TypOS: An “Operating System” for Typechecking Actors

Guillaume Allais Malin Altenmüller Conor McBride
Georgi Nakov **Fredrik Nordvall Forsberg** Craig Roy

University of St Andrews, University of Strathclyde, and Quantinuum

22 June 2022, TYPES, Nantes

A domain-specific language for typecheckers

An experiment in how to write typecheckers that make (more) sense.

A domain-specific language for typecheckers

An experiment in how to write typecheckers that make (more) sense.

Similar endeavours: Andromeda [Bauer, Haselwarter and Petkovic 2020], Redex [Felleisen, Findler and Flatt 2009], Turnstyle+ [Chang, Ballantyne, Turner and Bowman], ...

However we try to minimise demands on the order in which subproblems are solved.

A domain-specific language for typecheckers

An experiment in how to write typecheckers that make (more) sense.

Similar endeavours: Andromeda [Bauer, Haselwarter and Petkovic 2020], Redex [Felleisen, Findler and Flatt 2009], Turnstyle+ [Chang, Ballantyne, Turner and Bowman], ...

However we try to minimise demands on the order in which subproblems are solved.

Conor McBride, 20 years ago implementing Epigram 1:

> *"[redacted] me, I'm implementing an operating system!"*

A domain-specific language for typecheckers

An experiment in how to write typecheckers that make (more) sense.

Similar endeavours: Andromeda [Bauer, Haselwarter and Petkovic 2020], Redex [Felleisen, Findler and Flatt 2009], Turnstyle+ [Chang, Ballantyne, Turner and Bowman], ...

However we try to minimise demands on the order in which subproblems are solved.

Conor McBride, 20 years ago implementing Epigram 1:

> *"[redacted] me, I'm implementing an operating system!"*

A domain-specific language for typecheckers

An experiment in how to write typecheckers that make (more) sense.

Similar endeavours: Andromeda [Bauer, Haselwarter and Petkovic 2020], Redex [Felleisen, Findler and Flatt 2009], Turnstyle+ [Chang, Ballantyne, Turner and Bowman], ...

However we try to minimise demands on the order in which subproblems are solved.

Conor McBride, 20 years ago implementing Epigram 1:

> *"[redacted] me, I'm implementing an operating system!"*

Concrete motivation: implementing a type theory with rich equational theory for free monoids and free Abelian groups.

Why not just a shallow embedding?

Why not just a shallow embedding?

Logical Framework aspects: we implement syntax with binding once, and then it Just Works.

Why not just a shallow embedding?

Logical Framework aspects: we implement syntax with binding once, and then it Just Works.

Resumptions should be updatable: progress might have happened while a process was asleep.

Why not just a shallow embedding?

Logical Framework aspects: we implement syntax with binding once, and then it Just Works.

Resumptions should be updatable: progress might have happened while a process was asleep.

Ruling out design errors by construction: a first-order representation means we can do static analysis on the typecheckers themselves.

A Tour of TypOS

Syntax descriptions

Syntax descriptions

We support a Lisp-style generic syntax for terms:

- atoms 'a
- cons lists [t_0 t_1 $\ldots$ t_n]
- variables x and bindings \x. t

Syntax descriptions

We support a Lisp-style generic syntax for terms:

- atoms 'a
- cons lists [t_0 t_1 ... t_n]
- variables x and bindings \$x.\ t$

Simple and uniform to write and parse.

Syntax descriptions

We support a Lisp-style generic syntax for terms:

- atoms 'a
- cons lists $[t_0\ t_1\ \ldots\ t_n]$
- variables x and bindings $\backslash x.\, t$

Simple and uniform to write and parse.

Users can restrict the shape of terms using context-free syntax descriptions.

Syntax descriptions

We support a Lisp-style generic syntax for terms:

- atoms 'a
- cons lists `[`t_0 t_1 `...` t_n`]`
- variables x and bindings `\`$x.\, t$

Simple and uniform to write and parse.

Users can restrict the shape of terms using context-free syntax descriptions. We always offer a `Wildcard` description allowing anything.

Syntax descriptions

We support a Lisp-style generic syntax for terms:

- atoms 'a
- cons lists [t_0 t_1 ... t_n]
- variables x and bindings \$x.\,t$

Simple and uniform to write and parse.

