A Lambda Calculus for Gödel–Dummett Logic Capturing Waitfreedom

Yoichi Hirai

The University of Tokyo, JSPS Research Fellow yh@is.s.u-tokyo.ac.jp

Abstract. We propose a typed lambda calculus based on Avron's hypersequent calculus for Gödel–Dummett logic. This calculus turns out to model waitfree computation. Besides strong normalization and non-abortfullness, we give soundness and completeness of the calculus against the typed version of waitfree protocols. The calculus is not only proof theoretically interesting, but also valuable as a basis for distributed programming languages.

1 Introduction

The Curry–Howard isomorphism [20] is surprising because the same method works for two different purposes: a logical purpose of removing redundancy from proofs and a computational purpose of finding a class of terminating programs. We extend this surprise to Gödel–Dummett logic and waitfreedom. Gödel–Dummett logic [9] is one of the intermediate logics between classical and intuitionistic logics. Waitfreedom [14, 19] is a class of distributed computation without synchronization among processes.

We connect Gödel–Dummett logic and waitfreedom using Avron's hypersequent calculus [2]. We respond to his suggestion:

> it seems to us extremely important to determine the exact computational content of them [intermediate logics] — and to develop corresponding 'λ-calculi' —Avron [2].

Differently from intuitionistic logic, Gödel–Dummett logic validates all formulae of the form $(\varphi \supset \psi) \vee (\psi \supset \varphi)$. Our aim is building a typed lambda calculus with some terms witnessing those formulae. Such a term $M\colon (\varphi \supset \psi) \vee (\psi \supset \varphi)$ must choose $M \rightsquigarrow^* \mathsf{inl}\,(\cdots)$ or $M \rightsquigarrow^* \mathsf{inr}\,(\cdots)$. We devise a nondeterministic λ-calculus in Sect. 2.

Waitfreedom is a class of distributed computation where processes cannot wait for other processes. When two processes try to exchange information, the faster process can pass information to the slower one, but not always vice versa because the slower process might start after the faster one finishes. So, the computation is nondeterministic. The contribution of this paper is capturing this nondeterminism using the nondeterministic λ-calculus for Gödel–Dummett logic: in Sect. 4, we show that the λ-terms in the calculus can solve a typed input-output problems if and only if it is waitfreely solvable.

2 λ-GD

We first present a proof system for Gödel–Dummet logic. Then we turn the proof system into typing rules for λ-terms of λ-GD, give a set of reductions and prove strong-normalization and non-abortfullness.

2.1 Logic

Let us assume a countably infinite set of *propositional variables*. We define *local formulae* φ, ψ by the following BNF, where I is a propositional variable[1]:

$$\varphi, \psi ::= \bot \mid I \mid (\varphi \supset \psi) \mid (\varphi \wedge \psi) \mid (\varphi \vee \psi) \ .$$

Further, we define *global formulae* φ^+, ψ^+ with the following BNF:

$$\varphi^+, \psi^+ ::= [i]\varphi \mid (\varphi^+ \wedge \psi^+) \mid (\varphi^+ \vee \psi^+)$$

where i is a natural number (representing a process). The unary operators $[0], [1], \ldots$ are called *modalities*. We omit parentheses following the usual convention. Informally, the local formulae describe datatypes used by each process. The global formulae describe inputs or outputs of all processes together.

A *context* (denoted by Γ and Δ possibly subscripted) is a potentially empty finite sequence of global formulae. A *sequent* $\Gamma \vdash \varphi^+$ is a pair of a context and a global formula. A *hypersequent* is a finite sequence of sequents.

The underlying logic has the derivation rules in Fig. 1. If we omit all the modalities, these rules characterize Gödel–Dummett logic. However, the modalities have at least some sense: while $([0]\varphi \supset [0]\psi) \vee ([1]\psi \supset [1]\varphi)$ is provable, $([0]\varphi \supset [1]\psi) \vee ([0]\psi \supset [1]\varphi)$ is not. A semantics will be given in Sect. 4.

2.2 Term Assignment

We assume distinct, countably infinite sets of *variables*, *locations* and *processes*. Locations are denoted by l; process $i, j, \ldots$ and variables $x, y, \ldots$. Later, locations will be used to specify a pair of stores holding terms. Like in the λ-calculus, some terms reduces to other terms, but in this calculus, terms may interact with the store (like a program written in Haskell or OCaml does with an i-var). This behavior will be shown later in the definition of reductions.

We define *terms* $\mathcal{T}$ by the BNF where Γ is a sequence of variables:

$$\begin{aligned}\mathcal{T} ::= x \mid \overrightarrow{l^i_\Gamma}(\mathcal{T}) \mid \overleftarrow{l^i_\Gamma}(\mathcal{T}) \mid \langle \mathcal{T}, \mathcal{T}\rangle^{\mathsf{g}} \mid \pi^{\mathsf{g}}_{\mathrm{l}}(\mathcal{T}) \mid \pi^{\mathsf{g}}_{\mathrm{r}}(\mathcal{T}) \mid \mathsf{inl}^{\mathsf{g}}(\mathcal{T}) \mid \mathsf{inr}^{\mathsf{g}}(\mathcal{T}) \mid \\ \mathsf{match}^{\mathsf{g}}\, \mathcal{T}\, \mathsf{of}\, \mathsf{inl}^{\mathsf{g}}(x).\mathcal{T}/\mathsf{inr}^{\mathsf{g}}(y).\mathcal{T} \mid [\mathcal{T}, \mathcal{T}] \mid \mathsf{abort} \mid \pi_{\mathrm{l}}(\mathcal{T}) \mid \pi_{\mathrm{r}}(\mathcal{T}) \mid \langle \mathcal{T}, \mathcal{T}\rangle \mid \\ \mathsf{inl}(\mathcal{T}) \mid \mathsf{inr}(\mathcal{T}) \mid \lambda x.\mathcal{T} \mid (\mathcal{T}\mathcal{T}) \mid \mathsf{match}\, \mathcal{T}\, \mathsf{of}\, \mathsf{inl}(x).\mathcal{T}/\mathsf{inr}(y).\mathcal{T} \ .\end{aligned}$$

[1] We include $\bot$ because Gödel–Dummett logic has it although $\bot$ is not necessary for us to encode waitfree computation.

External Rules

$$\text{com'}\ \frac{\mathcal{H} \mid \varGamma, \varDelta \vdash [i]\varphi \qquad \mathcal{H} \mid \varGamma, \varDelta \vdash [j]\psi}{\mathcal{H} \mid \varGamma \vdash [i]\psi \mid \varDelta \vdash [j]\varphi} \qquad \text{EW}\ \frac{\mathcal{H}^+}{\mathcal{H}^+ \mid \varGamma \vdash \varphi^+}$$

$$\text{EC}\ \frac{\mathcal{H} \mid \varGamma \vdash \varphi^+ \mid \varGamma \vdash \varphi^+}{\mathcal{H} \mid \varGamma \vdash \varphi^+} \qquad \text{EE}\ \frac{\mathcal{H} \mid \varGamma \vdash \varphi^+ \mid \varDelta \vdash \psi^+ \mid \mathcal{H}'}{\mathcal{H} \mid \varDelta \vdash \psi^+ \mid \varGamma \vdash \varphi^+ \mid \mathcal{H}'}$$

Inner Global Rules

$$\text{IE}\ \frac{\mathcal{H} \mid \varGamma, \varphi^+, \psi^+, \varDelta \vdash \theta^+}{\mathcal{H} \mid \varGamma, \psi^+, \varphi^+, \varDelta \vdash \theta^+} \qquad \text{IW}\ \frac{\mathcal{H} \mid \varGamma \vdash \varphi^+}{\mathcal{H} \mid \psi^+, \varGamma \vdash \varphi^+} \qquad \text{IC}\ \frac{\mathcal{H} \mid \psi^+, \psi^+, \varGamma \vdash \varphi^+}{\mathcal{H} \mid \psi^+, \varGamma \vdash \varphi^+}$$

$$\wedge\mathcal{I}\ \frac{\mathcal{H} \mid \varGamma \vdash \varphi^+ \qquad \mathcal{H} \mid \varGamma \vdash \psi^+}{\mathcal{H} \mid \varGamma \vdash \varphi^+ \wedge \psi^+}$$

$$\wedge\mathcal{E}_0\ \frac{\mathcal{H} \mid \varGamma \vdash \varphi^+ \wedge \psi^+}{\mathcal{H} \mid \varGamma \vdash \varphi^+} \qquad \wedge\mathcal{E}_1\ \frac{\mathcal{H} \mid \varGamma \vdash \varphi^+ \wedge \psi^+}{\mathcal{H} \mid \varGamma \vdash \psi^+}$$

$$\vee\mathcal{I}_0\ \frac{\mathcal{H} \mid \varGamma \vdash \varphi^+}{\mathcal{H} \mid \varGamma \vdash \varphi^+ \vee \psi^+} \qquad \vee\mathcal{I}_1\ \frac{\mathcal{H} \mid \varGamma \vdash \psi^+}{\mathcal{H} \mid \varGamma \vdash \varphi^+ \vee \psi^+}$$

$$\vee\mathcal{E}\ \frac{\mathcal{H} \mid \varGamma \vdash \varphi^+ \vee \psi^+ \qquad \mathcal{H} \mid \varphi^+, \varGamma \vdash \theta^+ \qquad \mathcal{H} \mid \psi^+, \varGamma \vdash \theta^+}{\mathcal{H} \mid \varGamma \vdash \theta^+}$$

