Live Functional Programming with Typed Holes: Supplemental Appendix

ANONYMOUS AUTHOR(S)

This document provides supplemental material referenced in the paper.

A Additional Definitions for Hazelnut Live

A.1 Substitution

$\boxed{[d/x]d' = d''}$ d'' is obtained by substituting d for x in d'

$\boxed{[d/x]\sigma = \sigma'}$ σ' is obtained by substituting d for x in σ

$$\begin{array}{lcll}
[d/x]c & = & c & \\
[d/x]x & = & d & \\
[d/x]y & = & y & \text{when } x \neq y \\
[d/x]\lambda y{:}\tau.d' & = & \lambda y{:}\tau.[d/x]d' & \text{when } x \neq y \text{ and } y \notin \mathsf{FV}(d) \\
[d/x]d_1(d_2) & = & ([d/x]d_1)([d/x]d_2) & \\
[d/x]\llparenthesis\rrparenthesis^u_\sigma & = & \llparenthesis\rrparenthesis^u_{[d/x]\sigma} & \\
[d/x]\llparenthesis d' \rrparenthesis^u_\sigma & = & \llparenthesis [d/x]d' \rrparenthesis^u_{[d/x]\sigma} & \\
[d/x]d'\langle \tau_1 \Rightarrow \tau_2 \rangle & = & ([d/x]d')\langle \tau_1 \Rightarrow \tau_2 \rangle & \\
[d/x]d'\langle \tau_1 \Rightarrow \llparenthesis\rrparenthesis \not\Rightarrow \tau_2 \rangle & = & ([d/x]d')\langle \tau_1 \Rightarrow \llparenthesis\rrparenthesis \not\Rightarrow \tau_2 \rangle & \\
[d/x]\cdot & = & \cdot & \\
[d/x]\sigma, d/y & = & [d/x]\sigma, [d/x]d/y &
\end{array}$$

Lemma A.1 (Substitution).
(1) If $\Delta; \Gamma, x : \tau' \vdash d : \tau$ and $\Delta; \Gamma \vdash d' : \tau'$ then $\Delta; \Gamma \vdash [d'/x]d : \tau$.
(2) If $\Delta; \Gamma, x : \tau' \vdash \sigma : \Gamma'$ and $\Delta; \Gamma \vdash d' : \tau'$ then $\Delta; \Gamma \vdash [d'/x]\sigma : \Gamma'$.

Proof. By rule induction on the first assumption in each case. The conclusion follows from the definition of substitution in each case. □

A.2 Canonical Forms

Lemma A.2 (Canonical Value Forms). *If $\Delta; \emptyset \vdash d : \tau$ and d* val *then $\tau \neq \llparenthesis\rrparenthesis$ and*
(1) If $\tau = b$ then $d = c$.
(2) If $\tau = \tau_1 \rightarrow \tau_2$ then $d = \lambda x{:}\tau_1.d'$ where $\Delta; x : \tau_1 \vdash d' : \tau_2$.

Lemma A.3 (Canonical Boxed Forms). *If $\Delta; \emptyset \vdash d : \tau$ and d* boxedval *then*
(1) If $\tau = b$ then $d = c$.
(2) If $\tau = \tau_1 \rightarrow \tau_2$ then either
  i. $d = \lambda x{:}\tau_1.d'$ where $\Delta; x : \tau_1 \vdash d' : \tau_2$, or
  ii. $d = d'\langle \tau_1' \rightarrow \tau_2' \Rightarrow \tau_1 \rightarrow \tau_2 \rangle$ where $\tau_1' \rightarrow \tau_2' \neq \tau_1 \rightarrow \tau_2$ and $\Delta; \emptyset \vdash d' : \tau_1' \rightarrow \tau_2'$.
(3) If $\tau = \llparenthesis\rrparenthesis$ then $d = d'\langle \tau' \Rightarrow \llparenthesis\rrparenthesis \rangle$ where τ' ground *and $\Delta; \emptyset \vdash d' : \tau'$.*

