Sessions as propositions

Sam Lindley and J. Garrett Morris

The University of Edinburgh
{Sam.Lindley,Garrett.Morris}@ed.ac.uk

Abstract

Recently, Wadler presented a continuation-passing translation from a session-typed functional language, GV, to a process calculus based on classical linear logic, CP. However, this translation is one-way: CP is more expressive than GV. We propose an extension of GV, called HGV, and give translations showing that it is as expressive as CP. The new translations shed light both on the original translation from GV to CP, and on the limitations in expressiveness of GV.

1 Introduction

Linear logic has long been regarded as a potential typing discipline for concurrency. Girard [7] observes that the connectives of linear logic can be interpreted as parallel computation. Abramsky [1] and Bellin and Scott [2] interpret linear logic proofs as processes in Milner's π-calculus. While they provide π-calculus interpretations of all linear logic proofs, they do not provide a proof-theoretic interpretation for arbitrary π-calculus terms. Caires and Pfenning [3] observe that the multiplicative connectives can be interpreted as session types. Their process calculus, based on intuitionistic linear logic, is closer to π-calculus than are traditional session-typed languages. Wadler [8] shows that a core session-typed linear functional language, GV, patterned after a similar language due to Gay and Vasconcelos [6], may be translated into a process calculus, CP, whose terms are proofs in classical linear logic. However, GV is less expressive than CP: there are proofs which do not correspond to any GV program.

Our primary contribution is HGV (Harmonious GV), a version of GV extended with constructs for session forwarding, replication, and polymorphism. We identify HGVπ, the session-typed fragment of HGV, and give a type-preserving translations from HGV to HGVπ ($(-)^\star$); this translation depends crucially on the new constructs of HGV. We show that HGV is sufficient to express all linear logic proofs by giving type-preserving translations from HGVπ to CP ($[\![-]\!]$), and from CP to HGVπ ($(\!|-|\!)$). Factoring the translation of HGV into CP through $(-)^\star$ simplifies the presentation, and illuminates regularities that are not apparent in Wadler's original translation of GV into CP. Finally, we show that HGV, HGVπ, and CP are all equally expressive.

2 The HGV Language

This section describes our session-typed language HGV, contrasting it with Gay and Vasconcelos's functional language for asynchronous session types [6], which we call LAST, and Wadler's GV [8]. In designing HGV, we have opted for programming convenience over uniformity, while insisting on a tight correspondence with linear logic. The session types of HGV are given by the following grammar:

$$S ::= !T.S \mid ?T.S \mid \oplus\{l_i : S_i\}_i \mid \&\{l_i : S_i\}_i \mid \mathsf{end}_! \mid \mathsf{end}_? \mid X \mid \overline{X} \mid ![X].S \mid ?[X].S \mid \flat S \mid \sharp S$$

Types for input ($?T.S$), output ($!T.S$), selection ($\oplus\{l_i : S_i\}_i$) and choice ($\&\{l_i : S_i\}_i$) are standard. Like GV, but unlike LAST, we distinguish output ($\mathsf{end}_!$) and input ($\mathsf{end}_?$) session ends; this matches the situation in linear logic, where there is no conveniently self-dual proposition to represent the end of a session. Variables and their duals ($X, \overline{X}$) and type input ($?[X].S$) and output ($![X].S$), permit definition of polymorphic sessions. We include a notion of replicated sessions, corresponding to exponentials in linear logic: a channel of type $\sharp S$ is a "service", providing any number of channels of type S; a channel of type $\flat S$ is the "server" providing such a service. Each session type S has a dual $\overline{S}$ (with the obvious dual for variables X):

$$\begin{array}{lllll} \overline{!T.S} = ?T.\overline{S} & \overline{\oplus\{l_i : S_i\}_i} = \&\{l_i : \overline{S_i}\}_i & \overline{\mathsf{end}_!} = \mathsf{end}_? & \overline{![X].S} = ?[X].\overline{S} & \overline{\sharp S} = \flat\overline{S} \\ \overline{?T.S} = !T.\overline{S} & \overline{\&\{l_i : S_i\}_i} = \oplus\{l_i : \overline{S_i}\}_i & \overline{\mathsf{end}_?} = \mathsf{end}_! & \overline{?[X].S} = ![X].\overline{S} & \overline{\flat S} = \sharp\overline{S} \end{array}$$

