factor Graphs + constraints = Mixed Systems

Mixed System: $S = (\Omega, \pi, X, C)$, where:
- $(\Omega, \pi)$: private probability space (we address only discrete $\Omega$)
- $X$ set of public variables with domain $Q = \prod_{x \in X} Q_x$ of states
- $C \subseteq \Omega \times Q$ is a relation; write $\omega C q$ iff $(\omega, q) \in C$

Operational Semantics
- Let $\Omega^c = \{ \omega \in \Omega \mid \exists q : \omega C q \}$ be the consistent subset of $\Omega$; if $\pi(\Omega^c) > 0$ say that $S$ is consistent and define

\[
\pi^c(A) = \text{def } \pi(A \mid \Omega^c) = \frac{\pi(A \cap \Omega^c)}{\pi(\Omega^c)}
\]

- If $S$ is consistent it has an operational semantics called sampling, written $S \leadsto q$ and defined by
  1. draw $\omega$ at random using $\pi^c$
  2. for this $\omega$, select nondeterministically $q$ such that $\omega C q$
**Factor Graphs + constraints = Mixed Systems**

Mixed System: $S = (\Omega, \pi, X, C)$, where:

- $(\Omega, \pi)$: private probability space (we address only discrete $\Omega$)
- $X$ set of public variables with domain $Q = \prod_{x \in X} Q_x$ of states
- $C \subseteq \Omega \times Q$ is a relation; write $\omega C q$ iff $(\omega, q) \in C$
- $\Omega^c = \{\omega \in \Omega | \exists q : \omega C q\}$; $\pi^c = \pi(\cdot | \Omega^c)$, provided $\pi(\Omega^c) > 0$
- $S \rightsquigarrow q =_{\text{def}} \pi^c \rightsquigarrow \omega \mapsto$ some $q : \omega C q$

Compression and Equivalence, see [de Meent & al. 2018]

- Program equivalence is extensively discussed in this ref.; Mixed Systems provide a formalization
- Private space $\Omega$ can be too large $\implies$ compress it: $\omega \sim \omega' : \forall q. \omega C q \iff \omega' C q$; compress to $\bar{\Omega} = \Omega / \sim$, $\bar{\pi}(\bar{\omega}) = \sum_{\omega \in \bar{\omega}} \pi(\omega)$ and compress relation $C$ accordingly
- Equivalence: $S' \equiv S$ iff $\bar{S'}$ and $\bar{S}$ are isomorphic
Factor Graphs + constraints = Mixed Systems

Parallel composition: \( S_1 \times S_2 = (\Omega, \pi, X, C) \), where:

\[
\begin{align*}
\{ & \quad (\Omega, \pi) = (\Omega_1, \pi_1) \otimes (\Omega_2, \pi_2) \text{ (independent)} \\
& \quad X = X_1 \cup X_2 \quad ; \quad C = C_1 \land C_2 \}
\end{align*}
\]

\( S_1' \equiv S_1 \implies S_1' \times S_2 \equiv S_1 \times S_2 \) (\( \equiv \) is a congruence)
Factor Graphs + constraints = Mixed Systems

Parallel composition: \( S_1 \times S_2 = (\Omega, \pi, X, C) \), where:

\[
\begin{align*}
\left\{
\begin{array}{l}
(\Omega, \pi) = (\Omega_1, \pi_1) \otimes (\Omega_2, \pi_2) \text{ (independent)} \\
X = X_1 \cup X_2; \quad C = C_1 \land C_2
\end{array}
\right.
\]

\( S'_1 \equiv S_1 \implies S'_1 \times S_2 \equiv S_1 \times S_2 \) (\( \equiv \) is a congruence)