Lemma A.4 (Canonical Indeterminate Forms). *If $\Delta; \emptyset \vdash d : \tau$ and d* indet *then either*
(1) $d = \llparenthesis\rrparenthesis^u_\sigma$ and $u :: \tau[\Gamma'] \in \Delta$, or
(2) $d = \llparenthesis d' \rrparenthesis^u_\sigma$ and d' final *and $\Delta; \emptyset \vdash d' : \tau'$ and $u :: \tau[\Gamma'] \in \Delta$, or*
(3) $d = d_1(d_2)$ and $\Delta; \emptyset \vdash d_1 : \tau_2 \rightarrow \tau$ and $\Delta; \emptyset \vdash d_2 : \tau_2$ and d_1 indet *and d_2* final *and $d_1 \neq d_1\langle \tau_3 \rightarrow \tau_4 \Rightarrow \tau_3' \rightarrow \tau_4' \rangle$, or*
(4) $\tau = b$ and $d = d'\langle \llparenthesis\rrparenthesis \Rightarrow b \rangle$ and d' indet *and $d' \neq d''\langle \tau' \Rightarrow \llparenthesis\rrparenthesis \rangle$, or*
(5) $\tau = b$ and $d = d'\langle \tau' \Rightarrow \llparenthesis\rrparenthesis \not\Rightarrow b \rangle$ and τ' ground *and $\tau' \neq b$ and $\Delta; \emptyset \vdash d' : \tau'$, or*
(6) $\tau = \tau_{11} \rightarrow \tau_{12}$ and $d = d'\langle \tau_1 \rightarrow \tau_2 \Rightarrow \tau_{11} \rightarrow \tau_{12} \rangle$ and d' indet *and $\tau_1 \rightarrow \tau_2 \neq \tau_{11} \rightarrow \tau_{12}$, or*
(7) $\tau = \llparenthesis\rrparenthesis \rightarrow \llparenthesis\rrparenthesis$ and $d = d'\langle \llparenthesis\rrparenthesis \Rightarrow \llparenthesis\rrparenthesis \Rightarrow \llparenthesis\rrparenthesis \rangle$ and d' indet *and $d' \neq d''\langle \tau' \Rightarrow \llparenthesis\rrparenthesis \rangle$, or*
(8) $\tau = \llparenthesis\rrparenthesis \rightarrow \llparenthesis\rrparenthesis$ and $d = d'\langle \tau' \Rightarrow \llparenthesis\rrparenthesis \not\Rightarrow \llparenthesis\rrparenthesis \rightarrow \llparenthesis\rrparenthesis \rangle$ and $\tau' \neq \tau$ and τ' ground *and d'* indet *and $\Delta; \emptyset \vdash d' : \tau'$, or*
(9) $\tau = \llparenthesis\rrparenthesis$ and $d = d'\langle \tau' \Rightarrow \llparenthesis\rrparenthesis \rangle$ and τ' ground *and d'* indet*.*

The proofs for all three of these theorems follow by straightforward rule induction.

A.3 Complete Programs

$\boxed{\tau \text{ complete}}$ τ is complete

TCBase
$$\frac{}{b \text{ complete}}$$

TCArr
$$\frac{\tau_1 \text{ complete} \qquad \tau_2 \text{ complete}}{\tau_1 \to \tau_2 \text{ complete}}$$

$\boxed{e \text{ complete}}$ e is complete

ECVar
$$\frac{}{x \text{ complete}}$$

ECConst
$$\frac{}{c \text{ complete}}$$

ECLam1
$$\frac{\tau \text{ complete} \qquad e \text{ complete}}{\lambda x{:}\tau.e \text{ complete}}$$

ECLam2
$$\frac{e \text{ complete}}{\lambda x.e \text{ complete}}$$

ECAp
$$\frac{e_1 \text{ complete} \qquad e_2 \text{ complete}}{e_1(e_2) \text{ complete}}$$

ECAsc
$$\frac{e \text{ complete} \qquad \tau \text{ complete}}{e : \tau \text{ complete}}$$

$\boxed{d \text{ complete}}$ d is complete

DCVar
$$\frac{}{x \text{ complete}}$$

DCConst
$$\frac{}{c \text{ complete}}$$

DCLam
$$\frac{\tau \text{ complete} \qquad d \text{ complete}}{\lambda x{:}\tau.d \text{ complete}}$$

DCAp
$$\frac{d_1 \text{ complete} \qquad d_2 \text{ complete}}{d_1(d_2) \text{ complete}}$$

DCCast
$$\frac{d \text{ complete} \qquad \tau_1 \text{ complete} \qquad \tau_2 \text{ complete}}{d\langle \tau_1 \Rightarrow \tau_2 \rangle \text{ complete}}$$

Fig. 1. Complete types, external expressions, and internal expressions

We define Γ complete as follows.

Definition A.5 (Typing Context Completeness). Γ complete *iff for each* $x : \tau \in \Gamma$, *we have* τ complete.

When two types are complete and consistent, they are equal.

