

On affine usages in signal-based communication

Roberto AMADIO

Mehdi DOGGUY

Université Paris Diderot - CNRS
Laboratoire Preuves, Programmes et Systèmes



December 2008

APLAS'08

1 Introduction

Introduction

Goal

- ▶ Design a type system that guarantees **determinacy**
- ▶ **Folk Ideas** (see, e.g., Esterel)
- ▶ **Challenge** Have a type theory that works when signals are taken as *first-class* values.
- ▶ **Ref** We build on work of Girard 1987, Wadler 1993, Kobayashi et al. 1999, ...

Why ?

- ▶ Simplify program analysis

Concurrency model

▶ Synchronous/Timed model

- ▶ Notion of instant, phase, round, . . .
- ▶ Time passes (from an instant to the next) exactly when there is nothing to do.

▶ Signal-based interaction

- ▶ **Emission** a value emitted on a signal persists during the instant and disappears at the end of it.
- ▶ **Reception** Two possibilities:
 - ▶ During the instant, select non-deterministically a value emitted on the signal.
 - ▶ At the end of the instant, collect non-deterministically all the values emitted on the signal.

S π -calculus

- ▶ a **synchronous** π -calculus
- ▶ a dereferenciation operator on signals **!** s
- ▶ inputs depend on the instant
- ▶ outputs are persistent within the instant

$$\begin{array}{l} P ::= 0 \\ \quad | A(\mathbf{e}) \\ \quad | \bar{s} \langle v \rangle \\ \quad | s(x).P, K \\ \quad | P_1 \mid P_2 \\ \quad | \nu s P \\ \quad | [u \sqsupseteq p] P_1, P_2 \\ \quad | [s_1 = s_2] P_1, P_2 \end{array} \quad \text{processes}$$

$$K ::= A(\mathbf{r}) \quad \text{continuation}$$

S π -calculus

- ▶ a **synchronous** π -calculus
- ▶ a dereferenciation operator on signals **!s**
- ▶ inputs depend on the instant
- ▶ outputs are persistent within the instant

$$\begin{array}{l} P ::= 0 \\ \quad | A(\mathbf{e}) \\ \quad | \bar{s} \langle v \rangle \\ \quad | s(x).P, K \\ \quad | P_1 \mid P_2 \\ \quad | \nu s P \\ \quad | [u \sqsupseteq p] P_1, P_2 \\ \quad | [s_1 = s_2] P_1, P_2 \end{array} \quad \text{processes}$$

$$K ::= A(\mathbf{r}) \quad \text{continuation}$$

S π -calculus

- ▶ a **synchronous** π -calculus
- ▶ a dereferenciation operator on signals **!s**
- ▶ **inputs depend on the instant**
- ▶ outputs are persistent within the instant

$$\begin{array}{l} P ::= 0 \\ \quad | A(\mathbf{e}) \\ \quad | \bar{s} \langle v \rangle \\ \quad | s(x).P, K \\ \quad | P_1 \mid P_2 \\ \quad | \nu s P \\ \quad | [u \sqsupseteq p] P_1, P_2 \\ \quad | [s_1 = s_2] P_1, P_2 \end{array} \quad \text{processes}$$

$$K ::= A(\mathbf{r}) \quad \text{continuation}$$

S π -calculus

- ▶ a **synchronous** π -calculus
- ▶ a dereferenciation operator on signals **!s**
- ▶ inputs depend on the instant
- ▶ **outputs are persistent within the instant**

$$\begin{array}{l} P ::= 0 \\ \quad | A(\mathbf{e}) \\ \quad | \bar{s} \langle v \rangle \\ \quad | s(x).P, K \\ \quad | P_1 \mid P_2 \\ \quad | \nu s P \\ \quad | [u \sqsupseteq p] P_1, P_2 \\ \quad | [s_1 = s_2] P_1, P_2 \end{array} \quad \text{processes}$$

$$K ::= A(\mathbf{r}) \quad \text{continuation}$$

Example

$$\nu s_1, s_2 (\quad \overline{s_1} \langle v_1 \rangle \mid \overline{s_1} \langle v_2 \rangle \mid \\ s_1(x). (s_1(y). (s_2(z). (A(x, y), B(!s_1)) \\ \quad \quad \quad , Nil_2())) \\ \quad \quad \quad , Nil_1()) \\)$$

