Fine-Grained Semantics for Probabilistic Programs

Benjamin Bichsel

Timon Gehr

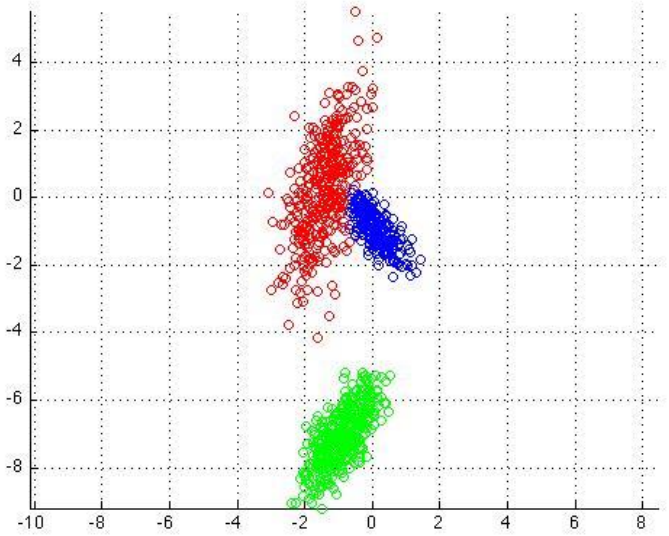
Martin Vechev

ETH zürich

SRL SOFTWARE RELIABILITY LAB

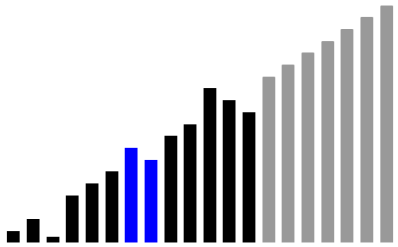
Probabilistic models

Networking



Machine Learning

Security



Randomized Algorithms

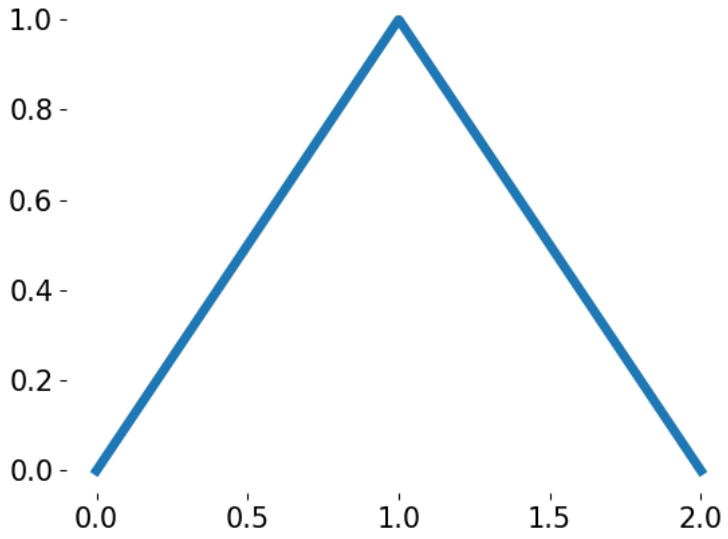
Robotics

Probabilistic programming

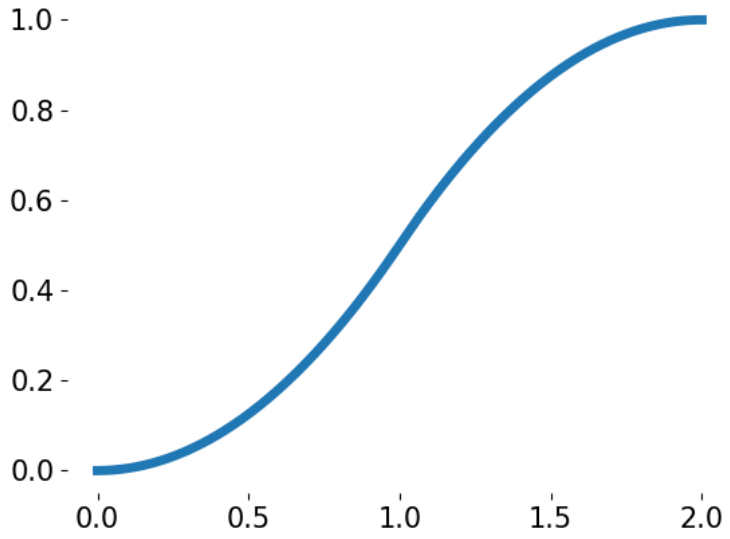
```
x := uniform(0,1);
y := uniform(0,1);

return x+y;
```