Lemma A.6 (Complete Consistency). *If* $\tau_1 \sim \tau_2$ *and* τ_1 complete *and* τ_2 complete *then* $\tau_1 = \tau_2$.

Proof. By straightforward rule induction. □

This implies that in a well-typed and complete internal expression, every cast is an identity cast.

A.4 Multiple Steps

$\boxed{d \mapsto^* d'}$ d multi-steps to d'

MultiStepRefl
$$\frac{}{d \mapsto^* d}$$

MultiStepSteps
$$\frac{d \mapsto d' \qquad d' \mapsto^* d''}{d \mapsto^* d''}$$

Fig. 2. Multi-Step Transitions

A.5 Hole Filling

Lemma A.7 (Filling).
(1) If $\Delta, u :: \tau'[\Gamma']; \Gamma \vdash d : \tau$ *and* $\Delta; \Gamma' \vdash d' : \tau'$ *then* $\Delta; \Gamma \vdash \llbracket d'/u \rrbracket d : \tau$.
(2) If $\Delta, u :: \tau'[\Gamma']; \Gamma \vdash \sigma : \Gamma''$ *and* $\Delta; \Gamma' \vdash d' : \tau'$ *then* $\Delta; \Gamma \vdash \llbracket d'/u \rrbracket \sigma : \Gamma''$.

Proof. In each case, we proceed by rule induction on the first assumption, appealing to the Substitution Lemma as necessary. □

We need the following auxiliary definitions, which lift hole filling to evaluation contexts taking care to consider the special situation where the mark is inside the hole that is being filled, to prove the Commutativity theorem.

$\boxed{\mathsf{inhole}(u; \mathcal{E})}$ The mark in $\mathcal{E}$ is inside non-empty hole closure u

$$\texttt{InHoleNEHole}\ \frac{}{\mathsf{inhole}(u; \llparenthesis \mathcal{E} \rrparenthesis^{u}_{\sigma})} \qquad \texttt{InHoleAp1}\ \frac{\mathsf{inhole}(u; \mathcal{E})}{\mathsf{inhole}(u; \mathcal{E}(d))} \qquad \texttt{InHoleAp2}\ \frac{\mathsf{inhole}(u; \mathcal{E})}{\mathsf{inhole}(u; \mathcal{E}(d))}$$

$$\texttt{InHoleCast}\ \frac{\mathsf{inhole}(u; \mathcal{E})}{\mathsf{inhole}(u; \mathcal{E}\langle \tau_1 \Rightarrow \tau_2 \rangle)} \qquad \texttt{InHoleFailedCast}\ \frac{\mathsf{inhole}(u; \mathcal{E})}{\mathsf{inhole}(u; \mathcal{E}\langle \tau_1 \Rightarrow \llparenthesis\rrparenthesis \not\Rightarrow \tau_2 \rangle)}$$

$\boxed{\llbracket d/u \rrbracket \mathcal{E} = \mathcal{E}'}$ $\mathcal{E}'$ is obtained by filling hole u in $\mathcal{E}$ with d

$$\texttt{EFillMark}\ \frac{}{\llbracket d/u \rrbracket \circ = \circ} \qquad \texttt{EFillAp1}\ \frac{\llbracket d/u \rrbracket \mathcal{E} = \mathcal{E}'}{\llbracket d/u \rrbracket \mathcal{E}(d_2) = \mathcal{E}'(\llbracket d/u \rrbracket d_2)} \qquad \texttt{EFillAp2}\ \frac{\llbracket d/u \rrbracket \mathcal{E} = \mathcal{E}'}{\llbracket d/u \rrbracket d_1(\mathcal{E}) = (\llbracket d/u \rrbracket d_1)(\mathcal{E}')}$$

$$\texttt{EFillNEHole}\ \frac{u \neq v \qquad \llbracket d/u \rrbracket \mathcal{E} = \mathcal{E}'}{\llbracket d/u \rrbracket \llparenthesis \mathcal{E} \rrparenthesis^{v}_{\sigma} = \llparenthesis \mathcal{E}' \rrparenthesis^{v}_{\llbracket d/u \rrbracket \sigma}} \qquad \texttt{EFillCast}\ \frac{\llbracket d/u \rrbracket \mathcal{E} = \mathcal{E}'}{\llbracket d/u \rrbracket \mathcal{E}\langle \tau_1 \Rightarrow \tau_2 \rangle = \mathcal{E}'\langle \tau_1 \Rightarrow \tau_2 \rangle}$$

$$\texttt{EFillFailedCast}\ \frac{\llbracket d/u \rrbracket \mathcal{E} = \mathcal{E}'}{\llbracket d/u \rrbracket \mathcal{E}\langle \tau_1 \Rightarrow \llparenthesis\rrparenthesis \not\Rightarrow \tau_2 \rangle = \mathcal{E}'\langle \tau_1 \Rightarrow \llparenthesis\rrparenthesis \not\Rightarrow \tau_2 \rangle}$$