Users can restrict the shape of terms using context-free syntax descriptions. We always offer a `Wildcard` description allowing anything.

There is a syntax description of syntax descriptions, which we use to check syntax descriptions.

Judgement forms as interaction protocols

We recast the notion of $\boxed{\text{judgement form}}$ as communication protocol:

- What to communicate (of what syntax description)?
- In which direction (input or output)?

Judgement forms as interaction protocols

We recast the notion of $\boxed{\text{judgement form}}$ as communication protocol:

- What to communicate (of what syntax description)?
- In which direction (input or output)?

A basic form of session types [Honda 1993].

Judgement forms as interaction protocols

We recast the notion of $\boxed{\text{judgement form}}$ as communication protocol:

- What to communicate (of what syntax description)?
- In which direction (input or output)?

A basic form of session types [Honda 1993].

For example:

```
type : ?'Type.
check : ?'Type. ?'Check.
synth : ?'Synth. !'Type.
```

Typing rules as actors

"A rule is a server for its conclusion, and a client for its premises."

Typing rules as actors

“A rule is a server for its conclusion, and a client for its premises.”

That is: typing rules are implemented by actors, which

- must fulfill their protocol with respect to their parent;
- typically spawns children processes for all its premises.

Typing rules as actors

"A rule is a server for its conclusion, and a client for its premises."

That is: typing rules are implemented by actors, which

- must fulfill their protocol with respect to their parent;
- typically spawns children processes for all its premises.

Inspired by the actor model [Hewitt, Bishop and Steiger 1973] of concurrent programming.

Typing rules as actors

"A rule is a server for its conclusion, and a client for its premises."

That is: typing rules are implemented by actors, which

- must fulfill their protocol with respect to their parent;
- typically spawns children processes for all its premises.

Inspired by the actor model [Hewitt, Bishop and Steiger 1973] of concurrent programming.

Typechecking process *actor* with parent channel p is defined by

```
actor@p = ...
```

Actor constructs: winning

␣

a successful, finished actor

(Victory is silent.)

Actor constructs: failing

`# "error message"`

an unsuccessful, finished actor

Actor constructs: printing

```
PRINTF "message text".
```

printing a message before continuing

Actor constructs: generating fresh meta variables

$$sd ? X \, .$$

generate a fresh meta X of syntax description sd

Actor constructs: generating fresh meta variables

$$sd?X\,.$$

generate a fresh meta X of syntax description sd

Meta variables stand for *unknown* terms.

Actor constructs: matching on terms

$$\texttt{case t} \left\{\ p_1\ \texttt{->}\ a_1\ \texttt{;}\ \texttt{...}\right\}$$

match term t against patterns p_i; continue as actor a_i when matching

Actor constructs: matching on terms

$$\texttt{case t } \{ \; p_1 \; \texttt{->} \; a_1 \; \texttt{;} \; \texttt{...} \}$$

match term t against patterns p_i; continue as actor a_i when matching

Blocks if t is a metavariable.

Actor constructs: forking

```
a | b
```

continue as `a` and `b` in parallel

Actor constructs: forking

```
a | b
```

continue as `a` and `b` in parallel

Progress in `b` might enable further progress in `a` and vice versa.

Actor constructs: declaring constraints

$$t_1 \sim t_2$$

make t_1 unify with t_2

Actor constructs: spawning children

$$actor@p.$$

spawn a new child *actor* on channel p

Actor constructs: sending and receiving messages

$$p\,!\,t.$$

send term t on channel p

Actor constructs: sending and receiving messages

$$p\,!\,t.$$

send term t on channel p

$$p?t.$$

receive term t on channel p

Actor constructs: sending and receiving messages

$$p\,!\,t.$$

send term t on channel p

$$p\,?\,t.$$

receive term t on channel p

Messages must conform to p's protocol.

