Quantitative Inhabitation for Different Lambda Calculi in a Unifying Framework

Victor Arrial\textsuperscript{1} \quad Giulio Guerrieri\textsuperscript{2,3} \quad Delia Kesner\textsuperscript{1,4}

\textsuperscript{1}Université Paris Cité, Paris \quad \textsuperscript{2}Aix Marseille Univ, Marseille
\textsuperscript{3}Edinburgh Research Centre, Huawei, Edinburgh
\textsuperscript{4}Institut Universitaire de France

Marseille - I2M, May 4, 2023
Quantitative Inhabitation for Different Lambda Calculi in a Unifying Framework

Victor Arrial\textsuperscript{1} Giulio Guerrieri\textsuperscript{2,3} Delia Kesner\textsuperscript{1,4}

\textsuperscript{1}Université Paris Cité, Paris \hspace{1cm} \textsuperscript{2}Aix Marseille Univ, Marseille
\hspace{1cm}
\textsuperscript{3}Edinburgh Research Centre, Huawei, Edinburgh
\hspace{1cm}
\textsuperscript{4}Institut Universitaire de France

Marseille - I2M, May 4, 2023
What is Inhabitation?
What is Inhabitation?

Typing Problem:
\[ t \]

Inhabitation Problem (IP):
\[ \Gamma \vdash t : \sigma \]
What is Inhabitation?

Typing Problem:

\[ \Gamma \vdash t : \sigma \]
What is Inhabitation?

Typing Problem:
\[ \Gamma \vdash t : \sigma \]

Computational: [Mil’78]
Typers
What is Inhabitation?

Typing Problem:
\[ \Gamma \vdash t : \sigma \]

Inhabitation Problem (IP):

Computational: [Mil’78] Typer
What is Inhabitation?

Typing Problem: \[ \Gamma \vdash t : \sigma \]

Inhabitation Problem (IP): \[ \Gamma \vdash \sigma \]

Computational: [Mil’78]
Typers
What is Inhabitation?

Typing Problem:
\[ \Gamma \vdash t : \sigma \]

Inhabitation Problem (IP):
\[ \Gamma \vdash t : \sigma \]

Computational: [Mil’78]
Typers
What is Inhabitation?

Typing Problem:
\[ \Gamma \vdash t : \sigma \]

Inhabitation Problem (IP):
\[ \Gamma \vdash t : \sigma \]

Computational: [Mil’78]
Typers

Program Synthesis

Logical: [HoMi’94]
Proof Search and Logic Programming

Computational: [HuOr’20]
What is Inhabitation?

Typing Problem:
\[ \Gamma \vdash t : \sigma \]

Inhabitation Problem (IP):
\[ \Gamma \vdash t : \sigma \]

Computational: [Mil’78]
Typers

Computational: [HuOr’20]
Program Synthesis

Logical: [HoMi’94]
Proof Search and Logic Programming
What is Inhabitation?

Typing Problem: \[ \Gamma \vdash t : \sigma \]

Computational: [Mil’78]
Typers

Inhabitation Problem (IP): \[ \Gamma \vdash t : \sigma \]

Computational: [HuOr’20]
Program Synthesis

Logical: [HoMi’94]
Proof Search and Logic Programming
Quantitative Inhabitation for Different Lambda Calculi in a Unifying Framework
Quantitative Inhabitation for Different Lambda Calculi in a Unifying Framework
Unifying Frameworks
Different Models of Computation:

**Call-by-Name**

**NAME**

**Call-by-Value**

**VALUE**

Unifying Frameworks

- Call-by-Push-Value [Levy'99]
- Distant Bang Calculus [BKRV'20]
Quantitative Inhabitation for Different Lambda Calculi in a Unifying Framework
Quantitative Inhabitation for Different Lambda Calculi in a Unifying Framework
Different Models of Computation:

- Call-by-Name
- Call-by-Value

Unifying Frameworks:

- Call-by-Push-Value [Levy’99]
Unifying Frameworks

Different Models of Computation:

- Call-by-Name
- Call-by-Value

Unifying Frameworks:

- Call-by-Push-Value [Levy’99]
- Bang Calculus [EG’16]
Quantitative Inhabitation for Different Lambda Calculi in a Unifying Framework

Different Models of Computation:

- Call-by-Name
- Call-by-Value

Unifying Frameworks:

- Call-by-Push-Value [Levy’99]
- Bang Calculus [EG’16]:

\[ t, u ::= x | \lambda x.t | tu \]
Quantitative Inhabitation for Different Lambda Calculi in a Unifying Framework

Unifying Frameworks

Different Models of Computation:

- Call-by-Name
- Call-by-Value

Unifying Frameworks:

- Call-by-Push-Value [Levy’99]
- Bang Calculus [EG’16]:

\[
t, u ::= x \mid \lambda x.t \mid tu \mid !t \quad \text{Values}
\]
Different Models of Computation:

- Call-by-Name
  - \textbf{NAME}
- Call-by-Value
  - \textbf{VALUE}

Unifying Frameworks:
- Call-by-Push-Value [Levy’99]
- Bang Calculus [EG’16]:

\[
t, u ::= x \mid \lambda x.t \mid tu \\
| !t \quad \text{Values} \\
| \text{der}(t) \quad \text{Computations}
\]
Unifying Frameworks

Different Models of Computation:

- **Call-by-Name**

- **Call-by-Value**

Unifying Frameworks:

- Call-by-Push-Value [Levy’99]

- Distant Bang Calculus [EG’16] [BKRV’20]:

\[
t, u ::= x | \lambda x.t | tu \\
| !t \quad \text{Values} \\
| \text{der}(t) \quad \text{Computations} \\
| t[x := u] \quad \text{Let}
\]
Distant Bang: A Subsuming Paradigm
Quantitative Inhabitation for Different Lambda Calculi in a Unifying Framework

Distant Bang: A Subsuming Paradigm

\[ t^N : \text{NAME} \rightarrow \text{BANG} \]
Distant Bang: A Subsuming Paradigm

\[ t^N : \text{NAME} \rightarrow \text{BANG} \]

Static Properties: [BKRV’20]

\[ \text{NAME} \quad t \text{ normal form} \]
Distant Bang: A Subsuming Paradigm

$$t^N : \text{NAME} \rightarrow \text{BANG}$$

Static Properties: [BKRV’20]

$$\text{NAME} \quad t \text{ normal form} \iff t^N \text{ normal form} \quad \text{BANG}$$
Quantitative Inhabitation for Different Lambda Calculi in a Unifying Framework