Fig. 3. Evaluation Context Filling

Lemma A.8 (Substitution Commutativity). *If*
(1) $\Delta, u :: \tau'[\Gamma']; x : \tau_2 \vdash d_1 : \tau$ *and*
(2) $\Delta, u :: \tau'[\Gamma']; \emptyset \vdash d_2 : \tau_2$ *and*
(3) $\Delta; \Gamma' \vdash d' : \tau'$
then $\llbracket d'/u \rrbracket [d_2/x] d_1 = [\llbracket d'/u \rrbracket d_2/x] \llbracket d'/u \rrbracket d_1$.

Proof. We proceed by structural induction on d_1 and rule induction on the typing premises, which serve to ensure that the free variables in d' are accounted for by every closure for u. □

Lemma A.9 (Instruction Commutativity). *If*
(1) $\Delta, u :: \tau'[\Gamma']; \emptyset \vdash d_1 : \tau$ *and*
(2) $\Delta; \Gamma' \vdash d' : \tau'$ *and*

(3) $d_1 \longrightarrow d_2$
then $[\![d'/u]\!]d_1 \longrightarrow [\![d'/u]\!]d_2$.

Proof. We proceed by cases on the instruction transition assumption (no induction is needed). For Rule `ITLam`, we defer to the Substitution Commutativity lemma above. For the remaining cases, the conclusion follows from the definition of hole filling. □

Lemma A.10 (Filling Totality). *Either* $\mathsf{inhole}(u; \mathcal{E})$ *or* $[\![d/u]\!]\mathcal{E} = \mathcal{E}'$ *for some* $\mathcal{E}'$.

Proof. We proceed by structural induction on $\mathcal{E}$. Every case is handled by one of the two judgements. □

Lemma A.11 (Discarding). *If*
(1) $d_1 = \mathcal{E}\{d_1'\}$ *and*
(2) $d_2 = \mathcal{E}\{d_2'\}$ *and*
(3) $\mathsf{inhole}(u; \mathcal{E})$
then $[\![d/u]\!]d_1 = [\![d/u]\!]d_2$.

Proof. We proceed by structural induction on $\mathcal{E}$ and rule induction on all three assumptions. Each case follows from the definition of instruction selection and hole filling. □

Lemma A.12 (Filling Distribution). *If* $d_1 = \mathcal{E}\{d_1'\}$ *and* $[\![d/u]\!]\mathcal{E} = \mathcal{E}'$ *then* $[\![d/u]\!]d_1 = \mathcal{E}'\{[\![d/u]\!]d_1'\}$.

Proof. We proceed by rule induction on both assumptions. Each case follows from the definition of instruction selection and hole filling. □

Theorem A.13 (Commutativity). *If*
(1) $\Delta, u :: \tau'[\Gamma']; \emptyset \vdash d_1 : \tau$ *and*
(2) $\Delta; \Gamma' \vdash d' : \tau'$ *and*
(3) $d_1 \mapsto^* d_2$
then $[\![d'/u]\!]d_1 \mapsto^* [\![d'/u]\!]d_2$.

Proof. By rule induction on assumption (3). The reflexive case is immediate. In the inductive case, we proceed by rule induction on the stepping premise. There is one rule, Rule `Step`. By Filling Totality, either $\mathsf{inhole}(u; \mathcal{E})$ or $[\![d/u]\!]\mathcal{E} = \mathcal{E}'$. In the former case, by Discarding, we can conclude by `MultiStepRefl`. In the latter case, by Instruction Commutativity and Filling Distribution we can take a `Step`, and we can conclude via `MultiStepSteps` by applying Filling, Preservation and then the induction hypothesis. □

We exclude these proofs and definitions from the Agda mechanization for two reasons. First, fill-and-resume is merely an optimization, and unlike the meta theory of Sec. **??**, these properties are generally not conserved by certain reasonable extensions of the core calculus (e.g., reference cells and other non-commuting effects). Second, to properly encode the hole filling operation, such a mechanization requires a significantly more complex representation of hole environments; unfortunately, Agda cannot be easily convinced that the definition of hole filling is well-founded (Nanevski et al. [2008] establish that it is in fact well-founded). By contrast, the developments in Sec. **??** do not require these more complex (and somewhat problematic) representations.