Actor constructs: binding local variables

$$\backslash\, x.$$

bring fresh object variable x into scope

Actor constructs: extending local contexts

$$ctx \;\; |\text{-} \;\; x \;\; \text{->} \;\; t$$

extend declared context *ctx* to map object variable *x* to term *t*

Actor constructs: querying local contexts

$$\texttt{if}\ x\ \texttt{in}\ ctx\ \{\ t\ \texttt{->}\ a\ \}\ \texttt{else}\ b$$

Look up variable x in declared context ctx;
if found, bind associated value as t and continue as a,
otherwise continue as b

Actors for bidirectional type checking of STLC

```
check@p = p?ty. p?tm. case tm
  { ['Lam \x. body] -> 'Type?S. 'Type?T.
     ( ty ~ ['Arr S T]
     | \x. ctxt |- x -> S. check@q. q!T. q!body.)
  ; ['Emb e] -> synth@q. q!e. q?S. S ~ ty }

synth@p = p?tm. if tm in ctxt
  { S -> p!S. }
  else case tm
  { ['Ann t T] -> ( type@q. q!T.
                  | check@r. r!T. r!t.
                  | p!T. )
  ; ['App f s] -> 'Type?S. 'Type?T. p!T.
     ( synth@q. q!f. q?F. F ~ ['Arr S T]
     | check@r. r!S. r!s.) }
```

Executing actors

Executing actors

We currently run actors on a stack-based virtual machine.

Executing actors

We currently run actors on a stack-based virtual machine.

We run each actor until it blocks, and then try the next one, until execution stabilises.

Executing actors

We currently run actors on a stack-based virtual machine.

We run each actor until it blocks, and then try the next one, until execution stabilises.

Metavariables are shared, which is okay, since they are updated monotonically [Kuper 2015].

Executing actors

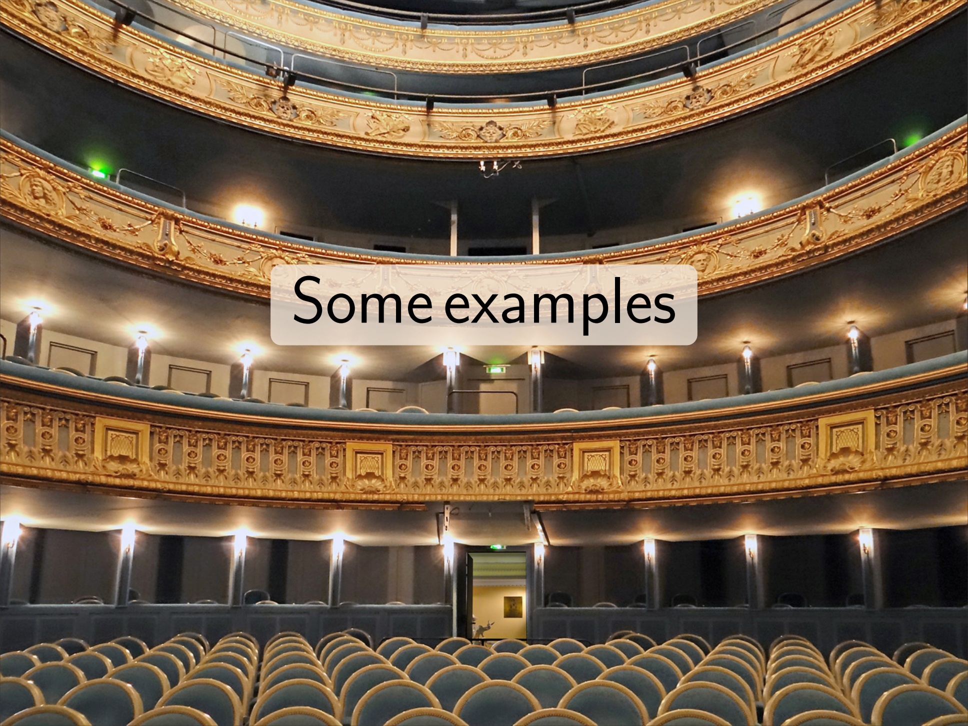
We currently run actors on a stack-based virtual machine.

We run each actor until it blocks, and then try the next one, until execution stabilises.

Metavariables are shared, which is okay, since they are updated monotonically [Kuper 2015].

We can extract a typing derivation from the final configuration of the stack.

