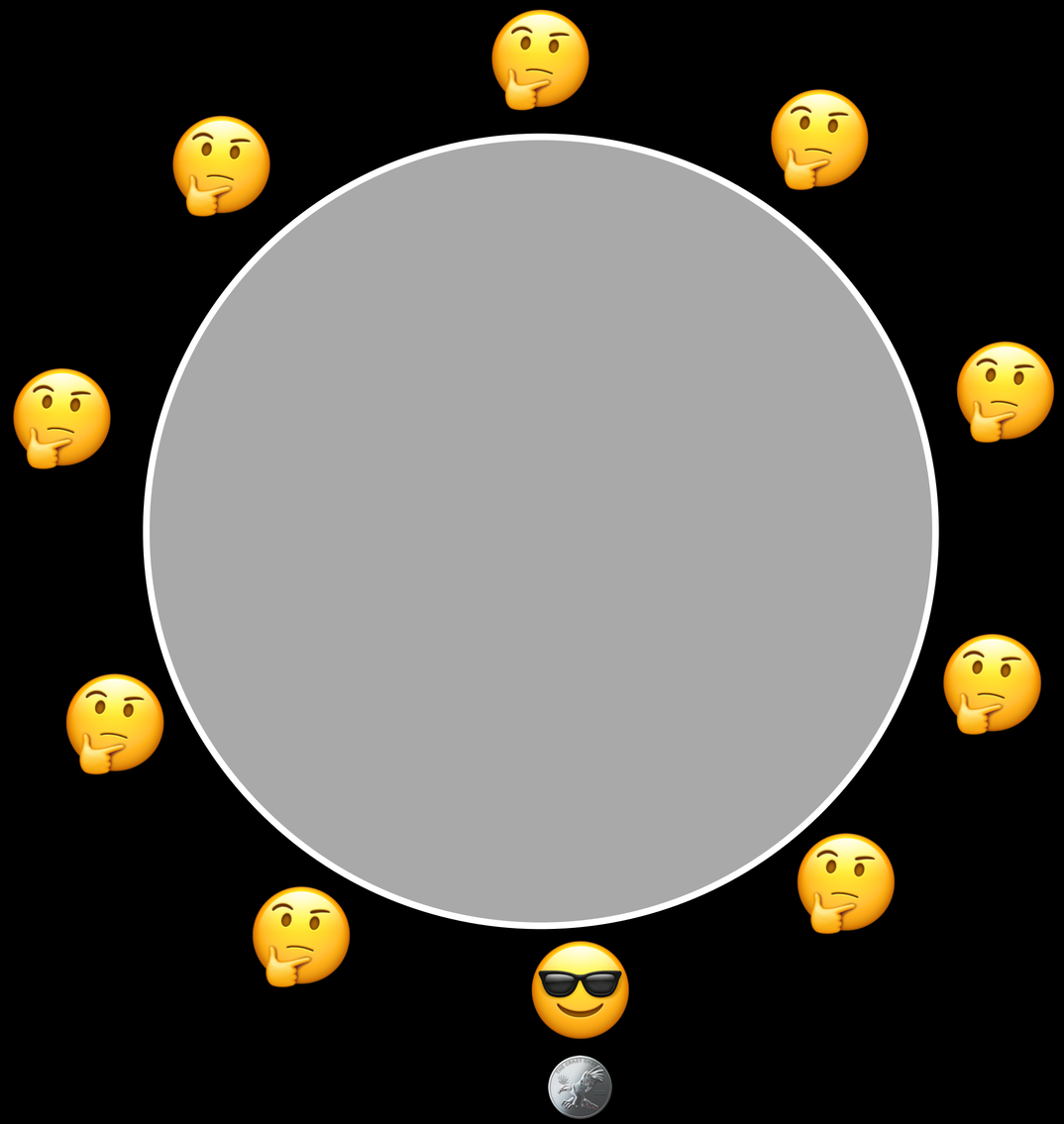
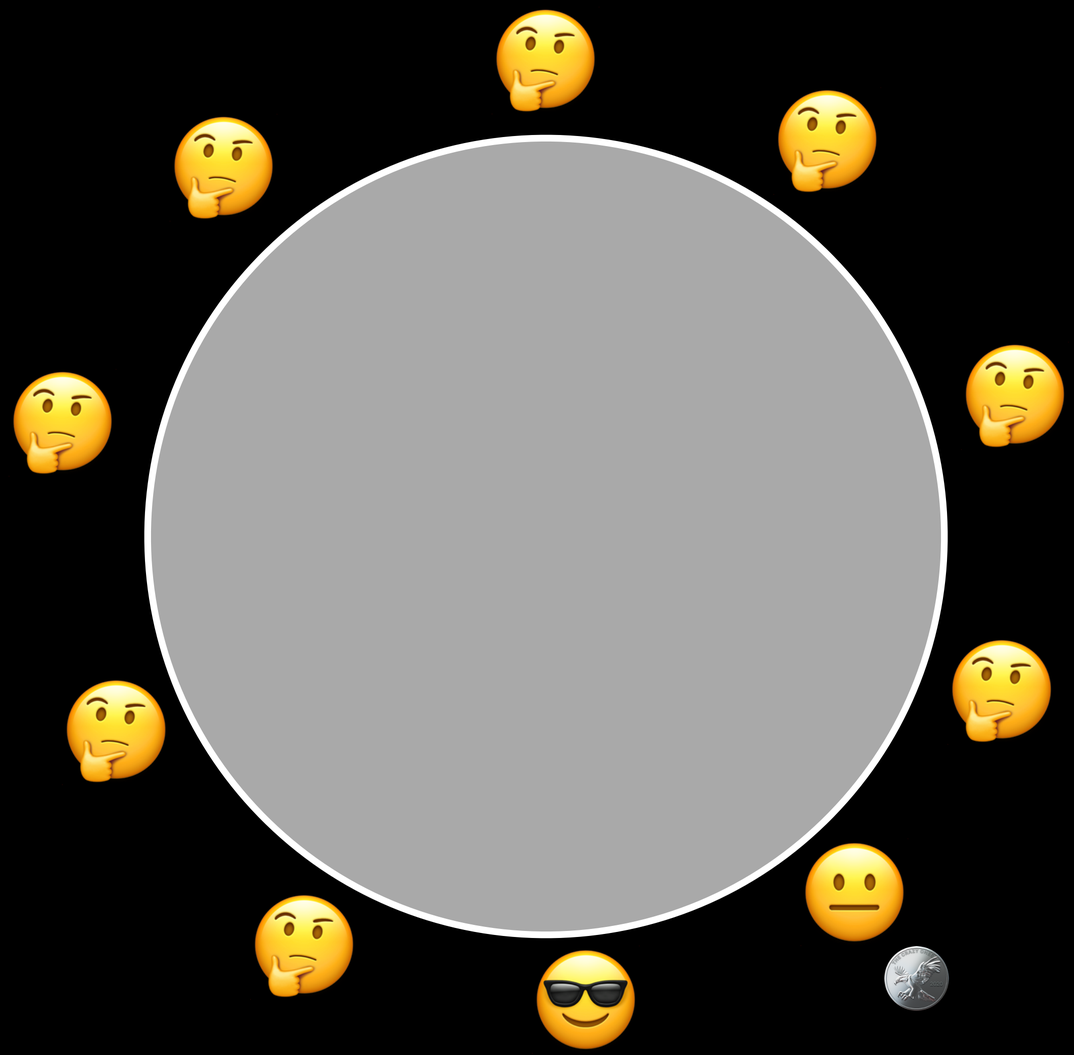
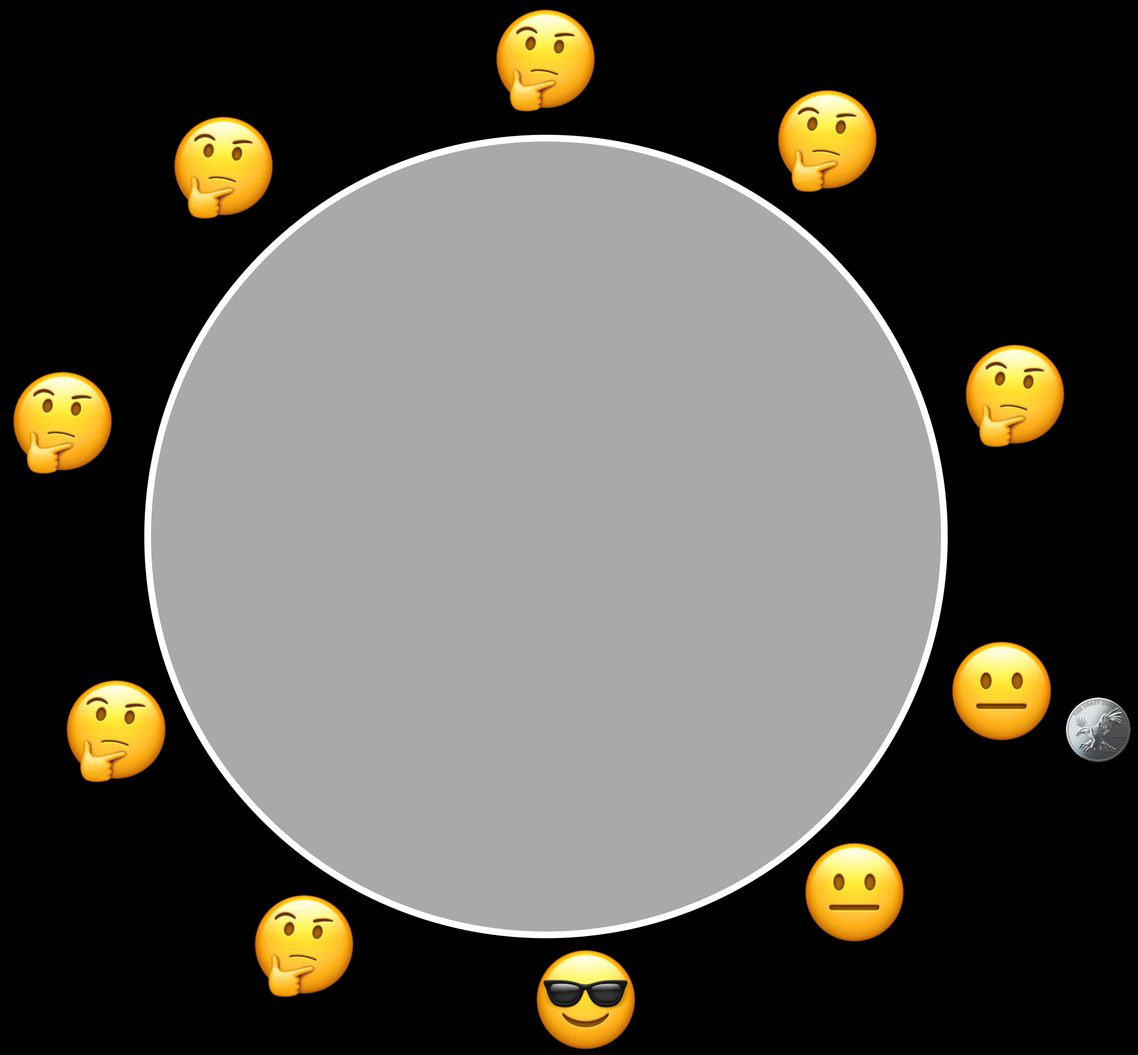
Probabilistic Programming