Inner Local Rules

$$[i]\text{Ax}\ \frac{}{[i]\varphi, \varGamma \vdash [i]\varphi} \qquad [i]\bot\mathcal{E}\ \frac{\mathcal{H} \mid \varGamma \vdash [i]\bot}{\mathcal{H} \mid \varGamma \vdash [i]\varphi}$$

$$[i] \supset \mathcal{I}\ \frac{\mathcal{H} \mid [i]\varphi, \varGamma \vdash [i]\psi}{\mathcal{H} \mid \varGamma \vdash [i](\varphi \supset \psi)} \qquad [i] \supset \mathcal{E}\ \frac{\mathcal{H} \mid \varGamma \vdash [i](\varphi \supset \psi) \qquad \mathcal{H} \mid \varGamma \vdash [i]\varphi}{\mathcal{H} \mid \varGamma \vdash [i]\psi}$$

$$[i] \wedge \mathcal{I}\ \frac{\mathcal{H} \mid \varGamma \vdash [i]\varphi \qquad \mathcal{H} \mid \varGamma \vdash [i]\psi}{\mathcal{H} \mid \varGamma \vdash [i](\varphi \wedge \psi)}$$

$$[i] \wedge \mathcal{E}_0\ \frac{\mathcal{H} \mid \varGamma \vdash [i](\varphi \wedge \psi)}{\mathcal{H} \mid \varGamma \vdash [i]\varphi} \qquad [i] \wedge \mathcal{E}_1\ \frac{\mathcal{H} \mid \varGamma \vdash [i](\varphi \wedge \psi)}{\mathcal{H} \mid \varGamma \vdash [i]\psi}$$

$$[i] \vee \mathcal{I}_0\ \frac{\mathcal{H} \mid \varGamma \vdash [i]\varphi}{\mathcal{H} \mid \varGamma \vdash [i](\varphi \vee \psi)} \qquad [i] \vee \mathcal{I}_1\ \frac{\mathcal{H} \mid \varGamma \vdash [i]\psi}{\mathcal{H} \mid \varGamma \vdash [i](\varphi \vee \psi)}$$

$$[i] \vee \mathcal{E}\ \frac{\mathcal{H} \mid \varGamma \vdash [i](\varphi \vee \psi) \qquad \mathcal{H} \mid [i]\varphi, \varGamma \vdash [i]\theta \qquad \mathcal{H} \mid [i]\psi, \varGamma \vdash [i]\theta}{\mathcal{H} \mid \varGamma \vdash [i]\theta}$$

Fig. 1. The underlying logic. Metavariables i and j stand for a process. $\mathcal{H}$ stands for a hypersequent. $\mathcal{H}^+$ stands for a nonempty hypersequent. $\varGamma$ and $\varDelta$ stand for possibly empty contexts. The most important com' rule is based on a rule by Avron [3].

All variable occurrences (including those in Γ) except the first clause are binding. The constructs with a g represent the global rules and the constructs without a g represent the local rules.

We extend a *sequent* to $\Gamma \rhd M\colon \varphi^+$, where Γ is a sequence like $x\colon \psi^+, y\colon \theta^+$ and M is a term. In a sequent $\Gamma \rhd M\colon \varphi^+$, we require the variables in Γ to be distinct from each other. A *contexted type* $\Gamma \rhd \varphi^+$ is a sequent without a term but with variables in Γ. A *hypersequent* is a finite sequence of sequents (called *components*) where the same variable has the same type even if it appears in different components. The typing rules for the terms are given in Fig. 2.

2.3 Reduction

A term M is *of* type φ^+ iff there is a derivation of $\Gamma \rhd M\colon \varphi^+$. A *local* term is a term without $\overleftarrow{l}$, $\overrightarrow{l}$ or any g constructs. A *hyperterm* $\mathcal{O}$ is a nonempty sequence of terms. A *store* maps a location to a local term or ϵ. For a store σ, the updated store $\sigma[l \mapsto x]$ maps l to x and l' to $\sigma(l')$ if l' is different from l. A *configuration* is a triple $(\sigma, \tau, \mathcal{O})$ of two stores σ, τ and a hyperterm $\mathcal{O}$.

To complete the definition of λ-GD, we define the *reductions* $\rightsquigarrow_\clubsuit$ of configurations for $\clubsuit \in \{\mathrm{B}, \mathrm{W}, \mathrm{R}, \mathrm{A}, \mathrm{P}\}$. We consider terms up to α-equivalence and implicitly require all instances of $\rightsquigarrow_\clubsuit$ to avoid free variable captures. Below, $\square$ and $\blacksquare$ match g or nothing.

Definition 1 (Basic Reduction). *The basic reduction $\rightsquigarrow_{\mathrm{B}}$ is the smallest congruence containing the followings:*

- $(\sigma, \tau, (\lambda x.M)O) \rightsquigarrow_{\mathrm{B}} (\sigma, \tau, M[O/x])$
- $(\sigma, \tau, \pi_{\mathrm{l}}^{\square}(\langle M, N\rangle^{\square})) \rightsquigarrow_{\mathrm{B}} (\sigma, \tau, M)$
- $(\sigma, \tau, \pi_{\mathrm{r}}^{\square}(\langle M, N\rangle^{\square})) \rightsquigarrow_{\mathrm{B}} (\sigma, \tau, N)$
- $(\sigma, \tau, \mathsf{match}^{\square}\, \mathsf{inl}^{\square}(M)\, \mathsf{of}\, \mathsf{inl}^{\square}(x).N/\mathsf{inr}^{\square}(y).O) \rightsquigarrow_{\mathrm{B}} (\sigma, \tau, N[M/x])$
- $(\sigma, \tau, \mathsf{match}^{\square}\, \mathsf{inr}^{\square}(M)\, \mathsf{of}\, \mathsf{inl}^{\square}(x).N/\mathsf{inr}^{\square}(y).O) \rightsquigarrow_{\mathrm{B}} (\sigma, \tau, O[M/y])$

There are two sorts of reductions that interact with the store. In summary, $\overleftarrow{l^i_\Gamma}(N)$ tries to write to the right store and read from the left store of the configuration. If a term tries to read from an empty location of a store, the term changes into abort. If a term writes to a full location of a store, it does not abort but the store is not updated. The formal definition follows.

Definition 2 (Write Reduction). *The write reduction $\rightsquigarrow_{\mathrm{W}}$ is the smallest congruence containing the followings where M is a local term:*

- $(\sigma, \tau[l \mapsto \epsilon], \overleftarrow{l^i_\Gamma}(M)) \rightsquigarrow_{\mathrm{W}} (\sigma, \tau[l \mapsto M], \overleftarrow{l^i_\Gamma}(M))$,
- $(\sigma[l \mapsto \epsilon], \tau, \overrightarrow{l^j_\Delta}(M)) \rightsquigarrow_{\mathrm{W}} (\sigma[l \mapsto M], \tau, \overrightarrow{l^j_\Delta}(M))$.

Definition 3 (Read Reduction). *The read reduction $\rightsquigarrow_{\mathrm{R}}$ is the smallest congruence containing the followings:*

- $(\sigma[l \mapsto N], \tau[l \mapsto M'], \overleftarrow{l^i_\Gamma}(M)) \rightsquigarrow_{\mathrm{R}} (\sigma[l \mapsto N], \tau[l \mapsto M'], N)$

External Rules

$$\frac{\mathcal{O}_0 \mid \Gamma, \Delta \rhd M \colon [i]\varphi \qquad \mathcal{O}_1 \mid \Gamma, \Delta \rhd N \colon [j]\psi}{[\mathcal{O}_0, \mathcal{O}_1] \mid \Gamma \rhd \overrightarrow{l^{i}_{\Delta}}(M) \colon [i]\psi \mid \Delta \rhd \overleftarrow{l^{j}_{\Gamma}}(N) \colon [j]\varphi} \qquad \frac{\mathcal{O}^+}{\mathcal{O}^+ \mid \Gamma \rhd \mathsf{abort} : \varphi^+}$$

$$\frac{\mathcal{O} \mid \Gamma \rhd M \colon \varphi^+ \mid \Gamma \rhd N \colon \varphi^+}{\mathcal{O} \mid \Gamma \rhd [M, N] \colon \varphi^+} \qquad \frac{\mathcal{O} \mid \Gamma \rhd M \colon \varphi^+ \mid \Delta \rhd N \colon \psi^+ \mid \mathcal{O}'}{\mathcal{O} \mid \Delta \rhd N \colon \psi^+ \mid \Gamma \rhd M \colon \varphi^+ \mid \mathcal{O}'}$$

