A APPENDIX: LANGUAGE PROPERTIES AND PROOFS

This appendix contains proof details for the paper "Parallelism in a Region Inference Context" (PLDI '23) by Martin Elsman and Troels Henriksen.

A.1 Properties and Proofs for the Internal Language

Proposition 3.1 (Decomposition). *If $\vdash e : \tau$ then either e is a value or there exist a redex e', a type τ', and a context E^Γ such that $\Gamma \vdash e' : \tau'$ and $e = E^\Gamma[e']$.*

Proof. By induction over the structure of e. There are three cases corresponding to the three typing rules for which e is not a value or a variable.

Case $e = e_0\ e_1$. From the typing rules, we have $\vdash e_0 : \tau'' \to \tau$ and $\vdash e_1 : \tau''$. By induction, either e_0 is a value or there exist a redex e', a type τ', and a context E_0^Γ such that $\Gamma \vdash e' : \tau'$ and $e_0 = E_0^\Gamma[e']$. If e_0 is not a value, we choose $E^\Gamma = E_0^\Gamma\ e_1$, which means that we have $e = E^\Gamma[e']$, as required.

Otherwise, e_0 is a value $e_0 = v_0$. By induction, again, either e_1 is a value or there exist a redex e', a type τ', and a context E_1^Γ, such that $\Gamma \vdash e' : \tau'$ and $e_1 = E_1^\Gamma[e']$. If e_1 is not a value, we choose $E^\Gamma = v_0\ E_1^\Gamma$, which means that we have $e = E^\Gamma[e']$, as required.

Otherwise, e_1 is also a value $e_1 = v_1$. From the typing rules and because $\vdash v_0 : \tau'' \to \tau$ (unique value typing), we know that $v_0 = (\lambda x : \tau''.e_0')\ v_1$, for some x and e_0'. It follows that $e = (\lambda x : \tau''.e_0')\ v_1$, which classifies as a redex. Thus, there exist $e' = e$, and $E^\Gamma = [.]$, and $\tau' = \tau$, such that e' is a redex and $\Gamma \vdash e' : \tau'$ and $e = E^\Gamma[e']$, as required.

Case $e = \texttt{get}\ e_0$. From the typing rules, we have $\vdash e_0 : \tau\ T$. By induction, either e_0 is a value or there exist a redex e', a type τ', and a context E_0^Γ, such that $\Gamma \vdash e' : \tau'$ and $e_0 = E_0^\Gamma[e']$. If e_0 is not a value, we choose $E^\Gamma = \texttt{get}\ E_0^\Gamma$, which means that we have $e = E^\Gamma[e']$, as required. Otherwise e_0 is a value $e_0 = v_0$. From the typing rules and because $\vdash v_0 : \tau\ T$ (unique value typing), we have that $v_0 = \langle v_0' \rangle$, for some v_0'. It follows that $e = \texttt{get}\ \langle v_0' \rangle$, which classifies as a redex, thus, there exist $e' = e$, a type $\tau' = \tau$, and a context $E^\Gamma = [.]$ such that e' is a redex and $\Gamma \vdash e' : \tau'$ and $e = E^\Gamma[e']$, as required.

Case $e = \texttt{letspawn}\ x : \tau\ T = e_0\ \texttt{in}\ e_1$. This case is similar to the case for function application. □

Proposition 3.2 (Context). *If $\Gamma_0 \vdash E^\Gamma[e'] : \tau$ then $\Gamma_0, \Gamma \vdash e' : \tau'$ for some τ'. Further, if $\Gamma_0 \vdash E^\Gamma[e] : \tau$ and $\Gamma_0, \Gamma \vdash e : \tau'$ and $\Gamma_0, \Gamma \vdash e' : \tau'$ then $\Gamma_0 \vdash E^\Gamma[e'] : \tau$.*

Proof. By straightforward induction over the structure of E^Γ. □

Proposition 3.3 (Value Substitution). *If $\Gamma, x : \tau \vdash e : \tau'$ and $\Gamma \vdash v : \tau$ then $\Gamma \vdash [v/x]e : \tau'$.*