Note that dualisation leaves input and output types unchanged. In addition to sessions, HGV's types include linear pairs, and linear and unlimited functions:

$$T, U, V ::= S \mid T \otimes U \mid T \multimap U \mid T \to U$$

Every type T is either linear ($lin(T)$) or unlimited ($un(T)$); the only unlimited types are services ($un(\sharp S)$), unlimited functions ($un(T \to U)$), and end input session types ($un(\mathsf{end}_?)$). In GV, $\mathsf{end}_?$ is linear. We choose to make it unlimited in HGV because then we can dispense with GV's explicit $\mathsf{terminate}$ construct while maintaining a strong correspondence with CP—$\mathsf{end}_?$ corresponds to $\perp$ in CP, for which weakening and contraction are derivable.

Figure 1 gives the terms and typing rules for HGV; the first block contains the structural rules, the second contains the (standard) rules for lambda terms, and the third contains the session-typed fragment. The fork construct provides session initiation, filling the role of GV's $\mathsf{with} \ldots \mathsf{connect}$ structure, but without the asymmetry of the latter. The two are interdefinable, as follows:

$$\mathsf{fork}\ x.M \equiv \mathsf{with}\ x\ \mathsf{connect}\ M\ \mathsf{to}\ x \qquad \mathsf{with}\ x\ \mathsf{connect}\ M\ \mathsf{to}\ N \equiv \mathsf{let}\ x = \mathsf{fork}\ x.M\ \mathsf{in}\ N$$

We add a construct $\mathsf{link}\ M\ N$ to implement channel forwarding; this form is provided in neither GV nor LAST, but is necessary to match the expressive power of CP. (Note that while we could define session forwarding in GV or LAST for any particular session type, it is not possible to do so in a generic fashion.) We add terms $\mathsf{sendType}\ S\ M$ and $\mathsf{receiveType}\ X.M$ to provide session polymorphism, and $\mathsf{serve}\ x.M$ and $\mathsf{request}\ M$ for replicated sessions. Note that, as the body M of $\mathsf{serve}\ x.M$ may be arbitrarily replicated, it can only refer to the unlimited portion of the environment. Channels of type $\sharp S$ offer arbitrarily many sessions of type S; correspondingly, channels of type $\flat S$ must consume arbitrarily many S sessions. The rule for $\mathsf{serve}\ x.M$ parallels that for fork: it defines the server (which replicates M) and returns the channel by which it may be used (of type $\overline{\flat S} = \sharp\overline{S}$). As a consequence, there is no rule involving type $\flat S$. We experimented with having such a rule, but found that it was always used immediately inside a fork, while providing no extra expressive power. Hence we opted for the rule presented here.

3 From HGV to HGVπ

The language HGVπ is the restriction of HGV to session types, that is, HGV without $\multimap$, $\to$, or $\otimes$. In order to avoid $\otimes$, we disallow plain $\mathsf{receive}\ M$, but do permit it to be fused with a pair elimination $\mathsf{let}\ (x, y) = \mathsf{receive}\ M\ \mathsf{in}\ N$. We can simulate all non-session types as session types

Structural rules

$$\frac{}{x : T \vdash x : T} \qquad \frac{\Phi \vdash N : U \qquad un(T)}{\Phi, x : T \vdash N : U} \qquad \frac{\Phi, x : T, x' : T \vdash N : U \qquad un(T)}{\Phi, x : T \vdash N[x/x'] : U}$$

Lambda rules

$$\frac{\Phi, x : T \vdash N : U}{\Phi \vdash \lambda x.N : T \multimap U} \qquad \frac{\Phi \vdash L : T \multimap U \qquad \Psi \vdash M : T}{\Phi, \Psi \vdash L\ M : U} \qquad \frac{\Phi \vdash L : T \multimap U \qquad un(\Phi)}{\Phi \vdash L : T \to U}$$

$$\frac{\Phi \vdash L : T \to U}{\Phi \vdash L : T \multimap U} \qquad \frac{\Phi \vdash M : T \qquad \Psi \vdash N : U}{\Phi, \Psi \vdash (M, N) : T \otimes U} \qquad \frac{\Phi \vdash M : T \otimes U \qquad \Psi, x : T, y : U \vdash N : V}{\Phi, \Psi \vdash \mathsf{let}\ (x, y) = M\ \mathsf{in}\ N : V}$$

