TU Delft

Gen: A General-Purpose Probabilistic Programming System with Programmable Inference

By: Chrysanthos Kindynis & Kylian Kropf

Cusumano-Towner, Marco F., et al. "Gen: a general-purpose probabilistic programming system with programmable inference." *Proceedings of the 40th acm sigplan conference on programming language design and implementation*. 2019.

Presentation timeline

1. Historically speaking: why was Gen needed
2. Technical flow
3. Practical code example
4. Quiz and conclusions

TU Delft

Background

- MIT authors (2019):
  - Marco Cusmano-Towner
  - Feras Saad
  - Alexander Lew
  - Vikash Mansinghka

- Impact: 176 citations

Massachusetts Institute of Technology

TU Delft

High level concept

- Modelling flexibility
  - Trade offs

- Inference flexibility
  - Inference library
  - Domain specific knowledge
  - Custom methods

=> improved performance

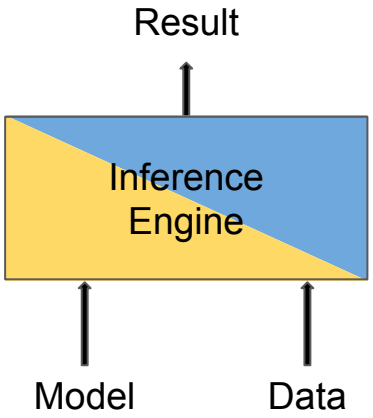
- Database view
  - Allows for taking care of technical steps automatically

TU Delft

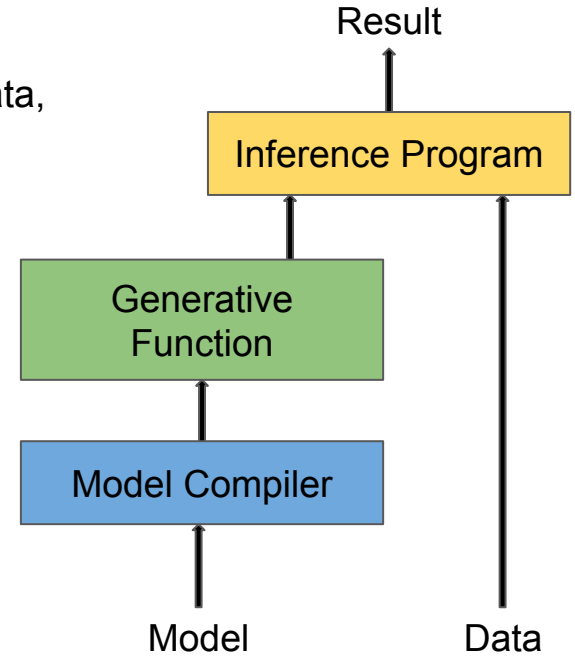
Theoretical program flow

In Gen:

1. First, we define a *generative model*
2. Second, we write an *inference program*
3. Finally, we run the inference program together with data, and return the results



Traditional



Gen

Trace

- Traces are no longer just a passive historical record
- It has an **active role** in our Probabilistic program, present in:
  - Arguments
  - Return statements
- This supports the generalizability and flexibility of the language

- @ChoiceMap: mapping from a set of addresses (A) to a set of values (V)

Illustration of code example we are trying to present

- **Goal**: Infer the orientation of the depth camera relative to the floor and ceiling

src: https://youtu.be/DIml6l_0yiM?t=19

TU Delft

1: Generative Model

- Let’s have a closer look

```
@gen function generative_model()

  floor = Plane([0.,0.,0.],[0.,0.,1.])
  room_height = @trace(uniform(2.5, 3.0), :room_height)
  ceiling = Plane([0., 0., room_height][0., 0., -1.])
  objects = [floor, ceiling]

  camera_z = @trace(uniform(0.2, 2.0), :z)
  camera_location = {0., 0., camera_z}

  camera_pitch = @trace(uniform(..., ...), :pitch)
  camera_roll= @trace(uniform(..., ...), :roll)
  camera_rotation =
    make_rotation_matrix(camera_pitch, 0., camera_roll)

  depths = render(objects, camera_location, camera_rotation)

  noise = 0.1
  @trace(realsense_sensor(depths, noise), :observation)

end
```