Example

$$\nu s_1, s_2 (\quad \overline{s_1} \langle v_1 \rangle \mid \overline{s_1} \langle v_2 \rangle \mid \\ s_1(x). (s_1(y). (s_2(z). (A(x, y), B(!s_1)) \\ \quad \quad \quad , Nil_2())) \\ \quad \quad \quad , Nil_1()) \\)$$

$\Downarrow$

$$\nu s_1, s_2 (s_2(z). A(\sigma(x), \sigma(y)))$$

Where $\sigma(x), \sigma(y) \in \{v_1, v_2\}$ and $\sigma(x) \neq \sigma(y)$

Example

$$\nu s_1, s_2 (\quad \overline{s_1} \langle v_1 \rangle \mid \overline{s_1} \langle v_2 \rangle \mid \\ s_1(x). (s_1(y). (s_2(z). (A(x, y), B(!s_1)) \\ \quad , Nil_2())) \\ \quad , Nil_1()) \\)$$

$$\Downarrow \\ \nu s_1, s_2 (\overline{s_1} \langle v_1 \rangle \mid \overline{s_1} \langle v_2 \rangle \mid s_2(z). (A(\sigma(x), \sigma(y)), B(!s_1)))$$

Example

$$\nu s_1, s_2 (\quad \overline{s_1} \langle v_1 \rangle \mid \overline{s_1} \langle v_2 \rangle \mid \\ s_1(x). (s_1(y). (s_2(z). (A(x, y), B(!s_1)) \\ \quad , Nil_2())) \\ \quad , Nil_1()) \\)$$

$$\Downarrow \\ \nu s_1, s_2 (\overline{s_1} \langle v_1 \rangle \mid \overline{s_1} \langle v_2 \rangle \mid s_2(z). (A(\sigma(x), \sigma(y)), B(!s_1))) \\ \Downarrow^N$$

Example

$$\nu s_1, s_2 (\quad \overline{s_1} \langle v_1 \rangle \mid \overline{s_1} \langle v_2 \rangle \mid \\ s_1(x). (s_1(y). (s_2(z). (A(x, y), B(!s_1)) \\ \quad , Nil_2())) \\ \quad , Nil_1()) \\)$$

$$\Downarrow \\ \nu s_1, s_2 (\overline{s_1} \langle v_1 \rangle \mid \overline{s_1} \langle v_2 \rangle \mid s_2(z). (A(\sigma(x), \sigma(y)), B(!s_1)))$$

$$\Downarrow^N \\ \nu s_1, s_2. B(I) \\ \text{Where } I \in \{[v_1; v_2], [v_2; v_1]\}$$

Example: Client-Server

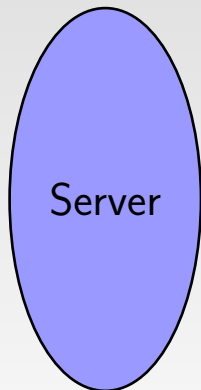
Server handles a **set of requests** of the shape $\text{req}(s', x)$ emitted in the previous instant on the signal s . The answer is a function of x emitted on s' .

$$\begin{aligned} \text{Server}(s) &= \text{pause.Handle}(s, !s) \\ \text{Handle}(s, \ell) &= [\ell \sqsupseteq \text{cons}(\text{req}(s', x), \ell')] \\ &\quad (\overline{s'} \langle f(x) \rangle \mid \text{Handle}(s, \ell')) \\ &\quad , \text{Server}(s) \end{aligned}$$

Client emits a request $\text{req}(s', x)$ on s and waits for a reply on s' in the following instant.