Some examples



typos --latex=stlc.tex stlc.act

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathrm{TYPE}\ \mathbb{N} \to \mathbb{N}^{\checkmark}} \qquad \dfrac{\dfrac{\dfrac{}{\mathbb{N} \ni ?u}}{w_1 : \mathbb{N} \vdash}}{\mathbb{N} \to \mathbb{N} \ni \lambda\_.?u}}{(\lambda\_.?u : \mathbb{N} \to \mathbb{N}) \in \mathbb{N} \to \mathbb{N}} \qquad \dfrac{\dfrac{}{z_0 \in \mathbb{N}^{\checkmark}}}{\mathbb{N} \ni \underline{z_0}^{\checkmark}}}{(\lambda\_.?u : \mathbb{N} \to \mathbb{N})\underline{z_0} \in \mathbb{N}}}{\mathbb{N} \ni \underline{(\lambda\_.?u : \mathbb{N} \to \mathbb{N})\underline{z_0}}}}{z_0 : \mathbb{N} \vdash}}{\mathbb{N} \to \mathbb{N} \ni \lambda z.\underline{(\lambda\_.?u : \mathbb{N} \to \mathbb{N})\underline{z}}}
$$

`typos --latex=stlc.tex stlc.act` completed

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{\text{TYPE}\ \mathbb{N} \to \mathbb{N}^{\checkmark}} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\mathbb{N} \ni \mathsf{Zero}^{\checkmark}}}{\mathbb{N} \ni [\mathsf{Succ\ Zero}]^{\checkmark}}}{w_1 : \mathbb{N} \vdash}}{\mathbb{N} \to \mathbb{N} \ni \lambda\_.[\mathsf{Succ\ Zero}]^{\checkmark}}}{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N}) \in \mathbb{N} \to \mathbb{N}^{\checkmark}} \quad \dfrac{\dfrac{}{z_0 \in \mathbb{N}^{\checkmark}}}{\mathbb{N} \ni \underline{z_0}^{\checkmark}}}{\dfrac{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0} \in \mathbb{N}^{\checkmark}}{\dfrac{\mathbb{N} \ni \underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0}}^{\checkmark}}{z_0 : \mathbb{N} \vdash}}}}{\mathbb{N} \to \mathbb{N} \ni \lambda z.\underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z}}^{\checkmark}}
$$

```
typos --latex-animated=stlc-ann.tex stlc.act
```

$$\frac{}{\mathbb{N} \to \mathbb{N} \ni \lambda z.\underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z}}}$$

```
typos --latex-animated=stlc-ann.tex stlc.act
```

$$\dfrac{\dfrac{}{z_0 : \quad \vdash}}{\mathbb{N} \to \mathbb{N} \ni \lambda z.\underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z}}}$$

```
typos --latex-animated=stlc-ann.tex stlc.act
```

$$\dfrac{\dfrac{}{\mathbb{N} \ni}}{z_0 : \mathbb{N} \vdash}$$

$$\mathbb{N} \to \mathbb{N} \ni \lambda z.\underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z}}$$

typos --latex-animated=stlc-ann.tex stlc.act

$$\dfrac{\dfrac{}{\mathbb{N} \ni \underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0}}}}{\dfrac{z_0 : \mathbb{N} \vdash}{\mathbb{N} \to \mathbb{N} \ni \lambda z.\underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z}}}}$$

```
typos --latex-animated=stlc-ann.tex stlc.act
```

$$\dfrac{\dfrac{\dfrac{\dfrac{}{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0} \in}}{\mathbb{N} \ni \underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0}}}}{z_0 : \mathbb{N} \vdash}}{\mathbb{N} \to \mathbb{N} \ni \lambda z.\underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z}}}$$

`typos --latex-animated=stlc-ann.tex stlc.act`

$$\dfrac{\dfrac{\dfrac{}{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0} \in ???}}{\mathbb{N} \ni \underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0}}}}{z_0 : \mathbb{N} \vdash}}{\mathbb{N} \to \mathbb{N} \ni \lambda z.\underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z}}}$$

`typos --latex-animated=stlc-ann.tex stlc.act`

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N}) \in}}{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0} \in ???}}{\mathbb{N} \ni \underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0}}}}{z_0 : \mathbb{N} \vdash}}{\mathbb{N} \to \mathbb{N} \ni \lambda z.\underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z}}}$$

typos --latex-animated=stlc-ann.tex stlc.act

$$\dfrac{\dfrac{\dfrac{\overline{\textsc{type}\ \mathbb{N} \to \mathbb{N}}}{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N}) \in} \qquad \dfrac{}{}}{\dfrac{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0} \in ???}{\mathbb{N} \ni \underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0}}}}}{\dfrac{z_0 : \mathbb{N} \vdash}{\mathbb{N} \to \mathbb{N} \ni \lambda z.\underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z}}}}$$

```
typos --latex-animated=stlc-ann.tex stlc.act
```