Session rules

$$\frac{\Phi \vdash M : T \qquad \Psi \vdash N : !T.S}{\Phi \vdash \mathsf{send}\ M\ N : S} \qquad \frac{\Phi \vdash M : ?T.S}{\Phi \vdash \mathsf{receive}\ M : T \otimes S}$$

$$\frac{\Phi \vdash M : \oplus\{l_i : S_i\}_i}{\Phi \vdash \mathsf{select}\ l_j\ M : S_j} \qquad \frac{\Phi \vdash M : \&\{l_i : S_i\}_i \qquad \{\Psi, x : S_i \vdash N_i : T\}_i}{\Phi, \Psi \vdash \mathsf{case}\ M\ \mathsf{of}\ \{l_i(x).N_i\}_i : T}$$

$$\frac{\Phi, x : S \vdash M : \mathsf{end}_!}{\Phi \vdash \mathsf{fork}\ x.M : \overline{S}} \qquad \frac{\Phi \vdash M : S \qquad \Phi \vdash N : \overline{S}}{\Phi \vdash \mathsf{link}\ M\ N : \mathsf{end}_!} \qquad \frac{\Phi \vdash M : ![X].S'}{\Phi \vdash \mathsf{sendType}\ S\ M : S'[S/X]}$$

$$\frac{\Phi \vdash M : ?[X].S \qquad X \notin FV(\Phi)}{\Phi \vdash \mathsf{receiveType}\ X.M : S} \qquad \frac{\Phi, x : S \vdash M : \mathsf{end}_! \qquad un(\Phi)}{\Phi \vdash \mathsf{serve}\ x.M : \sharp\overline{S}} \qquad \frac{\Phi \vdash M : \sharp S}{\Phi \vdash \mathsf{request}\ M : S}$$

Figure 1: Typing rules for HGV

via a translation from HGV to HGVπ. The translation on types is given by the homomorphic extension of the following equations:

$$(T \multimap U)^\star = !(T)^\star.\overline{(U)^\star} \qquad (T \to U)^\star = \sharp(!(T)^\star.\overline{(U)^\star}) \qquad (T \otimes U)^\star = ?(T)^\star.\overline{(U)^\star}$$

Each target type is the *interface* to the simulated source type. A linear function is simulated by input on a channel; its interface is output on the other end of the channel. An unlimited function is simulated by a server; its interface is the service on the other end of that channel. A tensor is simulated by output on a channel; its interface is input on the other end of that channel. This duality between implementation and interface explains the flipping of types in Wadler's original CPS translation from GV to CP. The translation on terms is given by the homomorphic extension of the following equations:

$$\begin{aligned}
(\lambda x.M)^\star &= \mathsf{fork}\ z.\mathsf{let}\ (x, z) = \mathsf{receive}\ z\ \mathsf{in}\ \mathsf{link}\ (M)^\star\ z \\
(L\ M)^\star &= \mathsf{send}\ (M)^\star\ (L)^\star \\
(M \otimes N)^\star &= \mathsf{fork}\ z.\mathsf{link}\ (\mathsf{send}\ (M)^\star\ z)\ (N)^\star \\
(\mathsf{let}\ (x, y) = M\ \mathsf{in}\ N)^\star &= \mathsf{let}\ (x, y) = \mathsf{receive}\ (M)^\star\ \mathsf{in}\ (N)^\star \\
(L : T \to U)^\star &= \mathsf{fork}\ z.\mathsf{link}\ z\ (\mathsf{serve}\ y.\mathsf{link}\ (L)^\star\ y) \\
(L : T \multimap U)^\star &= \mathsf{request}\ (L)^\star \\
(\mathsf{receive}\ M)^\star &= (M)^\star
\end{aligned}$$

$$\frac{}{w \leftrightarrow x \vdash w : A^\perp, x : A} \qquad \frac{P \vdash \Gamma, x : A \qquad Q \vdash \Delta, x : A^\perp}{\nu x.(P \mid Q) \vdash \Gamma, \Delta} \qquad \frac{P \vdash \Gamma, y : A \qquad Q \vdash \Delta, x : B}{x[y].(P \mid Q) \vdash \Gamma, \Delta, x : A \otimes B}$$