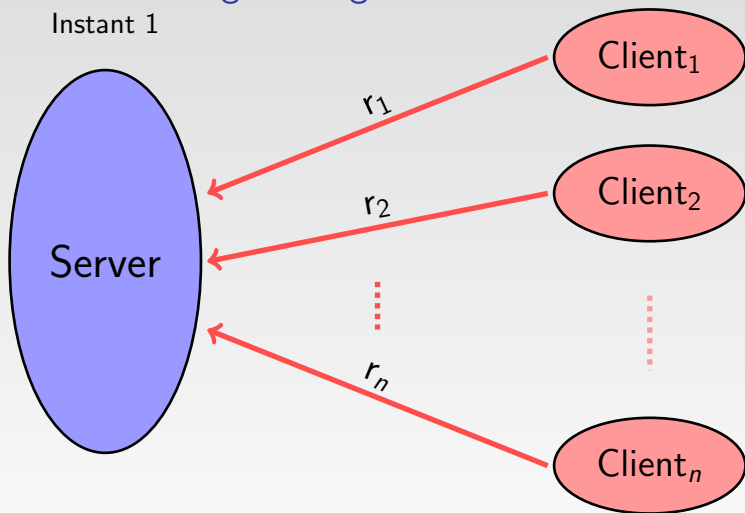
$$\text{Client}(x, s, t) = \nu s' (\overline{s} \langle \text{req}(s', x) \rangle \mid \text{pause}.s'(x).\overline{t} \langle x \rangle, 0)$$