Distant Bang: A Subsuming Paradigm

\[ t^N : \text{NAME} \rightarrow \text{BANG} \]

Static Properties: [BKRV'20]

\[ \text{NAME} t \text{ normal form} \iff t^N \text{ normal form} \]

Dynamic Properties: [BKRV'20]

\[ \text{NAME} t \rightarrow u \]

Can we do the same thing with inhabitation?
Distant Bang: A Subsuming Paradigm

Static Properties: [BKRV’20]

\[ t^N : \text{NAME} \rightarrow \text{BANG} \]

\[ \text{NAME} \quad t \text{ normal form} \iff t^N \text{ normal form} \]

Dynamic Properties: [BKRV’20]

\[ \text{NAME} \quad t \rightarrow u \iff t^N \rightarrow u^N \]

Can we do the same thing with inhabitation?
Distant Bang: A Subsuming Paradigm

Static Properties: [BKRV’20]

\[ t \overset{N}{\rightarrow} \text{NAME} \rightarrow \overset{N}{\text{BANG}} \]

\[ t \overset{V}{\rightarrow} \text{VALUE} \rightarrow \overset{V}{\text{BANG}} \]

\[ t \overset{N}{\text{normal form}} \Leftrightarrow t \overset{N}{\text{normal form}} \]

Dynamic Properties: [BKRV’20]

\[ t \rightarrow u \Leftrightarrow t \overset{N}{\rightarrow} u \overset{N}{\rightarrow} \]

Can we do the same thing with inhabitation?
Quantitative Inhabitation for Different Lambda Calculi in a Unifying Framework

Distant Bang: A Subsuming Paradigm

Static Properties: [BKRV’20]

```
NAME       t normal form ⇔ t^N normal form
VALUE      t normal form ⇔ t^V normal form
```

Dynamic Properties: [BKRV’20]

```
NAME       t ↠ u ⇔ t^N ↠ u^N
VALUE      t ↠ u ⇔ t^V ↠ u^V
```
Distant Bang: A Subsuming Paradigm

Static Properties: [BKRV’20]

- **NAME**
  - $t$ normal form $\iff t^N$ normal form
  - $t \rightsquigarrow u$ $\iff t^N \rightsquigarrow u^N$

- **VALUE**
  - $t$ normal form $\iff t^V$ normal form
  - $t \rightsquigarrow u$ $\iff t^V \rightsquigarrow u^V$

Dynamic Properties: [BKRV’20]

- **NAME**
  - $t \rightsquigarrow u$ $\iff t^N \rightsquigarrow u^N$

- **VALUE**
  - $t \rightsquigarrow u$ $\iff t^V \rightsquigarrow u^V$

Can we do the same thing with inhabitation?
Quantitative Inhabitation for Different Lambda Calculi in a Unifying Framework
Quantitative Inhabitation for Different Lambda Calculi in a Unifying Framework
Simple Types Versus Intersection Types

\[ A, B ::= \sigma \mid A \Rightarrow B \]
Simple Types Versus Intersection Types

\[ A, B ::= \sigma | A \Rightarrow B \]
Simple Types Versus Intersection Types

\[ A, B ::= \sigma | A \Rightarrow B \]

- Untyped terms
- Terminating terms
- Typable terms

**Qualitative properties**
- Associativity
- Commutativity
- Idempotency

**Quantitative properties**

[CoDe'78], [CoDe'80], [Gard'94], [Kfou'00]
Simple Types Versus Intersection Types

\[ A, B ::= \sigma \mid A \Rightarrow B \mid A \cap B \]

- Untyped terms
- Terminating terms
- Typable terms

Associativity:
\[ A \cap (B \cap C) = (A \cap B) \cap C \]

Commutativity:
\[ A \cap B = B \cap A \]

Idempotency?
\[ A \cap A = A \]

Qualitative properties
Quantitative properties

References:
- [dCarv'07]
- [CoDe'78], [CoDe'80]
- [Gard'94], [Kfou'00]
Simple Types Versus **Intersection Types**

\[ A, B ::= \sigma \mid A \Rightarrow B \mid A \cap B \]

- *Untyped terms*
  - *Terminating terms*
    - *Typable terms*
Simple Types Versus Intersection Types

\[ A, B ::= \sigma \mid A \Rightarrow B \mid A \cap B \]

Untyped terms

Terminating terms

Typable terms

Associativity:

\[ A \cap (B \cap C) = (A \cap B) \cap C \]

Commutativity:

\[ A \cap B = B \cap A \]

Idempotency:

\[ A \cap A = A \]

\[ A \cap A, A \cap A \]

Qualitative properties

Quantitative properties

[CoDe'78], [CoDe'80], [Gard'94], [Kfou'00]

[dCarv'07]
Simple Types Versus **Intersection Types**

\[ A, B ::= \sigma \mid A \Rightarrow B \mid A \cap B \]

- **Untyped terms**
- **Terminating terms**
- **Typable terms**

- **Associativity**:
  \[ A \cap (B \cap C) = (A \cap B) \cap C \]

- **Commutativity**:
  \[ A \cap B = B \cap A \]

- **Idempotency**?

  \[ A \cap A = A \]

  \[ A \cap A \cap \]
Simple Types Versus Intersection Types

\[ A, B ::= \sigma \mid A \Rightarrow B \mid A \cap B \]

- **Associativity:**
  \[ A \cap (B \cap C) = (A \cap B) \cap C \]

Untyped terms

Terminating terms

Typable terms
Simple Types Versus Intersection Types

\[ A, B ::= \sigma \mid A \Rightarrow B \mid A \cap B \]

- **Associativity:**
  \[ A \cap (B \cap C) = (A \cap B) \cap C \]
- **Commutativity:**
  \[ A \cap B = B \cap A \]
Simple Types Versus Intersection Types

\[ A, B ::= \sigma \mid A \Rightarrow B \mid A \cap B \]

- **Associativity:**
  \[ A \cap (B \cap C) = (A \cap B) \cap C \]

- **Commutativity:**
  \[ A \cap B = B \cap A \]

- **Idempotency?**

Untyped terms

Terminating terms

Typable terms

[CoDe'78], [CoDe'80], [Gard'94], [Kfou'00]

[dCarv'07]
Simple Types Versus Intersection Types

\[ A, B ::= \sigma \mid A \Rightarrow B \mid A \cap B \]

- **Untyped terms**
- **Terminating terms**
- **Typable terms**

<table>
<thead>
<tr>
<th>Qualitative properties</th>
<th>Quantitative properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Associativity:</td>
<td>[ A \cap (B \cap C) = (A \cap B) \cap C ]</td>
</tr>
<tr>
<td>Commutativity:</td>
<td>[ A \cap B = B \cap A ]</td>
</tr>
<tr>
<td>Idempotency?</td>
<td>[ A \cap A = A ]</td>
</tr>
</tbody>
</table>

