The Landscape: *Type* Theory

Simple Types

\[ \tau ::= \iota \mid \tau \to \tau \]
The Landscape: *Type Theory*

**Simple Types**

\[ \tau ::= \iota \mid \tau \to \tau \]

- Sound for termination, in absence of recursion.
- Poor expressive power.
- Intuitionistic Logic.
The Landscape: Type Theory

Simple Types
\[ \tau ::= \iota | \tau \to \tau \]

Polymorphic Types
\[ \tau ::= \cdots | \alpha | \forall \alpha. \tau \]

Sound for termination, in absence of recursion.

Poor expressive power.

Intuitionistic Logic.

Second-order Intuitionistic Logic.

Very expressive, extensionally.

Still poor, intensionally.

Motivated by Semantics.

Complete for termination.

Type inference is undecidable.

Reasonably expressive, intensionally.

Type inference remains decidable.
The Landscape: Type Theory

Simple Types

- Second-order Intuitionistic Logic.
- Very expressive, extensionally.
- Still poor, intensionally.

Polymorphic Types

\[ \tau ::= \cdots \mid \alpha \mid \forall \alpha. \tau \]
The Landscape: Type Theory

Simple Types
\[ \tau ::= \iota \mid \tau \to \tau \]

Polymorphic Types
\[ \tau ::= \cdots \mid \alpha \mid \forall \alpha. \tau \]

Intersection Types
\[ \tau ::= \cdots \mid \tau \land \tau \]
The Landscape: *Type Theory*

**Simple Types**
- Motivated by Semantics.
- Complete for termination.
- Type inference is undecidable.

**Polymorphic Types**
\[ \tau ::= \cdots \mid \alpha \mid \forall \alpha. \tau \]

**Intersection Types**
\[ \tau ::= \cdots \mid \tau \land \tau \]

▶ Sound for termination, in absence of recursion.
▶ Poor expressive power.
▶ Intuitionistic Logic.
▶ Second-order Intuitionistic Logic.
▶ Very expressive, extensionally.
▶ Still poor, intensionally.

▶ Motivated by Semantics.
▶ Complete for termination.
▶ Type inference is undecidable.

▶ Reasonably expressive, intensionally.
▶ Type inference remains decidable.
The Landscape: *Type Theory*

### Simple Types

\[ \tau ::= \iota \mid \tau \rightarrow \tau \]

### Polymorphic Types

\[ \tau ::= \cdots \mid \alpha \mid \forall \alpha. \tau \]

### Intersection Types

\[ \tau ::= \cdots \mid \tau \land \tau \]

### Sized Types

\[ \tau ::= \cdots \mid \iota[\xi] \]
The Landscape: Type Theory

Simple Types

- Reasonably expressive, intensionally.
- Type inference remains decidable

Polymorphic Types

\[ \tau ::= \cdots \mid \alpha \mid \forall \alpha. \tau \]

Intersection Types

\[ \tau ::= \cdots \mid \tau \land \tau \]

Sized Types

\[ \tau ::= \cdots \mid \nu[\xi] \]
The Landscape: *Recursion Theory*

**Determinism**

\[ M \bar{s} \rightarrow^* N_s \]
The Landscape: *Recursion Theory*

<table>
<thead>
<tr>
<th>Determinism</th>
<th>Probabilism</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M\overline{s} \rightarrow^* N_s$</td>
<td>$\left[ M\overline{s} \right] = \mathcal{D}_s$</td>
</tr>
</tbody>
</table>
The Landscape: *Recursion Theory*

$$\sum D_s \text{ can be smaller than 1.}$$

<table>
<thead>
<tr>
<th>Determinism</th>
<th>Probabilism</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M\overline{s} \rightarrow^* N_s$</td>
<td>$[M\overline{s}] = D_s$</td>
</tr>
</tbody>
</table>
The Landscape: *Recursion Theory*

<table>
<thead>
<tr>
<th></th>
<th>Determinism</th>
<th>Probabilism</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Termination</strong></td>
<td>$\exists N_s \in NF$</td>
<td>$[M \bar{s}] = \mathcal{D}_s$</td>
</tr>
<tr>
<td><strong>Expression</strong></td>
<td>$M \bar{s} \rightarrow^* N_s$</td>
<td></td>
</tr>
</tbody>
</table>
The Landscape: *Recursion Theory*

Undecidable; $\Sigma^0_1$-complete.