\( S = S_1 \text{ and } S_2 \text{ and } S_3 \text{ and } S_4 \) (previous values in gray)

\[
\begin{align*}
&\begin{cases}
S_1 : & \text{observed}(u) \\
S_4 : & \text{observed}(y) \\
S_1 : & v = u + \text{pre } y \\
S_1 : & y = \text{if } \text{fail} \text{ then } v + \text{noise} \text{ else } v \\
S_2 : & \text{noise} = 0.9 \times \text{pre } \text{noise} + w \\
S_3 : & \text{fail} = \text{rootfail} \land \neg \text{pre backup} \\
S_2 : & w \sim \mathcal{N}(0, \text{var}) \\
S_3 : & \text{rootfail} \sim \text{Bernoulli}(p)
\end{cases}
\end{align*}
\]

The semantics of \( S \) is the parallel composition of the semantics of \( S_1, S_2, S_3, \) and \( S_4 \)
Factor Graphs + constraints = Mixed Systems

Factor graph of $S = S_1$ and $S_2$ and $S_3$ and $S_4$:

$S_3$ and $S_1$ interact via fail, and similarly for all interactions:

$$
\begin{align*}
S_4 \\
y \\
S_3 &\text{fail} \\
S_1 &\text{noise} \\
S_2
\end{align*}
$$

$$
\begin{align*}
S_1 &: \quad \text{observed}(u) \\
S_4 &: \quad \text{observed}(y) \\
S_1 &: \quad v = u + \text{pre } y \\
S_1 &: \quad y = \text{if } \text{fail} \text{ then } v + \text{noise} \text{ else } v \\
S_2 &: \quad \text{noise} = 0.9 \times \text{pre noise} + w \\
S_3 &: \quad \text{fail} = \text{rootfail} \land \neg \text{pre backup} \\
S_2 &: \quad w \sim \mathcal{N}(0, \text{var}) \\
S_3 &: \quad \text{rootfail} \sim \text{Bernoulli}(p)
\end{align*}
$$

Parallel composition of mixed systems subsumes factor graphs
Bayesian networks from Mixed Systems

Marginal $M_Y(S)$: Let $S = (\Omega, \pi, X, C)$ and $Y \subset X$

1. Define $M_Y(S) =_{\text{def}} (\Omega, \pi, Y, \text{proj}_Y(C))$;

2. Projecting $C$ over $Y$ makes $(\Omega, \pi)$ possibly too large $\Rightarrow$ compress if needed

Subsumes marginals of distributions

Lemma (incremental sampling)

If composing $S_1$ with $S_2$ does not affect $S_1$: $M_{X_1}(S_1 \times S_2) \equiv S_1$, then, $S_1 \times S_2$ can be sampled incrementally, by

1. sampling $S_1 \rightsquigarrow q_1$, and,

2. sampling $S_2$ given $q_1$

formally, by sampling $(X_1 = q_1) \times S_2$.

To indicate incremental sampling, we write $S_1 \overset{\check{\times}}{\times} S_2$
Bayesian networks from Mixed Systems

Bayesian network $\vec{G} = (S, E)$:

- DAG $\vec{G} = (S, E)$, $S$ set of Mixed Systems, $E$ directed edges; let $\preceq$ be the partial order defined by this DAG;
- Say that $\vec{G}$ is a Bayesian network if, for all $S \in S$, the parallel composition $(\prod_{S' \prec S} S') \times S$ is incremental.

Example:

$S3 \rightarrow \text{fail} \rightarrow S1 \leftarrow \text{noise} \leftarrow S2$

\[
\begin{align*}
S1 &: \quad \text{observed}(u) \\
S1 &: \quad v = u + \text{pre } y \\
S1 &: \quad y = \text{if } \text{fail }\text{ then } v + \text{noise } \text{ else } v \\
S2 &: \quad \text{noise} = 0.9 \times \text{pre } \text{noise } + w \\
S2 &: \quad w \sim \mathcal{N}(0, \text{var}) \\
S3 &: \quad \text{fail} = \text{rootfail } \land \neg \text{pre backup} \\
S3 &: \quad \text{rootfail } \sim \text{Bernoulli}(p)
\end{align*}
\]
Bayesian networks from Mixed Systems

Revisiting Bayes’ rule \( p(x, y) = p(y|x) p(x) \)

- Message Passing algorithms from statistics and AI transform
  - tree-shaped Factor Graphs
  - to Bayesian Networks (\( \implies \) incremental sampling)

- Key idea to extend this to Mixed Systems: regard
  - conditional distribution \( p(y|x) \)
  - as a disjunction \( \bigvee_x p(y|x) \)
  (nondeterministic choice among the alternatives for \( x \))
Bayesian networks from Mixed Systems

Disjunction: $\bigvee_{i \in I} S_i$

Nondeterministic choice among the alternatives for $i$.