References:
- [CoDe’78], [CoDe’80]
- [dCarv’07]
Simple Types Versus Intersection Types

\[ A, B ::= \sigma \mid A \Rightarrow B \mid A \cap B \]

- **Untyped terms**
  - Terminating terms
  - Typable terms

<table>
<thead>
<tr>
<th>Qualitative properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
</tr>
<tr>
<td>✗</td>
</tr>
</tbody>
</table>

- **Associativity:**
  \[ A \cap (B \cap C) = (A \cap B) \cap C \]

- **Commutativity:**
  \[ A \cap B = B \cap A \]

- **Idempotency?**
  \[ A \cap A = A \]

- **[CoDe’78],[CoDe’80]**
  - Idempotent

- **[dCarv’07]**
  - Qualitative properties
Simple Types Versus Intersection Types

\[ A, B ::= \sigma \mid A \Rightarrow B \mid A \cap B \]

- **Associativity:**
  \[ A \cap (B \cap C) = (A \cap B) \cap C \]

- **Commutativity:**
  \[ A \cap B = B \cap A \]

- **Idempotency?**

<table>
<thead>
<tr>
<th>Idempotent</th>
<th>Non-Idempotent</th>
</tr>
</thead>
<tbody>
<tr>
<td>[CoDe’78], [CoDe’80]</td>
<td>[Gard’94], [Kfou’00]</td>
</tr>
</tbody>
</table>

\[ A \cap A = A \]

\[ A \cap A \neq A \]

Qualitative properties

- [✓]
- [✗]
Simple Types Versus Intersection Types

\[ A, B ::= \sigma \mid A \Rightarrow B \mid A \cap B \]

- **Untyped terms**
  - Terminating terms
  - Typable terms

- **Typable terms**

- **Untyped terms**

- **Types**

- **Qualitative properties**
  - Associativity:
    \[ A \cap (B \cap C) = (A \cap B) \cap C \]
  - Commutativity:
    \[ A \cap B = B \cap A \]
  - Idempotency?

- **Quantitative properties**

- **Idempotent**
  - [CoDe’78], [CoDe’80]
  - \[ A \cap A = A \]

- **Non-Idempotent**
  - [Gard’94], [Kfou’00]
  - \[ A \cap A \neq A \]

- **[dCarv’07]**
## Typability and Inhabitation in Intersection Types

<table>
<thead>
<tr>
<th>Typability</th>
<th>Inhabitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decidable</td>
<td>Decidable</td>
</tr>
<tr>
<td>Indecidable</td>
<td>Indecidable</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Simple Types: Decidable
- Non-Idempotent Types: Indecidable
- Idempotent Types: Decidable
- (CBN) Non-Idempotent Types: Indecidable
<table>
<thead>
<tr>
<th></th>
<th>Typing</th>
<th>Inhabitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sqsubset \vdash t : \square$</td>
<td>$\Gamma \vdash \square : \sigma$</td>
<td></td>
</tr>
</tbody>
</table>
## Typability and Inhabitation in Intersection Types

<table>
<thead>
<tr>
<th>Typing</th>
<th>Inhabitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \vdash t : ? )</td>
<td>( \Gamma \vdash ? : \sigma )</td>
</tr>
<tr>
<td>Simple Types</td>
<td></td>
</tr>
<tr>
<td>Idempotent Types</td>
<td></td>
</tr>
<tr>
<td>Non-Idempotent Types</td>
<td></td>
</tr>
</tbody>
</table>

- **Typing**
  - Simple Types
  - Idempotent Types
  - Non-Idempotent Types

- **Inhabitation**
  - Decidable
  - Indecidable

- **Typability**
  - Decidable
  - Indecidable

- **Idempotent Types**
  - Decidable
  - Indecidable

- **Non-Idempotent Types**
  - Decidable
  - Indecidable

---

\[ \text{Decidable} \quad \text{Indecidable} \]

\[ \text{[Urz'99]} \quad \text{[BKR'18]} \]
### Typability and Inhabitation in Intersection Types

<table>
<thead>
<tr>
<th>Simple Types</th>
<th>Typing ( \ ? \vdash t : ? )</th>
<th>Inhabitation ( \Gamma \vdash ? : \sigma )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idempotent Types</td>
<td>Decidable</td>
<td></td>
</tr>
<tr>
<td>Non-Idempotent Types</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Typability and Inhabitation in Intersection Types

<table>
<thead>
<tr>
<th></th>
<th>Typing $\vdash t : ?$</th>
<th>Inhabitation $\Gamma \vdash ? : \sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple Types</td>
<td>Decidable</td>
<td></td>
</tr>
<tr>
<td>Idempotent Types</td>
<td>Indecidable</td>
<td></td>
</tr>
<tr>
<td>Non-Idempotent Types</td>
<td>Indecidable</td>
<td></td>
</tr>
</tbody>
</table>
Typability and Inhabitation in Intersection Types

<table>
<thead>
<tr>
<th></th>
<th>Typing</th>
<th>Inhabitation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\Gamma \vdash ? : \sigma$</td>
<td></td>
</tr>
<tr>
<td>Simple Types</td>
<td>Decidable</td>
<td>Decidable</td>
</tr>
<tr>
<td>Idempotent Types</td>
<td>Indecidable</td>
<td></td>
</tr>
<tr>
<td>Non-Idempotent Types</td>
<td>Indecidable</td>
<td></td>
</tr>
</tbody>
</table>
Typability and Inhabitation in Intersection Types

<table>
<thead>
<tr>
<th></th>
<th>Typing</th>
<th>Inhabitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>? ⊢ t : ?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>∑ ? : σ</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Simple Types             | Decidable | Decidable |
| Non-Idempotent Types     | Indecidable | Indecidable [Urz’99] |
| Idempotent Types         | Indecidable | Indecidable |

Typing in simple types is decidable, while inhabitation is also decidable. For idempotent types, typing is decidable, but inhabitation is indecidable. Non-idempotent types exhibit the same indecidability for both typing and inhabitation.