$M \bar{s} \rightarrow^* N_s$

Termination: $\exists N_s \in NF$

Probabilism: $[M \bar{s}] = D_s$
<table>
<thead>
<tr>
<th>Determinism</th>
<th>Probabilism</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M\bar{s} \rightarrow^* N_s$</td>
<td>$[M\bar{s}] = D_s$</td>
</tr>
<tr>
<td>Termination</td>
<td>$\exists N_s \in NF$</td>
</tr>
<tr>
<td></td>
<td>$\sum D_s = 1$</td>
</tr>
</tbody>
</table>
The Landscape: *Recursion* Theory

### Termination

- **Uniform Termination**
  \[ \forall s. \exists N_s \in NF \sum D_s = 1 \]

- **Undecidable; \Sigma^0_1**-complete.

- **Almost-Sure Termination**
  \[ \Pi_2^0 \text{-complete.} \]

\[ M \bar{s} \rightarrow^* N_s \quad \llbracket M \bar{s} \rrbracket = D_s \]
**The Landscape: *Recursion Theory***

<table>
<thead>
<tr>
<th>Determinism</th>
<th>Probabilism</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M\overline{s} \rightarrow^* N_s$</td>
<td>$[M\overline{s}] = \mathcal{D}_s$</td>
</tr>
<tr>
<td><strong>Termination</strong></td>
<td><strong>Uniform Termination</strong></td>
</tr>
<tr>
<td>$\exists N_s \in NF$</td>
<td>$\sum \mathcal{D}_s = 1$</td>
</tr>
<tr>
<td><strong>Uniform Termination</strong></td>
<td></td>
</tr>
<tr>
<td>$\forall s. \exists N_s \in NF$</td>
<td></td>
</tr>
</tbody>
</table>
The Landscape: *Recursion Theory*

- Determinism: $\Pi^0_2$-complete.
  - Termination: $\exists N_s \in NF$
  - Uniform Termination: $\forall s. \exists N_s \in NF$

- Probabilism: $\lceil M\bar{s} \rceil = D_s$
  - Almost-Sure Termination: $\Pi^0_2$-complete.
  - Undecidable; $\Sigma^0_1$-complete.
  - Uniform Termination: $\sum D_s = 1$
<table>
<thead>
<tr>
<th>Determinism</th>
<th>Probabilism</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M \bar{s} \rightarrow^* N_s$</td>
<td>$\lfloor M \bar{s} \rfloor = \mathcal{D}_s$</td>
</tr>
<tr>
<td><strong>Termination</strong></td>
<td><strong>Uniform Termination</strong></td>
</tr>
<tr>
<td>$\exists N_s \in NF$</td>
<td>$\forall s. \exists N_s \in NF$</td>
</tr>
<tr>
<td>$\sum \mathcal{D}_s = 1$</td>
<td>$\forall s. \sum \mathcal{D}_s = 1$</td>
</tr>
</tbody>
</table>
The Landscape: *Recursion Theory*

### Determinism

- **Termination**
  \[ M \bar{s} \rightarrow^* N \]
- **Uniform Termination**
  \[ \forall s. \exists N_s \in NF \]

### Probabilism

- **Almost-Sure Termination**
  \[ \sum D_s = 1 \]
- **Uniform Termination**
  \[ \forall s. \sum D_s = 1 \]

\[ \Pi_2^0 \text{-complete.} \]
Section 1

Sized Types
Deterministic Sized Types

- Pure λ-calculus with simple types is terminating.
  - This can be proved in many ways, including by \textit{reducibility}.
  - But useless as a programming language.
Deterministic Sized Types

- Pure λ-calculus with simple types is terminating.
  - This can be proved in many ways, including by reducibility.
  - But useless as a programming language.

- For every type \( \tau \), define a set of reducible terms \( Red_\tau \).
- Prove that all reducible terms are normalizing...
- ...and that all typable terms are reducible.
Deterministic Sized Types

- Pure $\lambda$-calculus with simple types is terminating.
  - This can be proved in many ways, including by \textit{reducibility}.
  - But useless as a programming language.
- What if we endow it with \textbf{full recursion} as a \texttt{fix} binder?
Deterministic Sized Types

- Pure $\lambda$-calculus with simple types is terminating.
  - This can be proved in many ways, including by reducibility.
  - But useless as a programming language.
- What if we endow it with **full recursion** as a $\text{fix}$ binder?

\[
(fix \ x. M)V \rightarrow M\{fix \ x. M/x\}V
\]
Deterministic Sized Types

- Pure $\lambda$-calculus with simple types is terminating.
  - This can be proved in many ways, including by reducibility.
  - But useless as a programming language.
- What if we endow it with **full recursion** as a fix binder?
- All the termination properties are lost, for very good reasons.

$\text{fix } f \lambda x : \iota f(x - 1)$

$\text{fix } f \lambda x : \iota f(x - 2)$

$\text{fix } f \lambda x : \iota f(x - 3)$

$\text{M}$

$\text{fix } f \lambda x : \iota f(x - 1)$

$\text{fix } f \lambda x : \iota f(x - 2)$

$\text{fix } f \lambda x : \iota f(x - 3)$

$\text{M}$

**BAD!**

**GOOD!**

For every type $\tau$, define a set of reducible terms $\text{Red}_\tau$.

