Bayesian Synthesis of Probabilistic Programs for Automatic Data Modeling

Feras Saad, Marco Cusumano-Towner, Ulrich Schaechtle, Martin Rinard, and Vikash Mansinghka

Computer Science & Artificial Intelligence Laboratory
Massachusetts Institute of Technology

POPL 2019
what are the most salient patterns in this data?
what are the most salient patterns in this data?

overall linear trend
what are the most salient patterns in this data?

- overall linear trend
- periodically recurring large peak with growing amplitude
what are the most salient patterns in this data?

- overall linear trend
- periodically recurring large peak with growing amplitude
- periodically recurring small peak (after the large peak)
what are the most salient patterns in this data?

- overall linear trend
- periodically recurring large peak with growing amplitude
- periodically recurring small peak (after the large peak)
- periodically recurring small peak (before the large peak)
what are the most salient patterns in this data?

- overall linear trend
- periodically recurring large peak with growing amplitude
- periodically recurring small peak (after the large peak)
- periodically recurring small peak (before the large peak)

can we forecast future data?
what are the most salient patterns in this data?

- overall linear trend
- periodically recurring large peak with growing amplitude
- periodically recurring small peak (after the large peak)
- periodically recurring small peak (before the large peak)

can we forecast future data?
what are the most salient patterns in this data?

- overall linear trend
- periodically recurring large peak with growing amplitude
- periodically recurring small peak (after the large peak)
- periodically recurring small peak (before the large peak)

Global Airline Passenger Volume (Thousands) Between 1948 and 1962

can we forecast future data?
time series data

what patterns exist in the data?

what are the values for new time points?
time series data

what patterns exist in the data?

what are the values for new time points?
time series data

what patterns exist in the data?
what are the values for new time points?

- manual labor
- time consuming
- costly to hire a data scientist
- needs lots of expertise
time series data

what patterns exist in the data?

what are the values for new time points?

- manual labor
- time consuming
- costly to hire a data scientist
- needs lots of expertise
probabilistic programs that are likely to have generated the data

what patterns exist in the data?

what are the values for new time points?
time series data

\[ y \]

\[ x \]

space of probabilistic programs \( P \) that can generate time series data

probabilistic programs that are likely to have generated the data

what patterns exist in the data?

what are the values for new time points?
time series data

space of probabilistic programs $P$ that can generate time series data

Prior[$P$] prior distribution over programs $P$

what patterns exist in the data?

what are the values for new time points?

probabilistic programs that are likely to have generated the data
What patterns exist in the data?

What are the values for new time points?

Space of probabilistic programs \( P \) that can generate time series data

Prior\( [P] \) prior distribution over programs \( P \)

Likelihood\( [P](x, y) \) Probability that program \( P \) returns output \( y \) given input \( x \)

Probabilistic programs that are likely to have generated the data
space of probabilistic programs $P$ that can generate time series data

$\text{Prior}[P]$

prior distribution over programs $P$

$\text{Likelihood}[P](x, y)$

Probability that program $P$ returns output $y$ given input $x$

$\text{Posterior}[P](x, y)$

distribution over programs $P$ conditioned on data $(x, y)$

probabilistic programs that are \textbf{likely} to have generated the data

what patterns exist in the data?

what are the values for new time points?
time series data

space of probabilistic programs $P$ that can generate time series data

Prior$[P]$ prior distribution over programs $P$

Likelihood$[P](x, y)$ Probability that program $P$ returns output $y$ given input $x$

Bayesian Synthesis

Posterior$[P](x, y)$ distribution over programs $P$ conditioned on data $(x,y)$

probabilistic programs that are likely to have generated the data

syntactic analysis

what patterns probably exist in the data?

what are the values for new time points?
Probabilistic inference

Time series data

Bayesian Synthesis

space of probabilistic programs $P$ that can generate time series data

Prior$[P]$ prior distribution over programs $P$

Likelihood$[P](x, y)$ Probability that program $P$ returns output $y$ given input $x$

Posterior$[P](x, y)$ distribution over programs $P$ conditioned on data $(x, y)$

Probabilistic programs that are likely to have generated the data

Syntactic analysis

What patterns probably exist in the data?

Probabilistic inference

What are the probable values for new time points?
Key contributions

- First research to introduce fully-Bayesian synthesis of probabilistic programs.
- Rigorously formalized approach using denotational semantics.
- Proved sufficient conditions for synthesis problem to be probabilistically well-formed.
- Defined a Bayesian synthesis algorithm applicable to any DSL generated by a PCFG.
- Proved that synthesis algorithm is sound:
  - sampler for programs converges a.s. in total variation to the analytic posterior.
- Implemented two Bayesian synthesis systems:
  - time series data (this presentation)
  - multivariate data tables (see paper)
- Experimental results and comparisons on multiple real-world problems.
space of probabilistic programs $P$ that can generate time series data

Prior $[P]$ 

prior distribution over programs $P$

Likelihood $[P](x, y)$ 

Probability that program $P$ returns output $y$ given input $x$

Posterior $[P](x, y)$ 

distribution over programs $P$ conditioned on data $(x, y)$

probabilistic programs that are likely to have generated the data

Bayesian Synthesis

syntactic analysis

what patterns probably exist in the data?

probabilistic inference

what are the probable values for new time points?
To discover patterns and make forecasts we need a time series model
To discover patterns and make forecasts we need a time series model

- The Gaussian process (GP) is a rich family for modeling time series with widely-varying patterns.
To discover patterns and make forecasts we need a time series model

- The **Gaussian process (GP)** is a rich family for modeling time series with widely-varying patterns.
- A GP is a *random function* $f : [X] \rightarrow [Y]$ which takes list of $X$ and randomly samples a list of $Y$. 
To discover patterns and make forecasts we need a time series model

- The Gaussian process (GP) is a rich family for modeling time series with widely-varying patterns.
- A GP is a random function $f : [X] \rightarrow [Y]$ which takes list of $X$ and randomly samples a list of $Y$.

Standard Function: Fixed Input-Output Behavior
To discover patterns and make forecasts we need a time series model

- The **Gaussian process (GP)** is a rich family for modeling time series with widely-varying patterns.

- A GP is a *random function* \( f : [X] \rightarrow [Y] \) which takes list of \( X \) and randomly samples a list of \( Y \).

**Standard Function: Fixed Input-Output Behavior**

\[
[x_1, \ldots, x_n] \rightarrow f \rightarrow [f(x_1), \ldots, f(x_n)]
\]
To discover patterns and make forecasts we need a time series model

- The **Gaussian process (GP)** is a rich family for modeling time series with widely-varying patterns.
- A GP is a *random function* \( f : [X] \to [Y] \) which takes list of \( X \) and randomly samples a list of \( Y \).

**Standard Function: Fixed Input-Output Behavior**

\[
\begin{align*}
[x_1, \ldots, x_n] & \quad \rightarrow \quad f \quad \rightarrow \quad [f(x_1), \ldots, f(x_n)]
\end{align*}
\]
To discover patterns and make forecasts we need a time series model

- The **Gaussian process (GP)** is a rich family for modeling time series with widely-varying patterns.
- A GP is a *random function* \( f : [X] \rightarrow [Y] \) which takes list of \( X \) and randomly samples a list of \( Y \).

**Standard Function: Fixed Input-Output Behavior**

**Random Function: Stochastic Input-Output Behavior**

\[
\begin{align*}
[X_1, \ldots, X_n] & \quad \rightarrow \quad f \\
[f(X_1), \ldots, f(X_n)] & \quad \rightarrow \quad f
\end{align*}
\]
To discover patterns and make forecasts we need a time series model

- The **Gaussian process (GP)** is a rich family for modeling time series with widely-varying patterns.
- A GP is a *random function* \( f : [X] \rightarrow [Y] \) which takes list of \( X \) and randomly samples a list of \( Y \).

**Standard Function: Fixed Input-Output Behavior**

**Random Function: Stochastic Input-Output Behavior**

\[
\begin{align*}
[X_1, \ldots, X_n] & \quad \rightarrow \quad f \\
& \quad \rightarrow \quad [f(x_1), \ldots, f(x_n)]
\end{align*}
\]

\[
\begin{align*}
[X_1, \ldots, X_n] & \quad \rightarrow \quad f \\
& \quad \rightarrow \quad [f(x_1), \ldots, f(x_n)]
\end{align*}
\]
To discover patterns and make forecasts we need a time series model

- The **Gaussian process (GP)** is a rich family for modeling time series with widely-varying patterns.

- A GP is a *random function* \( f : [X] \rightarrow [Y] \) which takes list of \( X \) and randomly samples a list of \( Y \).

**Standard Function: Fixed Input-Output Behavior**

**Random Function: Stochastic Input-Output Behavior**
To discover patterns and make forecasts we need a time series model

- The **Gaussian process (GP)** is a rich family for modeling time series with widely-varying patterns.
- A GP is a *random function* $f : [X] \rightarrow [Y]$ which takes list of $X$ and randomly samples a list of $Y$.

**Standard Function: Fixed Input-Output Behavior**

$$[x_1, \ldots, x_n] \xrightarrow{f} [f(x_1), \ldots, f(x_n)]$$

**Random Function: Stochastic Input-Output Behavior**

$$[x_1, \ldots, x_n] \xrightarrow{f} [f(x_1), \ldots, f(x_n)]$$

![Graph of a Gaussian process](image)
To discover patterns and make forecasts we need a time series model

- The **Gaussian process (GP)** is a rich family for modeling time series with widely-varying patterns.
- A GP is a *random function* \( f : [X] \rightarrow [Y] \) which takes list of \( X \) and randomly samples a list of \( Y \).

**Standard Function: Fixed Input-Output Behavior**

\[
\begin{align*}
[X_1, \ldots, X_n] & \rightarrow f & [f(X_1), \ldots, f(X_n)]
\end{align*}
\]

**Random Function: Stochastic Input-Output Behavior**

\[
\begin{align*}
[X_1, \ldots, X_n] & \rightarrow f & [f(X_1), \ldots, f(X_n)]
\end{align*}
\]
To discover patterns and make forecasts we need a time series model

- The **Gaussian process (GP)** is a rich family for modeling time series with widely-varying patterns.
- A GP is a *random function* $f : [X] \rightarrow [Y]$ which takes list of $X$ and randomly samples a list of $Y$.