Probability density function

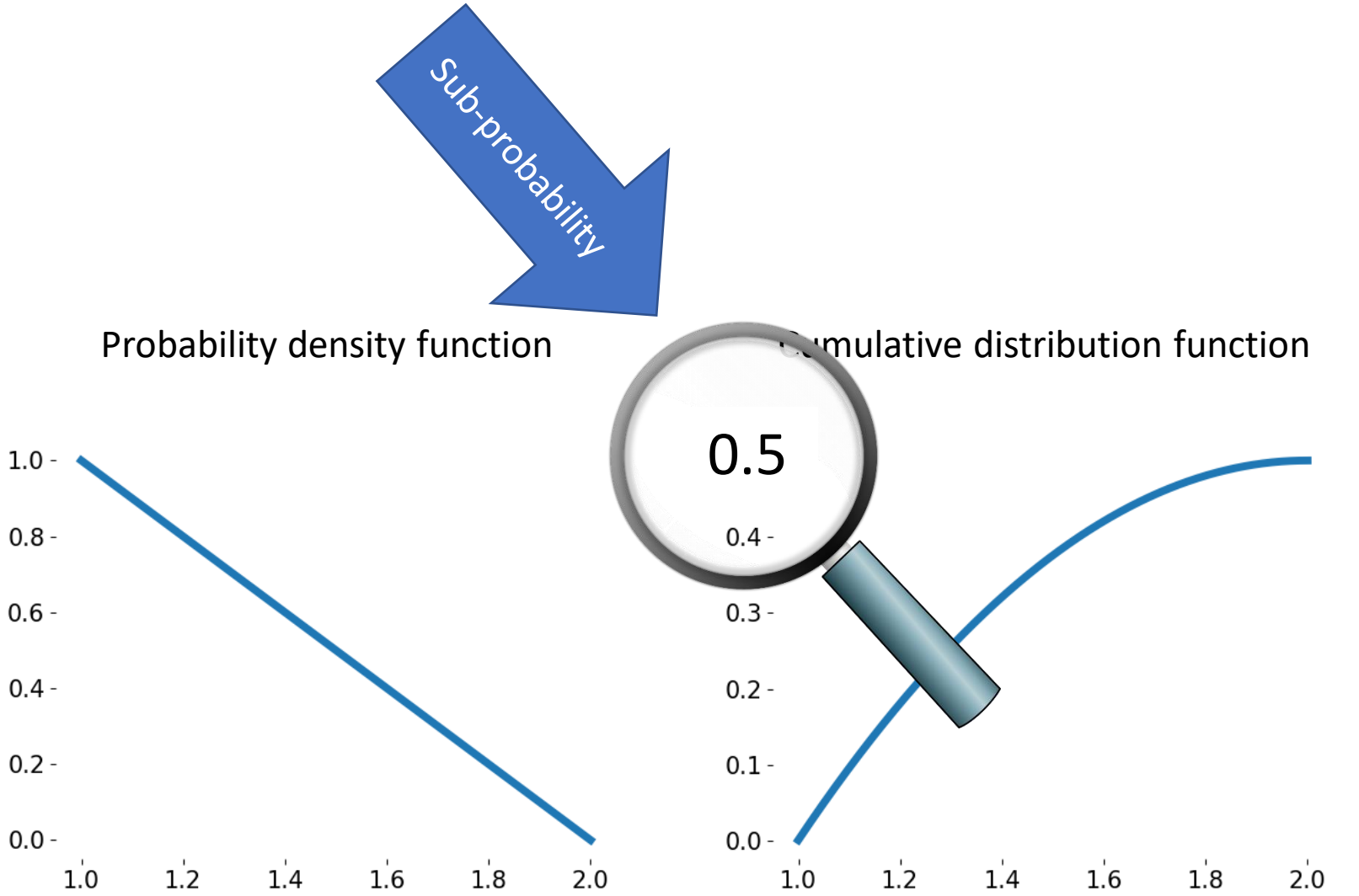


Cumulative distribution function



Observations

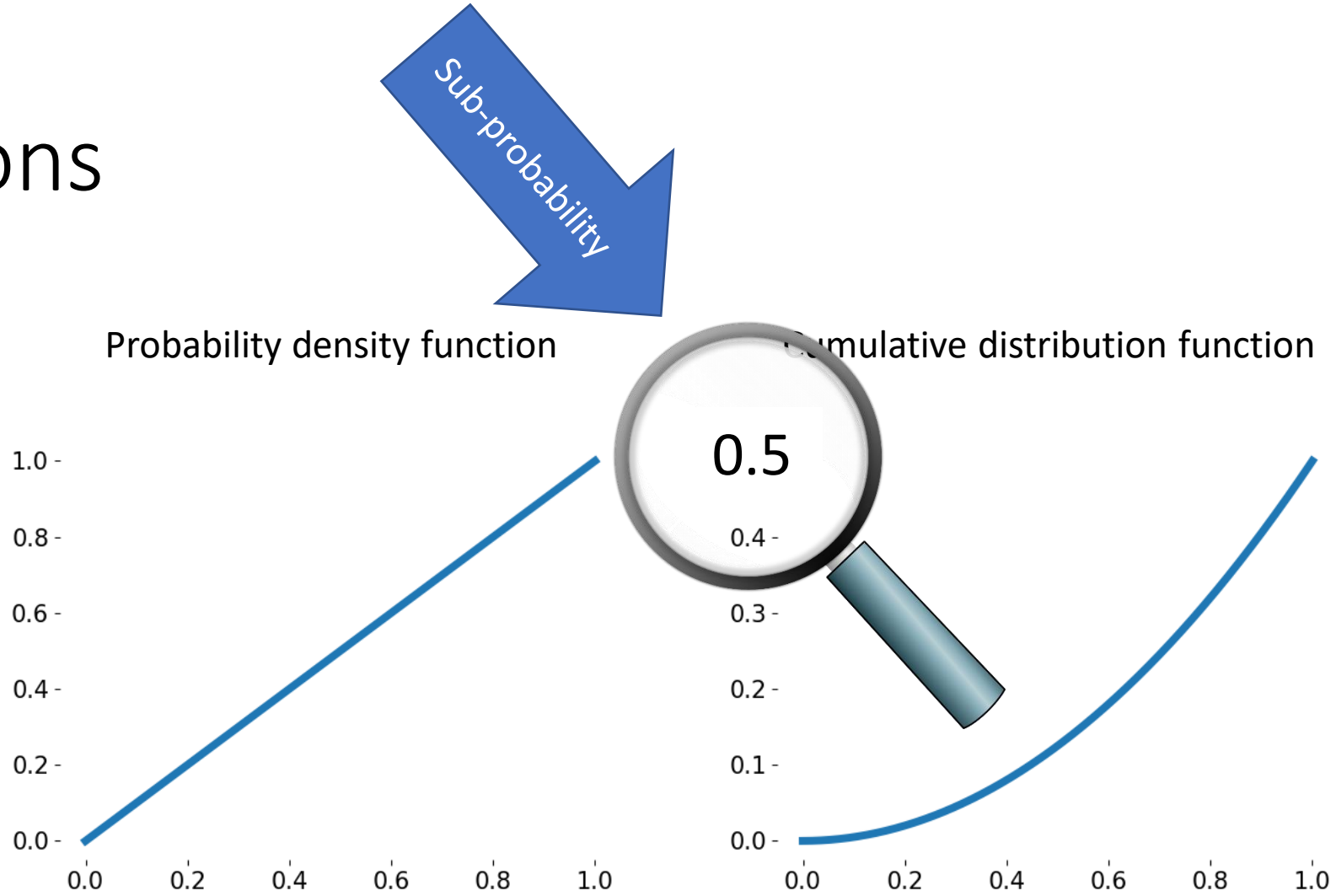
```
x := uniform(0,1);
y := uniform(0,1);
observe(x+y>=1);
return x+y;
```