Prove that all reducible terms are normalizing...

...and that all typable terms are reducible.

$(\text{fix } x . M)_V \rightarrow M\{\text{fix } x . M/x\}_V$
Deterministic Sized Types

- Pure $\lambda$-calculus with simple types is terminating.
  - This can be proved in many ways, including by reducibility.
  - But useless as a programming language.
- What if we endow it with full recursion as a fix binder?
- All the termination properties are lost, for very good reasons.
- Is everything lost?
Deterministic Sized Types

- Pure $\lambda$-calculus with simple types is terminating.
  - This can be proved in many ways, including by reducibility.
  - But useless as a programming language.
- What if we endow it with full recursion as a fix binder?
- All the termination properties are lost, for very good reasons.
- Is everything lost?
- NO!
Deterministic Sized Types

- Pure $\lambda$-calculus with simple types is terminating.
  - This can be proved in many ways, including by reducibility.
  - But useless as a programming language.
- What if we endow it with full recursion as a fix binder?
- All the termination properties are lost, for very good reasons.
- Is everything lost?
- NO!

For every type $\tau$, define a set of reducible terms $\text{Red}_\tau$.
Prove that all reducible terms are normalizing...
...and that all typable terms are reducible.

$$(\text{fix } x. M) \mapsto M\{\text{fix } x. M/x\}$$
Deterministic Sized Types

- Pure λ-calculus with simple types is terminating.
  - This can be proved in many ways, including by reducibility.
  - But useless as a programming language.
- What if we endow it with full recursion as a fix binder?
- All the termination properties are lost, for very good reasons.
- Is everything lost?
- NO!

\[
\text{fix } f \quad \text{BAD!} \\
\lambda x : \tau \quad \text{GOOD!}
\]

\[
\text{fix } f \quad \lambda x : \tau \\
f(x - 1) \quad f(x) \quad f(x + 1) \\
M
\]

\[
\text{fix } f \quad \lambda x : \tau \\
f(x - 1) \quad f(x - 2) \quad f(x - 3) \\
M
\]

For every type \( \tau \), define a set of reducible terms \( \text{Red}_{\tau} \).

Prove that all reducible terms are normalizing...

...and that all typable terms are reducible.

\[
(fix \ x. M) \ V \rightarrow M \{fix \ x. M/x\} \ V
\]
Deterministic Sized Types, Technically

- **Types.**

\[
\xi ::= a \mid \omega \mid \xi + 1; \quad \tau ::= \nu[\xi] \mid \tau \to \tau.
\]
Deterministic Sized Types, Technically

- Types.

\[ \xi ::= a \mid \omega \mid \xi + 1; \quad \tau ::= \nu[\xi] \mid \tau \rightarrow \tau. \]
Deterministic Sized Types, Technically

- **Types.**
  \[ \xi ::= a \mid \omega \mid \xi + 1; \quad \tau ::= \nu[\xi] \mid \tau \rightarrow \tau. \]

- **Typing Fixpoints.**
  \[
  \Gamma, x : \nu[a] \rightarrow \tau \vdash M : \nu[a + 1] \rightarrow \tau
  \]
  \[
  \Gamma \vdash \text{fix } x.M : \nu[\xi] \rightarrow \tau
  \]
Deterministic Sized Types, Technically

- **Types.**
  \[
  \xi ::= a \mid \omega \mid \xi + 1; \quad \tau ::= \iota[\xi] \mid \tau \to \tau. 
  \]

- **Typing Fixpoints.**
  \[
  \Gamma, x : \iota[a] \to \tau \vdash M : \iota[a + 1] \to \tau \\
  \frac{}{\Gamma \vdash \text{fix } x.M : \iota[\xi] \to \tau}
  \]

- **Quite Powerful.**
  - Can type many forms of structural recursion.
Deterministic Sized Types, Technically

- **Types.**
  \[ \begin{align*}
  \xi &::= a \mid \omega \mid \xi + 1; \\
  \tau &::= \nu[\xi] \mid \tau \to \tau.
  \end{align*} \]

- **Typing Fixpoints.**
  \[
  \Gamma, x : \nu[a] \to \tau \vdash M : \nu[a + 1] \to \tau
  \]
  \[
  \frac{}
  \quad \Gamma \vdash \text{fix } x. M : \nu[\xi] \to \tau
  \]

- **Quite Powerful.**
  - Can type many forms of structural recursion.

