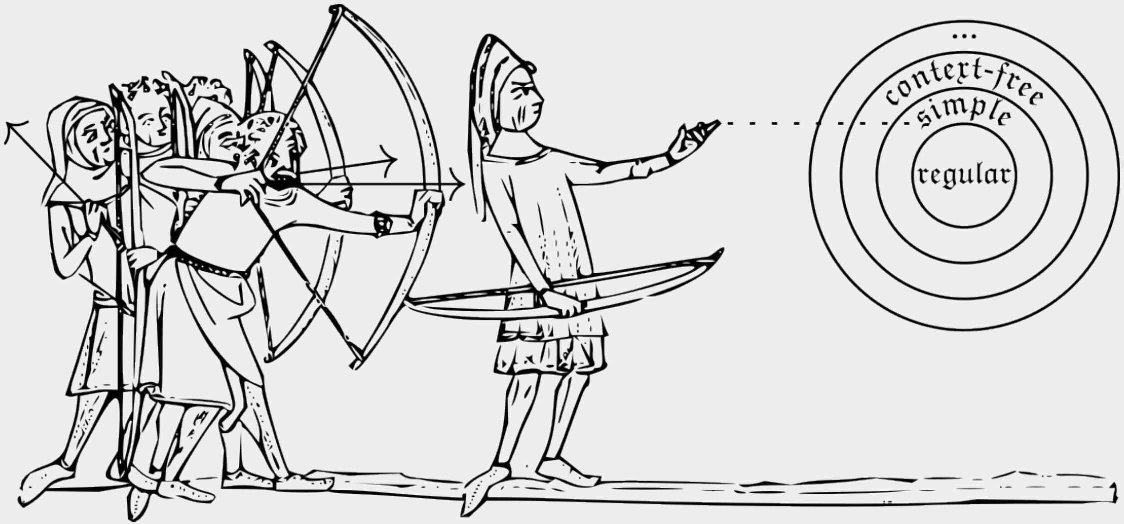
Types as grammars

Gil Silva
Bernardo Almeida
Diana Costa
Andreia Mordido
Diogo Poças
Vasco T. Vasconcelos

University of Lisbon

WITS @ POPL 2026
17 January 2026



Motivation — FreeST (Almeida *et al.* 2022)

FreeST is a functional programming language with **message-passing concurrency** governed by **session types**.

```
main : ()
main =
  let (i, o) = channel @(?String; Wait) ()
  in fork (\_ -> send “hello, world!” o |> close);
     recv i \s -> putStrLn s; wait
```

Motivation — session types (Honda *et al.* 1998)

Session types are a type discipline for safe message-passing concurrency.
They describe communication protocols on heterogeneous, bidirectional channels.

$T, U ::=$	$!T.U$	*send and continue*
$\mid$	$?T.U$	*receive and continue*
$\mid$	$!\{\overline{\ell : T_\ell}\}$	*choose a branch*
$\mid$	$?\{\overline{\ell : T_\ell}\}$	*offer choices*
$\mid$	Close	*close channel*
$\mid$	Wait	*wait for closing*
$\mid$	α	*recursion*
$\mid$	$\mu\alpha.T$	

math client

$$\mu\alpha.!\{\mathsf{Add} : !\mathsf{Int}.!\mathsf{Int}.?\mathsf{Int}.\alpha \\ , \mathsf{IsPrime} : !\mathsf{Int}.?\mathsf{Bool}.\alpha \\ , \mathsf{Exit} : \mathsf{Close}\}$$

math server

$$\mu\alpha.?\{\mathsf{Add} : ?\mathsf{Int}.?\mathsf{Int}.!\mathsf{Int}.\alpha \\ , \mathsf{IsPrime} : ?\mathsf{Int}.!\mathsf{Bool}.\alpha \\ , \mathsf{Exit} : \mathsf{Wait}\}$$

Motivation — context-free session types

(Thiemann & Vasconcelos 2016

Context-free session types uncouple messages from sequential composition.

regular

$$T, U ::= \,!T.U \mid ?T.U \mid \ldots$$

context-free

$T, U ::=$	$!T$	*send*
$\mid$	$?T$	*receive*
$\mid$	$T;U$	*sequential composition*
$\mid$	Skip	*no-op*
	$\ldots$	

e.g., tree serialization

$$\mu\alpha.\oplus\{\mathsf{Node} : !\mathsf{Int}; \alpha; \alpha \\ , \mathsf{Empty} : \mathsf{Skip}\}$$

Motivation — type operators

FreeST features polymorphism, equi-recursive types and *context-free* session types.
We are adding **type operators** to it.