$$\frac{R \vdash \Theta, y : A, x : B}{x(y).R \vdash \Theta, x : A \parr B} \qquad \frac{P \vdash \Gamma, x : A_i}{x[l_i].P \vdash \Gamma, x : \oplus\{l_i : A_i\}_i} \qquad \frac{\{Q_i \vdash \Delta, x_i : A_i\}_i}{x.\mathsf{case}\ \{l_i.Q_i\}_i \vdash \Delta, x : \&\{l_i : A_i\}_i}$$

$$\frac{P \vdash ?\Gamma, y : A}{!x(y).P \vdash ?\Gamma, x : !A} \qquad \frac{Q \vdash \Delta, y : A}{?x[y].Q \vdash \Delta, x : ?A} \qquad \frac{Q \vdash \Delta}{Q \vdash \Delta, x : ?A} \qquad \frac{Q \vdash \Delta, x : ?A, x' : ?A}{Q[x/x'] \vdash \Delta, x : ?A}$$

$$\frac{P \vdash \Gamma, x : B[A/X]}{x[A].P \vdash \Gamma, x : \exists X.B} \qquad \frac{Q \vdash \Delta, x : B \qquad X \notin \Delta}{x(X).Q \vdash \Delta, x : \forall X.B} \qquad \frac{}{x[].0 \vdash x : 1} \qquad \frac{P \vdash \Gamma}{x().P \vdash \Gamma, x : \perp}$$

Figure 2: Typing rules for CP

Formally, this is a translation on derivations. We write type annotations to indicate $\to$ introduction and elimination. For all other cases, it is unambiguous to give the translation on plain term syntax. Each introduction form translates to an interface $\mathsf{fork}\ z.M$ of type $\overline{S}$, where $M : \mathsf{end}_!$ provides the implementation, with $z : S$ bound in M. We can extend the translation on types to a translation on contexts:

$$(x_1 : T_1, \dots, x_n : T_n)^\star = x_1 : (T_1)^\star, \dots, x_n : (T_n)^\star$$

Finally, it is straightforward to verify that our translation preserves typing.

Theorem 1. *If $\Phi \vdash M : T$ then $(\Phi)^\star \vdash (M)^\star : (T)^\star$.*

4 From HGVπ to CP

We present the typing rules of CP in Figure 2. Due to lack of space we omit the definition of the cut relation $\longrightarrow$. A detailed description of CP, including the cut rules, can be found in Wadler's work [8]. Note that the propositions of CP are exactly those of classical linear logic, as are the cut rules (if we ignore the terms). Thus, CP enjoys all of the standard meta theoretic properties of classical linear logic, including confluence and weak normalisation. A minor syntactic difference between our presentation and Wadler's is that our sum ($\oplus$) and choice ($\&$) types are n-ary, matching the corresponding session types in HGV, whereas he presents binary and nullary versions of sum and choice. Duality on CP types ($(-)^\perp$) is standard:

$$\begin{array}{lllll} (A \otimes B)^\perp = A^\perp \parr B^\perp & (\oplus\{l_i : A_i\}_i)^\perp = \&\{l_i : {A_i}^\perp\}_i & 1^\perp = \perp & (\exists X.B)^\perp = \forall X.B^\perp & (!A)^\perp = ?A^\perp \\ (A \parr B)^\perp = A^\perp \otimes B^\perp & (\&\{l_i : A_i\}_i)^\perp = \oplus\{l_i : {A_i}^\perp\}_i & \perp^\perp = 1 & (\forall X.B)^\perp = \exists X.B^\perp & (?A)^\perp = !A^\perp \end{array}$$

