Capable GV: Capabilities for Session Types in GV*

Magdalena J. Latifa[1] and Ornela Dardha[1]

[1] University of Glasgow
2398248L@student.gla.ac.uk
[2] University of Glasgow
ornela.dardha@glasgow.ac.uk

Abstract

This is an ongoing work which introduces Capable GV (CGV)–a functional calculus with binary session types that uses *capabilities* and allows channel sharing. Capabilities allow to split channels into two entities: a linear capability and an unrestricted channel endpoint. As channels themselves are unrestricted, this allows sharing, and thus increases expressivity wrt previous work. Orthogonality, CGV allows cyclic processes at the cost of losing deadlock freedom. Our aim is to restore deadlock freedom in the future by using priorities, as in works by Kokke and Dardha [6, 5]. This work presents the language terms, types and typing rules. Semantics and safety results are work in progress.

1 Introduction

Good Variation (GV) is a functional calculus with binary session types that allows protocol-compliant concurrent communication. It was originally introduced by Wadler [9], based on [4]. Since then, many extensions to GV have been developed, including Exceptional GV [3], Priority GV [6] and Hypersequent GV [2], with the aim of improving its expressivity and guarantees, while taking advantage of the benefits that GV provides: higher-order functions, separation of run-time configuration and a more natural fit for implementations. *Capabilities* [8] are a method of allowing channel sharing. This method relies on splitting channels into two disjoint components, which allows channels to be safely shared. In this work, we develop an extension to GV that allows channel sharing, consequently increasing the expressivity of the system.

2 Capable GV

We present the statics of Capable GV (CGV), a GV-based functional language with shareable session types. Channel sharing in CGV is achieved by introducing unrestricted channel endpoints and linear capabilities that are managed by a flow-sensitive type-and-effect system. As linearity is enforced via capabilities, sharing does not violate communication safety. While CGV allows for greater expressivity, it comes at the cost of deadlock-freedom as the system allows cyclic processes. CGV types are defined by the following grammar:

$$\begin{array}{rcl} S & ::= & !T.S \mid ?T.S \mid \mathbf{end} \\ T, U & ::= & T \times U \mid T + U \mid \mathbf{1} \mid tr(\rho) \mid [\rho(S)] \mid T\ (C_1) \multimap (C_2)\ U \\ \Gamma, \Delta & ::= & \varnothing \mid \Gamma, x : T \\ C & ::= & \varnothing \mid C \otimes \rho(S) \end{array}$$

Constructs $tr(\rho)$ and $\rho(S)$ are the core of CGV and represent the separation of the channel endpoint and the capability of using it. Tracked type $tr(\rho)$ specifies that the channel endpoint

*Work supported by the EU H2020 MSCA RISE project BehAPI ID 778233.

is controlled by capability ρ. Capability ρ exists in a capability set C in the form $\rho(S)$ which specifies the channel's session type S. Additionally, a capability can be packed into a pack type $[\rho(S)]$ and subsequently relayed to another thread. In order to accommodate for capabilities in the type-and-effect system, functions have the type $T\ (C_1) \multimap (C_2)\ U$ which denotes a linear function $T \multimap U$ where the function body requires pre-evaluation capability set C_1 and produces capability set C_2 when evaluated. The remaining types are standard session types or linear λ-calculus types.

CGV terms are defined by the following grammar:

$$
\begin{array}{rcl}
V, W & ::= & () \mid x \mid \lambda x.M \mid (V, W) \mid \mathbf{inl}\ V \mid \mathbf{inr}\ V \\
L, M, N & ::= & V\ W \mid \mathbf{let}\ x = M\ \mathbf{in}\ N \mid \mathbf{let}\ (x, y) = V\ \mathbf{in}\ M \mid \mathbf{let}\ () = V\ \mathbf{in}\ M \\
& \mid & \mathbf{case}\ L\ \{\mathbf{inl}\ x \mapsto M;\ \mathbf{inr}\ y \mapsto N\} \mid \mathbf{return}\ V \\
& \mid & \mathbf{new} \mid \mathbf{send}\ V \mid \mathbf{recv}\ V \mid \mathbf{close}\ V \mid \mathbf{pack}\ V \mid \mathbf{unpack}\ V \mid \mathbf{spawn}\ M\ N
\end{array}
$$

Terms $\mathbf{pack}\ V$ and $\mathbf{unpack}\ V$ are unique to CGV and allow "inactivating" and "reactivating" channels by packing and unpacking capabilities. Terms $\mathbf{send}\ V$ and $\mathbf{recv}\ V$ are standard to GV but they no longer return a copy of the channel since channels are unrestricted. Terms $\mathbf{close}\ V$, $\mathbf{new}$ and $\mathbf{spawn}\ M\ N$ are analogous to Priority GV's terms [6] with the change of $\mathbf{new}$ also creating capabilities for the new channels and $\mathbf{spawn}$ requiring a packed capability to initialise the newly spawned thread with. The rest of the terms are standard linear λ-calculus terms.