Disjunction subsumes Transition Probabilities $p(y|x)$
Bayesian networks from Mixed Systems

Disjunction: $\bigvee_{i \in I} S_i$

and define $S \xrightarrow{\cdot} (S_1 \bigvee S_2) = \text{def} (S \xrightarrow{\cdot} S_1) \bigvee (S \xrightarrow{\cdot} S_2)$

Conditional $C_Y(S)$: Let $S = (\Omega, \pi, X, C)$ and $Y \subset X$:

$$C_Y(S) = \text{def} \bigvee_{q_Y : M_Y(S) \rightsquigarrow q_Y} ((Y = q_Y) \times S)$$

where $Y = q_Y$

Theorem (Bayes formula)

$$S \equiv M_Y(S) \xrightarrow{\cdot} C_Y(S)$$
Bayesian networks from Mixed Systems

Corollary (Message passing step)

\[ S = (\Omega, \pi, X, C) \text{ and } S' = (\Omega', \pi', X', C'), \ Y = X \cap X'. \text{ Then:} \]

\[ S' \times S \equiv (S' \times M_Y(S)) \times C_Y(S) \]

Theorem (Message passing algorithm)

Let \( S = \prod_{i \in I} S_i \) be a parallel composition of systems, whose factor graph is a tree. Select a root node of this tree. Applying the message passing step inward, starting from the leaves toward the root, yields a Bayesian network with incremental sampling.

See [Loeliger2004] (An introduction to Factor Graphs) for more info on Factor Graphs, Bayesian Networks, and Message Passing in statistics and signal processing.
Bayesian networks from Mixed Systems

Example: factor graph of $S_1$ and $S_2$ and $S_3$ and $S_4$

$S_4$
  |$
  y$
  |$
S_2$—fail—$S_1$—noise—$S_3$

The message passing algorithm yields the Bayesian network

$C_y(S_4)$
  ↑
  $y$
  ↑

$C_{\text{fail}}(S_3) \leftarrow \text{fail} \leftarrow S_1 \rightarrow \text{noise} \rightarrow C_{\text{noise}}(S_2)$

where

$S_1 = S_1 \times M_{\text{noise}}(S_2) \times M_{\text{fail}}(S_3) \times M_y(S_4)$
Basic issues in probabilistic paradigms

Approaches
Statisticians and AI people
Reactive Programming: ProbZelus

*ReactiveBayes* minilanguage

Factor Graphs + constraints = Mixed Systems

**Putting dynamics: Mixed Automata**

Preliminaries to Mixed Automata
Mixed Automata

*ReactiveBayes and its semantics*

Discussion and Comparisons
Probabilistic Automata
(Non reactive) Probabilistic Programming
Reactive Probabilistic Programming

Limitations of Mixed Automata and Fixes

Conclusion
Preliminaries to Mixed Automata

The idea:

Upgrade: automata $\xrightarrow{}$ probabilistic automata $\xrightarrow{}$ mixed automata:

1. automata: $q \xrightarrow{\alpha} q'$

2. simple probabilistic automata (Segala-Lynch) $q \xrightarrow{\alpha} \pi' \xRightarrow{} q'$

3. mixed automata:
   (to be defined / inherited from Mixed Systems)
   ▶ $q \xrightarrow{\alpha} S' \xRightarrow{} q'$
   ▶ parallel composition, equivalence (via compression)
   ▶ simulation relation
Preliminaries to Mixed Automata