We now give a translation from HGVπ to CP. Post composing this with the embedding of HGV in HGVπ yields a semantics for HGV. The translation on session types is as follows:

$$\begin{array}{llll} [\![!T.S]\!] = [\![T]\!]^\perp \otimes [\![S]\!] & [\![\oplus\{l_i : S_i\}_i]\!] = \oplus\{l_i : [\![S_i]\!]\}_i & [\![\flat S]\!] = ![\![S]\!] & [\![![X].S]\!] = \exists X.[\![S]\!] \\ [\![?T.S]\!] = [\![T]\!] \parr [\![S]\!] & [\![\&\{l_i : S_i\}_i]\!] = \&\{l_i : [\![S_i]\!]\}_i & [\![\sharp S]\!] = ?[\![S]\!] & [\![?[X].S]\!] = \forall X.[\![S]\!] \\ [\![\mathsf{end}_!]\!] = 1 & [\![\mathsf{end}_?]\!] = \perp & [\![X]\!] = X & [\![\overline{X}]\!] = X^\perp \end{array}$$

The translation is homomorphic except for output, where the output type is dualised. This accounts for the discrepancy between $\overline{!T.S} = ?T.\overline{S}$ and $(A \otimes B)^\perp = A^\perp \parr B^\perp$.

The translation on terms is formally specified as a CPS translation on derivations as in Wadler's presentation. We provide the full translations of weakening and contraction for $\mathsf{end}_?$, as these steps are implicit in the syntax of HGV terms. The other constructs depend only on the immediate syntactic structure, so we abbreviate their translations as mappings on plain terms:

$$\left[\!\!\left[\frac{\Phi \vdash N : S}{\Phi, x : \mathsf{end}_? \vdash N : S} \right]\!\!\right] z = \frac{[\![N]\!]z \vdash [\![\Phi]\!], z : [\![S]\!]^\perp}{x().[\![N]\!]z \vdash [\![\Phi]\!], x : \perp, z : [\![S]\!]^\perp}$$

$$\left[\!\!\left[\frac{\Phi, x : \mathsf{end}_?, x' : \mathsf{end}_? \vdash N : S}{\Phi, x : \mathsf{end}_? \vdash N[x/x'] : S} \right]\!\!\right] z = \frac{[\![N]\!]z \vdash [\![\Phi]\!], x : \perp, x' : \perp, z : [\![S]\!]^\perp}{\nu x'.([\![N]\!]z \mid x'[].0) \vdash [\![\Phi]\!], x : \perp, z : [\![S]\!]^\perp}$$

$$\begin{aligned}
[\![x]\!]z &= x \leftrightarrow z \\
[\![\mathsf{send}\ M\ N]\!]z &= \nu x.(x[y].([\![M]\!]y \mid x \leftrightarrow z) \mid [\![N]\!]x) \\
[\![\mathsf{let}\ (x,y) = \mathsf{receive}\ M\ \mathsf{in}\ N]\!]z &= \nu y.([\![M]\!]y \mid y(x).[\![N]\!]z) \\
[\![\mathsf{select}\ l\ M]\!]z &= \nu x.([\![M]\!]x \mid x[l].x \leftrightarrow z) \\
[\![\mathsf{case}\ M\ \mathsf{of}\ \{l_i(x).N_i\}_i]\!]z &= \nu x.([\![M]\!]x \mid x.\mathsf{case}\ \{l_i.[\![N_i]\!]z\}_i) \\
[\![\mathsf{fork}\ x.M]\!]z &= \nu x.(\nu y.([\![M]\!]y \mid y[].0) \mid x \leftrightarrow z) \\
[\![\mathsf{link}\ M\ N]\!]z &= z().\nu x.([\![M]\!]x \mid [\![N]\!]x) \\
[\![\mathsf{sendType}\ S\ M]\!]z &= \nu x.([\![M]\!]x \mid x[[\![S]\!]].x \leftrightarrow z) \\
[\![\mathsf{receiveType}\ X.M]\!]z &= \nu x.([\![M]\!]x \mid x(X).x \leftrightarrow z) \\
[\![\mathsf{serve}\ y.M]\!]z &= !z(y).\nu x.([\![M]\!]x \mid x[].0) \\
[\![\mathsf{request}\ M]\!]z &= \nu x.([\![M]\!]x \mid ?x[y].y \leftrightarrow z)
\end{aligned}$$

Channel z provides a continuation, consuming the output of the process representing the original HGVπ term. The translation on contexts is pointwise.

$$[\![x_1 : T_1, \ldots, x_n : T_n]\!] = x_1 : [\![T_1]\!], \ldots, x_n : [\![T_n]\!]$$

As with the translation from HGV to HGVπ, we can show that this translation preserves typing.