**Standard Function: Fixed Input-Output Behavior**

**Random Function: Stochastic Input-Output Behavior**
Why are Gaussian processes useful for modeling time series data?

- Statisticians use Gaussian processes to model a wide range of patterns.

- Key idea: the covariance $K$ dictates the “shape” of the time series.
  
  (Duvenaud et al. ICML 2013; Lloyd et al. AAAI 2014)
Why are Gaussian processes useful for modeling time series data?

- Statisticians use Gaussian processes to model a wide range of patterns.

- Key idea: the covariance $K$ dictates the “shape” of the time series.

(Duvenaud et al. ICML 2013; Lloyd et al. AAAI 2014)

**Primitive Covariances**

![Primitive Covariances Diagram]
Why are Gaussian processes useful for modeling time series data?

- Statisticians use Gaussian processes to model a wide range of patterns.

- Key idea: the covariance $K$ dictates the “shape” of the time series.

(Duvenaud et al. ICML 2013; Lloyd et al. AAAI 2014)

**Primitive Covariances**

<table>
<thead>
<tr>
<th>SE</th>
<th>LIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>PER</td>
<td>RQ</td>
</tr>
</tbody>
</table>

**Composition Rules**

<table>
<thead>
<tr>
<th>LIN \times LIN</th>
<th>LIN \times PER</th>
<th>SE \times PER</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIN \times SE</td>
<td>LIN + PER</td>
<td>SE + PER</td>
</tr>
</tbody>
</table>
Why are Gaussian processes useful for modeling time series data?

- Statisticians use Gaussian processes to model a wide range of patterns.

- Key idea: the covariance $K$ dictates the “shape” of the time series.
  (Duvenaud et al. ICML 2013; Lloyd et al. AAAI 2014)
Why are Gaussian processes useful for modeling time series data?

- Statisticians use Gaussian processes to model a wide range of patterns.

- Key idea: the covariance $K$ dictates the “shape” of the time series.
  (Duvenaud et al. ICML 2013; Lloyd et al. AAAI 2014)

- Given a Gaussian process model for a time series, we can:
  - **Discover patterns** in the time series by inspecting the structure of the covariance $K$.
  - **Make predictions** for new data by using probabilistic inference.
A brief intuition for how Gaussian processes work

- Each time we invoke \( f \), it returns a randomly sampled curve.
A brief intuition for how Gaussian processes work

- Each time we invoke $f$, it returns a randomly sampled curve.

- Formally, for any list of inputs $[x_1, ..., x_n]$
  the outputs are jointly normal:

  $[f(x_1), ..., f(x_n)] \sim \text{Multivariate-Normal}(0, [K(x_i, x_j)]_{ij})$
A brief intuition for how Gaussian processes work

- Each time we invoke $f$, it returns a randomly sampled curve.

- Formally, for any list of inputs $[x_1, ..., x_n]$ the outputs are jointly normal:
  
  $$[f(x_1), ..., f(x_n)] \sim \text{Multivariate-Normal}(0, [K(x_i, x_j)])_{ij}$$

- We write $f \sim \text{GP}(K)$ to emphasize that the GP is parameterized by the covariance $K : X^2 \rightarrow \text{Reals}$. 
A brief intuition for how Gaussian processes work

- Each time we invoke \( f \), it returns a randomly sampled curve.

- Formally, for any list of inputs \([x_1, \ldots, x_n]\),
the outputs are jointly normal:

\[
[f(x_1), \ldots, f(x_n)] \sim \text{Multivariate-Normal}(0, \{K(x_i, x_j)\}_{ij})
\]

- We write \( f \sim \text{GP}(K) \) to emphasize that
the GP is parameterized by the
\textbf{covariance} \( K \): \( X^2 \rightarrow \text{Reals} \).

- For any pair of inputs \((x, x')\) we have
\( K(x, x') := \text{covariance of } f(x) \text{ and } f(x') \)
A brief intuition for how Gaussian processes work

- Each time we invoke $f$, it returns a randomly sampled curve.

- Formally, for any list of inputs $[x_1, \ldots, x_n]$ the outputs are jointly normal:

  $$ [f(x_1), \ldots, f(x_n)] \sim \text{Multivariate-Normal}(0, [K(x_i, x_j)]_{ij}) $$

- We write $f \sim \text{GP}(K)$ to emphasize that the GP is parameterized by the covariance $K$: $X^2 \to \text{Reals}$.

- For any pair of inputs $(x, x')$ we have $K(x, x') := \text{covariance of } f(x) \text{ and } f(x')$
A brief intuition for how Gaussian processes work

- Each time we invoke $f$, it returns a randomly sampled curve.

- Formally, for any list of inputs $[x_1, ..., x_n]$ the outputs are jointly normal:
  
  $$[f(x_1), ..., f(x_n)] \sim \text{Multivariate-Normal}(0, [K(x_i, x_j)]_{ij})$$

- We write $f \sim \text{GP}(K)$ to emphasize that the GP is parameterized by the covariance $K: X^2 \to \text{Reals}$.

- For any pair of inputs $(x, x')$ we have $K(x, x') := \text{covariance of } f(x) \text{ and } f(x')$.

values of $f(x)$ and $f(x')$ on execution 1
A brief intuition for how Gaussian processes work

- Each time we invoke $f$, it returns a randomly sampled curve.

- Formally, for any list of inputs $[x_1, ..., x_n]$ the outputs are jointly normal:

$$[f(x_1), ..., f(x_n)] \sim \text{Multivariate-Normal}(0, [K(x_i, x_j)]_{ij})$$

- We write $f \sim \text{GP}(K)$ to emphasize that the GP is parameterized by the covariance $K: \mathbb{X}^2 \rightarrow \mathbb{Reals}$.

- For any pair of inputs $(x, x')$ we have $K(x, x') := \text{covariance of } f(x) \text{ and } f(x')$. 

\begin{itemize}
  \item Values of $f(x)$ and $f(x')$ on execution 1
  \item Values of $f(x)$ and $f(x')$ on execution 2
\end{itemize}
A brief intuition for how Gaussian processes work

- Each time we invoke $f$, it returns a randomly sampled curve.

- Formally, for any list of inputs $[x_1, ..., x_n]$ the outputs are jointly normal:

$$[f(x_1), ..., f(x_n)] \sim \text{Multivariate-Normal}(0, [K(x_i, x_j)]_{ij})$$

- We write $f \sim \text{GP}(K)$ to emphasize that the GP is parameterized by the covariance $K$: $X^2 \rightarrow \text{Reals}$.

- For any pair of inputs $(x, x')$ we have $K(x, x') := \text{covariance of } f(x) \text{ and } f(x')$.
A brief intuition for how Gaussian processes work

- Each time we invoke \( f \), it returns a randomly sampled curve.

- Formally, for any list of inputs \([x_1, \ldots, x_n]\) the outputs are jointly normal:

\[
[f(x_1), \ldots, f(x_n)] \sim \text{Multivariate-Normal}(0, [K(x_i, x_j)]_{ij})
\]

- We write \( f \sim \text{GP}(K) \) to emphasize that the GP is parameterized by the covariance \( K: X^2 \rightarrow \text{Reals} \).

- For any pair of inputs \((x, x')\) we have \( K(x, x') := \text{covariance of } f(x) \text{ and } f(x') \).
Representing Gaussian process models using probabilistic code

**Program inputs**

\[ [x_1, \ldots, x_n] \]

**Probabilistic program for a Gaussian process** \( f \)

```
make_gaussian_process() -> {0},
  ((x1,x2) -> {\exp(x1-x2)**2/1.00000})
```

**Samples of time series from the Gaussian process**

\[ [f(x_1), \ldots, f(x_n)] \]

**Program inputs**

\[ [x_1, \ldots, x_n] \]

**Probabilistic program for a Gaussian process** \( f \)

```
make_gaussian_process() -> {0},
  ((x1,x2) -> {((x1-0.1000)+(x2-0.1000))((x1,x2)
    + ((x1,x2) -> {-2/360.00000*\sin(2*pi/3.00000
    + abs(x1-x2)+2))((x1,x2))})
```

**Samples of time series from the Gaussian process**

\[ [f(x_1), \ldots, f(x_n)] \]

**Program inputs**

\[ [x_1, \ldots, x_n] \]

**Probabilistic program for a Gaussian process** \( f \)

```
make_gaussian_process() -> {0},
  ((x1, x2) -> { if (x1==x2) 0.10000
    else 0)((x1,x2))
    (x1,x2) + sig2 (((x1,x2) -> {-2/100.00000*\sin(2*pi/2.00000
    + abs(x1-x2)+2))((x1,x2))})
```