To support \( q \xrightarrow{\alpha} S' \rightsquigarrow q' \), extend Mixed System to Mixed System with previous state \( S = (\Omega, \pi, X, p, C) \):

- \((\Omega, \pi)\): private probability space (we address only denum \( \Omega \))
- \(X\) set of public variables with domain \( Q = \prod_{x \in X} Q_x \) of states
- \(\bullet Q\) a copy of \( Q \), \( p \in \bullet Q \) is a parameter; write \( S(p) \)
- \( C \subseteq \bullet Q \times \Omega \times Q \) is a relation; write \( \omega C q \) iff \((p, \omega, q) \in C\)

- Allows chaining transitions to form runs:
  \[
  q_0 \xrightarrow{1} S_1 \rightsquigarrow q_1 \xrightarrow{2} S_2 \rightsquigarrow q_2 \ldots q_{k-1} \xrightarrow{k} S_k \rightsquigarrow q_k
  \]
Preliminaries to Mixed Automata

To support simulation relations:
Define a simulation relation ≤ on pairs of states

1. Automata:

\[ q_1 \xrightarrow{\alpha} q'_1 \] \[ q_1 \leq q_2 \] \[ \implies \exists q'_2 : \left\{ q_2 \xrightarrow{\alpha} q'_2, q'_1 \leq q'_2 \right\} \]

2. Probabilistic Automata:

\[ q_1 \xrightarrow{\alpha} \pi'_1 \] \[ q_1 \leq q_2 \] \[ \implies \exists \pi'_2 : \left\{ \pi_2 \xrightarrow{\alpha} \pi'_2, \pi'_1 \leq^P \pi'_2 \right\} \left( \sim \implies q'_1 \leq q'_2 \right) \]

\[ \leq^P : \text{lifting of} \leq \text{to pairs of probabilistic states} \]

3. Mixed Automata:

\[ q_1 \xrightarrow{\alpha} S'_1 \] \[ q_1 \leq q_2 \] \[ \implies \exists S'_2 : \left\{ q_2 \xrightarrow{\alpha} S'_2, S'_1 \leq^s S'_2 \right\} \left( \sim \implies q'_1 \leq q'_2 \right) \]
Lifting relations, from pairs of states to pairs of systems

Given a relation $\rho \subseteq Q_1 \times Q_2$, $\rho^S \subseteq S(X_1) \times S(X_2)$ is the lifting of $\rho$ if there exists a weighting function $w : \Omega_1 \times \Omega_2 \rightarrow [0, 1]$ such that:

1. $\forall (\omega_1, \omega_2; q_1) : w(\omega_1, \omega_2) > 0 \implies \exists q_2 : \{ q_1 \rho q_2 \}

2. Weighting function $w$ projects to $\pi_1$ and $\pi_2$:

$$\sum_{\omega_2} w(\omega_1, \omega_2) = \pi_1(\omega_1)$$
$$\sum_{\omega_1} w(\omega_1, \omega_2) = \pi_2(\omega_2)$$

Lemma (equivalence preserves lifting)

$$S_1 \rho^S S_2 \quad \Longleftrightarrow \quad S'_1 \rho^S S_2$$
Mixed Automata

Mixed automaton: $M = (\Sigma, X, q_0, \rightarrow)$ and $q \xrightarrow{\alpha} S' \rightsquigarrow q'$

- $q \xrightarrow{\alpha} S'$ is the transition relation, $\alpha \in \Sigma$; deterministic
- $S'$ has previous state $q$ and $X$ as set of variables
- $S' \rightsquigarrow q'$ is Mixed Systems sampling

Simulation:

$$\begin{align*}
q_1 \xrightarrow{\alpha} S'_1 \\
q_1 \leq q_2
\end{align*} \implies \exists S'_2 : \begin{cases} 
q_2 \xrightarrow{\alpha} S'_2 \\
S'_1 \leq _S S'_2 ( \rightsquigarrow q'_1 \leq q'_2 )
\end{cases}$$