- $\lambda_\to$ — *Simple types*
- F — *Polymorphism*
- F^μ — *Equi-recursive types*
- $F^{\mu;}$ — ! *Context-free session types*
- $F^{\mu;}_\omega$ — *Type operators*

$$T, U ::= \dots \mid \lambda\alpha{:}\kappa.T \mid T\,U$$

```
data Tree a = Empty
            | Node a (Tree a) (Tree a)

type SendTree a =
  !{ Empty: Skip
   , Node : !a; SendTree; SendTree
   }
```

Problem — type equivalence

"Are two types interchangeable under any context?"

Any typechecker needs a way to answer this question

$\lambda_{\rightarrow}$

Simple types $T, U ::= \mathsf{Int} \mid T \rightarrow U$

easy — syntactical equality

Problem — type equivalence

"Are two types interchangeable under any context?"

Any typechecker needs a way to answer this question

$\lambda_{\to}$ *Simple types*

F *Polymorphism* $T, U ::= \ldots \mid \alpha \mid \forall\alpha.T$

bound variable names are not important — α-equivalence

$$\forall\alpha.\alpha \to \alpha \simeq \forall\beta.\beta \to \beta$$

Problem — type equivalence

"Are two types interchangeable under any context?"

Any typechecker needs a way to answer this question

$\lambda_{\rightarrow}$ *Simple types*

F *Polymorphism*

F^{μ} *Equi-recursive types*

$$T, U ::= \ldots \mid \mu\alpha.T$$

μ-types are equivalent to their unfolding

$$\mu\alpha.\mathsf{Int} \rightarrow \alpha \simeq \mathsf{Int} \rightarrow (\mu\beta.\mathsf{Int} \rightarrow \beta)$$

Problem — type equivalence

"Are two types interchangeable under any context?"

Any typechecker needs a way to answer this question

- $\lambda_{\to}$ — *Simple types*
- F — *Polymorphism*
- F^{μ} — *Equi-recursive types*
- $F^{\mu;}$ — *Context-free session types*

$$T, U ::= \dots \mid \sharp T \mid \sharp\{\overline{\ell : T_\ell}\} \mid \mathsf{End}^{\sharp} \mid T; U \mid \mathsf{Skip}$$

$$\sharp ::= \; ! \mid ?$$

$\mathsf{Close} \mid \mathsf{Wait}$

Problem — type equivalence

"Are two types interchangeable under any context?"

Any typechecker needs a way to answer this question

$\lambda_{\rightarrow}$	*Simple types*
F	*Polymorphism*
F^{μ}	*Equi-recursive types*
$F^{\mu;}$	*Context-free session types*

Identity	$\mathsf{Skip}; T \simeq T \simeq T; \mathsf{Skip}$
Associativity	$T_1; (T_2; T_3) \simeq (T_1; T_2); T_3$
Distributivity	$\sharp\{\overline{\ell : T_\ell}\}; U \simeq \sharp\{\overline{\ell : T_\ell; U}\}$
Absorption	$\mathsf{End}^{\sharp}; T \simeq \mathsf{End}^{\sharp}$

equivalence preserves the algebraic laws of (;)

(Problem — context-free session type equivalence)

E-Int
$$\mathsf{Int} \simeq \mathsf{Int}$$

E-Arrow
$$\frac{T_1 \simeq U_1 \qquad T_2 \simeq U_2}{T_1 \to T_2 \simeq U_1 \to U_2}$$

E-RecL
$$\frac{[\mu\alpha.T/\alpha]T \simeq U}{\mu\alpha.T \simeq}$$

E-RecR
$$\frac{T \simeq [\mu\alpha.U/\alpha]U}{T \simeq \mu x.U}$$

E-Msg
$$\frac{T \simeq U}{\sharp T \simeq \sharp U}$$

E-Choice
$$\frac{\overline{T_\ell \simeq U_\ell}}{\sharp\{\overline{\ell\colon T_\ell}\} \simeq \sharp\{\overline{\ell\colon U_\ell}\}}$$

S-End
$$\mathsf{End}^\sharp \simeq \mathsf{End}^\sharp$$

S-Skip
$$\mathsf{Skip} \simeq \mathsf{Skip}$$

E-Ms

$T \simeq$

gSeqR
$$\frac{U' \simeq \mathsf{Skip}}{\simeq \sharp U;U'}$$

E-MsgSeq
$$\frac{T \simeq U \qquad T' \simeq U'}{\sharp T;T' \simeq \sharp U;U'}$$

E-ChoiceSeqL
$$\frac{\sharp\{\overline{\ell\colon T_\ell;T}\} \simeq U}{\sharp\{\overline{\ell\colon T_\ell}\};T \simeq U}$$