**Samples of time series from the Gaussian process**

\[ [f(x_1), \ldots, f(x_n)] \]
Representing Gaussian process models using **probabilistic code**

```
program inputs

[x₁, ..., xₙ]
```

```
probabilistic program for a Gaussian process \( f \)

make_gaussian_process() -> {0},
  ((x1,x2) -> {(exp(-(x1-x2)**2/2.00000))})

[x₁, ..., xₙ]
```

```
make_gaussian_process() -> {0},
  ((x1,x2) ->
   ((x1-0.1000)**2 + (x2-0.1000)**2)*
    ((1-sigmoid(x1, 4.00000, 1.0)) +
     (1-sigmoid(x2, 4.00000, 1.0)))
    +
    ((1-sigmoid(x1, 4.50000, 0.1)) +
     (1-sigmoid(x2, 4.50000, 0.1)))
    +
    ((1-\text{sigmoid}(x1,0.1)) * \text{sigmoid}(x2,0.1))
    +
    ((1-\text{sigmoid}(x2,0.1)) * \text{sigmoid}(x1,0.1))
    +
    ((1-\text{sigmoid}(x1,4.0)) * \text{sigmoid}(x2,4.0))
    +
    ((1-\text{sigmoid}(x2,4.0)) * \text{sigmoid}(x1,4.0))
    +
    ((1-\text{sigmoid}(x1,4.5)) * \text{sigmoid}(x2,4.5))
    +
    ((1-\text{sigmoid}(x2,4.5)) * \text{sigmoid}(x1,4.5))
    +
    ((1-\text{sigmoid}(x1,1.0)) * \text{sigmoid}(x2,1.0))
    +
    ((1-\text{sigmoid}(x2,1.0)) * \text{sigmoid}(x1,1.0))
    +
    ((1-\text{sigmoid}(x1,1.5)) * \text{sigmoid}(x2,1.5))
    +
    ((1-\text{sigmoid}(x2,1.5)) * \text{sigmoid}(x1,1.5))
    +
    ((1-\text{sigmoid}(x1,2.0)) * \text{sigmoid}(x2,2.0))
    +
    ((1-\text{sigmoid}(x2,2.0)) * \text{sigmoid}(x1,2.0))
    +
    ((1-\text{sigmoid}(x1,2.5)) * \text{sigmoid}(x2,2.5))
    +
    ((1-\text{sigmoid}(x2,2.5)) * \text{sigmoid}(x1,2.5))
    +
    ((1-\text{sigmoid}(x1,3.0)) * \text{sigmoid}(x2,3.0))
    +
    ((1-\text{sigmoid}(x2,3.0)) * \text{sigmoid}(x1,3.0))
    +
    ((1-\text{sigmoid}(x1,3.5)) * \text{sigmoid}(x2,3.5))
    +
    ((1-\text{sigmoid}(x2,3.5)) * \text{sigmoid}(x1,3.5))
    +
    ((1-\text{sigmoid}(x1,4.0)) * \text{sigmoid}(x2,4.0))
    +
    ((1-\text{sigmoid}(x2,4.0)) * \text{sigmoid}(x1,4.0))
    +
    ((1-\text{sigmoid}(x1,4.5)) * \text{sigmoid}(x2,4.5))
    +
    ((1-\text{sigmoid}(x2,4.5)) * \text{sigmoid}(x1,4.5))
    +
    ((1-\text{sigmoid}(x1,5.0)) * \text{sigmoid}(x2,5.0))
    +
    ((1-\text{sigmoid}(x2,5.0)) * \text{sigmoid}(x1,5.0))
    +
    ((1-\text{sigmoid}(x1,5.5)) * \text{sigmoid}(x2,5.5))
    +
    ((1-\text{sigmoid}(x2,5.5)) * \text{sigmoid}(x1,5.5))
    +
    ((1-\text{sigmoid}(x1,6.0)) * \text{sigmoid}(x2,6.0))
    +
    ((1-\text{sigmoid}(x2,6.0)) * \text{sigmoid}(x1,6.0))
    +
    ((1-\text{sigmoid}(x1,6.5)) * \text{sigmoid}(x2,6.5))
    +
    ((1-\text{sigmoid}(x2,6.5)) * \text{sigmoid}(x1,6.5))
    +
    ((1-\text{sigmoid}(x1,7.0)) * \text{sigmoid}(x2,7.0))
    +
    ((1-\text{sigmoid}(x2,7.0)) * \text{sigmoid}(x1,7.0))
    +
    ((1-\text{sigmoid}(x1,7.5)) * \text{sigmoid}(x2,7.5))
    +
    ((1-\text{sigmoid}(x2,7.5)) * \text{sigmoid}(x1,7.5))
    +
    ((1-\text{sigmoid}(x1,8.0)) * \text{sigmoid}(x2,8.0))
    +
    ((1-\text{sigmoid}(x2,8.0)) * \text{sigmoid}(x1,8.0))
    +
    ((1-\text{sigmoid}(x1,8.5)) * \text{sigmoid}(x2,8.5))
    +
    ((1-\text{sigmoid}(x2,8.5)) * \text{sigmoid}(x1,8.5))
    +
    ((1-\text{sigmoid}(x1,9.0)) * \text{sigmoid}(x2,9.0))
    +
    ((1-\text{sigmoid}(x2,9.0)) * \text{sigmoid(x1,9.0)))

[x₁, ..., xₙ]
```

```
samples of time series from the Gaussian process

[f(x₁), ..., f(xₙ)]
```

```
[f(x₁), ..., f(xₙ)]
```

```
[f(x₁), ..., f(xₙ)]
```

```
[f(x₁), ..., f(xₙ)]
```

```
[f(x₁), ..., f(xₙ)]
```

```
[f(x₁), ..., f(xₙ)]
```

```
[f(x₁), ..., f(xₙ)]
```

```
[f(x₁), ..., f(xₙ)]
```

```
[f(x₁), ..., f(xₙ)]
```

```
[f(x₁), ..., f(xₙ)]
```

```
[f(x₁), ..., f(xₙ)]
```

```
[f(x₁), ..., f(xₙ)]
```

```
[f(x₁), ..., f(xₙ)]
```

```
[f(x₁), ..., f(xₙ)]
```
Representing Gaussian process models using **probabilistic code** a DSL

**Program inputs**

1. \([x_1, ..., x_n]\)
2. \([x_1, ..., x_n]\)
3. \([x_1, ..., x_n]\)

**Probabilistic DSL program for a Gaussian process \(f\)**

1. \((\text{SMOOTH 3.4})\)
2. \((\ast (\text{PERIODIC 60 3}) (\text{LINEAR .1}))\)
3. \((\text{CHangepoint 4.5 0.5} (\text{LINEAR 1}) (\text{WHITE-NOISE .01}) (\text{PERIODIC 10 2}))\)

**Samples of time series from the Gaussian process**

\([f(x_1), ..., f(x_n)]\)
\([f(x_1), ..., f(x_n)]\)
\([f(x_1), ..., f(x_n)]\)
Representing Gaussian process models using a DSL

**Syntax of DSL**

\( v \in \text{Numeric} \)

\( H \in \text{Params} ::= (\gamma \ v) \ [\text{Parameter}] \)

\( K \in \text{Covariance} ::= \)

  \( \text{(constant } H) \) \ [\text{Constant}] \)
  \( \text{(white-noise } H) \) \ [\text{WhiteNoise}] \)
  \( \text{(linear } H) \) \ [\text{Linear}] \)
  \( \text{(smooth } H) \) \ [\text{Smooth}] \)

  \( (+ K_1 K_2) \) \ [\text{Sum}] \)
  \( (* K_1 K_2) \) \ [\text{Product}] \)
  \( \text{(changepoint } v K_1 K_2) \) \ [\text{ChangePoint}] \)

\[ \text{primitive covariances} \]

\[ \text{composite covariances} \]

\[ \text{numerical parameters} \]
space of probabilistic programs \( P \) that can generate time series data

Prior\( [P] \) prior distribution over programs \( P \)

Likelihood\( [P](x, y) \) Probability that program \( P \) returns output \( y \) given input \( x \)

Bayesian Synthesis

\( \text{Posterior}[P](x, y) \) distribution over programs \( P \) conditioned on data \((x,y)\)

probabilistic programs that are likely to have generated the data

syntactic analysis

what patterns probably exist in the data?

probabilistic inference

what are the probable values for new time points?
space of probabilistic programs $P$ that can generate time series data

Prior$[P]$: prior distribution over programs $P$

Likelihood$[P](x, y)$: Probability that program $P$ returns output $y$ given input $x$

Posterior$[P](x, y)$: distribution over programs $P$ conditioned on data $(x, y)$

Bayesian Synthesis

probabilistic programs that are likely to have generated the data

syntactic analysis

what patterns probably exist in the data?

probabilistic inference

what are the probable values for new time points?
Q: Why not find DSL program P that maximizes Likelihood[\[P\]](x,y) (i.e. maximize probability of data)?
Q: Why not find DSL program $P$ that maximizes Likelihood $[P](x,y)$ (i.e. maximize probability of data)?
Q: Why not find DSL program $P$ that maximizes Likelihood $[P](x,y)$ (i.e. maximize probability of data)?

A DSL program $P$ that models dataset $(x_1, y_1), \ldots, (x_n, y_n)$ arbitrarily well

$P = (+ \text{ CHANGEPOINT } x_1$

$\text{ (LINEAR } (y_2-y_1)/(x_2-x_1) \ldots \text{)}$

$\text{ CHANGEPOINT } x_2$

$\text{ (LINEAR } (y_3-y_2)/(x_3-x_2) \ldots \text{)}$

$\text{ \ldots}$

$\text{ (LINEAR } (y_n-y_m)/(x_n-x_m) \ldots \text{)})\ldots)$

$\text{ (WHITE-NOISE } \sigma \text{))}

“linearly interpolate between observations and add some noise $\sigma$”

(this program is useless!)
Q: Why not find DSL program $P$ that maximizes $\text{Likelihood}[P](x, y)$ (i.e. maximize probability of data)?

A DSL program $P$ that models dataset $(x_1, y_1), \ldots, (x_n, y_n)$ arbitrarily well:

$$P = (+ \text{ CHANGEPOINT } x_1 \text{ (LINEAR } (y_2-y_1)/(x_2-x_1) \ldots) \text{ (CHANGEPOINT } x_2 \text{ (LINEAR } (y_3-y_2)/(x_3-x_2) \ldots) \text{ (CHANGEPOINT } x_3 \ldots \text{ (LINEAR } (y_n-y_m)/(x_n-x_m) \ldots)))) \text{ (WHITE-NOISE } \sigma))$$

"linearly interpolate between observations and add some noise $\sigma$"

(this program is useless!)

\[\sigma = 0.50\]
Q: Why not find DSL program $P$ that maximizes Likelihood$[P](x,y)$ (i.e. maximize probability of data)?

A DSL program $P$ that models dataset $(x_1, y_1), \ldots, (x_n, y_n)$ arbitrarily well

$$P = (+ \text{ (CHANGEPOINT } x_1 \text{ (LINEAR } (y_2-y_1)/(x_2-x_1) \ldots) \text{ (CHANGEPOINT } x_2 \text{ (LINEAR } (y_3-y_2)/(x_3-x_2) \ldots) \text{ (CHANGEPOINT } x_3 \ldots \text{(LINEAR } (y_n-y_m)/(xn-xm) \ldots))) \text{ (WHITE-NOISE } \sigma))$$

"linearly interpolate between observations and add some noise $\sigma$"

(this program is useless!)