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{\textsc{type}\ \mathbb{N} \to \mathbb{N}}}{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N}) \in}}{\dfrac{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0} \in \textcolor{red}{???}}{\mathbb{N} \ni \underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0}}}}}{z_0 : \mathbb{N} \vdash}}{\mathbb{N} \to \mathbb{N} \ni \lambda z.\underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z}}}
$$

typos --latex-animated=stlc-ann.tex stlc.act

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{\textsc{type}\ \mathbb{N} \to \mathbb{N}}}{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N}) \in}}{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0} \in ???}}{\dfrac{\mathbb{N} \ni \underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0}}}{z_0 : \mathbb{N} \vdash}}
$$

$$
\mathbb{N} \to \mathbb{N} \ni \lambda z.\underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z}}
$$

`typos --latex-animated=stlc-ann.tex stlc.act`

$$
\dfrac{
  \dfrac{
    \dfrac{
      \dfrac{}{\textsc{type}\ \mathbb{N} \to \mathbb{N}^{\checkmark}}
      \quad
      \dfrac{}{\mathbb{N} \to \mathbb{N} \ni}
    }{
      (\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N}) \in
    }
  }{
    \dfrac{
      (\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0} \in \text{???}
    }{
      \mathbb{N} \ni \underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0}}
    }
  }
  \qquad
}{
  \dfrac{z_0 : \mathbb{N} \vdash}{\mathbb{N} \to \mathbb{N} \ni \lambda z.\underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z}}}
}
$$

```
typos --latex-animated=stlc-ann.tex stlc.act
```

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\textsc{type}\ \mathbb{N}\to\mathbb{N}^{\checkmark}} \qquad \dfrac{}{\mathbb{N}\to\mathbb{N}\ni\lambda\_.[\mathsf{Succ\ Zero}]}}{(\lambda\_.[\mathsf{Succ\ Zero}]:\mathbb{N}\to\mathbb{N})\in}}{(\lambda\_.[\mathsf{Succ\ Zero}]:\mathbb{N}\to\mathbb{N})\underline{z_0}\in ???}}{\mathbb{N}\ni\underline{(\lambda\_.[\mathsf{Succ\ Zero}]:\mathbb{N}\to\mathbb{N})\underline{z_0}}}}{z_0:\mathbb{N}\vdash}
$$

$$
\mathbb{N}\to\mathbb{N}\ni\lambda z.\underline{(\lambda\_.[\mathsf{Succ\ Zero}]:\mathbb{N}\to\mathbb{N})\underline{z}}
$$

```
typos --latex-animated=stlc-ann.tex stlc.act
```

$$
\dfrac{
  \dfrac{
    \dfrac{
      \dfrac{
        \dfrac{
          \dfrac{}{\textsc{type}\ \mathbb{N} \to \mathbb{N}^{\checkmark}}
          \qquad
          \dfrac{\dfrac{}{w_1 : \quad \vdash}}{\mathbb{N} \to \mathbb{N} \ni \lambda\_.[\mathsf{Succ\ Zero}]}
        }{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N}) \in}
      }{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0} \in ???}
    }{\mathbb{N} \ni \underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0}}}
  }{z_0 : \mathbb{N} \vdash}
}{\mathbb{N} \to \mathbb{N} \ni \lambda z.\underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z}}}
$$

typos --latex-animated=stlc-ann.tex stlc.act

$$
\dfrac{
  \dfrac{
    \dfrac{
      \dfrac{
        \dfrac{}{\textsc{type}\ \mathbb{N} \to \mathbb{N}^{\checkmark}}
        \qquad
        \dfrac{
          \dfrac{}{\mathbb{N} \ni}
          \atop
          w_1 : \mathbb{N} \vdash
        }{\mathbb{N} \to \mathbb{N} \ni \lambda\_.[\mathsf{Succ\ Zero}]}
      }{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N}) \in}
    }{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0} \in ???}
  }{\mathbb{N} \ni \underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0}}}
  \atop
  z_0 : \mathbb{N} \vdash
}{\mathbb{N} \to \mathbb{N} \ni \lambda z.\underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z}}}
$$

```
typos --latex-animated=stlc-ann.tex stlc.act
```