Inner Global Rules

$$\frac{\mathcal{O} \mid \Gamma \rhd M \colon \varphi^+}{\mathcal{O} \mid x \colon \psi^+, \Gamma \rhd M \colon \varphi^+} \qquad \frac{\mathcal{O} \mid x \colon \varphi^+, y \colon \varphi^+, \Gamma \rhd M \colon \psi^+}{\mathcal{O} \mid x \colon \varphi^+, \Gamma \rhd M[x/y] \colon \psi^+}$$

$$\frac{\mathcal{O} \mid \Gamma, x \colon \varphi^+, y \colon \psi^+, \Delta \rhd M \colon \theta^+}{\mathcal{O} \mid \Gamma, y \colon \psi^+, x \colon \varphi^+, \Delta \rhd M \colon \theta^+} \qquad \frac{\mathcal{O}_0 \mid \Gamma \rhd M \colon \varphi^+ \qquad \mathcal{O}_1 \mid \Gamma \rhd N \colon \psi^+}{[\mathcal{O}_0, \mathcal{O}_1] \mid \Gamma \rhd \langle M, N \rangle^{\mathsf{g}} \colon \varphi^+ \wedge \psi^+}$$

$$\frac{\mathcal{O} \mid \Gamma \rhd M \colon \varphi^+ \wedge \psi^+}{\mathcal{O} \mid \Gamma \rhd \pi^{\mathsf{g}}_{\mathrm{l}}(M) : \varphi^+} \qquad \frac{\mathcal{O} \mid \Gamma \rhd M \colon \varphi^+ \wedge \psi^+}{\mathcal{O} \mid \Gamma \rhd \pi^{\mathsf{g}}_{\mathrm{r}}(M) : \psi^+}$$

$$\frac{\mathcal{O} \mid \Gamma \rhd M \colon \varphi^+}{\mathcal{O} \mid \Gamma \rhd \mathsf{inl}^{\mathsf{g}}(M) : \varphi^+ \vee \psi^+} \qquad \frac{\mathcal{O} \mid \Gamma \rhd M \colon \psi^+}{\mathcal{O} \mid \Gamma \rhd \mathsf{inr}^{\mathsf{g}}(M) : \varphi^+ \vee \psi^+}$$

$$\frac{\mathcal{O}_0 \mid \Gamma \rhd M \colon \varphi^+ \vee \psi^+ \qquad \mathcal{O}_1 \mid x \colon \varphi^+, \Gamma \rhd N_0 \colon \theta^+ \qquad \mathcal{O}_2 \mid y \colon \psi^+, \Gamma \rhd N_1 \colon \theta^+}{[[\mathcal{O}_0, \mathcal{O}_1], \mathcal{O}_2] \mid \Gamma \rhd \mathsf{match}^{\mathsf{g}}\, M \,\mathsf{of}\, \mathsf{inl}^{\mathsf{g}}(x).N_0/\mathsf{inr}^{\mathsf{g}}(y).N_1 \colon \theta^+}$$

Inner Local Rules

$$\frac{}{x \colon [i]\varphi, \Gamma \rhd x \colon [i]\varphi} \qquad \frac{\mathcal{O} \mid \Gamma \rhd M \colon [i]\bot}{\mathcal{O} \mid \Gamma \rhd \mathsf{abort} : [i]\varphi}$$

$$\frac{\mathcal{O} \mid x \colon [i]\varphi, \Gamma \rhd M \colon [i]\psi}{\mathcal{O} \mid \Gamma \rhd \lambda x.M \colon [i](\varphi \supset \psi)} \qquad \frac{\mathcal{O}_0 \mid \Gamma \rhd M \colon [i](\varphi \supset \psi) \qquad \mathcal{O}_1 \mid \Gamma \rhd N \colon [i]\varphi}{[\mathcal{O}_0, \mathcal{O}_1] \mid \Gamma \rhd MN \colon [i]\psi}$$

$$\frac{\mathcal{O}_0 \mid \Gamma \rhd M \colon [i]\varphi \qquad \mathcal{O}_1 \mid \Gamma \rhd N \colon [i]\psi}{[\mathcal{O}_0, \mathcal{O}_1] \mid \Gamma \rhd \langle M, N \rangle \colon [i](\varphi \wedge \psi)}$$

$$\frac{\mathcal{O} \mid \Gamma \rhd M \colon [i](\varphi \wedge \psi)}{\mathcal{O} \mid \Gamma \rhd \pi_{\mathrm{l}}(M) : [i]\varphi} \qquad \frac{\mathcal{O} \mid \Gamma \rhd M \colon [i](\varphi \wedge \psi)}{\mathcal{O} \mid \Gamma \rhd \pi_{\mathrm{r}}(M) : [i]\psi}$$

$$\frac{\mathcal{O} \mid \Gamma \rhd M \colon [i]\varphi}{\mathcal{O} \mid \Gamma \rhd \mathsf{inl}(M) : [i](\varphi \vee \psi)} \qquad \frac{\mathcal{O} \mid \Gamma \rhd M \colon [i]\psi}{\mathcal{O} \mid \Gamma \rhd \mathsf{inr}(M) : [i](\varphi \vee \psi)}$$

$$\frac{\mathcal{O}_0 \mid \Gamma \rhd M \colon [i](\varphi \vee \psi) \qquad \mathcal{O}_1 \mid x \colon [i]\varphi, \Gamma \rhd N_0 \colon [i]\theta \qquad \mathcal{O}_2 \mid y \colon [i]\psi, \Gamma \rhd N_1 \colon [i]\theta}{[[\mathcal{O}_0, \mathcal{O}_1], \mathcal{O}_2] \mid \Gamma \rhd \mathsf{match}\, M \,\mathsf{of}\, \mathsf{inl}(x).N_0/\mathsf{inr}(y).N_1 \colon [i]\theta}$$

Fig. 2. Term assignment. $\mathcal{O}$ stands for a possibly empty hypersequent (with possible subscripts). $\mathcal{O}^+$ stands for a non-empty hypersequent. Within each rule, $\mathcal{O}_0$, $\mathcal{O}_1$ and $\mathcal{O}_2$ have the same length and the same type so that $[\mathcal{O}_0, \mathcal{O}_1]$ can be defined as the elementwise application of $[\mathcal{T}, \mathcal{T}]$.

– $(\sigma[l \mapsto \epsilon], \tau[l \mapsto M'], \overleftarrow{l^i_\Gamma}(M)) \rightsquigarrow_{\mathrm{R}} (\sigma[l \mapsto \epsilon], \tau[l \mapsto M'], \mathsf{abort})$
– $(\sigma[l \mapsto M'], \tau[l \mapsto N], \overrightarrow{l^i_\Delta}(M)) \rightsquigarrow_{\mathrm{R}} (\sigma[l \mapsto M'], \tau[l \mapsto N], N)$
– $(\sigma[l \mapsto M'], \tau[l \mapsto \epsilon], \overrightarrow{l^i_\Delta}(M)) \rightsquigarrow_{\mathrm{R}} (\sigma[l \mapsto M'], \tau[l \mapsto \epsilon], \mathsf{abort})$.

Definition 4 (Abort Propagation Reduction). *The abort propagation reduction $\rightsquigarrow_{\mathrm{A}}$ is the smallest congruence containing the followings:*

– $(\sigma, \tau, [\mathsf{abort}, M]) \rightsquigarrow_{\mathrm{A}} (\sigma, \tau, M)$, *and* $(\sigma, \tau, [M, \mathsf{abort}]) \rightsquigarrow_{\mathrm{A}} (\sigma, \tau, M)$
– $(\sigma, \tau, C[\mathsf{abort}]) \rightsquigarrow_{\mathrm{A}} (\sigma, \tau, \mathsf{abort})$

where $C[\bullet]$ *is* $\bullet N$, $M\bullet$, $\overrightarrow{l^i_\Delta}(\bullet)$, $\overleftarrow{l^i_\Gamma}(\bullet)$, $\mathsf{inl}^{\Box}(\bullet)$, $\mathsf{inr}^{\Box}(\bullet)$, $\langle \bullet, N\rangle^{\Box}$, $\langle M, \bullet\rangle^{\Box}$, $\pi^{\Box}_i \bullet$. $\mathsf{match}^{\Box}\, M\, \mathsf{of}\, \mathsf{inl}^{\Box}(x).N/\mathsf{inr}^{\Box}(y).\bullet$, $\mathsf{match}^{\Box}\, \bullet\, \mathsf{of}\, \mathsf{inl}^{\Box}(x).N/\mathsf{inr}^{\Box}(y).O$, *or* $\mathsf{match}^{\Box}\, M\, \mathsf{of}\, \mathsf{inl}^{\Box}(x).\bullet/\mathsf{inr}^{\Box}(y).O$

In order to obtain subformula property via proof normalization we add yet another kind of reduction rules.