Determinism and signal usage



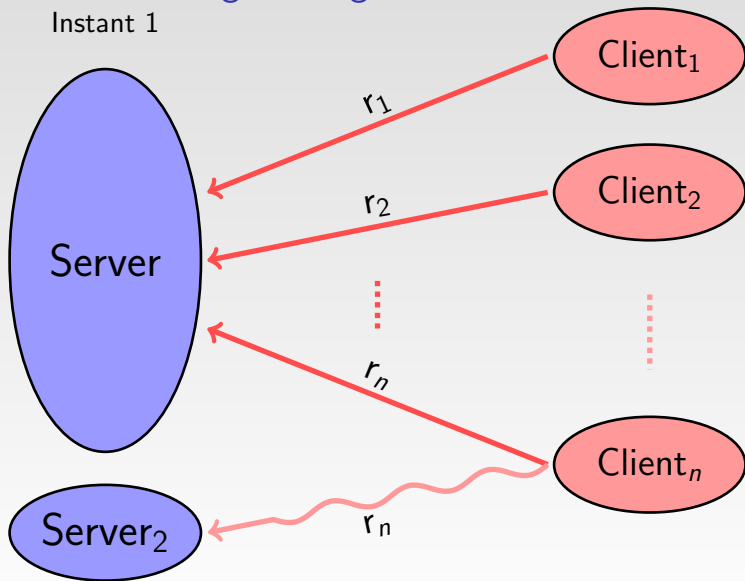
Determinism and signal usage

Instant 1



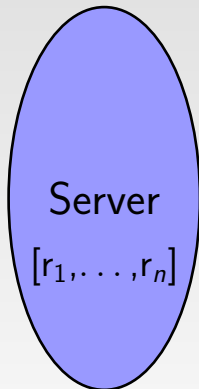
Determinism and signal usage

Instant 1



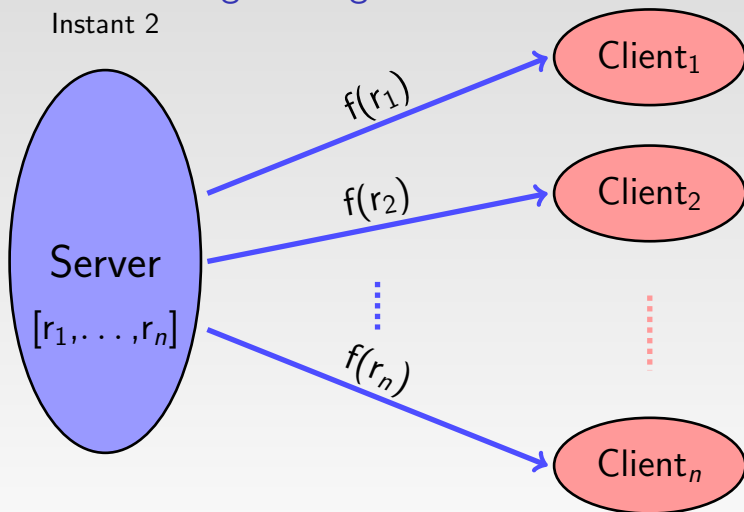
Determinism and signal usage

End of the instant 1



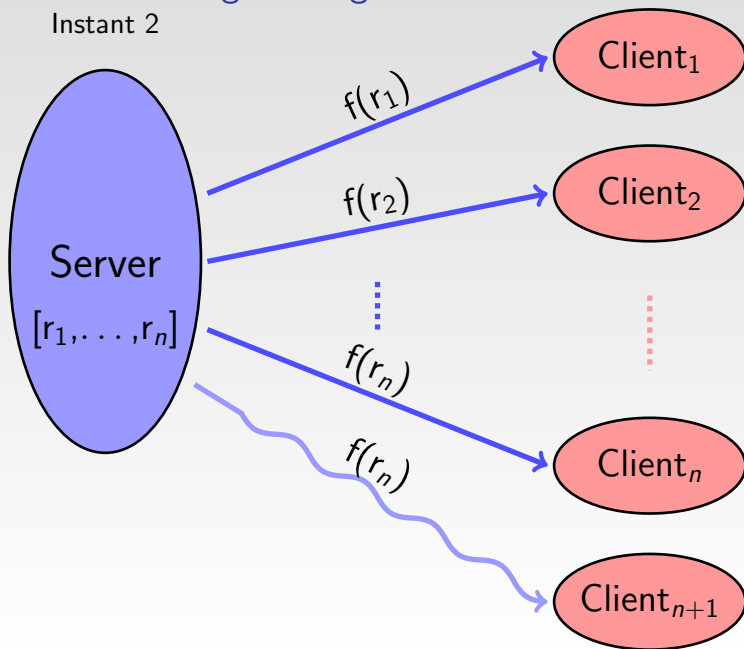
Determinism and signal usage

Instant 2



Determinism and signal usage

Instant 2



2 **Technical overview**

Technical overview

1. Notion of usage
2. Types with usages
3. Type system with usages
4. Set-types and proof obligations
5. Typed transition and *subject reduction*
6. Typed bisimulation
7. Typed programs are *deterministic*

1. Notion of usage (1/4)

Usages are:

$$u \in L = \left\{ \begin{array}{ll} 0 & \text{Cannot be used} \\ 1 & \text{Used at most once} \\ \infty & \text{Used an arbitrary number of times} \end{array} \right\}$$

Partial sum¹ operation on usages

$$u \oplus u' = \left\{ \begin{array}{ll} 0 \oplus x = x & x \in L \\ 1 \oplus 1 = \uparrow \\ \infty \oplus 1 = \uparrow \\ \infty \oplus x = \infty & x \in \{0, \infty\} \end{array} \right.$$

¹up to symmetry

1. Notion of usage (2/4)

$$s : \text{Sig} \binom{n}{m}{p} \omega(\sigma)$$

where: $\left\{ \begin{array}{ll} n & \text{maximum number of outputs} \\ m & \text{maximum number of inputs} \\ p & \text{maximum number of inputs at the end of the instant} \end{array} \right.$

$$n, m, p \in L = \{0, 1, \infty\}$$

Addition over signal usages

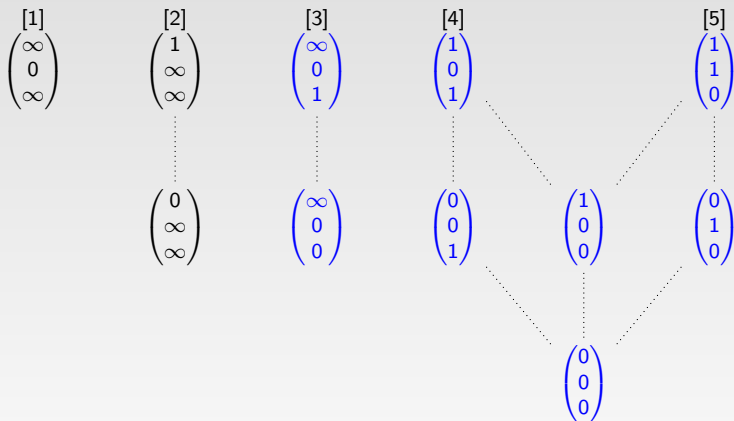
$$\text{Sig}_u(\sigma) \oplus \text{Sig}_v(\sigma) = \text{Sig}_{u \oplus v}(\sigma)$$