$$\Pr[x + y \leq \infty] = 0.5$$

Partial functions

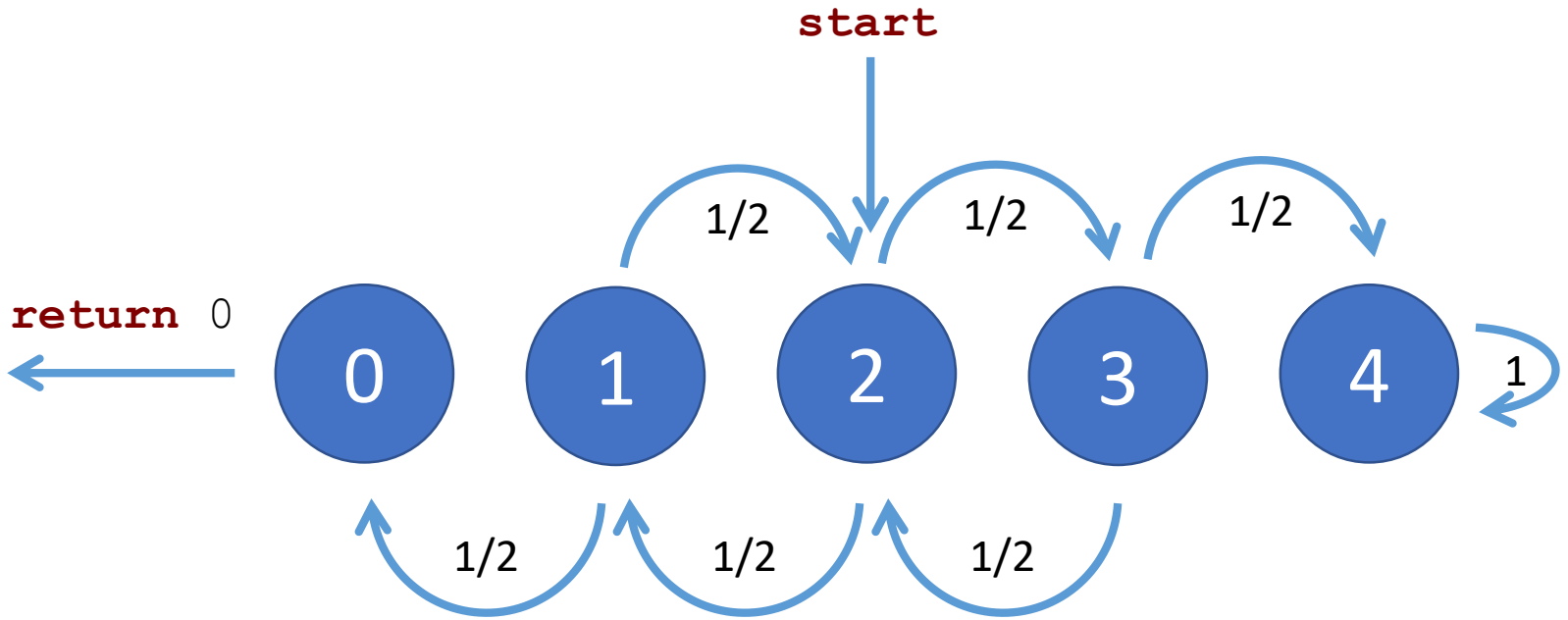
```
x := uniform(-1,1);
x = sqrt(x);
return x;
```

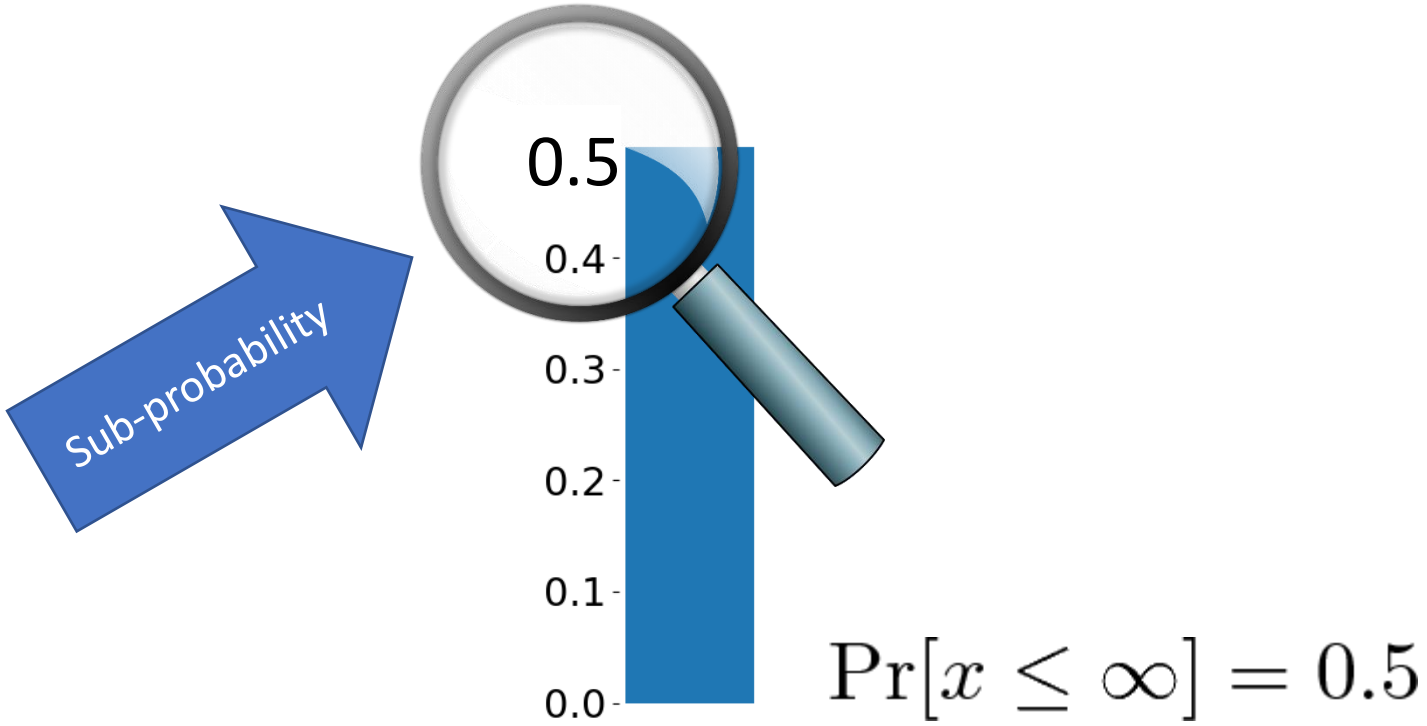


$$\Pr[x \leq \infty] = 0.5$$

Loops

```
x := 2;
while x>0 {
   if x<4 {
      if flip(0.5) {
         x++;
      } else {
         x--;
      }
   }
}
return x;
```