Definition 5 (Permutative Reduction). *The permutative reduction $\rightsquigarrow_{\mathrm{P}}$ is the smallest congruence containing the followings:*

– $(\sigma, \tau, \left(\mathsf{match}^{\Box}\, M\, \mathsf{of}\, \mathsf{inl}^{\Box}(x).N/\mathsf{inr}^{\Box}(y).O\right) P) \rightsquigarrow_{\mathrm{P}}$
  $(\sigma, \tau, \mathsf{match}^{\Box}\, M\, \mathsf{of}\, \mathsf{inl}^{\Box}(x).NP/\mathsf{inr}^{\Box}(y).OP)$
– $(\sigma, \tau, \pi^{\Box}_d \left(\mathsf{match}^{\blacksquare}\, M\, \mathsf{of}\, \mathsf{inl}^{\blacksquare}(x).N/\mathsf{inr}^{\blacksquare}(y).O\right)) \rightsquigarrow_{\mathrm{P}}$
  $(\sigma, \tau, \mathsf{match}^{\blacksquare}\, M\, \mathsf{of}\, \mathsf{inl}^{\blacksquare}(x).\pi^{\Box}_d N/\mathsf{inr}^{\blacksquare}(y).\pi^{\Box}_d O)$
– $(\sigma, \tau, \mathsf{match}^{\Box}\left(\mathsf{match}^{\blacksquare}\, M\, \mathsf{of}\, \mathsf{inl}^{\blacksquare}(x).N/\mathsf{inr}^{\blacksquare}(y).O\right)\mathsf{of}\, \mathsf{inl}^{\Box}(u).P/\mathsf{inr}^{\Box}(v).Q) \rightsquigarrow_{\mathrm{P}}$
  $(\sigma, \tau, \mathsf{match}^{\blacksquare}\, M\, \mathsf{of}\, \mathsf{inl}^{\blacksquare}(x).\left(\mathsf{match}^{\Box}\, N\, \mathsf{of}\, \mathsf{inl}^{\Box}(u).P/\mathsf{inr}^{\Box}(v).Q\right)/$
  $\mathsf{inr}^{\blacksquare}(y).\left(\mathsf{match}^{\Box}\, O\, \mathsf{of}\, \mathsf{inl}^{\Box}(u).P/\mathsf{inr}^{\Box}(v).Q\right))$
– $(\sigma, \tau, [M, N]P) \rightsquigarrow_{\mathrm{P}} (\sigma, \tau, [MP, NP])$
– $(\sigma, \tau, \pi^{\Box}_d[M, N]) \rightsquigarrow_{\mathrm{P}} (\sigma, \tau, [\pi^{\Box}_d M, \pi^{\Box}_d N])$
– $(\sigma, \tau, \mathsf{match}^{\Box}\, [M, N]\, \mathsf{of}\, \mathsf{inl}^{\Box}(x).P/\mathsf{inr}^{\Box}(y).Q) \rightsquigarrow_{\mathrm{P}}$
  $(\sigma, \tau, [\mathsf{match}^{\Box}\, M\, \mathsf{of}\, \mathsf{inl}^{\Box}(x).P/\mathsf{inr}^{\Box}(y).Q, \mathsf{match}^{\Box}\, N\, \mathsf{of}\, \mathsf{inl}^{\Box}(x).P/\mathsf{inr}^{\Box}(y).Q])$

We define $\rightsquigarrow$ to be the union of $\rightsquigarrow_{\mathrm{B}}$, $\rightsquigarrow_{\mathrm{W}}$, $\rightsquigarrow_{\mathrm{R}}$, $\rightsquigarrow_{\mathrm{A}}$ and $\rightsquigarrow_{\mathrm{P}}$. The reflexive transitive closure of $\rightsquigarrow$ is written as $\rightsquigarrow^*$. A *redex* is a subterm that can be rewritten by a reduction. A configuration $\mathcal{C}$ is *normal* when there is no configuration $\mathcal{D}$ with $\mathcal{C} \rightsquigarrow \mathcal{D}$. A term M is *normal* when the configuration (σ, τ, M) is normal (the choice of σ and τ is irrelevant).

2.4 Properties

An important property of λ-GDis *strong normalization*: every typed hyperterm has a maximal number of reductions it can take. Another is *non-abortfullness*: although some reductions yield abort terms, a typed hyperterm never reduces to a hyperterm that only contains abort's. We show this first for its simpler proof.

Theorem 1 (Non-abortfullness). *All normal forms of a typed configuration contain at least one term that is not* abort.

Proof. When a reduction sequence is fixed, for any location l, depending on whether the right or the left store is filled first, either: (i) no $\overrightarrow{l^i_\Delta}(M) \rightsquigarrow \mathsf{abort}$ occurs for any i, or (ii) no $\overleftarrow{l^i_\Gamma}(M) \rightsquigarrow \mathsf{abort}$ occurs for any i.

If the former is the case, we can rewrite a communication rule occurrence

$$\frac{\mathcal{O}_0 \;\|\; \Gamma, \Delta \rhd M \colon [i]\psi \qquad \mathcal{O}_1 \;\|\; \Gamma, \Delta \rhd N \colon [j]\tau}{[\mathcal{O}_0, \mathcal{O}_1] \;\|\; \Gamma \rhd \overrightarrow{l^i_\Delta}(M) \colon [i]\tau \;\|\; \Delta \rhd \overleftarrow{l^i_\Gamma}(N) \colon [j]\psi}$$

into successive external weakening occurrences

$$\frac{\mathcal{O}_0 \;\|\; \Gamma^j, \Delta^j \rhd M \colon [j]\psi}{\mathcal{O}_0 \;\|\; \rhd \mathsf{abort} : [i]\tau \;\|\; \Gamma^j, \Delta^j \rhd M \colon [j]\psi}$$

where Γ^j and Δ^j can be obtained from the originals by changing every modality $[k]$ to $[j]$. In the lattar case, we can do the symmetric.

After these rewritings for all appearing locations, we obtain a derivation not containing $\overrightarrow{l}$ or $\overleftarrow{l}$. Moreover, the end hypersequent of the resulting derivation has a component not containing abort. The reductions of the original hyperterm can be simulated by the resulting hyperterm. And, even after reductions, the resulting hyperterm has a component not containing abort.

Theorem 2 (Strong Normalization). *λ-GD is strongly normalizing.*

Proof. For proving this, we consider the *local fragment* that does not contain $\overrightarrow{l^i_\Delta}(M)$, $\overleftarrow{l^j_\Gamma}(N)$ or any construct with g. We first reduce the strong normalization of the λ-GD to that of the local fragment, and ultimately to that of de Groote's natural deduction with permutation-conversion [13][2].

We assume an infinite sequence of reductions $(\sigma_0, \tau_0, \mathcal{O}_0) \rightsquigarrow (\sigma_1, \tau_1, \mathcal{O}_1) \rightsquigarrow (\sigma_2, \tau_2, \mathcal{O}_2) \rightsquigarrow \cdots$. From this, we are going to construct an infinite sequence of reductions in the local fragment.

For that, we first build an infinite reduction sequence with constant stores. Using the original infinite sequence, we define a pair of stores called the *store prophecy* $(\sigma_\infty, \tau_\infty)$ where $\sigma_\infty(l) = \epsilon$ if $\sigma_k(l) = \epsilon$ for all $k \in \omega$ and $\sigma_\infty(l) = M$ if $\sigma_k(l) = M$ for some $k \in \omega$; and $\tau_\infty(l)$ is symmetrically defined. Since store contents are never overwritten, σ_∞ and τ_∞ are well-defined. Moreover, $\sigma_i(l)$ and $\sigma_\infty(l)$ coincide unless $\sigma_i(l) = \epsilon$.

We build another reduction sequence $(\sigma_\infty, \tau_\infty, \mathcal{O}_0) \rightsquigarrow^* (\sigma_\infty, \tau_\infty, \mathcal{O}'_1) \rightsquigarrow^* (\sigma_\infty, \tau_\infty, \mathcal{O}'_2) \rightsquigarrow^* \cdots$ with the following invariant: $\mathcal{M}'_i$ can be obtained by replacing some abort occurrences in $\mathcal{M}_i$ with some terms. More specifically, we translate each reduction as follows, keeping the invariant inductively on the number of steps (the base case is satisfied by $\mathcal{M}'_0 = \mathcal{M}_0$ immediately):

[2] To the same effect, we might be able to use other strong normalization results for lambda calculi with commutative conversions, like Balat, Di Cosmo and Fiore [5].

– a read reduction $(\sigma_k, \tau_k, \mathcal{C}\left[\overleftarrow{l^i_\Delta}(M)\right]) \rightsquigarrow_{\mathrm{R}} (\sigma_{k+1}, \tau_{k+1}, \mathcal{C}[O])$ is translated into $(\sigma_\infty, \tau_\infty, \mathcal{C}'\left[\overleftarrow{l^i_\Delta}(M)\right]) \rightsquigarrow_{\mathrm{R}} (\sigma_{k+1}, \tau_{k+1}, \mathcal{C}'[O'])$. If $\sigma_i(l)$ is a term, $\sigma_\infty(l)$ and O' are also identical to the term. Otherwise, O' must be abort. Thus, the invariant holds for $k+1$.
– a write reduction disappears;
– an abort propagation $\mathcal{C}[C[\mathsf{abort}]] \rightsquigarrow_{\mathrm{A}} \mathcal{C}[\mathsf{abort}]$ can be translated either to a similar reduction or no reduction if the abort in the redex is replaced by another term in the $\mathcal{M}'_k$. Note that even in that case, the result $\mathcal{M}'_{k+1}$ can be obtained by replacing some abort occurrences in $\mathcal{M}_{k+1}$ with other terms;
– any other reduction $(\sigma_k, \tau_k, \mathcal{C}[M]) \rightsquigarrow_{\mathrm{B/P}} (\sigma_{k+1}, \tau_{k+1}, \mathcal{C}[N])$ is translated into one similar reduction $(\sigma_\infty, \tau_\infty, \mathcal{C}'[M']) \rightsquigarrow_{\mathrm{B/P}} (\sigma_\infty, \tau_\infty, \mathcal{C}'[N'])$.