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\overline{\textsc{type}\ \mathbb{N} \to \mathbb{N}^{\checkmark}} \qquad \cfrac{\cfrac{\overline{\mathbb{N} \ni [\mathsf{Succ\ Zero}]}}{w_1 : \mathbb{N} \vdash}}{\mathbb{N} \to \mathbb{N} \ni \lambda\_.[\mathsf{Succ\ Zero}]}}{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N}) \in}}{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0} \in ???}}{\mathbb{N} \ni \underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0}}}}{z_0 : \mathbb{N} \vdash}}{\mathbb{N} \to \mathbb{N} \ni \lambda z.\underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z}}}$$

typos --latex-animated=stlc-ann.tex stlc.act

$$
\dfrac{
  \dfrac{
    \dfrac{
      \dfrac{
        \dfrac{
          \dfrac{}{\mathrm{TYPE}\ \mathbb{N}\to\mathbb{N}^{\checkmark}}
          \qquad
          \dfrac{
            \dfrac{
              \dfrac{
                \dfrac{}{\mathbb{N}\ni}
              }{\mathbb{N}\ni[\mathsf{Succ\ Zero}]}
            }{w_1 : \mathbb{N}\vdash}
          }{\mathbb{N}\to\mathbb{N}\ni\lambda\_.[\mathsf{Succ\ Zero}]}
        }{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N}\to\mathbb{N})\in}
      }{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N}\to\mathbb{N})\underline{z_0}\in {???}}
    }{\mathbb{N}\ni\underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N}\to\mathbb{N})\underline{z_0}}}
  }{z_0 : \mathbb{N}\vdash}
}{\mathbb{N}\to\mathbb{N}\ni\lambda z.\underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N}\to\mathbb{N})\underline{z}}}
$$

`typos --latex-animated=stlc-ann.tex stlc.act`

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathrm{TYPE}\ \mathbb{N} \to \mathbb{N}^{\checkmark}} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\mathbb{N} \ni \mathsf{Zero}}}{\mathbb{N} \ni [\mathsf{Succ\ Zero}]}}{w_1 : \mathbb{N} \vdash}}{\mathbb{N} \to \mathbb{N} \ni \lambda\_.[\mathsf{Succ\ Zero}]}}{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N}) \in}}{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0} \in ???}}{\dfrac{\mathbb{N} \ni \underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0}}}{z_0 : \mathbb{N} \vdash}}}{\mathbb{N} \to \mathbb{N} \ni \lambda z.\underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z}}}
$$

```
typos --latex-animated=stlc-ann.tex stlc.act
```

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\textsc{type}\ \mathbb{N} \to \mathbb{N}^{\checkmark}} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\mathbb{N} \ni \mathsf{Zero}^{\checkmark}}}{\mathbb{N} \ni [\mathsf{Succ\ Zero}]^{\checkmark}}}{w_1 : \mathbb{N} \vdash}}{\mathbb{N} \to \mathbb{N} \ni \lambda\_.[\mathsf{Succ\ Zero}]^{\checkmark}}}{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N}) \in \mathbb{N} \to \mathbb{N}}}{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0} \in ???}}{\mathbb{N} \ni \underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0}}}}{\dfrac{z_0 : \mathbb{N} \vdash}{\mathbb{N} \to \mathbb{N} \ni \lambda z.\underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z}}}}
$$

`typos --latex-animated=stlc-ann.tex stlc.act`

$$
\frac{
  \frac{
    \frac{}{\mathrm{TYPE}\ \mathbb{N} \to \mathbb{N}^{\checkmark}}
    \qquad
    \frac{
      \frac{
        \frac{
          \frac{}{\mathbb{N} \ni \mathsf{Zero}^{\checkmark}}
        }{\mathbb{N} \ni [\mathsf{Succ}\ \mathsf{Zero}]^{\checkmark}}
      }{w_1 : \mathbb{N} \vdash}
    }{\mathbb{N} \to \mathbb{N} \ni \lambda\_.[\mathsf{Succ}\ \mathsf{Zero}]^{\checkmark}}
  }{(\lambda\_.[\mathsf{Succ}\ \mathsf{Zero}] : \mathbb{N} \to \mathbb{N}) \in \mathbb{N} \to \mathbb{N}^{\checkmark}}
  \qquad
  \frac{}{\mathbb{N} \ni}
}{
  \frac{
    \frac{(\lambda\_.[\mathsf{Succ}\ \mathsf{Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0} \in \mathbb{N}}{\mathbb{N} \ni \underline{(\lambda\_.[\mathsf{Succ}\ \mathsf{Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0}}}
  }{z_0 : \mathbb{N} \vdash}
}
$$