1. Notion of usage (3/4)

Definition

- ▶ *Signal usage* is an infinite word on the alphabet $L \times L \times L$
- ▶ Non uniform *signal usage*: $u \cdot v^\omega$ ($u, v \in L \times L \times L$)
- ▶ Data structure (List, Set, ...) usage belongs to $\{1, \infty\}$
- ▶ Classification of usages:
 - ▶ **Affine**: Contains a 1.
 - ▶ **Uniform**: Does not depend on the instant (data structure and transmitted values have always a uniform usage).
 - ▶ **Affine preserving**: Can be composed with an affine usage.

1. Notion of usage (4/4)



- Leafs are called **neutral usages**
- Usages containing **1** are called affine
- **Blue** ones are affine preserving

2. Types with usages

Intuition: A constructor can be applied to an affine type only if it is affine preserving.

$$\kappa ::= 1 \mid List_{\infty}(\kappa) \mid Set_{\infty}(\kappa) \mid Sig_u(\kappa)$$

Classical uniform types
(*u is neutral*)

2. Types with usages

Intuition: A constructor can be applied to an affine type only if it is affine preserving.

$$\kappa ::= 1 \mid List_{\infty}(\kappa) \mid Set_{\infty}(\kappa) \mid Sig_u(\kappa)$$

Classical uniform types

(u is neutral)

$$\lambda ::= List_1(\sigma) \mid Set_1(\sigma) \mid Sig_u(\kappa) \mid Sig_v(\lambda)$$

Affine uniform types

(u is uniform and affine, v is uniform and affine preserving)

2. Types with usages

Intuition: A constructor can be applied to an affine type only if it is affine preserving.

$$\kappa ::= 1 \mid List_{\infty}(\kappa) \mid Set_{\infty}(\kappa) \mid Sig_u(\kappa)$$

Classical uniform types

(u is neutral)

$$\lambda ::= List_1(\sigma) \mid Set_1(\sigma) \mid Sig_u(\kappa) \mid Sig_v(\lambda)$$

Affine uniform types

(u is uniform and affine, v is uniform and affine preserving)

$$\sigma ::= \kappa \mid \lambda$$

Uniform types

$$\rho ::= \sigma \mid Sig_u(\kappa) \mid Sig_v(\lambda)$$

Non uniform types

(u is neutral and v is affine)

3. Type system with usages

We must guarantee what we emit and we can rely on what we receive.

$$(out) \frac{\Gamma_1 \vdash s : Sig_u(\sigma) \quad u(0)_1 \neq 0 \quad \Gamma_2 \vdash e : \sigma}{\Gamma_1 \oplus \Gamma_2 \vdash \bar{s} \langle e \rangle}$$

$$(in) \frac{\Gamma_1 \vdash s : Sig_u(\sigma) \quad u(0)_2 \neq 0 \quad \Gamma_2, x : \sigma \vdash P \quad (\Gamma_1 \oplus \Gamma_2) \vdash [A(\mathbf{r})]}{(\Gamma_1 \oplus \Gamma_2) \vdash s(x).P, A(\mathbf{r})}$$

4. Set-types and proof obligations

If a thread identifier A waits for an argument of type $Set_x(D)$ (where $x \in \{1, \infty\}$) then we have a proof obligation:

$$\frac{\ell \text{ a permutation of } \ell'}{A(\ell) \approx A(\ell')}$$

In the client-server example it suffices to appeal to simple algebraic properties of bisimulation: associativity and commutativity of parallel composition and preservation under static contexts.

5. Typed transition and *subject reduction*

First we have to define a few concepts:

- ▶ A **labelled-transition system** on the language. The definition is twisted so that the definition of bisimulation that we build on top of it looks standard (cf. 'asynchronous' π -calculus).
- ▶ The **compatibility** of typing context Γ with an action act : $(\Gamma, act) \downarrow$. Unconstrained transitions may fail to preserve typing.
- ▶ The **residual context** $\Gamma(act)$ after an action act compatible with Γ .