A.6 Confluence, and Friends

There are various ways to encode the intuition that "evaluation order does not matter". One way to do so is by establishing a confluence property (which is closely related to the Church-Rosser property [Church and Rosser 1936]).

The most general confluence property does not hold for the dynamic semantics in Sec. **??** for the usual reason: We do not reduce under binders (Blanc et al. [2005] discuss the standard counterexample). We could recover confluence by specifying reduction under binders, either generally or in a more restricted form where only closed sub-expressions are reduced [Blanc et al. 2005; Çagman and Hindley 1998; Lévy and Maranget 1999]. However, reduction under binders conflicts with the standard implementation approaches for most programming languages [Blanc et al. 2005]. A more satisfying approach considers confluence modulo equality [Huet 1980]. The simplest such approach restricts our interest to top-level expressions of base type that result in values, in which case the following special case of confluence does hold (trivially when the only base type has a single value, but also more generally for other base types).

Lemma A.14 (Base Confluence). *If* $\Delta; \emptyset \vdash d : b$ *and* $d \mapsto^* d_1$ *and* d_1 val *and* $d \mapsto^* d_2$ *then* $d_2 \mapsto^* d_1$.

We can then prove the following property, which establishes that fill-and-resume is sound.

Theorem A.15 (Resumption). *If* $\Delta, u :: \tau'[\Gamma']; \emptyset \vdash d : b$ *and* $\Delta; \Gamma' \vdash d' : \tau'$ *and* $d \mapsto^* d_1$ *and* $[\![d'/u]\!]d \mapsto^* d_2$ *and* d_2 val *then* $[\![d'/u]\!]d_1 \mapsto^* d_2$.

Proof. By Commutativity, $[\![d'/u]\!]d \mapsto^* [\![d'/u]\!]d_1$. By Base Confluence, we can conclude. □

B Extensions to Hazelnut Live

We give two extensions here, numbers and sum types, to maintain parity with Hazelnut as specified by Omar et al. [2017].

It is worth observing that these extensions do not make explicit mention of expression holes. The "non-obvious" machinery is almost entirely related to casts. Fortunately, there has been excellent work of late on generating "gradualized" specifications from standard specifications [Cimini and Siek 2016, 2017]. The extensions below, and other extensions of interest, closely follow the output of the gradualizer: http://cimini.info/gradualizerDynamicSemantics/ (which, like our work, is based on the refined account of gradual typing by [Siek et al. 2015]). The rules for indeterminate forms are mainly where the gradualizer is not sufficient. We leave to future work the related question of automatically generating a hole-aware static and dynamic semantics from a standard language specification.

B.1 Numbers

We extend the syntax as follows:

$$\begin{array}{llll} \mathsf{HTyp} & \tau & ::= \cdots & \mid \mathsf{num} \\ \mathsf{HExp} & e & ::= \cdots & \mid \underline{n} \mid e + e \\ \mathsf{IHExp} & d & ::= \cdots & \mid \underline{n} \mid d + d \end{array}$$

$\boxed{\Gamma \vdash e \Rightarrow \tau \rightsquigarrow d \dashv \Delta}$ e synthesizes type τ and expands to d

$$\frac{}{\Gamma \vdash \underline{n} \Rightarrow \mathsf{num} \rightsquigarrow \underline{n} \dashv \cdot} \qquad \frac{\Gamma \vdash e_1 \Leftarrow \mathsf{num} \rightsquigarrow d_1 : \mathsf{num} \dashv \Delta_1 \qquad \Gamma \vdash e_2 \Leftarrow \mathsf{num} \rightsquigarrow d_2 : \mathsf{num} \dashv \Delta_1}{\Gamma \vdash e_1 + e_2 \Rightarrow \mathsf{num} \rightsquigarrow d_1 + d_2 \dashv \Delta_1 \cup \Delta_2}$$

$\boxed{\Delta; \Gamma \vdash d : \tau}$ d is assigned type τ

$$\frac{}{\Delta; \Gamma \vdash \underline{n} : \mathsf{num}} \qquad \frac{\Delta; \Gamma \vdash d_1 : \mathsf{num} \qquad \Delta; \Gamma \vdash d_2 : \mathsf{num}}{\Delta; \Gamma \vdash d_1 + d_2 : \mathsf{num}}$$