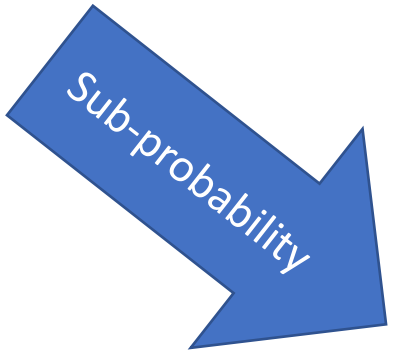




$$\Pr[x \leq \infty] = 0.5$$

Interaction of exceptions

```
x := 0;
while 1 {
   x = x/x;
}
return x;
```



$$\Pr[x \leq \infty] = 0$$

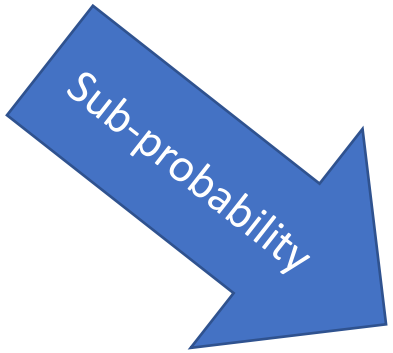
Better:

$$\Pr[\bot] = 1$$

$$\Pr[\circlearrowleft] = 0$$

Interaction of exceptions II

```
x := 0;
while x==0 {
   x = flip(0.5);
   observe(x==0);
}
return x;
```



$$\Pr[x \leq \infty] = 0$$

Better:

$$\Pr[\lightning] = 1$$

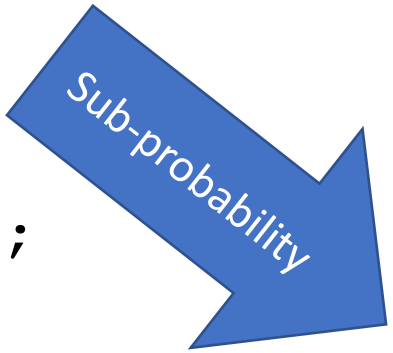
$$\Pr[\circlearrowleft] = 0$$

Interaction of exceptions III

```
x := 10;
d := gauss(0,1);
observe(d>=-0.5);

while x>=0 {
   x-=d;
}

return x;
```



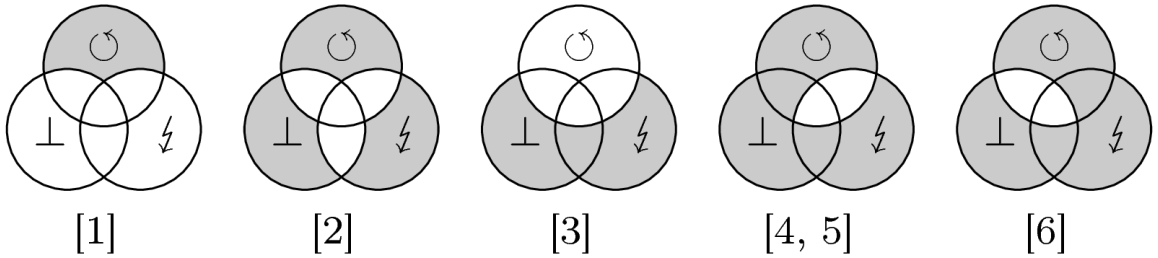
$$\Pr[x \leq \infty] = 0.50$$

Better:

$$\Pr[\lightning] = \Pr[\textbf{gauss}(0,1) < -0.5] \approx 0.31$$

$$\Pr[\circlearrowleft] = \Pr[-0.5 \leq \textbf{gauss}(0,1) \leq 0] \approx 0.19$$

Existing denotational semantics



[1] Dexter Kozen. Semantics of probabilistic programs. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, SFCS '79, pages 101–114, Washington, DC, USA, 1979. IEEE Computer Society.

[2] Dexter Kozen. A probabilistic pdl. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, pages 291–297, New York, NY, USA, 1983. ACM.

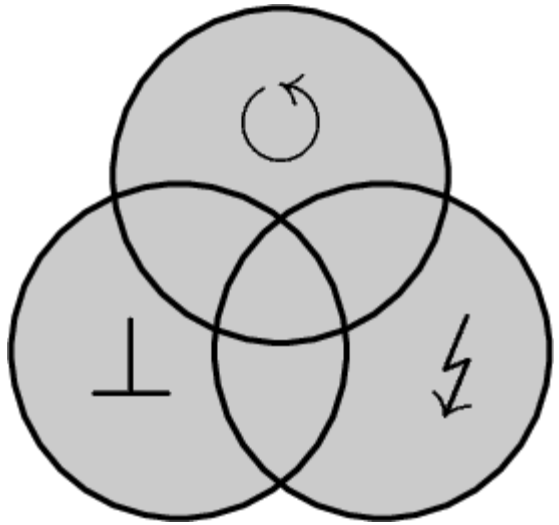
[3] Sam Staton, Hongseok Yang, Frank Wood, Chris Heunen, and Ohad Kammar. Semantics for probabilistic programming: Higher-order functions, continuous distributions, and soft constraints. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '16, pages 525–534, New York, NY, USA, 2016. ACM.

[4] Johannes Borgström, Ugo Dal Lago, Andrew D. Gordon, and Marcin Szymczak. A lambda-calculus foundation for universal probabilistic programming. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, ICFP 2016, pages 33–46, New York, NY, USA, 2016. ACM.