- **Termination.**
  - Proved by **Reducibility**.
  - ... but of an indexed form.
Deterministic Sized Types, Technically

- **Types.**
  \[ \xi ::= a \mid \omega \mid \xi + 1; \quad \tau ::= \nu[\xi] \mid \tau \rightarrow \tau. \]

- **Typing Fixpoints.**
  \[ \Gamma, x : \nu[a] \rightarrow \tau \vdash M : \nu[a + 1] \rightarrow \tau \]

- Reducibility sets are of the form \( \text{Red}^\theta_\tau \).
- \( \theta \) is an environment for index variables.
- Proof of reducibility for \( \text{fix} \, x. \, M \) is rather delicate.
- Can type many forms of structural recursion.

- **Termination.**
  - Proved by **Reducibility**.
  - …but of an indexed form.
Deterministic Sized Types, Technically

- **Types.**

\[
\xi ::= a \mid \omega \mid \xi + 1; \quad \tau ::= \iota[\xi] \mid \tau \rightarrow \tau.
\]

- **Typing Fixpoints.**

\[
\begin{align*}
\Gamma, x : \iota[a] \rightarrow \tau &\vdash M : \iota[a + 1] \rightarrow \tau \\
\Gamma \vdash \text{fix } x. M : \iota[\xi] \rightarrow \tau
\end{align*}
\]

- **Quite Powerful.**
  - Can type many forms of structural recursion.

- **Termination.**
  - Proved by Reducibility.
  - ...but of an indexed form.

- **Type Inference.**
  - It is indeed *decidable*.
  - But *nontrivial*.
Probabilistic Termination

- **Examples:**

\[
\begin{align*}
\text{fix } f & . \lambda x . \text{if } x > 0 \text{ then if } \text{FairCoin} \text{ then } f(x - 1) \text{ else } f(x + 1); \\
\text{fix } f & . \lambda x . \text{if } x > 0 \text{ then if } \text{BiasedCoin} \text{ then } f(x - 1) \text{ else } f(x + 1); \\
\text{fix } f & . \lambda x . \text{if } \text{BiasedCoin} \text{ then } f(x + 1) \text{ else } x.
\end{align*}
\]
Examples:

```plaintext
fix f.λx. if x > 0 then if FairCoin then f(x - 1) else f(x + 1);
fix f.λx. if x > 0 then if BiasedCoin then f(x - 1) else f(x + 1);
fix f.λx. if BiasedCoin then f(x + 1) else x.
```

Unbiased Random Walk

Biased Random Walk, the "wrong" way.
Probabilistic Termination

- Examples:

\[
\text{fix } f. \lambda x. \begin{aligned}
&\text{if } x > 0 \text{ then if } \text{FairCoin} \text{ then } f(x - 1) \text{ else } f(x + 1); \\
&\text{fix } f. \lambda x. \begin{aligned}
&\text{if } x > 0 \text{ then if } \text{BiasedCoin} \text{ then } f(x - 1) \text{ else } f(x + 1); \\
&\text{fix } f. \lambda x. \text{if } \text{BiasedCoin} \text{ then } f(x + 1) \text{ else } x.
\end{aligned}
\]

Unbiased Random Walk, with two upward calls.
Biased Random Walk, the "wrong" way.

- Probabilistic termination is thus:

Sensitive to the actual distribution from which we sample.
Sensitive to how many recursive calls we perform.
Probabilistic Termination

- **Examples:**

  ```
  fix f.λx.if x > 0 then if FairCoin then f(x - 1) else f(x + 1);
  fix f.λx.if x > 0 then if BiasedCoin then f(x - 1) else f(x + 1);
  fix f.λx.if BiasedCoin then f(x + 1) else x.
  ```

- **Non-Examples:**

  ```
  fix f.λx.if FairCoin then f(x - 1) else (f(x + 1); f(x + 1));
  fix f.λx.if BiasedCoin then f(x + 1) else f(x - 1);
  ```
Probabilistic Termination

- **Examples:**

  \[
  \text{fix } f \cdot \lambda x. \text{if } x > 0 \text{ then if } \text{FairCoin} \text{ then } f(x - 1) \text{ else } f(x + 1) \text{;}
  \]

  \[
  \text{fix } f \cdot \lambda x. \text{if } x > 0 \text{ then if } \text{BiasedCoin} \text{ then } f(x - 1) \text{ else } f(x + 1) \text{;}
  \]

  \[
  \text{fix } f \cdot \lambda x. \text{if } \text{BiasedCoin} \text{ then } f(x + 1) \text{ else } x. \]

- **Non-Examples:**

  \[
  \text{fix } f \cdot \lambda x. \text{if } \text{FairCoin} \text{ then } f(x - 1) \text{ else } (f(x + 1); f(x + 1)) \text{;}
  \]

  \[
  \text{fix } f \cdot \lambda x. \text{if } \text{BiasedCoin} \text{ then } f(x + 1) \text{ else } f(x - 1) \text{;}
  \]