Proof. By induction over the structure of e using an *environment extension property* of typing stating that if $\Gamma \vdash e : \tau$ then $\Gamma, \Gamma' \vdash e : \tau$ for any Γ' with $\mathrm{Dom}(\Gamma') \cap \mathrm{Dom}(\Gamma) = \emptyset$. As is standard, the proof also makes use of Barendregt's convention for renaming bound variables in expressions for avoiding environment capture. □

Proposition 3.4 (Type Preservation). *If $\Gamma \vdash e : \tau$ and $e \hookrightarrow e'$ then $\Gamma \vdash e' : \tau$.*

Proof. By induction over the structure of e. From the small-step reduction rules, there are four cases:

Case $e = (\lambda x : \tau_1.e_0)\ v$. From the typing rules, we have $\Gamma \vdash \lambda x : \tau_1.e_0 : \tau_1 \to \tau$ and $\Gamma \vdash v : \tau_1$ and further that $\Gamma, x : \tau_1 \vdash e_0 : \tau$. From the reduction rules, we have $e' = [v/x]e_0$, thus, we can apply Proposition 3.3 to get $\Gamma \vdash e' : \tau$, as required.

Case $e = \texttt{get}\ \langle v \rangle$. From the typing rules, we have $\Gamma \vdash v : \tau$ and from the reduction rules, we have $e' = v$ and thus $\Gamma \vdash e' : \tau$, as required.

Case $e = \texttt{letspawn}\ x : \tau_0\, T = v\ \texttt{in}\ e_0$. From the typing rules, we have $\Gamma \vdash v : \tau_0$ and $\Gamma, x : \tau_0\, T \vdash e_0 : \tau$. From the reduction rules, we have $e' = [\langle v\rangle/x]e_0$. We can now apply the typing rules to get $\Gamma\langle v\rangle : \tau_0\, T$, thus, we can apply Proposition 3.3 to get $\Gamma \vdash e' : \tau$, as required.

Case $e = E^{\Gamma'}[e_0]$. From the typing rules and from Proposition 3.2, we have there exists τ_0 such that $\Gamma, \Gamma' \vdash e_0 : \tau_0$. From the reduction rules, we have there exist e_0' and e' such that $e \hookrightarrow e'$ and $e' = E^{\Gamma'}[e_0']$ and $e_0 \hookrightarrow e_0'$. By induction, we have $\Gamma, \Gamma' \vdash e_0' : \tau_0$. Now, by applying Proposition 3.2 (second part), we have $\Gamma \vdash e' : \tau$, as required. □

Proposition 3.5 (Progress). *If $\vdash e : \tau$ then either e is a value or $e \hookrightarrow e'$ for some e'.*

Proof. From Proposition 3.1, we have either e is a value of there exist a redex e'', a type τ'', and a context E^Γ such that $\Gamma \vdash e'' : \tau''$ and $e = E^\Gamma[e'']$. Because e'' is a redex, we have from the small-step reduction rules and from the definition of redex that there exists e''' such that $e'' \hookrightarrow e'''$ and further that there exists e' such that $e' = E^\Gamma[e''']$ and $e \hookrightarrow e'$, as required. □

A.2 Properties and Proofs for the Region-Annotated Internal Language

Proposition 4.1 (Decomposition). *If $\vdash e : \tau, \varphi'$ then either (1) e is a value or (2) there exist a redex e', a type τ', and a context E^Γ_φ such that $e = E^\Gamma_\varphi[e']$ and $\Gamma \vdash e' : \tau', \varphi \cup \varphi'$.*

Proof. By induction over the structure of e. The proof resembles the proof of decomposition for the internal language with the additional complexity of dealing with region variables. □

Proposition 4.2 (Context). *If $\Gamma_0 \vdash E^\Gamma_\varphi[e'] : \tau, \varphi'$ then $\Gamma_0, \Gamma \vdash e' : \tau', \varphi \cup \varphi'$ for some τ'. Further, if $\Gamma_0 \vdash E^\Gamma_\varphi[e] : \tau, \varphi'$ and $\Gamma_0, \Gamma \vdash e : \tau', \varphi \cup \varphi'$ and $\Gamma_0, \Gamma \vdash e' : \tau', \varphi \cup \varphi'$ then $\Gamma_0 \vdash E^\Gamma_\varphi[e'] : \tau, \varphi'$.*