$$
\overline{\mathbb{N} \to \mathbb{N} \ni \lambda z.\underline{(\lambda\_.[\mathsf{Succ}\ \mathsf{Zero}] : \mathbb{N} \to \mathbb{N})\underline{z}}}
$$

typos --latex-animated=stlc-ann.tex stlc.act

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathrm{TYPE}\ \mathbb{N} \to \mathbb{N}^{\checkmark}} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\mathbb{N} \ni \mathsf{Zero}^{\checkmark}}}{\mathbb{N} \ni [\mathsf{Succ\ Zero}]^{\checkmark}}}{w_1 : \mathbb{N} \vdash}}{\mathbb{N} \to \mathbb{N} \ni \lambda\_.[\mathsf{Succ\ Zero}]^{\checkmark}}}{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N}) \in \mathbb{N} \to \mathbb{N}^{\checkmark}} \qquad \dfrac{}{\mathbb{N} \ni \underline{z_0}}}{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0} \in \mathbb{N}}}{\mathbb{N} \ni \underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0}}}}{z_0 : \mathbb{N} \vdash}}{\mathbb{N} \to \mathbb{N} \ni \lambda z.\underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z}}}
$$

```
typos --latex-animated=stlc-ann.tex stlc.act
```

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{\mathrm{TYPE}\ \mathbb{N} \to \mathbb{N}^{\checkmark}} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\mathbb{N} \ni \mathsf{Zero}^{\checkmark}}}{\mathbb{N} \ni [\mathsf{Succ\ Zero}]^{\checkmark}}}{w_1 : \mathbb{N} \vdash}}{\mathbb{N} \to \mathbb{N} \ni \lambda\_.[\mathsf{Succ\ Zero}]^{\checkmark}}}{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N}) \in \mathbb{N} \to \mathbb{N}^{\checkmark}} \qquad \dfrac{\dfrac{}{z_0 \in}}{\mathbb{N} \ni \underline{z_0}}}{\dfrac{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0} \in \mathbb{N}}{\mathbb{N} \ni \underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0}}}}}{\dfrac{z_0 : \mathbb{N} \vdash}{\mathbb{N} \to \mathbb{N} \ni \lambda z.\underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z}}}}
$$

`typos --latex-animated=stlc-ann.tex stlc.act`

$$
\dfrac{
  \dfrac{
    \dfrac{}{\textsc{type}\ \mathbb{N} \to \mathbb{N}^{\checkmark}}
    \qquad
    \dfrac{
      \begin{array}{c}
        \dfrac{\dfrac{}{\mathbb{N} \ni \mathsf{Zero}^{\checkmark}}}{\mathbb{N} \ni [\mathsf{Succ\ Zero}]^{\checkmark}} \\ \hline
        w_1 : \mathbb{N} \vdash
      \end{array}
    }{\mathbb{N} \to \mathbb{N} \ni \lambda\_.[\mathsf{Succ\ Zero}]^{\checkmark}}
  }{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N}) \in \mathbb{N} \to \mathbb{N}^{\checkmark}}
  \qquad
  \dfrac{\dfrac{}{z_0 \in \mathbb{N}}}{\mathbb{N} \ni \underline{z_0}}
}{
  \begin{array}{c}
    \dfrac{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0} \in \mathbb{N}}{\mathbb{N} \ni \underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0}}} \\ \hline
    z_0 : \mathbb{N} \vdash \\ \hline
    \mathbb{N} \to \mathbb{N} \ni \lambda z.\underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z}}
  \end{array}
}
$$