$\sigma = 0.50$

$\sigma = 0.25$

likelihood of data keeps increasing
Q: Why not find DSL program P that maximizes Likelihood[\(\mathbb{P}\)](x,y) (i.e. maximize probability of data)?

A DSL program P that models dataset \((x_1, y_1), \ldots, (x_n, y_n)\) arbitrarily well

\[
P = (+ \text{ (CHANGEPOINT } x_1 \text{ (LINEAR } \frac{y_2-y_1}{x_2-x_1} \ldots) \text{ (CHANGEPOINT } x_2 \text{ (LINEAR } \frac{y_3-y_2}{x_3-x_2} \ldots) \text{ (CHANGEPOINT } x_3 \ldots \text{ (LINEAR } \frac{y_n-y_{n-1}}{x_n-x_{n-1}} \ldots))) \text{ (WHITE-NOISE } \sigma))
\]

“linearly interpolate between observations and add some noise \(\sigma\)”

(this program is useless!)

\(\sigma = 0.50\) \hspace{1cm} \(\sigma = 0.25\) \hspace{1cm} \(\sigma = 0.10\)

likelihood of data keeps increasing
Q: Why not find DSL program $P$ that maximizes Likelihood[$P$]$(x,y)$ (i.e. maximize probability of data)?

A DSL program $P$ that models dataset $(x_1, y_1), \ldots, (x_n, y_n)$ arbitrarily well

$$P = (+ \text{(CHANGEPOINT } x_1 \text{ (LINEAR } (y_2-y_1)/(x_2-x_1) \text{ ...)} \text{(CHANGEPOINT } x_2 \text{ (LINEAR } (y_3-y_2)/(x_3-x_2) \text{ ...)} \text{(CHANGEPOINT } x_3 \text{ ... (LINEAR } (y_n-y_m)/(x_n-x_m) \text{ ...))}) \text{(WHITE-NOISE } \sigma))$$

"linearly interpolate between observations and add some noise $\sigma$"

(this program is useless!)

likelihood of data keeps increasing
Q: Why not find DSL program $P$ that maximizes Likelihood $[P](x,y)$ (i.e. maximize probability of data)?

A DSL program $P$ that models dataset $(x_1, y_1), \ldots, (x_n, y_n)$ arbitrarily well:

$$P = (+ (\text{PERIODIC } 2\pi .5) (\text{WHITE-NOISE } \sigma))$$

"periodic behavior and add some noise $\sigma$"

(this program is useful!)
Q: Why not find DSL program $P$ that maximizes Likelihood$[P](x,y)$ (i.e. maximize probability of data)?

$K = (+ \text{(CHANCEPOINT} x_1 \\
\text{(LINEAR} (y_2-y_1)/(x_2-x_1) ...) \\
\text{(CHANCEPOINT} x_2 \\
\text{(LINEAR} (y_3-y_2)/(x_3-x_2) ...) \\
\text{(CHANCEPOINT} x_3 \\
\text{...)} \\
\text{(LINEAR} (y_n-y_m)/(x_n-x_m) \text{...}) \\
\text{(WHITE-NOISE} \sigma))) \\
\text{(WHITE-NOISE} 0.05))$

**Lower** data likelihood  
**Simple** program  
**Useful** for discovering patterns  
**Useful** for making predictions

**Higher** data likelihood  
**Complex** program  
**Not useful** for discovering patterns  
**Not useful** for making predictions
Q: Why not find DSL program $P$ that maximizes Likelihood$[\![P]\!]$$(x,y) \text{ (i.e. maximize probability of data)}$?

A: To prevent overfitting

Recall: $P_{\text{Posterior}}(x,y) \propto P_{\text{Prior}}[P] \times \text{Likelihood}[P](x,y)$

Priors prevents us from finding too complex programs that fit data arbitrarily well without sufficient evidence (Bayes’ Occam’s Razor [MacKay03, Ch28])

Probabilistically coherent way to achieve “regularization”

Higher data likelihood
Complex program
Not useful for discovering patterns
Not useful for making predictions
Bayesian Synthesis

- space of probabilistic programs \( P \) that can generate time series data
- Prior[\( P \)]
  - prior distribution over programs \( P \)
- Likelihood[\( P \)](\( x, y \))
  - Probability that program \( P \) returns output \( y \) given input \( x \)

- Posterior[\( P \)](\( x, y \))
  - distribution over programs \( P \) conditioned on data \((x,y)\)
  - probabilistic programs that are likely to have generated the data
  - syntactic analysis
    - what patterns probably exist in the data?
  - probabilistic inference
    - what are the probable values for new time points?
space of probabilistic programs $P$ that can generate time series data

Prior[$P$] prior distribution over programs $P$

Likelihood[$P$]($x, y$) probability that program $P$ returns output $y$ given input $x$

Posterior[$P$]($x, y$) distribution over programs $P$ conditioned on data ($x,y$)

probabilistic programs that are likely to have generated the data

syntactic analysis

what patterns probably exist in the data?

probabilistic inference

what are the probable values for new time points?
Q: Why not find DSL program $P$ that maximizes Posterior$[P](x,y)$ (i.e. find the most likely program)?

DATA
Q: Why not find DSL program $P$ that maximizes Posterior$\mathbb{E}[P](x,y)$ (i.e. find the most likely program)?
Q: Why not find DSL program $P$ that maximizes Posterior$[P](x,y)$ (i.e. find the most likely program)?
Q: Why not find DSL program $P$ that maximizes Posterior $\mathbb{P}(x,y)$ (i.e. find the most likely program)?

**SCENARIO A**

- **DATA**: Observed Data, Unobserved Data
- **LINEAR + NOISE**: Observed Data
- **PERIODIC * LINEAR + NOISE**: Observed Data
Q: Why not find DSL program $P$ that maximizes Posterior$P(x,y)$ (i.e. find the most likely program)?

**SCENARIO A**

![Graphs showing data and forecasts for different scenarios with observed and unobserved data.]

- **DATA**
  - Observed Data
  - Unobserved Data

- **LINEAR + NOISE**
  - Forecasts
  - Observed Data
  - Unobserved Data
  - *good*

- **PERIODIC * LINEAR + NOISE**
  - Observed Data
Q: Why not find DSL program P that maximizes Posterior[⟨P⟩](x,y) (i.e. find the most likely program)?

**Scenario A**

**Data**
- Observed Data
- Unobserved Data

**Linear + Noise**
- Forecasts
- Observed Data
- Unobserved Data
- Good

**Periodic * Linear + Noise**
- Forecasts
- Observed Data
- Unobserved Data
- Overfit
Q: Why not find DSL program $P$ that maximizes Posterior $\mathbb{P}(x,y)$ (i.e. find the most likely program)?

**SCENARIO A**

DATA

LINEAR + NOISE

**good**

**PERIODIC * LINEAR + NOISE**

**overfit**

**SCENARIO B**

DATA

Observed Data

Unobserved Data
Q: Why not find DSL program $P$ that maximizes Posterior $\mathbb{P}(x,y)$ (i.e. find the most likely program)?

**SCENARIO A**

- **DATA**
  - Observed Data
  - Unobserved Data

- **LINEAR + NOISE**
  - Forecasts
  - Observed Data
  - Unobserved Data
  - **good**

- **PERIODIC * LINEAR + NOISE**
  - Forecasts
  - Observed Data
  - Unobserved Data
  - **overfit**

**SCENARIO B**

- **DATA**
  - Observed Data
  - Unobserved Data

- **LINEAR + NOISE**
  - Forecasts
  - Observed Data
  - Unobserved Data
  - **underfit**
Q: Why not find DSL program $P$ that maximizes Posterior $[P](x,y)$ (i.e. find the most likely program)?

**SCENARIO A**
- **DATA**
  - Observed Data
  - Unobserved Data

**LINEAR + NOISE**
- Good

**PERIODIC * LINEAR + NOISE**
- Overfit

**SCENARIO B**
- **DATA**
  - Observed Data
  - Unobserved Data

**LINEAR + NOISE**
- Underfit

**DATA**
- Good

**PERIODIC * LINEAR + NOISE**
- Good
Q: Why not find DSL program $P$ that maximizes Posterior$[P](x,y)$ (i.e. find the most likely program)?

A: To compare and combine multiple hypotheses for the data

- No single “most-likely” program is guaranteed to ideally fit the data in all scenarios (even plausible ones).

- Posteriors may be multi-modal.

- Bayesian sampling of programs allows us to obtain multiple candidate programs weighted by their probabilities.

- Bayesian sampling allows us to combine these (and many other) programs to get an idea of what the data-generating process may likely be [Robert2003, Ch7].
Probabilistic inference

Prior \[ P \]

prior distribution over programs \( P \)

Likelihood \[ P \](\( x, y \))
distribution over output \( y \) on input \( x \) for program \( P \)

Posterior \[ P \](\( x, y \))
posterior distribution over programs conditioned on data

Bayesian Synthesis

Space of probabilistic programs \( P \) that could have generated the data

Probabilistic programs that are likely to have generated the data

Syntactic analysis

What patterns probably exist in the data?