Parallel composition:

$$\begin{align*}
q_i \xrightarrow{\alpha_i} S'_i \\
\alpha_1 \sqcap \alpha_2 \\
\alpha = \alpha_1 \sqcup \alpha_2
\end{align*} \implies M_1 \times M_2 : (q_1, q_2) \xrightarrow{\alpha} S'_1 \times S'_2 \rightsquigarrow (q'_1, q'_2)$$

Thm: $N_1 \leq M_1 \implies N_1 \times M_2 \leq M_1 \times M_2$
Mixed Automata

Inheriting Bayes Calculus from Mixed Systems:

- Mixed Automata inherit, from Mixed Systems, Bayes Calculus in space; transition relations capture factor graphs and Bayesian networks.

- Mixed Automata, however, remain a causal model: the current transition depends on the past, not on the future.

- Consequently, Mixed Automata cannot be used to specify smoothing problems in time
  - e.g., estimating $z_k$ based on $X_0, \ldots, X_k, \ldots, X_N$

To overcome this, we must unfold time as space.
Basic issues in probabilistic paradigms

Approaches

Statisticians and AI people
Reactice Programming: ProbZelus

ReactiveBayes minilanguage

Factor Graphs + constraints = Mixed Systems

Putting dynamics: Mixed Automata
Preliminaries to Mixed Automata
Mixed Automata

ReactiveBayes and its semantics

Discussion and Comparisons

Probabilistic Automata
(Non reactive) Probabilistic Programming
Reactive Probabilistic Programming

Limitations of Mixed Automata and Fixes

Conclusion
**Stateless fragment of ReactiveBayes**

Syntax:

\[
e ::= c \mid x \mid \omega \mid (e, e) \mid \text{op}(e) \mid f(e)
\]

\[
e_v ::= c \mid x \mid (e_v, e_v) \mid \text{op}(e_v) \mid f(e_v)
\]

\[
S ::= \omega \sim P(e_v) \mid e = e \mid \text{observe } x \mid S \text{ and } S
\]

\[P(e_v): \text{proba distributions parameterized with visible expressions}\]

Semantics:

\[
\begin{align*}
\left[\text{observe } x\right] & = (\cdot, \cdot, \{x\}, x = c) \\
\left[\omega \sim P\right] & = (\Omega, P, \{x_\omega\}, x_\omega = \omega) \\
\left[\omega \sim P(e_v(x_1, \ldots, x_p))\right] & = \bigvee_{e_v(x_1, \ldots, x_p) \leftarrow c} \left[\omega \sim P(c)\right] \\
\left[e(x_1, \ldots, x_p) = e'(x_{p+1}, \ldots, x_{p+m})\right] & = (\cdot, \cdot, \{x_1, \ldots, x_{p+m}\}, e = e') \\
\left[S_1 \text{ and } S_2\right] & = \left[S_1\right] \times \left[S_2\right]
\end{align*}
\]

Factor graph of \(S\) follows.
Stateless fragment of *ReactiveBayes*

Compilation to extended Bayesian Networks

- Transition relation compiled to extended Bayesian Network $\vec{G}$:

  \[
  S \text{ is } \text{observe } x \vdash \vec{G} [S] = \{ \text{is-source}(x) \}
  \]

  \[
  S \text{ is } \omega \sim P \vdash \vec{G} [S] = \{ x_\omega \}
  \]

  \[
  S \text{ is } \omega \sim P(e_v(x_1, \ldots, x_p)) \vdash \vec{G} [S] = \{ x_1, \ldots, x_p \} \rightarrow S \rightarrow x_\omega
  \]

  \[
  S \text{ is } x = e(x_1, \ldots, x_p) \vdash \vec{G} [S] = \{ x_1, \ldots, x_p \} \rightarrow S \rightarrow x
  \]

  \[
  S \text{ is } S_1 \text{ and } S_2 \vdash \vec{G} [S] = \vec{G} [S_1] \cup \vec{G} [S_2]
  \]

- The compilation succeeds if $\vec{G}$ is circuitfree.