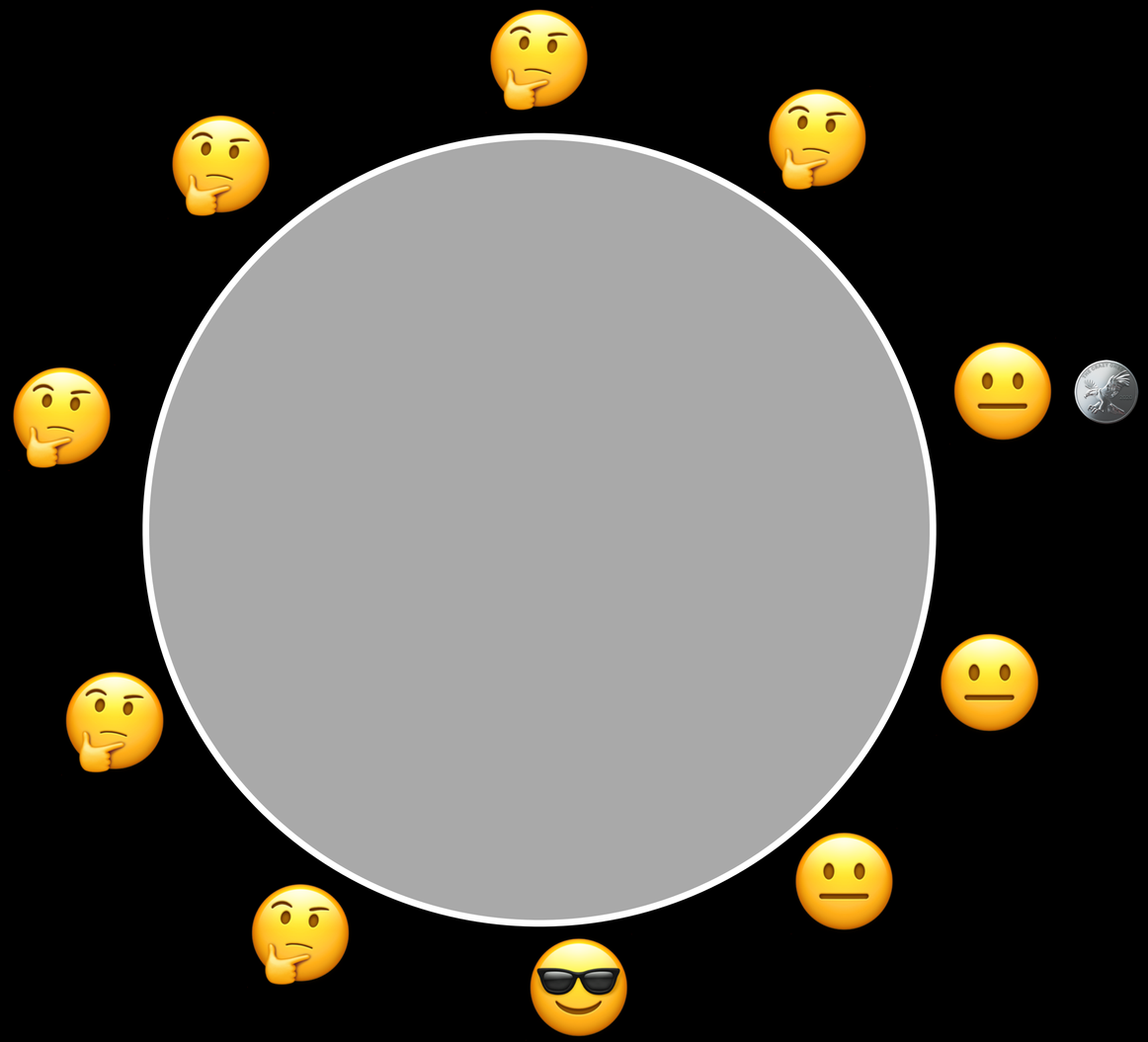
Session Preview

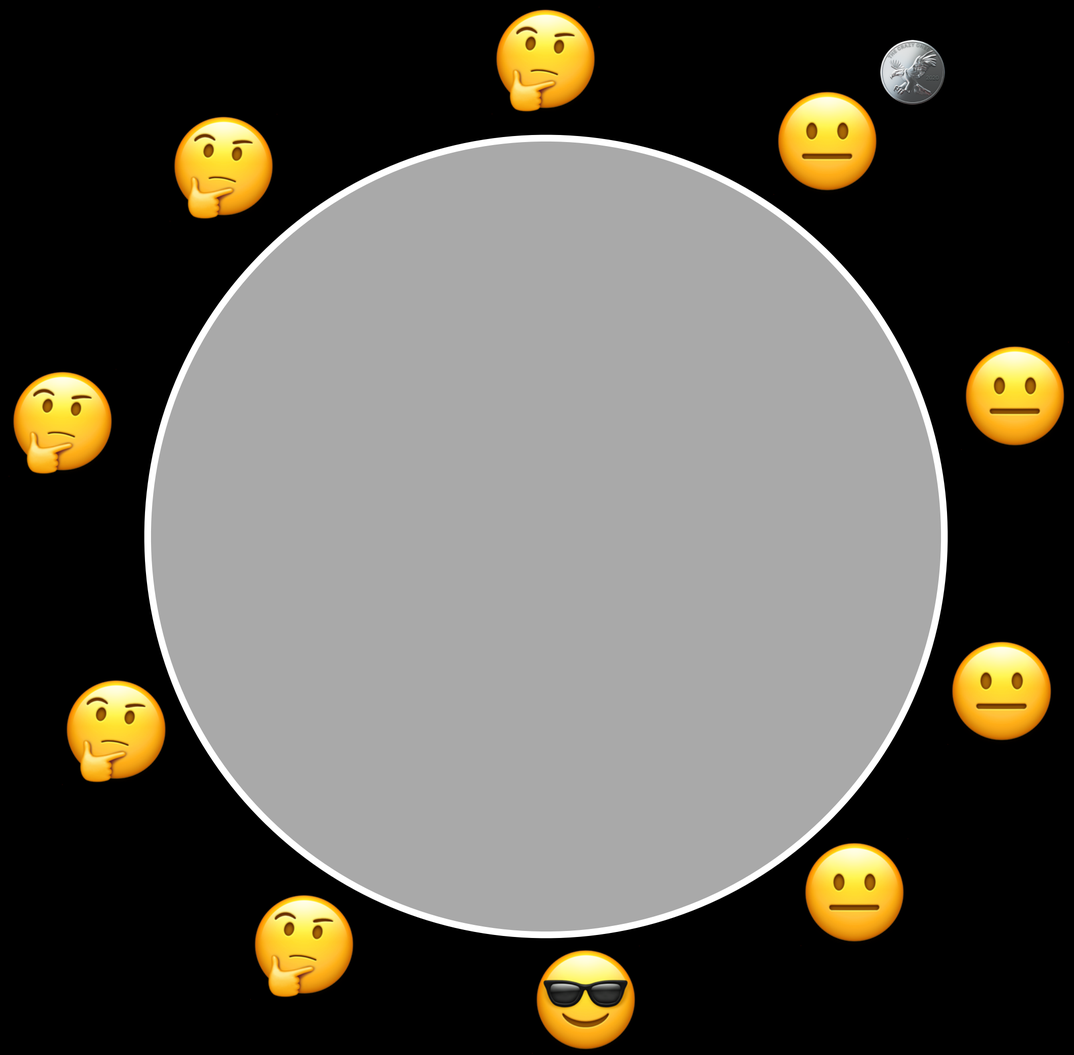
Jules Jacobs

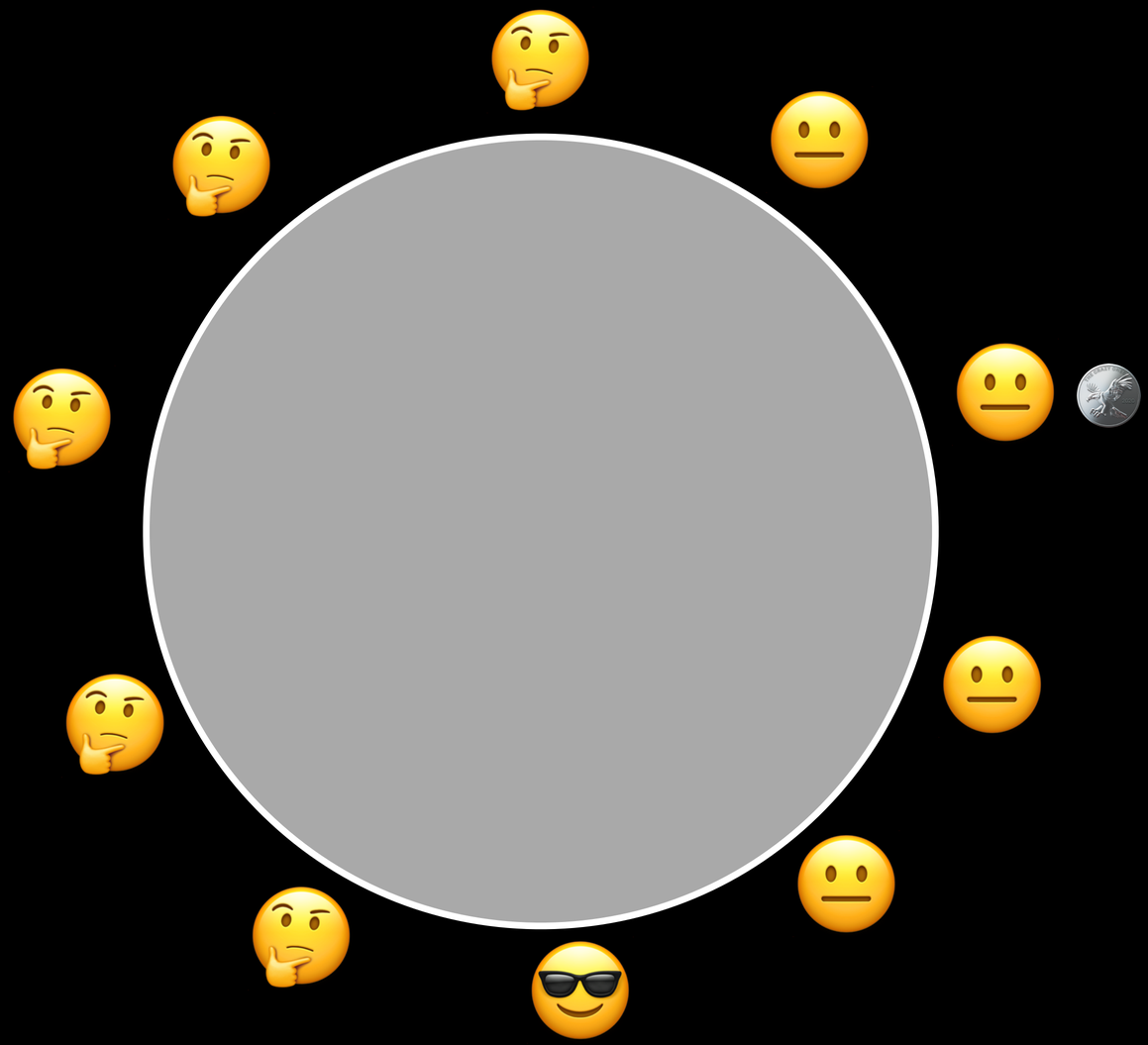


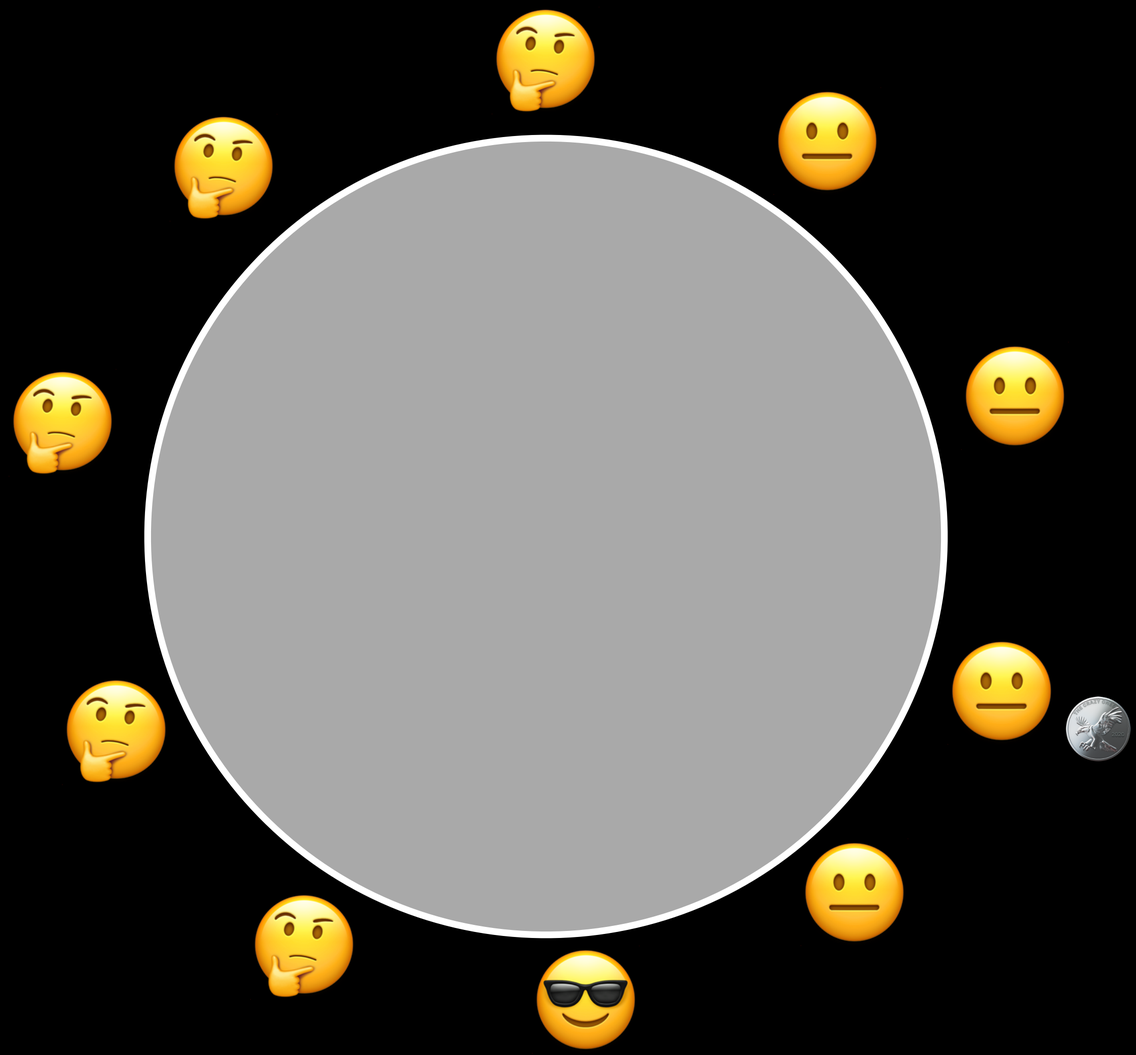


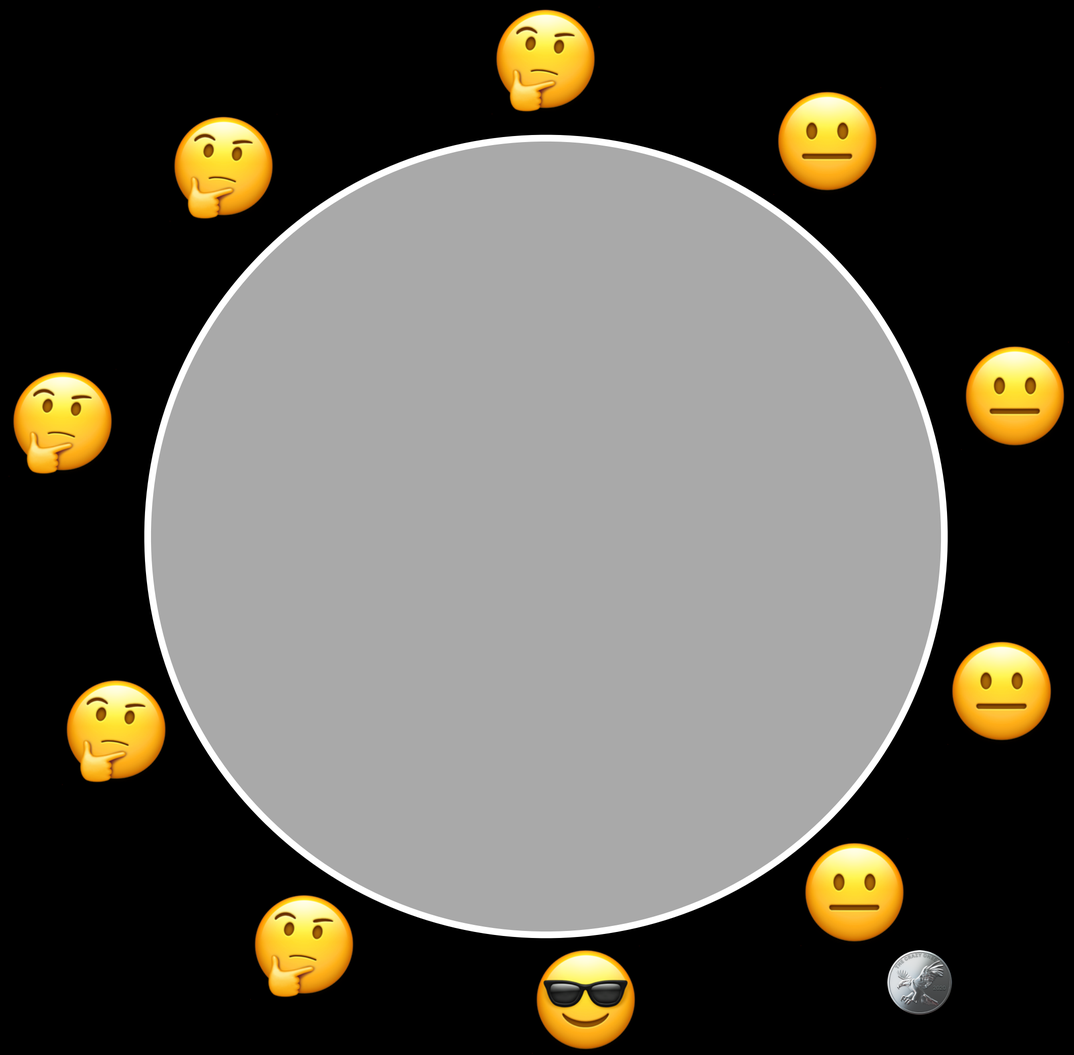


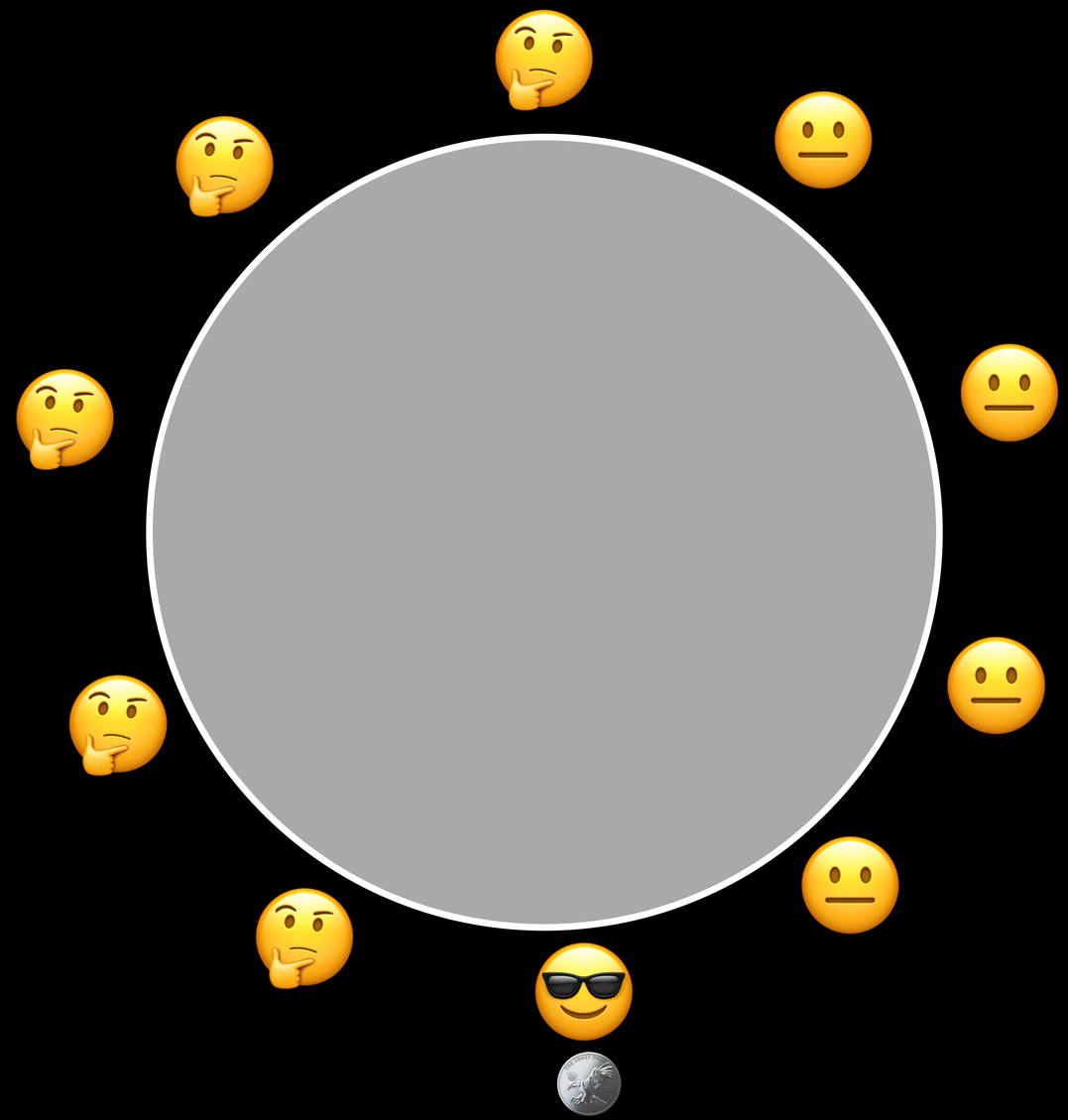


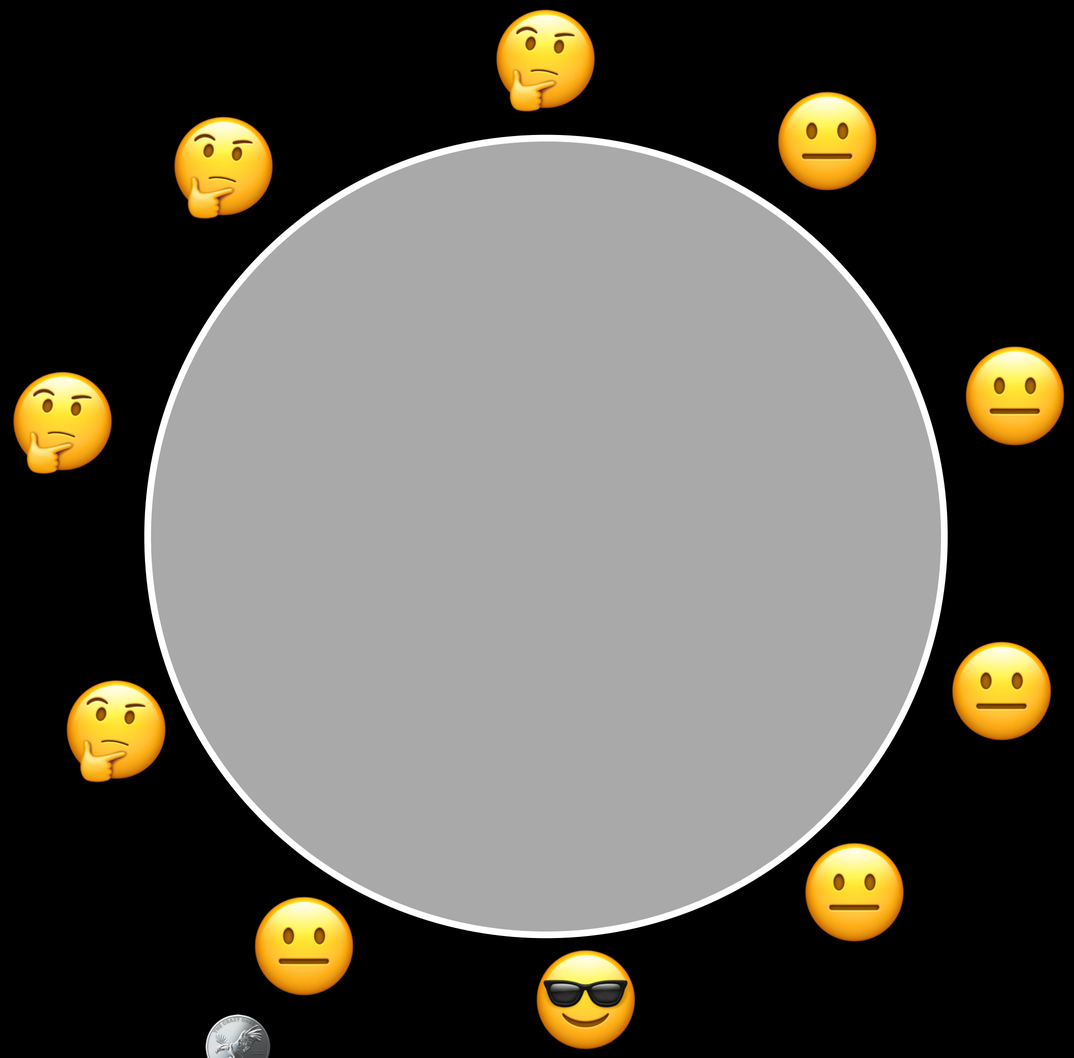


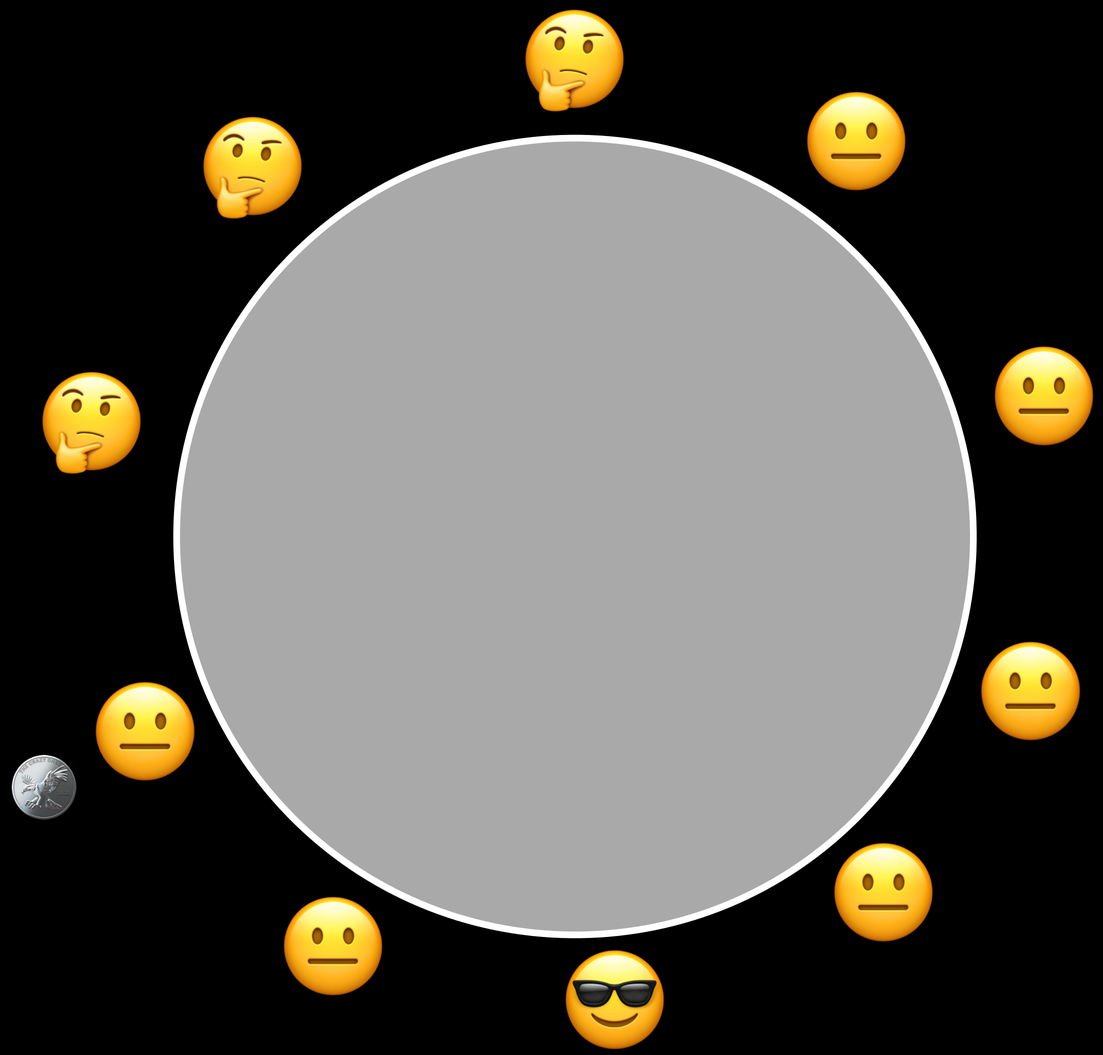


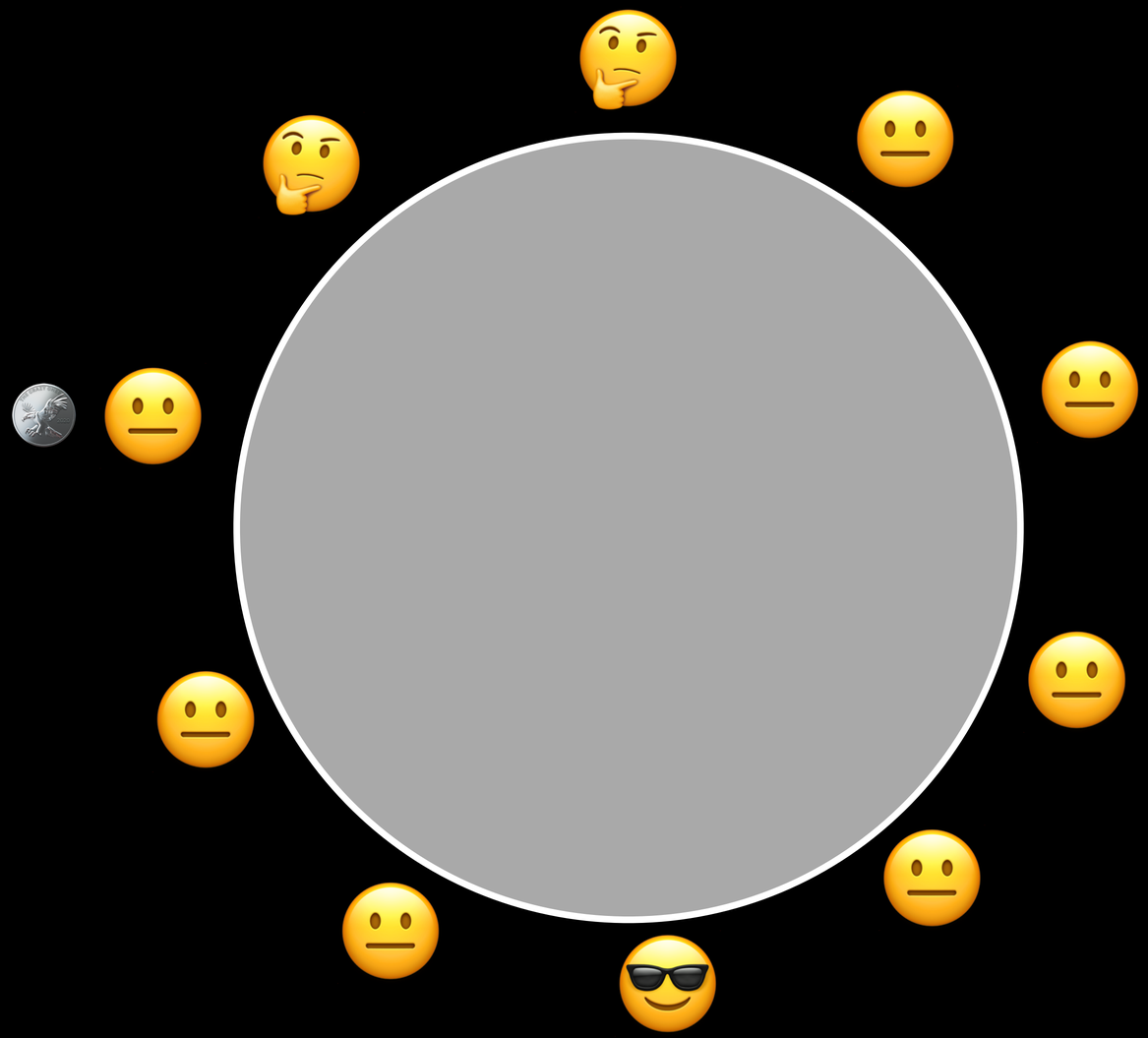


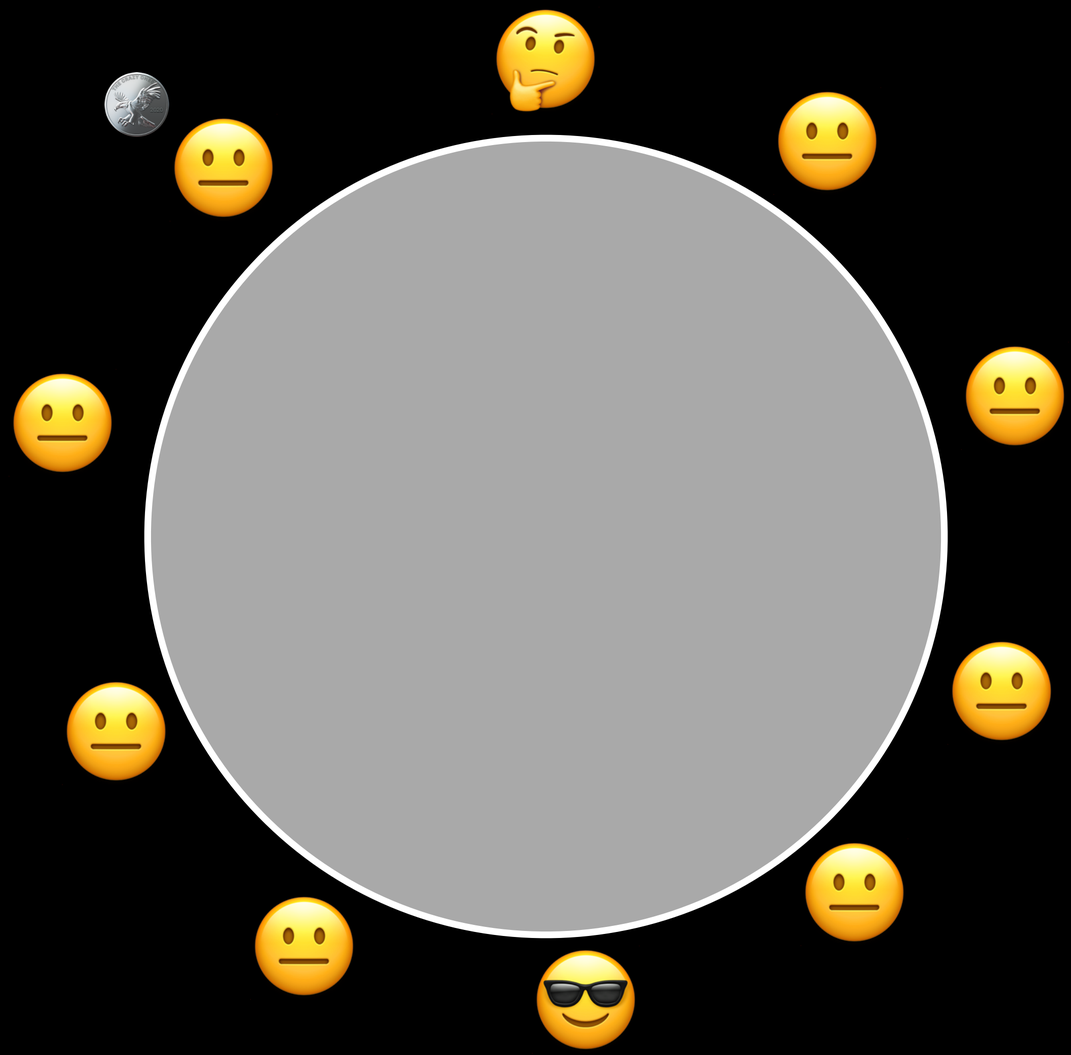


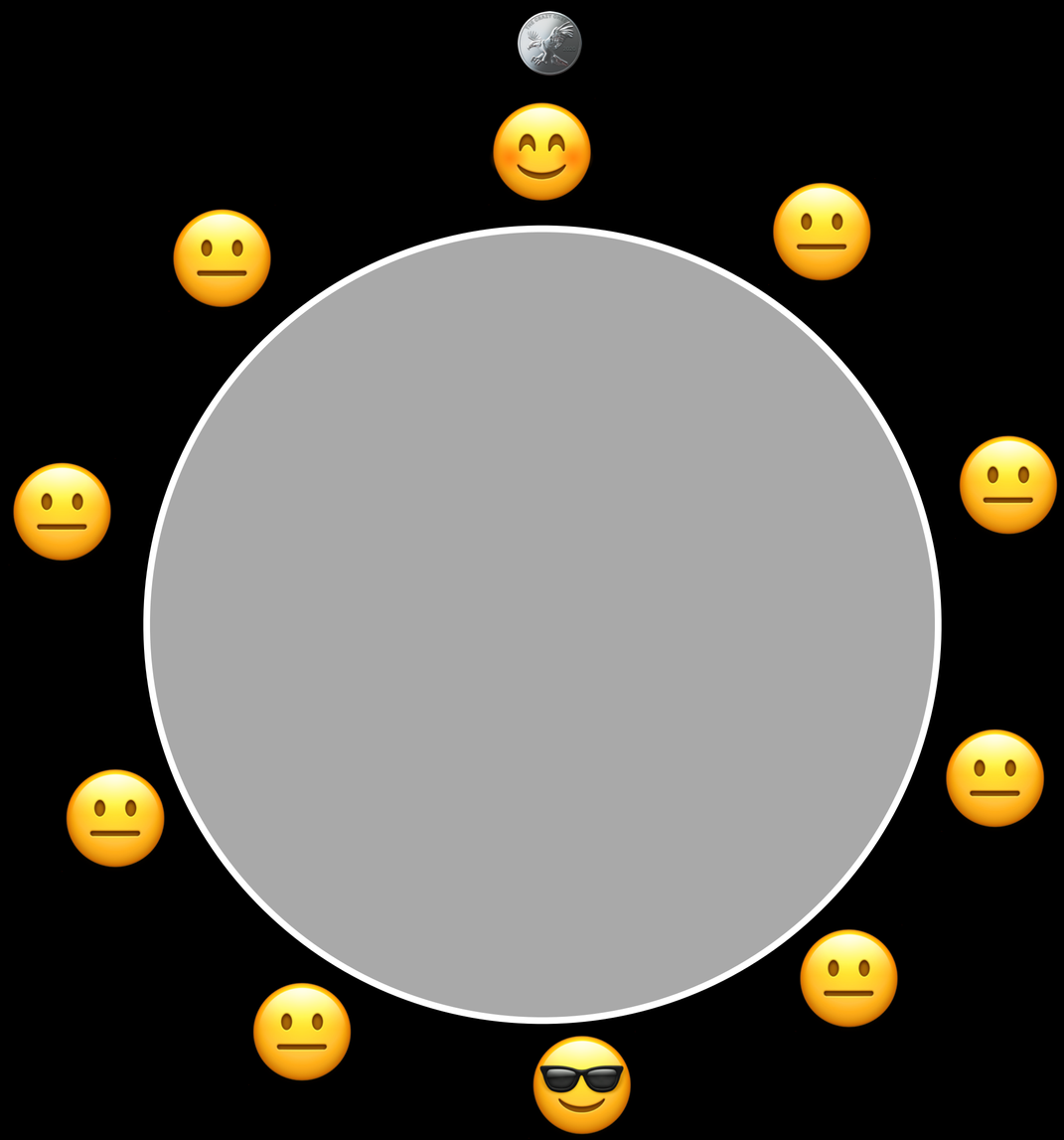


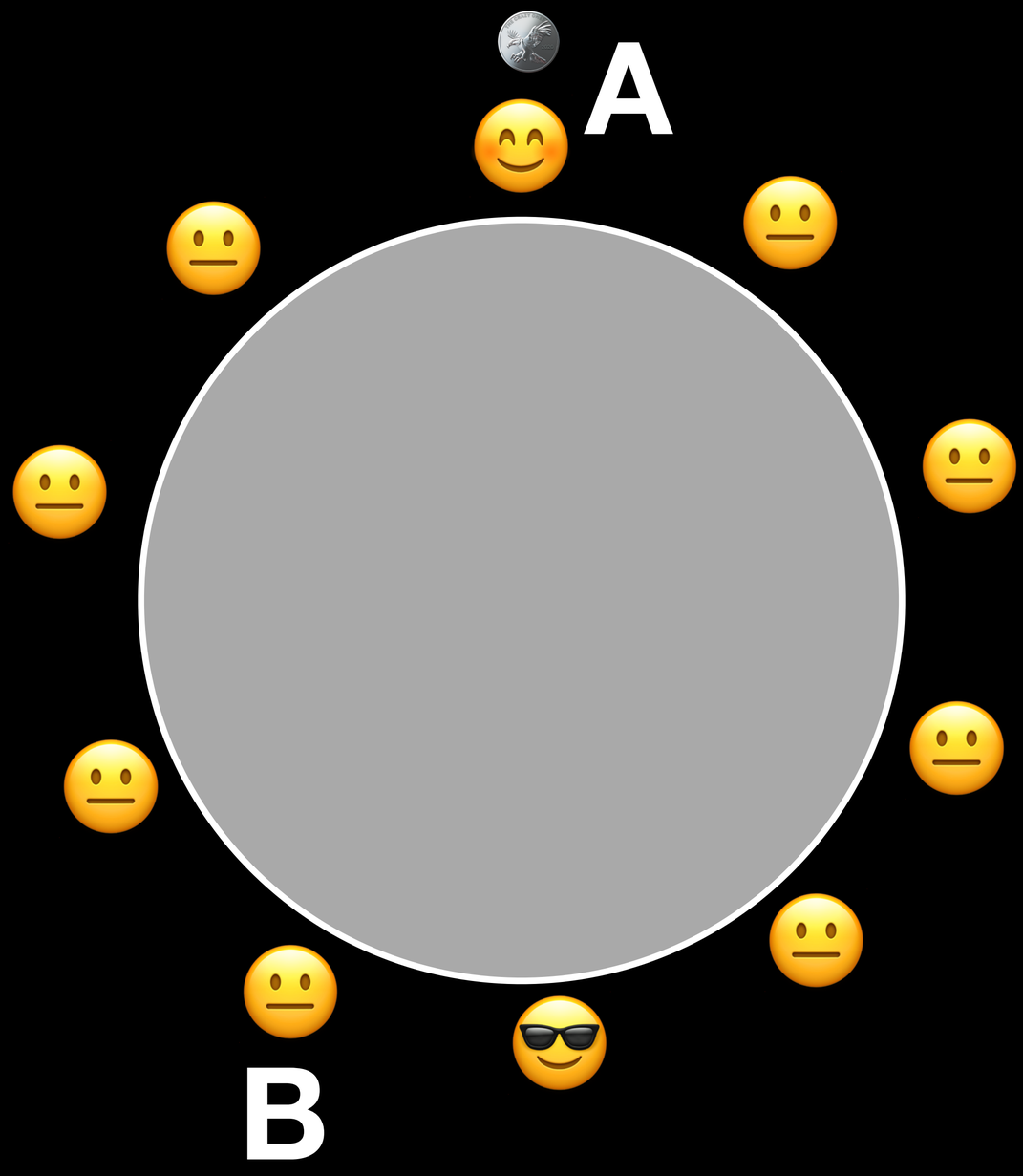








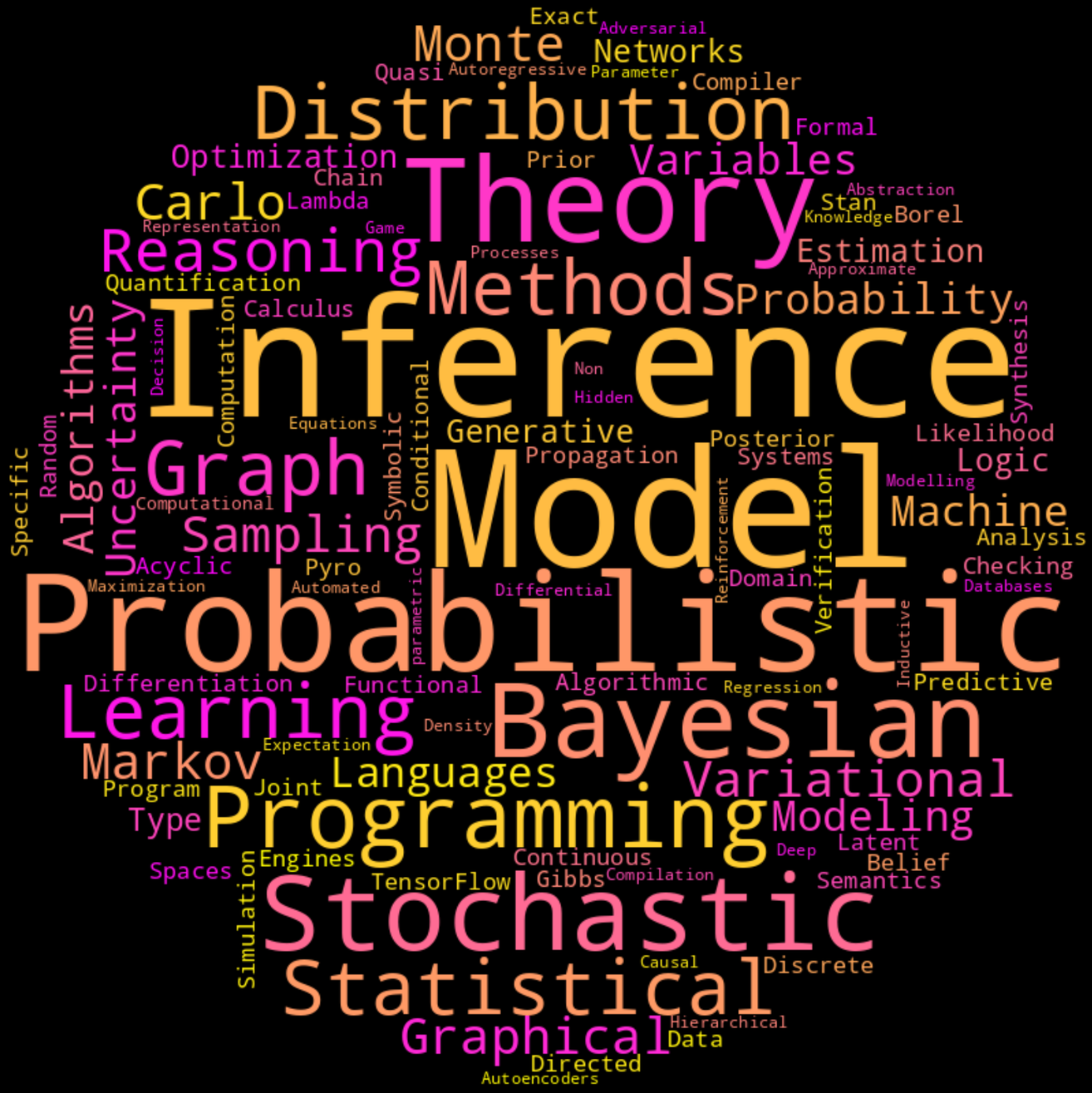



A
B

```
def play():
    n = 0
    H = {1..99}
    while H ≠ ∅:
        n = (n + flip()) % 100
        H = H ∖ {n}
    return n

def flip():
    if rand() < 0.5 then -1 else +1

results = []
while results.length < 100000:
    results.append(play())
```

```
def play():
    n = 0
    H = {1..99}
    while H ≠ ∅:
        n = (n + flip()) % 100
        H = H ∖ {n}
    observe(n == 1 or n == 50)
    return n

def flip():
    if rand() < 0.5 then -1 else +1

def observe(b): if not b then throw Skip

results = []
while results.length < 100000:
    try: results.append(play())
    catch Skip: continue
```

```
def play():
    n = 0
    H = {1..99}
    while H ≠ ∅:
        observe(1 ∈ H or 50 ∈ H)
        n = (n + flip()) % 100
        H = H ∖ {n}
    return n

def flip():
    if rand() < 0.5 then -1 else +1

def observe(b): if not b then throw Skip

results = []
while results.length < 100000:
    try: results.append(play())
    catch Skip: continue
```


Probabilistic
Programming
Model
Inference
Theory
Stochastic
Bayesian
Learning
Statistical
Distribution
Monte
Carlo
Reasoning
Methods
Graph
Sampling
Graphical
Markov
Variational
Languages
Networks
Optimization
Variables
Probability
Estimation
Machine
Logic
Algorithms
Uncertainty
Computation
Generative
Propagation
Posterior
Likelihood
Synthesis
Modeling