Theorem 2. *If* $\Phi \vdash M : S$ *then* $[\![M]\!]z \vdash [\![\Phi]\!], z : [\![S]\!]^\perp$.

5 From CP to HGVπ

We now present the translation $(\!|-|\!)$ from CP to HGVπ. The translation on types is as follows:

$$\begin{array}{rlrlrlrl}
(\!|A \otimes B|\!) &= !\overline{(\!|A|\!)}.(\!|B|\!) & (\!|\oplus\{l_i : A_i\}_i|\!) &= \oplus\{l_i : (\!|A_i|\!)\}_i & (\!|\exists X.A|\!) &= ![X].(\!|A|\!) & (\!|?A|\!) &= \sharp(\!|A|\!) \\
(\!|A \parr B|\!) &= ?(\!|A|\!).(\!|B|\!) & (\!|\&\{l_i : A_i\}_i|\!) &= \&\{l_i : (\!|A_i|\!)\}_i & (\!|\forall X.A|\!) &= ?[X].(\!|A|\!) & (\!|!A|\!) &= \flat(\!|A|\!) \\
(\!|1|\!) &= \mathsf{end}_! & (\!|\perp|\!) &= \mathsf{end}_? & (\!|X|\!) &= X & (\!|X^\perp|\!) &= \overline{X}
\end{array}$$

The translation on terms makes use of let expressions to simplify the presentation; these are expanded to HGVπ as follows:

$$\mathsf{let}\ x = M\ \mathsf{in}\ N \equiv ((\lambda x.N)M)^\star \equiv \mathsf{send}\ M\ (\mathsf{fork}\ z.\mathsf{let}\ (x,z) = \mathsf{receive}\ z\ \mathsf{in}\ \mathsf{link}\ N\ z).$$

$$
\begin{aligned}
(\!|x[y].(P \mid Q)|\!) &= \mathsf{let}\ x = \mathsf{send}\ (\mathsf{fork}\ y.(\!|P|\!))\ x\ \mathsf{in}\ (\!|Q|\!) \\
(\!|x(y).P|\!) &= \mathsf{let}\ (y, x) = \mathsf{receive}\ x\ \mathsf{in}\ (\!|P|\!) \\
(\!|x[l].P|\!) &= \mathsf{let}\ x = \mathsf{select}\ l\ x\ \mathsf{in}\ (\!|P|\!) \\
(\!|x.\mathsf{case}\ \{l_i.P_i\}_i|\!) &= \mathsf{case}\ x\ \mathsf{of}\ \{l_i(x).(\!|P_i|\!)\}_i \\
(\!|x[].0|\!) &= x \\
(\!|x().P|\!) &= (\!|P|\!) \\
(\!|\nu x.(P \mid Q)|\!) &= \mathsf{let}\ x = \mathsf{fork}\ x.(\!|P|\!)\ \mathsf{in}\ (\!|Q|\!) \\
(\!|x \leftrightarrow y|\!) &= \mathsf{link}\ x\ y \\
(\!|x[A].P|\!) &= \mathsf{let}\ x = \mathsf{sendType}\ (\!|A|\!)\ x\ \mathsf{in}\ (\!|P|\!) \\
(\!|x(X).P|\!) &= \mathsf{let}\ x = \mathsf{receiveType}\ X.x\ \mathsf{in}\ (\!|P|\!) \\
(\!|!s(x).P|\!) &= \mathsf{link}\ s\ (\mathsf{serve}\ x.(\!|P|\!)) \\
(\!|?s[x].P|\!) &= \mathsf{let}\ x = \mathsf{fork}\ x.\mathsf{link}\ (\mathsf{request}\ s)\ x\ \mathsf{in}\ (\!|P|\!)
\end{aligned}
$$

Again, we can extend the translation on types to a translation on contexts, and show that the translation preserves typing.

Theorem 3. *If $P \vdash \Gamma$ then $(\!|\Gamma|\!) \vdash (\!|P|\!) : \mathsf{end}_!$.*

6 Correctness

If we extend $[\![-]\!]$ to non-session types, as in Wadler's original presentation (Figure 3), then it is straightforward to show that this monolithic translation factors through $(-)^\star$.