  Unbiased Random Walk, with **two** upward calls.
Probabilistic Termination

- **Examples:**

  fix $f.\lambda x.\text{if } x > 0 \text{ then if } \text{FairCoin } \text{ then } f(x - 1) \text{ else } f(x + 1);$
  fix $f.\lambda x.\text{if } x > 0 \text{ then if } \text{BiasedCoin } \text{ then } f(x - 1) \text{ else } f(x + 1);$
  fix $f.\lambda x.\text{if } \text{BiasedCoin } \text{ then } f(x + 1) \text{ else } x.$

- **Non-Examples:**

  fix $f.\lambda x.\text{if } \text{FairCoin } \text{ then } f(x - 1) \text{ else } (f(x + 1); f(x + 1));$
  fix $f.\lambda x.\text{if } \text{BiasedCoin } \text{ then } f(x + 1) \text{ else } f(x - 1);$

Unbiased Random Walk, with two upward calls.

Biased Random Walk, the “wrong” way.
Probabilistic Termination

▶ Examples:

\[
\text{fix } f.\lambda x.\text{if } x > 0 \text{ then if } \text{FairCoin} \text{ then } f(x - 1) \text{ else } f(x + 1); \\
\text{fix } f.\lambda x.\text{if } x > 0 \text{ then if } \text{BiasedCoin} \text{ then } f(x - 1) \text{ else } f(x + 1); \\
\text{fix } f.\lambda x.\text{if } \text{BiasedCoin} \text{ then } f(x + 1) \text{ else } x.
\]

▶ Non-Examples:

\[
\text{fix } f.\lambda x.\text{if } \text{FairCoin} \text{ then } f(x - 1) \text{ else } (f(x + 1); f(x + 1)); \\
\text{fix } f.\lambda x.\text{if } \text{BiasedCoin} \text{ then } f(x + 1) \text{ else } f(x - 1);
\]

▶ Probabilistic termination is thus:

▶ Sensitive to \textit{the actual distribution} from which we sample.
▶ Sensitive to \textit{how many recursive calls} we perform.
One-Counter Blind Markov Chains

- They are automata of the form \((Q, \delta)\) where
  - \(Q\) is a finite set of states.
  - \(\delta : Q \rightarrow \text{Dist}(Q \times \{-1, 0, 1\})\).

- They are a very special form of One-Counter Markov Decision Processeses [BBEK2011].
  - Everything is purely deterministic.
  - The counter value is ignored.
They are automata of the form \((Q, \delta)\) where

- \(Q\) is a finite set of states.
- \(\delta : Q \rightarrow \text{Dist}(Q \times \{-1, 0, 1\})\).

They are a very special form of One-Counter Markov Decision Processeses [BBEK2011].

- Everything is purely deterministic.
- The counter value is ignored.

The probability of reaching a configuration where the counter is 0 can be approximated arbitrarily well \textit{in polynomial time}. 

Probabilistic Sized Types [DLGrellois2017]

- **Basic Idea**: craft a sized-type system in such a way as to mimic the recursive structure by an OCBMC.
Probabilistic Sized Types [DLGrellois2017]

- **Basic Idea:** craft a sized-type system in such a way as to mimic the recursive structure by a OCBMC.
- **Judgments.**

\[ \Gamma | \Delta \vdash M : \mu \]
Probabilistic Sized Types [DLGrellois2017]

- **Basic Idea:** Craft a sized-type system in such a way as to mimic the recursive structure by a OCBMC.

- **Judgments.**

  \[ \Gamma \vdash M : \mu \]

  Every higher-order variable occurs at most once.
Basic Idea: craft a sized-type system in such a way as to mimick the recursive structure by a OCBMC.

Judgments.

\[ \Gamma | \Delta \vdash M : \mu \]

Typing Fixpoints.

\[
\Gamma | x : \sigma \vdash V : \iota[a + 1] \rightarrow \tau \quad OCBMC(\sigma) \text{ terminates.} \\
\Gamma | \Theta \vdash \text{fix } x.V : \iota[\xi] \rightarrow \tau
\]
Probabilistic Sized Types [DLGrellois2017]

- **Basic Idea**: craft a sized-type system in such a way as to mimic the recursive structure by an OCBMC.

- **Judgments**.

- **Typing Fixpoints**.

  \[ \Gamma \mid x : \sigma \vdash V : \iota[a + 1] \rightarrow \tau \quad \text{OCBMC}(\sigma) \text{ terminates.} \]

  \[ \Gamma \mid \Theta \vdash \text{fix } x.V : \iota[\xi] \rightarrow \tau \]

  This is sufficient for typing:

  - Unbiased random walks;
  - Biased random walks.