References:
- [Urz’99]
## Typability and Inhabitation in Intersection Types

<table>
<thead>
<tr>
<th></th>
<th>Typing</th>
<th>Inhabitation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\Gamma \vdash \tau : \sigma$</td>
<td>$\Gamma \vdash ? : \sigma$</td>
</tr>
<tr>
<td><strong>Simple Types</strong></td>
<td>Decidable</td>
<td>Decidable</td>
</tr>
<tr>
<td><strong>Idempotent Types</strong></td>
<td>Indecidable</td>
<td>Indecidable [Urz’99]</td>
</tr>
<tr>
<td><strong>Non-Idempotent Types</strong></td>
<td>Indecidable</td>
<td>(CBN) Decidable [BKR’18]</td>
</tr>
<tr>
<td>Types</td>
<td>Typing $\ ? \vdash t : ?$</td>
<td>Inhabitation $\Gamma \vdash ? : \sigma$</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>Simple Types</td>
<td>Decidable</td>
<td>Decidable</td>
</tr>
<tr>
<td>Idempotent Types</td>
<td>Indecidable</td>
<td>Indecidable [Urz’99]</td>
</tr>
<tr>
<td>Non-Idempotent Types</td>
<td>Indecidable</td>
<td>(CBN) Decidable [BKR’18] (CBV) ?</td>
</tr>
<tr>
<td></td>
<td>Typing $\ ? \vdash t : ?$</td>
<td>Inhabitation $\Gamma \vdash ? : \sigma$</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td><strong>Simple Types</strong></td>
<td>Decidable</td>
<td>Decidable</td>
</tr>
<tr>
<td><strong>Idempotent Types</strong></td>
<td>Indecidable</td>
<td>Indecidable [Urz’99]</td>
</tr>
<tr>
<td><strong>Non-Idempotent Types</strong></td>
<td>Indecidable</td>
<td>(CBN) Decidable [BKR’18]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(CBV) ?</td>
</tr>
</tbody>
</table>
## Typability and Inhabitation in Intersection Types

<table>
<thead>
<tr>
<th></th>
<th>Typing $\vdash t : ?$</th>
<th>Inhabitation $\Gamma \vdash ? : \sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Simple Types</strong></td>
<td>Decidable</td>
<td>Decidable</td>
</tr>
<tr>
<td><strong>Idempotent Types</strong></td>
<td>Indecidable</td>
<td>Indecidable $[\text{Urz'99}]$</td>
</tr>
<tr>
<td><strong>Non-Idempotent Types</strong></td>
<td>Indecidable</td>
<td>(CBN) Decidable $[\text{BKR'18}]$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(CBV) Decidable</td>
</tr>
</tbody>
</table>
Intersection Types and Distant Bang Calculus
Quantitative Inhabitation for Different Lambda Calculi in a Unifying Framework

Intersection Types and Distant Bang Calculus

Three Typing Systems: [BKRV’20]

- **NAME**: \( N \)
- **VALUE**: \( V \)
- **BANG**: \( B \)
Quantitative Inhabitation for Different Lambda Calculi in a Unifying Framework

Intersection Types and Distant Bang Calculus

Three Typing Systems: [BKRV’20]

\[
\textbf{NAME} : N \\
\textbf{VALUE} : V \\
\textbf{BANG} : B
\]

Static Properties: [BKRV’20]

\[
\Gamma \vdash_N t : \sigma
\]
Three Typing Systems: [BKRV’20]

- **NAME**: $\mathcal{N}$
- **VALUE**: $\mathcal{V}$
- **BANG**: $\mathcal{B}$

Static Properties: [BKRV’20]

\[
\Gamma \vdash_{\mathcal{N}} t : \sigma \quad \iff \quad \Gamma \vdash_{\mathcal{B}} t^N : \sigma
\]
Intersection Types and Distant Bang Calculus

Three Typing Systems: [BKRV’20]

- **NAME**: \( N \)
- **VALUE**: \( V \)
- **BANG**: \( B \)

Static Properties: [BKRV’20]

- \( \Gamma \vdash_{N} t : \sigma \) \iff \( \Gamma \vdash_{B} t^{N} : \sigma \)
- \( \Gamma \vdash_{V} t : \sigma \) \iff \( \Gamma \vdash_{B} t^{V} : \sigma \)
Quantitative Inhabitation for Different Lambda Calculi in a Unifying Framework
Quantitative Inhabitation for Different Lambda Calculi in a Unifying Framework

Coming Back to Inhabitation

First Goal

More Ambitious Second Goal

Decidability of the (more general) Inhabitation Problem (IP).

More Ambitious Third Goal

Decidability by finding all inhabitants in the IP.

Decidability of the IP by finding all inhabitants from those of the IP.

Using generic properties so that other encodable models of computation can use these results.
Coming Back to Inhabitation

First Goal

- **Decidability** of the (more general) **BANG** Inhabitation Problem (IP).
First Goal + More Ambitious Second Goal

- **Decidability** of the (more general) **BANG** Inhabitation Problem (IP).
- **Decidability** of the **NAME** and **VALUE** IP from **decidability** of the **BANG** IP.
Coming Back to Inhabitation

First Goal + More Ambitious Second Goal

- **Decidability** of the (more general) BANG Inhabitation Problem (IP).
- **Decidability** of the NAME and VALUE IP from **decidability** of the BANG IP.

More Ambitious Third Goal
Coming Back to Inhabitation

First Goal + More Ambitious Second Goal

- **Decidability** of the (more general) \textbf{BANG} Inhabitation Problem (IP).
- **Decidability** of the \textbf{NAME} and \textbf{VALUE} IP from **decidability** of the \textbf{BANG} IP.

More Ambitious Third Goal

- Decidability by **finding all inhabitants** in the \textbf{BANG} IP.
Coming Back to Inhabitation

**First Goal + More Ambitious Second Goal**

- **Decidability** of the (more general) **BANG** Inhabitation Problem (IP).
- **Decidability** of the **NAME** and **VALUE** IP from **decidability** of the **BANG** IP.

**More Ambitious Third Goal**

- Decidability by **finding all inhabitants** in the **BANG** IP.
- Decidability of the **NAME** and **VALUE** IP by **finding all inhabitants** from those of the **BANG** IP.
Coming Back to Inhabitation

First Goal + More Ambitious Second Goal

- **Decidability** of the (more general) **BANG** Inhabitation Problem (IP).
- **Decidability** of the **NAME** and **VALUE** IP from **decidability** of the **BANG** IP.