E-ChoiceSeqR
$$\frac{T \simeq \sharp\{\overline{\ell\colon U_\ell;U}\}}{T \simeq \sharp\{\overline{\ell\colon U_\ell}\};U}$$

E-EndSeq
$$\mathsf{End}^\sharp;T \simeq \mathsf{End}^\sharp;U$$

E-SkipSeqL
$$\frac{T \simeq U}{\mathsf{Skip};T \simeq U}$$

E-SkipSeqR
$$\frac{T \simeq U}{T \simeq \mathsf{Skip};U}$$

E-SeqSeqL
$$\frac{T_1;(T_2;T_3) \simeq U}{(T_1;T_2);T_3 \simeq U}$$

E-SeqS
$$\frac{T \simeq U_1;(U_2;U_3)}{T \simeq (U_1;U_2);U_3}$$

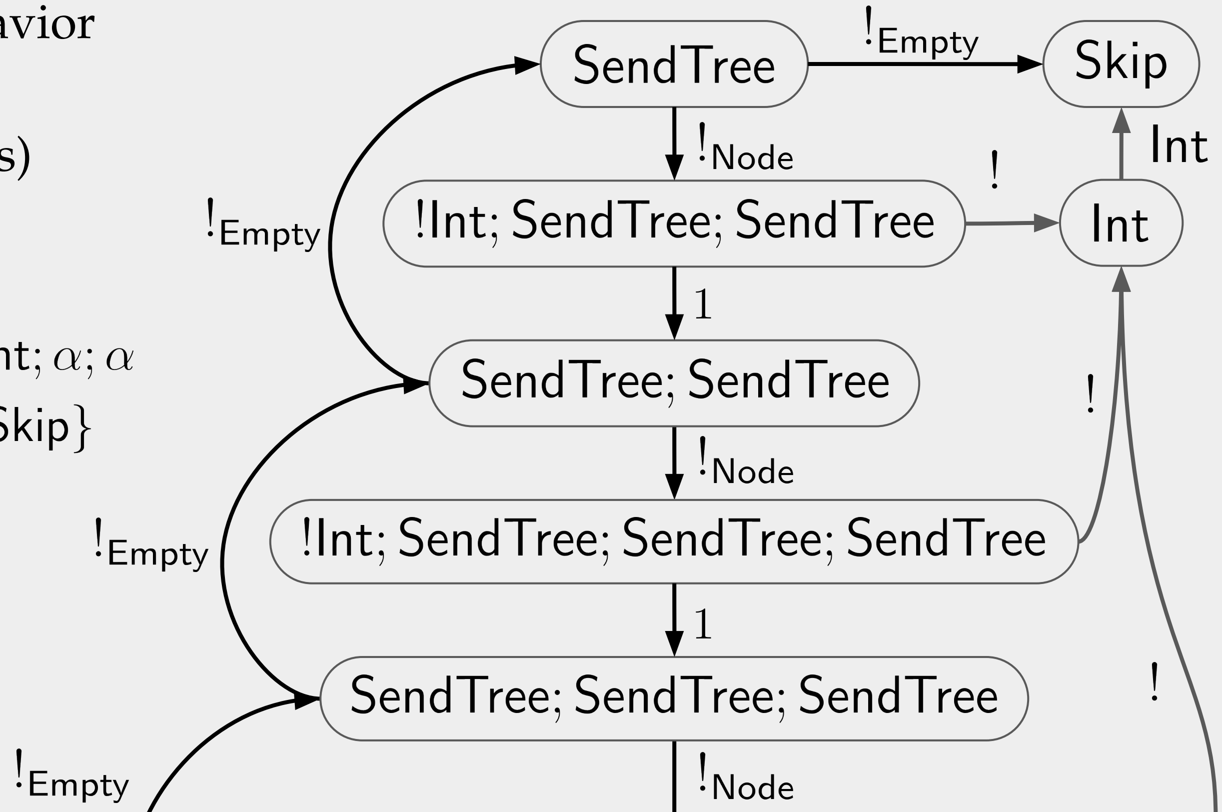
ecSeqL
$$\frac{([\mu\alpha.T_1/\alpha]T_1);T_2 \simeq U}{(\mu\alpha.T_1);T_2 \simeq U}$$

E-RecSeqR
$$\frac{T \simeq ([\mu\alpha.U_1/\alpha]U_1);U_2}{T \simeq (\mu\alpha.U_1);U_2}$$

(Solution — behavioral equivalence)

Forget syntax, look at behavior
Define an LTS for types
(Including functional types)

$$\mathsf{SendTree} = \mu\alpha.!\{\mathsf{Node} : !\mathsf{Int}; \alpha; \alpha \\ , \mathsf{Empty} : \mathsf{Skip}\}$$



(Solution — type equivalence as bisimilarity)

Definition. A type relation $\mathcal{R}$ is said to be a *bisimulation* if, whenever $T \mathrel{\mathcal{R}} U$ and for each a, we have:

1. If $T \xrightarrow{a} T'$ then there is U' such that $U \xrightarrow{a} U'$ with $T' \mathrel{\mathcal{R}} U'$;
2. If $U \xrightarrow{a} U'$ then there is T' such that $T \xrightarrow{a} T'$ with $T' \mathrel{\mathcal{R}} U'$.