- **Typing Probabilistic Choice**:

  \[ \Gamma \mid \Delta \vdash M : \tau \]

  \[ \Gamma \mid \Omega \vdash N : \rho \]

  \[ \Gamma \mid 1/2 \Delta + 1/2 \Omega \vdash M \oplus N : \iota \rightarrow \rho \]

- **Termination**.

  By a quantitative nontrivial refinement of reducibility. Every higher-order variable occurs at most once.

- A distribution type, i.e., a finite distribution of types.

  - This is sufficient for typing: Unbiased random walks; Biased random walks.
Probabilistic Sized Types [DLGrellois2017]

- **Basic Idea:** craft a sized-type system in such a way as to mimic the recursive structure by a OCBMC.

- **Judgments.**

  $\Gamma \mid \Delta \vdash M : \mu$

- **Typing Fixpoints.**

  \[
  \Gamma \mid x : \sigma \vdash V : \iota[a + 1] \rightarrow \tau \quad \text{OCBMC(\sigma) terminates.} \\
  \hline
  \Gamma \mid \Theta \vdash \text{fix } x.V : \iota[\xi] \rightarrow \tau
  \]

- **Typing Probabilistic Choice**

  \[
  \Gamma \mid \Delta \vdash M : \tau \quad \Gamma \mid \Omega \vdash N : \rho \\
  \hline
  \Gamma \mid \frac{1}{2}\Delta + \frac{1}{2}\Omega \vdash M \oplus N : \frac{1}{2}\tau + \frac{1}{2}\rho
  \]
Probabilistic Sized Types [DLGrellois2017]

- **Basic Idea**: craft a sized-type system in such a way as to mimick the recursive structure by a OCBMC.

- **Judgments**.

\[ \Gamma | \Delta \vdash M : \mu \]

- **Typing Fixpoints**.

\[
\frac{\Gamma | x : \sigma \vdash V : \nu[a+1] \to \tau \quad OCBMC(\sigma) \text{ terminates.} }{\Gamma | \Theta \vdash \text{fix } x.V : \nu[\xi] \to \tau}
\]

- **Typing Probabilistic Choice**

\[
\frac{\Gamma | \Delta \vdash M : \tau \quad \Gamma | \Omega \vdash N : \rho} {\Gamma | \frac{1}{2}\Delta + \frac{1}{2}\Omega \vdash M \oplus N : \frac{1}{2}\tau + \frac{1}{2}\rho}
\]

- **Termination**.
  - By a quantitative nontrivial refinement of reducibility.
  
  - A distribution type, i.e., a finite distribution of types.
  
  - By a quantitative nontrivial refinement of reducibility.
  
  - This is sufficient for typing:
    - Unbiased random walks;
    - Biased random walks.

  - Form \( \sigma \), one can build a OCBMC:
    - \( \sigma \) keeps track of the probability of each recursive call.

  - Reducibility sets are now on the form \( \text{Red} \theta,p \tau \)
    - \( p \) stands for the probability of being reducible.

  - Reducibility sets are continuous:
    \( \text{Red} \theta,p \tau = \bigcup q<p \text{Red} \theta,q \tau \)
Probabilistic Sized Types [DLGrellois2017]

- **Basic Idea:** craft a sized-type system in such a way as to mimic the recursive structure by a OCBMC.
- **Judgments.**

\[ \Gamma \mid \Delta \vdash M : \mu \]

- Reducibility sets are now on the form \( \text{Red}_{\tau}^{\theta,p} \)
- \( p \) stands for the *probability* of being reducible.
- Reducibility sets are continuous:

\[
\text{Red}_{\tau}^{\theta,p} = \bigcup_{q<p} \text{Red}_{\tau}^{\theta,q}
\]

\[ \Gamma \mid \frac{1}{2} \Delta + \frac{1}{2} \Omega \vdash M \oplus N : \frac{1}{2} \tau + \frac{1}{2} \rho \]

- **Termination.**
  - By a quantitative nontrivial refinement of reducibility.
Section 2

Intersection Types
Deterministic Intersection Types

- **Question:** what are simple types *missing* as a way to precisely capture termination?
Deterministic Intersection Types

- **Question**: what are simple types *missing* as a way to precisely capture *termination*?
- Very simple examples of normalizing terms which *cannot* be typed:

\[
\Delta = \lambda x.xx \quad \Delta(\lambda x.x).
\]
Deterministic Intersection Types

- **Question**: what are simple types *missing* as a way to precisely capture *termination*?
- Very simple examples of normalizing terms which *cannot* be typed:
  \[
  \Delta = \lambda x.x \quad \Delta(\lambda x.x).
  \]

- **Types**
  \[
  \tau ::= \star \mid A \rightarrow B \quad A ::= \{\tau_1, \ldots, \tau_n\}
  \]
Deterministic Intersection Types

- **Question**: what are simple types *missing* as a way to precisely capture *termination*?