More Ambitious Third Goal

- Decidability by **finding all inhabitants** in the **BANG** IP.
- Decidability of the **NAME** and **VALUE** IP by **finding all inhabitants** from those of the **BANG** IP.
- Using generic properties so that other encodable models of computation can use these results.
Solving the Inhabitation Problem - Methodology
Instead of just one solution:
\[ \Gamma \vdash t : \sigma \]
We want to compute all solutions:
\[ \text{Sol}(\Gamma, \sigma) := \{ t \mid \Gamma \vdash t : \sigma \} \]
Solving the Inhabitation Problem - Methodology

Instead of just one solution:
\[ \Gamma \vdash t : \sigma \]
We want to compute all solutions:
\[ \text{Sol}(\Gamma, \sigma) := \{ t \mid \Gamma \vdash t : \sigma \} \]
Instead of just one solution:
\[ \Gamma \vdash t : \sigma \]
We want to compute all solutions:
\[ \text{Sol}(\Gamma, \sigma) := \{ t \mid \Gamma \vdash t : \sigma \} \]

Problem

- The set \( \text{Sol}(\Gamma, \sigma) \) is either empty or infinite
Instead of just one solution:
\[ \Gamma \vdash t : \sigma \]
We want to compute all solutions:
\[ \text{Sol}(\Gamma, \sigma) := \{ t \mid \Gamma \vdash t : \sigma \} \]

**Problem**

- The set \( \text{Sol}(\Gamma, \sigma) \) is either empty or infinite

We compute a finite generator:
\[ \text{Basis}(\Gamma, \sigma) \]
Which is correct and complete:
\[ \text{span}(\text{Basis}(\Gamma, \sigma)) = \text{Sol}(\Gamma, \sigma) \]
Solving the Inhabitation Problem - Methodology

Instead of **just one** solution:
\[ \Gamma \vdash t : \sigma \]

We want to compute **all** solutions:
\[ \text{Sol}(\Gamma, \sigma) := \{ t \mid \Gamma \vdash t : \sigma \} \]

**Problem**

The set \( \text{Sol}(\Gamma, \sigma) \) is either empty or infinite

We compute a **finite** generator:
\[ \text{Basis}(\Gamma, \sigma) \]
Which is **correct** and **complete**:
\[ \text{span}(\text{Basis}(\Gamma, \sigma)) = \text{Sol}(\Gamma, \sigma) \]
Solving the Inhabitation Problem - Methodology

Instead of **just one** solution:
\[ \Gamma \vdash t : \sigma \]

We want to compute **all** solutions:
\[ \text{Sol}(\Gamma, \sigma) := \{ t \mid \Gamma \vdash t : \sigma \} \]

**Problem**

❌ The set \( \text{Sol}(\Gamma, \sigma) \) is either empty or infinite

We compute a **finite** generator:
\[ \text{Basis}(\Gamma, \sigma) \]

Which is **correct** and **complete**:
\[ \text{span}(\text{Basis}(\Gamma, \sigma)) = \text{Sol}(\Gamma, \sigma) \]
Solving the Inhabitation Problem - Methodology

Instead of **just one** solution:
\[ \Gamma \vdash t : \sigma \]
We want to compute **all** solutions:
\[ \text{Sol}(\Gamma, \sigma) := \{ t \mid \Gamma \vdash t : \sigma \} \]

**Problem**

- The set \( \text{Sol}(\Gamma, \sigma) \) is either empty or infinite

**Theorem**

- For any typing \((\Gamma, \sigma)\), \( \text{Basis}_{\mathcal{G}}(\Gamma, \sigma) \) exists, is **finite**, **correct** and **complete**.
Solving the Inhabitation Problem - Methodology

Instead of **just one** solution:

\[ \Gamma \vdash t : \sigma \]

We want to compute all solutions:

\[ \text{Sol}(\Gamma, \sigma) := \{ t \mid \Gamma \vdash t : \sigma \} \]

Problem

- The set \( \text{Sol}(\Gamma, \sigma) \) is either empty or infinite

We compute a **finite** generator:

\[ \text{Basis}(\Gamma, \sigma) \]

Which is **correct** and **complete**:

\[ \text{span}(\text{Basis}(\Gamma, \sigma)) = \text{Sol}(\Gamma, \sigma) \]

Theorem

- For any typing \((\Gamma, \sigma)\), \( \text{Basis}_{\text{B}}(\Gamma, \sigma) \) exists, is **finite**, **correct** and **complete**.
Following the Typing and a Grammar
Following the Typing and a Grammar

Computing the basis:
Recreate typing trees, but only on elements of the Basis.
Following the Typing and a Grammar

**Computing the basis:**
Recreate typing trees, but only on elements of the Basis.

Follows two sets of rules:
Following the Typing and a Grammar

Computing the basis:
Recreate typing trees, but only on elements of the Basis.

Follows two sets of rules:
- Typing rules

\( \text{Sol}(\Gamma, \sigma) \)
Following the Typing and a Grammar

**Computing the basis:**
Recreate typing trees, but only on elements of the Basis.

Follows two sets of rules:
- Typing rules
- Grammar rules
Following the Typing and a Grammar

Computing the basis:
Recreate typing trees, but only on elements of the Basis.

Follows two sets of rules:
- Typing rules
- Grammar rules

\[ \text{Sol}(\Gamma, \sigma) \cap \text{Basis}_{\mathcal{G}} \]
The Full Algorithm
The Full Algorithm

\[ g \mapsto \text{Var} \quad | \quad g \mapsto \text{Der}(g') \]
\[ x \mapsto_g H_{x[1]}(\sigma) \quad | \quad a \vdash_{g'} H_{x[1]}(\Gamma; [\sigma]) \]
\[ \text{dr} \]

\[ g \mapsto \text{App}(g_\alpha, g_\beta) \]
\[ \Gamma = \Gamma_\alpha + \Gamma_\beta \]
\[ M \Rightarrow \alpha \equiv S(\tau, \varnothing) \Rightarrow \alpha \]
\[ a \vdash_{g_\alpha} H_{x[1]}(\Gamma_\alpha; M \Rightarrow \alpha) \quad b \vdash_{g_\beta} N(\Gamma_\beta; M) \]
\[ ab \vdash_{g} H_{x[1]}(\Gamma; \alpha) \]

\[ g \mapsto g' \quad | \quad a \vdash_{g'} H_{x[1]}(\Gamma; \alpha) \]
\[ \Gamma \mapsto \Gamma' + a : \{ x \} \quad | \quad a \vdash_{g'} H_{x[1]}(\Gamma'; \alpha) \]
\[ \Gamma \mapsto \Gamma' \quad | \quad a \vdash_{g} N(\Gamma; \alpha) \]
\[ \vdash_{g} N(\Gamma; \sigma) \quad \vdash_{g} N(\Gamma; \sigma) \]

\[ g \mapsto \text{Lam}(g') \]
\[ \text{def} \]
\[ f \vdash_{x} x \notin \text{dom}(\Gamma) \quad | \quad a \vdash_{g'} N(\Gamma, x : M; \alpha) \]
\[ \lambda x. a \vdash_{g} N(\Gamma; M \Rightarrow \alpha) \]