- Resulting scheduling encodes incremental sampling.
Semantics of ReactiveBayes

Syntax (in red: additional items)

\[
e ::= c | x | \omega | (e, e) | \text{op}(e) | f(e) | \text{pre } x \\
e_v ::= c | x | (e_v, e_v) | \text{op}(e_v) | f(e_v) | \text{pre } x \\
S ::= \omega \sim P(e_v) | e = e | \text{observe } x | S \text{ and } S | \text{init } x = c \\
\alpha ::= "true" \text{ occurrences of } e_v \text{ of type Bool} \\
A ::= \text{on } \alpha \text{ then } S \text{ else } S | \text{init } x = c | A \text{ and } A
\]

Semantics (for additional items)

\[
\left[\text{pre } x\right] = \bigvee_{\text{pre } x \leftarrow c} (\cdot, \cdot, \{x\}, c, \cdot) \\
\left[\text{init } x = c\right] = (\emptyset, \{x\}, c, \emptyset) \\
\left[\text{on then } S_1 \text{ else } S_2 \right] = S_1 \text{ and } S_2 \text{ have previous state } p \left(\{\alpha, \neg \alpha\}, X_1 \cup X_2, \cdot\right) \\
\left[A_1 \text{ and } A_2\right] = [A_1] \times [A_2]
\]
Basic issues in probabilistic paradigms

Approaches
- Statisticians and AI people
- Reactive Programming: ProbZelus

ReactiveBayes minilanguage

Factor Graphs + constraints = Mixed Systems

Putting dynamics: Mixed Automata
- Preliminaries to Mixed Automata
- Mixed Automata

ReactiveBayes and its semantics

Discussion and Comparisons
- Probabilistic Automata
  (Non reactive) Probabilistic Programming
- Reactive Probabilistic Programming

Limitations of Mixed Automata and Fixes

Conclusion
Comparison with Probabilistic Automata

Probabilistic Automata:

[Segala94],[Segala & Lynch03], tutorial [Sokolova & de Vink04]

\[ P = (\Sigma, Q, q_0, \rightarrow) \]

Simple PA : \( \rightarrow \subseteq Q \times \Sigma \times \mathcal{P}(Q) \)

PA : \( \rightarrow \subseteq Q \times \mathcal{P}(\Sigma \times Q) \)

Theorem (SPA, PA, and Mixed Automata (MA))

1. \( \exists \) mapping: SPA \( \rightarrow \) MA, preserving simulation and product.
   \( \exists \) reverse mapping: MA \( \rightarrow \) SPA, preserving simulation.
   \( \not\exists \) reverse mapping preserving parallel composition.

2. \( \exists \) mapping PA \( \rightarrow \) MA, preserving simulation.
   Parallel composition, however, is not preserved.
Comparison with Probabilistic Automata

What is the problem with parallel composition?

- **SPA → ⊆ Q × Σ × ℙ(Q):**
  - Synchronization and parallel composition interleave
    ⇒ no difficulty for defining parallel composition;
  - however, impossible to describe any interaction among probability spaces ⇒ low expressive power.

- **PA → ⊆ Q × ℙ(Σ × Q):**
  - more expressive; supports nondeterminism; action α is probabilistically chosen;
  - conflict between (1) independent probabilistic choice of actions in each component, and (2) the need for synchronizing actions;
  - various ways of solving this conflict; most authors add a scheduler giving hand to one component that is itself randomly selected. Not appropriate for probabilistic programming.
(Non reactive) Probabilistic Programming

[J-W. van de Meent, B. Paige, H. Yang, F. Wood: An introduction to probabilistic programming, 2018]

- Mapping probabilistic programs to graphical models
  - Bayesian Networks (≈ functions)
  - or Factor Graphs (≈ relations), depending on source language
- Long discussions about program equivalence (Sect. 3.1)
- Covers many extensions including recursion in language (mainly targeting causal stochastic processes)

Our positioning with Mixed Systems:

++ Our semantics blends Bayesian Networks and Factor Graphs
++ Equivalence through compression
  - No recursion; reactive covered (Mixed Automata)
  - Supporting continuous proba distributions is technical
Reactive Probabilistic Programming