5. Typed transition and *subject reduction*

First we have to define a few concepts:

- ▶ A **labelled-transition system** on the language. The definition is twisted so that the definition of bisimulation that we build on top of it looks standard (cf. 'asynchronous' π -calculus).
- ▶ The **compatibility** of typing context Γ with an action act : $(\Gamma, act) \downarrow$. Unconstrained transitions may fail to preserve typing.
- ▶ The **residual context** $\Gamma(act)$ after an action act compatible with Γ .

Then **subject reduction** is stated as follows:

$$\frac{\Gamma \vdash P \quad P \xrightarrow{act} P' \quad (\Gamma, act) \downarrow}{\Gamma(act) \vdash P'}$$

6. Typed bisimulation

We specialise bisimulation with respect to a typing context and typed transitions

$$\frac{P \approx_{\Gamma} Q \quad P \xRightarrow{act} P'}{Q \xRightarrow{act} Q' \quad P' \approx_{\Gamma(act)} Q'}$$

6. Typed bisimulation

We specialise bisimulation with respect to a typing context and typed transitions

$$\frac{P \approx_{\Gamma} Q \quad P \xrightarrow{act} P'}{Q \xrightarrow{act} Q' \quad P' \approx_{\Gamma(act)} Q'}$$

As expected, this is a stronger notion than ‘type-free’ bisimulation. Note that typed bisimulation relies on a type system which relies on ‘type-free’ bisimulation.

Proposition

If $P, Q \in Pr(\Gamma)$ and $P \approx Q$ then $P \approx_{\Gamma(act)} Q$

7. Typed programs are *deterministic*

Theorem

Let P be a program and C a static context such that $C[P]$ is typable. If $C[P]$ reduces to Q_1 and Q_2 after n instants then Q_1 and Q_2 are typed bisimilar.

Proof steps

- ▶ Reductions within the instant commute strongly:

$$\frac{P \xrightarrow{\tau}_{\Gamma} P_i, \quad i = 1, 2}{\exists Q P_i (\xrightarrow{\tau}_{\Gamma} \cup =) Q, \quad i = 1, 2}$$

- ▶ Reductions at the end of the instant preserve typed bisimulation:

$$\frac{P \xrightarrow{N}_{\Gamma} P_i, \quad i = 1, 2}{P_1 \approx_{\uparrow(\Gamma)} P_2}$$

3 Conclusion

Conclusion

Main contribution

- ▶ identification of 5 kinds of usages
- ▶ and rules that allow their *composition* while preserving *determinacy*.

Conclusion

Main contribution

- ▶ identification of 5 kinds of usages
- ▶ and rules that allow their *composition* while preserving *determinacy*.

Ongoing work

- ▶ Implementation of the typing system in OCaml
- ▶ Linear case of the typing system
- ▶ Checking of process termination

Thank you !
Questions ?

Example: Client-Server

$$s : \text{Sig} \begin{pmatrix} \infty \\ 0 \\ 1 \end{pmatrix}^{\omega} (\text{Req}) \quad I : \text{Set}_1(\text{Req}) \quad s' : \text{Sig} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}^{\omega} (\text{Data})$$

$$\begin{aligned} \text{Server}(s) &= \text{pause.Handle}(s, !s) \\ \text{Handle}(s, \ell) &= [\ell \supseteq \text{cons}(\text{req}(s', x), \ell')] \\ &\quad (\overline{s'} \langle f(x) \rangle \mid \text{Handle}(s, \ell')) \\ &\quad , \text{Server}(s) \end{aligned}$$

$$\text{Client}(x, s, t) = \nu s' (\overline{s} \langle \text{req}(s', x) \rangle \mid \text{pause}.s'(x). \overline{t} \langle x \rangle, 0)$$