Here, we have to show that the translated sequence is infinite. For that, we can use the facts that there are only finite number of usedlocations each of which allows only one write, and that an abort propagation always strictly shortens the term under operation.

After that, we can replace every $\overleftarrow{l^i_\Gamma}(M)$ with $\tau_\infty(l)$. Since $\overleftarrow{l^i_\Gamma}(M)$ either reduces to $\tau_\infty(l)$ or abort, replacing it with $\tau_\infty(l)$ will only "shorten" the reduction sequence for at most one read step. We can do the same for $\overrightarrow{l^j_\Gamma}(N)$. Replacing every such occurrences makes an infinite reduction sequence where every occurring term is in the local fragment. Moreover, the result of the translation is also well-typed. A typing derivation of the resulting hyperterm can be obtained by replacing com' rules with EW rules and changing the process number in types of some variables (c.f. the proof of Thm. 1).

We are aiming at reducing the problem to the strong normalization result by de Groote [13]. Since we have eliminated $\overrightarrow{l^i_\Delta}(M)$ or $\overleftarrow{l^i_\Gamma}(N)$ occurrences, the remaining difference is small: some abort propagation reductions and some permutative reductions involving $[\mathcal{T}, \mathcal{T}']$. We just have to make sure that there are no infinite reduction sequences that consist of these two kinds of reductions only. We can deal with the permutative reductions following de Groote [13]'s strategy for introducing $\perp$. For abort propagation, we can apply the previous argument.

3 Typed Waitfreedom

Waitfreedom [14, 19] is a class of protocols that can solve some of the input-output problems [18, 6]. We define the typed version of waitfreedom.

3.1 Typed Input-Output Problem

Saks and Zaharoglou [19] formulated waitfreedom as a class of input-output problems. Given inputs for all processes and outputs of all processes, an input-output problem decides whether the processes have succeeded or not. We change the standard definition and have typed terms as inputs and outputs.

For that, we let $\mathcal{T}^{-}(\varphi)$ denote the set of closed, local terms of type φ, and $\mathcal{V}^{-}(\varphi)$ denote the set of normal terms in $\mathcal{T}^{-}(\varphi)$. For a finite set of processes $\mathbb{P}$, a *typed input-output problem* consists of each process's input type $(\iota_i)_{i\in\mathbb{P}}$, each process's output type $(o_i)_{i\in\mathbb{P}}$, and a task $R \subseteq \prod_{i\in\mathbb{P}} (\mathcal{T}^{-}(\iota_i)) \times \prod_{i\in\mathbb{P}} (\mathcal{V}^{-}(o_i))$.

3.2 Typed Protocol

We assume a finite set $\mathbb{P}$ of processes and a countably infinite set of program variables $\mathbf{ProV} = \{\mathtt{x}, \mathtt{y}, \mathtt{z}, \ldots\}$. We assume an injection from variables to program variables $x \mapsto \mathtt{x}_x$, whose image leaves infinitely many unused program variables.

A *program* is defined by BNF:

$$p ::= \epsilon \mid \mathtt{x} \leftarrow E; p \mid l \leftarrow E; p$$

where an *expression* is

$$\begin{aligned} E ::= {} & x \mid \mathtt{x} \mid l \mid (EE) \mid \lambda x.E \mid \langle E, E\rangle \mid \mathsf{inl}\,(E) \mid \mathsf{inr}\,(E) \mid \\ & \pi_{\mathrm{l}}\,(E) \mid \pi_{\mathrm{r}}\,(E) \mid \mathsf{match}\, E\, \mathsf{of\ inl}\,(x)\,.E/\mathsf{inr}\,(y)\,.E \mid \epsilon \ . \end{aligned}$$

A program is *well-formed* when a program variable (resp. location) first appears in a $\mathtt{x} \leftarrow E$ (resp. $l \leftarrow E$) sentence, and after that, only appears in expressions. For a contexted type $t = (\Gamma \rhd \varphi^{+})$, we write $M \colon t$ for $\Gamma \rhd M \colon \varphi^{+}$. For input types $(\iota_i)_{i\in\mathbb{P}}$ and output types $(o_i)_{i\in\mathbb{P}}$, a *typed protocol* has:

- two program variables $\mathtt{i}_i$ and $\mathtt{o}_i$ for each process i;
- a finite set of locations L;
- two functions $g \colon L \to \mathbb{P}$ and $d \colon L \to \mathbb{P}$ (specifying the left side writer and the right side writer of each location); $g(l)$ might read what $d(l)$ writes and vice versa;
- $W_{\mathrm{g}}, W_{\mathrm{d}}$: each maps a location in L to a contexted type; $g(l)$ writes a term of $W_{\mathrm{g}}(l)$ and $d(l)$ writes a term of $W_{\mathrm{d}}(l)$;
- a function t_i for each $i \in \mathbb{P}$; that maps a program variable to a contexted type $(x_k \colon \varphi_k)_k \rhd [i]\varphi$ with a special condition $t_i(\mathtt{i}_i) = \iota_i$;
- a typed program p_i for each $i \in \mathbb{P}$, where a *typed program* is a well-formed program where all sentences are typed according to $(t, g, d, i, W_{\mathrm{g}}, W_{\mathrm{d}})$. A sentence $\mathtt{x} \leftarrow E$ is typed iff $E \colon t(\mathtt{x})$ is derivable with assumptions of the form $\rhd \mathtt{y} \colon t(\mathtt{y})$, $\quad \rhd l \colon W_{\mathrm{d}}(l)$ and $\rhd l \colon W_{\mathrm{g}}(l)$.

3.3 Typed Waitfree Computation

We define when a protocol solves a typed input-output problem. These definitions are transferred from [19].

Let $\mathbb{P}$ be $\{0, \ldots, n-1\}$ and fix a typed protocol. A *program variable content* m for $i \in \mathbb{P}$ is a partial function that maps a program variable to a term of $t_i(\mathtt{x})$. A term M *is of* a contexted type $\Gamma \rhd \varphi$ when $\Gamma \rhd M \colon \varphi$ is derivable. A *process snapshot* of $i \in \mathbb{P}$ is a tuple $\langle p, m\rangle$ where p is either a program or abort and m is a program variable content for i. We let S_i denote the set of process snapshots for i. A *system snapshot* is a pair $\langle \vec{s}, \vec{v} \rangle$, where $\vec{s} = \langle s_0, s_1, \ldots, s_{n-1} \rangle \in \prod_{i \in \mathbb{P}} (S_i)$ and $\vec{v} = (\langle v_{l,\mathrm{g}}, v_{l,\mathrm{d}} \rangle)_{l \in L} \in \prod_{l \in L} ((\mathcal{V}(W_\mathrm{g}(l)) \cup \{\epsilon\}) \times (\mathcal{V}(W_\mathrm{d}(l)) \cup \{\epsilon\}))$.

For a nonempty subset J of $\mathbb{P}$, we define an operator $\lhd J$ that takes a system snapshot and produces a system snapshot. This operator depicts a computation step where the processes in J are fired.

We define $(\vec{s}, \vec{v}) \lhd J = (\vec{u}, \vec{m})$ by defining u_i and m_i where $s_i = \langle p, x \rangle$:

$$u_i = \begin{cases} \langle p', x \rangle & (\text{if } p = l \leftarrow E; p' \text{ and } [\![E]\!]_{x,\vec{v}} \neq \epsilon) \\ \langle \epsilon, x \rangle & (\text{if } p = l \leftarrow E; p' \text{ and } [\![E]\!]_{x,\vec{v}} = \epsilon) \\ \langle p', x[\mathtt{x} \mapsto [\![E]\!]_{x,\vec{v}}] \rangle & (\text{if } p = \mathtt{x} \leftarrow E; p', \quad x(\mathtt{x}) = \epsilon \text{ and } [\![E]\!]_{x,\vec{v}} \neq \epsilon) \\ \langle \epsilon, x \rangle & (\text{if } p = \mathtt{x} \leftarrow E; p', \quad x(\mathtt{x}) = \epsilon \text{ and } [\![E]\!]_{x,\vec{v}} = \epsilon) \\ \langle \epsilon, x \rangle & (\text{if } p = \mathtt{x} \leftarrow E; p' \text{ and } x(\mathtt{x}) \neq \epsilon) \\ s_i & (\text{if } p = \epsilon) \end{cases}$$

$$m_{l,\mathrm{g}} = \begin{cases} [\![E]\!]_{x,\vec{v}} & (\text{if } p = l \leftarrow E; p', \quad g(l) = i \text{ and } v_{l,\mathrm{g}} = \epsilon) \\ v_{l,\mathrm{g}} & (\text{otherwise}) \end{cases}$$

$$m_{l,\mathrm{d}} = \begin{cases} [\![E]\!]_{x,\vec{v}} & (\text{if } p = l \leftarrow E; p', \quad d(l) = i \text{ and } v_{l,\mathrm{d}} = \epsilon) \\ v_{l,\mathrm{d}} & (\text{otherwise}) \end{cases}$$

with the following notations. We let $i(l)$ to be $v_{l,\mathrm{g}}$ if $d(l) = i$ and $v_{l,\mathrm{d}}$ if $g(l) = i$. The term $[\![E]\!]_{x,\vec{v}}$ is defined as the unique normal form of $E[x(\vec{\mathtt{y}})/\vec{\mathtt{y}}][\vec{i(l)}/\vec{l}]$, where every program variable $\mathtt{y}$ is replaced by $x(\mathtt{y})$ and the uniqueness is guaranteed by the absence of $\overleftarrow{l}$ or $\overrightarrow{l}$. If any of the substitutes is ϵ, $[\![E]\!]_{x,\vec{v}}$ is ϵ.