Probabilistic inference

What are the probable values for new time points?
Outline

1. Motivation

2. Synthesizing probabilistic programs for time series data in Venture

3. Probabilistic programming formulation of structure discovery
Time series structure discovery using program synthesis in Venture

[Graph showing observed data with passenger volume on the y-axis and year on the x-axis.]
Time series structure discovery using program synthesis in Venture

// ** LOAD THE TIME SERIES DATA **
define xs_obs = get_data_xs("./data-train.csv");
define ys_obs = get_data_yx("./data-train.csv");
define xs_test = get_data_xs("./data-test.csv");
define ys_test = get_data_yx("./data-test.csv");
Time series structure discovery using program synthesis in Venture

```
// ** LOAD THE TIME SERIES DATA **
define xs_obs = get_data_xs("./data-train.csv");
define ys_obs = get_data_ys("./data-train.csv");
define xs_test = get_data_xs("./data-test.csv");
define ys_test = get_data_ys("./data-test.csv");

// ** SAMPLE ENSEMBLE OF GAUSSIAN PROCESSES FROM THE PRIOR **
resample(100);
assume dsl_code ~ generate_random_program();
assume gaussian_process_model = produce_gaussian_process(dsl_code);
observe gaussian_process_model(${xs_obs}) = ys_obs;
```
Time series structure discovery using program synthesis in Venture

// ** LOAD THE TIME SERIES DATA **
define xs_obs = get_data_xs("./data-train.csv");
define ys_obs = get_data_ys("./data-train.csv");
define xs_test = get_data_xs("./data-test.csv");
define ys_test = get_data_ys("./data-test.csv");

// ** SAMPLE ENSEMBLE OF GAUSSIAN PROCESSES FROM THE PRIOR **
resample(100);
assume dsl_code ~ generate_random_program();
assume gaussian_process_model = produce_gaussian_process(dsl_code);
observe gaussian_process_model('${xs_obs}') = ys_obs;

// ** RUN BAYESIAN SYNTHESIS **
repeat(1000, {
  infer resimulation_mh('//hypers/*);
  infer resimulation_mh('//structure/*'))
}
Time series structure discovery using program synthesis in Venture

```
// ** LOAD THE TIME SERIES DATA **
define xs_obs = get_data_xsl("./data-train.csv");
define ys_obs = get_data_ys("./data-train.csv");
define xs_test = get_data_xsl("./data-test.csv");
define ys_test = get_data_ys("./data-test.csv");

// ** SAMPLE ENSEMBLE OF GAUSSIAN PROCESSES FROM THE PRIOR **
resample(100);
assume dsl_code ~ generate_random_program();
assume gaussian_process_model = produce_gaussian_process(dsl_code);
observe gaussian_process_model(${xs_obs}) = ys_obs;

// ** RUN BAYESIAN SYNTHESIS **
repeat(1000, {
    infer resimulation_mh(//hypers/*);
    infer resimulation_mh(//structure/*)}

// ** OBTAIN FORECASTS **
sample_all(gaussian_process_model(${xs_test}$))
```
Are predictions from the synthesized probabilistic programs accurate?
Are predictions from the synthesized probabilistic programs accurate?

### Standardized Root Mean Squared Forecasting Error (RMSE)

<table>
<thead>
<tr>
<th>Model</th>
<th>temperature</th>
<th>airline</th>
<th>call</th>
<th>mauna</th>
<th>radio</th>
<th>solar</th>
<th>wheat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bayesian Synthesis</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.47</td>
<td>1.50</td>
</tr>
<tr>
<td>Gaussian Process (Squared Exponential Kernel)</td>
<td>1.70</td>
<td>2.01</td>
<td>4.26</td>
<td>1.54</td>
<td>2.03</td>
<td>1.63</td>
<td>1.37</td>
</tr>
<tr>
<td>Auto-Regressive Integrated Moving Average</td>
<td>1.85</td>
<td>1.32</td>
<td>2.44</td>
<td>1.09</td>
<td>2.08</td>
<td>1.0</td>
<td>1.41</td>
</tr>
<tr>
<td>Facebook Prophet</td>
<td>2.00</td>
<td>1.83</td>
<td>5.61</td>
<td>1.23</td>
<td>3.09</td>
<td>1.73</td>
<td>1.29</td>
</tr>
<tr>
<td>Hierarchical-DP Hidden Markov Model</td>
<td>1.77</td>
<td>4.61</td>
<td>2.26</td>
<td>14.77</td>
<td>1.19</td>
<td>3.49</td>
<td>1.89</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>1.30</td>
<td>1.79</td>
<td>6.23</td>
<td>2.19</td>
<td>2.73</td>
<td>1.57</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Quantifying uncertainty about properties of synthesized time series programs

Bayesian ensemble of synthesized programs
Quantifying uncertainty about properties of synthesized time series programs

Bayesian ensemble of synthesized programs

```
define count_kernels = (dsl kernel) ->{
    if contains(["*", "+", "CP"], dsl[0]){
        count_kernels(dsl[1], kernel)
        + count_kernels(dsl[2], kernel)
    } else {
        if (kernel == dsl[0]) {1} else {0}
    }
}
define count_operators = (dsl, oper) ->{
    if contains(["*", "+", "CP"], dsl[0]){
        (if (oper == dsl[0]) {1} else {0})
        + count_operators(dsl[1], oper)
        + count_operators(dsl[2], oper)
    } else {0}
};
```
Bayesian ensemble of synthesized programs

```
define count_kernels = (dsl kernel) -> {
  if contains(["*", "+", "CP"], dsl[0]) {
    count_kernels(dsl[1], kernel)
    + count_kernels(dsl[2], kernel)
  } else {
    if (kernel == dsl[0]) {1} else {0}
  }
};;

define count_operators = (dsl, oper) -> {
  if contains(["*", "+", "CP"], dsl[0]) {
    (if (oper == dsl[0]) {1} else {0})
    + count_operators(dsl[1], oper)
    + count_operators(dsl[2], oper)
  } else {0}
};;
```

Temporal Structure $p^{synth}$

- White Noise 100%
- Linear Trend 16%
- Periodicity 92%
- Change Point 4%

(a) Temporal Structure $p^{synth}$

- White Noise 100%
- Linear Trend 85%
- Periodicity 76%
- Change Point 76%

(b) Temporal Structure $p^{synth}$

- White Noise 100%
- Linear Trend 6%
- Periodicity 93%
- Change Point 23%

(d) Temporal Structure $p^{synth}$

- White Noise 3%
- Linear Trend 8%
- Periodicity 1%
- Change Point 2%
A closer look at the synthesis system

```plaintext
// ** LOAD THE TIME SERIES DATA **
define xs_obs = get_data_xs("./data-train.csv");
define ys_obs = get_data_ys("./data-train.csv");
define xs_test = get_data_xs("./data-test.csv");
define ys_test = get_data_ys("./data-test.csv");

// ** SAMPLE ENSEMBLE OF GAUSSIAN PROCESSES FROM THE PRIOR **
resample(100);
assume dsl_code ~ generate_random_program();
assume gaussian_process_model = produce_gaussian_process(dsl_code);
observe gaussian_process_model(${xs_obs}) = ys_obs;

// ** RUN BAYESIAN SYNTHESIS **
repeat(1000, {
infer resimulation_mh(/?hypers/*);
infer resimulation_mh(/?structure/*)}

// ** OBTAIN FORECASTS **
sample_all(gaussian_process_model(${xs_test}$))
```
A closer look at the synthesis system

```csharp
// ** LOAD THE TIME SERIES DATA **
define xs.obs = get_data_xs("./data-train.csv");
define ys.obs = get_data ys("./data-train.csv");
define xs.test = get_data_xs("./data-test.csv");
define ys.test = get_data ys("./data-test.csv");

// ** SAMPLE ENSEMBLE OF GAUSSIAN PROCESSES FROM THE PRIOR **
resample(100);
assume dsl_code ~ generate_random_program();
assume gaussian_process_model = produce_gaussian_process(dsl_code);
observe gaussian_process_model({xs.obs}) = ys.obs;

// ** RUN BAYESIAN SYNTHESIS **
repeat(1000, {
    infer resimulation_mh(~/hyps/*);
    infer resimulation_mh(~/structure/*))

// ** OBTAIN FORECASTS **
sample_all(gaussian_process_model({xs.test}))
```
**Probabilistic context-free grammar for sampling DSL code**

**Syntax of DSL**

$v \in \text{Numeric}$

$H \in \text{Params ::= (gamma } v \text{)}$ [Parameter]

$K \in \text{Covariance ::=}$

$(\text{constant } H)$ [Constant]

$(\text{white-noise } H)$ [WhiteNoise]

$(\text{linear } H)$ [Linear]

$(\text{smooth } H)$ [Smooth]

$(+ K_1 K_2)$ [Sum]

$(\ast K_1 K_2)$ [Product]

$(\text{change-point } v K_1 K_2)$ [ChangePoint]

**Production Rule Probabilities**

$P(\text{constant}) = P(\text{white-noise})$

$= P(\text{linear}) = P(\text{smooth}) := 0.14$

$P(+) = P(\ast) = 0.135$

$P(\text{change-point}) = 0.03$

**Terminal Symbol Probabilities**

$P(\text{gamma, } v) = \text{Gamma}(v; 1,1)$
Probabilistic context-free grammar for sampling DSL code

**Syntax of DSL**

$v \in \text{Numeric}$

$H \in \text{Params} ::= (\text{gamma } v)$ \hspace{1cm} [Parameter]

$K \in \text{Covariance} ::=$

$(\text{constant } H)$ \hspace{1cm} [Constant]

$(\text{white-noise } H)$ \hspace{1cm} [WhiteNoise]

$(\text{linear } H)$ \hspace{1cm} [Linear]

$(\text{smooth } H)$ \hspace{1cm} [Smooth]

$(+ K_1 K_2)$ \hspace{1cm} [Sum]

$(* K_1 K_2)$ \hspace{1cm} [Product]

$(\text{changepoint } v K_1 K_2)$ \hspace{1cm} [ChangePoint]