Typing rules T-Recv and T-Pack demonstrate the behaviour of capabilities in CGV. For T-Recv, in order to receive a message of type T on a channel of type $tr(\rho)$, the capability of using the endpoint $\rho(?T.S)$ needs to be in the pre-evaluation capability set. After this communication takes place, the capability must update the session type; hence the post-evaluation capability set must contain the capability in the form $\rho(S)$. T-Pack specifies the behaviour of packing the capability in order to relay it to another thread. In order to pack the capability of channel of type $tr(\rho)$, the capability $\rho(S)$ needs to be in the pre-evaluation capability set. After the channel is packed, the capability $\rho(S)$ is removed from the capability set and is instead "inactivated" and expressed in the pack type $[\rho(S)]$. The session type of this channel cannot change until the capability is unpacked and the channel becomes active again, which is key to ensuring linearity of communication.

$$
\text{T-Recv}\ \frac{\Gamma \vdash V : tr(\rho)}{\Gamma; C \otimes \rho(?T.S) \vdash \mathbf{recv}\ V : T \triangleright C \otimes \rho(S)} \qquad \text{T-Pack}\ \frac{\Gamma \vdash V : tr(\rho)}{\Gamma; C \otimes \rho(S) \vdash \mathbf{pack}\ V : [\rho(S)] \triangleright C}
$$

3 Conclusions and Future Work

We have presented our work in progress on CGV—Capable GV. At the time of writing, we have completed syntax of types and terms as well as typing rules and we are working on finalising the operational semantics and type safety results. CGV uses capabilities in order to introduce channel sharing and provide greater expressivity. This is achieved via tracking capabilities in a type-and-effect system and making the channel endpoints unrestricted. While this approach allows sharing, it also reintroduces deadlocks. Hence, a potential area for further work will be the combination of CGV with Priority GV [6, 5] to restore deadlock-freedom and tie the system back to logic. Additionally, we aim to explore channel sharing through different means, namely via manifest sharing [1].

References

[1] Stephanie Balzer, Bernardo Toninho, and Frank Pfenning. Manifest deadlock-freedom for shared session types. In *Programming Languages and Systems - 28th European Symposium on Programming, ESOP 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, volume 11423 of *Lecture Notes in Computer Science*, pages 611–639. Springer, 2019.

[2] Simon Fowler, Wen Kokke, Ornela Dardha, Sam Lindley, and J. Garrett Morris. Separating sessions smoothly. In *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, volume 203 of *LIPIcs*, pages 36:1–36:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[3] Simon Fowler, Sam Lindley, J. Garrett Morris, and Sára Decova. Exceptional asynchronous session types: session types without tiers. *Proc. ACM Program. Lang.*, 3(POPL):28:1–28:29, 2019.

[4] Simon J. Gay and Vasco Thudichum Vasconcelos. Linear type theory for asynchronous session types. *J. Funct. Program.*, 20(1):19–50, 2010.

[5] Wen Kokke and Ornela Dardha. Deadlock-free session types in linear haskell. In *Haskell 2021: Proceedings of the 14th ACM SIGPLAN International Symposium on Haskell, Virtual Event, Korea, August 26-27, 2021*, pages 1–13. ACM, 2021.

[6] Wen Kokke and Ornela Dardha. Prioritise the best variation. In *Formal Techniques for Distributed Objects, Components, and Systems - 41st IFIP WG 6.1 International Conference, FORTE 2021, Held as Part of the 16th International Federated Conference on Distributed Computing Techniques, DisCoTec 2021, Valletta, Malta, June 14-18, 2021, Proceedings*, volume 12719 of *Lecture Notes in Computer Science*, pages 100–119. Springer, 2021.

[7] Vasco T. Vasconcelos. Fundamentals of session types. *Inf. Comput.*, 217:52–70, 2012.

[8] A. Laura Voinea, Ornela Dardha, and Simon J. Gay. Resource sharing via capability-based multiparty session types. In *Integrated Formal Methods - 15th International Conference, IFM 2019, Bergen, Norway, December 2-6, 2019, Proceedings*, volume 11918 of *Lecture Notes in Computer Science*, pages 437–455. Springer, 2019.

[9] Philip Wadler. Propositions as sessions. In *ACM SIGPLAN International Conference on Functional Programming, ICFP'12, Copenhagen, Denmark, September 9-15, 2012*, pages 273–286. ACM, 2012.

A Context Split

Typing environments can be split according to rules in Figure 1 analogously to the work done by Vasconcelos [7].

That allows all channels to be unrestricted while preserving linearity for everything else.

$$\frac{}{\varnothing = \varnothing \circ \varnothing} \qquad \frac{\Gamma = \Gamma_1 \circ \Gamma_2 \qquad T = tr(\rho)}{\Gamma, x : T = (\Gamma_1, x : T) \circ (\Gamma_2, x : T)}$$

$$\frac{\Gamma = \Gamma_1 \circ \Gamma_2}{\Gamma, x : T = (\Gamma_1, x : T) \circ \Gamma_2} \qquad \frac{\Gamma = \Gamma_1 \circ \Gamma_2}{\Gamma, x : T = \Gamma_1 \circ (\Gamma_2, x : T)}$$

Figure 1: Context split.