$\boxed{d \ \mathsf{val}}$ d is a value

$$\frac{}{\underline{n} \ \mathsf{val}}$$

$\boxed{\tau \ \mathsf{ground}}$ τ is a ground type

$$\frac{}{\mathsf{num} \ \mathsf{ground}}$$

$\boxed{d \ \mathsf{indet}}$ d is indeterminate

$$\frac{d_1 \neq \underline{n} \qquad d_1 \ \mathsf{indet} \qquad d_2 \ \mathsf{final}}{d_1 + d_2 \ \mathsf{indet}} \qquad \frac{d_2 \neq \underline{n} \qquad d_1 \ \mathsf{final} \qquad d_2 \ \mathsf{indet}}{d_1 + d_2 \ \mathsf{indet}}$$

$$\mathsf{EvalCtx} \quad \mathcal{E} \quad ::= \quad \cdots \mid \mathcal{E} + d_2 \mid d_1 + \mathcal{E}$$

$\boxed{d = \mathcal{E}\{d'\}}$ d is obtained by placing d' at the mark in $\mathcal{E}$

$$\frac{d_1 = \mathcal{E}\{d_1'\}}{d_1 + d_2 = (\mathcal{E} + d_2)\{d_1'\}} \qquad \frac{d_2 = \mathcal{E}\{d_2'\}}{d_1 + d_2 = (d_1 + \mathcal{E})\{d_2'\}}$$

$\boxed{d_1 \longrightarrow d_2}$ d_1 steps to d_2

$$\frac{n_1 + n_2 = n_3}{\underline{n_1} + \underline{n_2} \longrightarrow \underline{n_3}}$$

B.2 Sum Types

We extend the syntax for sum types as follows:

$$\begin{array}{lrll} \mathsf{HTyp} & \tau & ::= \cdots & \mid \tau + \tau \\ \mathsf{HExp} & e & ::= \cdots & \mid \mathsf{inl}(e) \mid \mathsf{inr}(e) \mid \mathsf{case}(e, x.e, y.e) \\ \mathsf{IHExp} & d & ::= \cdots & \mid \mathsf{inl}_\tau(d) \mid \mathsf{inr}_\tau(d) \mid \mathsf{case}(d, x.d, y.d) \end{array}$$

$\boxed{\mathsf{join}(\tau_1, \tau_2) = \tau_3}$ Types τ_1 and τ_2 join (consistently), forming type τ_3

$$\begin{array}{lcl} \mathsf{join}(\tau, \tau) & = & \tau \\ \mathsf{join}(\llparenthesis\rrparenthesis, \tau) & = & \tau \\ \mathsf{join}(\tau, \llparenthesis\rrparenthesis) & = & \tau \\ \mathsf{join}(\tau_1 \to \tau_2, \tau_1 \to \tau_2) & = & \mathsf{join}(\tau_1, \tau_2) \to \mathsf{join}(\tau_1, \tau_2) \\ \mathsf{join}(\tau_1 + \tau_2, \tau_1 + \tau_2) & = & \mathsf{join}(\tau_1, \tau_2) + \mathsf{join}(\tau_1, \tau_2) \end{array}$$

Theorem B.1 (Joins). *If $join(\tau_1, \tau_2) = \tau$ then types τ_1, τ_2 and τ are pair-wise consistent, i.e., $\tau_1 \sim \tau_2$, $\tau_1 \sim \tau$ and $\tau_2 \sim \tau$.*

Proof. By induction on the derivation of $\mathsf{join}(\tau_1, \tau_2) = \tau$. □

$\boxed{\tau \blacktriangleright_+ \tau_1 + \tau_2}$ Type τ matches the sum type $\tau_1 + \tau_2$

$$\frac{}{\tau_1 + \tau_2 \blacktriangleright_+ \tau_1 + \tau_2} \qquad \frac{}{\llparenthesis\rrparenthesis \blacktriangleright_+ \llparenthesis\rrparenthesis + \llparenthesis\rrparenthesis}$$

$\boxed{\Gamma \vdash e \Leftarrow \tau_1 \rightsquigarrow d : \tau_2 \dashv \Delta}$ e analyzes against type τ_1 and expands to d of consistent type τ_2

$$\frac{\tau \blacktriangleright_+ \tau_1 + \tau_2 \qquad \Gamma \vdash e \Leftarrow \tau_1 \rightsquigarrow d : \tau_1' \dashv \Delta}{\Gamma \vdash \mathsf{inl}(e) \Leftarrow \tau \rightsquigarrow \mathsf{inl}_{\tau_2}(d) : \tau_1' + \tau_2 \dashv \Delta} \qquad \frac{\tau \blacktriangleright_+ \tau_1 + \tau_2 \qquad \Gamma \vdash e \Leftarrow \tau_2 \rightsquigarrow d : \tau_2' \dashv \Delta}{\Gamma \vdash \mathsf{inr}(e) \Leftarrow \tau \rightsquigarrow \mathsf{inl}_{\tau_1}(d) : \tau_1 + \tau_2' \dashv \Delta}$$