Theorem 4. *$[\![(M)^\star]\!]z \longrightarrow^* [\![M]\!]z$ (where $\longrightarrow^*$ is the transitive reflexive closure of $\longrightarrow$).*

The key soundness property of our translations is that if we translate a term from CP to HGVπ and back, then we obtain a term equivalent to the one we started with.

Theorem 5. *If $P \vdash \Gamma$ then $\nu z.(z[].0 \mid [\![(\!|P|\!)]\!]z) \longrightarrow^* P$.*

Together, Theorem 4 and 5 tell us that HGV, HGVπ, and CP are equally expressive, in the sense that every X program can always be translated to an equivalent Y program, where $X, Y \in \{\text{HGV, HGV}\pi\text{, CP}\}$.

Here our notion of expressivity is agnostic to the nature of the translations. It is instructive also to consider Felleisen's more refined notion of expressivity [5]. Both $(-)^\star$ and $(\!|-|\!)$ are local translations, thus both HGV and CP are *macro-expressible* [5] in HGVπ. However, the need for a global CPS translation from HGVπ to CP illustrates that HGVπ is not macro-expressible in CP; hence HGVπ is more expressive, in the Felleisen sense, than CP.

7 Conclusions and Future Work

We have proposed a session-typed functional language, HGV, building on similar languages of Wadler [8] and of Gay and Vasconcelos [6]. We have shown that HGV is sufficient to encode arbitrary linear logic proofs, completing the correspondence between linear logic and session types. We have also given an embedding of all of HGV into its session-typed fragment, simplifying translation from HGV to CP.

Dardha et al [4] offers an alternative foundation for session types through a CPS translation of π-calculus with session types into a linear π-calculus. There appear to be strong similarities between their CPS translation and ours. We would like to make the correspondence precise by studying translations between their systems and ours.

Types

$$[\![T \multimap U]\!] = ([\![T]\!]^\perp \parr [\![U]\!])^\perp$$

$$[\![T \to U]\!] = (!([\![T]\!]^\perp \parr [\![U]\!]))^\perp$$

$$[\![T \otimes U]\!] = ([\![T]\!] \otimes [\![U]\!])^\perp$$

Terms

$$\begin{aligned}
[\![\lambda x.N]\!]z &= z(x).[\![N]\!]z \\
[\![L\ M]\!]z &= \nu y.([\![L]\!]y \mid y[x].([\![M]\!]x \mid y \leftrightarrow z)) \\
[\![L : T \to U]\!]z &= !z(y).[\![L]\!]y \\
[\![L : T \multimap U]\!]z &= \nu y.([\![L]\!]y \mid ?y[x].x \leftrightarrow z) \\
[\![(M, N)]\!]z &= z[y].([\![M]\!]y \mid [\![N]\!]z) \\
[\![\mathsf{let}\ (x, y) = M\ \mathsf{in}\ N]\!]z &= \nu y.([\![M]\!]y \mid y(x).[\![N]\!]z)
\end{aligned}$$

The outer duals appear in the type translation because, as in Section 3, we must expose *interfaces* rather than implementations of simulated types. As in the definition of $(-)^\star$ in Section 3, we write type annotations to indicate $\to$ introduction and elimination.

Figure 3: Extension of $[\![-]\!]$ to non-session types

In addition we highlight several other areas of future work. First, the semantics of HGV is given only by cut elimination in CP. We would like to give HGV a semantics directly, in terms of reductions of configurations of processes, and then prove a formal correspondence with cut elimination in CP. Second, replication has limited expressive power compared to recursion; in particular, it cannot express services whose behaviour changes over time or in response to client requests. We believe that the study of fixed points in linear logic provides a mechanism to support more expressive recursive behaviour without sacrificing the logical interpretation of HGV. Finally, as classical linear logic proofs, and hence CP processes, enjoy confluence, HGV programs are deterministic. We hope to identify natural extensions of HGV that give rise to non-determinism, and thus allow programs to exhibit more interesting concurrent behaviour, while preserving the underlying connection to linear logic.

Acknowledgements We would like to thank Philip Wadler for his suggestions on the direction of this work, and for his helpful feedback on the results. This work was funded by EPSRC grant number EP/K034413/1.