[5] Sam Staton. Commutative semantics for probabilistic programming. In *European Symposium on Programming*, pages 855–879. Springer, 2017.

[6] Federico Olmedo, Friedrich Gretz, Nils Jansen, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Annabelle McIver. Conditioning in probabilistic programming. *ACM Transactions on Programming Languages and Systems*, 2018 (to appear).

Our work – Denotational semantics



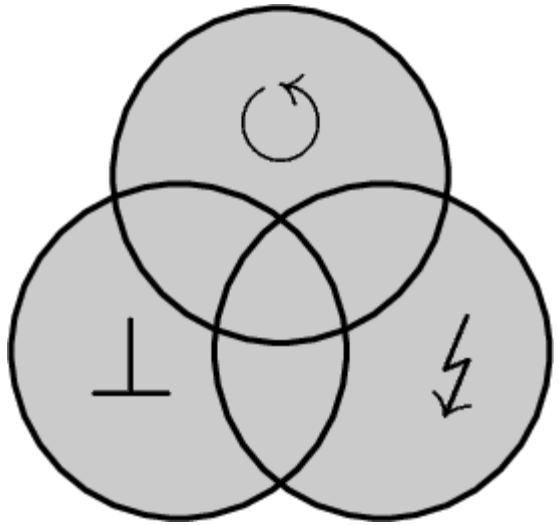
Explicitly distinguish exceptions:

- non-termination (↻)
- errors (⊥)
- observation failure (↯)

Our semantics also support

- Mixing continuous and discrete distributions
- Arrays
- The `score` primitive

Benefits

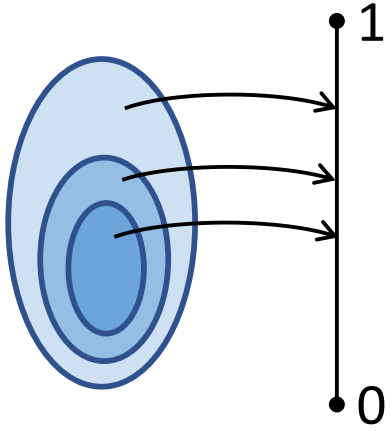


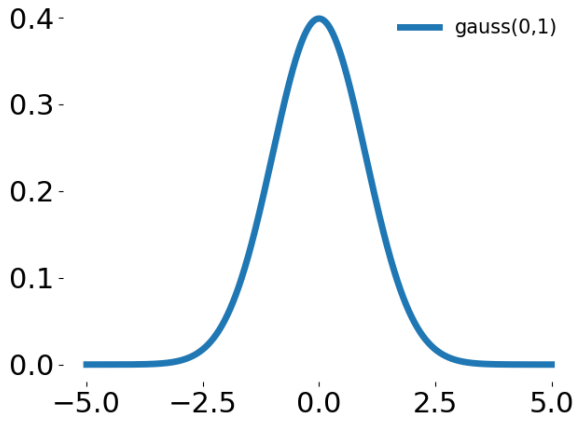
- Deeper **understanding** of probabilistic semantics
- Establish **correctness** of probabilistic solvers (e.g. PSI)
- More efficient **normalization** in probabilistic solvers
- Generalize to **arbitrary number** of exceptions

Measure theory

Probability measure

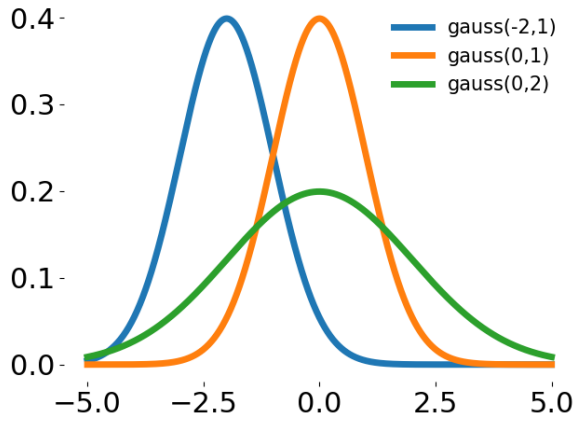
$$\mathbf{gauss}(0,1) : \Sigma_{\mathbb{R}} \to [0,1]$$





Probability kernel

$$\mathbf{gauss} : \mathbb{R}^2 \to \Sigma_{\mathbb{R}} \to [0,1]$$

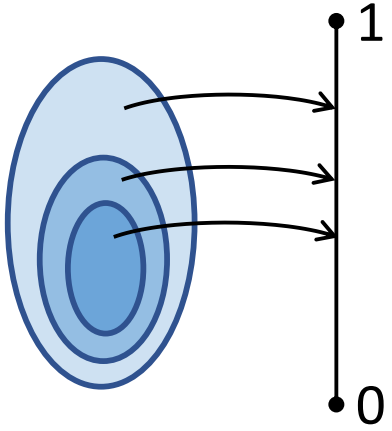


$$\mathbf{gauss}(0,1)(S) = \Pr[\mathbf{gauss}(0,1) \in S]$$

Measure theory

Probability measure

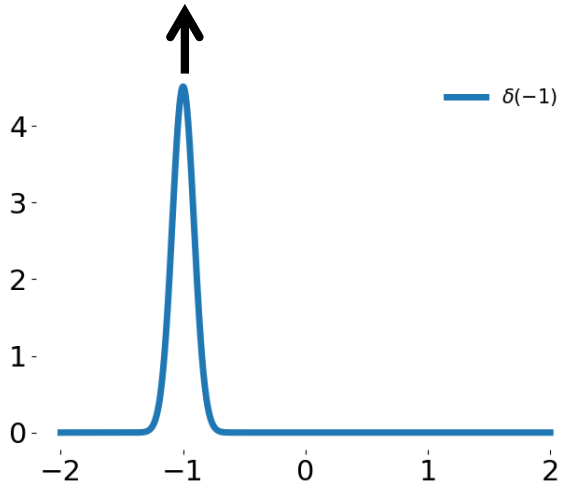
$$\delta(-1)\colon \Sigma_{\mathbb{R}} \to [0,1]$$