**Production Rule Probabilities**

$P(\text{constant}) = P(\text{white-noise})$

$= P(\text{linear}) = P(\text{smooth}) := 0.14$

$P(+) = P(*) = 0.135$

$P(\text{change-point}) = 0.03$

**Terminal Symbol Probabilities**

$P(\text{gamma}, v) = \text{Gamma}(v; 1,1)$

**Venture program implementing the PCFG**

```require
assume get_hyper ~ mem(node_index) -> {
    -log.logistic(log_odds.uniform() #hypers:node_index)
};

assume choose_primitive = (node) -> {
    kernel ~ categorical(simplex(.2, .2, .2, .2, .2),
        ["WN", "C", "LIN", "SE", "PER"])
    #structure:pair("kernel", node); 
    cond( 
        kernel == "WN".isTrue \{ ["WN", get_hyper(pair("WN", node)),
            kernel == "C".isTrue \{ ["C", get_hyper(pair("C", node)),
                kernel == "LIN".isTrue \{ ["LIN", get_hyper(pair("LIN", node)),
                    kernel == "SE".isTrue .1 + get_hyper(pair("SE", node)),
                    kernel == "PER".isTrue .01 + get_hyper(pair("PER.l", node)),
                    .01 + get_hyper(pair("PER.t", node))
            cond( 
                operator_symbol ~ categorical(simplex(0.45, 0.45, 0.1),
                    ["+", "+", "+", "+", "+"])
                #structure:pair("operator", node); 
            if (operator_symbol == "CP".isTrue)
                [operator_symbol, get_hyper.pair("CP", node))
            )
                operator_symbol
        }
    }
};

assume generate_random_program = mem((node) -> {
    if (flip(.3)) #structure:pair("branch", node) {
        operator ~ choose_operator(node);
        [operator, 
            generate_random_program(2 + node), 
            generate_random_program(2 + node + 1)]
    } else {
        choose_primitive(node)
    })
});
```
Probabilistic context-free grammar for sampling DSL code

Syntax of DSL

\[ v \in \text{Numeric} \]
\[ H \in \text{Params} ::= (\text{gamma } v) \quad \text{[Parameter]} \]
\[ K \in \text{Covariance} ::= \]

- \( (\text{constant } H) \) [Constant]
- \( (\text{white-noise } H) \) [WhiteNoise]
- \( (\text{linear } H) \) [Linear]
- \( (\text{smooth } H) \) [Smooth]

- \( (+ K_1 K_2) \) [Sum]
- \( (* K_1 K_2) \) [Product]
- \( (\text{change-point } v K_1 K_2) \) [ChangePoint]

Production Rule Probabilities

- \( P(\text{constant}) = P(\text{white-noise}) = P(\text{linear}) = P(\text{smooth}) := 0.14 \)
- \( P(+) = P(*) = 0.135 \)
- \( P(\text{change-point}) = 0.03 \)

Terminal Symbol Probabilities

- \( P(\text{gamma}, v) = \text{Gamma}(v; 1,1) \)

Venture program implementing the PCFG

```java
assume get_hyper ~ mem((node_index) -> {
    -log_logistic(log_odds_uniform() #hypers:node_index)
});

assume choose_primitive = (node) -> {
    kernel ~ categorical/simplex(2, 2, 2, 2, 2),
    ["WN", "C", "LIN", "SE", "PER"]
    @structure:pair("kernel", node);
    cond((
        kernel == "WN")([["WN", get_hyper(pair("WN", node))]],
        kernel == "C")([["C", get_hyper(pair("C", node))]],
        kernel == "LIN")((("LIN", get_hyper(pair("LIN", node)))),
        kernel == "SE")((("SE", 1 + get_hyper(pair("SE", node)))),
        kernel == "PER")((("PER", .01 + get_hyper(pair("PER", node)),
            .01 + get_hyper(pair("PER", node))
        )))
    ));

assume choose_operator = mem((node) -> {
    operator_symbol ~ categorical/simplex(0.45, 0.45, 0.1),
    ["+", "+", "CP"] @structure:pair("operator", node);
    if (operator_symbol == "CP") {
        [operator_symbol, get_hyper.prior(pair("CP", node))]
    } else {
        operator_symbol
    }
});

assume generate_random_program = mem((node) -> {
    if (flip(0.5) @structure:pair("branch", node)) {
        operator = choose_operator(node);
        [operator, generate_random_program(2 + node),
        generate_random_program(2 + node + 1)]
    } else {
        choose_primitive(node)
    }
});
```
Probabilistic context-free grammar for sampling DSL code

**Syntax of DSL**
\[ v \in \text{Numeric} \]
\[ H \in \text{Params} ::= (\text{gamma } v) \text{ [Parameter]} \]
\[ K \in \text{Covariance} ::= \]
\[ (\text{constant } H) \text{ [Constant]} \]
\[ (\text{white-noise } H) \text{ [WhiteNoise]} \]
\[ (\text{linear } H) \text{ [Linear]} \]
\[ (\text{smooth } H) \text{ [Smooth]} \]
\[ (+ K_1 K_2) \text{ [Sum]} \]
\[ (* K_1 K_2) \text{ [Product]} \]
\[ (\text{changepoint } v K_1 K_2) \text{ [ChangePoint]} \]

**Production Rule Probabilities**
P(constant) = P(white-noise) = P(linear) = P(smooth) := 0.14
P(+) = P(*) = 0.135
P(change-point) = 0.03

**Terminal Symbol Probabilities**
P(gamma, v) = Gamma(v; 1,1)

**Venture program implementing the PCFG**

```venture
assume get_hyper - mem(node_index) =>$\{ 
-\log\-logistic\(\log\-odds\-uniform()\#hyps\)\); 
\}

assume choose\-primitive = \(\text{node}\) =>$\{ 
kernel\-categorical\(\text{simplex}(1.2,1.2,1.2,1.2,1.2)\)\[\text{["WN", "C", "LIN", "SE", "PER"]}\); 
structure\-pair\(\text{kernel}, \text{node}\); 
\cond\(\text{(kernel} = \text{"WN")} \{\text{"WN", get\-hyper\(\text{pair}("WN", \text{node})\)}\}, 
\text{(kernel} = \text{"C")} \{\text{"C", get\-hyper\(\text{pair}("C", \text{node})\)}\}, 
\text{(kernel} = \text{"LIN")} \{\text{"LIN", get\-hyper\(\text{pair}("LIN", \text{node})\)}\}, 
\text{(kernel} = \text{"SE")} \{\text{"SE", 1.1} + \text{get\-hyper\(\text{pair}("SE", \text{node})\)}\}, 
\text{(kernel} = \text{"PER")} \{\text{"PER", 1.01} + \text{get\-hyper\(\text{pair}("PER_{l}", \text{node})\)}, 
\text{0.01} + \text{get\-hyper\(\text{pair}("PER_{t}", \text{node})\)}\}\); 
\}\); 
assume choose\-operator = \(\text{op}(node)\) =>$\{ 
operator\-symbol\-categorical\(\text{simplex}(0.45,0.45,0.1)\)\[\text{["+", "+", "+"]}\] #structure\-pair\(\text{operator}, \text{node}\); 
\text{if operator\-symbol} \text{=} \text{"C")} \{ 
operator\-symbol, get\-hyper\(\text{prior}\(\text{pair}("C", \text{node})\)) \} 
\text{else} \{ 
operator\-symbol 
\}\); 

assume generate\-random\-program = mem(node_index) =>$\{ 
if flip\(0.3) \#structure\-pair("branch", node)\) \{ 
operator, choose\-operator\(\text{node}\); 
[op, generate\-random\-program\(2 + \text{node}\), generate\-random\-program\(2 + \text{node} + 1\)] 
\} 
\text{else} \{ 
choose\-primitive\(\text{node}\) 
\}\};
```
Probabilistic context-free grammar for sampling DSL code

**Syntax of DSL**

- $v \in \text{Numeric}$
- $H \in \text{Params} ::= (\text{gamma } v)$ [Parameter]
- $K \in \text{Covariance} ::= $
  - $\text{constant } H$
  - $\text{white-noise } H$
  - $\text{linear } H$
  - $\text{smooth } H$

- $K_1 K_2$ [Sum]
- $(+ K_1 K_2)$ [Product]
- $(\text{changepoint } v K_1 K_2)$ [ChangePoint]

**Production Rule Probabilities**

- $P(\text{constant}) = P(\text{white-noise}) = P(\text{linear}) = P(\text{smooth}) := 0.14$
- $P(+) = P(*) = 0.135$
- $P(\text{change-point}) = 0.03$

**Terminal Symbol Probabilities**

- $P(\text{gamma}, v) = \text{Gamma}(v; 1,1)$

**Venture program implementing the PCFG**

```c
assume get_hyper - mem((node_index) -> {
  -log-logistic(log_odds_uniform() #hyps:node_index)
});

assume choose_primitive = (node) -> {
  kernel - categorical(simplex(.2, .2, .2, .2, .2),
  "WN", "C", "LIN", "SE", "PER")
  #structure:pair("kernel", node);
  cond(
    (kernel == "WN") ("WN", get_hyper(pair("WN", node))),
    (kernel == "C") ("C", get_hyper(pair("C", node))),
    (kernel == "LIN") ("LIN", get_hyper(pair("LIN", node))),
    (kernel == "SE") ("SE", .1 + get_hyper(pair("SE", node))),
    (kernel == "PER") ("PER", .01 + get_hyper(pair("PER", node)),
      .01 + get_hyper(pair("PER", node))
  )
});

assume choose_operator = mem((node) -> {
  operator_symbol - categorical(simplex(0.45, 0.45, 0.1),
  "+", ".", "CP") #structure:pair("operator", node);
  if (operator_symbol == "CP") {
    [operator_symbol, get_hyper_prior(pair("CP", node))]
  } else {
    operator_symbol
  }
});

assume generate_random_program = mem((node) -> {
  if (flip(.3) #structure:pair("branch", node)) {
    operator = choose_operator(node);
    [operator, generate_random_program(2 + node),
      generate_random_program(2 + node + 1)]
  } else {
    choose_primitive(node)
  }
});
```
interpreting the DSL program

```plaintext
// ** LOAD THE TIME SERIES DATA **
declare xs_obs = get_data_xs("./data-train.csv");
declare ys_obs = get_data_ys("./data-train.csv");
declare xs_test = get_data_xs("./data-test.csv");
declare ys_test = get_data_ys("./data-test.csv");

// ** SAMPLE ENSEMBLE OF GAUSSIAN PROCESSES FROM THE PRIOR **
resample(100);
assume dsl_code = generate_random_program();
assume gaussian_process_model = produce_gaussian_process(dsl_code);
observe gaussian_process_model(xs_obs) = ys_obs;

// ** RUN BAYESIAN SYNTHESIS **
repeat(1000), {
  infer resimulation_mh(??hyps/??);
  infer resimulation_mh(??structure/??)}

// ** OBTAIN FORECASTS **
sample_all(gaussian_process_model(xs_test))
```
Interpreting DSL code to produce a Venture object that inference can target