A *schedule* is an infinite sequence of nonempty subsets of $\mathbb{P}$, which looks like $\sigma = \sigma_0 \sigma_1 \sigma_2 \cdots$. We say i is *nonfaulty* in σ if it appears infinitely often. When every process is nonfaulty, the schedule is *fair*.

A *run* is a triple $\langle \Pi, \vec{x}, \sigma \rangle$, where Π is a typed protocol, $\vec{x} \in \prod_{i \in \mathbb{P}} \mathcal{T}^-(\iota_i)$ is the input, and σ is a schedule. The *execution* associated to the run is defined as the infinite sequence of system snapshots $C_0 C_1 C_2 \cdots$, where $C_0 = \langle \vec{s^0}, \vec{v^0} \rangle$ is defined by $\vec{s^0_i} = \langle p_i, [\mathtt{i}_i \mapsto x_i] \rangle$ and $v_{l,\mathrm{g}} = v_{l,\mathrm{d}} = \epsilon$, and $C_{k+1} = C_k \lhd \sigma_{i+1}$.

Process i's *output* $\hat{o_k}$ at step k is M if the i-th process snapshot of C_k is (p, x) and the $x[\mathtt{o}_i] = M$, which can be ϵ. The decision value of i on the run $\langle \Pi, \vec{x}, \sigma \rangle$, denoted $d_i \in \mathcal{V}^-(o_i) \cup \{\epsilon\}$ is the first non-ϵ element in the sequence $(\hat{o_k})_{k \in \omega}$, or ϵ if such element does not exist. The n-tuple $\vec{d}$ is defined by d_i's.

A vector $\vec{b} \in \prod_{i \in \mathbb{P}} (\mathcal{V}^-(o_i))$ is *compatible with* $\vec{d} \in \prod_{i \in \mathbb{P}} (\mathcal{V}^-(o_i) \cup \{\epsilon\})$ iff $d_i = b_i$ or $d_i = \epsilon$ holds for any process i. An input $\vec{x} \in \prod_{i \in \mathbb{P}} \mathcal{T}^-(\iota_i)$ is *R-permissible* iff there is at least one vector $\vec{d} \in \prod_{i \in \mathbb{P}} (\mathcal{V}^-(o_i))$ with $(\vec{x}, \vec{b}) \in R$. A typed protocol Π *solves* the typed input-output problem $\langle (\iota_i)_{i \in \mathbb{P}}, (o_i)_{i \in \mathbb{P}}, R \rangle$ on schedule σ iff for all R-permissible inputs $\vec{x}$ and a schedule σ, the decision value of every nonfaulty process i is a term M not ϵ, and there is a vector $\vec{b} \in \prod_{i \in \mathbb{P}} (\mathcal{V}^-(o_i))$ with $\langle \vec{x}, \vec{b} \rangle \in R$ which is compatible with the decision vector $\vec{d}$. A typed protocol is *waitfree* iff it solves the problem on every schedule σ. In that case, the typed input-output problem is *waitfreely solvable*.

4 Characterization of Waitfreedom and λ-GD

We compare the ability of the waitfree protocols and λ-GD .

Definition 6. *A typed input-output problem $\langle(\iota_i)_{i\in\mathbb{P}}, (o_i)_{i\in\mathbb{P}}, R\rangle$ is solvable by a term M of contexted type $(x_i\colon [i]\iota_i)_{i\in\mathbb{P}} \rhd \left(\bigwedge\!\!\!\bigwedge_{i\in\mathbb{P}}[i]o_i\right)$ iff for any closed $(N_i)_{i\in\mathbb{P}}$ of ι_i, all normal forms of $M[\vec{N_i}/\vec{x_i}]$ are in the form $\langle V_0, \langle V_1, \cdots \langle V_{n-2}, \langle V_{n-1}, \bullet\rangle\rangle\cdots\rangle\rangle$ where $\langle(N_i)_{i\in\mathbb{P}}, (V_i)_{i\in\mathbb{P}}\rangle \in R$.*

Theorem 3 (Soundness). *If a typed input-output problem is solvable by a term, there exists a typed protocol that solves the problem.*

We are going to translate a typed hyperterm into a protocol inductively on the type derivation. To make induction work, we use the following auxiliary notions. An *investigator* $\langle i, \mathtt{x}\rangle$ is a pair of a process and a program variable. For a local formula φ, a system snapshot $\langle\vec{s}, \vec{v}\rangle$ satisfies $\langle i, \mathtt{x}\rangle(\varphi)$ iff $s_i(\mathtt{x})$ is an expression of φ. For a set of investigators I, a system snapshot satisfies $I([i]\varphi)$ iff it satisfies $\langle i, \mathtt{x}\rangle(\varphi)$ for some $\mathtt{x}$ with $\langle i, \mathtt{x}\rangle \in I$. This can be extended to all global formulae, as $I(\varphi^+ \wedge \psi^+)$ iff $I(\varphi^+)$ and $I(\psi^+)$; $I(\varphi^+ \vee \psi^+)$ iff $I(\varphi^+)$ or $I(\psi^+)$. A *system snapshot satisfies a global formula* φ^+ iff there exists a finite set of investigators I such that the system snapshot satisfies $I(\varphi^+)$. A *system snapshot satisfies a context* iff it satisfies every global formula in the context. A *protocol p realizes a hypersequent* $\left(\Gamma_0 \vdash \varphi_0^+ \mid \cdots \mid \Gamma_k \vdash \varphi_k^+\right)$ iff for any initial system snapshot satisfying every $\Gamma_{k'}$, there exists a family of investigator sets $(I_{k'})_{k'\in\{0,\ldots,k\}}$ and, when p is executed with any fair schedule, the resulting system snapshots eventually always satisfies at least one of $\varphi_{k'}^+$.

For a typed hyperterm $\mathcal{O}$, we will give $\llbracket\mathcal{O}\rrbracket$, which is a tuple of programs indexed by $\mathbb{P}$. Also, we define $\llparenthesis\mathcal{O}\rrparenthesis$ at the same time as $\llbracket\mathcal{O}\rrbracket$, where $\llparenthesis\mathcal{O}\rrparenthesis$ is a sequence of finite sets of investigators whose length is the same as that of $\mathcal{O}$. We refer to the last element of $\llparenthesis\mathcal{O} \mid M\rrparenthesis$ as $\llparenthesis\mathcal{O} \mid \hat{M}\rrparenthesis$, the second to last element of $\llparenthesis\mathcal{O} \mid M \mid N\rrparenthesis$ as $\llparenthesis\mathcal{O} \mid \hat{M} \mid N\rrparenthesis$ and so on. We denote the right projection of $\llparenthesis\mathcal{O} \mid \hat{M}\rrparenthesis$ as $\llparenthesis\mathcal{O} \mid \hat{M}\rrparenthesis'$. If two sequents of investigator sets $\llparenthesis\mathcal{O}\rrparenthesis$ and $\llparenthesis\mathcal{O}'\rrparenthesis$ have the same length, we define $\llparenthesis\mathcal{O}\rrparenthesis \cup \llparenthesis\mathcal{O}'\rrparenthesis$ to be the elementwise union.

We let ϵ denote $(p_i)_{i\in\mathbb{P}}$ where $p_i = \epsilon$ for all $i \in \mathbb{P}$. Also, $(p_i)_{i\in\mathbb{P}}; (q_i)_{i\in\mathbb{P}}$ denotes $(p_i; q_i)_{i\in\mathbb{P}}$ where the same program variable does not have multiple substitutions (we rename variables in the original typing derivation to satisfy this). And $(p)_j$ denotes $(q_i)_{i\in\mathbb{P}}$ where $q_j = p$ and $q_i = \epsilon$ for all $i \neq j$. Below, we always choose fresh program variables. The definition is inductive over the type derivation.