Types T and U are said to be *bisimilar* $(T \sim U)$ if there is a weak bisimulation $\mathcal{R}$ such that $T \mathrel{\mathcal{R}} U$.

(Solution — decidability)

$Z \to a\overline{Z}$

By reduction to bisimilarity of BPA processes

=

words in context-free grammars (in Greibach normal form)

$T \overset{?}{\sim} U$

$\mathsf{word}(T) = \overline{X}$

$\mathsf{word}(U) = \overline{Y}$

$\mathcal{P} = \overline{Z \to a\overline{Z}}$

$\overline{X} \overset{?}{\sim}_{\mathcal{P}} \overline{Y}$

LTS: $\overline{Z\,\overline{W} \xrightarrow{a}_{\mathcal{P}} \overline{Z}\,\overline{W}}$

Lemma (*full abstraction*). The LTS of a type and its word coincide.

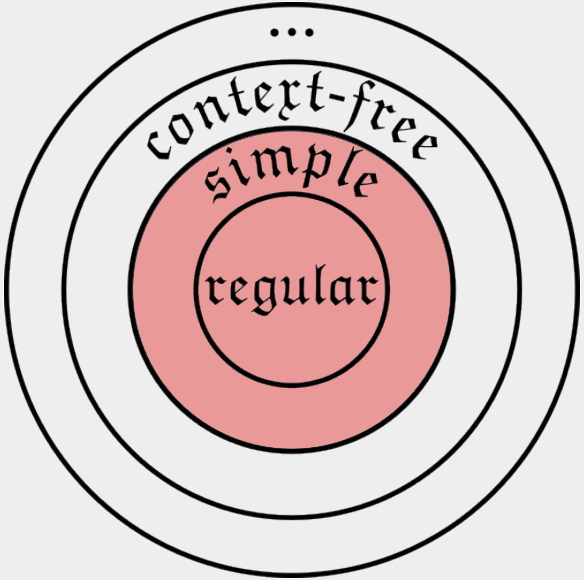
(Solution — algorithms)

No practical algorithm for general BPA/CFG bisimilarity.

But context-free session types are deterministic
I.e., they correspond to deterministic CFG in GNF
i.e., **simple grammars**.

Over the years, our team has developed 2 algorithms for simple grammar bisimilarity:

- **Almeida *et al.*, 2020**: doubly-exponential time*
- **Poças *et al.*, 2025**: polynomial time*