Verification
Checking
Analysis
Predictive
Differentiation
Functional
Algorithmic
Regression
Density
Expectation
Program
Joint
Type
Engines
Spaces
Simulation
TensorFlow
Gibbs
Continuous
Latent
Belief
Semantics
Discrete
Causal
Hierarchical
Data
Directed
Autoencoders
Exact
Adversarial
Quasi
Autoregressive
Parameter
Compiler
Formal
Prior
Chain
Lambda
Abstraction
Stan
Knowledge
Borel
Representation
Game
Processes
Approximate
Quantification
Calculus
Decision
Non
Hidden
Equations
Symbolic
Conditional
Specific
Random
Computational
Acyclic
Pyro
Maximization
Automated
Differential
parametric
Reinforcement
Domain
Databases
Inductive
Deep
Compilation

Probabilistic Programming Interfaces for Random Graphs: Markov Categories, Graphons, and Nominal Sets

NATE ACKERMAN, Harvard University, USA
CAMERON E. FREER, Massachusetts Institute of Technology, USA
YOUNESSE KADDAR, University of Oxford, UK
JACEK KARWOWSKI, University of Oxford, UK
SEAN MOSS, University of Birmingham, UK
DANIEL ROY, University of Toronto, Canada
SAM STATON, University of Oxford, UK
HONGSEOK YANG, School of Computing, KAIST, South Korea

Random graphs probabilistic programming

```
new : unit → vertex

edge : vertex * vertex → bool
```

Graphon ⟷ **Equational theory**

$P_{edge} : [0,1]^2 \to [0,1]$ ⟷ Programs A ≡ B

$$\begin{pmatrix} \mathrm{edge}(x_1, x_1) & \dots & \mathrm{edge}(x_1, x_n) \\ \vdots & & \vdots \\ \mathrm{edge}(x_n, x_1) & \dots & \mathrm{edge}(x_n, x_n) \end{pmatrix}$$

Probabilistic Programming Interfaces for Random Graphs: Markov Categories, Graphons, and Nominal Sets

NATE ACKERMAN, Harvard University, USA
CAMERON E. FREER, Massachusetts Institute of Technology, USA
YOUNESSE KADDAR, University of Oxford, UK
JACEK KARWOWSKI, University of Oxford, UK
SEAN MOSS, University of Birmingham, UK
DANIEL ROY, University of Toronto, Canada
SAM STATON, University of Oxford, UK
HONGSEOK YANG, School of Computing, KAIST, South Korea