ProbZelus [Baudart & al. 2020]

- Reactive Proba Programs $\approx$ Bayesian Network + functions
- No direct modeling of Factor Graphs; indirect support through the observe primitive, specifying inference problems

Our positioning with Mixed Automata:

- We support both Bayesian Networks and Factor Graphs
- Equivalence through compression
- Mixed Automata $\subset$ ProbZelus if we use only
Basic issues in probabilistic paradigms

Approaches
  Statisticians and AI people
  Reactive Programming: ProbZelus

*ReactiveBayes* minilanguage

Factor Graphs + constraints = Mixed Systems

Putting dynamics: Mixed Automata
  Preliminaries to Mixed Automata
  Mixed Automata

*ReactiveBayes* and its semantics

Discussion and Comparisons
  Probabilistic Automata
  (Non reactive) Probabilistic Programming
  Reactive Probabilistic Programming

Limitations of Mixed Automata and Fixes

Conclusion
Limitations of Mixed Automata and Fixes

Continuous Probability distributions

- Doable but technical
- Difficulty: interaction between \((\Omega, \pi)\) and \(C\) via conditioning: quite often \(\pi(\Omega^c) = 0\) for seemingly consistent models, which forbids the naive block \(\pi^c(A) = \frac{\pi(A \cap \Omega^c)}{\pi(\Omega^c)}\) used in sampling
  - Example: \((X, Y) \sim \pi\) with \(C\) stating that \(Y\) is observed \(\implies\) fixes the value of \(Y\), which has a zero probability

Addressing this difficulty (in progress) by using

- Conditional expectations \(E(f | G)\) where \(f : \Omega \to \mathbb{R}_+\) and \(G\) a sub-\(\sigma\)-algebra of \(\mathcal{F}\), the \(\sigma\)-algebra on \(\Omega\) (always exist)
- Regular versions of conditional expectations, which exist in restricted cases (covering usual needs)
  - Example: transition probability \(P(y, X)\)
Limitations of Mixed Automata and Fixes

Handling constraints

- **Sampling**: computing \( \{q \mid \omega Cq\} \)
- **Compressions**: computing \( \omega \sim \omega' : \forall q. \omega Cq \text{ iff } \omega' Cq \)
- **Projections**: computing \( \text{proj}_Y(C) \)

Addressing this difficulty (TBD) by using abstractions:

- **Sampling**: restrict computing \( \{q \mid \omega Cq\} \) to easy cases (Boolean); otherwise make sure that \( \omega \mapsto q \) is a function and use causality graphs
- **Compressions**: develop sufficient conditions showing \( S' \equiv S \)
- **Projections**: use graph based algorithms as abstractions
Basic issues in probabilistic paradigms

Approaches
  Statisticians and AI people
  Reactive Programming: ProbZelus

ReactiveBayes minilanguage

Factor Graphs + constraints = Mixed Systems

Putting dynamics: Mixed Automata
  Preliminaries to Mixed Automata
  Mixed Automata

ReactiveBayes and its semantics

Discussion and Comparisons
  Probabilistic Automata
  (Non reactive) Probabilistic Programming
  Reactive Probabilistic Programming

Limitations of Mixed Automata and Fixes

Conclusion
Conclusion

Mixed Systems

► They subsume factor graphs, Bayesian networks, and constraints
► They come with equivalence, parallel composition, and the lifting of relations from states to systems

Mixed Automata

► On top of Mixed Systems
► They subsume Probabilistic Automata regarding simulation relations and Simple PA regarding parallel composition
► The parallel composition of PA differs from ours and does not allow for describing interactions between probabilistic parts
Conclusion

Support for Probabilistic Programming?

▶ Mixed Systems provide the needed concepts for basic probabilistic programming (without recursion)

▶ Mixed Automata provide the needed concepts for reactive probabilistic programming

Further work

▶ Define abstractions to make effective MS and MA basic properties and operations, thus ensuring that they can become compilation steps