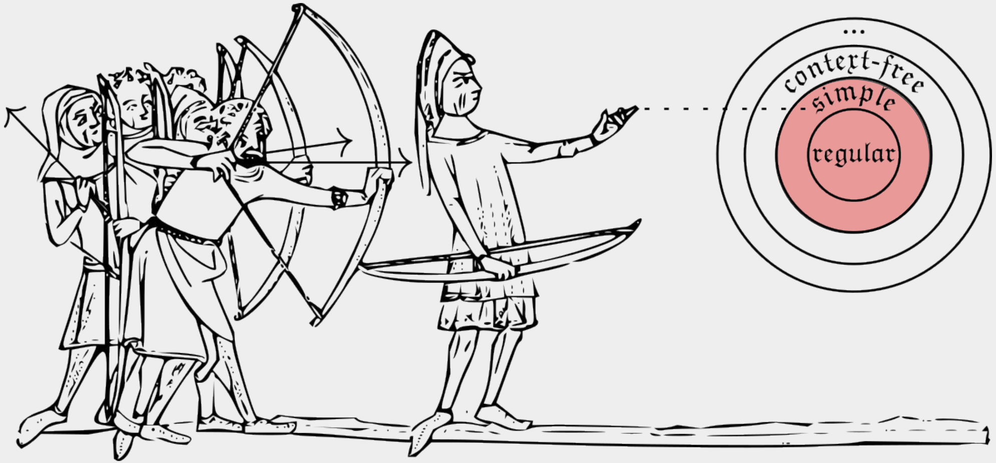
Problem — type equivalence

$\lambda_{\rightarrow}$ — *Simple types*

F — *Polymorphism*

F^{μ} — *Equi-recursive types*

$F^{\mu;}$ — *Context-free session types*



Problem — type equivalence

$\lambda_\rightarrow$ *Simple types*

$(\lambda\alpha.\alpha)\,\mathsf{Int} \simeq \mathsf{Int}$

F *Polymorphism*

F^μ *Equi-recursive types*

$\lambda\alpha.\alpha \simeq \lambda\beta.(\lambda\alpha.\alpha)\,\beta$

$F^{\mu;}$ *Context-free session types*

$F^{\mu;}_\omega$ *Type operators*

$$T, U \ ::= \ \ldots \mid \lambda\alpha{:}\kappa.T \mid T\,U$$

equivalence preserves type reduction — β/η-equivalence

Problem — type equivalence

$\lambda_{\rightarrow}$ — *Simple types*

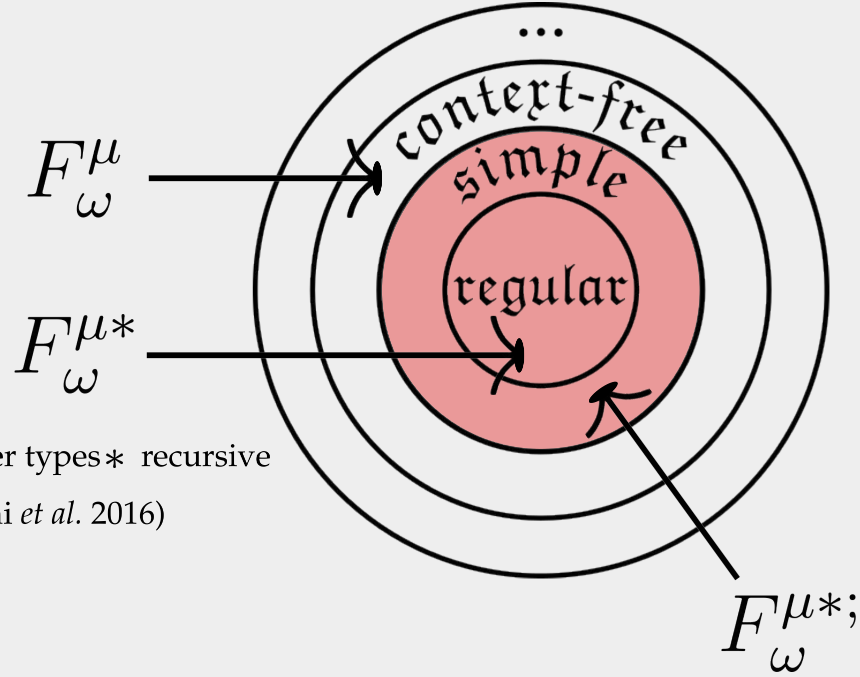
F — *Polymorphism*

F^{μ} — *Equi-recursive types*

$F^{\mu;}$ — *Context-free session types*

$F^{\mu*;}_{\omega}$ — *Type operators* $T, U ::= \ldots \mid \lambda\alpha{:}\kappa.T \mid T\,U$

equivalence preserves type reduction — β/η-equivalence



only proper types $*$ recursive

(Cai *et al.* 2016)

Problem — type equivalence

$\lambda_\to$ — *Simple types*

F — *Polymorphism*

F^μ — *Equi-recursive types*

$F^{\mu;}$ — *Context-free session types*

$F_\omega^{\mu*;}$ — *Type operators* $T, U ::= \dots \mid \lambda\alpha{:}\kappa.T \mid T\,U$

equivalence preserves type reduction — β/η-equivalence

$$T \overset{?}{\sim} U \longrightarrow \boxed{\begin{array}{l} \mathsf{word}(T) = \overline{X} \\ \mathsf{word}(U) = \overline{Y} \\ \mathcal{P} = \overline{Z \to a\overline{Z}} \end{array}} \longrightarrow \overline{X} \overset{?}{\sim}_{\mathcal{P}} \overline{Y}$$

Solution — silent actions

$$T \xrightarrow{\tau} U$$

silent actions
call-by-name reduction
unfolding
session type normalization

$$T \xrightarrow{a} U$$

observable actions
actual structure (functional types)
actual behavior (session types)