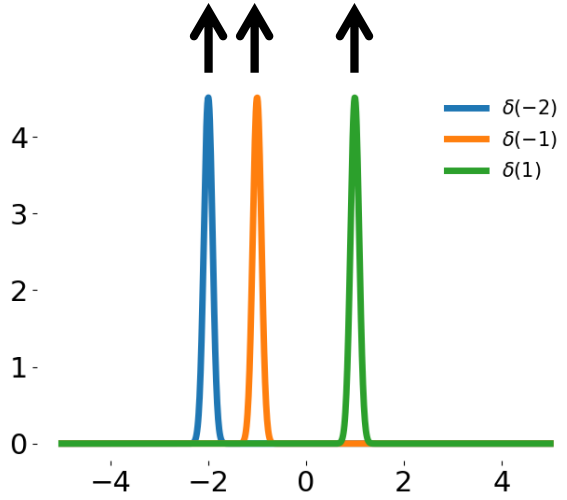


Probability kernel

$$\delta\colon \underbrace{\mathbb{R} \to \Sigma_{\mathbb{R}} \to [0,1]}_{\mathbb{R} \mapsto \mathbb{R}}$$

$$\delta(v)(S) = [v \in S]$$

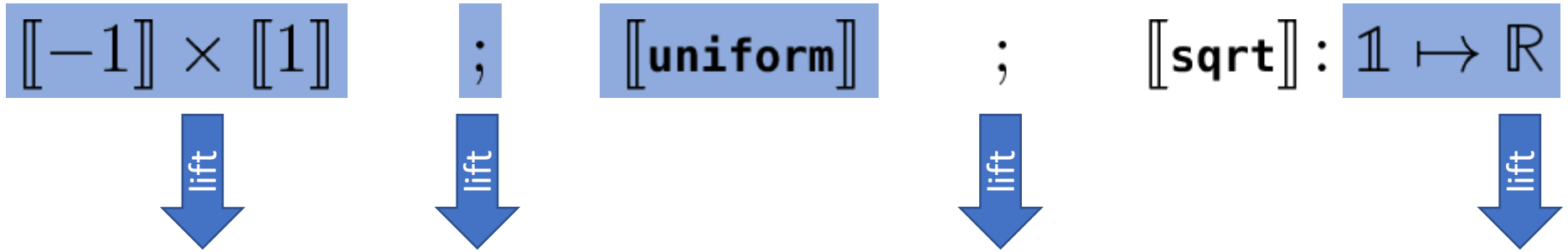




Denotational semantics

```
x := uniform(-1,1);
x = sqrt(x);
return x;
```

```
sqrt(uniform(-1,1))
```



$$\overline{\mathbb{R}} := \mathbb{R} \cup \{\bot, \circlearrowleft, \lightning\}$$

Denotational semantics - Constants

```
sqrt(uniform(-1,1))
```

Denotational semantics - Constants

```
sqrt(uniform(-1,1))
```

$$[\![-1]\!]: \quad \underbrace{[\![\Gamma]\!] \mapsto \overline{\mathbb{R}}}_{[\![\Gamma]\!] \to \Sigma_{\overline{\mathbb{R}}} \to [0,\infty)}$$

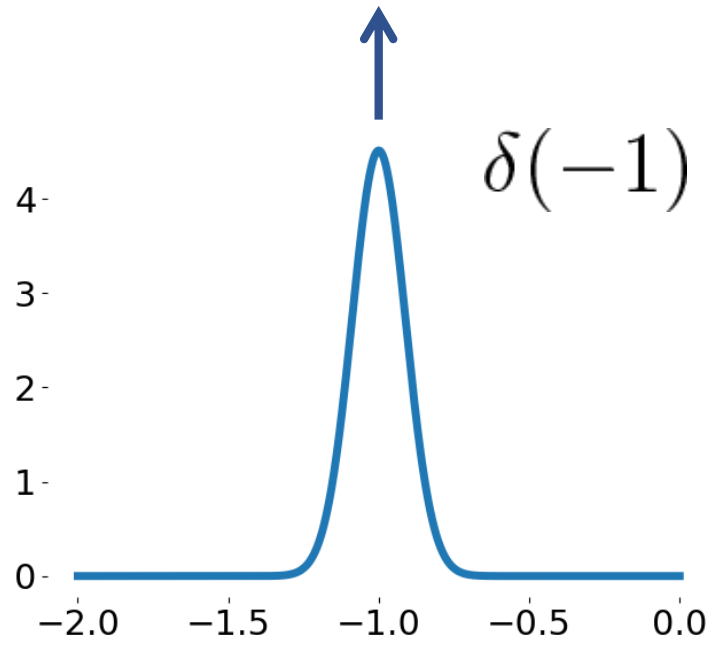
$$[\![-1]\!](\sigma)(S) = \delta(-1)(S) = [-1 \in S]$$

Denotational semantics - Constants

```
sqrt(uniform(-1,1))
```

$$[\![-1]\!]: \quad \underbrace{[\![\Gamma]\!] \mapsto \overline{\mathbb{R}}}_{[\![\Gamma]\!] \to \Sigma_{\overline{\mathbb{R}}} \to [0,\infty)}$$

$$[\![-1]\!](\sigma)(S) = \delta(-1)(S) = [-1 \in S]$$



Dirac delta

Denotational semantics - Constants

```
sqrt(uniform(-1,1))
```

$$\Big(\llbracket -1 \rrbracket \times \llbracket 1 \rrbracket\Big)(\sigma)(S)$$

$$= \int_{a \in \mathbb{R}} \int_{b \in \mathbb{R}} [(a,b) \in S] \; \llbracket 1 \rrbracket(\sigma)(db) \llbracket -1 \rrbracket(\sigma)(da)$$

$$\Big(\llbracket -1 \rrbracket \; \overline{\times} \; \llbracket 1 \rrbracket\Big)(\sigma)(S)$$

$$= \int_{a \in \overline{\mathbb{R}}} \int_{b \in \overline{\mathbb{R}}} [\overline{(a,b)} \in S] \quad \llbracket 1 \rrbracket(\sigma)(db) \llbracket -1 \rrbracket(\sigma)(da)$$