$$
\begin{aligned}
\llbracket[\mathcal{O}_0, \mathcal{O}_1] \mid \overrightarrow{l^j_\Delta}(M) \mid \overleftarrow{l^i_\Gamma}(N)\rrbracket =& \llbracket\mathcal{O}_0 \mid M\rrbracket; \llbracket\mathcal{O}_1 \mid N\rrbracket; \\
& (l \leftarrow \llparenthesis\mathcal{O}_0 \mid \hat{M}\rrparenthesis'; \mathtt{y} \leftarrow l;)_j; \\
& (l \leftarrow \llparenthesis\mathcal{O}_1 \mid \hat{N}\rrparenthesis'; \mathtt{x} \leftarrow l;)_i \ , \\
\llparenthesis[\mathcal{O}_0, \mathcal{O}_1] \mid \overrightarrow{l^j_\Delta}(M) \mid \overleftarrow{l^i_\Gamma}(N)\rrparenthesis =& \llparenthesis\hat{\mathcal{O}}_0 \mid M\rrparenthesis \cup \llparenthesis\hat{\mathcal{O}}_1 \mid N\rrparenthesis \mid \\
& \langle\{j, \mathtt{y}\}\rangle \mid \{\langle i, \mathtt{x}\rangle\} \ ,
\end{aligned}
$$

$$
\llbracket\mathcal{O} \mid \mathsf{abort}\rrbracket = \llbracket\mathcal{O}\rrbracket \ , \quad \llparenthesis\mathcal{O} \mid \mathsf{abort}\rrparenthesis = \llparenthesis\mathcal{O}\rrparenthesis \mid \emptyset \ , \quad \llbracket x\rrbracket = \epsilon \ ,
$$

$$
\begin{aligned}
\llparenthesis x \rrparenthesis &= \{\langle i, \mathtt{x}_x\rangle\} \text{ where } [i]\varphi \text{ is the type of } x \ , \\
\llbracket \mathcal{O} \mathbin{\vert} \langle M, N\rangle^{\mathsf{g}} \rrbracket &= \llbracket \mathcal{O} \mathbin{\vert} M \rrbracket; \llbracket \mathcal{O} \mathbin{\vert} N \rrbracket \ , \\
\llparenthesis \mathcal{O} \mathbin{\vert} \langle M, N\rangle^{\mathsf{g}} \rrparenthesis &= \llparenthesis \mathcal{O} \mathbin{\vert} M \rrparenthesis \cup \llparenthesis \mathcal{O} \mathbin{\vert} N \rrparenthesis \ , \\
\llbracket \mathcal{O} \mathbin{\vert} \pi^{\mathsf{g}}_a(M) \rrbracket &= \llbracket \mathcal{O} \mathbin{\vert} M \rrbracket \ , \\
\llparenthesis \mathcal{O} \mathbin{\vert} \pi^{\mathsf{g}}_a(M) \rrparenthesis &= \llparenthesis \mathcal{O} \mathbin{\vert} M \rrparenthesis \ , \\
\llbracket [[\mathcal{O}_0, \mathcal{O}_1], \mathcal{O}_2] \mathbin{\vert} \mathsf{match}^{\mathsf{g}}\, M \,\mathsf{of}\, \mathsf{inl}^{\mathsf{g}}(x).N_0/\mathsf{inr}^{\mathsf{g}}(y).N_1 \rrbracket &= \llbracket \mathcal{O}_0 \mathbin{\vert} M \rrbracket; \\
\mathtt{x}_x \leftarrow \mathsf{match}\, \llparenthesis \mathcal{O}_0 \mathbin{\vert} \hat{M} \rrparenthesis' \,\mathsf{of}\, \mathsf{inl}(x).x/\mathsf{inr}(y).\epsilon; &\llbracket \mathcal{O}_1 \mathbin{\vert} N_0 \rrbracket; \\
\mathtt{x}_y \leftarrow \mathsf{match}\, \llparenthesis \mathcal{O}_0 \mathbin{\vert} \hat{M} \rrparenthesis' \,\mathsf{of}\, \mathsf{inl}(x).\epsilon/\mathsf{inr}(y).y; &\llbracket \mathcal{O}_2 \mathbin{\vert} N_1 \rrbracket \ , \\
\llparenthesis [[\mathcal{O}_0, \mathcal{O}_1], \mathcal{O}_2] \mathbin{\vert} \mathsf{match}^{\mathsf{g}}\, M \,\mathsf{of}\, \mathsf{inl}^{\mathsf{g}}(x).N_0/\mathsf{inr}^{\mathsf{g}}(y).N_1 \rrparenthesis &= \llparenthesis \mathcal{O}_1 \mathbin{\vert} N_0 \rrparenthesis \cup \llparenthesis \mathcal{O}_2 \mathbin{\vert} N_1 \rrparenthesis \ , \\
\llbracket \mathcal{O} \mathbin{\vert} \lambda x.M \rrbracket &= \llbracket \mathcal{O} \mathbin{\vert} M \rrbracket; \mathtt{z} \leftarrow \lambda x. \llparenthesis \mathcal{O} \mathbin{\vert} \hat{M} \rrparenthesis' \ , \\
\llbracket [\mathcal{O}_0, \mathcal{O}_1] \mathbin{\vert} MN \rrbracket &= \llbracket \mathcal{O}_0 \mathbin{\vert} M \rrbracket; \llbracket \mathcal{O}_1 \mathbin{\vert} N \rrbracket; \\
&\quad \mathtt{z} \leftarrow \llparenthesis \mathcal{O}_0 \mathbin{\vert} \hat{M} \rrparenthesis' \llparenthesis \mathcal{O}_1 \mathbin{\vert} \hat{N} \rrparenthesis' \ , \\
\llparenthesis [\mathcal{O}_0, \mathcal{O}_1] \mathbin{\vert} MN \rrparenthesis &= (\llparenthesis \hat{\mathcal{O}}_0 \mathbin{\vert} M \rrparenthesis \cup \llparenthesis \hat{\mathcal{O}}_1 \mathbin{\vert} N \rrparenthesis) \mathbin{\vert} \{\langle i, \mathtt{z}\rangle\} \ , \\
\llbracket \mathcal{O} \mathbin{\vert} \pi_{\mathrm{l}}(M) \rrbracket &= \llbracket \mathcal{O} \mathbin{\vert} M \rrbracket; \mathtt{z} \leftarrow \pi_{\mathrm{l}}\left(\llparenthesis \mathcal{O} \mathbin{\vert} \hat{M} \rrparenthesis' \right) \ , \\
\llparenthesis \mathcal{O} \mathbin{\vert} \pi_{\mathrm{l}}(M) \rrparenthesis &= \llparenthesis \hat{\mathcal{O}} \mathbin{\vert} M \rrparenthesis \mathbin{\vert} \{\langle i, \mathtt{z}\rangle\} \ .
\end{aligned}
$$

We omitted translations of some constructs π_{r}, inl, inr, but they are defined in the same way as π_{l} case. When $(x_i \colon [i]\iota_i)_{i \in \mathbb{P}} \rhd M \colon (\bigwedge_{i \in \mathbb{P}}[i]o_i)$ is derivable, we can define a protocol using the above translation. We set $\mathtt{i}_i$ to be $\mathtt{x}_{x_i}$, $\mathtt{o}_i$ to be arbitrarily chosen fresh program variables, L to be the set of locations occurring in the derivation, we set the family of programs to be $\llbracket M \rrbracket; (\mathtt{o}_i \leftarrow \pi_i(\llparenthesis \hat{M} \rrparenthesis'))_{i \in \mathbb{P}}$, where π_i is obtained by composing i times π_{r} to π_{l}. We set g, d, t_i accordingly so that the program is typed. We can simulate a reduction sequence of the hyperterm using a fair execution of the protocol.

And, since the protocol solves a problem for any fair schedule, it solves the problem waitfreely. If we deny the claim, there must be an execution where a nonfaulty process either (a) gives a wrong answer or (b) never gives an answer. Either case, there is a step k when such a failure is inevitable. We can modify the schedule after step k to a fair one, keeeping the failing behavior of the process.

Theorem 4 (Completeness). *If there exists a typed protocol that solves a typed input-output problem, the problem is solvable by a term.*

Saks and Zaharoglou [19] showed that a finite repetition of the participating set problem universally solves any waitfreely solvable problem. Also, n-party participating problem can be solved by a tournament of the two-party participating set problem. It suffices to show a λ-GD term solving the two-party problem.

In the *participating set problem* [7], each process i receives an id c_i and returns a set of id's S_i. The outputs must satisfy (i) $i \in S_i$; (ii) either $S_i \subseteq S_j$ or $S_j \subseteq S_i$; and (iii) $S_i \subseteq S_j$ if $i \in S_j$ for any $i, j \in \mathbb{P}$. For two processes, $\langle S_0, S_1\rangle$ can be $\langle \{c_0\}, \{c_0, c_1\}\rangle$, $\langle \{c_0, c_1\}, \{c_1\}\rangle$ or $\langle \{c_0, c_1\}, \{c_0, c_1\}\rangle$.

We are going to encode the participating set problem in λ-GD. For this, we introduce a base type called Id for process id's. Let there be an injection that maps a natural number i to a constant $C_i \colon \mathsf{Id}$. The additional typing rules involving Id are as follows, where $2 = (\bot \supset \bot) \vee (\bot \supset \bot)$:

$$\frac{}{\rhd\, c_n \colon [i]\mathsf{Id}} \qquad\qquad \frac{\Gamma \rhd M_0 \colon [i]\mathsf{Id} \qquad \Gamma \rhd M_1 \colon [i]\mathsf{Id}}{\Gamma \rhd M_0 == M_1 \colon [i]2} \quad .$$

The additional reduction is

$$c_m == c_n \rightsquigarrow \begin{cases} \mathsf{inl}\,(\lambda x.x) & (\text{if } m = n) \\ \mathsf{inr}\,(\lambda x.x) & (\text{otherwise}) \end{cases} .$$

Also, $\mathsf{If}\ M\ \mathsf{then}\ N_0\ \mathsf{else}\ N_1$ is an abbreviation for $\mathsf{match}\ M\ \mathsf{of}\ \mathsf{inl}\,(x)\,.N_0/\mathsf{inr}\,(y)\,.N_1$.