Solution — weak bisimilarity

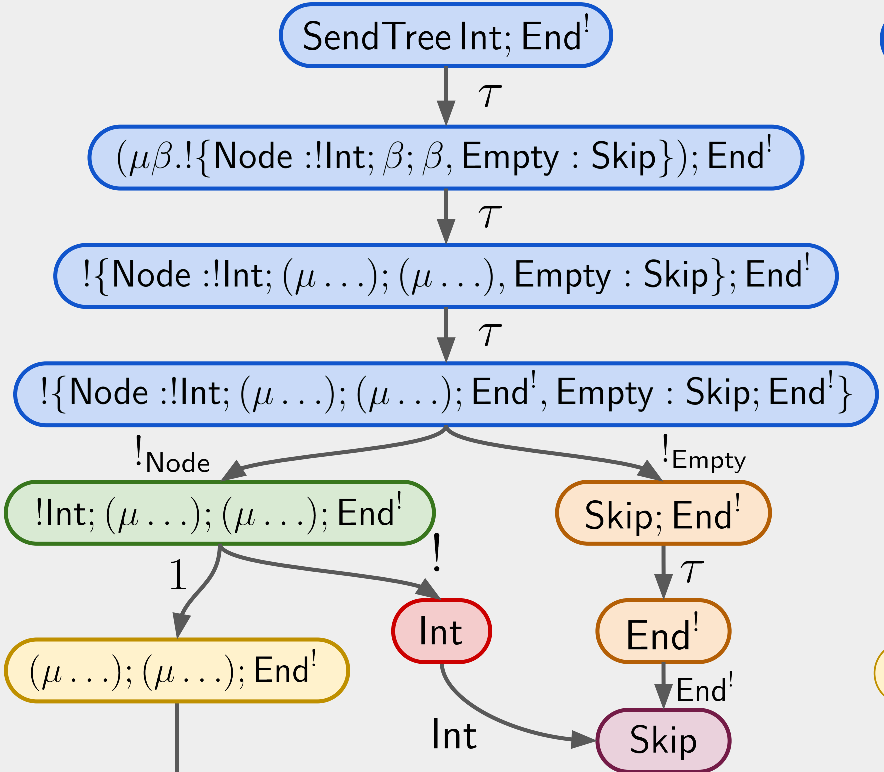
Definition. A type relation $\mathcal{R}$ is said to be a *weak bisimulation* if, whenever $T \mathrel{\mathcal{R}} U$ and for each a, we have:

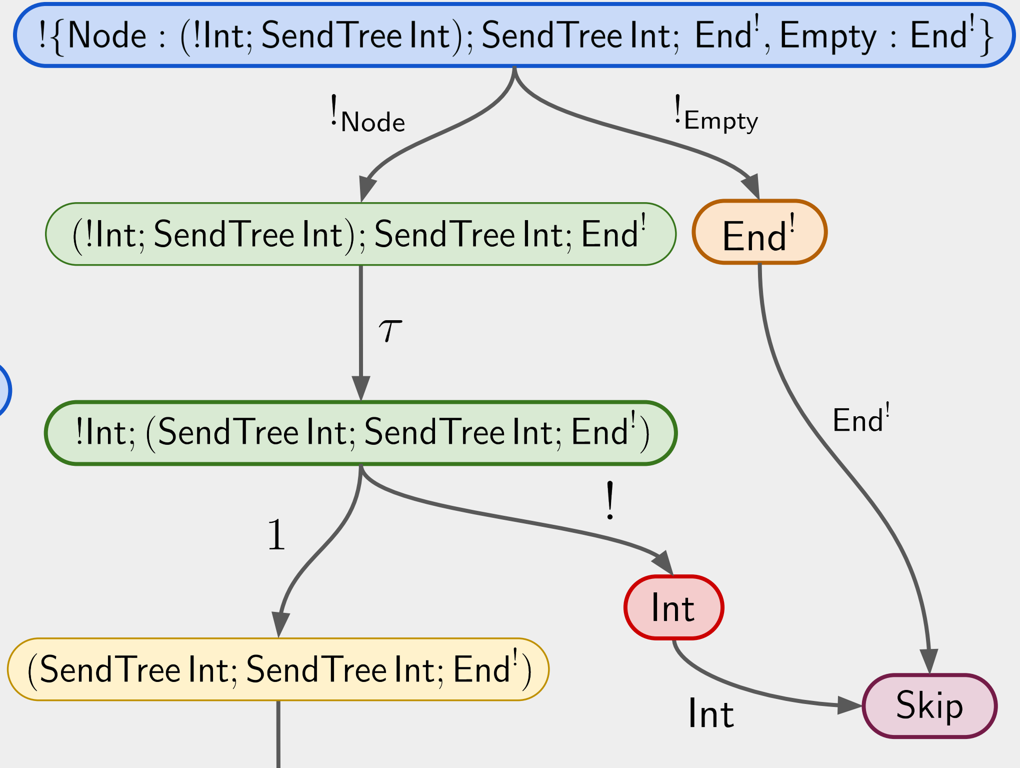
1. If $T \xrightarrow{a} T'$ then there is U' such that $U \stackrel{a}{\Longrightarrow} U'$ with $T' \mathrel{\mathcal{R}} U'$;
2. If $U \xrightarrow{a} U'$ then there is T' such that $T \stackrel{a}{\Longrightarrow} T'$ with $T' \mathrel{\mathcal{R}} U'$.