Venture program implementing the interpreter

```venture
assume produce_covariance = (source) -> {
    cond(
        (source[0] = "WN") (gp_cov_scale(source[1], gp_cov_bump)),
        (source[0] = "C") (gp_cov_const(source[1])),
        (source[0] = "LIN") (gp_cov_linear(source[1])),
        (source[0] = "SE") (gp_cov_se(source[1]**2)),
        (source[0] = "PER") (gp_cov_periodic(source[1]**2, source[2])),
        (source[0] = "+") (gp_cov_sum(
            produce_covariance(source[1]),
            produce_covariance(source[2]))),
        (source[0] = "*")) (gp_cov_product(
            produce_covariance(source[1]),
            produce_covariance(source[2]))),
        (source[0][0] = "CP") (gp_cov_cp(source[0][1], .1,
            produce_covariance(source[1]),
            produce_covariance(source[2]))))
    )
}

assume produce_gaussian_process = (source) -> {
    baseline_noise = gp_cov_scale(.1, gp_cov_bump);
    covariance_kernel = gp_cov_sum(
        produce_covariance(source), baseline_noise);
    make_gp(gp_mean_const(0.), covariance_kernel)
};
```
A closer look at the domain-specific-language for time series structures

```python
// ** LOAD THE TIME SERIES DATA **
define xs_obs = get_data_xs("./data-train.csv");
define ys_obs = get_data_yx("./data-train.csv");
define xs_test = get_data_xs("./data-test.csv");
define ys_test = get_data_yx("./data-test.csv");

// ** SAMPLE ENSEMBLE OF GAUSSIAN PROCESSES FROM THE PRIOR **
resample(100);
assume dsl_code ~ generate_random_program();
assume gaussian_process_model = produce_gaussian_process(dsl_code);
observe gaussian_process_model(x{xs_obs}) = ys_obs;

// ** RUN BAYESIAN SYNTHESIS **
repeat(1000, {
    infer resimulation_mh(/?hypers/);
    infer resimulation_mh(/?structure/)}

// ** OBTAIN FORECASTS **
sample_all(gaussian_process_model(x{xs_test}))
```

see main paper for general MCMC synthesis algorithm (sound for any PCFG, not just this DSL)
Outline

1. Motivation

2. Synthesizing probabilistic programs for time series data in Venture

3. Probabilistic programming formulation of structure discovery
Reducing implementation complexity

<table>
<thead>
<tr>
<th></th>
<th>Bayesian Synthesis(^1) (Venture)</th>
<th>ABCD System(^2) (MATLAB + Python)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Main System</strong></td>
<td>70</td>
<td>4,166</td>
</tr>
<tr>
<td><strong>Gaussian process libraries</strong></td>
<td>2,164 (Venture GP(^3))</td>
<td>13,945 (GPML Toolbox(^4))</td>
</tr>
<tr>
<td><strong>Generic inference implementation</strong></td>
<td>1,877 (Venture(^5))</td>
<td>—</td>
</tr>
</tbody>
</table>

\(^1\) Proposed method  
\(^2\) Lloyd et. al (AAAI 2013)  
\(^3\) Schaechtle et. al. (arXiv 2015)  
\(^4\) Rasumessen & Nickish (JMLR 2015)  
\(^5\) Mansinghka et. al (PLDI 2018)
What does it mean to be Bayesian for program synthesis?

**Probabilistic Model Family**

- $p(\text{structure})$
- $p(\text{params} \mid \text{structure})$
- $p(\text{data} \mid \text{params}, \text{structure})$

**Probabilistic Domain-Specific Language**

- $p(\text{code})$
- $p(\text{params} \mid \text{code})$
- $p(\text{data} \mid \text{params}, \text{code})$

**Statistics/Informal Description**

**Programming Languages Description**
Is it enough to write a probabilistic program over DSL expressions?

Reasoning about (probabilistic) programs can be subtle

<table>
<thead>
<tr>
<th>Probabilistic Domain-Specific Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>p(code)</td>
</tr>
<tr>
<td>DSL Program Structure</td>
</tr>
<tr>
<td>p(params</td>
</tr>
<tr>
<td>DSL Program</td>
</tr>
<tr>
<td>Numerical Constants</td>
</tr>
<tr>
<td>p(data</td>
</tr>
<tr>
<td>Data</td>
</tr>
</tbody>
</table>

Programming Languages Description
Is it enough to write a probabilistic program over DSL expressions?

Reasoning about (probabilistic) programs can be subtle

Posterior may not be computable (or may not exist at all)

Noncomputable conditional distributions

2011 26th Annual IEEE Symposium on Logic in Computer Science

Nathanael L. Ackerman
Department of Mathematics
Harvard University
nate@math.harvard.edu

Cameron E. Freer
Department of Mathematics
University of Hawai‘i at Mānoa
freer@math.hawaii.edu

Daniel M. Roy
Computer Science and AI Lab
Massachusetts Institute of Technology
droy@csail.mit.edu

Probabilistic Domain-Specific Language

DSL Program Structure

p(code)

DSL Program

Numerical Constants

p(params | code)

Data

p(data | params, code)

Programming Languages Description
Is it enough to write a probabilistic program over DSL expressions?

Reasoning about (probabilistic) programs can be subtle

Posterior may not be computable (or may not exist at all)

Noncomputable conditional distributions

- Is the prior well-defined?
- Is the likelihood of data given program finite?
- Does the posterior distribution exist?
- Does the inference algorithm converge to the posterior?
Towards a PL semantics for Bayesian inference over program spaces

Probabilistic Context-Free Grammar \( G = (\Sigma, N, R, Q, S) \)

Domain-Specific Language \( L = \text{Language}(G) \)

Denotational semantics for a program \( P \in L \)

Semantic Function Prior : \( L \rightarrow [0,1] \)
Semantic Function Likelihood : \( L \rightarrow (X \rightarrow Y \rightarrow \mathbb{R}_+) \)

\( \text{Prior}[[P]] = \) “probability of probabilistic program \( P \)”
\( \text{Likelihood}[[P]](x, y) = \) “probability that \( P \) returns \( y \) given input \( x \)”
**Sufficient conditions for Bayesian synthesis to be well-defined**

1. The prior is normalized \( \sum_{P \in L} \text{Prior}[P] = 1 \)

2. The likelihood is normalized \( \forall P \in L. \forall x \in X. \sum_{y \in Y} \text{Likelihood}[P](x, y) = 1 \)

3. The likelihood is bounded \( \forall x \in X. \forall y \in Y. \sup_{P \in L} \{ \text{Likelihood}[P](x, y) \} < \infty \)

These conditions are sufficient to yield the posterior distribution over programs \( P \):

\[
\text{Posterior}[P](x, y) \triangleq (\text{Prior}[P] \times \text{Likelihood}[P](x, y)) / Z_{xy}
\]
Sufficient conditions for Bayesian synthesis to be well-defined

1. The prior is normalized \[ \sum_{P \in L} \text{Prior}(P) = 1 \]
Sufficient conditions for Bayesian synthesis to be well-defined

1. The prior is normalized

\[ \sum_{P \in L} \text{Prior}[P] = 1 \]

sum over countable set of strings
Sufficient conditions for Bayesian synthesis to be well-defined

1. The prior is normalized \( \sum_{P \in L} \text{Prior}[P] = 1 \)

   sum over countable set of strings

   Equivalent to checking that DSL prior sampler halts with probability 1.
Sufficient conditions for Bayesian synthesis to be well-defined

1. The prior is normalized

\[ \sum_{P \in L} \text{Prior}[[P]] = 1 \]

sum over countable set of strings

Equivalent to checking that DSL prior sampler \textit{halts} with probability 1.

Normalized Prior

```plaintext
assume generate_random_program = (i) -> {
    if (flip(.49)) {
        operator ~ choose_operator(i);
        [operator,
            generate_random_program(2*i),
            generate_random_program(2*i + 1)]
    } else {
        choose_primitive(i)
    }
};
```

Unnormalized Prior

```plaintext
assume generate_random_program = (i) -> {
    if (flip(.51)) {
        operator ~ choose_operator(i);
        [operator,
            generate_random_program(2*i),
            generate_random_program(2*i + 1)]
    } else {
        choose_primitive(i)
    }
};
```
Sufficient conditions for Bayesian synthesis to be well-defined

1. The prior is normalized

\[ \sum_{P \in L} \text{Prior}[[P]] = 1 \]

Equivalent to checking that DSL prior sampler halts with probability 1.

---

**Normalized Prior**

```java
assume generate_random_program = (i) -> {
    if (flip(.49)) {
        operator ~ choose_operator(i);
        [operator,
            generate_random_program(2*i),
            generate_random_program(2*i + 1)]
    } else {
        choose_primitive(i)
    }
}
```

**Unnormalized Prior**

```java
assume generate_random_program = (i) -> {
    if (flip(.51)) {
        operator ~ choose_operator(i);
        [operator,
            generate_random_program(2*i),
            generate_random_program(2*i + 1)]
    } else {
        choose_primitive(i)
    }
}
```
1. The prior is normalized \[ \sum_{P \in L} \text{Prior}[[P]] = 1 \]

Equivalent to checking that DSL prior sampler *halts* with probability 1.

Do I need to solve the halting problem??

For general DSL priors with arbitrary recursion, yes.

Is hope lost for writing priors over expressions? No.

- use a finite language
- use carefully constructed PCFG
- or other things…

How to solve halting problem for PCFG (Booth & Thomson 1973)

Applying Probability Measures to Abstract Languages

TAYLOR L. BOOTH AND RICHARD A. THOMPSON
Sufficient conditions for Bayesian synthesis to be well-defined

1. The prior is normalized $\sum_{P \in L} \text{Prior}([P]) = 1$

   sum over countable set of strings

   Equivalent to checking that DSL prior sampler halts with probability 1.

Do I need to solve the halting problem??

For general DSL priors with arbitrary recursion, yes.

Is hope lost for writing priors over expressions? No.