We study semantic models of probabilistic programming languages over graphs, and establish a connection to graphons from graph theory and combinatorics. We show that every well-behaved equational theory for our graph probabilistic programming language corresponds to a graphon, and conversely, every graphon arises in this way.

We provide three constructions for showing that every graphon arises from an equational theory. The first is an abstract construction, using Markov categories and monoidal indeterminates. The second and third are more concrete. The second is in terms of traditional measure theoretic probability, which covers 'black-and-white' graphons. The third is in terms of probability monads on the nominal sets of Gabbay and Pitts. Specifically, we use a variation of nominal sets induced by the theory of graphs, which covers Erdős-Rényi graphons. In this way, we build new models of graph probabilistic programming from graphons.

CCS Concepts: • **Theory of computation** → **Semantics and reasoning**; **Probabilistic computation**.

Additional Key Words and Phrases: probability monads, exchangeable processes, graphons, nominal sets, Markov categories, probabilistic programming

ACM Reference Format:
Nate Ackerman, Cameron E. Freer, Younesse Kaddar, Jacek Karwowski, Sean Moss, Daniel Roy, Sam Staton, and Hongseok Yang. 2024. Probabilistic Programming Interfaces for Random Graphs: Markov Categories, Graphons, and Nominal Sets. *Proc. ACM Program. Lang.* 8, POPL, Article 61 (January 2024), 32 pages. https://doi.org/10.1145/3632903

1 INTRODUCTION

This paper is about the semantic structures underlying probabilistic programming with random graphs. Random graphs have applications in statistical modelling across biology, chemistry, epidemiology, and so on, as well as theoretical interest in graph theory and combinatorics (e.g. [Bornholdt and Schuster 2002]). Probabilistic programming, i.e. programming for statistical modelling [van de