Lifted tuple

$$\overline{(12,13)} = (12,13)$$

$$\overline{(12,\bot)} = \bot$$

$$\overline{(\circlearrowleft, \lightning)} = \circlearrowleft$$

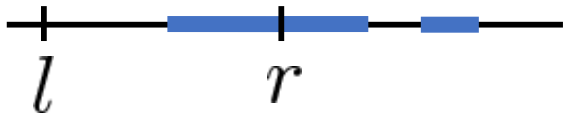
Denotational semantics - Functions

```
sqrt(uniform(-1,1))
```

$$[\![\texttt{uniform}]\!]: \underbrace{\mathbb{R} \times \mathbb{R} \mapsto \overline{\mathbb{R}}}_{\mathbb{R}\times\mathbb{R}\to\Sigma_{\overline{\mathbb{R}}}\to[0,\infty)}$$

Denotational semantics - Functions

```
sqrt(uniform(-1,1))
```



$$[\![\textbf{uniform}]\!] : \underbrace{\mathbb{R} \times \mathbb{R} \mapsto \overline{\mathbb{R}}}_{\mathbb{R}\times\mathbb{R}\to\Sigma_{\overline{\mathbb{R}}}\to[0,\infty)}$$

$$[\![\textbf{uniform}]\!](l,r)(S) = \begin{cases} \frac{1}{r-l}\lambda([l,r]\cap S) & l < r \\ \delta(\bot)(S) & \text{otherwise} \end{cases}$$

Denotational semantics - Functions

```
sqrt(uniform(-1,1))
```

$$[\![\texttt{uniform}]\!] : \underbrace{\mathbb{R} \times \mathbb{R} \mapsto \overline{\mathbb{R}}}_{\mathbb{R}\times\mathbb{R}\to\Sigma_{\overline{\mathbb{R}}}\to[0,\infty)}$$

$$[\![\texttt{uniform}]\!](l,r)(S) = \begin{cases} \frac{1}{r-l}\lambda([l,r]\cap S) & l < r \\ \delta(\bot)(S) & \text{otherwise} \end{cases}$$

Denotational semantics - Functions

```
sqrt(uniform(-1,1))
```

$$[\![\texttt{sqrt}]\!] : \underbrace{\mathbb{R} \mapsto \overline{\mathbb{R}}}_{\mathbb{R} \to \Sigma_{\overline{\mathbb{R}}} \to [0,\infty)}$$

$$[\![\texttt{sqrt}]\!](x)(S) = \begin{cases} \delta(\sqrt{x})(S) & x \geq 0 \\ \delta(\bot)(S) & x < 0 \end{cases}$$

Denotational semantics - Functions

```
sqrt(uniform(-1,1))
```

$$[\![\texttt{sqrt}]\!] : \underbrace{\mathbb{R} \mapsto \overline{\mathbb{R}}}_{\mathbb{R} \to \Sigma_{\overline{\mathbb{R}}} \to [0,\infty)}$$

$$[\![\texttt{sqrt}]\!](x)(S) = \begin{cases} \delta(\sqrt{x})(S) & x \geq 0 \\ \delta(\bot)(S) & x < 0 \end{cases}$$

Denotational semantics - Composition

```
sqrt(uniform(-1,1))
```

$$\Big([\![\texttt{uniform}]\!] \;\; ; \;\; [\![\texttt{sqrt}]\!] \Big)(l,r)(S)$$

$$\Big([\![\texttt{uniform}]\!] \;\; >\!\!=\!\!> \;\; [\![\texttt{sqrt}]\!] \Big)(l,r)(S)$$

$$= \int_{v \in \overline{\mathbb{R}}} \overline{[\![\texttt{sqrt}]\!]}(v)(S) \quad [\![\texttt{uniform}]\!](l,r)(dv)$$

Denotational semantics - Loops

```
n := 0;
while !flip(0.5) {
      n++;
}
```

$$[\![\textbf{while } !\textbf{flip}(0.5)\ \{\texttt{n++;}\}]\!]^{\text{trans}} : (\overline{[\![\Gamma]\!] \mapsto [\![\Gamma]\!]}) \to (\overline{[\![\Gamma]\!] \mapsto [\![\Gamma]\!]})$$

Kernel transformer

Semantics of n-1 iterations

Denotational semantics - Loops

```
n := 0;
while !flip(0.5) {
      n++;
}
```

$$[\![\textbf{while } !\textbf{flip}(0.5)\ \{\texttt{n++;}\}]\!]^{\text{trans}} : ([\![\Gamma]\!] \mapsto \overline{[\![\Gamma]\!]}) \to ([\![\Gamma]\!] \mapsto \overline{[\![\Gamma]\!]})$$

Kernel transformer

Semantics of n iterations

Denotational semantics - Loops

```
n := 0;
while !flip(0.5) {
      n++;
}
```

$$[\![\textbf{while } !\textbf{flip}(0.5)\ \{\mathsf{n{+}{+};}\}]\!]^{\mathsf{trans}} : ([\![\Gamma]\!] \mapsto \overline{[\![\Gamma]\!]}) \to ([\![\Gamma]\!] \mapsto \overline{[\![\Gamma]\!]})$$

$$[\![\textbf{while } !\textbf{flip}(0.5)\ \{\mathsf{n{+}{+};}\}]\!]^{\mathsf{trans}}(\kappa)$$

$$=_\delta \overline{\times}\ [\![!\textbf{flip}(0.5)]\!] >\!\!=\!\!> \lambda(\sigma, b).\begin{cases} \left([\![\mathsf{n{+}{+}}]\!] >\!\!=\!\!> \kappa\right)(\sigma) & b \neq 0 \\ \delta(\sigma) & b = 0 \end{cases}$$

Denotational semantics - Loops

```
n := 0;
while !flip(0.5) {
      n++;
}
```

$$[\![\textbf{while } !\textbf{flip}(0.5)\ \{\texttt{n++;}\}]\!]^{\text{trans}} : ([\![\Gamma]\!] \mapsto \overline{[\![\Gamma]\!]}) \to ([\![\Gamma]\!] \mapsto \overline{[\![\Gamma]\!]})$$