We represent a finite set of id's as a typed lambda term, whose type is $[i](\mathsf{Id} \supset 2)$. Intuitively, a set takes an id and decides whether it is *in* or *out*. The emptyset is represented by a term $\lambda x.\mathsf{inr}\,(\bullet)$. When a finite set S is represented by a term M, the set $S \cup \{c\}$ is represented by a term $\lambda x.\,(\mathsf{If}\ x == c\ \mathsf{then}\ \mathsf{inl}\,(\bullet)\ \mathsf{else}\ Mx)$. With the above construction, we define abbreviations like $\{c_0, c_1, c_2\}$.

Now, we are ready to construct a hyperterm solving the two-party participating set problem. We can obtain a derivation of:

$$x \colon [0]\mathsf{Id}, y \colon [1]\mathsf{Id} \rhd [\langle\{\overrightarrow{l^0_\epsilon}\,(x)\,, x\}, \{y\}\rangle^{\mathrm{g}}, \langle\{x\}, \{\overleftarrow{l^1_\epsilon}\,(y)\,, y\}\rangle^{\mathrm{g}}] \colon [0](\mathsf{Id} \supset 2) \wedge ([1](\mathsf{Id} \supset 2)) \ .$$

One possible reduction sequence is as follows:

$$\begin{aligned} &([], [], [\langle\{\overrightarrow{l^0_\epsilon}\,(c_0)\,, c_0\}, \{c_1\}\rangle^{\mathrm{g}}, \langle\{c_0\}, \{\overleftarrow{l^1_\epsilon}\,(c_1)\,, c_1\}\rangle^{\mathrm{g}}]) \\ \rightsquigarrow^*&([l \mapsto c_0], [], [\langle\{\mathsf{abort}\,, c_0\}, \{c_1\}\rangle^{\mathrm{g}}, \langle\{c_0\}, \{\overleftarrow{l^1_\epsilon}\,(c_1)\,, c_1\}\rangle^{\mathrm{g}}]) \\ \rightsquigarrow^*&([l \mapsto c_0], [l \mapsto c_1], \langle\{c_0\}, \{c_0, c_1\}\rangle^{\mathrm{g}}) \ . \end{aligned}$$

Moreover, the same initial configuration can reduce to

$$([l \mapsto c_0], [l \mapsto c_1], \langle\{c_1, c_0\}, \{c_1\}\rangle^{\mathrm{g}}) \text{ or } ([l \mapsto c_0], [l \mapsto c_1], \langle\{c_1, c_0\}, \{c_0, c_1\}\rangle^{\mathrm{g}}) \ .$$

There are no other normal forms. These three normal forms correspond to the three answers for the two-party participating set problem.

5 Related Work

Avron [2] formulates a hypersequent calculus for Gödel–Dummett logic and proves cut-elimination theorem using a method similar to Gentzen [12]. Also, he explains the intuition behind the communication rule as "the inputs through the ports in Γ_2' are transmitted to the component with output of type A_1. The inputs through Γ_1' are treated similarly." He did not mention the possibility of any transmission failures, which we exploited in order to characterize waitfreedom. Ciabattoni, Galatos and Terui [8] gives a class of logics that have hypersequent calculi with cut-elimination. Their cut-elimination proof is general but it does not obviously reveal the computational content.

Baaz, Ciabattoni and Fermüller [4] propose a hypersequent-style natural deduction for Gödel–Dummett logic, but did not define reduction. Fermüller [10] gives a game semantics for Gödel–Dummett logic, which is based on Lorenzen game [20] and essentially proof searching bottom-to-up.

Among numerous typed programming languages with parallelism, to our knowledge, none exhibits the connection of Gödel–Dummett logic and waitfreedom. Abramsky [1]'s calculus PE_2 for classical linear logic is deterministic [1, Theorem 7.9] so that it is impossible to model waitfreedom using PE_2. The π-calculus [17], Join calculus [11], and even asynchronous π-calculus [16] have too strong synchronization abilities to model waitfreedom because a process can wait for an input.

Hirai [15] compares the temporal order of waitfree computation and the Kripke models of a modal logic similar to Gödel–Dummett logic. The current work witnesses the constructive content of his model theoretic comparison.

6 Future Work

As a programming language, λ-GD allows efficient execution because it requires no synchronization among processes. We implemented a calculus similar to λ-GD in a programming language Haskell[3]. A possible extension is adding synchronization primitives. It would be interesting to compare different synchronization primitives and different intermediate logics, generalizing waitfreedom and Gödel–Dummett logic.

We are also planning to develop a waitfree protocol verification mechanism in Coq because it is valuable to remove unnecessary synchronization while keeping the program correct in high performance computing.

An anonymous refree pointed out that the introduction of modalities is interesting on its own. We have not investigated the semantics of these modalities.

In λ-GD, the source of nondeterminism can be explicitly expressed as the store prophecy. If we can find a semantic counterpart Sch of the store prophecy, possibly, we can obtain a denotation $\mathcal{D}^{\mathsf{Sch}}$ of terms using a denotation $\mathcal{D}$ for normal forms. If that succeeds for classical logic, it will be interesting[4].

7 Conclusion

We proposed λ-GD, a lambda calculus based on hypersequent calculus of Gödel–Dummett logic. We proved normalization and non-abortfullness. The calculus characterizes the typed version of waitfree computation. Our result hints broader correspondence between proof theory and distributed computation.

[3] Given Haskell Platform, a command `cabal waitfree` installs the implementation.

[4] Kazushige Terui suggested the potential impact for classical logic.

Acknowledgement. This work is encouraged by feedbacks from ACAN (Algebraic and Coalgebraic Approaches to Non-Classical Logics Workshop) and OPLSS'11 participants, supported by Grant-in-Aid for JSPS Fellows 23-6978, and partly prepared during the author's stay in ILPS, the University of Amsterdam.

References

1. Abramsky, S.: Computational interpretations of linear logic. Theo. Comp. Sci. 111(1-2), 3–57 (1993)
2. Avron, A.: Hypersequents, logical consequence and intermediate logics for concurrency. Ann. Math. Artif. Intell. 4, 225–248 (1991)
3. Avron, A.: The method of hypersequents in the proof theory of propositional nonclassical logics, pp. 1–32. Clarendon Press (1996)
4. Baaz, M., Ciabattoni, A., Fermüller, C.: A natural deduction system for intuitionistic fuzzy logic. Lectures on soft computing (2001)
5. Balat, V., Di Cosmo, R., Fiore, M.: Extensional normalisation and type-directed partial evaluation for typed lambda calculus with sums. SIGPLAN Not. 39(1), 64–76 (1 2004)
6. Biran, O., Moran, S., Zaks, S.: A combinatorial characterization of the distributed tasks which are solvable in the presence of one faulty processor. In: PODC '88. pp. 263–275. ACM (1988)
7. Borowsky, E., Gafni, E.: Immediate atomic snapshots and fast renaming. In: PODC '93. pp. 41–51. ACM (1993)
8. Ciabattoni, A., Galatos, N., Terui, K.: From axioms to analytic rules in nonclassical logics. In: LICS'08. pp. 229–240. IEEE (2008)
9. Dummett, M.: A propositional calculus with denumerable matrix. J. Symb. Logic 24(2), 97–106 (1959)
10. Fermüller, C.: Parallel dialogue games and hypersequents for intermediate logics. In: TABLEAUX '03, LNCS, vol. 2796, pp. 48–64. Springer (2003)
11. Fournet, C., Gonthier, G.: The Join calculus: A language for distributed mobile programming. In: Applied Semantics, LNCS, vol. 2395, pp. 268–332. Springer (2002)
12. Gentzen, G.: Investigations into logical deduction. American Philosophical Quarterly 1(4), pp. 288–306 (1964)
13. de Groote, P.: On the Strong Normalisation of Intuitionistic Natural Deduction with Permutation-Conversions. Information and Computation 178(2), 441–464 (2002)
14. Herlihy, M.: Impossibility and universality results for wait-free synchronization. In: PODC '88. pp. 276–290. ACM (1988)
15. Hirai, Y.: An intuitionistic epistemic logic for sequential consistency on shared memory. In: LPAR-16, LNCS, vol. 6355, pp. 272–289. Springer (2010)
16. Honda, K., Tokoro, M.: An object calculus for asynchronous communication. In: ECOOP'91, LNCS, vol. 512, pp. 133–147. Springer (1991)
17. Milner, R.: Communicating and mobile systems: the pi-calculus. Cambridge University Press (1999)
18. Moran, S.: Extended impossibility results for asynchronous complete networks. Information Processing Letters 26(3), 145–151 (1987)
19. Saks, M., Zaharoglou, F.: Wait-free k-set agreement is impossible: the topology of public knowledge. In: STOC '93. pp. 101–110. ACM (1993)
20. Sørensen, M.H., Urzyczyn, P.: Lectures on the Curry–Howard isomorphism. Elsevier (2007)