Authors' addresses: Nate Ackerman, Harvard University, USA, nate@aleph0.net; Cameron E. Freer, Massachusetts Institute of Technology, USA, freer@mit.edu; Younesse Kaddar, University of Oxford, UK, younesse.kaddar@chch.ox.ac.uk; Jacek Karwowski, University of Oxford, UK, jacek.karwowski@cs.ox.ac.uk; Sean Moss, University of Birmingham, UK, s.k.moss@bham.ac.uk; Daniel Roy, University of Toronto, Canada, daniel.roy@utoronto.ca; Sam Staton, University of Oxford, UK, sam.staton@cs.ox.ac.uk; Hongseok Yang, School of Computing, KAIST, South Korea, hongseok00@gmail.com.

ACM 2475-1421/2024/1-ART61
https://doi.org/10.1145/3632903

Inference of Probabilistic Programs with Moment-Matching Gaussian Mixtures

FRANCESCA RANDONE, IMT School for Advanced Studies Lucca, Italy
LUCA BORTOLUSSI, University of Trieste, Italy
EMILIO INCERTO, IMT School for Advanced Studies Lucca, Italy
MIRCO TRIBASTONE, IMT School for Advanced Studies Lucca, Italy

Inference of Probabilistic Programs with Moment-Matching Gaussian Mixtures

FRANCESCA RANDONE, IMT School for Advanced Studies Lucca, Italy
LUCA BORTOLUSSI, University of Trieste, Italy
EMILIO INCERTO, IMT School for Advanced Studies Lucca, Italy
MIRCO TRIBASTONE, IMT School for Advanced Studies Lucca, Italy

Computing the posterior distribution of a probabilistic program is a hard task for which no one-fit-for-all solution exists. We propose Gaussian Semantics, which approximates the exact probabilistic semantics of a bounded program by means of Gaussian mixtures. It is parametrized by a map that associates each program location with the moment order to be matched in the approximation. We provide two main contributions. The first is a universal approximation theorem stating that, under mild conditions, Gaussian Semantics can approximate the exact semantics arbitrarily closely. The