Proof. By straightforward induction over the structure of E^Γ_φ. □

Proposition 4.3 (Value Substitution). *If $\Gamma, x : \tau \vdash e : \tau', \varphi$ and $\Gamma \vdash v : \tau$ then $\Gamma \vdash [v/x]e : \tau', \varphi$.*

Proof. By induction over the structure of e. The proof is similar to the proof of value substitution for the internal language. It uses an environment extension property of the typing relation and Barendregt's convention for renaming bound variables. □

Proposition 4.4 (Type Preservation). *If $\Gamma \vdash e : \tau, \varphi$ and $e \overset{\varphi}{\hookrightarrow} e'$ then $\Gamma \vdash e' : \tau, \varphi$.*

Proof. By induction over the structure of e. From the small-step reduction rules, there are 7 cases. In each case it is straightforward to demonstrate that the reduction rule preserves typing. For the context case, when $e = E^\Gamma_{\varphi'}[e'']$, for some e'', φ', and Γ, the proof proceeds by case analysis on the structure of $E^\Gamma_{\varphi'}$. □

Proposition 4.5 (Progress). *If $\vdash e : \tau, \varphi$ then either e is a value or $e \overset{\varphi}{\hookrightarrow} e'$ for some e'.*

Proof. By Proposition 4.1, either e is a value or there exist a redex e'', a type τ', and a context $E^\Gamma_{\varphi'}$ such that $e = E^\Gamma_{\varphi'}[e'']$ and $\Gamma \vdash e'' : \tau', \varphi \cup \varphi'$.

We argue that $e'' \overset{\varphi \cup \varphi'}{\hookrightarrow} e'''$ for some e''', so that $E^\Gamma_{\varphi'}[e''] \overset{\varphi}{\hookrightarrow} E^\Gamma_{\varphi'}[e''']$ follows from the context evaluation rule. We now consider all cases where e'' could be stuck.

Case $e'' = \lambda x : \tau_0.e_0\ \texttt{at}\ \rho$, for some τ_0, e_0, and ρ. From the typing rules, we have $\Gamma \vdash \lambda x : \tau_0.e_0\ \texttt{at}\ \rho : \tau', \varphi \cup \varphi'$. This judgment must be derived from the typing rule for lambda expressions followed by a number of applications of the effect expansion rule, which implies that $\rho \in \varphi \cup \varphi'$ and $\tau' = (\tau_0 \xrightarrow{\varphi_0} \tau_1, \rho)$. It follows that we can apply the reduction rule for lambda expressions to get $e''' = \lambda^\rho x : \tau_0.e_0$.

Case $e'' = (\lambda^\rho x : \tau_0.e_0)\ v$, for some ρ, τ_0, e_0, and v. We have $\Gamma \vdash (\lambda^\rho x : \tau_0.e_0)\ v : \tau', \varphi \cup \varphi'$. This judgment must be derived from the typing rule for function application followed by a number of applications of the effect expansion rule. By applying the typing rule for lambda values, we have there exist τ_1 and φ_0 such that $\Gamma \vdash \lambda^\rho x : \tau_0.e_0 : (\tau_0 \xrightarrow{\varphi_0} \tau_1, \rho), \emptyset$ and $\Gamma \vdash v : \tau_0, \emptyset$ and $\rho \in \varphi \cup \varphi'$ and $\varphi_0 \subseteq \varphi \cup \varphi'$. Now, because $\rho \in \varphi \cup \varphi'$, we can apply the function-application reduction-rule, to get $e''' = [v/x]e_0$.

Case $e'' = \mathbf{get}\ \langle v \rangle^\rho$, for some v and ρ. Similar to the case for function application.

Case $e'' = \mathbf{letspawn}\ x : \tau_1 = e_1\ \mathbf{at}\ \rho\ \mathbf{in}\ e_2$, for some x, τ_1, e_1, ρ, and e_2. Similar to the case for lambda expressions. ◻

Received 2022-11-10; accepted 2023-03-31