1: Generative Model

- Place holders for floor and ceiling

```
floor = Plane([0.,0.,0.],[0.,0.,1.])
room_height = @trace(uniform(2.5, 3.0), :room_height)
ceiling = Plane([0., 0., room_height][0., 0., -1.])
objects = [floor, ceiling]
```

1: Generative Model

- **4 random variables**
- which are necessary and sufficient for describing our model

```
room_height = @trace(uniform(2.5, 3.0), :room_height)
ceiling = Plane([0., 0., room_height][0., 0., -1.])
objects = [floor, ceiling]

camera_z = @trace(uniform(0.2, 2.0), :z)
camera_location = {0., 0., camera_z}

camera_pitch = @trace(uniform(..., ...), :pitch)
camera_roll= @trace(uniform(..., ...), :roll)
```

1: Generative Model

- Sampled from their **prior distributions**

```
room_height = @trace(uniform(2.5, 3.0), :room_height)
ceiling = Plane([0., 0., room_height][0., 0., -1.])
objects = [floor, ceiling]

camera_z = @trace(uniform(0.2, 2.0), :z)
camera_location = {0., 0., camera_z}

camera_pitch = @trace(uniform(..., ...), :pitch)
camera_roll= @trace(uniform(..., ...), :roll)
camera_rotation =
```

1: Generative Model

- And stored to the **trace**

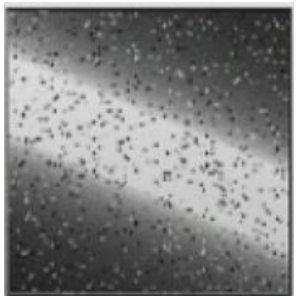
```
room_height = @trace(uniform(2.5, 3.0), :room_height)
ceiling = Plane([0., 0., room_height][0., 0., -1.])
objects = [floor, ceiling]

camera_z = @trace(uniform(0.2, 2.0), :z)
camera_location = {0., 0., camera_z}

camera_pitch = @trace(uniform(..., ...), :pitch)
camera_roll= @trace(uniform(..., ...), :roll)
```

1: Generative Model

- Render

```
# Create the image describing your parameters
depths = render(objects, camera_location, camera_rotation)
```

- Noise and likelihood



```
noise = 0.1
# Likelihood that the parameters came from this observation
@trace(realsense_sensor(depths, noise), :observation)
```

1: Generative Model

```
@gen function generative_model()

  floor = Plane([0.,0.,0.],[0.,0.,1.])
  room_height = @trace(uniform(2.5, 3.0), :room_height)
  ceiling = Plane([0., 0., room_height][0., 0., -1.])
  objects = [floor, ceiling]

  camera_z = @trace(uniform(0.2, 2.0), :z)
  camera_location = {0., 0., camera_z}

  camera_pitch = @trace(uniform(..., ...), :pitch)
  camera_roll= @trace(uniform(..., ...), :roll)
  camera_rotation =
    make_rotation_matrix(camera_pitch, 0., camera_roll)

  depths = render(objects, camera_location, camera_rotation)

  noise = 0.1
  @trace(realsense_sensor(depths, noise), :observation)

end
```

2: Inference Model

Again

- Step by step

```
#initialize trace with first observation
frame = get_frame(depth_camera)
constraints = Gen.choicemap((:observation, frame)
trace, = Gen.generate(generative_model, (), constraints)

#MCMC moves
for iter=1:1000

  # Global change
  trace, = Gen.mh(trace, Gen.select(:pitch, :roll, :z, :room_height))

  # Local changes
  trace, = Gen.mh(trance, random_walk, (pi/64, :pitch))
  trace, = Gen.mh(trance, random_walk, (pi/64, :roll))
  trace, = Gen.mh(trance, random_walk, (0.05, :z))
  trace, = Gen.mh(trance, random_walk, (0.05, :room_height))
end

return trace
```