second is an approximation that matches up to second-order moments analytically in face of the generally difficult problem of matching moments of Gaussian mixtures with arbitrary moment order. We test our second-order Gaussian approximation (SOGA) on a number of case studies from the literature. We show that it can provide accurate estimates in models not supported by other approximation methods or when exact symbolic techniques fail because of complex expressions or non-simplified integrals. On two notable classes of problems, namely collaborative filtering and programs involving mixtures of continuous and discrete distributions, we show that SOGA significantly outperforms alternative techniques in terms of accuracy and computational time.

CCS Concepts: • **Theory of computation → Denotational semantics**; • **Mathematics of computing → Probabilistic reasoning algorithms**.

Additional Key Words and Phrases: probabilistic programming, inference, Gaussian mixtures

ACM Reference Format:
Francesca Randone, Luca Bortolussi, Emilio Incerto, and Mirco Tribastone. 2024. Inference of Probabilistic Programs with Moment-Matching Gaussian Mixtures. *Proc. ACM Program. Lang.* 8, POPL, Article 63 (January 2024), 31 pages. https://doi.org/10.1145/3632905

1 INTRODUCTION

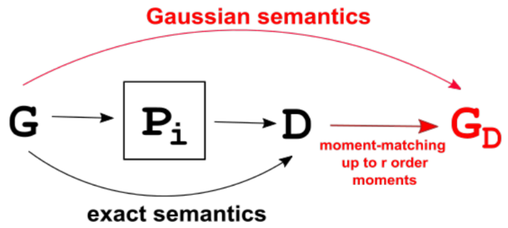
Probabilistic programming languages are programming languages augmented with primitives expressing probabilistic behaviours [Gordon et al. 2014]. Examples are random assignments ("program variable x is distributed according to the probability distribution D"), probabilistic choices ("do P_1 with probability p else P_2) or conditioning ("variable x is distributed according to D, under the constraint that it can only take positive values"). This has enabled a variety of applications such as the analysis of randomized algorithms, machine learning and biology [Gordon et al. 2014].