Non-termination

$$[\![\textbf{while} \ldots]\!] = \lim_{n\to\infty} \left([\![\textbf{while} \ldots]\!]^{\text{trans}} \right)^n \left(\lambda\sigma.\delta(\circlearrowleft) \right)$$

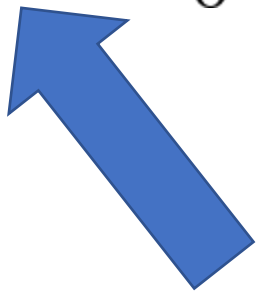
Probability kernel

Theorem: The semantics of each expression and each statement is a **probability kernel**.

$$\Pr[x \in \mathbb{R} \vee x \in \{\bot, \circlearrowleft, \lightning\}] = 1$$

Properties of Semantics - Commutativity

$$\frac{1}{0} + \mathsf{F}() \not\approx \mathsf{F}() + \frac{1}{0}$$



```
F() {
  while 1 {
    skip;
  }
  return 0;
}
```

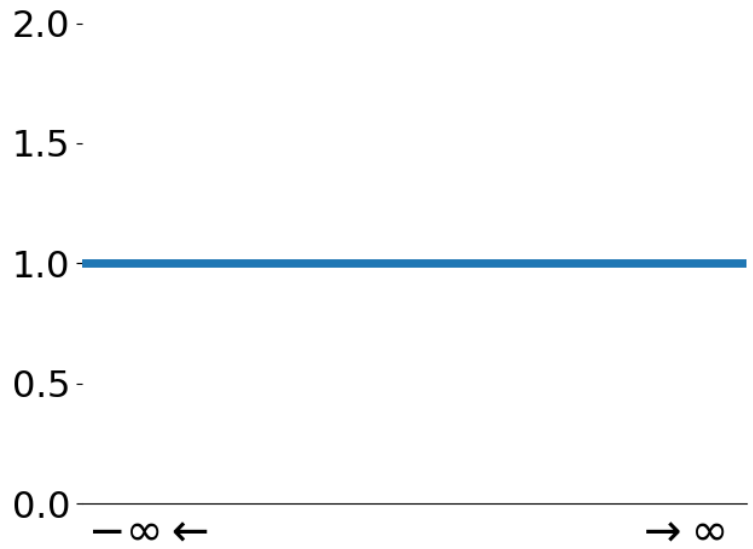
Properties of Semantics - Associativity

$$e_1 + (e_2 + e_3) \simeq (e_1 + e_2) + e_3$$
$$P_1; (P_2; P_3) \simeq (P_1; P_2); P_3$$

Proof: Relies on associativity of the product of kernels and composition of kernels.

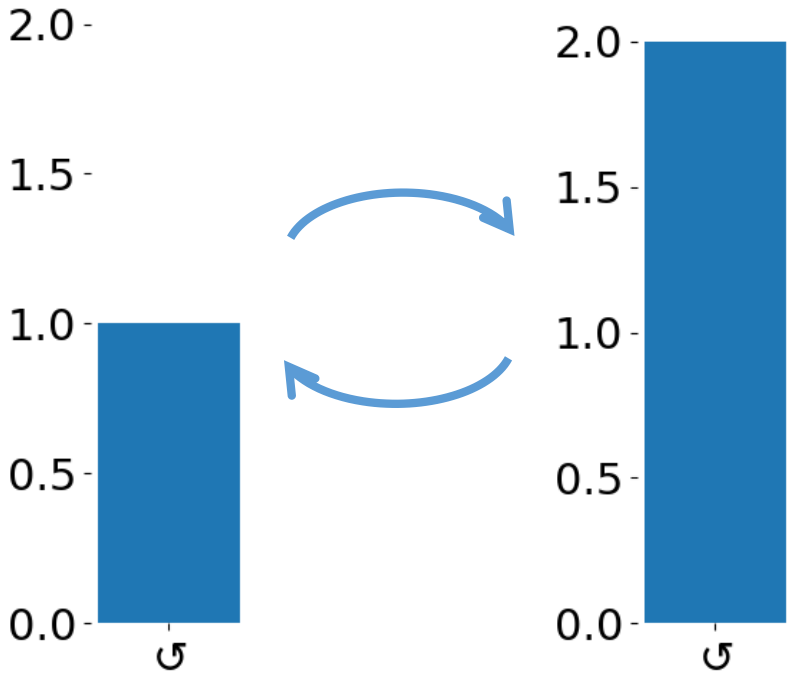
Score primitive

```
x := gauss(0,1);
score(√(2π)e^(x^2/2));
return x;
```



Score primitive vs non-termination

```
i := 0;
while 1 {
  if i==0 {
    score(2);
  } else {
    score(0.5);
  }
  i = 1-i;
}
```



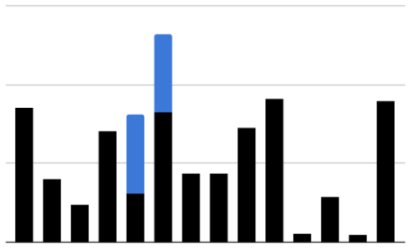
Score primitive – s-finite kernels

Theorem: After adding the `score` primitive and abolishing non-termination, the semantics of each expression and each statement is an **s-finite kernel**.

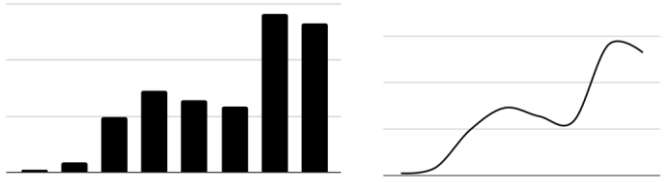
Informally:

$$0 \leq \Pr[x \in \mathbb{R} \vee x \in \{\bot, \cancel{\circlearrowleft, \lightning}\}] \leq \infty$$

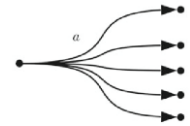
Features of probabilistic programming languages



score



mix discrete and continuous

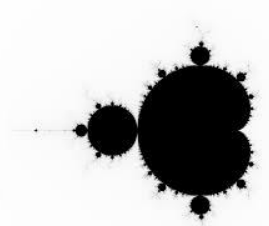


non-determinism

observations

errors

loops



recursion

higher-order