2: Inference Model

- Initialize your parameters

```
# Get the camera frame
frame = get_frame(depth_camera)

# choicemap: :observation address from now on contains our "frame".
constraints = Gen.choicemap((:observation, frame)

# Initialization of parameters
trace, = Gen.generate(generative_model, (), constraints)
```

2: Inference Model

- Apply inference

```
#MCMC moves
for iter=1:1000

  # Global change
  trace, = Gen.mh(trace, Gen.select(:pitch, :roll, :z, :room_height))

  # Local changes
  trace, = Gen.mh(trance, random_walk, (pi/64, :pitch))
  trace, = Gen.mh(trance, random_walk, (pi/64, :roll))
  trace, = Gen.mh(trance, random_walk, (0.05, :z))
  trace, = Gen.mh(trance, random_walk, (0.05, :room_height))
end

return trace
```

2: Inference Model

- For 1000 iterations…

```
#MCMC moves
for iter=1:1000

  # Global change
  trace, = Gen.mh(trace, Gen.select(:pitch, :roll, :z, :room_height))

  # Local changes
  trace, = Gen.mh(trance, random_walk, (pi/64, :pitch))
  trace, = Gen.mh(trance, random_walk, (pi/64, :roll))
  trace, = Gen.mh(trance, random_walk, (0.05, :z))
  trace, = Gen.mh(trance, random_walk, (0.05, :room_height))
end
```

2: Inference Model

- Large changes

```
#MCMC moves
for iter=1:1000

  # Global change
  trace, = Gen.mh(trace, Gen.select(:pitch, :roll, :z, :room_height))

  # Local changes
  trace, = Gen.mh(trance, random_walk, (pi/64, :pitch))
  trace, = Gen.mh(trance, random_walk, (pi/64, :roll))
  trace, = Gen.mh(trance, random_walk, (0.05, :z))
  trace, = Gen.mh(trance, random_walk, (0.05, :room_height))
end
```

2: Inference Model

- Small changes

```
#MCMC moves
for iter=1:1000

  # Global change
  trace, = Gen.mh(trace, Gen.select(:pitch, :roll, :z, :room_height))

  # Local changes
  trace, = Gen.mh(trance, random_walk, (pi/64, :pitch))
  trace, = Gen.mh(trance, random_walk, (pi/64, :roll))
  trace, = Gen.mh(trance, random_walk, (0.05, :z))
  trace, = Gen.mh(trance, random_walk, (0.05, :room_height))
end
```

2: Inference Model

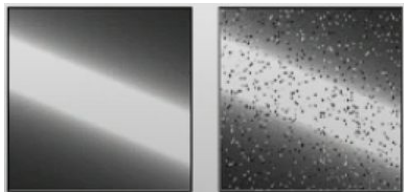
- With our choice of inference method (eg. Metropolis Hasting)

```
#MCMC moves
for iter=1:1000

  # Global change
  trace, = Gen.mh(trace, Gen.select(:pitch, :roll, :z, :room_height))

  # Local changes
  trace, = Gen.mh(trace, random_walk, (pi/64, :pitch))
  trace, = Gen.mh(trace, random_walk, (pi/64, :roll))
  trace, = Gen.mh(trace, random_walk, (0.05, :z))
  trace, = Gen.mh(trace, random_walk, (0.05, :room_height))
end
```