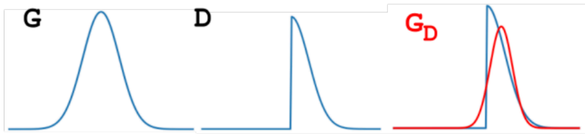
Given a probabilistic program, there are different equivalent ways in which its semantics can be defined [Kozen 1983]. Following Kozen's Semantics 2 [Kozen 1979], in this paper we see a program as a transformer: given an initial joint distribution over the program variables, each instruction

Authors' addresses: Francesca Randone, IMT School for Advanced Studies Lucca, Italy, francesca.randone@imtlucca.it; Luca Bortolussi, University of Trieste, Italy, lbortolussi@units.it; Emilio Incerto, IMT School for Advanced Studies Lucca, Italy, emilio.incerto@imtlucca.it; Mirco Tribastone, IMT School for Advanced Studies Lucca, Italy, mirco.tribastone@imtlucca.it.

ACM 2475-1421/2024/1-ART63
https://doi.org/10.1145/3632905

Gaussian Semantics





Mixtures of Gaussians

Algorithm 2 *SOGA*(*node*)

```
1: input_list = []
2: for par in node.parents do
3:     input_list.append((par.p, par.dist))
4: end for
5: input_p, input_dist = merge_dist(input_list)
6: node.p, node.dist = node_semantics(node, input_p, input_dist)
7: for child in node.children do
8:     SOGA(child)
9: end for
```

Propagate along control flow DAG

Merge mixture at join points

Higher Order Bayesian Networks, Exactly

CLAUDIA FAGGIAN, IRIF, CNRS, Université Paris Cité, France
DANIELE PAUTASSO, University of Turin, Italy
GABRIELE VANONI, IRIF, CNRS, Université Paris Cité, France

Higher Order Bayesian Networks, Exactly

CLAUDIA FAGGIAN, IRIF, CNRS, Université Paris Cité, France
DANIELE PAUTASSO, University of Turin, Italy
GABRIELE VANONI, IRIF, CNRS, Université Paris Cité, France

Bayesian networks are graphical *first-order* probabilistic models that allow for a compact representation of large probability distributions, and for efficient inference, both exact and approximate. We introduce a *higher-order* programming language—in the idealized form of a λ-calculus—which we prove *sound and complete* w.r.t. Bayesian networks: each Bayesian network can be encoded as a term, and conversely each (possibly higher-order and recursive) program of ground type *compiles* into a Bayesian network.

The language allows for the specification of recursive probability models and hierarchical structures. Moreover, we provide a *compositional* and *cost-aware* semantics which is based on factors, the standard mathematical tool used in Bayesian inference. Our results rely on advanced techniques rooted into linear logic, intersection types, rewriting theory, and Girard's geometry of interaction, which are here combined in a novel way.

CCS Concepts: • **Theory of computation** → **Lambda calculus**; **Probabilistic computation**; **Linear logic**; **Type theory**; **Denotational semantics**.

Additional Key Words and Phrases: lambda calculus, intersection types, Bayesian networks, probabilistic programming, denotational semantics

ACM Reference Format:
Claudia Faggian, Daniele Pautasso, and Gabriele Vanoni. 2024. Higher Order Bayesian Networks, Exactly. *Proc. ACM Program. Lang.* 8, POPL, Article 84 (January 2024), 33 pages. https://doi.org/10.1145/3632926

1 INTRODUCTION

This paper is a foundational study, taking a cost-aware approach to the semantics of higher-order probabilistic programming languages. Probabilistic models play a crucial role in several fields such as machine learning, cognitive science, and applied statistics, with applications spanning from finance to biology. A prominent example of such models are Bayesian networks (BNs) [Pearl 1988], a (first-order, static) graphical formalism able to represent complex systems in a *compact* way and enabling *efficient* inference algorithms. BNs decompose large joint distributions into smaller *factors*. These are used in inference algorithms, both exact (such as message passing and variable elimination) and approximate (sampling-based). Despite their significant strengths, the task of modeling using Bayesian networks is comparable to the task of programming using logical circuits.

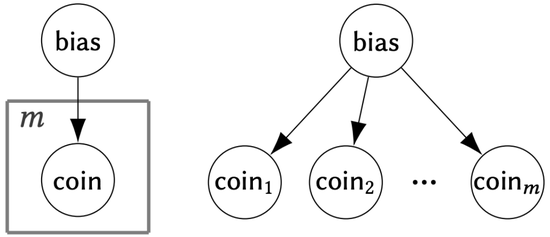
Probabilistic Programming Languages. A different approach is taken by *probabilistic programming languages* (PPLs), where statistical models are specified as programs. The fundamental idea behind PPLs is to separate the model description—the program—from the computation of the probability distribution specified by the program—the inference task. This separation aims at making stochastic modeling as accessible as possible, hiding the underlying inference engines, which typically encompass various sampling methods such as importance sampling, Markov Chain Monte Carlo, and

Authors' addresses: Claudia Faggian, IRIF, CNRS, Université Paris Cité, France, faggian@irif.fr; Daniele Pautasso, University of Turin, Italy, daniele.pautasso@unito.it; Gabriele Vanoni, IRIF, CNRS, Université Paris Cité, France, gabriele.vanoni@irif.fr.

ACM 2475-1421/2024/1-ART84
https://doi.org/10.1145/3632926

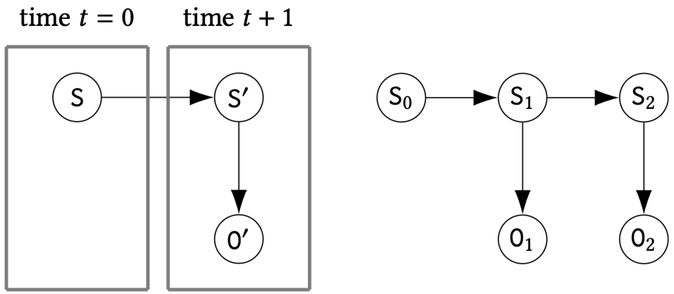
Statisticians

Plate Notation for Bayesian Networks



(a) Plate (b) Unrolling m times

Fig. 3. Modeling m coin tosses.



(a) Template (b) Unrolling (2 times)

Fig. 4. Discrete Time Dynamic Bayesian network.

This paper

Lambda calculus notation

Terms normalise to Bayesian Networks

Strong Invariants Are Hard

On the Hardness of Strongest Polynomial Invariants for (Probabilistic) Programs

JULIAN MÜLLNER, TU Wien, Austria

MARCEL MOOSBRUGGER, TU Wien, Austria

LAURA KOVÁCS, TU Wien, Austria

Strong Invariants Are Hard

On the Hardness of Strongest Polynomial Invariants for (Probabilistic) Programs

JULIAN MÜLLNER, TU Wien, Austria

MARCEL MOOSBRUGGER, TU Wien, Austria

LAURA KOVÁCS, TU Wien, Austria

We show that computing the strongest polynomial invariant for single-path loops with polynomial assignments is at least as hard as the SKOLEM problem, a famous problem whose decidability has been open for almost a century. While the strongest polynomial invariants are computable for *affine loops*, for polynomial loops the problem remained wide open. As an intermediate result of independent interest, we prove that reachability for discrete polynomial dynamical systems is SKOLEM-hard as well. Furthermore, we generalize the notion of invariant ideals and introduce *moment invariant ideals* for probabilistic programs. With this tool, we further show that the strongest polynomial moment invariant is (i) uncomputable, for probabilistic loops with branching statements, and (ii) SKOLEM-hard to compute for polynomial probabilistic loops without branching statements. Finally, we identify a class of probabilistic loops for which the strongest polynomial moment invariant is computable and provide an algorithm for it.

CCS Concepts: • **Theory of computation** → **Invariants**; **Probabilistic computation**; *Computability*; Random walks and Markov chains.

Additional Key Words and Phrases: Strongest algebraic invariant, Point-To-Point reachability, Skolem problem, Probabilistic programs

ACM Reference Format:

Julian Müllner, Marcel Moosbrugger, and Laura Kovács. 2024. Strong Invariants Are Hard: On the Hardness of Strongest Polynomial Invariants for (Probabilistic) Programs. *Proc. ACM Program. Lang.* 8, POPL, Article 30 (January 2024), 29 pages. https://doi.org/10.1145/3632872

1 INTRODUCTION

Loop invariants describe valid program properties that hold before and after every loop iteration. Intuitively, invariants provide correctness information that may prevent programmers from introducing errors while making changes to the loop. As such, invariants are fundamental to formalizing program semantics as well as to automate the formal analysis and verification of programs. While automatically synthesizing loop invariants is, in general, an uncomputable problem, when considering only single-path loops with linear updates (linear loops), the strongest polynomial invariant is in fact computable [Karr 1976; Kovács 2008; Müller-Olm and Seidl 2004a]. The computability

Authors' addresses: Julian Müllner, TU Wien, Vienna, Austria, julian.muellner@tuwien.ac.at; Marcel Moosbrugger, TU Wien, Vienna, Austria, marcel.moosbrugger@tuwien.ac.at; Laura Kovács, TU Wien, Vienna, Austria, laura.kovacs@tuwien.ac.at.

ACM 2475-1421/2024/1-ART30
https://doi.org/10.1145/3632872

$$\begin{bmatrix} f & g & x_1 & \dots & x_k \end{bmatrix} \leftarrow \begin{bmatrix} 1 & 0 & u_1(0) & \dots & u_k(0) \end{bmatrix}$$

while ⋆ **do**

$$\begin{bmatrix} x_1 \\ \vdots \\ x_k \\ f \\ g \end{bmatrix} \leftarrow \begin{bmatrix} p_1(x_1, \dots, x_k) \\ \vdots \\ p_k(x_1, \dots, x_k) \\ f \cdot \left((x_1 - t_1)^2 + \dots + (x_k - t_k)^2\right) \\ g + 1 \end{bmatrix}$$

end while

The SKOLEM Problem [Everest et al. 2003; Tao 2008]: Does a given linear recurrence sequence with constant coefficients have a zero?

The Point-To-Point Reachability Problem (P2P) : Given a single-path loop with polynomial updates, is a given target state reachable starting from a given initial state?

The SPINV Problem: Given a single-path loop with polynomial updates, compute the strongest polynomial invariant.

Skolem ≤ P2P ≤ SPInv

What happens in the probabilistic setting?

Probabilistic Programming Interfaces for Random Graphs: Markov Categories, Graphons, and Nominal Sets

NATE ACKERMAN, Harvard University, USA
CAMERON E. FREER, Massachusetts Institute of Technology, USA
YOUNESSE KADDAR, University of Oxford, UK
JACEK KARWOWSKI, University of Oxford, UK
SEAN MOSS, University of Birmingham, UK
DANIEL ROY, University of Toronto, Canada
SAM STATON, University of Oxford, UK
HONGSEOK YANG, School of Computing, KAIST, South Korea

We study semantic models of probabilistic programming languages over graphs, and establish a connection to graphons from graph theory and combinatorics. We show that every well-behaved equational theory for our graph probabilistic programming language corresponds to a graphon, and conversely, every graphon arises in this way.

We provide three constructions for showing that every graphon arises from an equational theory. The first is an abstract construction, using Markov categories and monoidal indeterminates. The second and third are more concrete. The second is in terms of traditional measure theoretic probability, which covers 'black-and-white' graphons. The third is in terms of probability monads on the nominal sets of Gabbay and Pitts. Specifically, we use a variation of nominal sets induced by the theory of graphs, which covers Erdős-Rényi graphons. In this way, we build new models of graph probabilistic programming from graphons.

CCS Concepts: • **Theory of computation → Semantics and reasoning**; **Probabilistic computation**.