References

[1] Samson Abramsky. Proofs as processes. MFPS '92, pages 5–9. Elsevier, 1994.

[2] Gianluigi Bellin and Philip J. Scott. On the π-Calculus and linear logic. *Theoretical Computer Science*, 135(1):11–65, 1994.

[3] Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In *International Conference on Concurrency Theory*, CONCUR '10, pages 222–236, 2010.

[4] Ornela Dardha, Elena Giachino, and Davide Sangiorgi. Session types revisited. In *PPDP*, pages 139–150, 2012.

[5] Matthias Felleisen. On the expressive power of programming languages. *Sci. Comput. Program.*, 17(1–3):35–75, 1991.

[6] Simon J. Gay and Vasco T. Vasconcelos. Linear type theory for asynchronous session types. *Journal of Functional Programming*, 20(01):19–50, 2010.

[7] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, January 1987.

[8] Philip Wadler. Propositions as sessions. In *Proceedings of the 17th ACM SIGPLAN International Conference on Functional Programming*, ICFP '12, pages 273–286. ACM, 2012.

A Cut reduction for CP

Structural cut equivalences

$$\begin{aligned}
x \leftrightarrow y &\equiv y \leftrightarrow x \\
\nu x.(P \mid Q) &\equiv \nu x.(Q \mid P) \\
\nu y.(\nu x.(P \mid Q) \mid R) &\equiv \nu x.(P \mid \nu y.(Q \mid R))
\end{aligned}$$

The cut relation $\longrightarrow$ is interpreted modulo both α-equivalence and structural cut equivalence. It is given by the compatible closure of the cut rules and commuting conversions defined below. We write $\longrightarrow^*$ for the transitive reflexive closure of $\longrightarrow$.

Cut rules

$$\begin{aligned}
\nu x.(w \leftrightarrow x \mid P) &\longrightarrow P[w/x] \\
\nu x.(x[y].(P \mid Q) \mid x(y).R) &\longrightarrow \nu y.(P \mid \nu x.(Q \mid R)) \\
\nu x.(x[l_j].P \mid x.\mathsf{case}\ \{l_i.Q_i\}_i) &\longrightarrow \nu x.(P \mid Q_j) \\
\nu x.(!x(y).P \mid ?x[y].Q) &\longrightarrow \nu y.(P \mid Q) \\
\nu x.(!x(y).P \mid Q) &\longrightarrow Q, \quad x \notin FV(Q) \\
\nu x.(!x(y).P \mid Q[x/x']) &\longrightarrow \nu x.(!x(y).P \mid \nu x'.(!x'(y).P \mid Q)) \\
\nu x.(x[A].P \mid x(X).Q) &\longrightarrow \nu x.(P \mid Q[A/X]) \\
\nu x.(x[].0 \mid x().P) &\longrightarrow P
\end{aligned}$$

Commuting conversions

$$\begin{aligned}
\nu z.(x[y].(P \mid Q) \mid R) &\longrightarrow x[y].(\nu z.(P \mid R) \mid Q), \quad z \in FV(P) \\
\nu z.(x[y].(P \mid Q) \mid R) &\longrightarrow x[y].(P \mid \nu z.(Q \mid R)), \quad z \in FV(Q) \\
\nu z.(x(y).P \mid Q) &\longrightarrow x(y).\nu z.(P \mid Q) \\
\nu z.(x[l].P \mid Q) &\longrightarrow x[l].\nu z.(P \mid Q) \\
\nu z.(x.\mathsf{case}\ \{l_i.Q_i\}_i \mid R) &\longrightarrow x.\mathsf{case}\ \{l_i.\nu z.(Q_i \mid R)\}_i \\
\nu z.(!x(y).P \mid Q) &\longrightarrow !x(y).\nu z.(P \mid Q) \\
\nu z.(?x[y].P \mid Q) &\longrightarrow ?x[y].\nu z.(P \mid Q) \\
\nu z.(x[A].P \mid Q) &\longrightarrow x[A].\nu z.(P \mid Q) \\
\nu z.(x(X).P \mid Q) &\longrightarrow x(X).\nu z.(P \mid Q) \\
\nu z.(x().P \mid Q) &\longrightarrow x().\nu z.(P \mid Q)
\end{aligned}$$