\[ g \mapsto \text{Brg}(g') \]
\[ \text{def} \]
\[ \Gamma \mapsto \Gamma' + a : \{ x \} \quad | \quad a \vdash_{g} N(\Gamma; \tau_\alpha) \quad \Gamma \mapsto \Gamma' \quad | \quad a \vdash_{g} N(\Gamma; \sigma) \]
\[ \vdash_{g} N(\Gamma; [\tau_\alpha]_{x}) \quad \vdash_{g} N(\Gamma; [\tau_\alpha]_{x}) \quad \Gamma \mapsto \Gamma' \quad | \quad \vdash_{g} N(\Gamma; \tau_\alpha) \]

\[ g \mapsto \text{Sub}(g_\alpha, g_\beta) \]
\[ \text{def} \]
\[ \Gamma = \Gamma_\alpha + \Gamma_\beta + a : \{ p \} \quad | \quad f \vdash_{y} y \notin \text{dom}(\Gamma') \cup \{ x \} \]
\[ n \in [0, \text{sz}(\rho)] \quad | \quad M \vdash_{x} S(\tau, \varnothing, \ldots, \varnothing) \]
\[ \epsilon \in [1, n] \quad | \quad a \vdash_{g_\alpha} H_{x[1]}(\Gamma_\alpha, y : M; \alpha) \quad b \vdash_{g_\beta} H_{x[1]}(\Gamma_\beta; M) \]
\[ \epsilon \vdash_{g_\alpha} H_{x[1]}(\Gamma_\alpha, y : M; \alpha) \quad \epsilon \vdash_{g_\beta} H_{x[1]}(\Gamma_\beta; M) \]

\[ g \mapsto g' \]
\[ \text{def} \]
\[ \Gamma = \Gamma_\alpha + \Gamma_\beta + a : \{ p \} \quad | \quad f \vdash_{y} y \notin \text{dom}(\Gamma) \cup \{ x \} \]
\[ n \in [1, \text{sz}(\rho)] \quad | \quad \epsilon \in [1, n] \quad | \quad a \vdash_{g_\alpha} H_{x[1]}(\Gamma_\alpha, y : \{ p \}_{\epsilon} [\lambda n. \varnothing]; \alpha) \quad b \vdash_{g_\beta} H_{x[1]}(\Gamma_\beta; M) \]
\[ \alpha \vdash_{g_\alpha} H_{x[1]}(\Gamma_\alpha, y : \{ p \}_{\epsilon} [\lambda n. \varnothing]; \alpha) \quad \alpha \vdash_{g_\beta} H_{x[1]}(\Gamma_\beta; M) \]

\[ g \mapsto g' \]
\[ \text{def} \]
\[ \Gamma = \Gamma_\alpha + \Gamma_\beta + a : \{ p \} \quad | \quad f \vdash_{y} y \notin \text{dom}(\Gamma) \cup \{ x \} \]
\[ n \in [0, \text{sz}(\rho)] \quad | \quad M \vdash_{x} S(\tau, \varnothing, \ldots, \varnothing) \]
\[ \epsilon \in [1, n] \quad | \quad a \vdash_{g_\alpha} N(\Gamma_\alpha, y : M; \alpha) \quad b \vdash_{g_\beta} H_{x[1]}(\Gamma_\beta; M) \]
\[ a \vdash_{g_\alpha} N(\Gamma_\alpha, y : M; \alpha) \quad a \vdash_{g_\alpha} N(\Gamma_\alpha, y : M; \alpha) \]
The Full Algorithm
The Full Algorithm
The Full Algorithm

\[ g \mapsto \text{App}(ga, gb) \]

\[ \Gamma = \Gamma_\alpha + \Gamma_\beta \]

\[ M \Rightarrow \sigma \vdash S(\tau, \diamond \Rightarrow \sigma) \]

\[ a \vdash_{ga} H^{x:[\tau]}(\Gamma_\alpha; M \Rightarrow \sigma) \]

\[ b \vdash_{gb} N(\Gamma_\beta; M) \]

\[ ab \vdash_{g} H^{x:[\tau]}(\Gamma; \sigma) \]
The Full Algorithm
The Full Algorithm
The Full Algorithm and its Implementation

An Implementation of the Quantitative Inhabitation Algorithm for Different Lambda Calculi in a Unifying Framework

github/ArrialVictor/InhabitationLambdaBang
Properties of the Inhabitation Algorithm

Theorem

The inhabitation algorithm terminates. The algorithm is sound and complete (i.e. it exactly computes $B(\Gamma, \sigma)$).

More Ambitious Third Goal

Decidability by finding all inhabitants in the IP.

Decidability of the and IP by finding all inhabitants from those of the IP.

Using generic properties so that other encodable models of computation can use these results.
Properties of the Inhabitation Algorithm

Non-deterministic algorithm
Properties of the Inhabitation Algorithm

Non-deterministic algorithm

Theorem

The inhabitation algorithm terminates.
Properties of the Inhabitation Algorithm

Non-deterministic algorithm

Theorem

- The inhabitation algorithm terminates.
- The algorithm is sound and complete (i.e. it exactly computes Basis_{B}(Γ, σ)).
Properties of the Inhabitation Algorithm

Non-deterministic algorithm

Theorem

✅ The inhabitation algorithm terminates.

✅ The algorithm is sound and complete (i.e. it exactly computes Basis$_B$(Γ, σ)).

More Ambitious Third Goal

✅ Decidability by **finding all inhabitants** in the BANG IP.
Properties of the Inhabitation Algorithm

Non-deterministic algorithm

Theorem

- The inhabitation algorithm terminates.
- The algorithm is sound and complete (i.e. it exactly computes $\text{Basis}_B(\Gamma, \sigma)$).

More Ambitious Third Goal

- Decidability by finding all inhabitants in the BANG IP.
- Decidability of the NAME and VALUE IP by finding all inhabitants from those of the BANG IP.
- Using generic properties so that other encodable models of computation can use these results.
Theorem (\[BKR'14\])

For any typing \((\Gamma, \sigma)\), there exists a basis \(N(\Gamma, \sigma)\) that is finite, correct and complete.

Built an algorithm computing \(N(\Gamma, \sigma)\): \[BKR'14\]
Theorem ([BKR'14])

For any typing \((\Gamma, \sigma)\), \(\text{Basis}_N(\Gamma, \sigma)\) exists, is finite, correct and complete.
Theorem ([BKR’14])

For any typing \((\Gamma, \sigma)\), \(\text{Basis}_N(\Gamma, \sigma)\) exists, is finite, correct and complete.