$$\frac{\begin{array}{c} \Gamma \vdash e_1 \Rightarrow \tau_1 \rightsquigarrow d_1 \dashv \Delta_1 \qquad \tau_1 \blacktriangleright_+ \tau_{11} + \tau_{12} \\ \mathsf{join}(\tau_2, \tau_3) = \tau' \qquad \Gamma, x : \tau_{11} \vdash e_2 \Leftarrow \tau \rightsquigarrow d_2 : \tau_2 \dashv \Delta_2 \qquad \Gamma, y : \tau_{12} \vdash e_3 \Leftarrow \tau \rightsquigarrow d_3 : \tau_3 \dashv \Delta_3 \end{array}}{\Gamma \vdash \mathsf{case}(e_1, x.e_2, y.e_3) \Leftarrow \tau \rightsquigarrow \mathsf{case}(d_1\langle \tau_1 \Rightarrow \tau_{11} + \tau_{12}\rangle, x.d_2\langle \tau_2 \Rightarrow \tau'\rangle, y.d_3\langle \tau_3 \Rightarrow \tau'\rangle) : \tau' \dashv \Delta_1 \cup \Delta_2 \cup \Delta_3}$$

$\boxed{\Delta; \Gamma \vdash d : \tau}$ d is assigned type τ

$$\frac{\Delta; \Gamma \vdash d : \tau_1}{\Delta; \Gamma \vdash \mathsf{inl}_{\tau_2}(d) : \tau_1 + \tau_2} \qquad \frac{\Delta; \Gamma \vdash d : \tau_2}{\Delta; \Gamma \vdash \mathsf{inr}_{\tau_1}(d) : \tau_1 + \tau_2}$$

$$\frac{\Delta; \Gamma \vdash d_1 : \tau_1 + \tau_2 \qquad \Delta; \Gamma, x : \tau_1 \vdash d_2 : \tau \qquad \Delta; \Gamma, y : \tau_2 \vdash d_3 : \tau}{\Delta; \Gamma \vdash \mathsf{case}(d_1, x.d_2, y.d_3) : \tau}$$

$\boxed{d\ \mathsf{val}}$ d is a value

$$\frac{d\ \mathsf{val}}{\mathsf{inl}_\tau(d)\ \mathsf{val}} \qquad \frac{d\ \mathsf{val}}{\mathsf{inr}_\tau(d)\ \mathsf{val}}$$

$\boxed{\tau\ \mathsf{ground}}$ τ is a ground type

$$\frac{}{\llparenthesis\rrparenthesis + \llparenthesis\rrparenthesis \text{ ground}}$$

$\boxed{d \text{ boxedval}}$ d is a boxed value

$$\frac{d \text{ boxedval}}{\mathsf{inl}_\tau(d) \text{ boxedval}} \qquad \frac{d \text{ boxedval}}{\mathsf{inr}_\tau(d) \text{ boxedval}} \qquad \frac{\tau_1 + \tau_2 \neq \tau_1' + \tau_2' \qquad d \text{ boxedval}}{d\langle \tau_1 + \tau_2 \Rightarrow \tau_1' + \tau_2' \rangle \text{ boxedval}}$$

$\boxed{d \text{ indet}}$ d is indeterminate

$$\frac{d \text{ indet}}{\mathsf{inl}_\tau(d) \text{ indet}} \qquad \frac{d \text{ indet}}{\mathsf{inr}_\tau(d) \text{ indet}} \qquad \frac{\tau_1 + \tau_2 \neq \tau_1' + \tau_2' \qquad d \text{ indet}}{d\langle \tau_1 + \tau_2 \Rightarrow \tau_1' + \tau_2' \rangle \text{ indet}}$$

$$\frac{d_1 \neq \mathsf{inl}_\tau(d_1') \qquad d_1 \neq \mathsf{inr}_\tau(d_1') \qquad d_1 \neq d_1'\langle \tau_1 + \tau_2 \Rightarrow \tau_1' + \tau_2' \rangle \qquad d_1 \text{ indet}}{\mathsf{case}(d_1, x.d_2, y.d_3)}$$

$$\text{EvalCtx} \quad \mathcal{E} \quad ::= \quad \cdots \mid \mathsf{inl}_\tau(\mathcal{E}) \mid \mathsf{inr}_\tau(\mathcal{E}) \mid \mathsf{case}(\mathcal{E}, x.d_1, y.d_2)$$