- use a finite language
- use carefully constructed PCFG
- or other things…

How to solve halting problem for PCFG (Booth & Thomson 1973)
A static analysis for halting: Take the PCFG

**Syntax of DSL**

- \( v \in \text{Numeric} \)
- \( H \in \text{Params} ::= (\text{gamma } v) \) [Parameter]
- \( K \in \text{Covariance} ::= \)
  - \((\text{constant } H)\) [Constant]
  - \((\text{white-noise } H)\) [WhiteNoise]
  - \((\text{linear } H)\) [Linear]
  - \((\text{smooth } H)\) [Smooth]

- \((+ K_1 K_2)\) [Sum]
- \((* K_1 K_2)\) [Product]
- \((\text{change-point } v K_1 K_2)\) [ChangePoint]

**Production Rule Probabilities**

- \( P(\text{constant}) = P(\text{white-noise}) = P(\text{linear}) = P(\text{smooth}) := 0.14 \)
- \( P(+) = P(*) = 0.135 \)
- \( P(\text{change-point}) = 0.03 \)

**Terminal Symbol Probabilities**

- \( P(\text{gamma}, v) = \Gamma(v; 1,1) \)
A static analysis for halting: Translate PCFG to Chomsky-Normal Form

\[
K \rightarrow CB_{0.14} | EB_{0.14} | GB_{0.14} | IB_{0.14} | PB_{0.14} | TM_{0.135} | VM_{0.135} | WX_{0.03}
\]

\[
C \rightarrow \text{constant}
\]

\[
E \rightarrow \text{white-noise}
\]

\[
G \rightarrow \text{linear}
\]

\[
I \rightarrow \text{smooth}
\]

\[
P \rightarrow \text{periodic}
\]

\[
B \rightarrow \text{gamma}
\]

\[
T \rightarrow +
\]

\[
V \rightarrow *
\]

\[
W \rightarrow \text{change-point}
\]

\[
M \rightarrow KK
\]

\[
M \rightarrow BM
\]
A static analysis for halting: Check spectral radius of expectation matrix

\[
\begin{bmatrix}
K & C & E & G & I & P & B & T & V & W & M & X \\
K & .14 & .14 & .14 & .14 & .14 & .7 & .135 & .135 & .03 & .27 & .03 \\
C & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
E & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
G & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
P & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
B & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
T & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
V & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
W & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
M & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
X & 2 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
\end{bmatrix}
\]

eigenvalues = [0.785, -0.67, -0.11] \Rightarrow \textbf{halts} (all between -1, 1)
Prior semantics for domain-specific languages described by a PCFG

[Evaluation: Non-Recursive Production Rule]

\[
N_i \rightarrow_G (t \ s) \quad \frac{N_i \downarrow_G^{P(t)Q(t,s)} (t \ s)}{N_i \downarrow_G (t \ s)}
\]

[Evaluation: Recursive Production Rule]

\[
N_i \rightarrow_G (t \ M_1 \ldots \ M_j), \ M_1 \downarrow_G^{p_1} E_1, \ldots, \ M_j \downarrow_G^{p_j} E_j
\]

\[
N_i \downarrow_G^{P(t) \prod_{i=1}^{j} p_i} (t \ E_1 \ldots \ E_j)
\]

Expand \([t \ s] (N_i) \) := \(P(t) \cdot Q(t, s)\) if \(t \in T_i\), else 0 \(\quad (N_i \in N, s \in \Sigma)\)

Expand \([t \ E_1 \ldots \ E_m] (N_i) \) := \(P(t) \cdot \prod_{j=1}^{m} \text{Expand} \left[ E_j \right] (U_{tj})\) if \(t \in T_i\), else 0 \(\quad (N_i \in N)\)

Prior \([E] \) := Expand \([E] (S)\)
Likelihood semantics for Gaussian process domain-specific language

\[
\text{Cov} \left[ (\text{const} (\gamma v)) \right] (x)(x') := v
\]

\[
\text{Cov} \left[ (\text{wn} (\gamma v)) \right] (x)(x') := v \cdot 1[ x = x']
\]

\[
\text{Cov} \left[ (\text{lin} (\gamma v)) \right] (x)(x') := (x - v)(x' - v)
\]

\[
\text{Cov} \left[ (\text{se} (\gamma v)) \right] (x)(x') := \exp((x - x')^2 / v)
\]

\[
\text{Cov} \left[ (\text{per} (\gamma v_1) (\gamma v_2)) \right] (x)(x') := \exp(-2/v_1 \sin((2\pi/v_2)|x - x'|)^2)
\]

\[
\text{Cov} \left[ (\text{K} K_1 K_2) \right] (x, x') := \text{Cov} \left[ K_1 \right] (x, x')
\]

\[
\text{Cov} \left[ (\ast K_1 K_2) \right] := \text{Cov} \left[ K_1 \right] (x)(x') + \text{Cov} \left[ K_2 \right] (x)(x')
\]

\[
\text{Cov} \left[ (\text{cp} (\gamma v) K_1 K_2) \right] := \Delta(x, v)\Delta(x', v)(\text{Cov} \left[ K_1 \right] (x)(x')) + (1 - \Delta(x, v))(1 - \Delta(x', v))(\text{Cov} \left[ K_1 \right] (x)(x'))
\]

where \( \Delta(x, v) := 0.5 \times (1 + \tanh(10(x - v))) \)

\[
\text{Lik} \left[ K \right] ((x, y)) := \exp \left( -1/2 \sum_{i=1}^{n} y_i \left[ \sum_{j=1}^{n} ([\text{Cov} \left[ K \right] (x_i)(x_j) + 0.01\delta(x_i, x_j)]_{i,j=1}^{n})^{-1} y_j \right] \right)
\]

\[
-1/2 \log \left| \text{Cov} \left[ K \right] (x_i)(x_j) + 0.01\delta(x_i, x_j) \right|_{i,j=1}^{n} - (n/2) \log 2\pi
\]
Inference semantics for Bayesian synthesis via Metropolis-Hastings

\[ a = (\) \quad \rightarrow \quad (N_i, \square) \]
\[ (a, (t_{ik} E_1 \cdots E_m)) \rightarrow_{\text{sever}} (N_i, E_{\text{hole}}) \]
\[ ((a_2, a_3, \ldots), E_j) \rightarrow_{\text{sever}} (N_i, E_{\text{sev}}) \quad \text{and} \quad a_1 = j \]
\[ (a, (t E_1 \cdots E_m)) \rightarrow_{\text{sever}} (N_i, (t E_1 \cdots E_{j-1} E_{\text{sev}} E_{j+1} \cdots E_m)) \]

\[ \text{Sever}_a \llbracket (t E_1 E_2 \cdots E_k) \rrbracket := \begin{cases} (N_i, E_{\text{hole}}) & \text{if } (a, (t E_1 E_2 \cdots E_m)) \rightarrow_{\text{sever}} (N_i, E_{\text{hole}}) \\ \emptyset & \text{otherwise} \end{cases} \]

\[ \text{Subexpr}_a \llbracket (t E_1 E_2 \cdots E_k) \rrbracket := \begin{cases} \emptyset & \text{if } a = (\) \text{ or } a_1 > k \\ E_j & \text{if } a = (j) \text{ for some } 1 \leq j \leq k \\ \text{Subexpr}_{(a_2, a_3, \ldots)} \llbracket E_j \rrbracket & \text{if } i \neq (j) \text{ and } a_1 = j \text{ for some } 1 \leq j \leq k \end{cases} \]

\[ \mathcal{T}(X, E \rightarrow E') = \frac{1}{|A_E|} \sum_{a \in A_E} \mathcal{T}(X, E \rightarrow E'; a) = \frac{1}{|A_E|} \sum_{a \in A_E \cap A_{E'}} \mathcal{T}(X, E \rightarrow E'; a), \]

where

\[ \mathcal{T}(X, E \rightarrow E'; a) := \begin{cases} \text{Expand } \llbracket \text{Subexpr}_a \llbracket E' \rrbracket \rrbracket (N_i) \cdot \alpha(E, E') & \text{(if } \text{Sever}_a \llbracket E \rrbracket = \text{Sever}_a \llbracket E' \rrbracket) \\ +[E = E'](1 - \alpha(E, E')) & \text{(otherwise)} \end{cases} \]

\[ = (N_i, E_{\text{hole}}) \text{ for some } E_{\text{hole}} \text{ and } N_i \]
Overview of related synthesis and model discovery literature

- **Fully Bayesian synthesis of probabilistic programs**
  This paper

- **Non-Bayesian synthesis of probabilistic programs**
  Beam Search [Hwang11], Approx. Bayesian Computation [Perov14], Greedy Search [Nori15],
  SMT solvers [Elis15], ABCD [Tong16], Simulated Annealing [Chasins17]

- **Probabilistic synthesis of non-probabilistic programs**
  MCMC Optimization [Schkufza13], MCMC Sampling [Liang2010]
  Approximate Rejection Sampling [Ellis2015], A* Search [Lee2018]

- **Non-probabilistic synthesis of non-probabilistic programs**
  deductive logic with program transformations [Burstall77], [Manna79], [Manna80]
  genetic programming [Koza92], [Koza97]
  constrained-based solvers [Solar-Lezama06], [Jha10], [Gulwa11], [Gulwani11], [Feser15]
  syntax-guided synthesis [Alur13]
  neural networks [Graves14], [Reed16], [Balog17], [Bosnak18]

- **Model discovery in probabilistic machine learning**
  Bayesian network structures [Mansinghka06], matrix-decomposition models [Grosse12],
  univariate time series [Duvenaud13], multivariate data tables [Mansinghka18], multivariate time series [Saad18]

* see paper for full bibliography
Key contributions

- First research to introduce fully-Bayesian synthesis of probabilistic programs.

- Rigorously formalized approach using denotational semantics.

- Proved sufficient conditions for synthesis problem to be probabilistically well-formed.

- Defined a Bayesian synthesis algorithm applicable to any DSL generated by a PCFG.

- Proved that synthesis algorithm is sound:
  - sampler for programs converges a.s. in total variation to the analytic posterior.

- Implemented two Bayesian synthesis systems:
  - time series data (this presentation)
  - multivariate data tables (see paper)

- Experimental results and comparisons on multiple real-world problems.