Built an algorithm computing \(\text{Basis}_N(\Gamma, \sigma)\) : [BKR’14]

\[
\frac{\Gamma = \Gamma_1 + \Gamma_2 \quad a \vdash T(x : A, \tau) \quad x \notin \text{dom}(\Gamma)}{\lambda x.a \vdash T(\Gamma, A \to \tau)} \quad \text{(Abs)}
\]

\[
\frac{\bigvee_{i \in I} a_i \vdash T(\Gamma_i, \sigma_i)}{\text{Union}}
\]

\[
\frac{\Gamma = \Gamma_1 + \Gamma_2 \quad \vdash \text{H}^x[A_1 \to \ldots A_n \to B \to \tau](\Gamma_1, B \to \tau) \quad b \vdash T(\Gamma_2, B) \quad n \geq 0}{ab \vdash \text{H}^x[A_1 \to \ldots A_n \to B \to \tau](\Gamma, \tau) \quad \text{(Head\_\_0)}}
\]

\[
\frac{\vdash \text{H}^x[\tau](\emptyset, \tau)}{\text{(Head\_0)}}
\]

\[
\frac{\vdash \text{H}^x[A_1 \to \ldots A_n \to \tau](\Gamma, \tau)}{a \vdash T(\Gamma + x : [A_1 \to \ldots A_n \to \tau], \tau) \quad \text{(Head)}}
\]
Solving Inhabitation: through Inhabitation

The Basis is preserved by the embedding:

\[ t \in \text{Basis}_N(\Gamma, \sigma) \iff t \in \text{Basis}_B(\Gamma, \sigma) \]

Embedding

Decoding
The Basis is preserved by the embedding:

**Theorem**

\[ t \in \text{Basis}_N(\Gamma, \sigma) \]
The Basis is preserved by the embedding:

Theorem

\[ t \in \text{Basis}_N(\Gamma, \sigma) \iff t^N \in \text{Basis}_B(\Gamma, \sigma) \]
The Basis is preserved by the embedding:

**Theorem**

\[ t \in \text{Basis}_N(\Gamma, \sigma) \iff t^N \in \text{Basis}_B(\Gamma, \sigma) \]
The Basis is preserved by the embedding:

**Theorem**

\[ t \in \text{Basis}_N(\Gamma, \sigma) \iff t^N \in \text{Basis}_B(\Gamma, \sigma) \]
The Basis is preserved by the embedding:

**Theorem**

\[ t \in \text{Basis}_N(\Gamma, \sigma) \iff t^N \in \text{Basis}_B(\Gamma, \sigma) \]
The Basis is preserved by the embedding:

**Theorem**

\[ t \in \text{Basis}_N(\Gamma, \sigma) \iff t^N \in \text{Basis}_{B}(\Gamma, \sigma) \]
The Basis is preserved by the embedding:

**Theorem**

\[ t \in \text{Basis}_N(\Gamma, \sigma) \iff t^N \in \text{Basis}_B(\Gamma, \sigma) \]
The Basis is preserved by the embedding:

Theorem

\[ t \in \text{Basis}_N(\Gamma, \sigma) \iff t^N \in \text{Basis}_B(\Gamma, \sigma) \]
Solving Inhabitation: through Inhabitation

The Basis is preserved by the embedding:

Theorem

\[ t \in \text{Basis}_N(\Gamma, \sigma) \iff t^N \in \text{Basis}_B(\Gamma, \sigma) \]
Solving Inhabitation: through Inhabitation

The Basis is preserved by the embedding:

Theorem

\[ t \in \text{Basis}_N(\Gamma, \sigma) \iff t^N \in \text{Basis}_B(\Gamma, \sigma) \]
The Basis is preserved by the embedding:

**Theorem**

\[ t \in \text{Basis}_N(\Gamma, \sigma) \iff t^N \in \text{Basis}_B(\Gamma, \sigma) \]
The Basis is preserved by the embedding:

Theorem

\[ t \in \text{Basis}_N(\Gamma, \sigma) \iff t^N \in \text{Basis}_B(\Gamma, \sigma) \]
Solving Inhabitation - Usual Methodology
For any typing $(\Gamma, \sigma)$, $\text{Basis}_V(\Gamma, \sigma)$ exists, is finite, correct and complete.
Theorem

For any typing \((\Gamma, \sigma)\), \(\text{Basis}_V(\Gamma, \sigma)\) exists, is finite, correct and complete.

Built an algorithm computing \(\text{Basis}_V(\Gamma, \sigma)\):
Theorem

For any typing \((\Gamma, \sigma)\), \(\text{Basis}_V(\Gamma, \sigma)\) exists, is finite, correct and complete.

Built an algorithm computing \(\text{Basis}_V(\Gamma, \sigma)\):

\[
\begin{align*}
\Gamma = \Gamma_0 + \Gamma_n, \quad & \text{fix } y \notin \text{dom}(\Gamma) \cup \{x\} \\
\Gamma = \Gamma_0 + \Gamma_b, \quad & \text{fix } y \notin \text{dom}(\Gamma) \cup \{x\} \\
\Gamma = \Gamma_0 + \Gamma_b + z : [\tau], \quad & \text{fix } y \notin \text{dom}(\Gamma) \cup \{x\} \\
\end{align*}
\]
Quantitative Inhabitation for Different Lambda Calculi in a Unifying Framework

The Basis is preserved by the embedding:

\[ \text{Theorem } t \in \text{Basis } V(\Gamma, \sigma) \iff t \nu \in \text{Basis } B(\Gamma, \sigma) \]
The Basis is preserved by the embedding:

**Theorem**

\[ t \in \text{Basis}_V(\Gamma, \sigma) \]
The Basis is preserved by the embedding:

Theorem

\[ t \in \text{Basis}_V(\Gamma, \sigma) \iff \text{t}^V \in \text{Basis}_B(\Gamma, \sigma) \]
The Basis is preserved by the embedding:

\[ t \in \text{Basis}_V(\Gamma, \sigma) \iff t^V \in \text{Basis}_B(\Gamma, \sigma) \]
The Basis is preserved by the embedding:

**Theorem**

\[ t \in \text{Basis}_V(\Gamma, \sigma) \iff t^V \in \text{Basis}_B(\Gamma, \sigma) \]
The Basis is preserved by the embedding:

\[ t \in \text{Basis}_V(\Gamma, \sigma) \iff t^V \in \text{Basis}_B(\Gamma, \sigma) \]
The Basis is preserved by the embedding:

**Theorem**

\[ t \in \text{Basis}_V(\Gamma, \sigma) \iff t^V \in \text{Basis}_B(\Gamma, \sigma) \]
The Basis is preserved by the embedding:

Theorem

\[ t \in \text{Basis}_V(\Gamma, \sigma) \iff t^V \in \text{Basis}_B(\Gamma, \sigma) \]
The Basis is preserved by the embedding:

**Theorem**

\[ t \in \text{Basis}_V(\Gamma, \sigma) \iff t^V \in \text{Basis}_B(\Gamma, \sigma) \]
The Basis is preserved by the embedding:

**Theorem**

\[ t \in \text{Basis}_V(\Gamma, \sigma) \iff t^V \in \text{Basis}_B(\Gamma, \sigma) \]
The Basis is preserved by the embedding:

**Theorem**

\[ t \in \text{Basis}_V(\Gamma, \sigma) \iff t^V \in \text{Basis}_B(\Gamma, \sigma) \]
The Basis is preserved by the embedding:

**Theorem**

\[ t \in \text{Basis}_V(\Gamma, \sigma) \iff t^V \in \text{Basis}_B(\Gamma, \sigma) \]
The Basis is preserved by the embedding:

**Theorem**

\[ t \in \text{Basis}_V(\Gamma, \sigma) \quad \iff \quad t^V \in \text{Basis}_B(\Gamma, \sigma) \]
Properties of the Indirect NAME and VALUE Algorithm

- The inhabitation algorithm terminates.
- The algorithm is sound and complete (i.e. it exactly computes Basis $B(\Gamma, \sigma)$).

- More Ambitious Third Goal

Decidability by finding all inhabitants in the IP.

Decidability of the IP by finding all inhabitants from those of the IP.

- Using generic properties so that other encodable models of computation can use these results.
<table>
<thead>
<tr>
<th>Theorem</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔️ The inhabitation algorithm terminates.</td>
</tr>
<tr>
<td>✔️ The algorithm is sound and complete (i.e. it exactly computes Basis(_g(\Gamma, \sigma))).</td>
</tr>
</tbody>
</table>
Properties of the Indirect **NAME** and **VALUE** Algorithm

### Theorem

- ✔ The inhabitation algorithm terminates.
- ✔ The algorithm is sound and complete
  *i.e. it exactly computes Basis$\_B(\Gamma, \sigma)$.*

### More Ambitious Third Goal

- ✔ Decidability by **finding all inhabitants** in the **BANG** IP.
  - Decidability of the **NAME** and **VALUE** IP by **finding all inhabitants** from those of the **BANG** IP.
  - Using generic properties so that other encodable models of computation can use these results.
Theorem

- The inhabitation algorithm terminates.
- The algorithm is sound and complete (i.e. it exactly computes Basis \((\Gamma, \sigma)\)).

More Ambitious Third Goal

- Decidability by finding all inhabitants in the \textbf{BANG} IP.
  - Decidability of the \textbf{NAME} and \textbf{VALUE} IP by finding all inhabitants from those of the \textbf{BANG} IP.
  - Using generic properties so that other encodable models of computation can use these results.
Properties of the Indirect **NAME** and **VALUE** Algorithm

**Theorem**

- The inhabitation algorithm terminates.
- The algorithm is sound and complete (i.e. it exactly computes Basis \((\Gamma, \sigma)\)).

**More Ambitious Third Goal**

- Decidability by finding all inhabitants in the **BANG** IP.
- Decidability of the **NAME** and **VALUE** IP by finding all inhabitants from those of the **BANG** IP.
- Using generic properties so that other encodable models of computation can use these results.
Properties of the Indirect **NAME** and **VALUE** Algorithm

Theorem

- ✔ The inhabitation algorithm terminates.
- ✔ The algorithm is sound and complete (i.e. it exactly computes Basis \((\Gamma, \sigma)\)).

More Ambitious Third Goal

- ✔ Decidability by **finding all inhabitants** in the **BANG** IP.
- ✔ Decidability of the **NAME** and **VALUE** IP by **finding all inhabitants** from those of the **BANG** IP.
- Using generic properties so that other encodable models of computation can use these results.
Properties of the Indirect NAME and VALUE Algorithm

Theorem

- The inhabitation algorithm terminates.
- The algorithm is sound and complete (i.e. it exactly computes Basis \((\Gamma, \sigma)\)).

More Ambitious Third Goal

- Decidability by finding all inhabitants in the BANG IP.
- Decidability of the NAME and VALUE IP by finding all inhabitants from those of the BANG IP.
- Using generic properties so that other encodable models of computation can use these results.
Properties of the Indirect NAME and VALUE Algorithm

Theorem

- The inhabitation algorithm terminates.
- The algorithm is sound and complete (i.e. it exactly computes Basis ($\Gamma, \sigma$)).

More Ambitious Third Goal

- Decidability by finding all inhabitants in the BANG IP.
- Decidability of the NAME and VALUE IP by finding all inhabitants from those of the BANG IP.
- Using generic properties so that other encodable models of computation can use these results.
Conclusion

Summary:
Solving the generalized inhabitation problem
A several-for-one deal:
An implementation: (github/ArrialVictor/InhabitationLambdaBang)

Further questions and ongoing work:
Solvability (for Different Calculi in a Unified Framework)
Strengthening inhabitation for lambda-calculus with pattern matching [BKRdR'21]

Thanks for your attention!
Summary:
- Solving the generalized inhabitation problem
- A several-for-one deal: [BANG] [NAME] [VALUE] [OTHERS]
- An implementation: (github/ArrialVictor/InhabitationLambdaBang)

Further questions and ongoing work:
Solvability (for Different Calculi in a Unified Framework)
Strengthening inhabitation for lambda-calculus with pattern matching [BKRdR'21]
Conclusion

Summary:

- Solving the generalized inhabitation problem
- A several-for-one deal: \textbf{BANG NAME VALUE OTHERS}
- An implementation: (github/ArrialVictor/InhabitationLambdaBang)

Further questions and ongoing work:

- Solvability (for Different Calculi in a Unified Framework)
- Strengthening inhabitation for lambda-calculus with pattern matching [BKRdR’21]
Summary:

- Solving the generalized inhabitation problem
- A several-for-one deal: BANG NAME VALUE OTHERS
- An implementation: (github/ArrialVictor/InhabitationLambdaBang)

Further questions and ongoing work:

- Solvability (for Different Calculi in a Unified Framework)
- Strengthening inhabitation for lambda-calculus with pattern matching [BKRdR’21]

Thanks for your attention!