2: Inference Model

- And return final trace

:height = 2.7
:z = 0.9
:pitch = 1.96
:roll = -0.39
:observation = xyz….



```
#MCMC moves
for iter=1:1000

  # Global change
  trace, = Gen.mh(trace, Gen.select(:pitch, :roll, :z, :room_height))

  # Local changes
  trace, = Gen.mh(trace, random_walk, (pi/64, :pitch))
  trace, = Gen.mh(trace, random_walk, (pi/64, :roll))
  trace, = Gen.mh(trace, random_walk, (0.05, :z))
  trace, = Gen.mh(trace, random_walk, (0.05, :room_height))
end

return trace
```

Summarized:

1. Define the **parameters** that describe your model (“Generative Model”)

2. For *n* iterations, **apply inference** (e.g. metropolis hasting)

3. **Return** parameter values which describe the observations best

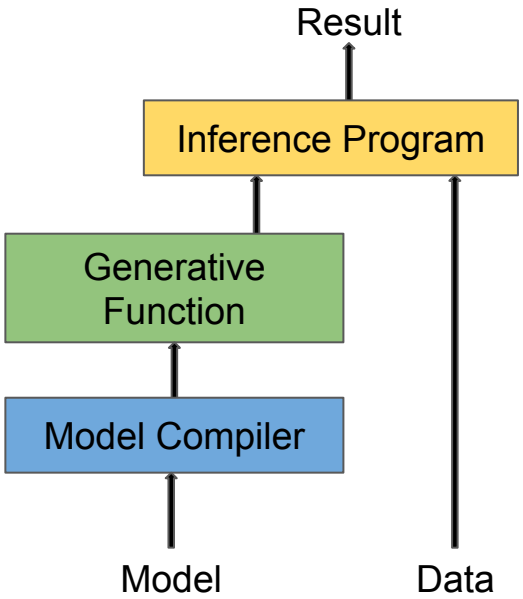
Some questions..

Let’s see what you have learned

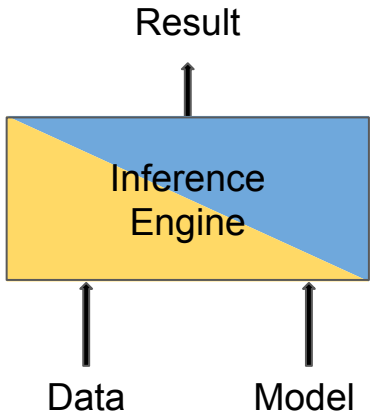
(and if you paid attention)

Question 1:

- Which picture represents Gen's architecture?



1)



2)

Question 2:

- If you decided to go for a different inference method, what part(s) of the code would you change?

```
#MCMC moves
for iter=1:1000

  # Global change
  trace, = Gen.mh(trace, Gen.select(:pitch, :roll, :z, :room_height))

  # Local changes
  trace, = Gen.mh(trace, random_walk, (pi/64, :pitch))
  trace, = Gen.mh(trace, random_walk, (pi/64, :roll))
  trace, = Gen.mh(trace, random_walk, (0.05, :z))
  trace, = Gen.mh(trace, random_walk, (0.05, :room_height))
end

return trace
```

Question 3:

- What is the use of the ‘Global change’ line? What may happen if it’s removed?

```
#MCMC moves
for iter=1:1000

  # Global change
  trace, = Gen.mh(trace, Gen.select(:pitch, :roll, :z, :room_height))

  # Local changes
  trace, = Gen.mh(trace, random_walk, (pi/64, :pitch))
  trace, = Gen.mh(trace, random_walk, (pi/64, :roll))
  trace, = Gen.mh(trace, random_walk, (0.05, :z))
  trace, = Gen.mh(trace, random_walk, (0.05, :room_height))
end

return trace
```

Conclusion:

- Flexibility
- Improved performance

Best practices for project

- Be prepared: learning curve is steep
- Do the tutorials on gen.dev
- Use visualisations while programming
- Get familiar with the different inference algorithms

TU Delft

Thank you for your attention!

TU Delft

Questions?

TU Delft