Typing expressions

$$\langle k \rangle \frac{\Gamma_i \vdash \langle r_i \rangle : \sigma_i \quad i = 1, \dots, n \quad k : (\sigma_1, \dots, \sigma_n) \rightarrow \sigma \quad k = f \text{ or } k = c}{\Gamma_0 \oplus \Gamma_1 \oplus \dots \oplus \Gamma_n \vdash \langle k(r_1, \dots, r_n) \rangle : \sigma} \quad (k) \frac{\Gamma_i \vdash e_i : \sigma_i \quad i = 1, \dots, n \quad k : (\sigma_1, \dots, \sigma_n) \rightarrow \sigma \quad k = f \text{ or } k = c}{\Gamma_0 \oplus \Gamma_1 \oplus \dots \oplus \Gamma_n \vdash k(e_1, \dots, e_n) : \sigma}$$

$$\langle \text{var}_C \rangle \frac{Op = C \quad Op = Set}{\Gamma, x : Op_u(\sigma) \vdash \langle x \rangle : Op_u(\sigma)}$$

$$\langle \text{var}_{sig} \rangle \frac{y^\omega \geq u}{\Gamma, s : Sig_{xy^\omega}(\sigma) \vdash \langle s \rangle : Sig_u(\sigma)}$$

$$(var) \frac{u \geq u' \quad Op \in \{Sig, Set, C\}}{\Gamma, x : Op_u(\sigma) \vdash x : Op_{u'}(\sigma)}$$

$$\langle !_{Set} \rangle \frac{(u(0) \geq (\infty, 0, \infty) \wedge x = \infty) \vee (u(0) \geq (\infty, 0, 1) \wedge x = 1)}{\Gamma, s : Sig_u(\sigma) \vdash \langle !s \rangle : Set_x(\sigma)}$$

$$\langle !_{List} \rangle \frac{(u(0) \geq (0, \infty, \infty) \wedge x = \infty) \vee (u(0) \geq (0, 0, 1) \wedge x = 1)}{\Gamma, s : Sig_u(\sigma) \vdash \langle !s \rangle : List_x(\sigma)}$$

Typing programs

$$(0) \frac{}{\Gamma \vdash 0} \quad (out) \frac{\Gamma_1 \vdash s : Sig_u(\sigma) \quad u(0)_1 \neq 0 \quad \Gamma_2 \vdash e : \sigma}{\Gamma_1 \oplus \Gamma_2 \vdash \bar{s} \langle e \rangle}$$

$$(\nu) \frac{\Gamma, s : Sig_u(\sigma) \vdash P}{\Gamma \vdash \nu s : Sig_u(\sigma) P} \quad (in) \frac{\Gamma_1 \vdash s : Sig_u(\sigma) \quad u(0)_2 \neq 0 \quad \Gamma_2, x : \sigma \vdash P \quad (\Gamma_1 \oplus \Gamma_2) \vdash [A(r)]}{(\Gamma_1 \oplus \Gamma_2) \vdash s(x).P, A(r)}$$

$$(m_s) \frac{s_1, s_2 \in dom(\Gamma) \quad \Gamma \vdash P_i \quad i = 1, 2}{\Gamma \vdash [s_1 = s_2] P_1, P_2} \quad (m_c) \frac{c : (\sigma_1, \dots, \sigma_n) \rightarrow \sigma \quad \Gamma_1 \vdash u : \sigma \quad \Gamma_2, x_1 : \sigma_1, \dots, x_n : \sigma_n \vdash P_1 \quad \Gamma_1 \oplus \Gamma_2 \vdash P_2}{\Gamma_1 \oplus \Gamma_2 \vdash [u \triangleright c(x_1, \dots, x_n)] P_1, P_2}$$

$$(par) \frac{\Gamma_i \vdash P_i \quad i = 1, 2}{\Gamma_1 \oplus \Gamma_2 \vdash P_1 \mid P_2} \quad (rec) \frac{A : (\sigma_1, \dots, \sigma_n) \quad \Gamma_i \vdash e_i : \sigma_i \quad i = 1, \dots, n}{\Gamma_1 \oplus \dots \oplus \Gamma_n \vdash A(e_1, \dots, e_n)}$$

$$(out) \frac{\Gamma \vdash s : Sig_u(\sigma) \quad u(0) = (1, 1, 0)}{\Gamma \vdash \bar{s} \langle e \rangle} \quad \langle rec \rangle \frac{A : (\sigma_1, \dots, \sigma_n) \quad \Gamma_i \vdash \langle r_i \rangle : \sigma_i \quad i = 1, \dots, n}{\Gamma_1 \oplus \dots \oplus \Gamma_n \vdash \langle A(r_1, \dots, r_n) \rangle}$$