typos --latex-animated=stlc-ann.tex stlc.act

$$
\dfrac{
  \dfrac{
    \dfrac{}{\mathrm{TYPE}\ \mathbb{N} \to \mathbb{N}^{\checkmark}}
    \qquad
    \dfrac{
      \begin{array}{c}
      \dfrac{
        \dfrac{}{\mathbb{N} \ni \mathsf{Zero}^{\checkmark}}
      }{\mathbb{N} \ni [\mathsf{Succ\ Zero}]^{\checkmark}} \\
      w_1 : \mathbb{N} \vdash
      \end{array}
    }{\mathbb{N} \to \mathbb{N} \ni \lambda\_.[\mathsf{Succ\ Zero}]^{\checkmark}}
  }{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N}) \in \mathbb{N} \to \mathbb{N}^{\checkmark}}
  \qquad
  \dfrac{
    \dfrac{}{z_0 \in \mathbb{N}^{\checkmark}}
  }{\mathbb{N} \ni \underline{z_0}^{\checkmark}}
}{
  \begin{array}{c}
  \dfrac{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0} \in \mathbb{N}^{\checkmark}}{\mathbb{N} \ni \underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z_0}}^{\checkmark}} \\
  z_0 : \mathbb{N} \vdash
  \end{array}
}
$$

$$
\overline{\mathbb{N} \to \mathbb{N} \ni \lambda z.\underline{(\lambda\_.[\mathsf{Succ\ Zero}] : \mathbb{N} \to \mathbb{N})\underline{z}}^{\checkmark}}
$$

Verification of actors

What do we get by construction?

Verification of actors

What do we get by construction?

- Protocols and modes $\Longrightarrow$ rely/guarantee contracts

Verification of actors

What do we get by construction?

- Protocols and modes $\implies$ rely/guarantee contracts
- Actors only knowing about free variables they themselves create $\implies$ stability under substitution

Verification of actors

What do we get by construction?

- Protocols and modes $\implies$ rely/guarantee contracts
- Actors only knowing about free variables they themselves create $\implies$ stability under substitution
- "Schematic variables" have one explicit binding site $\implies$ scopes are not escaped

Verification of actors

What do we get by construction?

- Protocols and modes $\Longrightarrow$ rely/guarantee contracts
- Actors only knowing about free variables they themselves create $\Longrightarrow$ stability under substitution
- "Schematic variables" have one explicit binding site $\Longrightarrow$ scopes are not escaped
- ...

Summary and future work

TypOS is an domain-specific language for writing typecheckers.

Judgements have modes (input/output protocols), typing rules are actors (spawning and communicating with children).

A wide range of typechecking, evaluation and elaboration processes can be implemented this way.

In the future: a truly concurrent runtime.

`https://github.com/msp-strath/TypOS`

Summary and future work

TypOS is an domain-specific language for writing typecheckers.

Judgements have modes (input/output protocols), typing rules are actors (spawning and communicating with children).

A wide range of ty[illegible]oration processes can be implemented this way.

Thank you!

In the future: a truly concurrent runtime.

`https://github.com/msp-strath/TypOS`

References

In order of appearance

- Andrej Bauer, Philipp G. Haselwarter, and Anja Petkovic. Equality checking for general type theories in Andromeda 2. In Anna Maria Bigatti, Jacques Carette, James H. Davenport, Michael Joswig, and Timo de Wolff, editors, *ICMS 20*, pages 253–259. Springer, 2020.
- Matthias Felleisen, Robert Bruce Findler, and Matthew Flatt. *Semantics Engineering with PLT Redex*. MIT Press, 2009.
- Stephen Chang, Michael Ballantyne, Milo Turner, and William J. Bowman. Dependent type systems as macros. *Proc. ACM Program. Lang.*, 4(POPL):3:1–3:29, 2020.
- Kohei Honda. Types for dyadic interaction. In Eike Best, editor, *CONCUR 93*, pages 509–523. Springer, 1993.
- Carl Hewitt, Peter Bishop, and Richard Steiger. A universal modular actor formalism for artificial intelligence. In *IJCAI 73*, pages 235–245. Morgan Kaufmann Publishers, 1973.
- Lindsey Kuper. *Lattice-Based Data Structures For Deterministic Parallel And Distributed Programming*. PhD thesis, Indiana University, 2015.

Image credits:

- "Shakespeare's Globe Theatre, London" by Neil Howard, `https://flic.kr/p/LtqfmA`, CC BY-NC 2.0
- "L'Opéra Graslin (Le Voyage à Nantes)" by Jean-Pierre Dalbra, `https://flic.kr/p/f9BB5h`, CC BY 2.0