Additional Key Words and Phrases: probability monads, exchangeable processes, graphons, nominal sets, Markov categories, probabilistic programming

ACM Reference Format:
Nate Ackerman, Cameron E. Freer, Younesse Kaddar, Jacek Karwowski, Sean Moss, Daniel Roy, Sam Staton, and Hongseok Yang. 2024. Probabilistic Programming Interfaces for Random Graphs: Markov Categories, Graphons, and Nominal Sets. *Proc. ACM Program. Lang.* 8, POPL, Article 61 (January 2024), 32 pages. https://doi.org/10.1145/3632903

1 INTRODUCTION

This paper is about the semantic structures underlying probabilistic programming with random graphs. Random graphs have applications in statistical modelling across biology, chemistry, epidemiology, and so on, as well as theoretical interest in graph theory and combinatorics (e.g. [Bornholdt and Schuster 2002]). Probabilistic programming, i.e. programming for statistical modelling [van de

Authors' addresses: Nate Ackerman, Harvard University, USA, nate@alephθ.net; Cameron E. Freer, Massachusetts Institute of Technology, USA, freer@mit.edu; Younesse Kaddar, University of Oxford, UK, younesse.kaddar@chch.ox.ac.uk; Jacek Karwowski, University of Oxford, UK, jacek.karwowski@cs.ox.ac.uk; Sean Moss, University of Birmingham, UK, s.k.moss@bham.ac.uk; Daniel Roy, University of Toronto, Canada, daniel.roy@utoronto.ca; Sam Staton, University of Oxford, UK, sam.staton@cs.ox.ac.uk; Hongseok Yang, School of Computing, KAIST, South Korea, hongseok00@gmail.com.

ACM 2475-1421/2024/1-ART61
https://doi.org/10.1145/3632903

Inference of Probabilistic Programs with Moment-Matching Gaussian Mixtures

FRANCESCA RANDONE, IMT School for Advanced Studies Lucca, Italy
LUCA BORTOLUSSI, University of Trieste, Italy
EMILIO INCERTO, IMT School for Advanced Studies Lucca, Italy
MIRCO TRIBASTONE, IMT School for Advanced Studies Lucca, Italy

Computing the posterior distribution of a probabilistic program is a hard task for which no one-fit-for-all solution exists. We propose Gaussian Semantics, which approximates the exact probabilistic semantics of a bounded program by means of Gaussian mixtures. It is parametrized by a map that associates each program location with the moment order to be matched in the approximation. We provide two main contributions. The first is a universal approximation theorem stating that, under mild conditions, Gaussian Semantics can approximate the exact semantics arbitrarily closely. The second is an approximation that matches up to second-order moments analytically in face of the generally difficult problem of matching moments of Gaussian mixtures with arbitrary moment order. We test our second-order Gaussian approximation (SOGA) on a number of case studies from the literature. We show that it can provide accurate estimates in models not supported by other approximation methods or when exact symbolic techniques fail because of complex expressions or non-simplified integrals. On two notable classes of problems, namely collaborative filtering and programs involving mixtures of continuous and discrete distributions, we show that SOGA significantly outperforms alternative techniques in terms of accuracy and computational time.

CCS Concepts: • **Theory of computation → Denotational semantics**; • **Mathematics of computing → Probabilistic reasoning algorithms**.

Additional Key Words and Phrases: probabilistic programming, inference, Gaussian mixtures

ACM Reference Format:
Francesca Randone, Luca Bortolussi, Emilio Incerto, and Mirco Tribastone. 2024. Inference of Probabilistic Programs with Moment-Matching Gaussian Mixtures. *Proc. ACM Program. Lang.* 8, POPL, Article 63 (January 2024), 31 pages. https://doi.org/10.1145/3632905

1 INTRODUCTION

Probabilistic programming languages are programming languages augmented with primitives expressing probabilistic behaviours [Gordon et al. 2014]. Examples are random assignments ("program variable x is distributed according to the probability distribution D"), probabilistic choices ("do P_1 with probability p else P_2) or conditioning ("variable x is distributed according to D, under the constraint that it can only take positive values"). This has enabled a variety of applications such as the analysis of randomized algorithms, machine learning and biology [Gordon et al. 2014].

Given a probabilistic program, there are different equivalent ways in which its semantics can be defined [Kozen 1983]. Following Kozen's Semantics 2 [Kozen 1979], in this paper we see a program as a transformer: given an initial joint distribution over the program variables, each instruction

Authors' addresses: Francesca Randone, IMT School for Advanced Studies Lucca, Italy, francesca.randone@imtlucca.it; Luca Bortolussi, University of Trieste, Italy, lbortolussi@units.it; Emilio Incerto, IMT School for Advanced Studies Lucca, Italy, emilio.incerto@imtlucca.it; Mirco Tribastone, IMT School for Advanced Studies Lucca, Italy, mirco.tribastone@imtlucca.it.

ACM 2475-1421/2024/1-ART63
https://doi.org/10.1145/3632905

Higher Order Bayesian Networks, Exactly

CLAUDIA FAGGIAN, IRIF, CNRS, Université Paris Cité, France
DANIELE PAUTASSO, University of Turin, Italy
GABRIELE VANONI, IRIF, CNRS, Université Paris Cité, France

Bayesian networks are graphical *first-order* probabilistic models that allow for a compact representation of large probability distributions, and for efficient inference, both exact and approximate. We introduce a *higher-order* programming language—in the idealized form of a λ-calculus—which we prove *sound and complete* w.r.t. Bayesian networks: each Bayesian network can be encoded as a term, and conversely each (possibly higher-order and recursive) program of ground type *compiles* into a Bayesian network.

The language allows for the specification of recursive probability models and hierarchical structures. Moreover, we provide a *compositional* and *cost-aware* semantics which is based on factors, the standard mathematical tool used in Bayesian inference. Our results rely on advanced techniques rooted into linear logic, intersection types, rewriting theory, and Girard's geometry of interaction, which are here combined in a novel way.

CCS Concepts: • **Theory of computation → Lambda calculus**; **Probabilistic computation**; **Linear logic**; **Type theory**; **Denotational semantics**.

Additional Key Words and Phrases: lambda calculus, intersection types, Bayesian networks, probabilistic programming, denotational semantics

ACM Reference Format:
Claudia Faggian, Daniele Pautasso, and Gabriele Vanoni. 2024. Higher Order Bayesian Networks, Exactly. *Proc. ACM Program. Lang.* 8, POPL, Article 84 (January 2024), 33 pages. https://doi.org/10.1145/3632926

1 INTRODUCTION

This paper is a foundational study, taking a cost-aware approach to the semantics of higher-order probabilistic programming languages. Probabilistic models play a crucial role in several fields such as machine learning, cognitive science, and applied statistics, with applications spanning from finance to biology. A prominent example of such models are Bayesian networks (BNs) [Pearl 1988], a (first-order, static) graphical formalism able to represent complex systems in a *compact* way and enabling *efficient* inference algorithms. BNs decompose large joint distributions into smaller *factors*. These are used in inference algorithms, both exact (such as message passing and variable elimination) and approximate (sampling-based). Despite their significant strengths, the task of modeling using Bayesian networks is comparable to the task of programming using logical circuits.

Probabilistic Programming Languages. A different approach is taken by *probabilistic programming languages* (PPLs), where statistical models are specified as programs. The fundamental idea behind PPLs is to separate the model description—the program—from the computation of the probability distribution specified by the program—the inference task. This separation aims at making stochastic modeling as accessible as possible, hiding the underlying inference engines, which typically encompass various sampling methods such as importance sampling, Markov Chain Monte Carlo, and

Authors' addresses: Claudia Faggian, IRIF, CNRS, Université Paris Cité, France, faggian@irif.fr; Daniele Pautasso, University of Turin, Italy, daniele.pautasso@unito.it; Gabriele Vanoni, IRIF, CNRS, Université Paris Cité, France, gabriele.vanoni@irif.fr.

ACM 2475-1421/2024/1-ART84
https://doi.org/10.1145/3632926

Strong Invariants Are Hard

On the Hardness of Strongest Polynomial Invariants for (Probabilistic) Programs

JULIAN MÜLLNER, TU Wien, Austria
MARCEL MOOSBRUGGER, TU Wien, Austria
LAURA KOVÁCS, TU Wien, Austria

We show that computing the strongest polynomial invariant for single-path loops with polynomial assignments is at least as hard as the Skolem problem, a famous problem whose decidability has been open for almost a century. While the strongest polynomial invariants are computable for *affine loops*, for polynomial loops the problem remained wide open. As an intermediate result of independent interest, we prove that reachability for discrete polynomial dynamical systems is Skolem-hard as well. Furthermore, we generalize the notion of invariant ideals and introduce *moment invariant ideals* for probabilistic programs. With this tool, we further show that the strongest polynomial moment invariant is (i) uncomputable, for probabilistic loops with branching statements, and (ii) Skolem-hard to compute for polynomial probabilistic loops without branching statements. Finally, we identify a class of probabilistic loops for which the strongest polynomial moment invariant is computable and provide an algorithm for it.

CCS Concepts: • **Theory of computation → Invariants**; **Probabilistic computation**; *Computability*; Random walks and Markov chains.

Additional Key Words and Phrases: Strongest algebraic invariant, Point-To-Point reachability, Skolem problem, Probabilistic programs

ACM Reference Format:
Julian Müllner, Marcel Moosbrugger, and Laura Kovács. 2024. Strong Invariants Are Hard: On the Hardness of Strongest Polynomial Invariants for (Probabilistic) Programs. *Proc. ACM Program. Lang.* 8, POPL, Article 30 (January 2024), 29 pages. https://doi.org/10.1145/3632872

1 INTRODUCTION

Loop invariants describe valid program properties that hold before and after every loop iteration. Intuitively, invariants provide correctness information that may prevent programmers from introducing errors while making changes to the loop. As such, invariants are fundamental to formalizing program semantics as well as to automate the formal analysis and verification of programs. While automatically synthesizing loop invariants is, in general, an uncomputable problem, when considering only single-path loops with linear updates (linear loops), the strongest polynomial invariant is in fact computable [Karr 1976; Kovács 2008; Müller-Olm and Seidl 2004a]. The computability

Authors' addresses: Julian Müllner, TU Wien, Vienna, Austria, julian.muellner@tuwien.ac.at; Marcel Moosbrugger, TU Wien, Vienna, Austria, marcel.moosbrugger@tuwien.ac.at; Laura Kovács, TU Wien, Vienna, Austria, laura.kovacs@tuwien.ac.at.

ACM 2475-1421/2024/1-ART30
https://doi.org/10.1145/3632872

10:50 - 12:10 **Probabilistic Programs at Turing Lecture** POPL
Chair(s): Alexandra Silva Cornell University

10:50 **20m** *Talk* ☆ **Probabilistic programming interfaces for random graphs: Markov categories, graphons, and nominal sets**
Nate Ackerman Harvard University, Cameron Freer Massachusetts Institute of Technology, Younesse Kaddar University of Oxford, Jacek Karwowski University of Oxford, Sean Moss University of Oxford, Daniel Roy University of Toronto, Sam Staton University of Oxford, Hongseok Yang KAIST; IBS

11:10 **20m** *Talk* ☆ **Inference of Probabilistic Programs with Moment-Matching Gaussian Mixtures**
Francesca Randone IMT School for Advanced Studies Lucca, Luca Bortolussi Department of Mathematics and Geosciences, University of Trieste, Emilio Incerto IMT School for Advanced Studies Lucca, Mirco Tribastone IMT Institute for Advanced Studies Lucca, Italy

11:30 **20m** *Talk* ☆ **Higher Order Bayesian Networks, Exactly**
Claudia Faggian Université de Paris & CNRS, Daniele Pautasso Università di Torino, Gabriele Vanoni IRIF, Université Paris Cité

11:50 **20m** *Talk* ☆ **Strong Invariants Are Hard: On the Hardness of Strongest Polynomial Invariants for (Probabilistic) Programs**
Julian Müllner TU Wien, Marcel Moosbrugger TU Wien, Laura Kovács TU Wien

NOW!