B Static Typing Rules

Full static typing rules are presented in Figure 3 and Figure 3.

Values typing judgements

Typing judgements for values are of the form

$$\Gamma \vdash V : T$$

which states that *Under typing environment* Γ*, term* V *is of type* T.

Computations typing judgements

Typing judgements for computations are of the form

$$\Gamma; C \vdash M : T \triangleright C'$$

which states that *Under typing environment* Γ *and with capability set* C*, term* M *is of type* T *and produces capability set* C'.

$$\textsc{T-Var}\ \frac{}{x : T \vdash x : T} \qquad \textsc{T-Unit}\ \frac{}{\varnothing \vdash () : \mathbf{1}} \qquad \textsc{T-Lam}\ \frac{\Gamma, x : T; C \vdash M : U \triangleright C'}{\Gamma \vdash \lambda x.M : T\ (C) \multimap (C')\ U}$$

$$\textsc{T-PairVal}\ \frac{\Gamma \vdash V : T \qquad \Delta \vdash W : U}{\Gamma \circ \Delta \vdash (V, W) : T \times U}$$

$$\textsc{T-Inl}\ \frac{\Gamma \vdash V : T}{\Gamma \vdash \mathbf{inl}\ V : T + U} \qquad \textsc{T-Inr}\ \frac{\Gamma \vdash V : U}{\Gamma \vdash \mathbf{inr}\ V : T + U}$$

Figure 2: Static Typing Rules for Values.

$$\textsc{T-App}\ \frac{\Gamma \vdash V : T\ (C) \multimap (C')\ U \qquad \Delta \vdash W : T}{\Gamma \circ \Delta; C \vdash V\ W : U \triangleright C'} \qquad \textsc{T-Return}\ \frac{\Gamma \vdash V : T}{\Gamma; C \vdash \mathbf{return}\ V : T \triangleright C'}$$

$$\textsc{T-LetUnit}\ \frac{\Gamma \vdash V : \mathbf{1} \qquad \Delta; C \vdash M : T \triangleright C'}{\Gamma \circ \Delta; C \vdash \mathbf{let}\ () = V\ \mathbf{in}\ M : T \triangleright C'} \qquad \textsc{T-LetPair}\ \frac{\Gamma \vdash V : T \times T' \qquad \Delta, x : T, y : T'; C \vdash M : U \triangleright C'}{\Gamma \circ \Delta; C \vdash \mathbf{let}\ (x, y) = V\ \mathbf{in}\ M : U \triangleright C'}$$

$$\textsc{T-LetBind}\ \frac{\Gamma; C \vdash M : T \triangleright C' \qquad \Delta, x : T; C' \vdash N : U \triangleright C''}{\Gamma \circ \Delta; C \vdash \mathbf{let}\ x = M\ \mathbf{in}\ N : U \triangleright C''}$$

$$\textsc{T-CaseSum}\ \frac{\Gamma \vdash L : T + T' \qquad \Delta, x : T; C \vdash M : U \triangleright C' \qquad \Delta, y : T'; C \vdash N : U \triangleright C'}{\Gamma \circ \Delta; C \vdash \mathbf{case}\ L\ \{\mathbf{inl}\ x \mapsto M;\ \mathbf{inr}\ y \mapsto N\} : U \triangleright C'}$$

$$\textsc{T-New}\ \frac{}{\varnothing; C \vdash \mathbf{new} : tr(\rho) \times tr(\rho') \triangleright C \otimes \rho(S) \otimes \rho'(\overline{S})}$$

$$\textsc{T-Spawn}\ \frac{\Gamma; C \vdash M : [\rho(S)] \triangleright C' \qquad \Delta; \rho(S) \vdash N : \mathbf{1} \triangleright \varnothing}{\Gamma \circ \Delta; C \vdash \mathbf{spawn}\ MN : \mathbf{1} \triangleright C'} \qquad \textsc{T-Close}\ \frac{\Gamma \vdash V : tr(\rho)}{\Gamma; C \otimes \rho(\mathbf{end}) \vdash \mathbf{close}\ V : \mathbf{1} \triangleright C}$$

$$\textsc{T-Send}\ \frac{\Gamma \vdash V : T \times tr(\rho)}{\Gamma; C \otimes \rho(!T.S) \vdash \mathbf{send}\ V : \mathbf{1} \triangleright C \otimes \rho(S)} \qquad \textsc{T-Recv}\ \frac{\Gamma \vdash V : tr(\rho)}{\Gamma; C \otimes \rho(?T.S) \vdash \mathbf{recv}\ V : T \triangleright C \otimes \rho(S)}$$

$$\textsc{T-Pack}\ \frac{\Gamma \vdash V : tr(\rho)}{\Gamma; C \otimes \rho(S) \vdash \mathbf{pack}\ V : [\rho(S)] \triangleright C} \qquad \textsc{T-Unpack}\ \frac{\Gamma \vdash V : [\rho(S)]}{\Gamma; C \vdash \mathbf{unpack}\ V : \mathbf{1} \triangleright C \otimes \rho(S)}$$

Figure 3: Static Typing Rules for Computations.