Types T and U are said to be *weakly bisimilar* ($T \stackrel{\tau}{\sim} U$) if there is a weak bisimulation $\mathcal{R}$ such that $T \mathrel{\mathcal{R}} U$.

Solution — example

$$\mathsf{SendTree} = \lambda\alpha.\mu\beta.!\{\mathsf{Node} : !\alpha; \beta; \beta, \mathsf{Empty} : \mathsf{Skip}\}$$





Solution — syntax

Types $T, U ::= \iota \mid \alpha_\kappa \mid \lambda\alpha_\kappa.T \mid T\,U$

Kinds $\kappa ::= * \mid \kappa \Rightarrow \kappa$

Constants $\iota ::= \mathsf{Int}_{\mathbf{T}}$

$\mid \;\rightarrow_{*\Rightarrow *'\Rightarrow \mathbf{T}}$

$\mid \;\forall_{(*\Rightarrow *')\Rightarrow \mathbf{T}}$

$\mid \;\mu_{(*\Rightarrow *)\Rightarrow *}$

$\mid \;\sharp_{*\Rightarrow \mathbf{S}}$

$\mid \;\sharp\{\overline{\ell}\}_{\overline{\mathbf{S}\Rightarrow}\mathbf{S}}$

$\mid \;;_{\mathbf{S}\Rightarrow\mathbf{S}\Rightarrow\mathbf{S}}$

$\mid \;\mathsf{End}^{\sharp}_{\mathbf{T}}$

$\mid \;\mathsf{Skip}_{\mathbf{S}}$

Kinds of proper types $* ::= \mathbf{T} \mid \mathbf{S}$

Abbreviations

$$T \to U = (\to)\,T\,U$$

$$\forall\alpha.T = \forall(\lambda\alpha.T)$$

$$\mu\alpha.T = \mu(\lambda\alpha.T)$$

$$\sharp\{\overline{\ell : T_\ell}\} = \sharp\{\overline{\ell}\}\,\overline{T_\ell}$$

$$T;U = (;)\,T\,U$$

Solution — LTS part I: silent actions

call-by-name reduction

$$(\lambda\alpha.T)\,U \xrightarrow{\tau} [U/\alpha]T$$

$$\frac{T \xrightarrow{\tau} T'}{T\,U \xrightarrow{\tau} T'\,U}$$

unfolding

$$\mu\,T \xrightarrow{\tau} T\,(\mu\,T)$$

session type normalization

$$\mathsf{Skip}; T \xrightarrow{\tau} T$$

$$(T_1; T_2); T_3 \xrightarrow{\tau} T_1; (T_2; T_3)$$

$$\sharp\{\overline{\ell : T_\ell}\}; U \xrightarrow{\tau} \sharp\{\overline{\ell : T_\ell; U}\}$$

$$\frac{T \neq T_1; T_2 \quad T \xrightarrow{\tau} T'}{T; U \xrightarrow{\tau} T'; U}$$

Solution — LTS part II: observable actions

fully-applied constants

$$\frac{\iota \neq \mu, ;, \mathsf{Skip}}{\iota_{\overline{\kappa_i} \Rightarrow *}\, \overline{T_i} \xrightarrow{\iota} \mathsf{Skip}} \qquad \frac{\iota \neq \mu, ;, \mathsf{Skip}}{\iota_{\overline{\kappa_i \Rightarrow}*}\, \overline{T_i} \xrightarrow{i} T_i}$$

fully-applied variables

$$\alpha_{\overline{\kappa_i \Rightarrow}*}\, \overline{T_i} \xrightarrow{\alpha} \mathsf{Skip} \qquad \alpha_{\overline{\kappa_i \Rightarrow}*}\, \overline{T_i} \xrightarrow{i} T_i$$

type operators

$$T_{\kappa \Rightarrow \kappa'} \xrightarrow{\lambda^1_\kappa} T\, \mathsf{var1}(\kappa) \qquad T_{\kappa \Rightarrow \kappa'} \xrightarrow{\lambda^2_\kappa} T\, \mathsf{var2}(\kappa)$$