$\boxed{d = \mathcal{E}\{d'\}}$ d is obtained by placing d' at the mark in $\mathcal{E}$

$$\frac{d_1 = \mathcal{E}\{d_1'\}}{\mathsf{case}(d_1, x.d_2, y.d_3) = \mathsf{case}(\mathcal{E}, x.d_2, y.d_3)\{d_1'\}}$$

$\boxed{d_1 \longrightarrow d_2}$ d_1 steps to d_2

$$\frac{[d_1 \text{ final}]}{\mathsf{case}(\mathsf{inl}_\tau(d_1), x.d_2, y.d_3) \longrightarrow [d_1/x]d_2} \qquad \frac{[d_1 \text{ final}]}{\mathsf{case}(\mathsf{inr}_\tau(d_1), x.d_2, y.d_3) \longrightarrow [d_1/x]d_3}$$

$$\frac{[d_1 \text{ final}]}{\mathsf{case}(d_1\langle \tau_1 + \tau_2 \Rightarrow \tau_1' + \tau_2' \rangle, x.d_2, y.d_3) \longrightarrow \mathsf{case}(d_1, x.[x\langle \tau_1 \Rightarrow \tau_1' \rangle/x]d_2, y.[y\langle \tau_2 \Rightarrow \tau_2' \rangle/y]d_3)}$$

$\boxed{\tau \blacktriangleright_{\text{ground}} \tau'}$ τ has matched ground type τ'

$$\frac{\tau_1 + \tau_2 \neq \llparenthesis\rrparenthesis + \llparenthesis\rrparenthesis}{\tau_1 + \tau_2 \blacktriangleright_{\text{ground}} \llparenthesis\rrparenthesis + \llparenthesis\rrparenthesis}$$

REFERENCES

Tomasz Blanc, Jean-Jacques Lévy, and Luc Maranget. 2005. Sharing in the Weak Lambda-Calculus. In *Processes, Terms and Cycles: Steps on the Road to Infinity, Essays Dedicated to Jan Willem Klop, on the Occasion of His 60th Birthday (Lecture Notes in Computer Science)*, Aart Middeldorp, Vincent van Oostrom, Femke van Raamsdonk, and Roel C. de Vrijer (Eds.), Vol. 3838. Springer, 70–87. https://doi.org/10.1007/11601548_7

Naim Çagman and J. Roger Hindley. 1998. Combinatory Weak Reduction in Lambda Calculus. *Theor. Comput. Sci.* 198, 1-2 (1998), 239–247. https://doi.org/10.1016/S0304-3975(97)00250-8

Alonzo Church and J Barkley Rosser. 1936. Some properties of conversion. *Trans. Amer. Math. Soc.* 39, 3 (1936), 472–482.

Matteo Cimini and Jeremy G. Siek. 2016. The gradualizer: a methodology and algorithm for generating gradual type systems. In *POPL*.

Matteo Cimini and Jeremy G. Siek. 2017. Automatically generating the dynamic semantics of gradually typed languages. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, Giuseppe Castagna and Andrew D. Gordon (Eds.). ACM, 789–803. http://dl.acm.org/citation.cfm?id=3009863

Gérard Huet. 1980. Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems: Abstract Properties and Applications to Term Rewriting Systems. *J. ACM* 27, 4 (1980), 797–821.

Jean-Jacques Lévy and Luc Maranget. 1999. Explicit substitutions and programming languages. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*. Springer, 181–200.

Aleksandar Nanevski, Frank Pfenning, and Brigitte Pientka. 2008. Contextual modal type theory. *ACM Trans. Comput. Log.* 9, 3 (2008). https://doi.org/10.1145/1352582.1352591

Cyrus Omar, Ian Voysey, Michael Hilton, Jonathan Aldrich, and Matthew Hammer. 2017. Hazelnut: A Bidirectionally Typed Structure Editor Calculus. In *Principles of Programming Languages (POPL)*.

Jeremy G. Siek, Michael M. Vitousek, Matteo Cimini, and John Tang Boyland. 2015. Refined Criteria for Gradual Typing. In *1st Summit on Advances in Programming Languages, SNAPL 2015, May 3-6, 2015, Asilomar, California, USA*. 274–293. https://doi.org/10.4230/LIPIcs.SNAPL.2015.274