- Very simple examples of normalizing terms which *cannot* be typed:

\[ \Delta = \lambda x.xx \quad \Delta(\lambda x.x). \]

- **Types**

\[ \tau ::= \star \mid A \rightarrow B \quad A ::= \{\tau_1, \ldots, \tau_n\} \]

- **Typing Rules: Examples**

\[
\begin{align*}
\{\Gamma \vdash M : \tau_i\}_{1 \leq i \leq n} & \quad \Gamma \vdash M : \{\tau_1, \ldots, \tau_n\} \\
\Gamma \vdash M : \{A \rightarrow B\} & \quad \Gamma \vdash N : A \\
\Gamma \vdash MN : B
\end{align*}
\]

- Completeness

By subject expansion, the dual of subject reduction.
Deterministic Intersection Types

▶ **Question:** what are simple types *missing* as a way to precisely capture *termination*?

▶ Very simple examples of normalizing terms which *cannot* be typed:

\[ \Delta = \lambda x.xx \quad \Delta(\lambda x.x) . \]

▶ **Types**

\[ \tau ::= * \mid A \to B \quad A ::= \{\tau_1, \ldots, \tau_n\} \]

▶ **Typing Rules: Examples**

\[
\frac{\{\Gamma \vdash M : \tau_i\}_{1 \leq i \leq n}}{\Gamma \vdash M : \{\tau_1, \ldots, \tau_n\}} \quad \frac{\Gamma \vdash M : \{A \to B\} \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}
\]

▶ **Termination**

▶ Again by reducibility.
Deterministic Intersection Types

- **Question**: what are simple types *missing* as a way to precisely capture *termination*?
- Very simple examples of normalizing terms which *cannot* be typed:
  \[
  \Delta = \lambda x.x \, x \quad \Delta(\lambda x.x).
  \]

- **Types**
  \[
  \tau ::= \star \mid A \to B \quad A ::= \{\tau_1, \ldots, \tau_n\}
  \]

- **Typing Rules: Examples**
  \[
  \frac{\Gamma \vdash M : \tau_i \, 1 \leq i \leq n}{\Gamma \vdash M : \{\tau_1, \ldots, \tau_n\}} \quad \frac{\Gamma \vdash M : \{A \to B\} \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}
  \]

- **Termination**
  - Again by reducibility.

- **Completeness**
  - By *subject expansion*, the dual of subject reduction.
Probabilistic choice can be seen as a form of read operation:

\[ M \oplus N = \text{if } \text{BitInput} \text{ then } M \text{ else } N \]
Oracle Intersection Types [BreuvartDL2018]

- Probabilistic choice can be seen as a form of read operation:

\[ M \oplus N = \text{if BitInput then } M \text{ else } N \]

- Types

\[ \tau ::= \star \mid A \rightarrow s \cdot B \quad A ::= \{ \tau_1, \ldots, \tau_n \} \quad s \in \{0, 1\}^* \]
Oracle Intersection Types [BreuvartDL2018]

- Probabilistic choice can be seen as a form of read operation:

\[ M \oplus N = \text{if } \text{BitInput} \text{ then } M \text{ else } N \]

- Types

\[ \tau ::= \star \mid A \to s \cdot B \quad A ::= \{\tau_1, \ldots, \tau_n\} \quad s \in \{0, 1\}^* \]

- Typing Rules: Examples

\[
\frac{\Gamma \vdash M : s \cdot A}{\Gamma \vdash M \oplus N : 0s \cdot A} \quad \frac{\frac{\Gamma \vdash M : r \cdot \{A \to s \cdot B\}}{\Gamma \vdash N : q \cdot A}}{\Gamma \vdash MN : (rqs) \cdot B}
\]

- Termination and Completeness

Formulated in a rather unusual way. Proved as usual, but relative to a single probabilistic branch

\[ \mathbb{P}(M \downarrow) = \sum \vdash M : \star \]

This is unavoidable, due to recursion theory.
Oracle Intersection Types [BreuvartDL2018]

- Probabilistic choice can be seen as a form of read operation:
  \[ M \oplus N = \text{if } \text{BitInput} \text{ then } M \text{ else } N \]

- Types
  \[ \tau ::= \star \mid A \rightarrow s \cdot B \quad A ::= \{\tau_1, \ldots, \tau_n\} \quad s \in \{0, 1\}^* \]