sequential composition

$$\mathsf{End}^\sharp; T \xrightarrow{\mathsf{End}^\sharp} \mathsf{Skip}$$

$$(\alpha\, \overline{T_i}); U \xrightarrow{i} T_i$$

$$(\alpha\, \overline{T_i}); U \xrightarrow{\alpha} U$$

$$(\sharp T); U \xrightarrow{\sharp} U$$

$$(\sharp T); U \xrightarrow{1} T$$

Solution — grammar: constants & variables

$\mathsf{word}(\mathsf{Skip}) = \varepsilon$	
$\mathsf{word}(\mathsf{End}^\sharp) = X$	$X \to \mathsf{End}^\sharp \perp$
$\mathsf{word}(\sharp T) = X$	$X \to \sharp$ $X \to 1\,\mathsf{word}(T) \perp$
$\mathsf{word}(T; U) = \mathsf{word}(T)\,\mathsf{word}(U)$	
$\mathsf{word}(\iota_{\overline{\kappa_i \Rightarrow *}}\,\overline{T_i}) = X$ *when* $\iota = \to, \sharp\{\bar{\ell}\}, \forall$	$X \to \iota \perp$ $X \to i\,\mathsf{word}(T_i)$
$\mathsf{word}(\alpha_{\overline{\kappa_i \Rightarrow *}}\,\overline{T_i}) = X$	$X \to \alpha \perp$ $X \to i\,\mathsf{word}(T_i)$

Solution — grammar: recursion, τ, type operators

$\mathsf{word}(T) = \varepsilon$ *when* $(T = \mu T' \vee T = (\mu T'); T'') \wedge T \Downarrow \mathsf{Skip}$	
$\mathsf{word}(T) = X$ *when* $(T = \mu T' \vee T = (\mu T'); T'') \wedge T \Downarrow U$	$X \to a\,\overline{Y}\,\overline{Z}$ *where* $W\,\overline{Z} = \mathsf{word}(U) \wedge W \to a\,\overline{Y}$
$\mathsf{word}(T) = \mathsf{word}(U)$ *when* $T \xrightarrow{\tau} U$	
$\mathsf{word}(T_{\kappa \Rightarrow \kappa'}) = X$	$X \to \lambda^1_\kappa\,\mathsf{word}(T\,\mathsf{var1}(\kappa))$ $X \to \lambda^2_\kappa\,\mathsf{word}(T\,\mathsf{var2}(\kappa))$

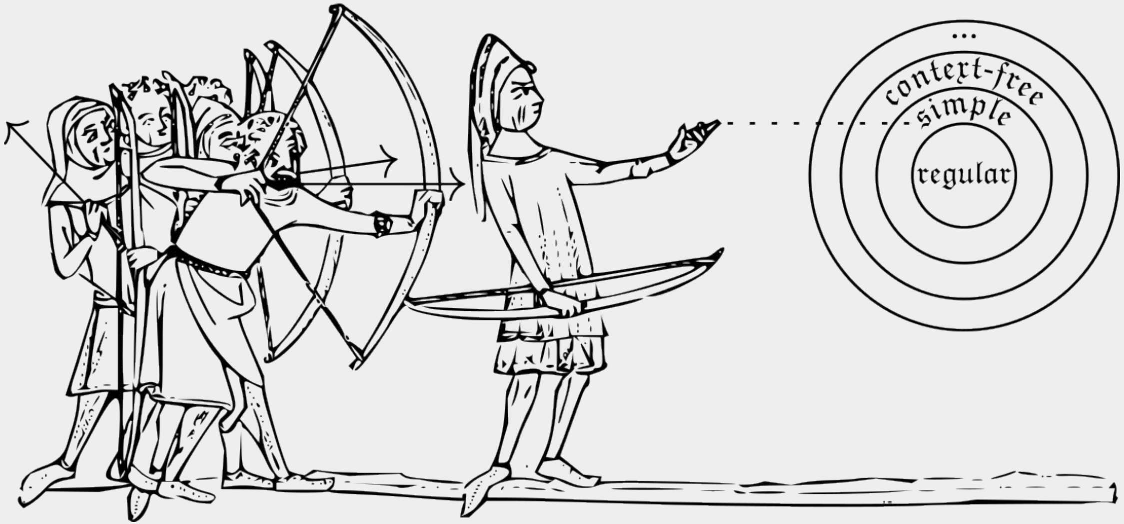
Solution

$$T \overset{\tau}{\sim} U \longrightarrow \boxed{\begin{array}{c} \mathsf{word}(T) = \overline{X} \\ \mathsf{word}(U) = \overline{Y} \\ \mathcal{P} = \overline{Z \to a\overline{Z}} \end{array}} \longrightarrow \overline{X} \sim_{\mathcal{P}} \overline{Y}$$

~~**Theorem (*full abstraction*).** The LTS of a type and its grammar word coincide.~~
But a similar result that takes into account silent actions.

Which means we can still use our simple grammar bisimilarity algorithms to decide type equivalence for $F_\omega^{\mu*;}$! $\square$

Thank you!



for more information:
freest-lang.github.io

LASIGE driven by excellence · Ciências ULisboa · fct Fundação para a Ciência e a Tecnologia