- Typing Rules: Examples
  \[
  \begin{align*}
  \Gamma \vdash M : s \cdot A & \quad \Gamma \vdash M \oplus N : 0s \cdot A \\
  \Gamma \vdash M : r \cdot \{A \rightarrow s \cdot B\} & \quad \Gamma \vdash N : q \cdot A \\
  \Gamma \vdash MN : (rqs) \cdot B 
  \end{align*}
  \]

- Termination and Completeness
  - Formulated in a rather unusual way.
  - Proved as usual, but relative to a single probabilistic branch.
Oracle Intersection Types [BreuvartDL2018]

- Probabilistic choice can be seen as a form of read operation:

\[ M \oplus N = \text{if } \text{BitInput} \text{ then } M \text{ else } N \]

- Types

\[ \tau ::= \star \mid A \rightarrow s \cdot B \quad A ::= \{\tau_1, \ldots, \tau_n\} \quad s \in \{0, 1\}^* \]

- Typing Rules: Examples

\[
\begin{align*}
\Gamma \vdash M : s \cdot A \\
\Gamma \vdash M \oplus N : 0s \cdot A \\
\Gamma \vdash M : r \cdot \{A \rightarrow s \cdot B\} \\
\Gamma \vdash N : q \cdot A \\
\Gamma \vdash MN : (rqs) \cdot B
\end{align*}
\]

- Termination and Completeness

- Formulated in a rather unusual way.
- Proved as usual, but relative to a single probabilistic branch.
Oracle Intersection Types [BreuvartDL2018]

- Probabilistic choice can be seen as a form of read operation:

\[ M \oplus N = \text{if BitInput then } M \text{ else } N \]

- Types

\[
\begin{align*}
\tau &::= \star \mid A \rightarrow s \cdot B \\
A &::= \{\tau_1, \ldots, \tau_n\} \\
&\quad s \in \{0, 1\}^*
\end{align*}
\]

\[
\mathbb{P}(M \downarrow) = \sum_{\Gamma \vdash M : s \cdot \star} 2^{|s|}
\]

- Typing Rules:

\[
\begin{align*}
\Gamma \vdash M : s \cdot A \\
\Gamma \vdash M : r \cdot \{A \rightarrow s \cdot B\} \\
\Gamma \vdash N : q \cdot A
\end{align*}
\]

This is **unavoidable**, due to recursion theory.

- Termination and Completeness

  - Formulated in a rather *unusual* way.
  - Proved as usual, but relative to a single probabilistic branch
Intersection Types and Computations

They are a combination of oracle and sized types.

Intersections are needed for preciseness.

Distributions of types allow to analyse more than one probabilistic branch in the same type derivation.
They are a combination of oracle and sized types. Intersections are needed for preciseness. Distributions of types allow to analyse more than one probabilistic branch in the same type derivation.
Intersection Types and Computations

They are a combination of oracle and sized types.

Intersections are needed for preciseness.

Distributions of types allow to analyse more than one probabilistic branch in the same type derivation.
Intersection Types and Computations

Oracle Intersection Types

They are a combination of oracle and sized types.

Intersections are needed for preciseness.

Distributions of types allow to analyse more than one probabilistic branch in the same type derivation.
Intersection Types and Computations

They are a combination of oracle and sized types. Intersections are needed for preciseness. Distributions of types allow to analyse more than one probabilistic branch in the same type derivation.
Monadic Intersection Types [BDL2018]

- They are a combination of oracle and sized types.
- Intersections are needed for preciseness.
- Distributions of types allow to analyse more than one probabilistic branch in the same type derivation.
Ongoing and Future Work

- **Non-Idempotent Intersection Types**
  - Monadic and Oracle Intersection Types are idempotent.
Non-Idempotent Intersection Types

- Monadic and Oracle Intersection Types are idempotent.
- Conjecture:

\[
\text{IDEMP} : AST = \text{NONIDEMP} : PAST
\]
Ongoing and Future Work

- **Non-Idempotent Intersection Types**
  - Monadic and Oracle Intersection Types are idempotent.
  - Conjecture:
    \[ \text{IDEMP} : AST = \text{NONIDEMP} : PAST \]

- **Linear Dependent Types**
  - Intersection Types are complete, but only for computations.
  - In linear dependent types [DLG2011], one is (relatively complete) for deterministic first-order functions.
Non-Idempotent Intersection Types

- Monadic and Oracle Intersection Types are idempotent.
- Conjecture:

\[
\text{IDEMP} : AST = \text{NONIDEMP} : PAST
\]

Linear Dependent Types

- Intersection Types are complete, but only for computations.
- In linear dependent types [DLG2011], one is (relatively complete) for deterministic first-order functions.
- How about probabilism?
  - Monadic types becomes indexed:

\[
\mu ::= \{ \sigma[i] : p[i] \}_{i \in I}
\]

- Subtyping is coupling-based.
Thank You!

Questions?